Mestrado em Informática e Sistemas

JavaScript para aplicações móveis

Relatório de estágio para a obtenção do grau de Mestre em

Informática e Sistemas

Especialização em Engenharia de Software

Autor

Hugo José Beleza Brito

Orientadores

Anabela Gomes

Álvaro Santos

ISEC

Supervisor

Cristóvão Cleto

Crossing Answers

Coimbra, Junho, 2019

Agradecimentos

Em primeiro lugar, quero agradecer à Crossing Answers por me conceder esta oportunidade de desenvolver um projeto tão aliciante numa das áreas de desenvolvimento de *software* de eleição.

Aos professores que encontrei durante o meu percurso académico, agradeço por todo o conhecimento que me transmitiram incluindo os docentes que me apoiaram do ISEC neste projeto, professora Anabela Gomes e professor Álvaro Santos, ao qual agradeço pelo acompanhamento ao longo do estágio.

A todos os meus colegas da empresa que contribuíram e estiveram sempre disponíveis para ajudar em todo o projeto, principalmente ao Filipe Rainho que para além de colega na empresa foi com quem realizei todos os trabalhos do mestrado, juntos ultrapassámos dificuldades e transmitimos conhecimento mútuo.

Por último e não menos importante: a quem sempre acreditou em mim e que sem eles nada teria sido possível. Aos meus pais, principalmente à minha mãe, um eterno obrigado.

Resumo

O desenvolvimento de aplicações móveis está cada vez mais integrado nas empresas de desenvolvimento de software, isto porque a utilização de dispositivos móveis está também cada vez mais disseminada. O desenvolvimento móvel pode ser dividido em três grandes grupos: aplicações nativas, aplicações híbridas e aplicações web. O objetivo principal deste projeto de estágio, realizado na empresa Crossing Answers, foi a mudança de todo o desenvolvimento móvel nativo, em Java para Android e Swift para iOS, para uma *framework* híbrida de desenvolvimento em JavaScript, já que todo o desenvolvimento da empresa assentava em JavaScript. Neste âmbito, e de acordo com os objetivos identificados pela empresa para este estágio, foram feitas análises comparativas entre as abordagens de desenvolvimento nativo e as abordagens híbridas focando-se estas últimas nas frameworks React Native, NativeScript e Ionic. Os estudos foram realizados a diferentes níveis, desde revisões teóricas da literatura a atividades com um cariz mais prático envolvendo os programadores da empresa. O *framework* React Native apresentou melhores resultados globais nos diferentes estudos e análises realizadas, levando a que a transição de todo o processo de desenvolvimento móvel da empresa fosse feita de Java e Swift para React Native. Após a finalização de todo o processo de transição e de implementação de novas metodologias de desenvolvimento móvel foi testada a *framework* num projeto mais pequeno, mas que permitisse a preparação de alguns componentes que pudessem ser adaptados a outras aplicações idênticas com o mesmo modelo de negócio, neste caso audioguias. Por fim, foi implementada uma aplicação com maior complexidade e de outro âmbito de forma a consolidar as vantagens e mitigar as desvantagens da mudança para o novo modelo de desenvolvimento. Todos os objetivos propostos no início do estágio foram cumpridos com sucesso, fazendo com que o desenvolvimento móvel atual da empresa esteja mais produtivo e flexível para futuros projetos.

Palavras Chave*:* *JavaScript, desenvolvimento móvel, React Native, Ionic, NativeScript, Android e iOS*

Abstract

The development of mobile applications is becoming more and more integrated into software development companies, because the use of mobile devices is also becoming increasingly widespread. The mobile development can be divided into three main groups: native applications, hybrid applications and web applications. The main goal of this internship project, accomplished in the Crossing Answers company, was to move all of the native mobile development, in Java for Android and Swift for iOS, towards a hybrid JavaScript development framework, since the entire development of the company was based on JavaScript. For this, and according to the goals identified by the company for this internship, comparative analyses were performed among native and hybrid approaches, the latter focusing on React Native, NativeScript and Ionic frameworks. The studies were carried out at different levels, from theoretical reviews of the literature to activities with a more practical nature involving the company programmers. The React Native framework obtained the best global results in the different studies and analyses carried out, so that, the transition of the entire process of development of the company was made from Java and Swift to React Native. After completing the entire transition and implementing new mobile development methodologies, the framework was tested in a smaller project, but allowing the preparation of some components that could be adapted to other identical applications with the same business model, in this case audio guides. Finally, an application with more complexity and with another scope was implemented in order to consolidate the advantages and mitigate the disadvantages of the change to the new development model. All of the goals proposed at the beginning of the internship have been successfully fulfilled, making the company's current mobile development more flexible and productive for future projects.

Keywords*:* *JavaScript, mobile development, React Native, Ionic, NativeScript, Android e iOS*

Índice

Lista de figuras

Lista de tabelas

Definições e abreviaturas

Audioguia – É uma aplicação móvel que permite a descrição de passeios turísticos com recursos de áudio e conteúdos interativos.

API – *Application programming interface* – É um serviço consumível que permite que as aplicações comuniquem entre si.

API KEY – Chave que identifica quem está a aceder a uma determinada API e se tem os devidos privilégios para isso.

App Store – É o serviço disponibilizado pela Apple para distribuição de aplicações em sistemas operativos iOS.

Beacons – É um pequeno dispositivo que autónomo que fornece um serviço baseado em localização permitindo ter normalmente localizações quase exatas dentro de um determinado espaço.

Boilerplate – É uma parte de código criada uma única vez de forma genérica que faz com que seja reutilizada em todos os projetos do mesmo tipo.

CSS – *Cascading Style Sheets* **–** É uma forma de associar estilos a um determinado documento *web*.

Endpoint – É o caminho utilizado por uma conexação *http* para fazer um pedido de informação a um servidor.

ES6 – *ECMAScript 6* **–** Padrão de desenvolvimento de JavaScript que surgiu em 2015.

GeoJSON – Formato de pradrão utilizado para representação de recursos geográficos e baseado em JSON.

GPX – *GPS eXchange Format* **–** É um esquema de representação de dados de localização, representados através de uma norma definida em XML (Extensible Markup Language).

HTML – *Hyper Text Markup Language* **–** É a linguagem de marcação mais conhecida do desenvolvimento *web*, que através do seu formado é interpretada por *browsers* ou outros documentos.

HTTP – *HyperText Transfer Protocol* – Protocolo de comunicação utilizado em sistemas de informação e principalmente baseado na *web*.

ISEC – *Instituto Superior de Engenharia de Coimbra* – Instituto onde é lecionado o mestrado em informática e sistemas.

JSON – *JavaScript Object Notation* – Formato de comunicação, através de uma norma de representação de informação.

JSX – *JavaScript Syntax Extension* – Tipo de extensão a ficheiros JavaScript, normalmente associados a componentes de React.

PlayStore – É a plataforma de distribuição oficial das aplicações para o sistema operativo Android.

QRCode – Sigla do ingês (Quick Response) e é um código de barras bidimensional que pode ser lido por grande parte dos *smartphones* atuais.

Refactoring – É todo o processo que envolve a modificação e melhoria de código já existente num determinado *software.*

REST API – *Representantional State Transfer* – É uma forma de comunicação via *http* com um conjunto de regras, e que permite utilizar as principais funcionalidades do protocolo *http*.

RGPD – *Regulamento Geral de Proteção de Dados* **–** Regulamento do direito europeu sobre privacidade e proteção de dados pessoais e aplicável a todos os indivíduos da União Europeia.

SCRUM – É um processo iterativo e incremental, usado em desenvolvimento de *software* para a gestão de projetos e desenvolvimento de software ágil.

SmartTV – Televisão digital inteligente que tem um comportamento idêntico a um computador com capacidade de armazenamento, conexão à *internet* com foco principal no entretenimento.

Software – Nome dado ao comportamento de um conjunto de instruções quando executados em um computador ou máquina.

SPA – *Single Page Application* **–** É uma aplicação executada na *World Wide Web* e que tem todo o código num único carregamento de página, fornecendo uma melhor experiência ao utilizador final pois os dados são atualizados numa página sem ter de existir uma atualização total da página.

Spike – Termo utilizado para definir uma pesquisa de como fazer uma funcionalidade, que normalmente não tem um tempo definido, por ser associada a um assunto que não é trivial para uma certa organização.

Sprint – Na metodologia SCRUM, um *sprint* é um ciclo de desenvolvimento. A duração do *sprint*, geralmente, é de uma a quatro semanas.

Statefull – Componentes que utilizam encapsulamento de classes e que têm um estado.

Stateless – Componentes que se definem normalmente como função e que não tem estado.

Token – Termo que define um identificador que permite aceder a informações privadas de um utilizador quando o mesmo está autenticado numa plataforma. Caso não exista um *token* (constante) definido, automaticamente a plataforma redireciona ocliente para o *login* de forma a obter um novo *token*.

Trigger – Termo utilizado em *software* para definir algo que é executado automaticamente como resposta a um determinado evento.

TypeScript – É um subconjunto da linguagem JavaScript que adiciona tipos de dados e alguns outros recursos à linguagem base JavaScript.

URL – Uniform Resource Locator - Endereço de rede em que está algo (site, servidor, etc), normalmente também denominado pelo termo *link*.

Vista – Termo utilizado que define toda a interface que está presente para o utilizador e que pode invocar determinadas funcionalidades de uma aplicação através da acção do utilizador.

XML – *eXtended Markup Language* – Formato de comunicação e representação de dados que está associado a uma determinada norma.

1 Introdução

O desenvolvimento de aplicações móveis está cada vez mais integrado nas empresas de desenvolvimento de *software*. Atualmente, os clientes esperam aplicações com bom desempenho, tempo de desenvolvimento pequeno e custos reduzidos.

A empresa Crossing Answers é uma empresa que desenvolve várias soluções focadas em três segmentos: *Interactive* (Soluções de multimédia para fins empresariais e turísticos), *Can Play* (Desenvolvimento de jogos) e *Technology* (Desenvolvimento de *software* e *hardware*) [1]. Em todas as áreas, na Crossing Answers, é realizado desenvolvimento de aplicações móveis, e em todos os projetos, a organização utiliza o JavaScript como linguagem-base, exceto no desenvolvimento móvel em que é utilizado o Java para Android e o Swift para iOS.

Neste documento, descrevem-se os procedimentos realizados no sentido de adaptar os processos de desenvolvimento móvel na empresa e a criação de novas metodologias mais adequadas aos objetivos da mesma.

No subcapítulo 1.1, será abordado o âmbito de todo o projeto de estágio. Em 1.2, será apresentada a empresa onde foi realizado o estágio (Crossing Answers). Em 1.3, será apresentado o ISEC, a instituição de ensino que possibilitou a realização de estágio. Em 1.4, serão apresentados o problema e a pertinência da proposta de estágio pela organização. Em 1.5, será feito um resumo dos principais objetivos do estágio e, em 1.6, será descrita a organização de todo o documento.

1.1 Âmbito

Este documento tem como objetivo apresentar o trabalho desenvolvido, inserido na unidade curricular de "Projeto ou Estágio" do Mestrado em Informática e Sistemas, lecionado no ISEC (Instituto Superior de Engenharia de Coimbra). Esta unidade curricular tem como objetivo principal a integração dos alunos num ambiente empresarial para aquisição de experiência, com resolução de problemas não triviais e com elementos novos que envolvam estudo e investigação, aliada ao desenvolvimento de uma solução do problema proposto, consoante as boas práticas em uso no domínio em que se insere.

O estágio foi realizado nas instalações da empresa Crossing Answers, tendo como orientadores os professores Anabela Gomes e Álvaro Santos, professores do ISEC, e o CEO (Diretor Executivo) da empresa

Crossing Answers, Cristóvão Cleto. Este estágio teve início no dia 11 de dezembro de 2017 e o seu término ocorreu no dia 28 de setembro de 2018.

1.2 Crossing Answers

A Crossing Answers (Figura 1), fundada em 2012 e sediada na Rua Miguel Torga em Coimbra, é "um estúdio de desenvolvimento digital focado na criação de novos produtos e serviços com uma forte componente tecnológica", com a missão de "criar soluções tecnológicas com uma diferença mensurável para o mundo" [1].

A empresa teve o seu início com o lançamento de um produto inovador, ONEsw, o primeiro sistema de verificação de idade para máquinas de tabaco com recurso ao cartão de cidadão, e atualmente desenvolve atividades nas seguintes vertentes:

- *Interactive* – Soluções multimédia e interativas com fins turísticos e empresariais (realidade virtual, realidade aumentada, guias interativos, equipamentos virtuais e multimédia);
- *Can Play* – Desenvolvimento de Jogos para dispositivos móveis (Android e iOS) e para Playstation;
- *Technology* – Desenvolvimento de *software* e *hardware*.

Figura 1 – Logótipo Crossing Answers

1.3 ISEC

O ISEC é uma unidade orgânica de ensino do Instituto Politécnico de Coimbra (IPC) e conta com quatro décadas de integração no ensino superior politécnico. No entanto, já pratica o ensino de tecnologia e de engenharia há mais de 90 anos [2]. A missão do ISEC passa por criar e transmitir conhecimentos aos seus estudantes, bem como criar uma *interface* entre o mundo académico e o mundo empresarial da Engenharia. Para além de 12 cursos de licenciaturas, existem mais 9 cursos de Ano Zero que visam a preparação dos alunos para a frequência das várias licenciaturas existentes. O ISEC também conta com 8 Cursos Técnicos Superiores Profissionais, bem como com 9 Mestrados e 2 Pós-Graduações. O ISEC está dividido em 6 departamentos, cada um com a sua especialidade:

- Departamento de Engenharia Civil (DEC);
- Departamento de Engenharia Eletrotécnica (DEE);

- Departamento de Engenharia Informática e de Sistemas (DEIS);
- Departamento de Engenharia Mecânica (DEM);
- Departamento de Engenharia Química e Biológica (DEQB);
- Departamento de Física e Matemática (DFM).

Saliente-se que o ISEC conta, neste momento, com mais de 2500 alunos e que o DEIS tem recebido mais de uma centena de propostas de estágios curriculares por ano letivo, o que atesta o seu reconhecimento por parte das empresas da região e a qualidade do ensino ministrado.

1.4 Descrição do problema

Como foi referido, a empresa Crossing Answers realiza desenvolvimento de aplicações móveis em todas as suas áreas de negócio. Na área de *Technology*, em aplicações à medida para clientes, na área de *Interactive*, criando guias interativos e soluções de realidade virtual e aumentada para turismo ou museus, e na área de *Can Play*, desenvolvendo jogos para dispositivos móveis.

O problema da organização surge quando se inicia o desenvolvimento de um determinado projeto e se percebe que é necessário configurar vários ambientes de desenvolvimento e aprender a programar duas ou mais linguagens distintas para realizar uma única aplicação móvel (Java, no caso de Android, e Swift, no caso de iOS), levando à necessidade de integrar pessoas especializadas nessas tecnologias. Isto leva a que várias vezes o desenvolvimento Android e iOS seja realizado por equipas distintas e especializadas numa única linguagem.

A Crossing Answers tem todo o seu desenvolvimento baseado na linguagem JavaScript, desde o *back-end* em NodeJS com Express [3], o *front-end web* em AngularJS [4] e aplicações *desktop* em Electron [5]. Assim, para o desenvolvimento de aplicações móveis, é necessário recorrer a programadores especializados em Java para Android e Swift para iOS. Esta situação leva a que não haja possibilidade de ter uma equipa com programadores capazes de trabalhar em todas as áreas de desenvolvimento da empresa, devido às linguagens serem bastante diferentes. Talvez por essa razão alguns projetos de aplicações móveis não alcançaram o sucesso esperado para a empresa, surgindo a proposta de estágio, no contexto do mestrado, no sentido de mudar as práticas de desenvolvimento móvel tradicional nativo (Java/Swift) para o desenvolvimento móvel híbrido através de JavaScript. Desta forma, toda a *stack* de desenvolvimento da empresa passa a ser baseada numa única linguagem de programação.

1.5 Objetivos do estágio

A proposta de estágio (Anexo A), apresentada pela Crossing Answers ao ISEC, tem como foco a mudança de paradigma de todo o desenvolvimento móvel da organização, sendo que os objetivos propostos por parte da empresa para a concretização desse propósito são os seguintes:

- Estudo e análise de *frameworks* de desenvolvimento móvel JavaScript (React Native [6], NativeScript [7] e Ionic [8]) e obtenção das respetivas vantagens de cada uma para a Crossing Answers;
- Recolha de dados, análises e conclusões sobre a adaptação, facilidade de aprendizagem e *feedback* dos programadores da Crossing Answers para as várias *frameworks* em comparação;
- Finalização de todo o desenvolvimento de aplicações por entregar em Java/Swift, com as respetivas publicações, ou identificação de viabilidade de novas implementações em JavaScript;
- Implementação e alteração de todas as práticas de desenvolvimento móvel necessárias dentro da Crossing Answers, de forma a que todos os projetos passem a ser implementados em desenvolvimento móvel híbrido na linguagem JavaScript;
- Desenvolvimento de arquitetura para as aplicações de audioguias em JavaScript, que permita criar uma aplicação de audioguia de forma rápida, independentemente do cliente;
- Desenvolvimento de um projeto maior, já implementado na *framework* usada na organização, recorrendo às novas práticas. Neste caso, consiste numa rede social de contactos profissionais e pretende-se perceber o que implica a inclusão de uma *framework* de JavaScript em projetos maiores, e quais as vantagens/desvantagens daí advindas.

Tendo em conta os objetivos principais do estágio, estes foram agrupados em quatro grupos essenciais: conclusão do desenvolvimento de aplicações nativas, estudo e seleção da plataforma JavaScript, mudança para a nova *framework* (planeamento e execução) e desenvolvimento de novas aplicações. A execução dos principais pontos por ordem cronológica, ao longo do estágio, é apresentada na Figura 2.

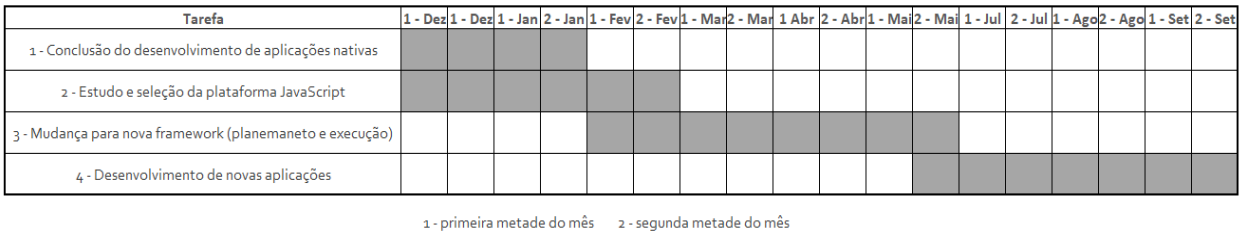


Figura 2 – Diagrama de Gantt com a representação do trabalho realizado ao longo do estágio

1.6 Estrutura do documento

Além deste capítulo introdutório, o presente relatório encontra-se dividido em nove capítulos e onze anexos:

- Capítulo 2 – É feito um levantamento do contexto atual do desenvolvimento de aplicações móveis, retratando como está atualmente o mercado e quais as *frameworks* de JavaScript com melhor possibilidade de evolução;
- Capítulo 3 – É feito um estudo comparativo, através da análise de trabalhos relacionados com comparações de *frameworks* e/ou linguagens, obtendo as variáveis mais importantes para o estudo, aplicando-o e retirando conclusões do mesmo;
- Capítulo 4 – É feito um estudo de aprendizagem, em que foram realizados testes a vários programadores da empresa com base numa taxonomia adaptada de Bloom, de forma a entender qual a *framework* em que os programadores da empresa obtinham melhores resultados e davam melhor *feedback*;
- Capítulo 5 – Devido ao conceito de o estágio ser a transição de todo o desenvolvimento móvel da empresa, foi necessário terminar dois projetos cujo desenvolvimento ainda estava a decorrer com as práticas anteriores à mudança. É explicado o que foi feito nesses projetos;
- Capítulo 6 – É onde é detalhado todo o processo que foi realizado na empresa, quais as diferentes metas que foram criadas e como foram aplicadas para a alteração de todo o processo de desenvolvimento;
- Capítulo 7 – É detalhado como foi idealizado o desenvolvimento de uma arquitetura baseada em componentes para aplicação em projetos do tipo audioguias da empresa. É apresentado também o trabalho final da primeira aplicação desenvolvida no novo processo de desenvolvimento da empresa;
- Capítulo 8 – É apresentado um projeto de maiores dimensões, onde a nova *framework* de desenvolvimento escolhida é utilizada, sendo apresentado o resultado final e as vantagens e desvantagens de ter sido selecionada aquela *framework*;
- Capítulo 9 – São feitas as considerações finais ao trabalho, apresentadas as principais contribuições, dificuldades e o trabalho futuro, deixando indicações de um possível caminho positivo que a empresa poderá seguir após esta transição.

No final, o documento é completado com as referências bibliográficas e com os anexos produzidos ao longo do estágio:

- Anexo A – Proposta de estágio da Crossing Answers ao ISEC;

- Anexo B – Artigo publicado em Cáceres na conferência ibérica de Sistemas e Tecnologias de Informação 2018, na sequência do estudo comparativo entre *frameworks*;
- Anexo C – Testes realizados, com todas as perguntas que foram feitas aos programadores nos testes práticos;
- Anexo D – Tutoriais fornecidos aos programadores, a que estes puderam aceder antes e durante a realização de cada teste;
- Anexo E –Artigo publicado em Coimbra na conferência ibérica de Sistemas e Tecnologias de informação 2019, na sequência do estudo de análise de aprendizagem;
- Anexo F – Resultado final da aplicação Portugal dos Pequenitos;
- Anexo G – Resultado final da aplicação Casa Museu Bissaya Barreto;
- Anexo H – Toda a documentação que foi criada relativa ao *boilerplate*;
- Anexo I – Documentação de como pode ser criada uma *bridge* em React Native para utilização de código nativo;
- Anexo J – Resultado final da aplicação audioguia de Sesimbra;
- Anexo K – Artigo publicado em Coimbra na conferência ibérica de Sistemas e Tecnologias de informação 2019, na sequência do desenvolvimento das aplicações de audioguias nativas em comparação com o desenvolvimento da desenvolvida em JavaScript;
- Anexo L – Resultado final da aplicação Connects;
- Anexo M – *Spike* realizada sobre implementação de navegação *offline* em React Native.

2 Estado da arte

Os dispositivos móveis estão cada vez mais avançados e eficientes, estimando-se que até 2020 existam cerca de 2,87 mil milhões de utilizadores de *smartphones* [9]. O número de utilizadores de Internet a nível global previsto para 2020 é de 52,4% [10], julgando-se que cerca de 34% da população mundial terá pelo menos um *smartphone*. Isto vem reforçar a importância dos dispositivos móveis no dia a dia de toda a sociedade, pois aliam o poder de computação à portabilidade.

Este capítulo faz uma abordagem inicial do estudo feito sobre o desenvolvimento de aplicações móveis e sobre as *frameworks* de JavaScript para o seu desenvolvimento. No subcapítulo 2.1, é feita uma descrição do estado atual do mercado das aplicações móveis e dos sistemas mais importantes atualmente. Em 2.2 são abordadas as principais estratégias atuais de desenvolvimento de aplicações móveis, e em 2.3 são apresentadas algumas das principais *framework*s para desenvolvimento móvel que recorrem à linguagem JavaScript.

2.1 Plataformas móveis

Uma aplicação móvel é um *software* projetado para ser executado em *smartphones*, *tablets* ou outros dispositivos móveis. O conjunto formado pelas aplicações e pelos dispositivos móveis transformaram drasticamente a vida quotidiana da população [11]. Uma aplicação será bem-sucedida se permitir um envolvimento interativo associado à mobilidade fornecida por um *smartphone* [12]. Originalmente, a maior parte das aplicações móveis foram concebidas para fins informativos e de produtividade, como, por exemplo, *e-mail*, calendário, contactos ou calculadora. Com a rápida evolução da tecnologia, o desenvolvimento das aplicações móveis foi expandido para outras categorias, tais como jogos, localização (GPS) ou *media* sociais. Atualmente, é possível satisfazer as necessidades diárias através de aplicações móveis, pois oferecem funcionalidades idênticas ou superiores às aplicações tradicionais usadas nos computadores, podendo ser acedidas em qualquer lugar, tirando ou não partido de conexão à Internet.

Hoje em dia, a Google e a Apple, com Android e iOS respetivamente, detêm 99,9% [13] do mercado global de dispositivos móveis (Figura 3) [13], sendo a distribuição das aplicações feita respetivamente através das lojas Play Store e App Store, que estão associadas a cada um dos sistemas operativos referidos. Para um cliente que contacta uma empresa para o desenvolvimento de uma aplicação móvel, um dos primeiros objetivos normalmente associados à realização de uma aplicação é que a mesma alcance um vasto

mercado. Esse objetivo faz com que os requisitos incluam a disponibilização da aplicação para os sistemas operativos de Android e iOS.

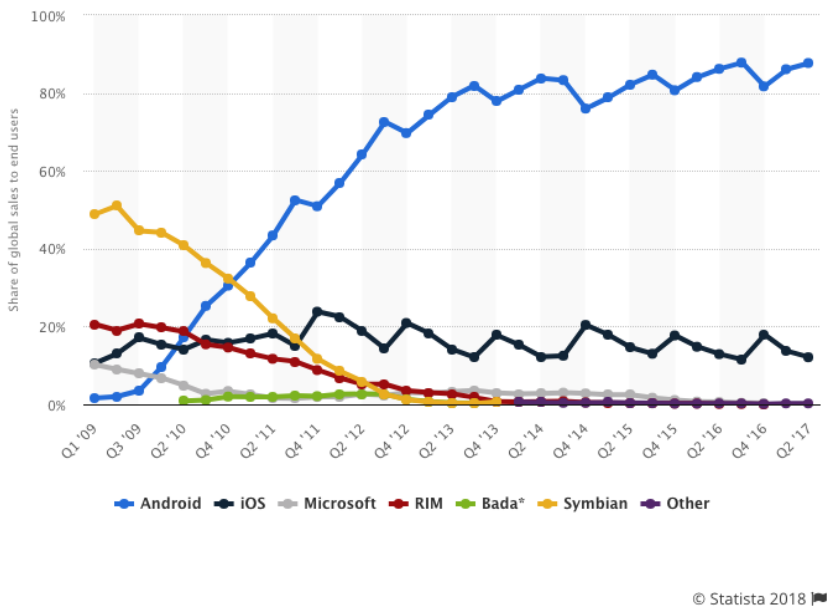


Figura 3 – Participação do mercado global de sistemas operativos móveis em vendas [13].

2.1.1 Android

O Android é uma plataforma aberta desenvolvida pela Google para dispositivos móveis baseada no kernel do Linux [14]. Inicialmente, o Android foi projetado para *smartphones* baseados em toque, contudo, agora, é também usado em sistemas Desktop, SmartTV, SmartWatch ou carros [15]. No primeiro trimestre de 2017, a quota de mercado dos dispositivos móveis com sistema operativo Android era de 87,8% [13].

Grande parte das aplicações Android está implementada em Java. Inicialmente, o desenvolvimento de aplicações Android era realizado tirando partido de uma extensão ao ambiente de desenvolvimento (IDE) Eclipse. Posteriormente, tendo por base o ambiente de desenvolvimento IntelliJ da empresa JetBrains, a Google disponibilizou o Android Studio que é considerado o IDE mais intuitivo e mais acessível para a construção de aplicações móveis Android [16]. Mais recentemente, de forma a tornar o desenvolvimento menos suscetível a erros, mais simples, conciso, seguro e rápido, a Google adotou a linguagem Kotlin, criada pela Jetbrains, como uma linguagem oficial de desenvolvimento Android [17]. A razão prende-se com o facto de o desenvolvimento em Android ser cerca de 30% mais lento do que a concorrência iOS [18]. A linguagem Kotlin está atualmente a ser adotada por grande parte da comunidade que desenvolveu aplicações Android de forma nativa para apenas esse sistema operativo [19].

2.1.2 iOS

O iOS é um sistema operativo móvel projetado pela Apple e distribuído para ser executado em dispositivos iPhone, iPad, iPod e Apple TV. Foi introduzido pela primeira vez a 29 de junho de 2007 na Mac World Conference [20]. O iOS catalisou uma transição de indústria móvel tradicional para a indústria de redes de valor, pois a Internet móvel só era conectável através dos portais das operadoras móveis, mas o iOS ultrapassou o limite entre os dispositivos móveis e a Internet móvel [21]. No segundo trimestre de 2017, cerca de 12,1% dos dispositivos móveis globais executavam o sistema operativo iOS e, consequentemente, da marca Apple [13].

O iOS é reconhecido, entre outros aspetos, pelos seus recursos de segurança, estando associado a um processo mais exaustivo de revisão de novas aplicações disponíveis através da App Store aquele que é utilizado em sistemas Android [22]. A Apple durante vários anos utilizou o Objective-C como a principal linguagem para o desenvolvimento das suas aplicações. A partir de 2014, houve uma mudança gradual para Swift, uma vez que se tratava de uma linguagem mais recente e com um tempo de implementação mais curto, para além das similaridades com linguagens como o JavaScript ou o Python, conseguindo um desempenho quase idêntico ao C++ [23].

2.1.3 Outras plataformas

Tal como é percetível na Figura 3, durante o aparecimento dos *smartphones*, no final da primeira década do século XXI, existiam vários sistemas operativos com presença no mercado dos dispositivos móveis, incluindo a Microsoft, que, a par com a Google e a Apple, tentaram entrar em força no mercado. Porém, a partir de finais de 2013, o mercado começou a ficar bipolarizado apenas pela Google com o Android e pela Apple com o iOS, tendo atualmente em conjunto uma quota de mercado de 99,9% [13]. As quotas de mercado dos dois sistemas apenas sofrem oscilações aquando de lançamentos de novos modelos de iPhone e nos novos modelos de Android com as últimas versões do sistema, disponibilizadas em telemóveis da Google ou de marcas de referência. Devido a esta polarização dos dois sistemas operativos, o desenvolvimento de aplicações móveis é essencialmente focado para Android e iOS.

2.2 Desenvolvimento de aplicações móveis

O número de aplicações publicadas vai aumentando a cada semestre [24] e, devido ao crescimento do mercado das aplicações móveis, as empresas querem construir aplicações mais eficientes e com mais funcionalidades, mas ao mesmo tempo pretendem manter a mesma sensação de desempenho de qualquer outra aplicação atualmente no mercado.

Apesar do esforço, tanto da Google como da Apple, para uma melhoria constante das linguagens e componentes com o objetivo de diminuir o esforço dos programadores, o problema central da distribuição de aplicações móveis continua. Verifica-se que as aplicações para *smartphones* Android e iOS são desenvolvidas com linguagens de programação totalmente distintas [22], o que normalmente tem como consequência um maior tempo de desenvolvimento de uma ideia de negócio. Esta situação deve-se ao facto de ser necessária a implementação de duas lógicas de programação diferentes, uma após a outra, ou então de ter uma equipa de desenvolvimento especializada para cada uma das plataformas [25].

Embora o nível de especialização que se obtém nessas equipas seja benéfico em muitas situações, se fosse possível desenvolver aplicações Android e iOS numa única equipa e num único projeto de desenvolvimento, isso poderia reduzir substancialmente o tempo e custos de desenvolvimento.

Numa tentativa de mitigar estes problemas e agilizar o processo de desenvolvimento, tem-se assistido ao aparecimento de soluções híbridas, que permitem a disponibilização de aplicações para várias plataformas móveis, tendo por base um único projeto de desenvolvimento. Atualmente, as principais soluções de desenvolvimento móvel podem ser divididas em três grupos: desenvolvimento nativo, desenvolvimento híbrido e desenvolvimento de aplicações *web*, como demonstra a Figura 4.

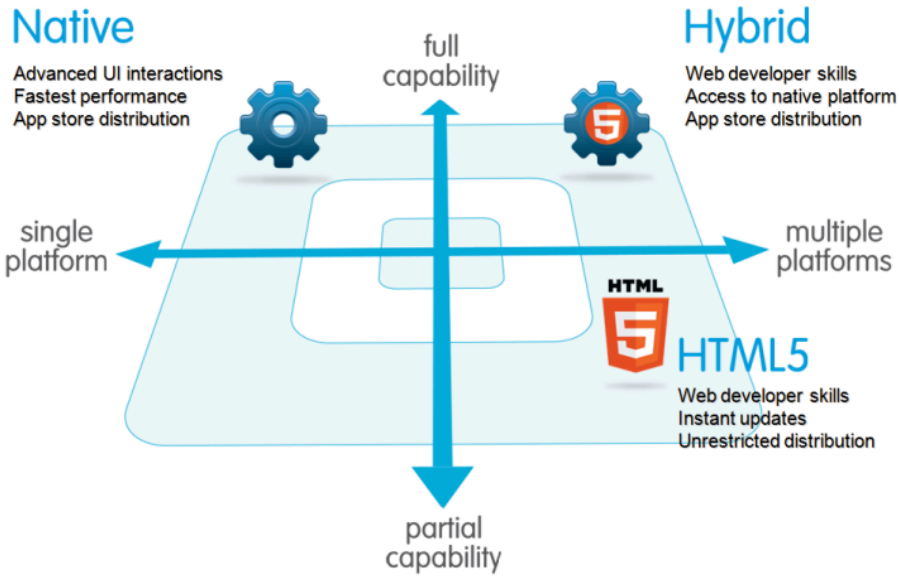


Figura 4 – Classificação do tipo de desenvolvimento de aplicações [26].

2.2.1 Desenvolvimento de aplicações nativas

O desenvolvimento nativo é o método de desenvolvimento de aplicações mais antigo e mais bem estabelecido. Este método consiste em criar um projeto com código distinto para todas as plataformas de destino desejadas. No caso de uma aplicação para Android e iOS, consiste em dois projetos com código totalmente distinto, mas que tem como intuito final a criação da mesma aplicação.

O desenvolvimento nativo apresenta diversas vantagens, já que podem ser usados todos os recursos do sistema operativo e do dispositivo, obtendo-se melhores níveis de desempenho. Assim, as aplicações nativas são habitualmente as mais rápidas e mais suaves, pois o código é focado na otimização da aplicação para o sistema operativo desejado.

Por outro lado, para suportar *n* plataformas, têm de ser criadas *n* aplicações com *n* linguagens de programação diferentes [27], [28].

2.2.2 Desenvolvimento de aplicações híbridas

As aplicações desenvolvidas de forma híbrida são escritas numa única linguagem de programação e, em seguida, o código é compilado para cada uma das plataformas móveis de destino.

A vantagem desta abordagem é que as aplicações finais se comportam como aplicações nativas em termos de *interface*, já que todos os elementos exibidos na aplicação são realmente elementos de cada plataforma destino. Regra geral, as aplicações também apresentam um desempenho elevado para o utilizador final, apesar de o desempenho poder variar consoante a *framework* de desenvolvimento escolhida.

O exemplo-piloto mais conhecido desta abordagem foi o Xamarin [29], em que o código comum é C#, mas atualmente existem muitas alternativas para outras linguagens de programação, como, por exemplo: JavaScript com o React Native [6], NativeScript [7] ou Ionic [8]; Python com a BeeWere [30]; ou até mesmo Ruby com o RubyMotion [31], [27], [32]. Como este estudo será focado no desenvolvimento de aplicações móveis para JavaScript para posterior implementação na Crossing Answers, serão essencialmente analisadas as *frameworks* React Native, NativeScript e Ionic.

2.2.3 Desenvolvimento de aplicações web

Esta é a abordagem mais rápida para alcançar uma aplicação para dispositivos móveis entre diferentes plataformas móveis. As aplicações funcionam sobre uma *WebView,* que é uma forma de disponibilizar conteúdo *web* através de uma única instância no dispositivo móvel [33], sendo essa *Webview* aberta no início da execução da aplicação. A *WebView* está presente durante a execução de toda a aplicação e toda a lógica da aplicação é executada dentro dessa mesma *WebView* de forma transparente para o utilizador final [25].

Uma das vantagens desta abordagem reside na simplificação do processo de criação de uma aplicação, consistindo apenas na criação de uma página *web* que é mostrada em toda a exibição. Desta forma, as tecnologias-base normalmente utilizadas são o HTML, CSS e JavaScript.

O problema destas aplicações está principalmente relacionado com o desempenho, pois as aplicações necessitam de executar conteúdo HTML num único componente do tipo *WebView*, tarefa que por si só já é pesada para um dispositivo móvel.

Exemplos de *frameworks* baseados numa abordagem *WebView* são o Apache Cordova, a AppGyver Supersonic e a TouchstoneJS. Apesar de esta abordagem ter sido vista como uma solução para os problemas do desenvolvimento móvel multiplataforma, aquela foi perdendo popularidade devido ao desempenho ser realmente muito inferior, sendo percetível ao utilizador final o menor desempenho da aplicação [34], [35].

2.3 *Frameworks* JavaScript de desenvolvimento móvel

O JavaScript tornou-se uma linguagem muito popular, tendo no ano de 2017 mais do dobro de repositórios de GitHub, em relação à segunda linguagem mais popular, o Python (2,3 milhões para 1,1 milhões) [36]. A popularidade tem vindo a aumentar devido a uma evolução rápida da linguagem e ao propósito geral para que foi construída, tornando possível a construção de uma equipa de trabalho em diferentes especialidades, tecnologias ou projetos [37] .

Assim, como em outras linguagens, têm vindo a aparecer *frameworks* que permitem o desenvolvimento móvel de forma mais rápida, através do JavaScript. De entre todas, as que têm maior reconhecimento ou possibilidade de evolução são o React Native, o NativeScript e o Ionic [38], estando de acordo com as recomendações iniciais da Crossing Answers.

2.3.1 React Native

Antes de aparecer o React Native foi criado o ReactJS. Esta é uma *framework* de JavaScript para desenvolvimento *web*, construída pelo Facebook, tendo sido lançada no ano de 2013 [39].

O React é uma *framework* que vai ao encontro da visão principal de desenvolvimento do Facebook: reduzir o tempo de desenvolvimento, optando pela criação de código reutilizável e eficiente [40]. Desta forma, o React tem por objetivo principal a divisão de todas as vistas em diferentes componentes para ser possível a reutilização de código para outras vistas do projeto, ou até mesmo em projetos totalmente diferentes, levando a um ciclo de desenvolvimento mais rápido [41].

Em 2015, e após dois anos do lançamento do ReactJS, na React.js Conf 2015 [42], foi anunciado que, com o sucesso que o ReactJS alcançou no desenvolvimento de aplicações para a *web*, iria ser preparada uma solução para desenvolvimento móvel, React Native, de forma a acabar com a desconexão total no desenvolvimento entre Android e iOS, mas sem a necessidade de recorrer à utilização de *WebViews*.

O React Native teve a sua primeira aparição em 2015, contendo algumas lacunas relacionadas com uma maior dificuldade em criar uma *framework* de desenvolvimento híbrido sem recorrer às já mencionadas

WebViews e com componentes nativos para Android e para iOS, sendo que atualmente já foram ultrapassadas com a estabilização da *framework*.

A lógica da aplicação é gravada em JavaScript, enquanto a vista é totalmente nativa, havendo sempre a possibilidade de implementação de código nativo para cada uma das plataformas.

2.3.2 Ionic

O Ionic é um produto de desenvolvimento de aplicações híbridas da Drifty [43], uma *startup* de Silicon Valley. Esta estrutura foi lançada a 12 de maio de 2015, dando a possibilidade aos programadores de criar aplicações para o mercado sem qualquer custo adicional associado, já que o Ionic é disponibilizado em código aberto. É uma *framework* bastante intuitiva, pois torna possível criar aplicações a partir dos conteúdos *web* utilizados normalmente como HTML, CSS e JavaScript, recorrendo para isso ao Apache Cordova.

O Apache Cordova é o que permite que a aplicação aceda a todos os recursos nativos do sistema operativo tendo por base a *framework* de *web* AngularJS, que é uma das estruturas mais utilizadas no desenvolvimento JavaScript [44], facilitando aos programadores a criação de aplicações do tipo *single page application* de forma fácil e organizada.

Cada plataforma móvel tem os seus requisitos em componentes de *interface* de utilizador. Para ajudar os programadores a economizar tempo de desenvolvimento, o Ionic fornece uma *interface* preparada com alguns componentes móveis que são aplicados automaticamente com base na compilação do tipo de plataforma como as Listas de dados, Menus Laterais, ou navegação entre vistas [44], [45].

2.3.3 NativeScript

O NativeScript é uma *framework* de código aberto, disponibilizado sob a licença Apache e suportado pela Telerik [46], que permite o desenvolvimento com recurso a JavaScript ou TypeScript.

O NativeScript executa o código específico de cada plataforma através do seu mecanismo de execução nativa. O resultado é uma *interface* para o utilizador com desempenho e experiência semelhante à das aplicações nativas [7]. No conceito principal do NativeScript, a aplicação consiste em páginas *web*, cada uma representando um ecrã de uma aplicação [47]. A lógica de negócio é desenvolvida e armazenada em ficheiros JavaScript (JS) ou TypeScript (TS), e o *design* é desenvolvido e armazenado em ficheiros XML. Os nomes dos ficheiros XML de vista e dos ficheiros JS ou TS de lógica devem ter o mesmo nome para se ligarem em termos de vistas/lógica de negócio. O estilo especificado é definido através de ficheiros CSS, tornando o desenvolvimento muito idêntico à *web*, exceto na linguagem de marcação em que é utilizado o XML e não o HTML [48].

3 Estudo comparativo

Para existir uma mudança de uma nova tecnologia no desenvolvimento móvel interno de toda a organização, é necessário fazer um estudo aprofundado de quais as vantagens e desvantagens de cada uma das opções que podem ser escolhidas. Neste caso, foi essencial fazer um estudo aprofundado de qual a melhor solução para o desenvolvimento móvel em JavaScript com mais vantagens e, também, a nível interno da organização, qual a que melhor se poderá adaptar aos objetivos a médio/longo prazo.

No subcapítulo 3.1, são apresentados trabalhos relacionados com comparações entre abordagens de desenvolvimento móvel, estudos comparativos de análise de *frameworks* híbridas e estudos sobre quais as variáveis mais importantes a ter em conta na avaliação de uma *framework* de desenvolvimento de aplicações móveis. Em 3.2, são apresentadas as variáveis que foram selecionadas para o estudo comparativo realizado e o que cada uma delas avalia. Em 3.3, são apresentadas as conclusões principais do estudo comparativo realizado entre soluções nativas e as *frameworks* de JavaScript. Por fim, em 3.4, é apresentada uma conclusão de todo o estudo.

3.1 Trabalho relacionado

Existem bastantes estudos relacionados com aplicações móveis, focando-se maioritariamente em aplicações nativas Android e iOS. O desenvolvimento de aplicações híbridas está ainda em constante avanço, sendo alvo de melhorias em comparação com o desenvolvimento nativo que já tem práticas bem estabelecidas. Porém, existem já algumas pesquisas e comparações sobre desenvolvimento móvel híbrido, com algumas conclusões que se passam a descrever.

Halidovic e Karli [49] afirmam que cada vez mais é vincada a fragmentação entre os dois principais sistemas operativos de dispositivos móveis, sendo progressivamente um desafio desenvolver aplicações similares, eficientes e sem erros para ambos os sistemas operativos. Para verificarem isso, fizeram uma comparação entre o Icenium (desenvolvimento híbrido) [50] e o desenvolvimento Android/iOS, de forma nativa em Java e Swift, através de 14 critérios considerados pelo autor importantes no desenvolvimento de aplicações móveis, concluindo que o Icenium é mais vantajoso em relação ao Java/Swift, pela grande diferença no tempo de desenvolvimento (quase metade do tempo) e pelas equipas de desenvolvimento serem constituídas por especialistas apenas numa linguagem de programação. Os autores, porém, alertam

para o facto de a diferença de desempenho entre o Icenium e as soluções nativas ser ligeiramente percetível, e que outras *frameworks* poderiam apresentar melhor desempenho.

Singh [22] fez uma comparação entre o Cordova e o Android/iOS em desenvolvimento nativo, através de 19 critérios diferentes considerados relevantes pelo autor para o desenvolvimento de aplicações móveis, concluindo que não existe uma opção completamente correta. Concluiu ainda que, num modelo de negócio em constante mudança e com várias atualizações, ou no lançamento de aplicações simples de teste a um nicho de mercado, o Cordova é a melhor opção, sendo que as aplicações nativas só têm vantagem em produtos pouco dinâmicos e com tempo de mercado mais extenso. Segundo Singh, com a maturidade do desenvolvimento híbrido, as vantagens ainda serão mais vincadas, dando como exemplo o facto de grandes empresas, como a Amazon ou Linkedin, já terem implementado as suas aplicações através de soluções híbridas.

Após um trabalho realizado pela Optimus Information [25], esta empresa afirma que a estratégia de desenvolvimento móvel de duas organizações pode ser totalmente distinta, mas que ambas podem ter a solução de desenvolvimento móvel ideal para o modelo de negócio em que se inserem. Isto porque a solução de desenvolvimento móvel de uma organização deve ser adotada, após uma análise profunda, tendo em consideração os critérios mais importantes para a mesma. Grande parte das organizações tem em conta critérios importantes, tais como o desempenho, a experiência do utilizador, o custo de desenvolvimento e de custos de atualizações e manutenção. Através da análise desses critérios e de indicadas as vantagens e desvantagens para desenvolvimento móvel do tipo nativo, híbrido e *web*, conclui-se que, no geral, o desenvolvimento híbrido é a melhor opção para a grande maioria das organizações. Relembrando, como conclusão, que o modelo de negócio, em que uma empresa se insere, é um dos critérios mais importantes para este tipo de decisão, algo que a Optimus Information não teve em atenção, por ser muito variável de organização para organização.

Redda [51] fez uma escolha abrangente de vários critérios que ajudariam a perceber e a analisar a facilidade de desenvolvimento de uma linguagem ou *framework* para aplicações móveis. O estudo cingiu-se em perceber quais os critérios que tinham mais preponderância no desenvolvimento móvel. Como conclusão, os critérios mais importantes, aquando da escolha de uma nova *framework* de desenvolvimento móvel, são: plataformas suportadas simultaneamente, o nível de facilidade de aprendizagem da linguagem de programação, recursos que esse tipo de desenvolvimento envolve e velocidade de acesso a dados no decorrer da execução da aplicação.

Ahuja e Johari [52] fizeram um caso de estudo com uma aplicação denominada "Portal de Estudantes", que estava em pleno funcionamento em Android. Ao surgir a necessidade de ser disponibilizada em outras plataformas, com incidência principal no iOS, o estudo consistiu em fazer a migração para uma aplicação híbrida baseada em Cordova, de forma a ser suportada por mais sistemas

operativos móveis e a tentar manter a maior identidade possível da aplicação existente. Após a implementação da aplicação híbrida, foi observada a aplicação original nativa e a migrada híbrida multiplataforma. A aplicação híbrida funcionou da forma esperada, estando, segundo os autores, mais apta para mais sistemas operativos e com resposta mais rápida que a mesma aplicação nativa. O autor concluiu que qualquer alteração de código ou atualização na aplicação já não levava à necessidade de reescrever o código em duas linguagens totalmente distintas por causa da migração implementada, ao contrário do que aconteceria se tivesse partido para o desenvolvimento em Swift para iOS em detrimento do híbrido.

Xanthopoulos e Xinogalos [35] fizeram um estudo em que escolheram várias abordagens de desenvolvimento de aplicações móveis, em vários ambientes e usando várias bibliotecas para os 3 tipos de desenvolvimento (*web*, híbrido e nativo). Testando essas várias abordagens propostas para o desenvolvimento de um protótipo de uma aplicação simples, chegaram à conclusão que, entre as três, a implementação de aplicações híbridas é a mais promissora como solução de desenvolvimento, pois tem um forte apoio de ferramentas comerciais prontas para a produção das mesmas. Contudo, na opinião dos autores, para aplicações de grande escala ou de negócios estabelecidos, em que não existe a necessidade de tempo de mercado ou custos baixos, sendo o foco a apresentação de qualidade ao cliente e segurança, as aplicações nativas continuam a ser a melhor opção em relação às híbridas. De realçar que as soluções de desenvolvimento móvel do tipo *web* foram desde logo alvo de muitas críticas por parte dos autores, devido ao evidente desempenho inferior.

Apesar de todos os estudos referenciados serem convergentes nas vantagens atuais do desenvolvimento híbrido, em relação ao desenvolvimento nativo, existem poucos estudos no que concerne à comparação entre *frameworks* de JavaScript para desenvolvimento de aplicações híbridas, devido a ser um mercado recente. Por exemplo, à data de escrita deste documento, o React Native ainda não tinha uma versão estável lançada. Desta forma, foi necessário fazer um estudo inicial de modo a verificar as vantagens e desvantagens das várias *frameworks* de JavaScript, para uma escolha mais assertiva de qual selecionar para o desenvolvimento na empresa.

3.2 Variáveis selecionadas

Antes da avaliação das diferentes *frameworks*, é necessário ter em consideração quais os critérios que são essenciais no desenvolvimento de aplicações móveis e que podem ter preponderância na escolha do modelo de implementação do negócio por parte de uma entidade ou programador. A seleção das variáveis foi com base no subcapítulo anterior, em estudos idênticos apresentados, mas com outras

frameworks ou linguagens de desenvolvimento móvel [22], [49], [51]. Os critérios selecionados foram os seguintes:

- **Aprendizagem e qualidade da documentação:** Avalia o progresso de um programador durante o seu primeiro contacto com a documentação e respetiva rapidez de aprendizagem; verifica a comunidade ativa para cada uma das *frameworks* incluídas na análise e a qualidade da própria documentação oficial;
- **Custo de desenvolvimento:** Avalia os custos inerentes ao desenvolvimento de uma aplicação através da linguagem selecionada. Este é um dos critérios mais relevantes para pequenas e médias empresas;
- **Emuladores e depuração:** Avalia a capacidade de utilização dos recursos do emulador e sua instalação, mas também o tempo necessário para, após uma alteração de código, ser novamente possível testar a aplicação;
- **Tempo de resposta e velocidade:** Avalia a latência da resposta que uma aplicação pode ter ao toque e a capacidade de resposta na interação com o utilizador;
- **Reconhecimento comercial:** Este critério pretende aferir o reconhecimento em termos de quantidade de utilizadores das aplicações criadas e publicadas pelas respetivas *frameworks* analisadas, de forma a haver garantias de um maior LTS (*Long Term Support*);
- **Reutilização de código e trabalho em equipa:** Avalia a possibilidade da fácil reutilização de código-fonte para outros projetos ou outras vistas/lógicas do mesmo projeto, de forma a aumentar a velocidade de desenvolvimento de um projeto com recursos já criados. Também é avaliada a facilidade que a *framework* proporciona para trabalho em equipa dentro de uma organização;
- **Manutenções e atualizações:** Este critério avalia a capacidade para efetuar modificações e alterações às funcionalidades existentes, para incluir novas funcionalidades e para aplicar melhorias gerais e velocidade de implementação de todas essas modificações.

3.3 Estudo comparativo entre soluções nativas, React Native, Ionic e NativeScript

Para cada uma das *frameworks* foi realizado um estudo comparativo, baseado na análise de aplicações já existentes no mercado e em análises práticas, principalmente no que concerne às variáveis de emuladores e depuração e da aprendizagem e qualidade da documentação.

Os resultados principais do estudo, que está detalhado no Anexo B (publicação aceite na conferência ibérica de Sistemas e Tecnologias de Informação em Cáceres, 2018), são apresentados numa escala de Likert de 1 a 5. O valor 1 representa uma eficácia muito baixa para o critério em análise ou não é aplicado de todo. O valor 2 significa que a eficácia para cumprir esse critério fica abaixo do esperado. O valor 3 é atribuído quando não são encontradas vantagens nem desvantagens significativas. O valor 4 representa que uma certa *framework* ou linguagem tem vantagens que poderão ser aproveitadas. O valor 5 aplica-se quando o critério é satisfeito em tudo o que é pretendido na sua análise.

Na Tabela 1, podem ser consultados os resultados finais do estudo comparativo realizado.

Tabela 1 – Resumo dos resultados do estudo

Critérios	***Nativas***	**React Native**	***NativeScript***	***Ionic***
Aprendizagem e qualidade da documentação	2	5	3	4
Custo de desenvolvimento	2	4	4	4
Emuladores e depuração	4	5	4	4
Tempo de resposta e velocidade	5	5	4	1
Reconhecimento comercial	5	5	1	2
Reutilização do código e trabalho em equipa	2	5	5	4
Manutenção e atualizações	2	3	4	4
Total	***22***	***32***	***25***	***23***

3.4 Considerações finais

De referir que todo o estudo comparativo e todos os estudos posteriores foram realizados tendo em mente apenas as principais *frameworks* de desenvolvimento para JavaScript. Outras linguagens de programação mencionadas neste capítulo devem-se ao facto de, pelo conhecimento que temos, não existirem grandes estudos comparativos até à data de escrita deste documento e, desta forma, ter sido necessário fazer uma análise de outros trabalhos relacionados em que foram feitas comparações de abordagens híbridas e *web* para outras linguagens em relação ao desenvolvimento nativo.

Como principais conclusões do estudo comparativo realizado, constata-se que as soluções nativas, apesar de apresentarem um maior desempenho e reconhecimento comercial, não evidenciam mais vantagens relevantes devido à grande diferença entre linguagens e sistemas operativos. Em relação às *frameworks* de JavaScript, o React Native apresentou os melhores resultados, devido ao apoio por parte do

Facebook na criação de uma *framework* eficiente com componentes o mais aproximados possível dos nativos. Isto explica a inexistência de uma versão estável até à escrita deste documento pelas constantes melhorias que vão sendo aplicadas. O NativeScript, apesar do menor reconhecimento e de ter poucas aplicações de relevo publicadas, apresenta uma velocidade de execução, resposta e teste muito semelhante ao React Native, tendo de melhorar na qualidade da documentação e comunidade para tornar o desenvolvimento mais fácil e viável. Já o Ionic, apesar de ser mais reconhecido do que o NativeScript, apresenta algumas lacunas a nível de velocidade de execução, sendo recomendado apenas para aplicações com pouca complexidade a nível de navegação, animações ou de necessidades reduzidas de sincronização de dados com servidores. Desta forma, foi possível perceber que o desenvolvimento híbrido em JavaScript apresenta mais vantagens em relação ao nativo para as variáveis selecionadas para o estudo, tendo todas as *frameworks* obtido uma classificação superior em relação ao desenvolvimento nativo.

4 Estudo de aprendizagem

Após o estudo comparativo, foi necessário validar os resultados com uma análise mais próxima dos programadores da empresa. Assim, enveredou-se por uma análise mais pragmática aplicando testes práticos aos programadores, de forma a entender qual a *framework* de JavaScript que melhor se adaptava ao desenvolvimento móvel da empresa e a que se apresentava como melhor solução relativamente aos tempos de desenvolvimento, *feedback* dos programadores e resultados dos testes obtidos pelos programadores da empresa. Estes testes foram planeados tendo em mente apenas os profissionais com conhecimentos na linguagem JavaScript, de forma a poder incluir algumas questões com alguma complexidade sem que a linguagem de implementação constituísse um entrave para o resultado final dos testes, fazendo com que as particularidades associadas a cada uma das *frameworks* fossem o foco da avaliação.

No subcapítulo 4.1, é detalhado um plano geral de como foram realizados os testes e sobre os tópicos em que incidiram. Em 4.2, é apresentada a taxonomia de Bloom, que foi o referencial utilizado para a criação do estudo de aprendizagem. Devido à necessidade de adaptação ao contexto prático dos testes de programação, em 4.3, é apresentada uma classificação, adaptada a partir da taxonomia de Bloom, com apenas três níveis. Em 4.4, são apresentados e justificados os itens da prova e quais as variáveis a analisar com os testes. Estes mesmos testes são detalhados em 4.5, com todos os resultados obtidos para cada programador. Em 4.6, é apresentada uma validação e análise de todos os resultados. Por fim, em 4.7, são apresentadas conclusões sobre o estudo de aprendizagem realizado.

4.1 Plano geral

Foram criados 3 testes diferentes para cada uma das *frameworks*: React Native, Ionic e NativeScript, com 15 questões divididas em 5 capítulos (Anexo C), que incidem sobre alguns dos principais conceitos considerados importantes no desenvolvimento de aplicações em *cross-platform*, nomeadamente:

- ***Capítulo 1*** – *Desenvolvimento de vistas*: para ser possível desenvolver uma aplicação móvel, a *interface* de utilizador é quase sempre obrigatória, sendo o que permite uma melhor aceitação da aplicação por parte dos utilizadores;
- ***Capítulo 2*** – *Comunicação entre vistas e lógica de negócio*: para existir algum tipo de interação entre a vista e a lógica de negócio ou funcionalidades da aplicação, tem de haver uma comunicação entre a *interface* e a lógica de negócio;
- ***Capítulo 3*** – *Listagem de dados*: grande parte das aplicações apresenta dados de formas dinâmicas e iterativas, sendo as listas de dados a principal forma de disponibilizar dados dinâmicos em desenvolvimento móvel;
- ***Capítulo 4*** – *Networking*: quase todas as aplicações têm funcionamento reduzido ou praticamente inexistente em modo *offline*. Assim, é necessário existir algum tipo de comunicação entre o cliente (aplicação móvel) e o servidor ou servidores de dados;
- ***Capítulo 5*** – *Diferenciação entre plataformas*: mesmo quando o desenvolvimento é multiplataforma, ou seja, o mesmo código é usado para Android e iOS, é necessário ter muita atenção em como organizar o código, uma vez que poderão existir situações em que têm que ser executadas diferentes lógicas de negócio ou usados diferentes componentes nativos dos sistemas para os quais se está a fazer o desenvolvimento.

4.2 Taxonomia de Bloom

Considerou-se que a realização de atividades de avaliação deveria seguir um referencial que tivesse em conta algumas considerações pedagógicas. Desta forma, realizaram-se atividades de teste com base numa taxonomia de objetivos educacionais. Existem várias taxonomias de objetivos educacionais que permitem planear os conteúdos e avaliar a aprendizagem em termos de níveis de dificuldade crescente e gradual. A Taxonomia de Bloom é uma das mais antigas (tendo sido a primeira taxonomia descrita como um modelo hierárquico para o domínio cognitivo) e que continua a ser amplamente usada [53].

Esta taxonomia é representada em três domínios: cognitivo, afetivo e psicomotor [53]. O domínio cognitivo está relacionado com a memória, o desenvolvimento de capacidades intelectuais e de retenção de conceitos. O domínio afetivo está relacionado com interesses, atitudes e valores de um indivíduo. Já o

domínio psicomotor centra-se em habilidades motoras de um certo indivíduo. No entanto, para os objetivos da análise das *frameworks* de desenvolvimento de aplicações móveis, considerámos que os testes fossem baseados apenas no domínio cognitivo.

O domínio cognitivo da taxonomia encontra-se dividido em seis níveis: Conhecimento, Compreensão, Aplicação, Análise, Síntese e Avaliação. A Tabela 2 sintetiza os objetivos de todos os níveis da Taxonomia de Bloom [54]. O primeiro é o nível de conhecimento, representando níveis mais simples de aprendizagem; enquanto o último nível é o mais complexo e abstrato. Sendo assim, num teste implementado, segundo a taxonomia de Bloom, é geralmente esperado um nível de sucesso maior nos primeiros níveis em relação aos últimos [55].

A Tabela 2 ilustra os objetivos de cada um dos níveis da taxonomia e os verbos normalmente associados e usados para a elaboração de questões desse mesmo nível.

Tabela 2 – Níveis da taxonomia de Bloom e seus objetivos [53]

Nível da Taxonomia de Bloom	***Objetivos***	***Verbos***
Conhecimento	Relembrar processos ou informação específica da atividade que aprendeu	Definir, recordar, nomear
Compreensão	Explicar determinada informação sem associar relações com outros conteúdos	Identificar, localizar, transcrever
Aplicação	Explicar determinada informação sem associar relações com outros conteúdos	Aplicar, demonstrar, usar
Análise	Ser capaz de distinguir informação	Comparar, diferenciar
Síntese	Projetar uma solução a partir da decomposição de um problema	Criar, construir, formular
Avaliação	Analisar uma solução, otimizá-la, identificar e/ou corrigir eventuais erros	Avaliar, gerar

Considera-se que cada nível ou categoria é cumulativo, isto é, depende dos conhecimentos anteriores para dar suporte à seguinte. Os processos caracterizados pela taxonomia devem resultar de algum tipo de aprendizagem. Alêm disso, têm sempre em conta que o indivíduo adquiriu algum tipo de aprendizagem e não que já dominava previamente o assunto.

4.3 Taxonomia de Bloom adaptada

Neste trabalho, pensou-se no enquadramento das perguntas criadas para os testes em três contextos principais: aquisição dos conhecimentos, utilização dos conhecimentos e generalização dos conhecimentos. Desta forma, entendeu-se como necessário fazer uma adaptação e simplificação da taxonomia de Bloom aos testes realizados. Assim, foram considerados os seguintes níveis:

- **Iniciante:** integra, essencialmente, os níveis de Conhecimento e Compreensão da taxonomia de Bloom e tem como objetivos principais que o indivíduo consiga reconhecer, recordar e compreender informação adquirida ao longo da aprendizagem. As perguntas devem-se sempre cingir apenas a conceitos abordados ao longo da aprendizagem e devem ser de fácil interpretação por parte do indivíduo;
- **Intermédio:** integra os níveis de Aplicação e Análise da taxonomia de Bloom e tem como objetivos principais o indivíduo conseguir resolver um problema que difira dos exemplos de aprendizagem, mas em que a pergunta explicite bem ao indivíduo como deve aplicar e de que forma o deve fazer;
- **Avançado:** integra os níveis de Síntese e Compreensão da taxonomia de Bloom e tem como objetivos principais que o indivíduo consiga resolver um problema não trivial, implicando as capacidades de abstração e generalização em relação aos exemplos fornecidos.

Cada um dos níveis contempla obrigatoriamente questões sobre cada tópico. Os formatos das questões são variados e considerou-se que, no âmbito do desenvolvimento de *software*, fosse útil a existência de documentação de ajuda como incentivo à pesquisa individual, de forma a poder progredir e realizar os testes com sucesso. Os testes realizados referentes a cada uma das *frameworks* encontram-se no Anexo C.

4.4 Estruturação dos itens da prova

Foram criados, para posteriormente serem fornecidos, dois documentos. Um documento continha questões (Anexo C) para serem resolvidas após a leitura e interiorização de conceitos explanados num outro documento de tutorial (Anexo D). O documento de tutorial continha a explicação de todos os conceitos cuja aprendizagem se queria testar e ligações para *websites*, para possibilitar o acesso a mais documentação sobre o capítulo. Não só o tutorial como também as questões foram divididas nos 5 capítulos já referenciados, em que cada prova, fornecida ao programador, continha sempre 3 questões associadas a cada capítulo, uma para cada nível de dificuldade (Iniciante, Intermédio, Avançado).

Na fase de estruturação das perguntas por níveis, foram definidos os verbos e o tipo de perguntas que poderiam ser indicadores para cada um dos 3 níveis definidos. Os verbos aplicados em cada nível foram os seguintes:

- **Nível iniciante:** Indique, Selecione;
- **Nível intermédio:** Complete, Demonstre;
- **Nível avançado:** Produza, Gere.

É importante referir que nos documentos de perguntas das três *frameworks* em análise, React Native, Ionic e NativeScript, foram usados sempre os mesmos verbos e tipo de funcionalidade a criar, para cada pergunta de cada capítulo do respetivo nível, de forma a não existirem incoerências no tipo de perguntas.

Todos os testes foram realizados de modo presencial e individualmente, de forma a que, ao longo dos testes, fosse possível recolher *feedback* individual bem como o tempo despendido em cada capítulo. As variáveis obtidas para análise foram as seguintes:

- **Tempos das respostas no capítulo 1, 2, 3, 4 e 5:** Tempo despendido por cada programador a resolver todos os problemas propostos associados a cada um dos capítulos;
- **Resultados do capítulo 1, 2, 3, 4 e 5:** Resultados das respostas dos programadores avaliados numa escala de 0 a 100%, para cada capítulo de cada uma das *frameworks*;
- **Resultado global do teste:** Resultado global, em percentagem, obtido através da média de todas as questões propostas no teste;
- **Resultado por nível Iniciante:** Resultado da média de todas as questões que se enquadram no nível iniciante da taxonomia adaptada referida em 4.3;
- **Resultado por nível Intermédio:** Resultado da média de todas as questões que se enquadram no nível intermédio da taxonomia adaptada referida em 4.3;
- **Resultado por nível Avançado:** Resultado da média de todas as questões que se enquadram no nível avançado da taxonomia adaptada referida em 4.3;
- ***Feedback* livre sobre cada capítulo:** No fim da realização das perguntas associadas a cada capítulo, todos os programadores puderam, de forma livre, dar uma opinião relativamente ao capítulo em questão para essa mesma *framework*;
- ***Feedback* pessoal sobre facilidades e dificuldades na *framework* em geral:** No final de cada um dos testes, o utilizador foi questionado sobre as suas principais facilidades e dificuldades que teve com a *framework*;
- **Avaliação pessoal sobre a realização de uma aplicação simples:** Numa escala entre 0 e 100, o programador avaliou o seu à-vontade em desenvolver uma aplicação simples com

uma lista de dados com estilos já definidos, em que esses dados foram todos obtidos a partir de um servidor através de um *request http*;

- **Avaliação pessoal sobre a realização de uma aplicação para um cliente:** Numa escala entre 0 e 100, o programador avaliou o seu à-vontade em integrar uma equipa de desenvolvimento para um cliente em React Native, Ionic ou NativeScript.

4.5 Resultados

Os testes foram realizados recorrendo a programadores da Crossing Answers, que apresentavam no mínimo mais de 6 meses de experiência profissional em JavaScript. De realçar que nenhum dos programadores que realizaram os testes apresentavam conhecimento em qualquer das *frameworks* de desenvolvimento móvel às quais foram sujeitos.

4.5.1 Teste 1

O primeiro teste foi realizado a um programador com 1 ano de experiência em JavaScript mais propriamente em NodeJS. Os resultados gerais globais do teste foram apresentados na Tabela 3.

Tabela 3 – Resultados gerais do teste 1

Critérios	***React Native***	***NativeScript***	***Ionic***
Tempo de resolução das questões (minutos)	**120**	148,75	130,75
Resultado por nível iniciante	**100%**	75%	90%
Resultado por nível intermédio	**100%**	**100%**	85%
Resultado por nível avançado	**95%**	80%	80%
Resultado final do teste	**98%**	77%	81%

Na Figura 5, podem ser consultados os tempos de desenvolvimento (tempo demorado, em minutos, em cada capítulo para cada *framework*) e os resultados finais por capítulo para o programador testado (em percentagem das respostas corretas).

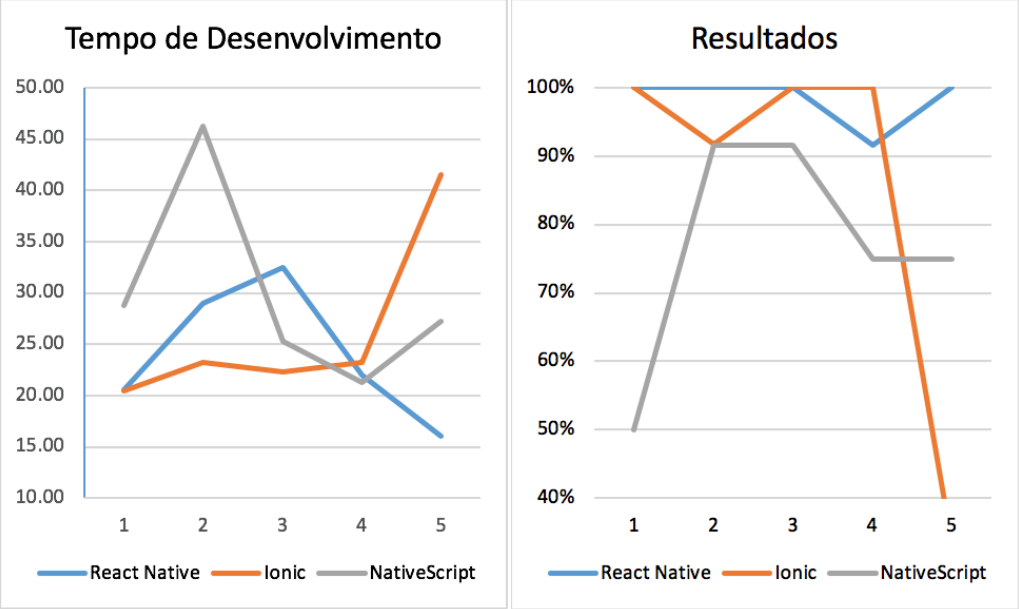


Figura 5 – Tempos de desenvolvimento e resultados finais do teste 1

Percebeu-se que o programador em causa apresentou em NativeScript, em quase todos os capítulos, piores resultados e, em React Native, os melhores resultados.

O *feedback* individual apresentado pelo programador testado está ilustrado na Tabela 4 para React Native, na Tabela 5 para NativeScript, e na Tabela 6 para Ionic.

Tabela 4 – *Feedback* do programador do teste 1 para React Native

Positivo	***Negativo***
"Estilos bons para programadores *web*"	
"Controlo de estados intuitivo"	
"Listas em JSON muito legíveis"	
"Módulo *fetch* nativo bastante fácil"	
"Desenvolvimento baseado em componentes e num só ficheiro simplifica o desenvolvimento"	

Tabela 5 – *Feedback* do programador do teste 1 para NativeScript

Positivo	***Negativo***
"Facilidade em fazer *http requests* por ser igual ao Javascript puro"	"*Grid* é muito complicada com o XML para fazer vistas"
"Alteração de plataformas intuitiva, bastando colocar apenas o nome da plataforma"	"Forma de chamar funções demasiado confusa"
	"XML torna-se ainda mais confuso na criação de listas de dados"
	"Muito difícil de entender como controlar um estado de uma variável num controlador"

Tabela 6 – *Feedback* do programador do teste 1 para Ionic

Positivo	***Negativo***
"CSS puro é muito fácil para quase todos os programadores"	"Separação de ficheiros de código pode ser confusa a chamar funções"
"*ng-repeat* torna a criação de listas de dados fácil e intuitiva"	"Muitas dificuldades em controlar o código para cada uma das plataformas"

Como avaliação final, numa escala de 0 a 100, sobre o à-vontade para realizar uma aplicação simples consoante o tipo de *framework*, o programador registou a seguinte autoavaliação:

- React Native – 80;
- NativeScript – 50;
- Ionic – 60.

Já numa avaliação final de 0 a 100, sobre o à-vontade para integrar uma equipa de desenvolvimento para um cliente, para cada *framework*, o programador registou a seguinte autoavaliação:

- React Native – 30;
- NativeScript – 10;
- Ionic – 10.

4.5.2 Teste 2

O segundo teste foi realizado a um programador com 4 anos de experiência em desenvolvimento de *software*, dos quais 3 anos e meio em JavaScript, mais propriamente em *front-end web*. Os resultados gerais globais do teste estão ilustrados na Tabela 7.

Tabela 7 – Resultados gerais do teste 2

Critérios	*React Native*	*NativeScript*	*Ionic*
Tempo de resolução das questões (minutos)	**72,25**	116,5	93,50
Resultado por nível iniciante	**100%**	75%	**100%**
Resultado por nível intermédio	**100%**	**100%**	**100%**
Resultado por nível avançado	**95%**	90%	**95%**
Resultado final do teste	**98%**	70%	95%

Na Figura 6 podem ser consultados os tempos de desenvolvimento (tempo demorado, em minutos, em cada capítulo para cada *framework*) e os resultados finais por capítulo para o programador testado (em percentagem das respostas corretas).

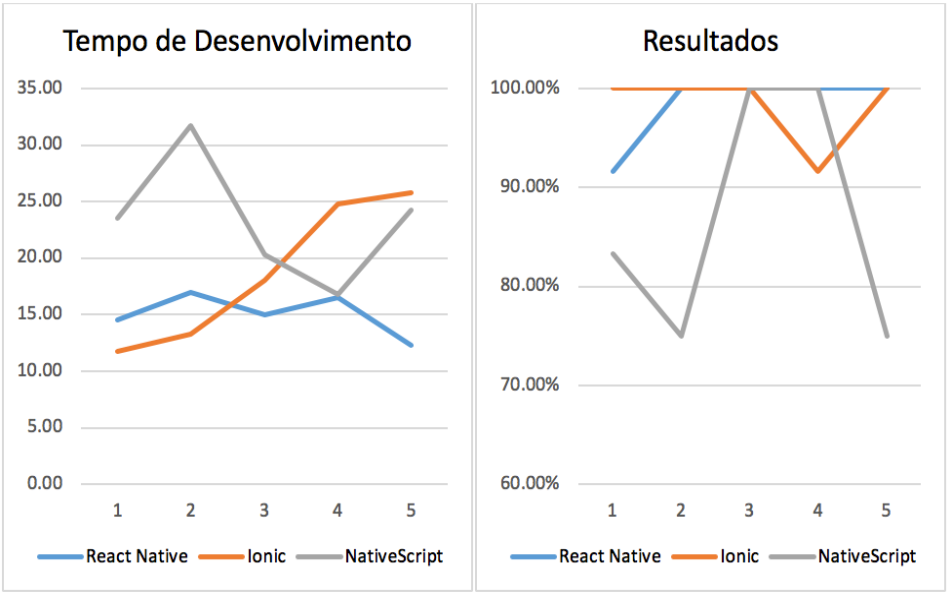


Figura 6 – Tempos de desenvolvimento e resultados finais do teste 2

Neste teste, o programador apresentou novamente em NativeScript resultados mais baixos e tempos de desenvolvimento mais elevados. Já em React Native apresentou tempos de resolução dos problemas muito mais baixos, estando o resultado mais alto dos testes a par com o Ionic.

O *feedback* individual apresentado pelo programador testado está ilustrado na Tabela 8 para React Native, na Tabela 9 para NativeScript e na Tabela 10 para Ionic.

Tabela 8 – *Feedback* do programador do teste 2 para React Native

Positivo	***Negativo***
"Poder utilizar *flex* torna os estilos muito fáceis de aplicar"	"Estilos apesar de serem totalmente iguais à *web* o pormenor de ser *camelCase* pode fazer alguma 'impressão' no desenvolvimento"
"Controlar todo o estado de um componente numa só página é só fantástico"	
"Para fazer listas é apenas necessário só saber trabalhar com *arrays*"	
"Módulo *fetch* nativo simplifica bastante os pedidos"	
"Basta aplicar apenas um ternário, seja na vista ou na lógica, sendo muito intuitivo"	

Tabela 9 – *Feedback* do programador do teste 2 para NativeScript

Positivo	***Negativo***
"Componente de *item* dentro da própria lista torna fácil a criação de listas de dados"	"Muito confuso de fazer vistas; não havia necessidade de um sistema de estilos tão complexo"
"*http request* trabalha exatamente como o JavaScript puro, o que pode facilitar por haver muita documentação nesse formato"	"Passar argumentos por funções é pouco intuitivo e pouco legível"
	"Controlo de variáveis muito confuso"

Tabela 10 – *Feedback* do programador do teste 2 para Ionic

Positivo	***Negativo***
"CSS é muito acessível para programadores *web*"	"Controlo de estado no *React* é mais intuitivo e de mais fácil manuseamento"
"Fazer listas como se fosse um ciclo numa vista é super fácil"	"Muito difícil de entender como se muda de plataforma consoante o sítio do código (vista ou controlador)"
"*Bindings* automáticos na vista"	

Como avaliação final, numa escala de 0 a 100, sobre o à-vontade para realizar uma aplicação simples consoante o tipo de *framework*, o programador registou a seguinte autoavaliação:

- React Native – 100;
- NativeScript – 70;
- Ionic – 90.

Já numa avaliação final de 0 a 100, sobre o à-vontade para integrar uma equipa de desenvolvimento para um cliente, para cada *framework* o programador registou a seguinte autoavaliação:

- React Native – 75;
- NativeScript – 40;
- Ionic – 65.

4.5.3 Teste 3

O terceiro teste foi realizado a um programador com 2 anos de experiência em JavaScript, mais propriamente com JavaScript puro (sem *frameworks*) em *front-end web*. Os resultados gerais globais do teste estão ilustrados na Tabela 11.

Tabela 11 – Resultados gerais do teste 3

Critérios	***React Native***	***NativeScript***	***Ionic***
Tempo de resolução das questões (minutos)	92,00	100,25	**91,25**
Resultado por nível iniciante	**100%**	75%	95%
Resultado por nível intermédio	**100%**	**100%**	90%
Resultado por nível avançado	85%	**100%**	45%
Resultado final do teste	**95%**	92%	75%

Na Figura 7, podem ser consultados os tempos de desenvolvimento (tempo demorado, em minutos, em cada capítulo para cada *framework*) e os resultados finais por capítulo para o programador testado (em percentagem das respostas corretas).

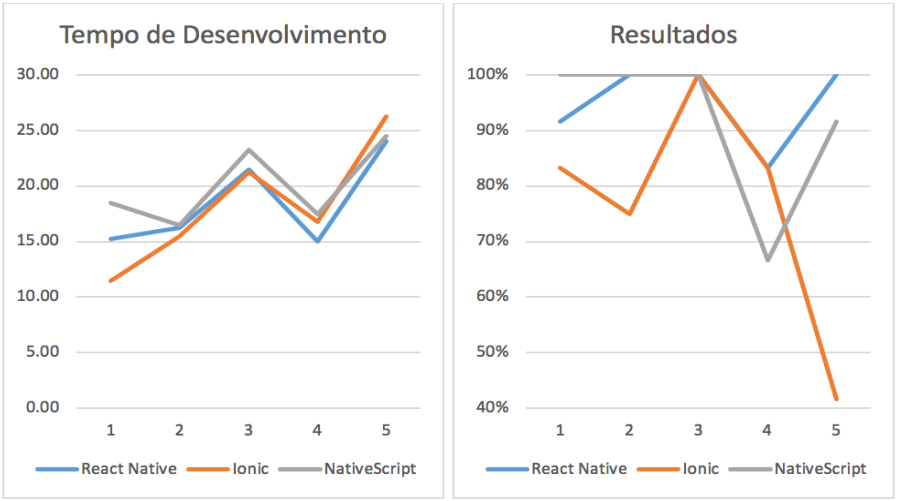


Figura 7 – Tempos de desenvolvimento e resultados finais do teste 3

Para este programador, todos os tempos de desenvolvimento nas 3 *frameworks* foram muito idênticos. Os resultados mais baixos foram obtidos em Ionic, enquanto em NativeScript e React Native obteve resultados globais muito idênticos.

O *feedback* individual apresentado por este programador testado está ilustrado na Tabela 12 para React Native, na Tabela 13 para NativeScript e na Tabela 14 para Ionic.

Tabela 12 – *Feedback* do programador do teste 3 para React Native

Positivo	***Negativo***
"Estilos muito intuitivos para quem percebe de CSS"	"Sintaxe de ES6 pode ser um pouco confusa"
"Muito fácil a construção de vistas com base em componentes|"	
"Bom poder ter todas as possibilidades do JavaScript desde *Async/Await* a um *callback*"	

Tabela 13 – *Feedback* do programador do teste 3 para NativeScript

Positivo	***Negativo***
"Listas muito acessíveis de fazer na vista"	"Para programadores *web* trabalharem com XML e com um formato de estilos totalmente diferente é penoso"
	"*ViewModel* é muito menos percetível em relação à *Stack* do *React*, que é toda na mesma página"
	"Várias opções na documentação para vistas, para controladores, conduzem à existência de erros por não ser muito clara a documentação"

Tabela 14 – *Feedback* do programador do teste 3 para Ionic

Positivo	***Negativo***
"CSS é super intuitivo para criação de vistas"	"Ter ficheiros separados faz com que não seja tão evidente o controlo de variáveis"
"Muito fácil de fazer listas"	"É muito difícil controlar o código por plataformas"

Como avaliação final, numa escala de 0 a 100, sobre o à-vontade para realizar uma aplicação simples consoante o tipo de *framework*, o programador registou a seguinte autoavaliação:

- React Native – 90;
- NativeScript – 75;
- Ionic – 50.

Já numa avaliação final de 0 a 100, sobre o à-vontade para integrar uma equipa de desenvolvimento para um cliente, para cada *framework* o programador registou a seguinte autoavaliação:

- React Native – 70;
- NativeScript – 30;
- Ionic – 10.

4.5.4 Teste 4

O quarto teste foi realizado a um programador com 3 anos de experiência essencialmente em *Backend* (PHP), tendo tido apenas experiência num estágio curricular em JavaScript durante cerca de 6 meses. Os resultados gerais globais do teste estão ilustrados na Tabela 15.

Tabela 15 – Resultados gerais do teste 4

Critérios	*React Native*	*NativeScript*	*Ionic*
Tempo de resolução das questões (minutos)	120	138	**119,25**
Resultado por nível iniciante	**100%**	95%	95%
Resultado por nível intermédio	**100%**	70%	85%
Resultado por nível avançado	**85%**	70%	70%
Resultado final do teste	**85%**	70%	70%

Na Figura 8, podem ser consultados os tempos de desenvolvimento (tempo demorado, em minutos, em cada capítulo para cada *framework*) e os resultados finais por capítulo para o programador testado (em percentagem das respostas corretas).

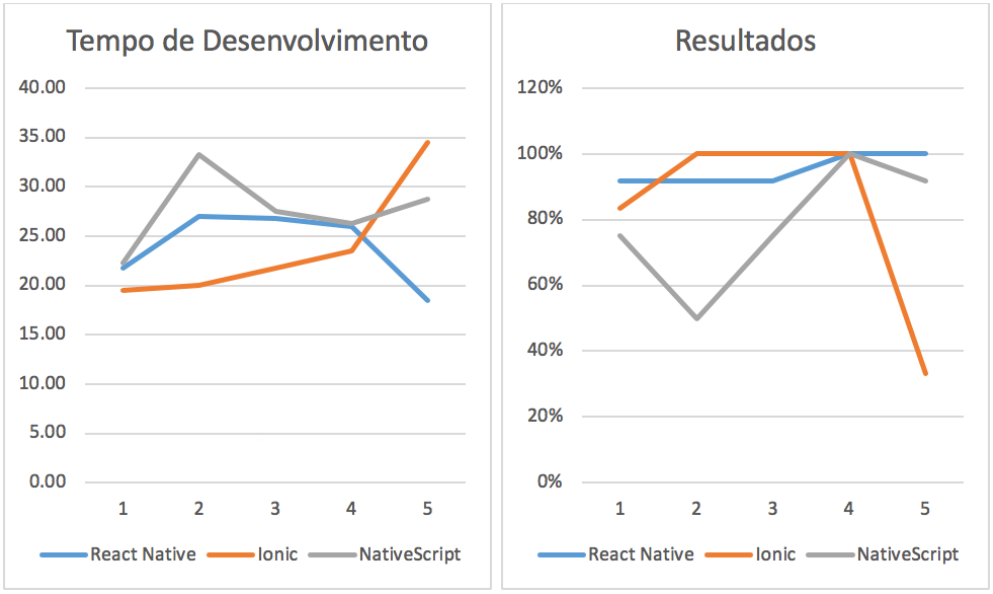


Figura 8 – Tempos de desenvolvimento e resultados finais do teste 4

Neste teste, o programador apresentou em Ionic um tempo de resolução mais rápido seguido daquele que apresentou em React Native, sendo que o pior cenário se verificou, novamente, nos testes de NativeScript, exceto no último capítulo em que o programador demorou mais tempo a resolver o problema de Ionic. Nos resultados, o programador em React Native e em Ionic apresenta resultados globais idênticos. Em NativeScript, apresenta um menor nível de sucesso na resolução das questões realizadas.

O *feedback* individual apresentado pelo programador testado está ilustrado na Tabela 16 para React Native, na Tabela 17 para NativeScript e na Tabela 18 para Ionic.

Tabela 16 – *Feedback* do programador do teste 4 para React Native

Positivo	***Negativo***
"*Tags* de marcação da vista bastante intuitivas"	"Sintaxe de ES6 difícil de entender"
"Bastante personalizável a nível de elementos em listas de dados"	
"É bom ter um módulo para não ter de utilizar *Curls* externos"	

Tabela 17 – *Feedback* do programador do teste 4 para NativeScript

Positivo	***Negativo***
"Diferenciação entre plataformas até é fácil de entender, e tem uma lógica acessível"	"Difícil de controlar as vistas e muito difícil de compreender a estruturação do XML"
	"Fiquei sem perceber como se enviam argumentos nas funções, mesmo depois de pesquisar"
	"Sintaxe demasiado complexa"
	"Documentação pouco objetiva"

Tabela 18 – *Feedback* do programador do teste 4 para Ionic

Positivo	***Negativo***
"Para quem desenvolve *web,* os estilos são bastante intuitivos"	"É muito difícil de compreender em como controlar plataformas ou fazer código diferente para cada plataforma"
"Controlador diferente da vista pode tornar o código mais organizado"	
"Listas com ciclo dentro de uma vista funcionam muito bem"	
"*Binding* de estados bastante fácil"	

Como avaliação final, numa escala de 0 a 100, sobre o à-vontade para realizar uma aplicação simples consoante o tipo de *framework*, o programador registou a seguinte autoavaliação:

- React Native – 80;
- NativeScript – 20;
- Ionic – 95.

Já numa avaliação final de 0 a 100, sobre o à-vontade para integrar uma equipa de desenvolvimento para um cliente, para cada *framework* o programador registou a seguinte autoavaliação:

- React Native – 15;
- NativeScript – 0;
- Ionic – 10.

4.5.5 Teste 5

O quinto teste foi realizado a um programador com 6 meses de experiência em desenvolvimento de *software* em NodeJS & AngularJS. Os resultados gerais globais do teste estão ilustrados na Tabela 19.

Tabela 19 – Resultados gerais do teste 5

Critérios	***React Native***	***NativeScript***	***Ionic***
Tempo de resolução das questões (minutos)	**81**	115,5	98,75
Resultado por nível iniciante	**100%**	90%	**100%**
Resultado por nível intermédio	**100%**	**100%**	85%
Resultado por nível avançado	**80%**	45%	75%
Resultado final do teste	**93%**	78%	83%

Na Figura 9, podem ser consultados os tempos de desenvolvimento (tempo demorado, em minutos, em cada capítulo para cada *framework*) e os resultados finais por capítulo para o programador testado (em percentagem das respostas corretas).

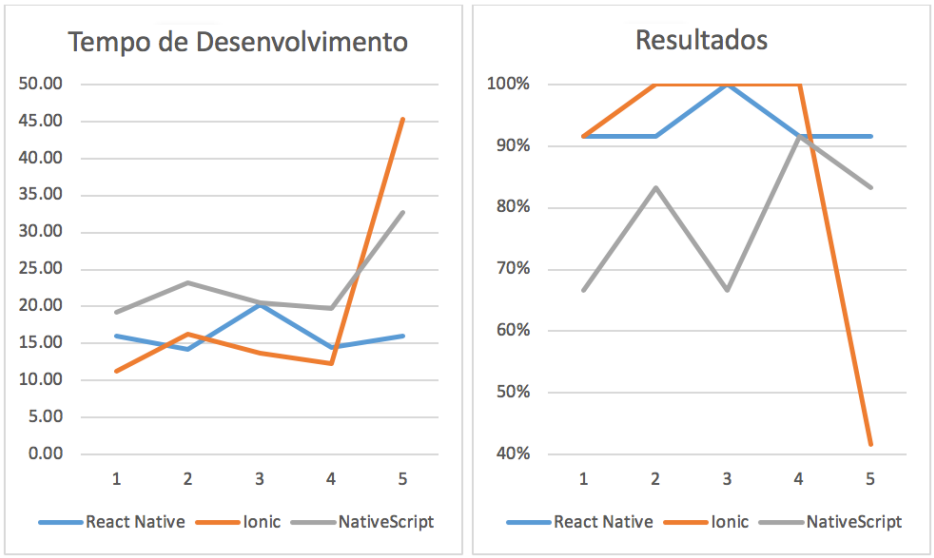


Figura 9 – Tempos de desenvolvimento e resultados finais do teste 5

Neste teste, tanto para Ionic como para React Native, o programador apresentou tempos de resolução do problema idênticos, exceto no último capítulo. O NativeScript, em geral, conduziu sempre o maior tempo de resolução. Nos resultados, o programador em React Native teve uma média de resultados melhor, apesar de em Ionic apresentar mais capítulos totalmente certos. Já em NativeScript, em todos os capítulos, este programador obteve resultados abaixo ou iguais ao React Native e apenas em só um capítulo foi superior ao Ionic.

O *feedback* individual apresentado por este programador está ilustrado na Tabela 20 para React Native, na Tabela 21 para NativeScript e na Tabela 22 para Ionic.

Tabela 20 – *Feedback* do programador do teste 5 para React Native

Positivo	***Negativo***
"Utilização de *flex* da *web* torna muito simples a criação de vistas"	"Perceber sintaxe do ES6 nas listas de dados"
"Muito intuitivo por ser tudo baseado em objetos JavaScript num único ficheiro"	
"Basta apenas uma simples condição em todo o tipo de código e já é possível fazer a diferenciação de plataforma"	

Tabela 21 – *Feedback* do programador do teste 5 para NativeScript

Positivo	***Negativo***
	"Controlo de variáveis difíceis de entender e, por vezes, a própria lógica de execução do código"
	"*fetch* do React Native parece ser mais simples para *requests http*"
	"Muitas formas de diferenciar plataformas, umas na vista, outras para os controladores, torna difícil perceber qual usar"
	"Conceito de *grid*/*column*/*rows* não é muito intuitivo nas vistas"

Tabela 22 – *Feedback* do programador do teste 5 para Ionic

Positivo	***Negativo***
"Para quem desenvolve *web* estilos são iguais, exceto as *tags* do Ionic"	"Controlador separado da vista não é intuitivo. O conceito do React no mesmo ficheiro é melhor"
	"Muito difícil de controlar a diferenciação de plataformas no Ionic"

Como avaliação final, numa escala de 0 a 100, sobre o à-vontade para realizar uma aplicação simples consoante o tipo de *framework*, o programador registou a seguinte autoavaliação:

- React Native – 80;
- NativeScript – 20;
- Ionic – 80.

Já numa avaliação final de 0 a 100, sobre o à-vontade para integrar uma equipa de desenvolvimento para um cliente, para cada *framework* o programador registou a seguinte autoavaliação:

- React Native – 40;
- NativeScript – 0;
- Ionic – 30.

4.5.6 Teste 6

O sexto teste foi realizado a um programador com 5 anos de experiência em desenvolvimento de *Software*, dos quais 3 anos em JavaScript Web *front-end* e NodeJS em *Backend*. Os resultados gerais globais do teste estão ilustrados na Tabela 23.

Tabela 23 – Resultados gerais do teste 6

Critérios	*React Native*	*NativeScript*	*Ionic*
Tempo de resolução das questões (minutos)	**101,75**	133	103,25
Resultado por nível iniciante	**100%**	95%	**100%**
Resultado por nível intermédio	95%	**100%**	**100%**
Resultado por nível avançado	95%	50%	**100%**
Resultado final do teste	97%	82%	**100%**

Na Figura 10, podem ser consultados os tempos de desenvolvimento (tempo demorado, em minutos, em cada capítulo para cada *framework*) e os resultados finais por capítulo para o programador testado (em percentagem das respostas corretas).

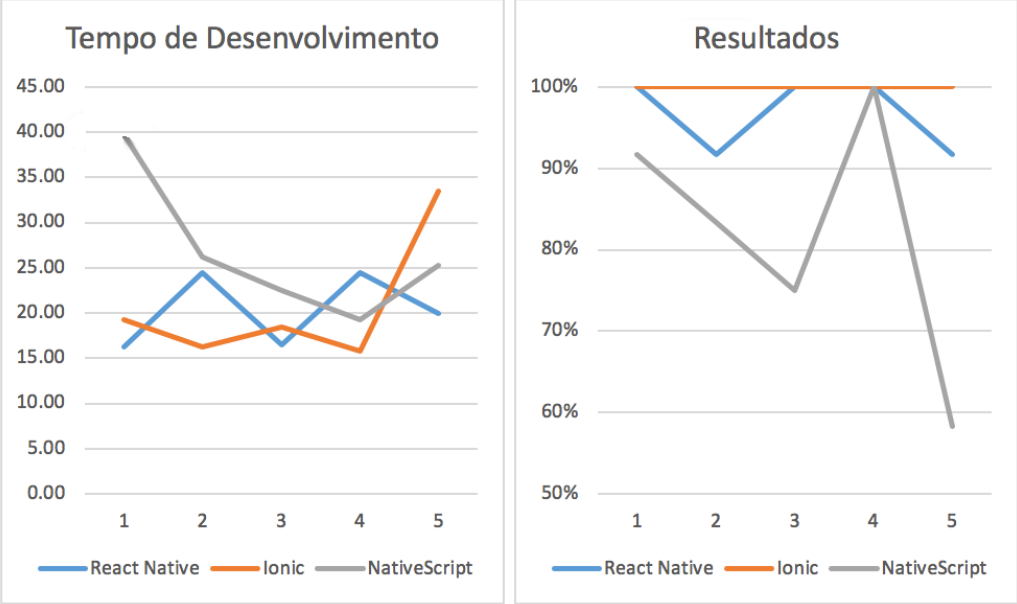


Figura 10 – Tempos de desenvolvimento e resultados finais do teste 6

Neste teste, os tempos de desenvolvimento dos testes em Ionic e React Native foram muito idênticos. O destaque vai para o tempo de resolução dos problemas em NativeScript que, em geral, é sempre mais elevado do que nas outras duas *frameworks*.

Em termos de resultados, o programador teve todas as respostas totalmente corretas para Ionic, sendo que em React Native apenas teve falhas mínimas em declarações ou na passagem de propriedades para componentes. É claramente identificável que o programador em NativeScript ficou muito aquém na concretização das perguntas com sucesso.

O *feedback* individual apresentado pelo programador testado está ilustrado na Tabela 24 para React Native, na Tabela 25 para NativeScript e na Tabela 26 para Ionic.

Tabela 24 – *Feedback* do programador do teste 6 para React Native

Positivo	***Negativo***
"Vistas fáceis de fazer para quem sabe *flex*"	"Sintaxe de *async/await* poderá ser confusa, mas como existem outras possibilidades também não é um total entrave"
"Muito fácil controlar estados"	"Função *fetch* poderia ter mais parâmetros possíveis de configurar"
"Componente *FlatList* estar totalmente otimizado para *mobile* parece um recurso bastante interessante"	
"Desenvolvimento baseado em componentes"	

Tabela 25 – *Feedback* do programador do teste 6 para NativeScript

Positivo	***Negativo***
"Pedido *http* limpo, claro, tal como se usa no JavaScript puro"	"Miserável, imensas opções de fazer uma vista na documentação e nenhuma é clara"
	"Nem sequer dá para perceber o que pertence ao *scope* do JavaScript e o que já se encontra fora desse mesmo *scope*"
	"Não é trivial perceber como se acede/diferenciam funções públicas de privadas"
	"Adaptação fraca nas listas do *ng-repeat* do Ionic, pelo menos podiam manter a lógica igual"

Tabela 26 – *Feedback* do programador do teste 6 para Ionic

Positivo	***Negativo***
"Vistas para programadores *web* é muito simples"	"Ter tudo num só ficheiro é mais percetível para controlar o estado de uma variável"
"*ng-repeat* com *two way binding* torna o desenvolvimento de listas trivial"	"Se houver um serviço para diferenciar plataformas da vista, pode até ser simples; como não havia documentação de serviço nenhuma, pode ser difícil controlar as plataformas e o código específico de cada plataforma na vista"

Como avaliação final, numa escala de 0 a 100, sobre o à-vontade para realizar uma aplicação simples consoante o tipo de *framework*, o programador registou a seguinte autoavaliação:

- React Native – 75;
- NativeScript – 10;
- Ionic – 80.

Já numa avaliação final de 0 a 100, sobre o à-vontade para integrar uma equipa de desenvolvimento para um cliente, para cada *framework* o programador registou a seguinte autoavaliação:

- React Native – 60;
- NativeScript – 0;
- Ionic – 75.

4.6 Análise de resultados

Antes de uma análise dos resultados é discutida a adequação das questões realizadas à taxonomia proposta. Em 4.6.1 é abordada a adequabilidade das questões realizadas e, em 4.6.2, é feita uma análise global de todos os resultados através das diferentes métricas retiradas ao longo dos testes.

4.6.1 Adequação das questões realizadas

Todos os programadores testados sentiram, em quase todos os casos, um aumento no nível de dificuldade das perguntas à medida que avançavam em cada teste, o que era expectável face à aplicação de uma Taxonomia com esses pressupostos. Desta forma, e apesar de poucas exceções, na generalidade, os programadores apresentaram piores resultados em percentagem no nível Avançado e melhores resultados no nível Iniciante da taxonomia adotada. A dificuldade crescente das perguntas foi confirmada através do tempo despendido por cada programador ser maior em cada pergunta do capítulo. Assim, de todos os 90

capítulos e 270 respostas realizadas, em apenas 12 respostas, não se comprovou um aumento de tempo despendido pelo programador em relação à questão anterior. Destas 12 respostas, 8 eram referentes à passagem entre a pergunta 1 e 2 do capítulo 1 dos estilos. Esta situação aconteceu, provavelmente, porque a 1.ª pergunta era de escolha múltipla com 4 opções de código, que envolvia uma leitura da pergunta mais extensa, sendo necessário apenas reconhecer qual das opções não continha erro. Pensa-se que a 2.ª pergunta, apesar de ser já de aplicação ao contexto do capítulo, como implicava resposta mais direta a um problema, levou a que, em alguns casos, o tempo de resolução do problema fosse menor para alguns programadores, não inviabilizando a maior dificuldade inerente à pergunta 2 em relação à 1.

4.6.2 Análise dos resultados

De forma a obter resultados globais relacionados com todos os programadores avaliados, mas sem fazer médias de resultados, foram atribuídos valores absolutos às variáveis em estudo consoante o desempenho em cada uma das variáveis para cada um dos programadores testados. Esta avaliação foi feita numa escala de 0 a 2, em que o valor 2 foi atribuído à *framework* com melhores resultados numa determinada variável, 1 à *framework* com resultado intermédio e 0 à *framework* com pior resultado. Em caso de resultados iguais num critério para duas *frameworks,* se porventura essas mesmas fossem as que obtiveram melhores resultados, eram ambas avaliadas com 2 e a outra, que apresentava piores resultados, com 1. Em caso de igualdade, foi dada a mesma classificação – 2 pontos no caso de se verificar como melhor resultado e 1 ponto no caso da igualdade ser verificada no pior resultado de entre as três *frameworks*.

Na Tabela 27, é possível aferir que grande parte dos capítulos apresentaram uma resolução mais rápida em Ionic, exceto na diferenciação de plataformas (Capítulo 5), em que todos os programadores apresentaram dificuldades generalizadas em implementar um código distinto para cada uma das plataformas (Android e iOS). O React Native apresentou-se como a *framework* mais equilibrada e sem grandes desvios, de tal forma que o resultado global acumulado foi melhor em React Native do que em Ionic. Já o NativeScript foi ao encontro do *feedback* de grande parte dos programadores testados que demonstraram grandes dificuldades em compreender o fluxo de construção e arquitetura da *framework*.

O resumo total dos resultados de todos os programadores para o tempo de resolução dos problemas realizados é apresentado na Tabela 27.

Tabela 27 – Resumo dos tempos de respostas dos programadores

Critérios	***React Native***	***NativeScript***	***Ionic***
Tempo de resposta Capítulo 1	8	0	11
Tempo de resposta Capítulo 2	7	0	11
Tempo de resposta Capítulo 3	7	1	10
Tempo de resposta Capítulo 4	7	4	7
Tempo de resposta Capítulo 5	12	6	0
Tempo de resposta ao teste	10	0	8
Total	**51**	**11**	**47**

Devido ao facto de todos os programadores testados terem bases sólidas de JavaScript, todas as respostas foram geralmente positivas, havendo poucas em que o programador não conseguiu realizar as perguntas. Contudo, e apesar disso, apenas um único programador conseguiu obter um resultado de 100% em todas as perguntas dos 18 testes realizados, tendo sido este resultado alcançado na *framework* Ionic.

Em termos de resultados quase ideais, existiram mais 2 testes em React Native e 1 outro em Ionic em que o resultado ultrapassou os 97%. De referir que nenhum teste de NativeScript conseguiu atingir estas percentagens tão altas.

O React Native foi a *framework* com melhores resultados gerais para todos os programadores, tendo existido melhores resultados apenas em Ionic no capítulo 3, indo totalmente ao encontro do *feedback* reportado pelos programadores que afirmam que o *ng-repeat* em Ionic torna a criação de listas de dados num procedimento quase trivial.

O NativeScript, indo novamente ao encontro do *feedback* dos programadores, foi a *framework* cujos resultados ficaram totalmente abaixo das expetativas. Isto porque a maior parte dos programadores, apesar de entenderem o procedimento necessário para a resolução do problema, a meio da resolução perdiam, por vezes, o controlo do fluxo de desenvolvimento, desencadeando alguns erros em todas as respostas. Verificou-se, novamente, que o *feedback* dos programadores, em relação à grande dificuldade de construção de vistas em NativeScript, através de uma mistura de XML com CSS, foi ao encontro dos resultados globais finais em que o NativeScript apresentou um rácio de respostas corretas demasiado baixo, comparando com as outras *frameworks*.

O resumo da totalidade dos resultados é apresentado na Tabela 28.

Tabela 28 – Resumo dos resultados às questões dos programadores

Critérios	***React Native***	***NativeScript***	***Ionic***
Resultados Capítulo 1	10	2	9
Resultados Capítulo 2	9	3	9
Resultados Capítulo 3	11	4	12
Resultados Capítulo 4	10	7	10
Resultados Capítulo 5	11	4	4
Resultado total do teste	11	1	7
Total	***62***	***21***	***51***

O React Native apresentou melhores resultados em todos os níveis de dificuldade da taxonomia adaptada, demonstrando que, apesar do nível de dificuldade, os programadores conseguiram atingir os resultados corretos mais facilmente em React Native. Já em NativeScript e em Ionic obtiveram os piores resultados duas e uma vez, respetivamente.

O resumo total dos resultados para cada nível de dificuldade é apresentado na Tabela 29.

Tabela 29 – Resultados por taxonomia de Bloom dos programadores

Critérios	***React Native***	***NativeScript***	***Ionic***
Resultados Nível Iniciante	12	3	9
Resultados Nível Intermédio	10	8	5
Resultados Nível Avançado	10	3	7
Total	**32**	**14**	**21**

Todos os programadores fizeram uma avaliação subjetiva num valor entre 0 e 100 para cada uma das duas variáveis, sobre a confiança que teriam no desenvolvimento de uma aplicação num contexto real, tanto numa aplicação simples como numa outra variável no desenvolvimento de uma aplicação para um cliente. Quase todos os programadores elegeram o React Native como a *framework* na qual teriam mais confiança para desenvolver uma aplicação protótipo num futuro próximo. A maior parte dos programadores, aproximaram o Ionic de avaliações semelhantes ao React Native. Porém, o NativeScript, para quase todos

os programadores, apresentou um *feedback* bastante baixo em comparação com o React Native e o Ionic, indo ao encontro de todo o *feedback* demonstrado.

O resumo total dos resultados para cada avaliação subjetiva é apresentado na Tabela 30.

Tabela 30 – Resumo global de confiança demonstrada com cada *framework*

Critérios	***React Native***	***NativeScript***	***Ionic***
Avaliação de confiança para o desenvolvimento de uma aplicação simples	10	8	1
Avaliação de confiança para o desenvolvimento de uma aplicação para um cliente	11	6	2
Total	**21**	**14**	**3**

4.6.3 Análise global do *feedback* dos programadores

De acordo com todo o *feedback* dos programadores, foi recolhido um resumo final, com agrupamento das críticas positivas ou negativas mais relevantes enumeradas por todos os programadores. As críticas positivas e negativas encontram-se ordenadas por número de vezes que foram enumeradas, sendo possível concluir que grande parte dos programadores teceram as mesmas críticas para cada uma das *frameworks*, sem existir muitas incoerências no *feedback* dado por cada um deles.

No React Native, foi bastante referida a facilidade de utilização dos estilos e das listas de dados baseados em *arrays*, mas também o facto de o desenvolvimento da *framework* ser totalmente baseado em componentes, que é um dos focos principais do React Native e que teve *feedback* positivo por parte dos programadores. Por outro lado, o facto de a sintaxe de React Native já ser baseada em JavaScript 2015 ou ES6, levou a que alguns programadores a criticassem, pois não trabalhavam em versões tão recentes de JavaScript.

No Ionic, tal como esperado, foram enumeradas as vantagens de ser um desenvolvimento quase replicado da *web* com CSS puro e *ng-repeat* típico de AngularJS da Web. O conceito de *binding* do Angular também foi um dos pontos referidos fáceis de entender por parte dos programadores. Por ser muito semelhante ao desenvolvimento *web* com AngularJS, apareceram dificuldades em controlar o código consoante a plataforma móvel, tendo este aspecto sido inclusivamente referido por todos os programadores testados. A separação de lógica com a vista, tal como a maior parte das *frameworks web* trabalham, foi criticada em relação ao desenvolvimento do React Native.

O NativeScript foi bastante criticado por todos os programadores, tendo obtido um *feedback* positivo da *framework* relativamente aos *requests http* serem feitos tal como no JavaScript padrão, ao contrário do React Native que já utilizava ES6. No NativeScript, quase todas as opiniões incidiram sobre os aspetos negativos da *Framework*. Destaca-se a criação de vistas em XML, com recurso a Grids, que é descrita como um tipo de implementação muito pobre de tão confusa que se torna. Quase todos os programadores referiram que, mesmo depois do teste, não sabiam como controlar bem a lógica entre uma vista e um controlador, sendo que alguns salientaram mesmo que ficaram sem perceber como chamar funções entre vista e controladores.

O resumo geral do *feedback* de todos os programadores encontra-se na Tabela 31 para React Native, na Tabela 32 para NativeScript e na Tabela 33 para Ionic, em que o número antes de cada *feedback* se refere ao número de vezes que o mesmo *feedback* foi enumerado pelos programadores testados.

Tabela 31 – *Feedback* geral acumulado de todos os programadores para React Native

Positivo	***Negativo***
6 – Estilos são muito intuitivos pelo recurso ao *flex* utilizado no desenvolvimento *web*	**4** – Sintaxe de ES6 por ser recente
5 – Listas através de *arrays* com o componente nativo *FlatList* torna a criação de listas intuitiva	
4 – Controlo de estado de uma variável num ficheiro e desenvolvimento baseado em pequenos componentes torna o desenvolvimento simples	

Tabela 32 – *Feedback* geral acumulado de todos os programadores para NativeScript

Positivo	***Negativo***
3 – *http request* ter a mesma lógica que o JavaScript puro é vantajoso	**6** – Criação de vistas com XML e com recurso a *Grids* é demasiado complexa e difícil de entender
2 – Fazer listas de dados através de iteração é rápido e intuitivo no desenvolvimento	**5** – Difícil de entender como se controla o estado de uma variável no código com a pouca organização do código
2 – Fazer código específico para cada plataforma apenas com uma *tag* Android e iOS na vista é simples	**5** – Não se entende como se passa/chama uma função entre a vista e a lógica de negócio

Tabela 33 – *Feedback* geral acumulado de todos os programadores para Ionic

Positivo	***Negativo***
6 – CSS igual à *web* é vantajoso	**6** – Dificuldade em controlar o código específico para cada uma das plataformas (Android e iOS)
6 – Listas de dados com o *ng-repeat*	**5** – Separação de ficheiros de lógica, mesmo quando são pequenos componentes é confuso
3 – Conceito de *binding* da vista é intuitivo	

4.7 Considerações finais

Todo este estudo se encontra ainda mais detalhado no Anexo E (publicação aceite na conferência ibérica de Sistemas e Tecnologias de Informação em 2019).

Após a aplicação e análise de todos os testes, é facilmente deduzível que o NativeScript não é de todo uma escolha popular para os programadores, não apenas pela dificuldade demonstrada nas atividades realizadas pelos programadores, mas também pelo seu *feedback*.

O Ionic revelou-se como a *framework* mais próxima do React Native em alguns resultados, contudo, e ainda assim, quase sempre com inferioridade em relação ao React Native. Logo, o Ionic poderia também ser uma solução para programadores que sabem AngularJS e para uma aplicação que envolvesse apenas vistas e lógica, em que a *performance* e utilização de *hardware* de cada uma das plataformas não fossem importantes.

O React Native, em quase todos os pontos, apresentou-se como a *framework* mais estável, nunca tendo nenhum resultado baixo em qualquer dos capítulos, não apenas em tempo de desenvolvimento como também em percentagem de acerto das perguntas. Para além disso, o *feedback* sobre o React Native dado pelos programadores foi muito bom, os quais acharam o desenvolvimento através dessa mesma *framework* bastante intuitivo. Este resultado dos testes práticos é totalmente coincidente com o resultado apresentado no capítulo 3 e no trabalho desenvolvido em (Anexo B) [56], concluindo-se também que o React Native era a *framework* JavaScript que se apresentava como melhor alternativa.

5 Finalização do desenvolvimento dos projetos nativos

Para existir uma mudança de uma nova tecnologia no desenvolvimento móvel interno de toda uma organização, é necessário, de forma paralela à análise das vantagens e desvantagens da mudança, terminar os projetos pendentes na tecnologia antiga, ou em caso de evidentes vantagens iniciar novamente o desenvolvimento com a nova tecnologia escolhida. Neste capítulo, irão ser abordados os projetos que ainda estavam a decorrer aquando da realização deste trabalho e que tiveram de ser terminados antes da transição final para desenvolvimento híbrido. Tanto nos projetos Android, com a implementação das aplicações, como nos projetos de iOS, com a gestão dos requisitos do produto final, em ambas houve intervação do aluno no contexto do estágio.

A presente proposta, por parte da Crossing Answers, prendeu-se com o facto de o desenvolvimento dos últimos projetos de aplicações móveis, de forma nativa em Swift e Java, não terem decorrido do modo esperado para a organização. Assim, foi necessário terminar os projetos pendentes que não tinham corrido da melhor forma com colaboradores anteriores e analisar a melhor maneira de os terminar devido ao facto de ninguém na empresa ter experiência profissional nem em Java para Android nem em Swift para iOS. Em 5.1, é descrito o trabalho desenvolvido referente à aplicação "Portugal dos Pequenitos", e em 5.2, o trabalho realizado relativamente à aplicação "Casa Museu Bissaya Barreto".

5.1 Aplicação Portugal dos Pequenitos

Situado em Coimbra, o Portugal dos Pequenitos é desde 8 de junho de 1940, data da sua inauguração, um parque lúdico-pedagógico nascido pela mão e pelo génio de Bissaya Barreto e projetado pelo arquiteto Cassiano Branco, integrando desde 1959 o património da Fundação Bissaya Barreto [57].

O objetivo principal da aplicação era o de criar um guia virtual que pudesse oferecer uma visita guiada e completa ao utilizador com mapa em 3D, galeria, questionários interativos, áudio, *beacons* para localização no próprio mapa em 3D, tentando recorrer a componentes nativos que permitissem uma maior facilidade de utilização por parte do utilizador, sem nunca alterar a identidade das aplicações entre Android e iOS.

O resultado final da aplicação funcional encontra-se disponível na Play Store [58], App Store [59] e, no Anexo F, são apresentados *screenshoots* das principais funcionalidades da aplicação.

5.1.1 Android

A aplicação em Android, iniciada por um antigo colaborador da Crossing Answers, foi submetida a uma transformação total em termos de implementação das funcionalidades, uma vez que não apenas existiam erros em quase todas as funcionalidades previamente desenvolvidas, mas também muitas delas se encontravam por iniciar ou concluir, sendo as principais as seguintes:

- **Autenticação:** A autenticação apenas funcionava através de inserção de *e-mail* e palavra-passe e existiam vários problemas de persistência de sessão e de utilização e armazenamento de *tokens* de sessão. A solução consistiu em refazer toda a autenticação na aplicação, criando persistência correta das sessões e inserindo as funcionalidades de *login* através das contas de Facebook e da Google que, apesar de previstas nos requisitos iniciais, não estavam implementadas;
- **Multi-idioma:** Toda a lógica de alteração de idiomas teve de ser implementada devido à aplicação ainda não apresentar a funcionalidade correta. Através dos requisitos iniciais, era necessário haver a possibilidade de inserir novos idiomas para além dos existentes atualmente na aplicação. Para isso, foi implementada a inserção de novas línguas de uma forma generalizada, não apenas do lado do servidor com os conteúdos dinâmicos como também na aplicação nos textos estáticos;
- **Inserção do mapa em 3D para comunicação com o JavaScript:** A implementação existente apresentava problemas. Durante o lançamento da aplicação, era usual o mapa entrar numa situação de "carregamento infinito". Isso devia-se a um carregamento incorreto da *WebView* do mapa para o Android e do controlo do ciclo de vida da atividade na *WebView*. Essa correção e simplificação do processo de carregamento do mapa levou a que fosse possível criar mais interação entre o utilizador e o mapa, inserindo/alterando funcionalidades no mapa em 3D, de forma a tornar mais simples a utilização do mapa através do *smartphone*;
- **Criação de questionários na aplicação:** Grande parte dos pontos de interesse (pontos de visita) tinham várias questões para o utilizador responder e, apesar de existir um modelo de dados para questionários, toda a funcionalidade de questionário e controlo de questões que já tinham aparecido para o utilizador estavam por desenvolver corretamente. Desta forma, foi implementada toda a lógica dos questionários;
- ***Beacons*:** Os *Beacons* eram uma das funcionalidades da aplicação mais desconhecida, pois a única coisa ainda realizada era a escolha de uma biblioteca para facilitar a comunicação

entre os *Beacons* e a aplicação, tendo sido analisada toda a documentação associada à biblioteca [60]. Posteriormente a isso, foi implementado todo o processo, desde a análise de sinais de *Beacons* ao tratamento do sinal mais relevante e à repercussão do comportamento de piscar do ponto de interesse ao qual o utilizador estaria próximo;

- **Áudio:** Toda a lógica de carregamento e reprodução de áudio já se encontrava implementada, contudo não existiam controlos de áudio de reprodução, pausa e mudança para outro áudio, reproduzindo, por vezes, o anterior. Foi feito um *refactoring* de todo o código e inserção de todas as funcionalidades que ainda faltavam implementar;
- **RGPD:** Todo o serviço RGPD (Regulamento geral sobre a proteção de dados) associado à aplicação e à Fundação Bissaya Barreto foi implementado devido à alteração da lei nacional, não sendo uma funcionalidade prevista inicialmente;
- **Publicação:** Preparação de todos os requisitos necessários para publicação da aplicação Android e respetiva disponibilização na Play Store.

5.1.2 iOS

A aplicação em iOS encontrava-se no mesmo ponto de situação em relação à de Android, levando igualmente a uma modificação total em termos de implementação das funcionalidades e de concordância com a de Android, pois não estavam coerentes, em algumas *interfaces* e funcionalidades. Para não estender o tempo de estágio demasiadamente e terminar os projetos de desenvolvimento nativo da empresa, foi subcontratado um colaborador externo especializado em Swift apenas para terminar o desenvolvimento dos projetos pendentes em Swift. No entanto, o aluno estagiário teve ainda de realizar as seguintes tarefas:

- **Rastreamento dos Requisitos:** Análise da aplicação de iOS já existente e verificação de funcionalidades por implementar ou de erros implícitos que afetavam a execução da aplicação. Posteriormente, de forma a que o colaborador externo pudesse entender tudo o que era pretendido desenvolver, foram definidas *user stories*, consoante as funcionalidades em concordância com a aplicação em Android que foi, primeiramente, desenvolvida;
- **Gestão do desenvolvimento:** Ao longo do desenvolvimento, foram feitos os ajustes das prioridades da implementação consoante o *feedback* por parte do cliente e das estimativas de desenvolvimento de cada *user story*;
- **Validação do desenvolvimento:** Ao longo do desenvolvimento, foram validadas várias *user stories* a nível de funcionalidade, verificando sempre a coerência com a aplicação em Android. A validação era feita através de uma *Kanban board* [61], ferramenta utilizada para

gerir o trabalho em pequenas tarefas. Essa validação era feita alterando o estado na *Kanban board* de **ready for test** para ***done***, como apresenta a Figura 11;

- **Publicação:** Foi também realizado, já a nível interno, todo o processo de publicação através das chaves privadas da própria empresa para colocar a aplicação na App Store.

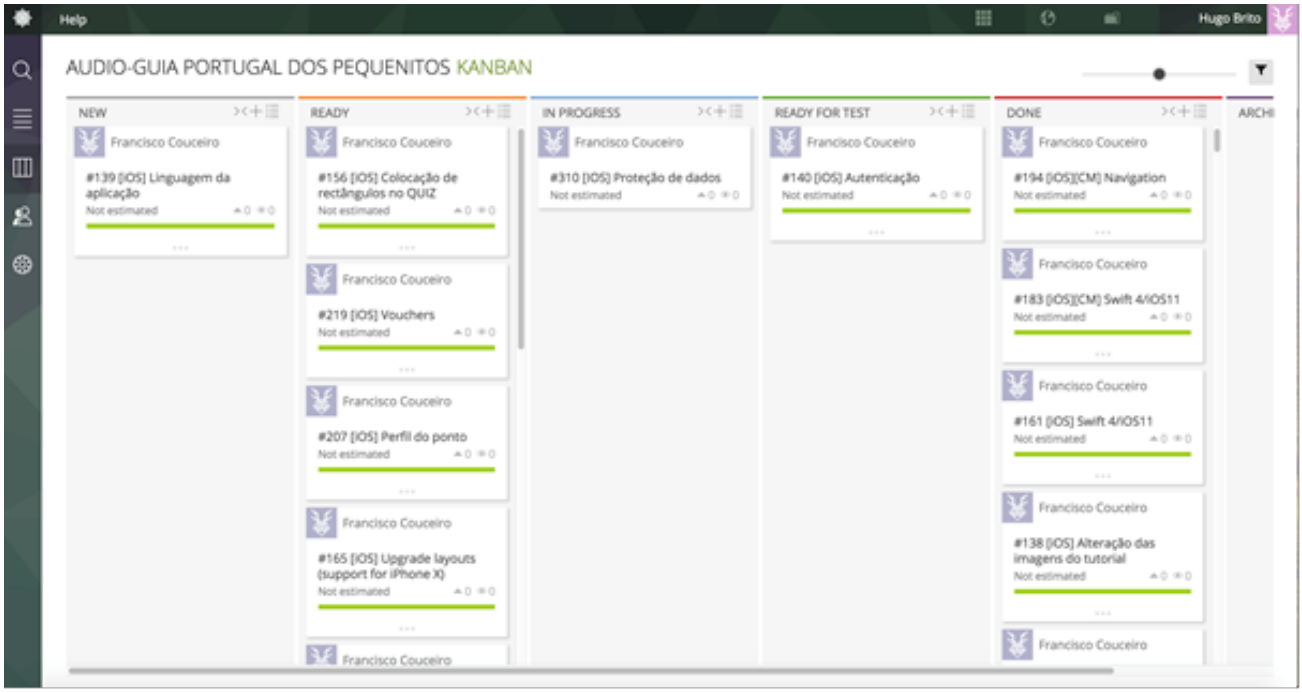


Figura 11 – *Kanban board* do projeto de iOS

5.2 Aplicação Casa Museu Bissaya Barreto

A Casa Museu Bissaya Barreto foi, durante quase 50 anos, a residência particular de Fernando Bissaya Barreto (1886-1974). Projetada pelo arquiteto Fiel Viterbo, a construção desta ampla residência foi iniciada em 1923 e concluída em 1925 [62].

Nesta aplicação, pretendida para o mesmo cliente (Fundação Bissaya Barreto), o objetivo passou por fazer uma cópia das funcionalidades já criadas na aplicação, removendo algumas que não fariam sentido neste caso e fazendo apenas a alteração para um mapa em 2D das divisões da casa museu, cores globais da aplicação e servidor-fonte para obter informações dos pontos de interesse.

O resultado final da aplicação Casa Museu Bissaya Barreto encontra-se disponível na Play Store [63], App Store [64] e, no Anexo G, são apresentados *screenshoots* das principais funcionalidades da aplicação.

5.3 Considerações finais

Apesar de ter sido apenas uma integração num projeto já existente, que ficou por concluir por colaboradores antigos, foi notória a necessidade de mudança de paradigma por parte da empresa Crossing Answers. Os dois projetos, "Portugal dos Pequenitos" e "Casa Museu Bissaya Barreto", foram arrastados durante um período de tempo bem mais alargado do que o esperado pela organização. Esta situação deveu-se, entre outros fatores, à falta de colaboradores com conhecimento em Java para Android e de Swift para iOS, por todo o desenvolvimento de *software* da empresa ser centrado em JavaScript. No contexto do estágio, tal como referido, foi possível terminar toda a parte do Android em Java, sendo que o iOS, através de Swift, teve de ser subcontratado a outra pessoa externa, de forma a poder dar os projetos como terminados.

Esta experiência no desenvolvimento nativo dentro da organização veio tornar mais vincada a necessidade de a empresa centralizar todo o desenvolvimento apenas numa única linguagem, que possa integrar diferentes projetos, desde *mobile, back-end, front-end web* ou aplicações *desktop*. Isto porque a empresa, dependendo dos projetos que recebe, por vezes tem uma maior necessidade em integrar mais colaboradores numa certa especialidade.

Por fim, uma conclusão também muito importante foi a dificuldade que houve em manter a coerência em termos de funcionalidades e de vistas entre Android e iOS, que também dificultou o desenvolvimento da aplicação. Contudo, importa salientar que a complexidade deste processo pode ser reduzida com a adoção de uma das *frameworks* híbridas referidas nos capítulos anteriores.

6 Mudança do desenvolvimento móvel da organização

Este capítulo aborda a forma como foi realizado todo o processo de mudança do desenvolvimento móvel de Java/Swift para a *framework* escolhida, o React Native. São abordados diversos tópicos que vão desde a razão da escolha da *framework*, passando pelo planeamento do processo de transição e definição de metas, até à produção de toda a documentação para integrar, facilmente e sempre que necessário, um novo membro no desenvolvimento móvel da organização.

Foi preciso interiorizar bastantes conceitos sobre JavaScript e React Native, realizar análise de literatura sobre as melhores práticas, de acordo com a documentação fornecida pelos criadores da *framework* [65] e pelo livro "Learning React Native", de Kirupa Chinnathambi [66].

Em 6.1, é descrita a escolha final da *framework* de desenvolvimento a adotar por parte da empresa e a razão dessa escolha. Em 6.2, é detalhado todo o planeamento da transição dentro da empresa e o que foi feito em cada um dos pontos desse mesmo plano. E, por fim, em 6.3, é apresentada uma conclusão sobre a mudança do desenvolvimento móvel dentro da empresa.

6.1 Escolha da *framework*

Apesar de atualmente existir um grande número de *frameworks* de JavaScript para desenvolvimento de aplicações móveis, o principal critério para a seleção inicial entre React Native, NativeScript ou Ionic foi o facto de estas serem baseadas em AngularJS e ReactJS. A relação com o AngularJS foi importante uma vez que este, de momento, está a ser utilizado pela Crossing Answers para desenvolvimento *web*. Relativamente ao ReactJS, este poderia ser uma boa possibilidade futura caso o React Native fosse inserido com sucesso no desenvolvimento móvel da empresa. Esse foi o critério-base escolhido pela Crossing Answers para manter a uniformização entre o desenvolvimento *web* e móvel.

Após isso, e através do estudo comparativo feito entre as três *frameworks* no capítulo 3 [56], concluiu-se que o React Native apresentava grandes vantagens em relação ao NativeScript e, principalmente, em relação ao Ionic, que apresentou lacunas óbvias a nível do desempenho. As vantagens do React Native, em relação às outras foram de tal forma evidentes que a escolha da *framework*, logo à

partida, estaria determinada. Para além disso, o estudo prático, abordado no capítulo 4 com os programadores da Crossing Answers, revelou que o React Native era o que obtinha um *feedback* mais positivo, melhores tempos de resolução e melhores resultados. O Ionic, apesar de resultados ligeiramente mais baixos, teve também um bom *feedback* por parte dos programadores; porém, a nível de desempenho, ainda é atualmente o pior dos três. Já o NativeScript, apesar de apresentar bons resultados de desempenho, ficou completamente fora de questão para utilização na empresa quando quase todos os programadores criticaram a forma de desenvolvimento nessa *framework*.

Em concordância com o resultado dos estudos teóricos e práticos, outro fator muito importante para a escolha final ter recaído no React Native foi a comunidade de grandes empresas que o suporta e que têm ajudado na rápida evolução e estabilização dessa *framework*, como, por exemplo, o Airbnb, Facebook, Instagram, Skype, Tesla, Uber, Pinterest, entre outros [67].

6.2 Planeamento do processo de transição

Após terminar todos os processos associados ao desenvolvimento nativo em Android e todos os estudos comparativos entre *frameworks*, que foram feitos sempre em paralelo, foi necessário definir todo o processo de transição para o React Native. Durante todo esse processo, não houve projetos paralelos a nível de desenvolvimento móvel em toda a organização. O processo de transição foi efetuado pelo aluno estagiário, através da realização das tarefas listadas na Tabela 34. Este processo teve uma duração de cerca de dois meses e meio.

Tabela 34 – Planeamento de tarefas associadas à transição

Tarefa	**Duração**	***Início***	***Fim***
1 – Aprendizagem de JavaScript	**2 semanas**	**19/02/2018**	**02/03/2018**
Aprender conceitos de JavaScript	1 semana	19/02/2018	23/02/2018
Aprender JavaScript 2015 (ES6)	1 semana	26/02/2018	02/03/2018
2 – Aprendizagem de React Native	**1 semana**	**05/03/2018**	**09/03/2018**
Ler documentação base de React Native	1 dia	05/03/2018	05/03/2018
Fazer tutoriais Learn Basic React [68]	1 dia	06/03/2018	06/03/2018
Fazer tutoriais Learn Advanced React [69]	1 dia	07/03/2018	07/03/2018
Ver tutoriais de criação de aplicações simples	2 dias	08/03/2018	09/03/2018
3 – Criação de aplicação para aprendizagem	**2 semanas**	**12/03/2018**	**23/03/2018**
Criar clone de NetFlix com recurso a tutoriais	2 semanas	12/03/2018	23/03/2018
4 – Análise de literatura sobre React Native	**1 semana**	**26/03/2018**	**30/03/2018**
Analisar livro Learning React Native [66]	3 dias	26/03/2018	28/03/2018
Analisar na web boas práticas para React Native	1 dia	29/03/2018	29/03/2018
Recolher boas práticas	1 dia	30/03/2018	30/03/2018
5 – Criação de Boilerplate	**3 semanas e 1 dia**	**03/04/2018**	**24/04/2018**
Integrar Navegação	3 dias	03/04/2018	04/04/2018
Ficheiro utilitário para pedidos HTTP	2 dias	05/04/2018	06/04/2018
Integrar arquitetura redux	2 dias	09/04/2018	10/04/2018
Criar exemplos de Actions e Reducers	2 dias	11/04/2018	12/04/2018
Criar persistência de dados	2 dias	13/04/2018	16/04/2018
Definir estruturação de pastas	1 dia	17/04/2018	17/04/2018
Definir packages	2 dias	18/04/2018	19/04/2018
Definir ficheiros de configuração	1 dia	20/04/2018	20/04/2018
Integrar o Fabric	2 dias	23/04/2018	24/04/2018
6 – Cobertura de Código (Linting)	**3 dias**	**26/04/2018**	**30/04/2018**
Implementar ESLint	3 dias	26/04/2018	30/04/2018
***7* – Definição do ambiente de desenvolvimento**	**2 dias**	**02/05/2018**	**03/05/2018**
Definir as regras do ambiente de desenvolvimento	2 dias	02/05/2018	03/05/2018
8* – Documentação do *Boilerplate	**2 dias**	**04/05/2018**	**05/05/2018**
Escrever documento de auxílio ao Boilerplate	2 dias	05/05/2018	04/05/2018

Nos seguintes subcapítulos, será abordada a forma como foram realizadas todas as tarefas.

6.2.1 Aprendizagem de JavaScript

Como passo inicial, foi necessário aprender e entender todos os conceitos de programação associados ao JavaScript. Uma vez que toda a base de desenvolvimento da Crossing Answers é em JavaScript, já existia alguma documentação interna da empresa a seguir para aprender da melhor forma a linguagem e os conceitos principais de JavaScript. Posteriormente, foi necessário focar a aprendizagem em JavaScript 2015 (ES6) [70], o principal padrão JavaScript suportado pelo React Native, que contém algumas funcionalidades que diferem em relação ao padrão utilizado pela empresa de que se destacam, por exemplo, as seguintes funcionalidades, tais como:

- **Arrow functions:** São funções com uma sintaxe mais curta e que isola o *scope* do JavaScript;
- **Declarações de variáveis:** As variáveis mudaram para identificadores que não podem ser reatribuídos (*const*) ou variáveis dinâmicas que podem ser reatribuídas (*let*) e que podem ser utilizadas somente no bloco de código em que estão definidas, ao contrário de (var), que é a declaração mais fraca disponível e a única existente nos padrões mais antigos de JavaScript [71];
- **Destructuring:** É uma expressão JavaScript, que possibilita descompactar valores de matrizes ou propriedades de objetos em variáveis distintas;
- **Operações otimizadas sobre arrays:** Surgiram com o ES6, permitindo fazer operações sobre *arrays* de forma otimizada, como, por exemplo [72]:
    - **array.find** – para encontrar um objecto num array;
    - **array.filter** – para filtrar um array por uma determinada condição;
    - **array.map** – para fazer operações num array e retornar para outra variável, de forma a manter a imutabilidade dos dados originais.
- **Classes:** Com o ES6, é possível criar classes e instanciar objetos, contrariamente aos padrões anteriores de JavaScript em que não existia o contexto de uma classe.

De referir que, apesar da Crossing Answers ter todo o seu desenvolvimento assente em JavaScript, não era utilizado JavaScript 2015 (ES6), ou seja, a mudança para React Native fez com que, de forma progressiva, toda a empresa fosse adotando as práticas de ES6, desde o *back-end* em NodeJS com Express até ao *front-end web* em AngularJS. Atualmente, toda a empresa faz todo o seu desenvolvimento baseado em ES6, demonstrando já uma das boas influências que o processo de transição para React Native trouxe.

6.2.2 Aprendizagem de React Native

Após a ambientação inicial ao JavaScript e ao padrão JavaScript 2015, foi necessário passar por um processo de aprendizagem de React Native, lendo uma extensa documentação de React Native [65] e

fazendo tutoriais de aprendizagem [68], [69], [73], de React Native e de ReactJS, por estarem assentes nas mesmas bases.

Após a interiorização do funcionamento do JSX (extensão de ficheiro normalmente utilizada em ficheiros de componentes React), foram feitos pequenos exemplos de aplicações simples para pôr em prática os conhecimentos obtidos e perceber quais os pontos que causavam maior dificuldade no desenvolvimento de aplicações móveis em React Native. Entre estas etapas de aprendizagem, podem salientar-se os exemplos da navegação entre vistas ou os de controlo do ciclo de vida dos componentes de React, através dos estados/propriedades e dos dados mutáveis/imutáveis.

6.2.3 Criação de aplicação para aprendizagem

Foi proposto fazer uma aplicação mais complexa que envolvesse a maior parte dos conceitos-base para uma aplicação em React Native. Dessa forma, foi desenvolvido um pequeno clone da aplicação Netflix, para Android e para iOS, com recurso a alguns vídeos de tutoriais de aprendizagem. Foram sendo criadas pequenas funcionalidades baseadas na arquitetura *redux* [74], uma arquitetura de desenvolvimento muito associada ao React e que permite o controlo geral de dados da aplicação numa *store,* explicada no capítulo 6.2.5. A navegação começou a ser baseada no *react-navigation* [75], a biblioteca mais utilizada para controlo de navegação entre vistas em React Native. O resultado final do protótipo criado pode é apresentado na Figura 12 e Figura 13.

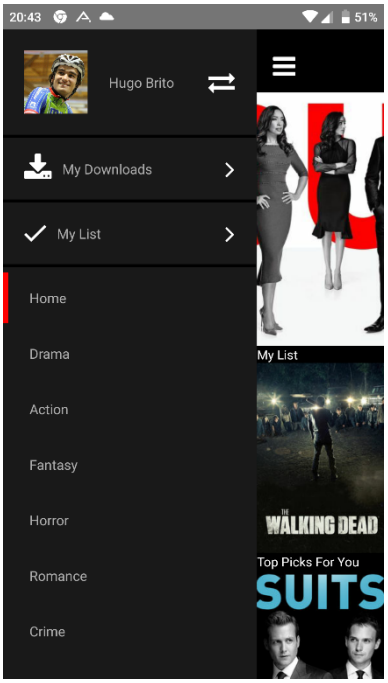


Figura 12 – Resultado final da aplicação de aprendizagem I

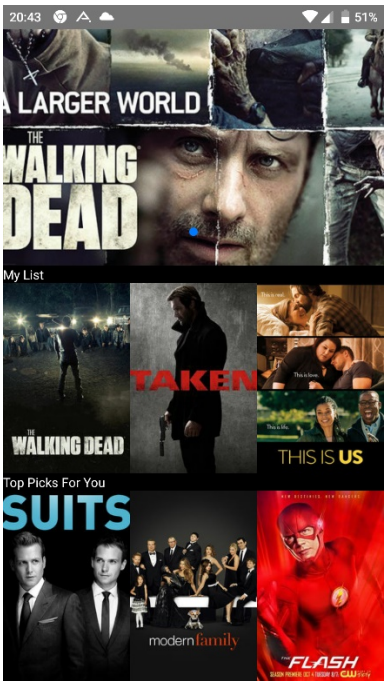


Figura 13 – Resultado final da aplicação de aprendizagem II

6.2.4 Análise de literatura sobre React Native

Existindo já a noção de como funcionava o desenvolvimento em JavaScript, através de React Native, com todo o estudo e prática por meio da criação de aplicações simples e de um exemplo mais completo, foi necessário fazer uma análise de literatura existente sobre React Native, de modo a recolher dados sobre as melhores práticas de desenvolvimento que ainda não tinham sido adotadas nos exemplos criados. Algumas das boas práticas recolhidas com a leitura de várias documentações [67] e que ainda não tinham sido utilizadas foram:

- **Indexação de componentes:** Permite que, quando os componentes são importados em qualquer parte do programa, não seja necessário especificar todo o seu caminho de localização na estrutura de pastas do projeto, mas sim usar um indexador no ficheiro raiz dos componentes e importar o componente desse mesmo indexador, como no exemplo da Figura 14.

```
import { Component1, Component2, Component3 } from '../Components';
```

Figura 14 – Importação de componentes após indexação

- **Dependências:** Uma das principais desvantagens do React Native é ainda não ter uma versão estável publicada. Assim, é necessário bloquear as versões de todas as dependências, de forma a não sofrerem *updates*, eventualmente incompatíveis com a versão do React Native utilizada ou com outras dependências do projeto. Isto porque, normalmente, quando se instala uma nova dependência num determinado projeto, essa mesma dependência aparece no formato ^1.2.5, caso seja uma dependência que, por exemplo, se encontra na versão 1.2.5, fazendo o símbolo "^" antes da versão, o que significa que a dependência possa sofrer *updates* para a mais recente. Devido ao React Native não ter ainda uma versão estável à data de escrita deste documento, isso faz com que a melhor opção seja bloquear a dependência, removendo o "^", de forma a ser sempre compatível com a versão utilizada do React Native num determinado projeto;
- **Componentes baseados em classes:** A criação de componentes baseados em classes tornou-se uma abordagem mais fluída e mais rápida do que a criação de componentes sem estado ou chamados de PureComponent. Assim, estes componentes baseados em classe do tipo *statefull* tornam mais fácil cada componente ter a responsabilidade de poder controlar o seu próprio estado e ser alterado apenas quando necessário, através das propriedades que são influenciadas por um componente pai. Os componentes sem serem baseados em classe do tipo *stateless* apenas devem ser utilizados para pequenas funcionalidades que nunca envolvam um estado de um componente;
- **Utilização de react-native-typography** [76]**:** O *react-native-typography* permite que, consoante a plataforma onde é executada a aplicação (Android ou iOS), a fonte padrão seja mantida para essa mesma plataforma, fazendo com que o utilizador mantenha toda a experiência nativa;
- **Componentes reutilizáveis:** Ao invés de criar componentes de maior dimensão, a programação através da biblioteca React é caraterizada por ter uma opção óbvia por múltiplos componentes mais pequenos. Na prática, esta técnica traz vantagens, quer ao nível da implementação quer ao nível do desempenho da aplicação. A melhoria no desempenho deve-se ao facto de, quando um ecrã é apresentado, o estado e as funcionalidades específicas de cada um dos pequenos componentes serem geridos pelo próprio. Caso não fosse feito desta forma, seria necessário aumentar a complexidade de toda a lógica associada à classe do ecrã.

6.2.5 Criação de boilerplate

De forma a facilitar a realização de novos projetos, foi identificada a necessidade de criação de um projeto-base que incluísse todas as funcionalidades importantes e comuns para qualquer projeto relacionado com aplicações móveis na empresa. Este tipo de projeto designa-se por Boilerplate, deixando de ser necessário integrar, repetidamente, as funcionalidades em cada novo projeto. Outra mais-valia que se obtém refere-se à estabilização das versões usadas das diversas bibliotecas, evitando problemas de incompatibilidade entre as mesmas em integrações futuras. As principais funcionalidades integradas no *boilerplate* que foi criado foram as seguintes:

- **Navegação:** A navegação é o que permite a uma aplicação a passagem entre vistas através de diferentes toques, acedendo a diferentes funcionalidades consoante a navegação feita pelo utilizador. Foi selecionado o *react-navigation* com a versão 1.0, que é a versão que oferece mais compatibilidade com as diferentes dependências. O *react-navigation* é a solução de navegação mais utilizada no desenvolvimento de aplicações em React Native e é desenvolvida inteiramente em JavaScript. Assim, foram criados os seguintes componentes genéricos, associados à navegação e que normalmente são utilizados na generalidade de todas as aplicações móveis:
    - **Tabs:** Correspondem aos ícones de navegação inferiores ou superiores embutidos diretamente numa vista, como os representados na Figura 15 (botões Home, First e Second). Este é um dos estilos mais utilizados, atualmente, nas mais conhecidas aplicações móveis, que permitem uma navegação fluída e rápida para acesso às principais funcionalidades da aplicação. Foi, desta forma, criado um componente genérico chamado *tabs-navigator*, focado nas necessidades da empresa. Neste, apenas é necessário especificar os nomes de cada uma das *tabs*, a quantidade de *tabs* que vai ter a aplicação, o estilo e eventuais ícones das *tabs*, sendo automaticamente possível ter um componente de *tabs* na aplicação que se está a criar;

Figura 15 – Exemplo de *Tabs* [77]

- **Stacks:** Uma *stack* é um conjunto de ecrãs que estão associados à mesma funcionalidade ou, no caso de uma aplicação com *tabs*, é normalmente associada uma *stack* de vistas a cada uma das *tabs*. Ou seja, para o exemplo mostrado na Figura 15, iriam ser criadas 3 *stacks*, cada uma associada a uma das *tabs*. Desta forma, foi criado um componente genérico chamado *stack-navigator* com o qual é possível realizar toda a navegação entre vistas da mesma *stack* de funcionalidades e, também, controlar toda a pilha de vistas associadas;
- **Drawer:** Outro dos estilos que é bastante utilizado para aplicações com muitas funcionalidades principais é a *drawer* lateral, também conhecido como *menu* "hambúrguer". Este *menu* permite, através de um gesto ou clique, abrir um componente sobre a vista atual, a partir do qual existe a possibilidade da navegação para as diferentes funcionalidades principais da aplicação. Foi também criado um componente genérico denominado *drawer-navigator*, fazendo com que apenas seja necessário configurar quais as *stacks* de funcionalidades que vão ser introduzidas como opções, ficando automaticamente integrado na aplicação.

- **Ficheiros utilitários para pedidos HTTP:** Apesar de o React Native fornecer um módulo de *fetch* que permite trabalhar com *requests*, existiu a necessidade de criar funções mais específicas. Deste modo, foi criado um ficheiro unicamente de funções utilitárias que permitem fazer todo o trabalho com *requests http*. Neste ficheiro, foram criadas funções genéricas de trabalho com uma REST API, com funções *get*, *post*, *put* e *del*, fazendo com que a complexidade associada à comunicação com uma REST API e de tratamento correto das respostas fosse simplificada para todos os programadores que venham a usar o ficheiro.

Assim, os programadores apenas têm de enviar o caminho do *endpoint* (*url* de comunicação) com o qual querem comunicar, o *body* em JSON, caso haja necessidade de enviar informação, e o *token*, caso o pedido solicitado necessite de autenticação prévia. Após o envio desse pedido, o ficheiro genérico faz todo o processamento e tratamento da resposta para o ficheiro em que foi chamado;

- **Redux:** O *redux* é uma arquitetura de controlo de estados global de uma aplicação em JavaScript, estando normalmente associado à biblioteca de React, mas que pode ser utilizado como arquitetura de qualquer outra *framework* de JavaScript, como Angular ou VueJS [78]. O *redux* é baseado em *triggers*, como demonstra a Figura 16. A vista (*View*) desperta uma ação de *trigger*, através de um clique, mudança de estado ou *callback*. Essa ação faz o tratamento do evento recebido (*Action*) e envia-o através do *dispatcher* para a *store* onde o *redux* tem a informação toda do estado global da aplicação. Essa mesma ação no *dispatcher* é sempre desencadeada com um identificador (normalmente constante em *string*). Esta constante é que identifica qual o *reducer* que irá afetar a alteração do estado, que é uma forma de designar o modelo de dados convencial, mas em objetos JavaScript da arquitetura *redux*. Estas ações nos *reducers* conduzem, normalmente, a uma alteração do estado geral da aplicação. A alteração é, por isso, passada para todas as vistas, fazendo com que os dados sejam alterados em todos os componentes. Após a integração e instalação do *redux* no *boilerplate*, foram integrados todos os componentes genéricos de navegação também com a arquitetura *redux*, de forma a ser mais fácil aceder a todos os estados da navegação em qualquer ponto da aplicação com o *redux*;

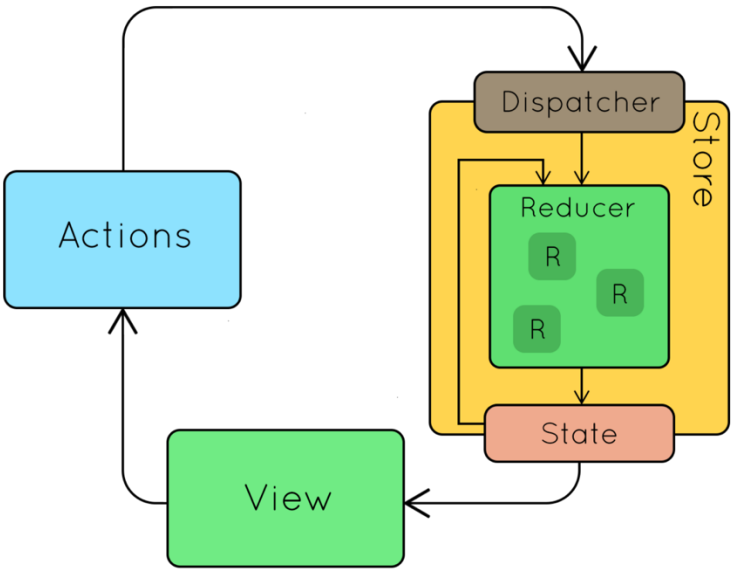


Figura 16 – Arquitetura *redux* [79]

- **Actions/Reducers:** A criação de *actions* e *reducers* já é, usualmente, feita de forma muito específica consoante as diferentes lógicas de negócio da aplicação. Contudo, foram criados no *boilerplate actions* e *reducers*, associados a um utilizador, que é uma funcionalidade partilhada por grande parte das aplicações. *Actions*, como, por exemplo, alterar linguagem do utilizador, e um *reducer* com o modelo de dados usado por um utilizador, com o identificador da linguagem atual a utilizar na aplicação, e outros dados relevantes que podem surgir num modelo de dados de utilizador. Assim, qualquer programador, que implemente uma futura aplicação móvel com o *boilerplate* criado, pode entender como é feito o processo, desde a vista até à alteração do estado no próprio *reducer*;
- **Persistência de dados:** A arquitetura de persistência de dados foi criada com o auxílio da biblioteca *redux-persist* [80], que facilita a lógica de armazenamento de dados em aplicações móveis criadas através de React Native. Porém, o objetivo principal passava por criar uma dependência entre os *reducers* e o *redux-persist*, de forma a que, após a criação do *boilerplate*, qualquer que fosse a aplicação criada posteriormente, tivesse a persistência de dados incluída. Assim, nas novas aplicações não será necessário recorrer a qualquer tipo de integração adicional, tornando-se em mais um processo otimizado para qualquer programador. Após a idealização de diferentes propostas, a solução passou por persistir toda a *store* do *redux*, ou seja, tudo o que na Figura 16 está incluído, na *store* é persistido. Para controlar de forma simples quais os *reducers* do *redux* que o programador desejava persistir, como, por exemplo, dados de utilizador, e os que não desejava persistir entre processos da aplicação, como, por exemplo, o estado da navegação, foi criada uma camada intermédia chamada *appReducer*. Esta camada controla todos os *reducers*, fazendo com que apenas uma simples propriedade sirva para definir se o *reducer* de dados é para persistir com o *redux-persist* ou se é para descartar no final da utilização do processo. Esta abordagem de persistência de dados levou a que qualquer aplicação agora criada pela empresa, que derive do *boilerplate* inicial de código, já tenha todo o processo de persistência de dados incluído;
- **Definição de estruturas de pastas:** Através da análise dos vários documentos e dos exemplos iniciais das aplicações criadas, foi-se idealizando a melhor estruturação que se poderia dar às pastas com os ficheiros de código. Este processo foi sendo melhorado até ser obtida uma solução que fazia sentido para a empresa e que estivesse de acordo com a documentação de React Native, que é demonstrada na Figura 17. As pastas de imagens, Android e iOS, foram mantidas na raiz do projeto principal por serem totalmente isoladas da lógica de desenvolvimento do JavaScript e do React Native. Na raiz do projeto, também foi mantido o ficheiro de ponto de entrada da aplicação no JavaScript (“index,js”) e a pasta *src*

que contém toda a lógica desenvolvida em React Native. O desafio principal passou pela organização da pasta *src*, que contém toda a lógica desenvolvida de JavaScript. Foram mantidos dois ficheiros na raiz de *src*, "index.js" e "configureStore.js". O primeiro para servir como ponto de entrada na pasta *src*, e o segundo que contém a solução genérica criada a partir da persistência de dados do *redux* descrita anteriormente. Após todos os processos incrementais de alteração da estrutura de pastas, foram criadas as seguintes pastas, para simplificar a organização de todo o código:

- **Components:** pasta com todos os pequenos componentes que podem ser utilizados ao longo da aplicação, sendo todos eles genéricos e reutilizáveis. Nesta pasta, também se encontram os estilos e as cores da aplicação. Isto permite que, no futuro, um profissional na área do *design*, que não possua conhecimentos de programação, mas que perceba de criação de componentes em React, consiga facilmente, através desta pasta, criar componentes genéricos sem lógica de negócio associada;
- **Containers:** pasta onde se encontram todas as vistas da aplicação, contendo a integração dos vários componentes para a vista e o controlo do estado de todos eles, através das propriedades e da ligação com a *store* do *redux*;
- **I18n:** pasta com o módulo que permite dar suporte a múltiplas línguas à aplicação. Este módulo basta ser importado, caso a mesma necessite desta funcionalidade, e removido, caso não seja necessário, pois foi criado para ser um módulo independente a qualquer projeto;
- **Modules:** pasta onde está toda a arquitetura *redux*, em que cada módulo é composto por uma pasta de *actions* e uma de *reducers*, pois estão inevitavelmente ligadas na arquitetura;
- **Router:** pasta onde está toda a lógica de navegação da aplicação;
- **Utils:** pasta que serve para ter todos os ficheiros que podem ser utilizados em todos os projetos, como, por exemplo, funções genéricas, utilização de permissões ou configurações. O objetivo das funções desta pasta é o de serem sempre utilizadas em todos os projetos sem terem de ser alteradas à medida do projeto por serem genéricas o suficiente para tal, tendo um desenvolvimento incremental genérico, nunca objetivando um projeto único.

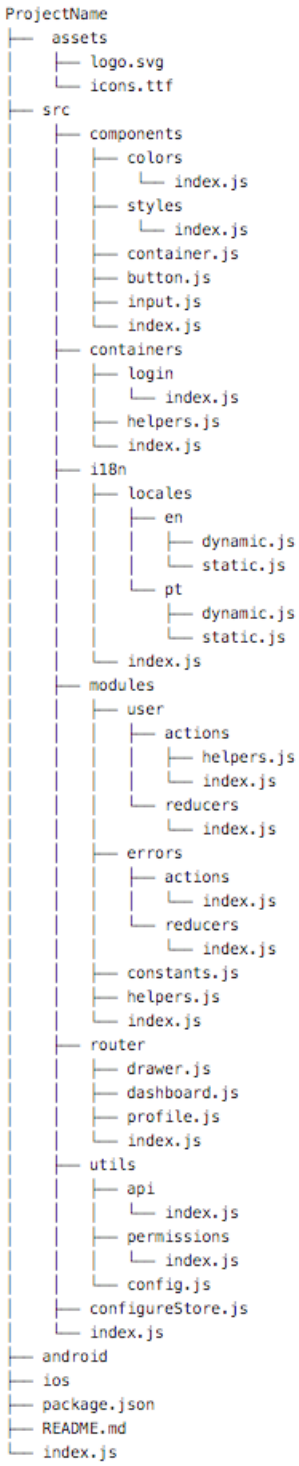


Figura 17 – Definição final da estruturação de pastas criada para a empresa

- **Definição de packages:** O React é apenas uma biblioteca focada no desenvolvimento da vista com o JSX, ou seja, para que sejam utilizadas todas as funcionalidades associadas ao

React, é necessário utilizar mais dependências, sendo muitas delas criadas exclusivamente para a utilização com o React. Desta forma, foram definidos dois grupos de dependências: obrigatórias, que devem ser utilizadas em todos os projetos da empresa, e opcionais, para quando existe a necessidade de utilização de alguma funcionalidade mais específica. Esta definição das dependências faz com que todas as aplicações utilizem as mesmas versões de dependências e, acima de tudo, para que sejam utilizadas sempre as mesmas dependências para um determinado objetivo. As dependências definidas como obrigatórias e opcionais na iniciação de qualquer projeto foram selecionadas pela versão que fosse compatível com a versão selecionada para o React Native, neste caso a 0.51. As dependências definidas como obrigatórias foram as seguintes:

- Redux – 3.7.2;
- Redux Thunk – 2.2.0;
- React Redux – 5.0.6;
- Redux Logger – 3.0.6;
- React Navigation – 1.0.0;
- React Native Vector Icons – 4.4.2;
- Redux Persist – 5.4.0;
- Redux Form – 7.2.0;
- React Native i18n – 2.0.9;
- Moment – 2.19.3;
- Device Info – 0.13.0.

As dependências definidas como opcionais foram as seguintes:

- React Native Debugger - 0.7.13;
- React Native Fetch blob - 0:10.0;
- React Native FS - 2.9.12;
- React Native Contacts - 2.1.13;
- React Native Image Picker – 0.23.0;
- React Native Fast Image: 4.0.14;
- React Native Android Dialogs: 1.0.2;
- React Native Snap Carousel: 3.6.0;
- React Native Maps: 0.20.0.

- **Definição de ficheiros de configuração:** Foram definidos dois ficheiros principais de configuração, o de "constants.js" e o "api.js". O primeiro permite armazenar constantes associadas ao *redux*, de forma a que cada ação diferente faça uma alteração num e só num

reducer diferente consoante a constante passada através do *dispatcher*. O outro ficheiro define configurações associadas à execução da aplicação, como detalhes do *url* da REST API de desenvolvimento ou produção, tipo de execução da aplicação, localizações de mapa por omissão, configurações, como credenciais de acesso a serviços de mapas, entre outros;

- **Integração de Fabric:** De maneira a obter sempre dados sobre possíveis erros das aplicações criadas, era necessário integrar um serviço que fornecesse todas essas estatísticas e que, em caso de interrupção da aplicação em produção, enviasse *email* com o erro que ocorreu. A ferramenta escolhida foi o *crashlytics* do Fabric [81] e foi documentado todo o processo de como integrar o *crashlytics* tanto para iOS como para Android. No caso do *boilerplate*, foi integrado o *crashlytics* para Android e para iOS, com comentários bem explícitos das linhas de código referentes à integração do *crashlytics*, tanto no *build.gradle* do Android como nos *podFile* do iOS. Desta forma, apenas é necessário criar no *site* do Fabric uma nova aplicação para Android e outra para iOS e copiar as *keys* geradas pelo Fabric. Estando todo o resto do processo integrado, com a alteração das *keys*, o *crashlytics* fica automaticamente ativado para essas duas aplicações.

6.2.6 Cobertura de código (*Linting*)

O *Linting* é uma das técnicas de *Static Analysis Testing* [82] que consiste numa parte do processo de desenvolvimento, em que tem a função de verificar em *background* das normas estipuladas e, por vezes, da qualidade de estruturação do código que está a ser desenvolvido sem envolver execuções de código. Alguns IDEs já vêm com este tipo de ferramentas, como, por exemplo, o ponto e vírgula de C ou Java, em que o IDE automaticamente faz *Static Analysis Testing* e, mesmo antes do utilizador compilar o código, alerta se o que foi codificado está correto [83].

No caso do código JavaScript, que normalmente é escrito em editores de texto, como o Visual Studio Code [84] ou o Atom [85], sendo uma linguagem dinâmica e que não requer definições de tipo é especialmente propensa a erros por parte do programador, tornando-se mais importante a utilização de *Static Analysis Testing*, mais propriamente o *Linting*. O *Linting* ajuda os programadores a entender a estrutura do código e também pode ser usado para impor padrões de codificação.

O ESLint é uma ferramenta *open-source* de *linting* para JavaScript, criada em 2013 por Nicholas C. Zakas [86], usada para localizar padrões problemáticos ou código que não segue determinadas diretrizes mesmo antes de ser compilado. Embora o ESLint disponibilize um conjunto de regras, podem ser criadas as regras que a equipa ou o programador achem úteis ou até mesmo utilizar *plugins* com regras definidas por grandes empresas ou comunidades.

O objetivo inicial para a empresa passou por criar regras que ajudassem os programadores, que não tinham tido contacto ainda com o padrão JavaScript 2015 (ES6), a fazer o código segundo essas mesmas diretrizes. A implementação inicial do ESLint na empresa foi de tal forma vista como importante que foi criado um ficheiro de ESLint alargado e mais complexo, baseado nas regras-base do Airbnb [87] e nas regras da própria empresa. Assim, para além de *linting* de JavaScript, foram incluídas muitas regras de "*coding standard*", levando a que no próprio ESLint fosse configurado um padrão de desenvolvimento de código da própria empresa.

Sem grande esforço adicional, foram criadas regras transversais a todos os projetos da empresa em JavaScript, desde o *back-end*, *front-end web* até às aplicações móveis em React Native, que foi a primeira *framework* com a introdução de ESLint.

6.2.7 Definição do ambiente de desenvolvimento

Para desenvolver aplicações simultaneamente para Android e para iOS, é aconselhada a utilização do sistema operativo macOS, de forma a poder facilmente ter acesso a todas as ferramentas de desenvolvimento para Android e para iOS. Para além disso, tal como referido na documentação do *boilerplate* (Anexo H), é obrigatória a instalação do Android Studio com o JDK (Java Development kit) e do Xcode para criação da execução de cada uma das aplicações.

Para além das ferramentas associadas ao desenvolvimento móvel, foi definida a utilização do Visual Studio Code, para o desenvolvimento do código com as seguintes extensões obrigatórias:

- **Prettier - Code formatter:** Permite que, com um ficheiro de configuração com regras de indentação de código já criado (.prettierrc.json) no *boilerplate*, aquando da ação de guardar um ficheiro, o ficheiro de código seja automaticamente formatado consoante as regras de indentação definidas no ficheiro-base;
- **Eslint:** Extensão que permite que o *linting* seja feito em *run time* e os problemas/avisos apareçam automaticamente listados no editor "problemas de código" consoante as regras definidas no ficheiro (.eslintrc.json);
- **Trailing Spaces:** Extensão que proíbe a existência de espaços em branco desnecessários em todo o código;
- **React Native Tools:** Extensão que recomenda o código de React Native ao programador, tal como se de um IDE se tratasse. É aconselhável a sua utilização, principalmente nos primeiros contactos com a *framework*.

6.2.8 Documentação do boilerplate

A documentação toda, associada ao *boilerplate* (Anexo H), foi escrita em Markdown [88], que é uma linguagem de marcação criada por John Gruber e Aaron Swartz, e que converte texto descritivo em HTML válido. A documentação criada, para além de explicar o *boilerplate*, também teve em consideração o objetivo de integração de novos membros no desenvolvimento de aplicações móveis em React Native. Para isso, foi criada documentação diversa, como o editor de texto que o programador deve utilizar, e tutoriais que aquele deve ver e realizar antes de avançar para os capítulos seguintes do documento. Desta forma, o documento não foi criado para ser lido sequencialmente como um livro, mas sim como um processo que integra vários passos necessários para um programador aprender os conceitos básicos a fim de poder desenvolver aplicações em React Native.

6.3 Considerações finais

De acordo com o objetivo principal do estágio e com o referido no capítulo de finalização dos projetos nativos, havia a necessidade de mudar todo o desenvolvimento, inclusive o de aplicações móveis para a mesma linguagem, neste caso, JavaScript. A mudança acabou por surgir também numa altura boa para a empresa, porque houve uma boa visão por parte da organização. Além disso, tendo em conta os projetos que tinham sido realizados e os que estavam para aparecer, a empresa fez uma pequena quebra no desenvolvimento móvel, de forma a que a mudança fosse concluída com sucesso, sem que se perdessem projetos que estavam a surgir.

Esta mudança, inicialmente prevista para o desenvolvimento móvel, acabou por ir mais além, fazendo com que, progressivamente, toda a empresa adotasse os novos padrões de desenvolvimento de JavaScript (ES6) e a análise estática de código através do ESlint. Atualmente, toda a organização usa o padrão ES6, inserido simultaneamente com o React Native, e o ESLint, que foi criado inicialmente para ajudar na adaptação do novo padrão JavaScript, mas que no final serviu para gerar uma pequena convenção interna de organização e estruturação do código global a usar na organização.

Outra metodologia que afetou a organização de forma positiva foi a criação do *boilerplate*. Com o *boilerplate*, é facilitado todo o processo de criação de um novo projeto de desenvolvimento móvel com todas as arquiteturas definidas, persistência de dados, navegação e componentes genéricos. Com as vantagens que se obtiveram com o *boilerplate* criado, nomeadamente no que se refere a tempos de inicialização de projetos e normalização das bibliotecas, a empresa conseguiu avançar com a criação de novos *boilerplates* para outros tipos de *frameworks*. Por exemplo, foi criado também internamente um *boilerplate* para NodeJS com Express para Backend e outro para AngularJS para *front-end web*.

7 Desenvolvimento de arquitetura de audioguias em JavaScript

Atualmente, as aplicações móveis são utilizadas para os mais variados fins, lúdicos e profissionais. Um audioguia tem como principal objetivo auxiliar um turista de uma região ou visitante de museus/exposições a fim de que este tenha uma visita guiada de forma autónoma. Estes serviços podem ser totalmente gratuitos ou, em alguns casos, com custos mais baixos em relação a um guia turístico tradicional.

A empresa Crossing Answers tem uma vertente focada em soluções multimédia com fins turísticos, ou seja, regularmente surgem projetos relacionados com a melhoria turística de uma determinada região. Normalmente, um dos principais produtos requisitados pelos clientes são audioguias. Isto leva a que sejam sempre implementadas funcionalidades similares em projetos diferentes, tais como: mapas, perfil de pontos de interesse, galeria, áudio, favoritos, entre outras funcionalidades que surgem repetitivamente.

Com o processo de mudança de todo o desenvolvimento móvel para JavaScript, neste caso para o React Native, houve a necessidade de implementar de raiz todos os conceitos dos guias interativos. Com esta nova forma de construção das aplicações, o objetivo era o de criar uma arquitetura que permitisse, independentemente do projeto do guia, reutilizar grande parte do código. Como cada um dos clientes pretende, frequentemente, algumas funcionalidades à medida, era inviável criar um produto que servisse para todos e que fosse apenas configurável em cores e conteúdo. A solução para a implementação foi a criação de módulos que fossem totalmente independentes e que permitissem em cada novo projeto serem inseridos ou retirados consoante as pretensões do cliente para o guia. Esta nova forma de desenvolvimento originou que tivessem de ser realizadas algumas mudanças nos modelos de dados existentes, de modo a tornar a aplicação mais genérica.

O facto do desenvolvimento em React Native ser orientado a componentes, significa que os módulos devem ser criados como componentes isolados e atómicos, para que possam receber propriedades quando chamados, adaptando-se assim à vista principal em que vão ser inseridos.

7.1 Definição de componentes de módulos genéricos

Para criar um componente genérico e reutilizável, é essencial entender quais são as propriedades necessárias a enviar para o componente, para este se comportar de forma autónoma em qualquer projeto e, posteriormente, perceber o tipo de herança relativamente a outros componentes mais genéricos [91-93]. Após uma análise profunda de implementações genéricas no React Native, foram criados os componentes genéricos explicados em cada um dos próximos subcapítulos. O esquema geral dos componentes que foram implementados modularmente com as suas propriedades obrigatórias e opcionais é mostrado na Figura 18.

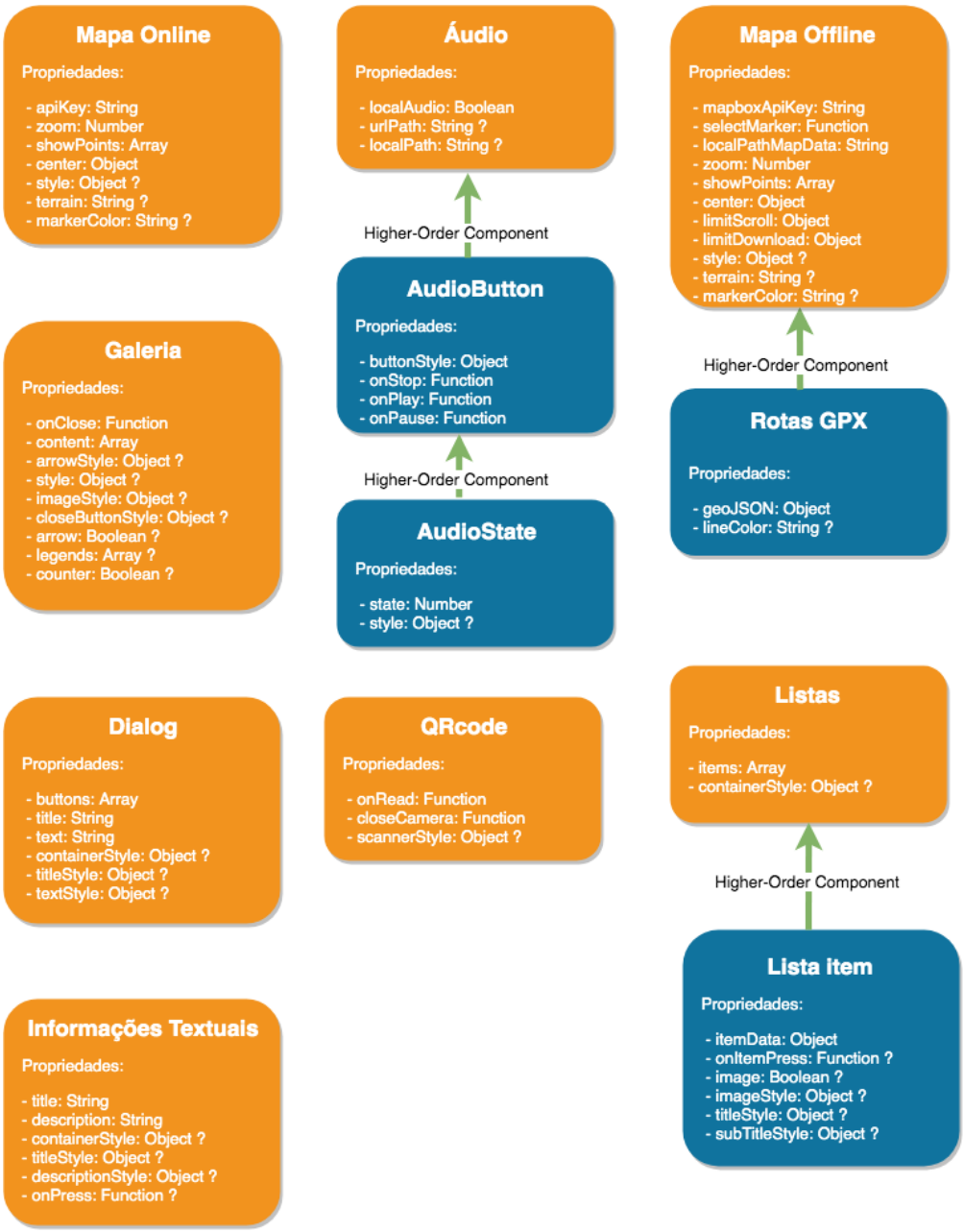


Figura 18 – Esquema geral da arquitetura de componetes criados

7.1.1 Mapa *online*

A visualização de mapas online é uma das funcionalidades mais importantes para guias interativos para espaços exteriores, possibilitando que o utilizador obtenha todas as orientações da sua localização e dos seus pontos de interesse. A primeira abordagem de mapa foi feita com a utilização do Google Maps. De acordo com o definido para os projetos a realizar, este componente deverá receber a lista de pontos que serão mostrados através de marcadores a incluir no mapa, bem como uma coordenada de latitude e longitude correspondente ao centro da sua primeira execução, o nível zoom do foco nessa localização, representado através de um valor entre 0 e 20, e a Google API Key [92], associada a esse mesmo projeto. Como parâmetros opcionais, é possível definir, através de um estilo que é aplicado no container de todo o componente de mapa, as cores dos marcadores no mapa, o tipo de terreno a mostrar (satélite ou mapa), e o seu tamanho na vista. Neste momento, para utilizar o componente mapa, basta importar esse mesmo componente e ler a documentação associada ao mesmo sobre quais as propriedades obrigatórias e opcionais.

7.1.2 Mapa *offline*

Logo após a implementação do mapa *online*, surgiu a necessidade de implementar mapas *offline*, em que fosse possível ao utilizador navegar no mapa mesmo em zonas onde não tivesse acesso a uma rede de dados (informação *online*). Devido às maiores restrições por parte da Google, para a utilização do *google maps*, foi utilizado para este componente o *mapbox* [93], que tem soluções gratuitas de grande escala de utilização e que permite usar de forma totalmente *offline* o serviço. Indo ao encontro da decisão tomada em utilizar o React Native como a *framework* para desenvolvimento móvel, salienta-se que o *mapbox* fornece documentação apenas para Android, iOS, Unity [94] e React Native [95]. Isto demonstra que alguns serviços de plataformas móveis já perceberam que o React Native é uma *framework* em ascensão. Havendo esta mesma documentação para React Native, foi muito mais fácil a implementação de todo o serviço de mapas *offline* e de toda a sincronização e armazenamento de dados associados ao mapa. Isto fez com que, para além das propriedades necessárias para o componente mapa *online*, fossem adicionadas como propriedades obrigatórias os pontos-limite sudoeste e nordeste de *download* do mapa e os pontos-limite em que o utilizador pode navegar, tanto para nordeste como para sudoeste. Para além disso, foi necessário acrescentar o caminho de armazenamento dos dados do mapa, para armazenamento aquando do *download* e para acesso posterior, e a API KEY privada do *mapbox*. Após essas mesmas propriedades terem sido incluídas, foi implementada uma lógica totalmente independente no componente principal (mapbox.js), bem como em componentes associados a esse mesmo componente genérico de controlo de *download*. O componente de mapa offline implementado é apresentado na Figura 19.

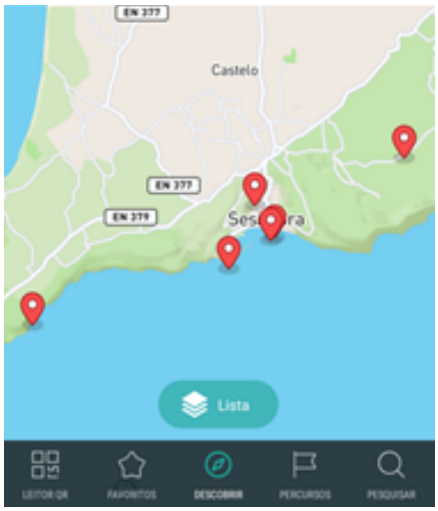


Figura 19 – Componente Mapa offline com recurso a Mapbox

7.1.3 Rotas GPX

Outro componente utilizado, relacionado com mapas, foram as rotas através de ficheiro no formato GPX [96], recebidas pelo servidor. Apesar da aparente dificuldade que este componente poderia ter para a sua implementação, com o componente de mapa *offline* já criado, esta tarefa ficou bastante facilitada, sendo apenas necessário criar um subcomponente filho do mapa *offline* principal, que colocava uma *layer* de GPX no componente principal de mapa *offline*, recebendo apenas dados no formato *geojson* e funcionando como um HOC (Higher-Order Component), que é uma função de alto nível que recebe um componente e retorna o mesmo componente com alguma alteração, neste caso com o desenho da *layer* GPX no mapa.

7.1.4 Direções em *offline*

A criação de um pequeno GPS integrado para navegação *offline*, para pontos de interesse selecionado, foi um dos componentes que ainda não foi implementado, pela complexidade inerente ao mesmo e por não ter existido ainda essa necessidade de integração num guia interativo. Apesar disso, foi feita uma *Spike*[1], para perceber as implicações que a implementação de navegação *offline* teria. Foi concluído que a implementação envolvia a integração de bibliotecas C++ diretamente com Java/Swift, que depois fariam comunicação direta com o React Native. A documentação detalhada da *Spike* pode ser vista no Anexo I.

[1] Forma de fazer uma pesquisa e documentar, se necessário, em engenharia de *software*, que analisa o possível trabalho inerente a uma implementação ou funcionalidade

7.1.5 Áudio

Outro dos módulos quase sempre utilizado num guia interativo é o módulo de áudio, que pode variar entre *streaming* ou por carregamento local após um *download* antecipado do ficheiro na sincronização dos dados.

Todo o processo de reprodução de áudio foi feito com recurso à dependência *react-native-sound* [97], a qual já permite a reprodução de áudio em React Native, tanto para Android como para iOS, através de vários formatos diferentes. A complexidade maior na criação deste componente foi o controlo de *streaming* e de carregamento de ficheiros locais, de forma a tornar o componente totalmente independente, apenas sendo enviada uma propriedade com a forma como o ficheiro deve ser executado. No caso de um ficheiro em *streaming*, é enviada a propriedade *localAudio* a *false* e, em conjunto, um *endpoint* de onde está disponível o ficheiro para reprodução. No caso de a propriedade *localAudio* ser *true*, então é também necessário enviar o caminho local completo a partir de onde o ficheiro pode ser carregado. Este componente de áudio levou à criação de outros componentes dependentes deste mesmo áudio, como o AudioButton, que permite reproduzir o áudio ou colocá-lo em pausa, e o componente AudioState, que mostra o estado do progresso da reprodução do ficheiro de áudio.

7.1.6 Galeria

A galeria em *swipe* é um componente que permite criar uma galeria de imagens. Assim, os utilizadores que pretendam ver um conjunto de imagens diferentes em *fullscreen* no telemóvel só precisam de deslizar para a frente e para trás. É um componente cujo código livre facilmente se pode encontrar em qualquer repositório público. Contudo, optou-se por criar um módulo genérico de galeria em *swipe* próprio, disponibilizando opções adequadas aos tipos de projetos mais comuns realizados na empresa. Este é um dos módulos que tem mais possibilidades de configuração ao nível dos estilos, como, por exemplo: os estilos de setas de *swiper*, o formato da imagem, os pontos para obter informação sobre o estado da galeria ou a representação desse mesmo estado através de numeração. O componente de galeria é apresentado na Figura 20.

Figura 20 – Componente galeria

7.1.7 Listas

Foram criadas listas de pontos de interesse. Deste modo, a partir de um conjunto de pontos, o componente mostra automaticamente toda a lista. Este componente é totalmente configurável no que concerne aos estilos, desde os estilos do *container* da lista até aos estilos dos próprios itens de cada uma das listas. O exemplo de uso do componente de lista é apresentado na Figura 21.

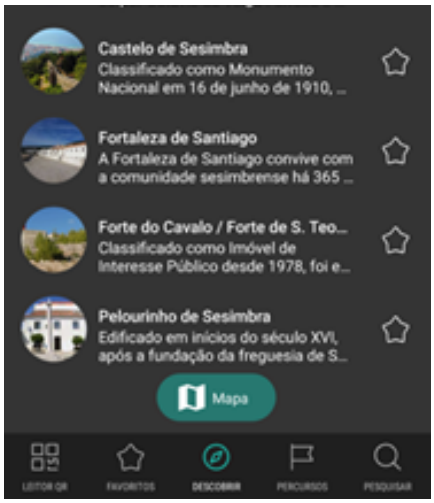


Figura 21 – Componente de lista de itens

7.1.8 Informações texturais

Numa aplicação, a informação muitas vezes é mostrada sobre o formato de título com um texto associado, como, por exemplo: contacto (título) e um texto sobre o respetivo contacto. Para isso, foi criado

um componente para informações textuais que, dando um estilo para o título e outro para o valor, mostra os dados de informações texturais. A aplicação do componente textual é apresentado na Figura 22.

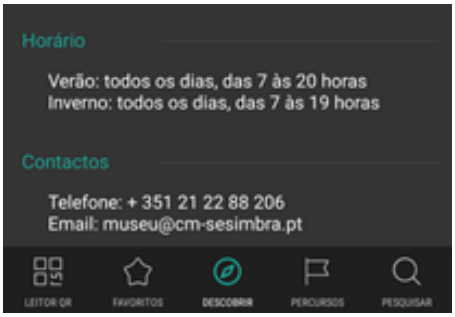


Figura 22 – Componente de informação textual

7.1.9 QRcode

O módulo de QRCode, quando incluído em qualquer projeto, tem toda a lógica de solicitação de permissões associadas à câmara, não sendo necessário adicionar um código para esse efeito no projeto. Quando este módulo é instanciado num projeto, existe a necessidade de registar na vista o evento que é gerado assim que é feita a descodificação de um QRCode. Quando o evento for desencadeado, o código associado ao processamento do evento será executado. Nesse código, é verificado se o identificador obtido do QRCode corresponde a um elemento conhecido. Caso encontre o que pesquisa, fecha a câmara e o componente, e redireciona a aplicação para o sítio pretendido. Caso não encontre nenhum identificador daquele tipo, volta a registar o evento, com a câmara a trabalhar sem qualquer interrupção. A utilização do componente com a funcionalidade de QRcode é apresentado na Figura 23.

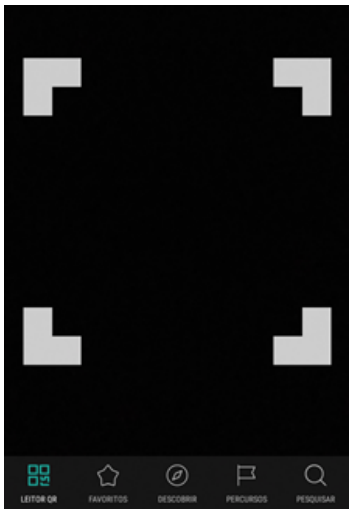


Figura 23 – Componente de QRcode a ser executado

7.1.10 Dialogs

As Dialogs, utilizadas normalmente para dar informação sobreposta à vista a um utilizador, são um módulo que normalmente está associado a todo o tipo de projetos. Assim, surgiu a necessidade de criar um módulo de Dialogs que possibilitasse *n* eventos de clique, tantos quantas as opções de escolha que o utilizador possa realizar sobre a Dialog. Desta forma, é possível instanciar Dialogs em qualquer sítio de um projeto do tipo informativo, com apenas uma opção de escolha, ou com duas ou mais opções para o utilizador selecionar.

7.2 Audioguia Sesimbra

O audioguia de Sesimbra foi o primeiro projeto para um cliente feito na nova metodologia de desenvolvimento de aplicações móveis implementada. Como também foi o primeiro projeto audioguia, os componentes criados e referidos no subcapítulo anterior foram feitos à medida da necessidade do desenvolvimento do audioguia, mas fazendo sempre com que os mesmos se tornassem genéricos para outros projetos. A implementação toda demorou cerca de 6 semanas para ter as duas plataformas prontas ao mesmo tempo com as mesmas funcionalidades. Sendo esta uma implementação que foi criada totalmente de raiz, nas 6 semanas, estavam incluídos todos os tempos de desenvolvimento dos componentes genéricos. Apesar disso, o tempo de desenvolvimento da aplicação acabou por ser muito pequeno, comparativamente com o que seria necessário caso a solução fosse realizada de forma nativa em Java para Android e Swift para iOS.

O desenvolvimento da aplicação foi feito em 4 *sprints*, dos quais os dois primeiros de uma semana, e os outros 2 de duas semanas. Apesar de não existir uma regra estrita da empresa na duração das iterações de desenvolvimento do SCRUM [98], normalmente são feitas iterações de 1 semana. No caso do desenvolvimento móvel, apesar de ter sido essa a abordagem inicial, chegou-se à conclusão de que com o React Native (multiplataforma) o desenvolvimento de Sprints de duas semanas era mais viável. Esta decisão deveu-se ao facto de, segundo a metodologia SCRUM, no final de cada *sprint*, ser entregue para validação ao cliente ou *Product Owner* um incremento do produto realizado. No caso do desenvolvimento móvel, envolve a geração de um *apk* de produção para teste no Android e a geração de uma *build* de iOS para colocar no *testflight*. Este processo fazia com que em *sprints* de uma semana esse procedimento ocupasse demasiado tempo em relação às funcionalidades já desenvolvidas. Com esta conclusão e apesar da empresa adotar *sprints* de uma semana, no desenvolvimento móvel passou a adotar-se *sprints* de duas semanas a partir da adoção do React Native. O resultado final da aplicação em *screenshoots* pode ver visto no Anexo J.

7.3 Considerações finais

Após a criação dos componentes genéricos e da implementação do primeiro projeto para um cliente em React Native, através desses mesmos componentes genéricos, são percetíveis os benefícios que o Facebook procura com a programação orientada a componentes no *frontend*. Com esta técnica, toda a lógica de um componente desenvolvido de forma genérica pode ser reutilizado, sem fazer componentes particularizados num projeto específico. Estes componentes em módulos genéricos, que foram criados para o audioguia, fazem com que nos próximos projetos do tipo audioguia seja mais fácil fazer as funcionalidades dos módulos já implementados. Isto porque, com a modularidade dos componentes que foi criada, apenas é necessário o programador injetar as propriedades que necessita no módulo e esse mesmo módulo fica totalmente funcional, como se de um módulo externo do *npm*[2] se tratasse.

O objetivo principal do desenvolvimento foi cumprido com sucesso, ao criar estes módulos que permitem qualquer outro programador usar os mesmos para todos os projetos de audioguia. O audioguia de Sesimbra foi finalizado, com todas as funcionalidades pedidas pelo cliente. Com os outros dois audioguias (Penela e Alter do Chão), como não traziam nada de novo nem apresentavam grande dificuldade e, além disso, como era apenas mais um desenvolvimento de duas novas aplicações foi decidido que era mais importante fazer um estudo comparativo do projeto que foi desenvolvido em híbrido, em relação aos audioguias nativos desenvolvidos no capítulo de finalização do desenvolvimento nativo, e posteriormente passar para um outro projeto mais complexo que envolvesse outro tipo de exigência para a *framework*.

O estudo comparativo realizado foi baseado no tempo e problemas de desenvolvimento para o mesmo tipo de projeto com as mesmas funcionalidades, neste caso, com audioguias que tanto foram implementadas em React Native como nativamente.

Como resultado do trabalho, verificou.se que o Swift é a solução que apresenta o tempo de desenvolvimento mais baixo, enquanto o Java e o React Native apresentaram tempos de desenvolvimento muito idênticos, com a particularidade do React Native dar suporte aos dois principais sistemas operativos móveis.

Este trabalho comparativo realizado encontra-se detalhado no Anexo K e foi uma terceira publicação que foi aceite para conferência ao longo do estágio, neste caso na conferência ibérica de Sistemas e Tecnologias de Informação em 2019.

[2] Gestor de pacotes para o ambiente de execução JavaScript

8 Desenvolvimento da aplicação Connects

Para melhor validar a mudança do desenvolvimento nativo para o desenvolvimento em React Native, foi necessário testar a *framework* num contexto de um projeto de maior dimensão, de modo a perceber as reais vantagens e desvantagens e o sucesso da mudança de paradigma do desenvolvimento móvel da empresa. Neste caso, o maior projeto incidiu no desenvolvimento de uma rede social de contactos profissionais, denominada de Connects. Em 8.1, é explicado o âmbito do projeto Connects. Em 8.2, são apresentadas as funcionalidades principais a incluir no projeto, as quais são detalhadas e divididas em iterações de implementação, em 8.3. Por fim, em 8.4, são apresentadas as conclusões do desenvolvimento de um projeto maior em React Native, sendo referidas as vantagens e desvantagens da realização de um projeto de maior âmbito em React Native.

8.1 Âmbito do projeto

De uma forma geral, a aplicação Connects permite a integração de contactos existentes nas redes sociais com a criação de um perfil pessoal. Esses perfis pessoais contêm vários cartões pessoais e profissionais que podem ter perfis públicos ou privados. O connects deverá ainda permitir adicionar outros perfis à lista de contactos pessoais através dos seus cartões. Quando se acede aos detalhes de um contacto, é possível obter informações atualizadas dos cartões partilhados, ver a informação relativa ao primeiro contacto estabelecido com o outro perfil, registar informações, anexar fotografias/imagens e até mesmo marcar reuniões. Quando o contacto é mútuo, é possível iniciar uma conversação por *chat*, realizar uma chamada para essa pessoa e também visualizar o histórico profissional da pessoa em questão. Para isso, teve de ser criada uma REST API para guardar e obter todas as informações relativas à aplicação.

8.2 Funcionalidades principais

Antes de se iniciar o desenvolvimento já existia um levantamento de todos os requisitos principais da aplicação, de tal forma que existia um bom refinamento de todas as funcionalidades da aplicação. Assim, as principais funcionalidades da aplicação a criar foram as seguintes:

- Registo na aplicação através por *e-mail*, Facebook ou Google;

- Edição de informações pessoais do perfil;
- Inserção de *skills* no perfil;
- Adição/Edição do cartão com detalhes pessoais ou profissionais;
- Configuração de permissões dos cartões (públicos, privados ou invisíveis);
- Adição de outros contactos por pesquisa pública ou leitura de QRCode;
- Inclusão de galeria de fotografias pessoal;
- Adição de gostos ou notas a um determinado perfil;
- Marcação de reuniões com outro perfil;
- Notificações internas da aplicação de novos pedidos de contacto, resultado de pedidos ou respostas de marcações de reuniões;
- Adição de contactos aos favoritos;
- Configuração do primeiro encontro entre dois contactos, através de data, localização ou eventos;
- Inclusão de calendário para marcar/ver eventos ou reuniões;
- Criação de sistemas de grupos de contactos, com criação de grupos e gestão dos membros dos grupos;
- Criação de *chat* para conversa entre perfis e em grupos;
- Realização de chamadas entre perfis.

Todo o resultado final da implementação da aplicação estão no Anexo L, onde todos os requisitos enumerados foram implementados com sucesso.

8.3 Considerações finais

Apesar de o desenvolvimento deste projeto não ser o principal foco do estágio, todo o projeto foi finalizado com sucesso, sendo que o resultado final da aplicação pode ser consultado no Anexo L. Para além do desenvolvimento, também importante, o principal foco era o de perceber qual o impacto do uso de React Native num projeto de maior dimensão/complexidade, e quais as vantagens e desvantagens da adoção do React Native nestas circunstâncias. Foram encontradas tanto vantagens como desvantagens associadas ao desenvolvimento do projeto. No que concerne às vantagens, algumas delas já eram previsíveis e outras foram descobertas ao longo do desenvolvimento. Em relação às desvantagens encontradas, estas, de um modo ou outro, foram ou serão ultrapassadas.

Após a conclusão deste projeto, as principais vantagens encontradas com o desenvolvimento do React Native foram:

- **Programação orientada a componentes:** Tal como refere a empresa responsável pelo React Native, Facebook, a programação orientada a componentes simplifica bastante o trabalho de desenvolvimento de *front-end web* com ReactJS e, posteriormente, de *front-end mobile* com React Native [99]. O que se verificou foi que quanto mais tempo um programador se dedicar a componentes, mais facilmente consegue fazer vistas com poucas falhas. Indo mais além, e apesar da pouca experiência por parte da empresa no desenvolvimento em React Native, é facilmente percetível que certos componentes, quando criados de forma genérica, podem ser facilmente importados para qualquer tipo de projeto da empresa. Uma das vantagens ainda não explorada com a programação orientada a componentes é que os mesmos podem ser criados por *designers*, que sabem criar vistas através de linguagens de marcação, mas que não precisam perceber programação, uma vez que os componentes são apenas vistas controladas por propriedades enviadas pelo programador;
- **Vistas multiplataforma:** Quando a aplicação alcança um determinado reconhecimento, grande parte das empresas procura que as *interfaces* de iOS e Android sejam o mais idênticas possível, exceto nos seus componentes nativos como DatePickers, Switch, Inputs ou outros em que são mantidos os formatos nativos de cada sistema operativo. Com o React Native, a criação de vistas iguais foi um processo totalmente transparente e, para além disso, estas têm componentes próprios do React Native que, quando utilizados, variam o seu comportamento de acordo com o sistema operativo no qual a aplicação está a ser executada, mantendo assim a *interface* nativa;
- **Código específico:** Durante o desenvolvimento do projeto, houve, apenas e só nas chamadas em Android, a necessidade de fazer condições que executassem código diferenciado consoante o tipo de sistema operativo. O processo para essa implementação passou apenas pela criação de uma condição de *if-else*, pois o React Native tem uma função própria que identifica o sistema operativo em que está a ser executada a aplicação. Isto torna bastante fácil a utilização de código nativo em casos estritamente necessários, apesar de ser uma prática quase nunca recorrente no React Native.
- **Builds e hotReload:** Após ter existido a experiência em desenvolvimento de código nativo na finalização dos projetos existentes, uma das principais vantagens encontradas, relativamente ao React Native, foi o facto de a compilação ser feita instantaneamente, e ter

uma opção denominada por *hotReload*. Esta opção permite que uma alteração de código ou estilo por parte do programador seja repercutida para o dispositivo que está em depuração em menos de um segundo;

- **Desenvolvimento numa única linguagem de programação:** Na Crossing Answers, a maior vantagem encontrada, e provavelmente a mais esperada nesta mudança de paradigma do desenvolvimento móvel interno, foi a facilidade com que um programador de *back-end* em NodeJS (com Express) ou de *front-end web* em AngularJS consegue ajudar um programador de React Native a resolver algum tipo de problema que esteja a encontrar, ou até mesmo a efetuar algum desenvolvimento por causa de algum prazo de entrega mais apertado.

Apesar de tudo o que foi referido, o desenvolvimento em React Native também apresenta desvantagens, caso contrário seria adotado por todas as empresas. Assim no desenvolvimento deste projeto foram encontradas as seguintes desvantagens:

- **Integrações com serviços externos:** A integração principal feita neste projeto foi realizada com o serviço de Twilio, para automação do serviço de *chat* e chamadas. Porém este serviço apenas disponibiliza documentação em Java para Android e em Swift para iOS, o que fez com que todo o processo de integração tenha sido mais difícil comparativamente à implementação do mesmo serviço de forma nativa;
- **Criação de Bridge:** Após algumas tentativas de implementação do serviço de Twilio de chamadas em React Native no Android, houve necessidade de recorrer a código nativo em Java para a implementação do serviço, de acordo com a documentação disponibilizada. A criação de uma *bridge* acabou por ser um processo simples, contudo, isto conduz a um problema, pois caso uma funcionalidade seja difícil de implementar em JavaScript e caso os programadores não tenham conhecimento de Java/Kotlin ou Swift/Objective-C, a criação de uma bridge poderá ser um entrave a programadores com esse perfil. Pela necessidade referida no capítulo da arquitetura de audioguia, relativamente à utilização de bibliotecas de C++ através do Java, para posteriormente serem utilizadas através do React Native, e pela implementação realizada na *bridge*, foi feito um documento interno sobre como fazer comunicação entre JavaScript (React Native) – Java – C++ e como controlar o devido retorno de informação até ao JavaScript. Esse documento criado está no Anexo M;
- **Incompatibilidades entre bibliotecas:** Por não existir ainda uma versão estável do React Native, uma das maiores desvantagens atuais do desenvolvimento em React Native é a utilização de bibliotecas ou dependências externas que não estejam de acordo com a versão utilizada do React Native. A solução para esse problema consistiu na utilização da versão das bibliotecas associadas aos meses correspondentes à versão em que o React Native

utilizado no projeto estivesse a ser lançado. Assim, para uma dependência externa que atualmente esteja na versão 2.1.13, mas que em julho de 2018 tenha estado com a versão 2.0.4, na versão utilizada no projeto do React, bastaria instalar a dependência com essa versão e bloquear qualquer tipo de *update* na biblioteca. Com este processo, conseguiu-se manter as compatibilidades entre versões;

- **Mutabilidade/Imutabilidade dos dados:** Por vezes, ainda é difícil controlar quando é que os dados devem ser alterados ou mantidos por causa da arquitetura e do modelo de programação associado a componentes. É um problema que apenas poderá ser ultrapassado com a experiência no desenvolvimento em React e, algumas vezes, com os *logs*, de forma a perceber o comportamento da arquitetura dentro de todos os seus componentes filhos. Foi uma dificuldade demonstrada por vários programadores, tanto de ReactJS como de React Native, de tal modo que já foi anunciado que a nova versão do React terá a possibilidade de registo de eventos para controlo da mutabilidade, ou não, dos dados para além das propriedades que são agora utilizadas com esse propósito.

Atendendo ao primeiro projeto de maior dimensão, conclui-se, então, que os resultados foram positivos, não só em termos de qualidade de produto final, mas também no que diz respeito à velocidade com que foi feita a sua implementação, uma vez que esta foi realizada sem que o desempenho do produto para o cliente fosse posto em causa. De referir que algumas dificuldades que aqui foram descritas serão agora mais facilmente ultrapassadas num novo projeto, tendo em consideração todo o conhecimento adquirido sobre a *framework*. No anexo L, encontram-se *screenshots* do resultado final da aplicação, sendo que esta, apesar de já estar totalmente terminada até à data de escrita deste documento, ainda não foi publicada na PlayStore e AppleStore, por opção do cliente.

9 Conclusão e trabalho futuro

O principal objetivo que levou a Crossing Answers a lançar o estágio descrito neste documento foi concluído com sucesso. Isto porque, no início do estágio, o desenvolvimento de projetos na empresa, relacionados com aplicações móveis, era um processo que revelava sempre alguns entraves, principalmente pela dependência exagerada por programadores dessa área. Após todas as mudanças implementadas, e após o término do estágio, os projetos relacionados com aplicações móveis tornaram-se muito mais triviais e fáceis de implementar por ter ficado muito mais claro todo o processo de desenvolvimento e, acima de tudo, por todo o desenvolvimento ser baseado em JavaScript, a mesma linguagem-base utilizada em toda a organização.

No subcapítulo 9.1, são detalhadas as principais conclusões referentes ao problema inicial existente. Em 9.2, são apontadas as principais contribuições deste trabalho, tanto a nível teórico como prático. Em 9.3, são descritas algumas dificuldades e limitações que foram encontradas e a forma como foram ultrapassadas. Por fim, em 9.4, são apresentadas direções sobre o trabalho futuro a fazer nesta área, a nível teórico, bem como sugestões sobre direções futuras que a empresa deveria procurar.

9.1 Conclusões gerais

Todos os objetivos propostos no início do estágio foram realizados com sucesso e ultrapassados da seguinte forma:

- **Estudo e análise de *frameworks* de desenvolvimento móvel JavaScript:** Foi realizado um estudo comparativo com as diversas *frameworks*, tendo-se chegado à conclusão que o React Native era a *framework* que apresentava mais vantagens em relação a todas as outras. Este estudo deu origem a uma publicação numa conferência (Anexo B).
- **Recolha de dados de adaptação e aprendizagem:** Foi criado um modelo que permitiu avaliar os programadores da empresa nas diferentes *frameworks* testadas. Os resultados finais, tanto em termos de tempo de execução como em termos de respostas corretas, foi ao encontro do próprio *feedback* pessoal apresentado por cada programador para as respostas, concluindo-se novamente que o React Native era a *framework* que apresentava mais

vantagens para poder ser utilizada na Crossing Answers, dando origem a uma segunda publicação numa conferência (Anexo E).

- **Finalização do desenvolvimento nativo:** Devido a alguns projetos já terem sido iniciados, antes da realização do estágio, a empresa decidiu terminar os projetos nativos, de modo a poder entregar as aplicações o mais rapidamente possível. As aplicações foram terminadas e publicadas com sucesso, estando já a ser utilizadas pelos clientes. Assim, foi finalizado com sucesso todo o desenvolvimento móvel que ainda estava pendente na Crossing Answers.
- **Implementação de novas práticas de desenvolvimento móvel:** Apesar de toda a adaptação inicial, o processo foi decorrendo sempre de forma consciente, de modo a procurar as melhores práticas, não entrando de imediato no desenvolvimento de produtos finais sem reconhecer as boas práticas de desenvolvimento em React Native. Este processo foi realizado com sucesso, e várias práticas que foram implementadas acabaram por ser replicadas para todos os outros ambientes de desenvolvimento da empresa.
- **Desenvolvimento de arquitetura de audioguias:** Foram criados alguns componentes totalmente independentes que agora podem ser usados em qualquer projeto do tipo audioguia e outros que foram criados de forma genérica também, mas ao ritmo da necessidade de desenvolvimento da audioguia de Sesimbra. Grande parte deles são mostrados, no Anexo J, nos *screenshoots* da audioguia de Sesimbra. Foi também feito um estudo de comparação das vantagens e desvantagens de implementação em React Native, em relação ao desenvolvimento nativo em Java e Swift, com os tempos de desenvolvimento para as principais funcionalidades implementadas, dando origem a uma terceira publicação numa conferência (Anexo K).
- **Desenvolvimento de um projeto maior com a nova *framework*:** Apesar de, desde o início, se prever que em projetos pequenos o React Native era muito viável, faltava testá-lo em projetos de maior dimensão, que envolvessem uma lógica de negócio maior e uma integração de serviços externos. Essa implementação do projeto com React Native acabou por ser feita com sucesso (Anexo L), sendo encontradas vantagens e também algumas desvantagens, como a falta de documentação para a realização de chamadas. Este problema foi, no entanto, ultrapassado com o desenvolvimento de uma *bridge* para código nativo detalhado (Anexo J).

Após terem sido cumpridos todos os objetivos principais associados ao projeto, foi possível concluir que a escolha da *framework* tinha sido a mais acertada. A empresa tem agora uma base bem construída que possibilita uma adaptação mais fácil dos programadores de JavaScript ao desenvolvimento móvel. Isto

permite reduzir os custos da empresa, uma vez que toda a sua base de trabalho é em JavaScript, fazendo com que seja possível realizar projetos de forma mais rápida e com menores riscos de desenvolvimento.

9.2 Principais contribuições

Consistindo este trabalho na realização de um estágio, as principais contribuições foram essencialmente práticas e focadas nos objetivos da empresa, na qual o estágio foi realizado e explicadas em 9.2.2. Contudo, existiram também contribuições teóricas que são explicadas em 9.2.1.

9.2.1 Teóricas

Uma vez que o React Native ainda é uma *framework* que se encontra no seu início de desenvolvimento, não existindo ainda uma versão estável até à data de escrita deste documento, no princípio do estágio não existiam estudos teóricos ou práticos que comparassem o React Native com outras *frameworks* ou, até mesmo, com o desenvolvimento nativo. Desta forma, foram publicados dois estudos relacionados com as principais *frameworks* JavaScript e um outro com maior foco no React Native em relação ao desenvolvimento nativo. As publicações foram as seguintes:

1. **JavaScript in mobile applications: React Native vs. Ionic vs. NativeScript vs. native development:** Estudo mais teórico considerando as variáveis mais importantes a ter em conta no desenvolvimento móvel e em que foram comparadas as principais *frameworks* de desenvolvimento híbrido em Javascript (React Native, Ionic e NativeScript) e soluções de desenvolvimento nativo (Anexo B);
2. **Análise de aprendizagem de *frameworks* mobile JavaScript:** Estudo mais prático através dos testes realizados aos programadores, em que foram comparadas as principais *frameworks* de desenvolvimento móvel JavaScript (Anexo E);
3. **Desenvolvimento móvel em Swift, Java e React Native:** Estudo de comparação para o mesmo projeto (audioguias), dos tempos de desenvolvimento das mesmas funcionalidades em Java / Swift e React Native e das principais dificuldades de cada solução para cada funcionalidade (Anexo K).

Para além das publicações, foi também criada uma taxonomia adaptada que permite mais facilmente avaliar programadores. O estudo de aprendizagem criado é um processo que pode ser aplicado a outras *frameworks*, servindo até para comparar a capacidade de adaptação entre diferentes linguagens de programação, através do conjunto de variáveis analisadas.

9.2.2 Práticas

A nível prático, as contribuições foram centradas principalmente na empresa. A principal contribuição foi a mudança de todo o processo de desenvolvimento móvel da empresa ter sido um sucesso. Este sucesso pode ser comprovado através das novas aplicações realizadas já em React Native, pois estas possuem um tempo de desenvolvimento muito mais reduzido do que as anteriores realizadas de forma nativa, sem evidenciarem qualquer alteração a nível de desempenho.

Também foram implementadas algumas práticas que foram usadas globalmente na empresa, como a adoção do novo padrão de JavaScript (ES6) e também do recurso a ferramentas de análise estática de código (*linting)*, abordados neste documento, o que contribuiu para uma maior normalização do código dentro da empresa.

Foi ainda criada uma arquitetura para a conceção de novos audioguias realizados na empresa, a qual permite que seja implementado qualquer audioguia de forma facilitada, bastando importar para o projeto os componentes/módulos já feitos e testados noutros projetos similares.

Foi também criada documentação para toda a empresa desde o *boilerplate* à utilização de código nativo através do React Native. O sucesso alcançado com o *boilerplate* criado fez com que esta técnica fosse posteriormente replicada para outras *frameworks* utilizadas na empresa, como o NodeJS para *back-end* e o AngularJS para *front-end web*.

9.3 Limitações e dificuldades

A maior dificuldade que surgiu neste projeto de estágio prendeu-se com o facto de não existir qualquer conhecimento da linguagem JavaScript, porque, apesar de ser uma das linguagens mais utilizadas/requisitadas atualmente no desenvolvimento de *software*, esta não é lecionada no ISEC.

9.4 Trabalho Futuro

Dada a rápida evolução a que se assiste nesta área de desenvolvimento, este tipo de trabalho nunca pode ser dado como finalizado, podendo sempre ser melhorado e evoluído para novas técnicas que vão surgindo no mercado.

Era interessante perceber, a nível teórico e prático, até que ponto, numa plataforma global com *back-end*, *front-end mobile* e *web*, poderá existir código reutilizável nos diversos contextos, caso seja tudo feito em React (a aplicação móvel em React Native e a parte *web* em ReactJS).

Seria também interessante a implementação de um modelo de avaliação de *frameworks*, totalmente validado, para outras empresas, baseado no estudo de aprendizagem realizado e nas variáveis que foram avaliadas nos testes aos programadores da Crossing Answers.

Para a Crossing Answers, seria interessante avaliar a possibilidade de transição do desenvolvimento *web*, de AngularJS para React, neste caso ReactJS. Como seria apenas uma alteração da *framework* e não de todo o processo de desenvolvimento, pensa-se que as vantagens poderão suplantar as desvantagens.

Referências bibliográficas

[1] C. Answers, “Crossing Answers Website,” 2018. [Online]. Available: https://crossinganswers.com/. [Accessed: 01-Sep-2018].

[2] ISEC, “ISEC,” 2018. [Online]. Available: https://www.isec.pt. [Accessed: 10-Dec-2018].

[3] NodeJS, “NodeJS,” 2018. [Online]. Available: https://nodejs.org. [Accessed: 10-Dec-2018].

[4] AngularJS, “AngularJS,” 2018. [Online]. Available: https://angularjs.org. [Accessed: 10-Dec-2018].

[5] ElectronJS, “ElectronJS,” 2018. [Online]. Available: https://electronjs.org/. [Accessed: 10-Dec-2018].

[6] R. Native, “React Native.” [Online]. Available: https://facebook.github.io/react-native/. [Accessed: 05-Apr-2018].

[7] NativeScript, “NativeScript.” [Online]. Available: 05/04/2018.

[8] Ionic, “Ionic.” [Online]. Available: https://ionicframework.com/. [Accessed: 05-Apr-2018].

[9] Statista, “Number of smartphone users worldwide from 2014 to 2020,” 2017. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed: 27-Dec-2018].

[10] Statista, “Worldwide internet user penetration from 2014 to 2021,” 2017. [Online]. Available: https://www.statista.com/statistics/325706/global-internet-user-penetration/. [Accessed: 27-Dec-2017].

[11] V. N Inukollu, D. D. Keshamon, T. Kang, and M. Inukollu, “Factors Influncing Quality of Mobile Apps: Role of Mobile App Development Life Cycle,” *Int. J. Softw. Eng. Appl.*, vol. 5, no. 5, pp. 15–34, 2014.

[12] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, “Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation,” *2015 2nd ACM Int. Conf. Mob. Softw. Eng. Syst.*, pp. 56–59, 2015.

[13] Statista, “Global mobile OS market share 2009-2017, by quarter,” 2017. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. [Accessed: 28-Dec-2017].

[14] A. Developers, “Kernel.” [Online]. Available: https://source.android.com/devices/architecture/kernel. [Accessed: 10-Dec-2018].

[15] F. Look, “What is Android?,” no. January, pp. 48–50, 2011.

[16] T. Vilcek and T. Jakopec, “Comparative analysis of tools for development of native and hybrid mobile applications,” *2017 40th Int. Conv. Inf. Commun. Technol. Electron. Microelectron.*, pp. 1516–1521, 2017.

[17] Kotlin, “Kotlin,” 2018. [Online]. Available: https://kotlinlang.org. [Accessed: 10-Dec-2018].

[18] TomislavCar, “Android development is 30% more expensive than iOS. And we have the numbers to prove it!,” 2015. [Online]. Available: https://infinum.co/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios. [Accessed: 31-Dec-2017].

[19] G. Motroc, “Can Kotlin overtake Java for Android development? New report says yes,” 2018. [Online]. Available: https://jaxenter.com/can-kotlin-overtake-java-report-138079.html. [Accessed: 15-Jan-2018].

[20] S. Jobs, “IPhone First Official Presentation,” in *Worldwide Developers Conference*, 2007.

[21] M. Kenney and B. Pon, “Structuring the smartphone industry: Is the mobile Internet OS platform the key?,” *J. Ind. Compet. Trade*, vol. 11, no. 3, pp. 239–261, 2011.

[22] N. Singh, “An comparative analysis of Cordova Mobile Applications V/S Native Mobile Application,” *Int. J. Recent Innov. Trends Comput. Commun.*, pp. 3777–3782.

[23] C. González García, J. Pascual-Espada, C. Pelayo G-Bustelo, and J. M. Cueva-Lovelle, “Swift vs. Objective-C: A New Programming Language,” *Int. J. Interact. Multimed. Artif. Intell.*, vol. 3, no. 3, p. 74, 2015.

[24] Statista, “App Download and Usage Statistics 2017,” 2017. [Online]. Available: http://www.businessofapps.com/data/app-statistics/. [Accessed: 14-Jan-2018].

[25] Optimus Information, “Native , Hybrid or Mobile Web Application Development Develop a Mobile Application Strategy for Your Business,” 2015.

[26] Ian Blair, “How To Choose Between Native, Hybrid or Web App For Your Business.” [Online]. Available: https://buildfire.com/choose-native-hybrid-web-mobile-app/. [Accessed: 07-Dec-2018].

[27] IBM, “Native, web or hybrid mobile-app development,” 2012.

[28] A. A. Menegassi and A. T. Endo, “An evaluation of automated tests for hybrid mobile applications,” *Proc. 2016 42nd Lat. Am. Comput. Conf. CLEI 2016*, 2017.

[29] Lavish, “What is a Xamarin,” 2017. [Online]. Available: https://www.thewindowsclub.com/what-is-xamarin-and-cross-platform-mobile-development. [Accessed: 12-Mar-2019].

[30] BeWare, “BeWare,” 2018. [Online]. Available: https://pybee.org. [Accessed: 10-Dec-2018].

[31] Rubymotion, “Rubymotion,” 2018. [Online]. Available: http://www.rubymotion.com/. [Accessed: 10-Dec-2018].

[32] R. Godwin-jones, “Emerging Technologies Mobile Apps for Language Learning,” *Lang. Learn. Technol.*, vol. 15, no. 2, pp. 2–11, 2011.

[33] A. Developers, "WebView," 2018. [Online]. Available: https://developer.android.com/reference/android/webkit/WebView. [Accessed: 10-Dec-2018].

[34] F. Asp Handledare, A. Palanisamy, O. Karlsson Examinator, and K. Sandahl, "A comparison of Ionic 2 versus React Native and Android in terms of performance, by comparing the performance of applications," 2017.

[35] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," *Proc. 6th Balk. Conf. Informatics - BCI '13*, p. 213, 2013.

[36] GitHub, "The State of the Octoverse 2017," 2017. [Online]. Available: https://octoverse.github.com/. [Accessed: 14-Jan-2018].

[37] M. Hafiz, S. Hasan, Z. King, and A. Wirfs-Brock, "Growing a language: An empirical study on how (and why) developers use some recently-introduced and/or recently-evolving JavaScript features," *J. Syst. Softw.*, vol. 121, pp. 191–208, 2016.

[38] StackShare, "Ionic vs React Native vs NativeScript," 2017. [Online]. Available: https://stackshare.io/stackups/ionic-vs-nativescript-vs-react-native. [Accessed: 31-Dec-2017].

[39] Facebook, "React," 2018. [Online]. Available: https://reactjs.org/. [Accessed: 15-May-2018].

[40] I. Suzdalnitski, "React.js: a better introduction to the most powerful UI library ever created," 2018. .

[41] Y. El Azizi, "React tools you need to use in your components development," 2018. [Online]. Available: https://dev.to/elaziziyoussouf/tools-you-need-to-use-in-your-react-components-development--13a7. [Accessed: 12-Oct-2018].

[42] T. Occhino, "React JS conf," 2015. [Online]. Available: https://code.facebook.com/videos/786462671439502/react-js-conf-2015-keynote-introducing-react-native-/. [Accessed: 15-May-2018].

[43] Drifty, "Drifty." [Online]. Available: https://www.crunchbase.com/organization/ionic#section-overview. [Accessed: 12-Oct-2018].

[44] H. Khanna, R., Yusuf S., & Phan, *Ionic: Hybrid Mobile App Development*. Birmingham, 2017.

[45] C. Kunkku and S. Oy, "Developing a hybrid mobile application with Ionic," Lahti University, 2018.

[46] Telerik, "Telerik," 2018. [Online]. Available: https://www.telerik.com/. [Accessed: 12-Oct-2018].

[47] B. D. Veselý, "Analysis and experiments with NativeScript and React Native framework," *Masaryk Univ. Fac. Informatics Anal.*, 2017.

[48] L. Šmitalová, "NativeScript application for Continuous Improvement of Software Engineer's skills," 2017.

[49] R. Halidovic and G. Karli, "www.iosrjen.org Cross-Platform Mobile App Development using HTML5 and JavaScript while leveraging the Cloud," *IOSR J. Eng.*, vol. 04, no. 2, pp. 2250–3021,

2014.

[50] Icenium, “Icenium,” 2018. [Online]. Available: https://github.com/Icenium. [Accessed: 12-Oct-2018].

[51] Y. A. Redda, “Cross platform Mobile Applications Development Mobile Apps Mobility,” no. June, pp. 1–77, 2012.

[52] H. Ahuja and R. Johari, “Optimization of the issues in the migration from Android Native to Hybrid Application: Case study of Student’s Portal application,” *Proc. 2014 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2014*, pp. 245–249, 2014.

[53] K. Goals, “Bloom ’ s Taxonomy of Educational Objectives.”

[54] K. Words, “Bloom ’ s Taxonomy : A New Look at an Old Standby,” pp. 1–6, 2012.

[55] D. R. Krathwohl, “A Revision of Bloom ’ s Taxonomy :,” vol. 41, no. 4, pp. 212–219, 2002.

[56] B. J. Brito Hugo, Gomes Anabela, Santos Álvaro, “JavaScript em aplicações móveis: React Native vs Ionic vs NativeScript vs desenvolvimento nativo,” *CISTI*, 2018.

[57] F. B. Barreto, “Portugal dos Pequenitos,” 2018. [Online]. Available: http://www.portugaldospequenitos.pt/. [Accessed: 01-Sep-2018].

[58] C. Answers, “Aplicação Android Portugal dos Pequenitos,” 2018. [Online]. Available: https://play.google.com/store/apps/details?id=com.crossinganswers.bissaya.guide.

[59] C. Answers, “Aplicação iOS Portugal dos Pequenitos,” 2018. [Online]. Available: https://itunes.apple.com/us/app/portugal-dos-pequenitos/id1383686371?l=pt&ls=1&mt=8. [Accessed: 03-Sep-2018].

[60] D. G. Young, “Android Beacon Library.” [Online]. Available: https://github.com/AltBeacon/android-beacon-library. [Accessed: 10-Dec-2018].

[61] Atlassian, “What is a kanban board?” [Online]. Available: https://www.atlassian.com/agile/kanban/boards. [Accessed: 13-Mar-2019].

[62] F. B. Barreto, “Casa Museu Bissaya Barreto,” 2018. [Online]. Available: http://www.cmbb.pt/. [Accessed: 03-Sep-2018].

[63] C. Answers, “Aplicação Android Casa Museu Bissaya Barreto,” 2018. [Online]. Available: https://play.google.com/store/apps/details?id=com.crossinganswers.bissaya.casamuseu.

[64] C. Answers, “Aplicação iOS Casa Museu Bissaya Barreto,” 2018. [Online]. Available: https://itunes.apple.com/us/app/casa-museu-bissaya-barreto/id1383727184?l=pt&ls=1&mt=8. [Accessed: 04-Sep-2018].

[65] Facebook, “React Native Documentation,” 2019. [Online]. Available: https://facebook.github.io/react-native/docs/getting-started.html. [Accessed: 15-Dec-2019].

[66] K. Chinnathambi, *Learning react*. New York: Pearson Education, Inc., 2018.

[67] R. Native, “Who’s using React Native?,” 2017. [Online]. Available: http://facebook.github.io/react-native/showcase.html. [Accessed: 30-Dec-2017].

[68] CodeCademy, “Learn Basic React,” 2018. [Online]. Available: https://www.codecademy.com/learn/react-101. [Accessed: 10-Mar-2018].

[69] CodeCademy, “Learn Adavanced React,” 2018. [Online]. Available: https://www.codecademy.com/learn/react-102. [Accessed: 10-Mar-2018].

[70] R. S. Engelschall, “ECAMAScript 6 (ES6).” [Online]. Available: http://es6-features.org. [Accessed: 13-Mar-2019].

[71] E. Elliott, “JavaScript ES6+: var, let, or const?,” 2015. [Online]. Available: https://medium.com/javascript-scene/javascript-es6-var-let-or-const-ba58b8dcde75. [Accessed: 13-Mar-2019].

[72] Mozilla, “JavaScript 2015 Array.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array. [Accessed: 27-Jan-2019].

[73] Codedamn, “React Native tutorial.” [Online]. Available: https://www.youtube.com/watch?v=EMoXvr0Q9LE&list=PLYxzS__5yYQlHANFLwcsSzt3elIbYTG1h. [Accessed: 13-Mar-2019].

[74] Redux, “Redux.” [Online]. Available: https://redux.js.org/introduction. [Accessed: 05-Apr-2018].

[75] R. Navigation, “React Navigation,” 2018. [Online]. Available: https://reactnavigation.org. [Accessed: 12-Oct-2018].

[76] Hectahertz, “Biblioteca react-native-typography.” [Online]. Available: https://github.com/hectahertz/react-native-typography. [Accessed: 12-Oct-2018].

[77] R. Peters, “TabNavigator,” 2018. [Online]. Available: https://codeburst.io/ricky-figures-it-out-simple-react-native-tabnavigator-using-react-navigation-592945a3b211. [Accessed: 11-Dec-2018].

[78] Vue, “Framework Vue.” [Online]. Available: https://vuejs.org. [Accessed: 11-Dec-2018].

[79] Chirpy_me, “Redux Sagas.” [Online]. Available: https://medium.com/@aksudupa11/redux-sagas-714370b61692. [Accessed: 11-Dec-2018].

[80] Rt2zz, “Redux-persist,” 2018. [Online]. Available: https://github.com/rt2zz/redux-persist. [Accessed: 10-Dec-2018].

[81] G. Developers, “Fabric.” [Online]. Available: https://get.fabric.io. [Accessed: 14-Mar-2019].

[82] I. Paul Anderson, GrammaTech, “Static analysis vs dynamic testing - No competition,” 2011. [Online]. Available: https://www.embedded.com/design/other/4214351/Static-Analysis-vs--Dynamic-Testing---No-Competition-. [Accessed: 14-Mar-2019].

[83] M. Zitser, D. E. S. Group, and T. Leek, “Testing Static Analysis Tools using Exploitable Buffer Overflows from Open Source Code,” pp. 97–106, 1996.

[84] Microsoft, “Visual Studio Code.” [Online]. Available: https://code.visualstudio.com. [Accessed: 14-Mar-2019].

[85] GitHub, “Atom.” [Online]. Available: https://atom.io. [Accessed: 14-Mar-2019].

[86] C. Zakas, “ESLint.” [Online]. Available: https://eslint.org. [Accessed: 11-Dec-2018].

[87] Airbnb, “Airbnb ESLint Rules.” [Online]. Available: https://www.npmjs.com/package/eslint-config-airbnb. [Accessed: 11-Dec-2018].

[88] M. Cone, “Markdown Guide,” 2018. [Online]. Available: https://www.markdownguide.org/basic-syntax. [Accessed: 10-Dec-2018].

[89] S. Buna, *React.js Succintly*. Morrisville: Syncfusion, 2016.

[90] P. S. Meter, *React.js by example*. PACKT, 2016.

[91] Facebook, “Composition vs Inheritance,” 2018. [Online]. Available: https://reactjs.org/docs/composition-vs-inheritance.html. [Accessed: 10-Dec-2018].

[92] Google, “Maps JavaScript API.” [Online]. Available: https://developers.google.com/maps/documentation/javascript/get-api-key. [Accessed: 11-Dec-2018].

[93] Mapbox, “Mapbox,” 2018. [Online]. Available: https://www.mapbox.com/. [Accessed: 10-Dec-2018].

[94] Unity, “Unity.” [Online]. Available: https://unity.com/. [Accessed: 11-Dec-2018].

[95] Mapbox, “Mapbox Documentation.” [Online]. Available: https://www.mapbox.com/documentation. [Accessed: 11-Dec-2018].

[96] TopoGrafix, “GPX Developer’s Manual,” 2018. [Online]. Available: https://www.topografix.com/gpx_manual.asp. [Accessed: 10-Dec-2018].

[97] Z. Wang, “React Native Sound,” 2018. [Online]. Available: https://github.com/zmxv/react-native-sound. [Accessed: 10-Dec-2018].

[98] K. Schwaber, “What is a SCRUM?” [Online]. Available: https://www.scrum.org/resources/what-is-scrum?gclid=Cj0KCQiAgMPgBRDDARIsAOh3uyJo9ac2G2HCaDpKYxPe9DKFCOFYNf1gOh-u-iRJDWPeZJnPt8BFT4kaAmaWEALw_wcB. [Accessed: 12-Dec-2018].

[99] Facebook, “Design Principles.” [Online]. Available: https://reactjs.org/docs/design-principles.html. [Accessed: 11-Dec-2018].

Anexo A – Proposta de Estágio

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA E DE SISTEMAS

CROSSING ANSWERS

PROPOSTA DE ESTÁGIO

Ano Lectivo de 2017/2018

em Mestrado em Informática e Sistemas (Desenvolvimento de Software)

TEMA

JavaScript para aplicações móveis

SUMÁRIO

O estágio consiste na investigação das vantagens e desvantagens de JavaScript na construção de aplicações móveis. Bem como a implementação dos projectos de Áudio Guias e CONNECTS.

1. ÂMBITO

O projeto que aqui se propõe, enquadra-se no âmbito da disciplina de projeto/estágio do curso de informática e sistemas do Departamento de Informática e de Sistemas (DEIS) do Instituto Superior de Engenharia de Coimbra (ISEC). Tem como áreas de trabalho essenciais o desenvolvimento de aplicações móveis, investigação de novas tecnologias como o react-native, ionic e nativeScript, com outras tecnologias nativas.

2. OBJECTIVOS

O presente projeto pretende atingir os seguintes objetivos genéricos:

- Investigação e comparação de frameworks de desenvolvimento mobile em JavaScript (react-native, ionic e nativeScript);
- Investigação sobre tempos de aprendizagem, e vantagens das várias frameworks, e comparação com desenvolvimento nativo.
- Desenvolvimento de competências ao nível de construção de software com arquitetura adaptativa a várias soluções futuras;
- Aquisição de um domínio completo dos diversos meios necessários ao desenvolvimento de um produto com “principio, meio e fim”;

- Realização de trabalho de equipa com colaboração e cooperação num ambiente multidisciplinar;

3. PROGRAMA DE TRABALHOS

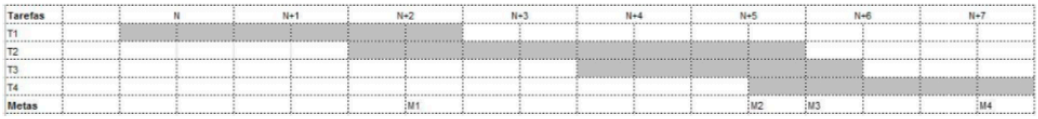
O estágio consistirá nas seguintes actividades e respectivas tarefas:

- ***T1*** – *Investigação, estudo e comparação das várias plataformas* – Pesquisa e comparações de react-native, ionic, nativeScript com as nativas (Android, iOs), e estudo de comparação de vantagens, desvantagens para todas.
- ***T2*** – *Recolha de dados das tecnologias* – Estudo do tempo de aprendizagem em cada uma das tecnologias e de tempo de inicialização ao desenvolvimento em cada uma, preparando e aplicando testes práticos com outros programadores e ou estudantes.
- ***T3*** – *Desenvolvimento de arquitetura de áudio guias* – Através de uma arquitetura criada e optimizada pelo estagiário, e criação de áudio guias para Alter do Chão, Sesimbra e Penela.
- ***T4*** – Desenvolvimento CONNECTS – Desenvolvimento do projeto de rede social de contactos profissionais.

4. CALENDARIZAÇÃO DAS TAREFAS

As Tarefas acima descritas, incluindo os testes de validação de cada módulo, serão executadas de acordo com a seguinte calendarização:

O plano de escalonamento dos trabalhos é apresentado em seguida:



INI		Início dos trabalhos
M1	(INI + 12 Semanas)	Tarefa T1 terminada
M2	(INI + 24 Semanas)	Tarefa T2 terminada
M3	(INI + 26 Semanas)	Tarefa T3 terminada
M4	(INI + 32 Semanas)	Tarefa T4 terminada

5. RESULTADOS

No final do estágio pretende-se ter um estudo pormenorizado sobre as vantagens das diferentes frameworks em JavaScript em relação a desenvolvimento nativo e comparação de todas as frameworks JavaScript entre si mencionando as vantagens e desvantagens de cada uma.

6. LOCAL DE TRABALHO

Crossing Answers, Coimbra. O horário de trabalho será das 10:00 Horas ás 19:00 Horas, com uma hora de descanso diário para almoço, e com direito a dois dias de descanso semanal sendo estes Sábado e Domingo

7. METODOLOGIA

A metodologia associada é a metodologia SCRUM. O progresso do trabalho é avaliado através da realização de reuniões periódicas de forma a fazer um acompanhamento de todas as etapas e tarefas. As principais etapas serão documentadas através da escrita de relatórios intercalares e documentação adequada.

8. ORIENTAÇÃO

ISEC:

Nome (nome@isec.pt)
Categoria

Entidade de Acolhimento:
Nome: Cristóvão Cleto (cristovaocleto@crossinganswers.com)
Categoria: Diretor Executivo

9. CARACTERIZAÇÃO E REMUNERAÇÃO

- Data de início – Quando possível pela parte do ISEC
- Data de fim – De acordo com a regulamentação do ISEC e do estágio
- Horário – 10:00 às 19:00

Anexo B – Artigo JavaScript em aplicações móveis: React Native vs Ionic vs NativeScript vs desenvolvimento nativo

JavaScript em aplicações móveis: React Native vs Ionic vs NativeScript vs desenvolvimento nativo

JavaScript in mobile applications: React Native vs Ionic vs NativeScript vs native development

Hugo Brito, Anabela Gomes, Álvaro Santos e Jorge Bernardino
ISEC – Instituto Superior de Engenharia de Coimbra
IPC – Instituto Politécnico de Coimbra
Coimbra, Portugal
a21220118@isec.pt, anabela@isec.pt, ans@isec.pt, jorge@isec.pt

Resumo — **A utilização de dispositivos móveis está cada vez mais disseminada, sendo o seu desenvolvimento dividido em três grupos: aplicações nativas, híbridas e web. Neste artigo é feita uma análise comparativa entre aplicações móveis nativas e híbridas baseadas em JavaScript, nomeadamente React Native, NativeScript e Ionic. A comparação é efetuada usando os 7 critérios considerados mais relevantes para o desenvolvimento de aplicações móveis. Este trabalho mostra que o React Native apresenta os melhores resultados em quase todos os critérios analisados, tendo ainda vantagem no desenvolvimento híbrido em relação ao nativo. Com o aparecimento de frameworks para desenvolvimento móvel e algumas com pouco mais de um ano de existência, existe a dificuldade de perceber quais as mais vantajosas para um determinado objetivo de negócio, mostrando neste artigo as melhores opções entre as frameworks, fazendo sempre a comparação com o desenvolvimento nativo.**

Palavras Chave - React Native, Ionic, NativeScript, JavaScript, aplicações móveis híbridas

Abstract — **The mobile applications development is composed by three groups: natives, hybrids and web. In this paper a comparison between the native and hybrid mobile applications build on JavaScript (Reactive Native, NativeScript and Ionic) is done. The analysis is done using the 7 more relevant principles to the mobile applications development. This paper shows that React Native exhibits the best results in all the analyzed principles and still having benefits in the hybrid development in relation to native. With the emergence of frameworks for mobile development, some of them with a little more than a year of existence, there is the difficulty to perceive which are the most advantageous for a given business objective, this article shows the best options among the frameworks used, always comparing with the native development.**

Keywords - React Native, Ionic, NativeScript, JavaScript, hybrid mobile applications.

I. INTRODUÇÃO

Os dispositivos móveis estão cada vez mais avançados e eficientes, estimando-se que até 2020 existam cerca de 2.87 mil milhões de utilizadores de smartphones [1]. Segundo Singh [2], a Myntra.com, uma famosa plataforma de compras online asiática encerrou as vendas através do seu site, passando a ser apenas possível efetuar compras a partir da aplicação móvel. Esta decisão é justificada por ter sido verificado que 90% da receita total provinha apenas da aplicação móvel.

O número de utilizadores de Internet a nível global previsto para 2020 é de 52,4% [3], estimando-se que cerca de 34% da população mundial terá pelo menos um smartphone. Isto vem reforçar a importância dos dispositivos móveis no dia-a-dia de toda a sociedade, pois aliam o poder de computação à portabilidade.

Assim, os smartphones são utilizados para tarefas mais complexas e de maior processamento, devido à diferença de eficiência entre um computador e um smartphone estar a diminuir, justificada pelo rápido aumento da qualidade dos processadores, estabilização do sistemas operativos, e melhoria na conectividade e armazenamento [4].

Consequentemente há um interesse crescente no desenvolvimento de aplicações móveis, cujo número de publicações e atualizações aumenta a cada semestre [5]. Estas aplicações são normalmente construídas para que sejam distribuídas tanto na Google Play (plataforma de distribuição de aplicações para Android) como na App Store (plataforma de distribuição de aplicações para iOS). O desenvolvimento tem de ter em conta as particularidades de cada um dos sistemas operativos referidos, acrescido do facto da linguagem de programação para cada sistema ser totalmente distinta.

A solução para o problema da construção de aplicações de forma rápida, alocando menos recursos, mas dando a possibilidade de distribuição em lojas oficiais, passa pelo desenvolvimento híbrido, com a maioria do código a servir para ambos os sistemas operativos.

Atualmente existem muitas frameworks para desenvolvimento móvel. O JavaScript tornou-se uma linguagem muito popular, tendo em 2017, mais do dobro de repositórios de GitHub em relação à segunda linguagem mais popular, o Python (2.3 milhões para 1.1 milhões) [6]. Esta popularidade tem vindo a aumentar devido a uma evolução rápida da linguagem e ao

propósito geral para que foi construída, tornando possível a construção de uma equipa de trabalho em diferentes especialidades, tecnologias ou projetos [7].

Assim, têm vindo a ser criadas frameworks que permitem o desenvolvimento móvel de forma mais simples, através do JavaScript. As selecionadas para este estudo, React Native, NativeScript e Ionic. surgiram de uma análise inicial relativamente às que têm maior reconhecimento ou possibilidade de evolução e estabilização notória no futuro [8].

Na realização deste estudo são analisados os critérios que mais influenciam a escolha de uma linguagem de programação para dispositivos móveis. Foi assim possível aferir mais facilmente todas as vantagens e desvantagens do desenvolvimento móvel híbrido vs nativo e, no desenvolvimento híbrido, qual a melhor solução entre React Native, NativeScript e Ionic de acordo com os vários critérios.

Este artigo está estruturado da seguinte forma. Na secção II são apresentados os principais conceitos de aplicações móveis, mais importantes para este estudo. Na secção III são descritos os critérios para aplicação da análise e a forma como foram aplicados. Na secção IV são apresentados e justificados todos os resultados obtidos da metodologia aplicada. Na secção V é feita uma discussão sobre os resultados comparando com outros trabalhos relacionados. Por fim, na secção VI são apresentadas algumas conclusões e sugestões de trabalho futuro.

II. APLICAÇÕES MÓVEIS

Apesar da existência de vários sistemas operativos para dispositivos móveis como o Windows Phone, Firefox OS ou Tizen, o Android e o iOS alcançaram no segundo trimestre de 2017 uma quota de mercado conjunta de 99.8% nos novos dispositivos, sendo os dois sistemas operativos de maior relevância [9].

O Android é um sistema operativo baseado no kernel do Linux. Inicialmente o Android foi projetado para smartphones baseados em toque, contudo agora é também usado em sistemas Desktop, TVs ou relógios [2]. A Google utilizou o Java como a principal linguagem para desenvolvimento das suas aplicações. Inicialmente desenvolveu uma extensão ao ambiente de desenvolvimento (IDE) Eclipse para facilitar o desenvolvimento de aplicações Android. Posteriormente, tendo por base o ambiente de desenvolvimento IntelliJ da empresa JetBrains disponibilizou o Android Studio, que é um IDE mais intuitivo e mais acessível para a construção de aplicações Android [10]. De forma a tornar o desenvolvimento menos suscetível a erros, mais simples, conciso, seguro e rápido, a Google criou a linguagem Kotlin. A razão prende-se com o facto do desenvolvimento em Android ser cerca de 30% mais lento do que a concorrência iOS [11], estando atualmente a ser adotada por grande parte da comunidade [12]. No primeiro trimestre de 2017, a participação de mercado dos dispositivos móveis com sistema operativo Android era de 87.7% [9].

O iOS é um sistema operativo móvel projetado pela Apple e distribuído apenas para equipamentos Apple. Funcionando em iPhone, iPad, iPod e Apple TV, é reconhecido entre outros aspetos pelos seus recursos de segurança, associado a um processo de revisão de novas aplicações para a App Store mais exaustivo do que o utilizado em sistemas Android [2]. A Apple durante vários anos utilizou o Objective-C como a principal linguagem para desenvolvimento das suas aplicações. A partir de 2014 houve uma gradual mudança para Swift, devido a ser uma linguagem mais recente e com um tempo de implementação mais curto, para além das similaridades com linguagens como JavaScript ou Python, conseguindo uma performance quase idêntica ao C++ [13]. No segundo trimestre de 2017, cerca de 12.1% dos dispositivos móveis globais executavam o sistema operativo iOS e, consequentemente, da marca Apple [9].

Apesar do esforço tanto da Google como da Apple, para uma melhoria constante nas linguagens e componentes com o objetivo de diminuir o esforço necessário dos programadores, o problema central da distribuição de aplicações móveis continua. Verifica-se que, as aplicações para smartphones Android e iOS são desenvolvidas com linguagens de programação diferentes, levando a um maior tempo de desenvolvimento de uma ideia de negócio, sendo necessária a implementação de duas lógicas de programação diferentes, uma após a outra ou então ter uma equipa de desenvolvimento especializada para cada uma das plataformas.

Atualmente a construção de aplicações móveis pode ser classificada em três grandes grupos [14]:

- *Aplicações nativas:* Desenvolvidas utilizando a plataforma de programação nativa do dispositivo, acedendo diretamente a funções do sistema operativo e possibilidade de distribuição das mesmas nas lojas oficiais [14], [15].
- *Aplicações Web:* Desenvolvidas com tecnologias utilizadas em Web como HTML5, CSS3 e JavaScript. São executadas sobre uma Web View, não tendo acesso aos recursos mais avançados do sistema operativo nem possibilidade de distribuição nas lojas oficiais [14], [15].
- *Aplicações Híbridas:* Tentam combinar as vantagens das aplicações Web e nativas numa única plataforma e linguagem, com possibilidade de execução em vários sistemas operativos, havendo por vezes a possibilidade de a compilação ser feita de forma nativa [14], [16].

As frameworks híbridas analisadas são as seguintes:

- *React Native* - Criada pela empresa Facebook, sendo esperada uma primeira versão estável ainda em 2018. Apesar do desenvolvimento ser feito em JavaScript e baseando os componentes visuais em React. Esta framework é considerada revolucionária devido à sua tentativa de aproximação com os componentes e código nativo de cada um dos sistemas operativos.
- *Ionic* - É uma estrutura de desenvolvimento de aplicações móveis que pode ser dividida em três componentes principais. O HTML que serve para executar os componentes originais, a interface com estilo nativo, fornecida pela framework, e a plataforma de wrapper que é o Cordova [17].
- *NativeScript* - é uma framework de código aberto disponibilizado sob a licença Apache e suportado pela Telerik. O código base pode ser escrito em Angular 2, TypeScript ou JavaScript 6. A lógica é desenvolvida em

JavaScript e a interface em ficheiros XML com recurso a CSS, podendo haver a possibilidade de desenvolvimento de código específico para alguma das duas plataformas [18].

III. METODOLOGIA

Na execução do trabalho de análise foram sempre utilizados os mesmos dispositivos móveis para testes. No caso do Android foi utilizado Xiaomi Mi A1 com AndroidOne muito próximo do Android puro e no caso iOS o iPhone 6, com a versão 11.

Antes da avaliação das diferentes frameworks é necessário ter em consideração quais os critérios que são essenciais no desenvolvimento de aplicações móveis, e que podem ter preponderância na escolha do modelo de implementação do negócio por parte de uma entidade ou programador.

Os critérios selecionados foram escolhidos de acordo com as necessidades principais do desenvolvimento móvel [2], [14]

- *Aprendizagem e qualidade da documentação:* Avalia o progresso de um programador durante o seu primeiro contacto com a documentação e respetiva rapidez de aprendizagem; verifica a comunidade ativa para cada uma das frameworks incluídas na análise e a qualidade da própria documentação oficial.
- *Custo de desenvolvimento:* Avalia os custos inerentes ao desenvolvimento de uma aplicação através da linguagem selecionada. Este é um dos critérios mais relevantes para pequenas e médias empresas.
- *Emuladores e depuração:* Avalia a capacidade de utilização dos recursos do emulador e sua instalação, mas também o tempo necessário para após uma alteração de código ser novamente possível testar a aplicação.
- *Tempo de resposta e velocidade:* Avalia a latência da resposta que uma aplicação pode ter ao toque e capacidade de resposta na interação com o utilizador.
- *Reconhecimento Comercial:* Este critério pretende aferir o reconhecimento em termos de quantidade de utilizadores das aplicações criadas e publicadas pelas respetivas frameworks analisadas, de forma a haver garantias de um maior LTS (*long term support*).
- *Reutilização de código e trabalho em equipa:* Avalia a possibilidade da fácil reutilização de código-fonte para outros projetos ou outras abordagens do mesmo projeto, de forma a aumentar a velocidade de desenvolvimento de um projeto com recursos já criados, bem como a facilidade de trabalho em equipa dentro de uma organização.
- *Manutenções e atualizações:* Este critério avalia a capacidade para efetuar modificações e alterações às funcionalidades existentes, para incluir novas funcionalidades, para aplicar melhorias gerais e velocidade de implementação de todas essas modificações.

Para a análise dos critérios, foram realizados testes simples como a instalação e criação de pequenos componentes, com as várias frameworks de forma a poder analisar os vários critérios. A análise foi completada com a execução das três aplicações referenciadas nos sites das frameworks como aplicações de maior sucesso. No site do React Native [19] as aplicações com maior sucesso referenciadas são: Instagram, Skype e Airbnb. De acordo com o site do NativeScript [20] as aplicações com maior sucesso foram: PNP 2017 Portable North Pole, Mijn Inkomen Later e DockBooking. Segundo o site da Ionic [21], as aplicações de maior sucesso com a framework foram: Untappd Discover Beer, JustWatch – Filmes e Séries e MarketWatch. Relativamente às aplicações nativas, as selecionadas foram baseadas no maior número de downloads, no caso do Youtube, ou o Gmail e Google Maps por serem criadas pela mesma empresa do sistema operativo Android.

IV. RESULTADOS

Após uma avaliação das aplicações mais reconhecidas para cada framework e após a realização de testes de implementação simples, os resultados são apresentados numa escala de Likert de 1 a 5. O valor 1 representa uma eficácia muito baixa para o critério em análise ou não é aplicado de todo. O valor 2 significa que a eficácia para cumprir esse critério é abaixo do esperado. O valor 3 é atribuído quando não são encontradas vantagens nem desvantagens significativas. O valor 4 representa que uma certa framework ou linguagem têm vantagens que poderão ser aproveitadas. O valor 5 aplica-se quando o critério é satisfeito em tudo o que é pretendido na análise de um dado critério.

Neste caso os resultados para os critérios analisados foram os seguintes:

- *Aprendizagem e qualidade da documentação:* Para programar em multiplataforma de forma nativa, é necessário aprender duas linguagens totalmente distintas e também conhecer duas abordagens totalmente diferentes na criação de interfaces. Contudo existe uma grande estabilização no mercado, uma documentação de excelência e comunidades de desenvolvimento muito grandes.
  No caso das soluções híbridas analisadas existe apenas uma única linguagem a aprender (JavaScript), sendo que, dentro de cada framework, existem os seus detalhes mais complicados como o redux no React Native, o typeScript no NativeScript ou as animações em Ionic. Por isso, entre as frameworks JavaScript, nenhuma delas se destaca mais que a outra em rapidez de aprendizagem. No que diz respeito à qualidade da documentação entre as híbridas, o React Native destaca-se consideravelmente em relação às demais, tendo uma documentação extremamente bem estruturada muito ao estilo da existente para Android. A comunidade de React Native é muito extensa com aproximadamente 6 vezes mais referências que a de Ionic e cerca de 12 vezes mais referências do que a de NativeScript, referências contabilizadas após pesquisas em motores de busca sobre as frameworks referenciadas.
- *Custo de Desenvolvimento:* As aplicações móveis nativas têm um custo de desenvolvimento mais elevado do que as híbridas, principalmente pelo facto de serem realizadas duas aplicações totalmente iguais mas com lógica de programação totalmente distinta. No caso das híbridas é apenas implementada uma aplicação, tendo

um custo de desenvolvimento mais reduzido. Contudo, não se pode considerar que a nível de custos as soluções híbridas versus nativas estejam em avaliações totalmente extremas. Isto porque os custos no caso das híbridas podem aumentar caso existam integrações com serviços externos como mapas, chat, voip ou push notifications. Esta situação deve-se ao facto de existirem exemplos de código nativo, muitas vezes fornecidos pelo serviço externo, o qual não fornece exemplos similares para soluções híbridas. Ou seja, neste cenário, uma aplicação nativa com muitas integrações, os custos de desenvolvimento até podem tornar-se menores.

- *Emuladores e depuração:* Todas as frameworks analisadas permitem a utilização de emuladores, contudo, a utilização do NativeScript revelou-se mais difícil em termos de configurações iniciais. Relativamente à depuração, nas nativas para ser testada uma alteração de código num emulador ou dispositivo físico normalmente é feita uma nova build do código, apesar de agora existir o Instant Run para Android que acelera o processo de compilação do código. No Ionic, após ser guardado o código no editor de texto, só é necessário recarregar a página do browser para testar as alterações na aplicação, sendo um processo com apenas duas ações e rápido de executar, sendo muito idêntico à forma de programação Web. Tanto no React Native como no NativeScript é possível ver as alterações no código logo após ser guardado no editor de texto, denominado por "hot reload", fazendo quase instantaneamente o carregamento da nova versão do código. Isto é muito útil para criação de componentes e estilos que requerem por vezes vários testes, de forma a verificar se o resultado da interface está de acordo com o design idealizado para a aplicação.
- *Tempo de resposta e velocidade:* Após o teste de várias aplicações, as nativas têm um tempo de resposta quase instantâneo. Tanto o Youtube, como as duas aplicações da Google, têm tempos de resposta ao clique quase impercetíveis. No React Native, as aplicações mostram uma resposta exatamente igual às nativas. Apenas em navegações da aplicação mais profundas, como por exemplo no 4º ou 5º nível na pilha de navegação, se nota uma ligeira diferença em relação ao comportamento nativo, mas impercetível à vista do utilizador comum. No Native Script, nas três aplicações testadas, os tempos de resposta ao clique foram em todas semelhantes às do React Native. Nesta situação constata-se o mesmo problema relativo às pilhas de navegação mais profundas. No Ionic, em todas as aplicações instaladas, o tempo de resposta ao clique é maior e já com uma latência percetível por qualquer utilizador. As diferenças tanto para o React Native como para o NativeScript são impercetíveis, ou seja, estas duas frameworks de JavaScript mostram resultados muito fiáveis a nível de velocidade. Isto verifica-se apesar de o Native Script não utilizar muitos componentes nativos móveis, sendo mais idênticos à web, com consequente repercussão na usabilidade. O Ionic apresenta os piores resultados de velocidade tanto de carregamento como de disponibilização de conteúdos, contudo sem nunca por em causa uma utilização fluída e rápida da aplicação.
- *Reconhecimento Comercial:* Grande parte do mercado atualmente está desenvolvido de forma nativa, sendo que grandes aplicações como Youtube, Google Maps ou uma parte do Facebook estão implementadas nativamente. O React Native, apesar de recente, tem tido um reconhecimento de mercado exponencial, de tal forma que parte do Facebook está desenvolvido em React Native. O Instagram, Skype, Airbnb ou até mesmo a Tesla, são exemplos de nomes sonantes desenvolvidos nesta framework. Em NativeScript não há aplicações de grande reconhecimento e, apesar de um número elevado de aplicações já distribuídas em NativeScript nas lojas, em termos de reconhecimento comercial ainda é baixo. Em Ionic não há aplicações de conhecimento global, mas já há várias com níveis de utilização ligeiramente superiores ao NativeScript, contudo sempre muito longe quando comparadas com as nativas ou até mesmo React Native.
- *Reutilização de código e trabalho em equipa:* As soluções nativas não apresentam vantagens nem desvantagens a nível da reutilização. No caso das frameworks multiplaforma todas as soluções foram criadas com o objetivo de criação de componentes possíveis de reutilizar em outros projetos. Na construção da interface do utilizador é possível utilizar todos esses componentes já pré-feitos, passando apenas como argumentos dos componentes as propriedades ou estilos específicos de configuração de cada um. Para além disso, o React Native vai mais além, dando possibilidade de reutilização de código relativo à lógica desenvolvida para uma página web do tipo React com as mesmas funcionalidades, sendo só alterados os componentes nativos móveis para componentes Web para tornar uma aplicação numa página web com as mesmas funcionalidades. Desta forma, o React Native e NativeScript têm também a vantagem de terem por base duas das frameworks mais utilizadas na Web, React e Angular 2, respetivamente. O trabalho de equipa nestes dois casos ainda se torna mais multidisciplinar, podendo facilmente existir cooperação ou resoluções de problemas em projetos de Web. Já no caso das nativas existem programadores específicos alocados para cada um dos sistemas operativos. Assim, a ajuda mútua torna-se difícil pelo facto de serem linguagens totalmente distintas. Em relação às plataformas nativas, o Ionic apresenta vantagem por poder ser utilizada a mesma linguagem por toda a equipa de desenvolvimento móvel, mas possui desvantagens no reaproveitamento em relação às outras duas frameworks híbridas.
- *Manutenções e atualizações:* Para fazer manutenção ou atualização de uma aplicação já existente no mercado surge o problema principal para as soluções nativas: a alteração do código tem que ser realizada em dois

projetos diferentes (Android e iOS). Neste caso, as híbridas têm vantagem em relação às nativas, pois a manutenção pode ser feita através de um único projeto. No entanto, quando existe um erro específico num sistema operativo este pode ser facilmente resolvido no caso das soluções nativas, num único projeto, enquanto que nas soluções híbridas essa correção pode ter repercussões para outro sistema operativo que poderia estar funcional. Esta situação referida pode ocorrer apesar de nas três frameworks ser possível fazer a diferenciação de código consoante a plataforma. O Ionic e o NativeScript têm uma ligeira vantagem em relação ao React Native, pela existência de versões estáveis, que faz com que as manutenções ou atualizações tenham um grau de dificuldade menor em relação ao React Native. De notar que o React Native ainda está a receber constantes melhorias.

Na Tabela 1 podem ser consultados os resultados finais alcançados.

TABELA I – RESULTADOS DA ANÁLISE DOS CRITÉRIOS NA ESCALA DE LIKERT

Critérios	Soluções			
	Nativas	*React Native*	*Native Script*	*Ionic*
Aprendizagem e qualidade da documentação	2	5	3	4
Custo de Desenvolvimento	2	4	4	4
Emuladores e Depuração	4	5	4	4
Tempo de resposta e velocidade	5	5	4	1
Reconhecimento comercial	5	5	1	2
Reutilização de código e trabalho em equipa	2	5	5	4
Manutenção e Atualizações	2	3	4	4
TOTAL	**22**	**32**	**25**	**23**

V. DISCUSSÃO DOS RESULTADOS

Neste trabalho fornecemos a comparação entre quatro diferentes soluções para programação de aplicações móveis, sendo essencialmente soluções utilizadas por empresas com o objetivo de criarem aplicações para serem disponibilizadas na App Store e Google Play. As frameworks explicadas neste artigo foram avaliadas em sete critérios explicados na seção III.

Já em estudos comparativos realizados entre soluções nativas e híbridas, mesmo com outras frameworks diferentes das analisadas neste artigo, as conclusões vão ao encontro dos resultados apresentados. Entre esses estudos podemos salientar alguns em que as soluções híbridas são referidas como melhores por se evitar a implementação de duas lógicas de código, como em [22], através da migração de uma aplicação Android para híbrida, verificou-se que a performance manteve-se igual dando a possibilidade de distribuição tanto para iOS como para Android, fazendo com que o tempo de desenvolvimento de um produto seja reduzido consideravelmente tal como constatado em [23], após uma análise comparativa entre aplicações nativas e híbridas. Os autores em [10] concluem que as soluções híbridas estão cada vez mais próximas das soluções nativas em velocidade de execução e renderização de conteúdo através de testes em vários ambientes de desenvolvimento, estando de acordo com os resultados apresentados neste estudo, contudo a diferença em velocidade de renderização e execução ainda é muitas vezes percetível em certas frameworks. As soluções híbridas são a melhor escolha para produtos do tipo MVP (Minimum Viable Product), ou seja, a versão mais simples que pode ser lançada e com a quantidade mínima de desenvolvimento [16] e, face ao mercado da tecnologia em constante mudança, grandes marcas já começam a explorar a implementação híbrida tal como o Instagram, Airbnb, Skype ou segundo Singh [2], Amazon ou Linkedin.

Apesar dos resultados entre soluções híbridas e nativas estarem de acordo com alguns estudos já realizados em [2], [10], [16], [22] e [23], existem poucos estudos no que concerne à comparação entre as frameworks de JavaScript, devido a ser um mercado ainda em crescimento e expansão. Por exemplo, o React Native ainda não tem uma versão estável, contudo, nos testes realizados, este demonstra grande vantagem em relação às outras frameworks híbridas. Os resultados finais mostrados na Tabela 1 constata-se que as soluções nativas, apesar de apresentarem maior desempenho e reconhecimento comercial, não evidenciam mais vantagens relevantes devido à grande diferença entre linguagens e sistemas operativos. Em relação às frameworks de JavaScript, o React Native apresenta os melhores resultados, devido ao apoio por parte do Facebook na criação de uma framework eficiente com componentes o mais similar possível aos nativos. Isto explica a inexistência de uma versão estável pelas constantes melhorias que vão sendo aplicadas. O NativeScript apesar de menor reconhecimento e com poucas aplicações de relevo publicadas, apresenta uma velocidade de execução, resposta e teste muito semelhante ao React Native, tendo de melhorar na qualidade da documentação para tornar o desenvolvimento mais fácil e viável. Já o Ionic, apesar de maior reconhecimento do que o NativeScript, apresenta algumas lacunas a nível de velocidade de execução, sendo recomendado apenas para aplicações com pouca complexidade a nível de navegação ou de necessidades reduzidas de sincronização de dados com servidores. Desta forma é possível perceber que o desenvolvimento híbrido em JavaScript apresenta mais vantagens em relação ao nativo, tendo todas as frameworks obtido uma classificação superior. Assim é possível fazer um desenvolvimento de mais alto nível, de mais fácil aprendizagem, com menor custos de desenvolvimento aliado ao bom desempenho. O JavaScript é claramente uma opção para o futuro no desenvolvimento móvel. Apesar disso, não é recomendável para modelos de negócio que implementem muitas integrações com serviços externos de outras empresas, porque normalmente esses serviços apresentam a documentação nas linguagens oficiais de Android e iOS, não dando muitas vezes suporte ao desenvolvimento híbrido. Pelo contrário, é a melhor opção para desenvolvimento de aplicações de forma ágil e incremental, quando não existe a definição exata do produto final. Além disso, com uma equipa dinâmica e multidisciplinar é possível ultrapassar os problemas de forma conjunta, usando uma linguagem única, não tendo que haver especialização como necessário no desenvolvimento móvel nativo.

VI. CONCLUSÕES E TRABALHO FUTURO

O desenvolvimento de aplicações móveis está em enorme evolução e atrai um grande interesse por parte de várias entidades. O desenvolvimento nativo oferece a melhor experiência para o utilizador pelo desenvolvimento ser apenas focado num tipo de sistema operativo e seguindo as diretrizes do mesmo. No entanto, com o crescente mercado das aplicações o desenvolvimento móvel nativo é ineficiente tanto em termos de tempo e custo de desenvolvimento quanto em custo de manutenção.

Para pequenas e médias empresas, a utilização de React Native é a melhor opção de rentabilidade versus qualidade da solução. Por exemplo, uma equipa de desenvolvimento de React Native, pode implementar soluções Web baseadas em React, sendo só necessário mudar os componentes de interface móveis para web, mantendo grande parte da lógica de negócio de um produto. Verifica-se então que a React Native é a melhor solução para ter uma equipa focada numa única linguagem e a desenvolver todo o tipo de soluções front-end. Com este trabalho foi ainda possível concluir que programadores de aplicações Web podem criar aplicações móveis com qualidade e rapidez, não tendo de recorrer a soluções nativas que diferem consideravelmente na forma de desenvolvimento em relação ao JavaScript.

Como trabalho futuro propomos a avaliação de duas abordagens, baseadas na framework que apresentou melhores resultados gerais na análise. Uma, comparando o React Native com outras ferramentas, como o Xamarim baseado em C# ou Beeware em Python, de forma a comparar diferentes frameworks de distintas linguagens de programação. Outra seria estudar qual a repercussão em tempo das alterações para transformar uma aplicação móvel em Web ou vice-versa e qual a quantidade de código reutilizada entre React e React Native, para que pequenas e médias empresas possam perceber até que ponto a mudança de todo o front-end para React pode ser vantajosa.

REFERÊNCIAS

[1] Statista, "Number of smartphone users worldwide from 2014 to 2020," 2017. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed: 27-Dec-2018].

[2] N. Singh, "International Journal on Recent and Innovation Trends in Computing and Communication An comparative analysis of Cordova Mobile Applications V/S Native Mobile Application," pp. 3777–3782.

[3] Statista, "Worldwide internet user penetration from 2014 to 2021," 2017. [Online]. Available: https://www.statista.com/statistics/325706/global-internet-user-penetration/. [Accessed: 27-Dec-2017].

[4] C. Hope, "Computer vs Smartphone," 2017. [Online]. Available: https://www.computerhope.com/issues/ch001398.htm. [Accessed: 27-Dec-2017].

[5] Statista, "App Download and Usage Statistics 2017," 2017. [Online]. Available: http://www.businessofapps.com/data/app-statistics/. [Accessed: 14-Jan-2018].

[6] GitHub, "The State of the Octoverse 2017," 2017. [Online]. Available: https://octoverse.github.com/. [Accessed: 14-Jan-2018].

[7] M. Hafiz, S. Hasan, Z. King, and A. Wirfs-Brock, "Growing a language: An empirical study on how (and why) developers use some recently-introduced and/or recently-evolving JavaScript features," *J. Syst. Softw.*, vol. 121, pp. 191–208, 2016.

[8] StackShare, "Ionic vs React Native vs NativeScript," 2017. [Online]. Available: https://stackshare.io/stackups/ionic-vs-nativescript-vs-react-native. [Accessed: 31-Dec-2017].

[9] Statista, "Global mobile OS market share 2009-2017, by quarter," 2017. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. [Accessed: 28-Dec-2017].

[10] T. Vilcek and T. Jakopec, "Comparative analysis of tools for development of native and hybrid mobile applications," *2017 40th Int. Conv. Inf. Commun. Technol. Electron. Microelectron.*, pp. 1516–1521, 2017.

[11] TomislavCar, "Android development is 30% more expensive than iOS. And we have the numbers to prove it!," 2015. [Online]. Available: https://infinum.co/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios. [Accessed: 31-Dec-2017].

[12] G. Motroc, "Can Kotlin overtake Java for Android development? New report says yes," 2018. [Online]. Available: https://jaxenter.com/can-kotlin-overtake-java-report-138079.html. [Accessed: 15-Jan-2018].

[13] C. González García, J. Pascual-Espada, C. Pelayo G-Bustelo, and J. M. Cueva-Lovelle, "Swift vs. Objective-C: A New Programming Language," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 3, no. 3, p. 74, 2015.

[14] IBM, "Native, web or hybrid mobile-app development", 2012.

[15] A. A. Menegassi and A. T. Endo, "An evaluation of automated tests for hybrid mobile applications," *Proc. 2016 42nd Lat. Am. Comput. Conf. CLEI 2016*, 2017.

[16] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," *Proc. 6th Balk. Conf. Informatics - BCI '13*, p. 213, 2013.

[17] M. Huynh and D. Truong, "Hybrid App Approach: Could it mark the end of native app domination?," *Issues Informing Sci. + Inf. Technol.*, vol. 14, pp. 49–65, 2017.

[18] L. Šmitalová, "NativeScript application for Continuous Improvement of Software Engineer's skills," 2017.

[19] R. Native, "Who's using React Native?," 2017. [Online]. Available: http://facebook.github.io/react-native/showcase.html. [Accessed: 30-Dec-2017].

[20] NativeScript, "Showcases," 2017. [Online]. Available: https://www.nativescript.org/showcases. [Accessed: 30-Dec-2017].

[21] Ionic, "A showcase of the most beautiful apps built with Ionic," 2017. [Online]. Available: http://showcase.ionicframework.com/apps/top. [Accessed: 30-Dec-2017].

[22] H. Ahuja and R. Johari, "Optimization of the issues in the migration from Android Native to Hybrid Application: Case study of Student's Portal application," *Proc. 2014 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2014*, pp. 245–249, 2014.

[23] R. Halidovic and G. Karli, "www.iosrjen.org Cross-Platform Mobile App Development using HTML5 and JavaScript while leveraging the Cloud," *IOSR J. Eng.*, vol. 4, no. 2, pp. 2250–3021, 2014.

Anexo C – Testes implementados aos programadores

React Native Questões

Dados da Amostra

1. Sabe JavaScript?

2. Se sim, anos de experiência na linguagem JavaScript?

3. Anos de experiência em desenvolvimento de Software?

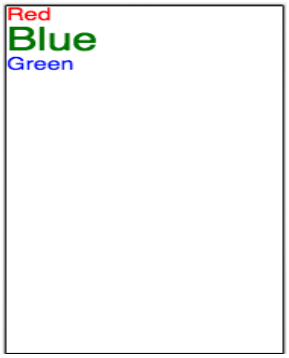
4. Alguma vez desenvolveu alguma aplicação em React Native ou já fez alguma autoaprendizagem sobre React Native?

5. Já desenvolveu aplicações móveis de forma nativa para Android (Java ou Kotlin) ou iOS (Objective C ou Swift), se sim qual e quanto tempo em cada uma?

6. Já desenvolveu aplicações móveis de forma híbrida? Se sim qual a linguagem e o tempo que trabalhou nessa Framework?

1. Criação de vistas

1.1. **Indique** qual o código associado ao seguinte output?



	`` render() { return (<View style={{ flexDirection: 'column' }}> <Text style={{ color: 'red' }}>Red</Text> <Text style={{ color: 'green', fontSize: 30 }}>Blue</Text> <Text style={{ color: 'blue' }}>Green</Text> </View>); } ```
	`` render() { return (<View style={{ flexDirection: 'column' }}> <Text style={{ color: 'red' }}>Red</Text> <Text style={{ color: 'green', fontSize: 30 }}>Blue</Text> <Text style={{ color: 'blue' }}>Green</Text> </View>); } ```
	`` render() { return (<View style={{ flexDirection: 'column' }}> <Text style={{ color: 'red', fontSize: 30 }}>Red</Text> <Text style={{ color: 'blue', fontSize: 30 }}>Blue</Text> <Text style={{ color: 'green', fontSize: 30 }}>Green</Text> </View>); } ```

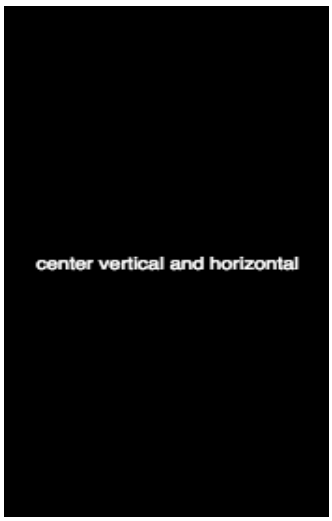
```
render() {
  return (
    <View style={{ flexDirection: 'column' }}>
      <Text style={{ color: 'red' }}>Red</Text>
      <Text style={{ color: 'blue' }}>Blue</Text>
      <Text style={{ color: 'green' }}>Green</Text>
    </View>
  );
}
```

1.2. Para obter o output da seguinte imagem **implemente** o seguint código que está em comentário na imagem abaixo:


```
export default class StyleExample extends Component {
    render() {
        return (
            <View style={styles.container}>
                <Text style={[styles.bigblue, /* --- a) --- */]}>Blue</Text>
                <Text style={styles.bigblue}>Big Blue</Text>
            </View>
        );
    }
}

const styles = StyleSheet.create({
    container: {
        paddingTop: 10,
        paddingLeft: 10,
    },
    bigblue: {
        color: 'blue',
        fontSize: 20,
    },
    greenBold: {
        paddingTop: 20,
        marginTop: 5,
        backgroundColor: 'green',
        fontWeight: 'bold',
    },
});
```

a)

1.3. Complete o código de forma a **gerar** o seguinte output:




```
export default class LotsOfStyles extends Component {
    render() {
        return (
            <View style={/* a) */}>
                <Text style={/* b) */}>center vertical and horizontal</Text>
            </View>
        );
    }
}

const styles = StyleSheet.create({
    container: {
        /* c */
    },
    bigblue: {
        /* d */
    },
});
```

a)

b)

c)

d)

2. Alteração de estado de uma aplicação

2.1. Para mudar o estado de uma variável pertencente ao estado do componente, **selecione** qual a função que deve ser chamada?

1 - setState

2 - state

3 - onPress

4 - this.state

2.2. Tendo em conta os exemplos do documento de tutorial associados a este capítulo, **implemente** uma classe **Contador**, em que tem um estado **total**.

2.3. Tendo em conta os exemplos fornecidos no documento de tutorial associados a este capítulo, na Classe **Contador**, **produza** um render e return associado a uma vista em React Native, e **gere** dois botões, um com o title + e outro com o title -, que chamem uma função **changeTotal** para alterar o estado de total na vista para +1 ou -1. A vista também deve conter uma label em Text com o valor atual do contador.

3. Listas de dados

3.1. O código da seguinte figura mudou as propriedades do array da variável **exampleData** em relação ao render que se mantêm igual ao do documento de tutorial associado a este capítulo. Imaginemos que queremos renderizar na FlatList o nome das pessoas do array **exampleData**. **Indique** qual a linha de código que tem de ser alterada e pelo que substituía?

```
1  import React, { Component } from 'react';
2  import { Text, View, FlatList } from 'react-native';
3
4  export default class ListExample extends Component {
5    constructor(){
6      super();
7      this.state = {
8        total: 0,
9        exampleData: [
10         {
11           key: 'a',
12           personName: 'Pessoa 1'
13         },
14         {
15           key: 'b',
16           personName: 'Pessoa 2'
17         }
18       ]
19     }
20   }
21
22   render() {
23     const {exampleData} = this.state;
24
25     return (
26       <View>
27         <FlatList
28             data={exampleData}
29             keyExtractor={(item) => item.key}
30             renderItem={({item}) => <Text>{item.text}</Text>}
31         />
32       </View>
33     );
34   }
35 }
36
```

3.2. Existe uma lista que renderiza nomes de pessoas tendo por base, os dados dessa mesma lista. Que propriedades teria de ter o array **finalData** obrigatoriamente para o componente FlatList se comportar da forma correta? **Complete** com um objeto que inseria no array **finalData** com essas propriedades para o código ser funcional.

```
1   import React, { Component } from 'react';
2   import { Text, TouchableOpacity, FlatList } from 'react-native';
3
4   export default class ListExample extends Component {
5     constructor(){
6       super();
7       this.state = {
8         total: 0
9         /* finalData: [
10            a)
11        ]
12        */
13      }
14    }
15
16    renderFlatListItems = (personName) => {
17      return (
18        <TouchableOpacity>
19          <Text>{personName}</Text>
20        </TouchableOpacity>
21      )
22    }
23
24    render() {
25      const {finalData} = this.state;
26
27      return (
28        <View>
29          <FlatList
30              data={finalData}
31              keyExtractor={(item) => item.id}
32              renderItem={({item}) => this.renderFlatListItems(item.name)}
33          />
34        </View>
35      );
36    }
37  }
38
```

a)

3.3. Caso no exemplo anterior tivesse um evento que obtinha o clique sobre um item da lista e que chamava uma função **countIncrement** que aumentava a variável total do estado em um. **Produza** o código necessário para fazer uma função que trata o estado do total, e qual o código necessário para obter o evento clique sobre um item de uma FlatList?

4. NetWorking

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    firstParam: 'yourValue',
    secondParam: 'yourOtherValue',
  }),
});
```

4.1. Caso tenha de enviar como Body do POST da imagem anterior apenas e só a propriedade size com o valor de 40, **indique** o que removia e o que inseria?

4.2. Tendo em conta a função **getMoviesFromAPi** com ES8, ou **getMoviesFromApiAsync** com promisses que estão no documento de tutorial, e que na resposta ao pedido recebe sempre um atributo **code. Demonstre** como faria uma verificação caso o code seja igual a 401 retorne a string '**Unauthorized**'

4.3. Tendo em conta o código da seguinte imagem, e sabendo que existe uma função que obtém os filmes de um determinado servidor **getMoviesFromAPi** com ES8, ou **getMoviesFromApiAsync** com promisses, **gere** uma função **getMovies**, que obtém um array de filmes através das funções já existentes e que atualiza o estado de moviesList para a resposta.

```
import React, { Component } from 'react';
import { Text } from 'react-native';

export default class ListOfMovies extends Component {
  constructor(){
    super();
    this.state = {
      moviesList: []
    }
  }

  componentDidMount() {
      this.getMovies(); ///// function getMovies a)
  }

  render() {
    const {movies} = this.state;

    return (
      <View>
        <Text>Number of movies: {movies.length}</Text>
      </View>
    );
  }
}
```

5. Diferenciação entre Plataformas

5.1. Faça a **distinção** entre as duas plataformas móveis (Android e iOS) através de uma condição. Caso a plataforma seja do tipo Android, a variável **value** assume o valor de 'android' senão assume o valor de 'ios'.

```
import React, { Component } from 'react';
import { Text, Platform } from 'react-native';

export default class PlatformDiference extends Component {
  constructor(){
    super();
    this.state = {
      value: 'not exist'
    }
  }

  componentDidMount() {
      /*
          a)
      */
  }

  render() {
    const {value} = this.state;

    return (
      <View>
        <Text>{value}</Text>
      </View>
    );
  }
}
```

5.2. Assuma que tem de fazer uma renderização condicional de <Text></Text>, em que caso seja Android apresenta 'android' e caso seja iOS apresenta 'ios', **demonstre** como faria esse render sabendo que apenas é possível de alteração a linha de código associada ao componente <**Text>{value}</Text>**.

Nota: Pode utilizar condições ternária

5.3. Assuma que existe um ficheiro denominada label.ios.js que apresenta o texto Hello World iOS com o fundo em azul e que existe um ficheiro label.android.js que apresenta o texto Hello World Android com o fundo em verde. Produza todo o componente genérico chamado de GenericLabel, que quando chamado tenha comportamentos diferentes consoante a plataforma em que está a ser executada a aplicação.

Questões finais

1. Quais as principais dificuldades que teve na aprendizagem dos conceitos de React Native?

2. Quais os conceitos que para si foram fáceis de entender no React Native?

3. Que vantagens e desvantagens encontrou no React Native?

4. Após toda a documentação fornecida e respostas dadas, qual o seu à vontade para realizar uma aplicação protótipo com uns pedidos e umas listagens das respostas tendo por base um layout pré-definido? Avalie numa escala de (0 – 100)? Também pode dar uma justificação mais alargada ao valor respondido!

5. Após toda a documentação fornecida e respostas dadas, qual o seu à vontade para realizar uma aplicação para um cliente, recebendo requisitos e implementando (0 – 100)? Também pode dar uma justificação mais alargada ao valor respondido!

NativeScript Questões

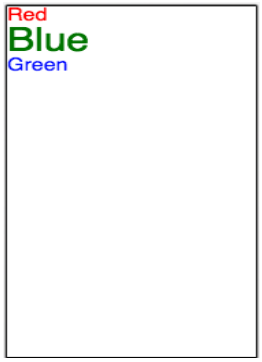
Teste ___

Questões de caraterização

1. Alguma vez desenvolveu alguma aplicação em NativeScript ou já fez alguma autoaprendizagem sobre NativeScript?

1. **Criação de vistas**

1.1. **Indique** qual o código associado ao seguinte output?



	`<Page>` `  <GridLayout rows="3" columns="3" style="background-color: white">` `  <Label row="0" col="0" text="Red" style="color: red" />` `  <Label row="1" col="0" text="Blue" style="color: green; font-size: 30"/>` `  <Label row="2" col="0" text="Green" style="color: blue"/>` `  </GridLayout>` `</Page>`
	`<Page>` `  <GridLayout rows="3" columns="3" style="background-color: white">` `  <Label row="0" col="0" text="Red" style="color: red font-size: 30;" />` `  <Label row="1" col="0" text="Blue" style="color: green; font-size: 30;" />` `  <Label row="2" col="0" text="Green" style="color: blue font-size: 30;" />` `  </GridLayout>` `</Page>`
	`<Page>` `  <GridLayout rows="50, auto, *" columns="50, auto, *" style="background-color: whit` `  <Label row="0" col="0" text="Red" style="color: red" />` `  <Label row="1" col="0" text="Blue" style="color: green; font-size: 30;"/>` `  <Label row="2" col="0" text="Green" style="color: blue"/>` `  </GridLayout>` `</Page>`
	`<Page>` `  <GridLayout rows="50, auto, *" columns="50, auto, *" style="background-color: whit` `  <Label row="0" col="0" text="Red" style="color: red; font-size: 30;" />` `  <Label row="1" col="0" text="Blue" style="color: green; font-size: 30;"/>` `  <Label row="2" col="0" text="Green" style="color: blue; font-size: 30;"/>` `  </GridLayout>` `</Page>`

1.2. Para obter o output da seguinte imagem **implemente** o seguinte código que está em comentário na imagem abaixo:

Código XML

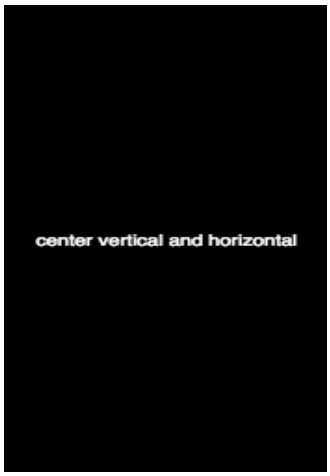
```
<Page>
  <GridLayout rows="0, auto, *" columns="*" class="container">
    <Label row="1" col="0" text="Blue" class="big-big-blue <!-- a) -->" />
    <Label row="2" col="0" text="Big Blue" class="bigblue"/>
  </GridLayout>
</Page>
```

Código CSS

```
.container {
    padding-top: 20;
    padding-left: 10;
}

.big-big-blue {
    color: blue;
    font-size: 20;
}

.green-back {
    padding-top: 20;
    margin-top: 5;
    background-color: green;
    font-weight: bold;
}
```

a)

1.3. Complete o código de forma a **gerar** o seguinte output:



Código XML

```
<Page>
    <GridLayout <!-- a) --> >
        <Label rows="*" columns="auto" text="Center vertical and horizontal aa" class="<!-- b) -->" />
    </GridLayout>
</Page>
```

Código CSS

```
.container {
    <!-- c) -->
}

.big-big-blue {
    <!-- d) -->
}
```

a)

b)

c)

d)

2. Alteração do estado de uma aplicação

2.1. Para mudar o estado de uma variável através de um clique no botão, **selecione** como poderia alterar o valor de uma variável no lado do JavaScript?

1 viewModel.setState
2 viewModel.state
3 viewModel.onClick
4 viewModel.set

2.2. Tendo em conta os exemplos do documento de tutorial associados a este capítulo, **implemente** um viewModel denominado **Contador**, em que tem um estado **total** com o valor 0.

2.3. Tendo em conta os exemplos fornecidos no documento de tutorial associados a este capítulo, no XML, **produza** uma vista que mostra dois botões, um com o title **+** e outro com o title **-**, que chamem no JavaScript uma função **changeTotal** para alterar o valor de total no viewModel e consequentemente na vista para +1 ou -1. A vista também deve conter uma Label com o valor atual do contador.

3. Listas de dados

3.1. O código da seguinte figura mudou as propriedades do array da variável **exampleData** em relação ao código da ListView que se mantêm igual ao do documento de tutorial associado a este capítulo. Imaginemos que queremos renderizar a ListView o nome das pessoas do array **exampleData**. **Indique** qual a **linha de código e ficheiro** que tem de ser alterada e pelo que substituía?

Código XML

```xml
<Page navigatingTo="onNavigatingTo" xmlns="http://schemas.nativescript.org/tns.xsd">
  <ListView
    items="{{ exampleData }}"
    itemTap="onItemTap"
    loaded="{{ onListViewLoaded }}"
    separatorColor="orangered"
    rowHeight="50"
    class="list-group"
    id="listView"
    row="2">
    <ListView.itemTemplate>
      <StackLayout class="list-group-item">
        <Label text="{{ text || 'Downloading...' }}" textWrap="true" />
      </StackLayout>
    </ListView.itemTemplate>
  </ListView>
</Page>
```

Código JavaScript

```javascript
var fromObject = require("tns-core-modules/data/observable").fromObject;

function onNavigatingTo(args) {
  var page = args.object;
  var vm = fromObject({
    exampleData: [
      {
        key: "a",
        personName: "Pessoa 1"
      },
      {
        key: "b",
        personName: "Pessoa 2"
      }
    ]
  });
  page.bindingContext = vm;
}
exports.onNavigatingTo = onNavigatingTo;

function onListViewLoaded(args) {
  var listView = args.object;
}
exports.onListViewLoaded = onListViewLoaded;

function onItemTap(args) {
  var index = args.index;
  console.log(`Second ListView item tap ${index}`);
}
exports.onItemTap = onItemTap;
```

3.2. Existe uma lista que renderiza nomes de pessoas tendo por base, os dados dessa mesma lista. Que propriedades teria de ter o array **finalData** obrigatoriamente para o componente ListView se comportar da forma correta? **Complete** com um objeto que

inseria no array **finalData** que se encontra associado às duas seguintes imagens, com essas propriedades para o código ser funcional.

Código XML

```xml
<Page navigatingTo="onNavigatingTo" xmlns="http://schemas.nativescript.org/tns.xsd">
  <ListView
    items="{{ finalData }}"
    itemTap="onItemTap"
    loaded="{{ onListViewLoaded }}"
    separatorColor="orangered"
    rowHeight="50"
    class="list-group"
    id="listView"
    row="2">
      <ListView.itemTemplate>
        <StackLayout class="list-group-item">
          <Label text="{{ name || 'Downloading...' }}" textWrap="true" />
        </StackLayout>
      </ListView.itemTemplate>
    </ListView>
</Page>
```

Código JavaScript

```javascript
var fromObject = require("tns-core-modules/data/observable").fromObject;

function onNavigatingTo(args) {
  var page = args.object;
  var vm = fromObject({
    /* finalData: [
      a)
    ]
    */
  });
  page.bindingContext = vm;
}
exports.onNavigatingTo = onNavigatingTo;

function onListViewLoaded(args) {
  var listView = args.object;
}
exports.onListViewLoaded = onListViewLoaded;

function onItemTap(args) {
  var index = args.index;
  console.log(`Second ListView item tap ${index}`);
}
exports.onItemTap = onItemTap;
```

a)

3.3. Caso no exemplo anterior tivesse um evento que obtinha o clique sobre um item da lista e que chamava uma função **countIncrement** que aumentava a variável total do estado em um. **Produza** o código necessário para fazer uma função que trata o estado do total, e qual o código necessário para obter o evento clique sobre um item de uma ListView?

4. NetWorking

4.1. Caso tenha de enviar como Body do POST da seguinte imagem apenas e só a propriedade size com o valor de 40, **indique** o que removia e o que inseria?

```
httpModule.request({
    url: "https://movies.org/post",
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    content: JSON.stringify({
        firstParam: "Example 1",
        secondParam: "Example 2"
    })
}).then((response) => {
    const result = response.content.toJSON();
}, (e) => {
});
```

Resposta:

4.2. Tendo em conta o **httpRequest**, que está na seguinte imagem que realiza um get a um determinado url, e que na resposta ao pedido recebe sempre um atributo **test. Demonstre** como faria uma verificação caso **test** fosse igual ao valor 401 que retornasse a string '**Unauthorized**'

```
httpModule.request({
    url: "https://movies.org/get",
    method: "GET"
}).then((response) => {

    var obj = response.toJSON();

}, (e) => {
    console.error(e);
});
```

Resposta:

4.3. Tendo em conta o código das seguintes imagens, e sabendo que existe uma função que obtém os filmes de um determinado servidor denominada por **getMoviesFromAPI**, **gere** uma função denominada por **getMovies**, que obtém um array de filmes através da função **getMoviesFromAPI** já existente e que após receber o array de filmes atualiza o estado de **stateVariable** para a resposta obtida na função **getMoviesFromAPI**.

Código JavaScript

```
var fromObject = require("data/observable").fromObject;
var viewModel;

function onNavigatingTo(args) {
  var page = args.object;

  viewModel = fromObject({
    stateVariable: "state"
  });

  page.bindingContext = viewModel;
}

function exampleFunction() {
  viewModel.set('stateVariable', "otheValue");
}

exports.exampleFunction = exampleFunction;
exports.onNavigatingTo = onNavigatingTo;
```

Resposta:

5. Diferenciação entre Plataformas

5.1. Tendo por base a seguinte imagem, faça a **distinção** entre as duas plataformas móveis (Android e iOS) através de uma condição numa função criada denominada de changePlatform. Caso a plataforma seja do tipo Android, a variável **stateVariable** assume o valor de 'android' senão assume o valor de 'ios'.

Código JavaScript

```
var fromObject = require("data/observable").fromObject;
var viewModel;

function onNavigatingTo(args) {
  var page = args.object;

  viewModel = fromObject({
    stateVariable: "state"
  });

  page.bindingContext = viewModel;
}

function exampleFunction() {
  viewModel.set('stateVariable', "otheValue");
}

exports.exampleFunction = exampleFunction;
exports.onNavigatingTo = onNavigatingTo;
```

Resposta:

5.2. Assuma que tem um ficheiro **main.xml** que necessita de ter uma **Label** em ios e um componente **Button** em Android. **Demonstre** como realizaria a vista de main.xml de forma a cumprir os seguintes requisitos.

Resposta:

5.3. Assuma que tem de criar um componente genérico que permita ter resultados diferentes em Android e iOS. Em iOS é apresentado o texto Hello World iOS com o fundo em azul e em Android é apresentado o texto Hello World Android com o fundo verde. Assumindo que esses dois ficheiros já existem respetivamente com o nome hello.ios.xml e hello.android.xml. **Produza** o código necessário em genericLabel.xml que permita consoante a plataforma em que está a ser executada a aplicação ter os componentes anteriormente descritos para Android e iOS.

Questões finais

1. Quais as principais dificuldades que teve na aprendizagem dos conceitos de NativeScript?

2. Quais os conceitos que para si foram fáceis de entender no NativeScript?

3. Que vantagens e desvantagens encontrou no NativeScript?

4. Após toda a documentação fornecida e respostas dadas, qual o seu à vontade para realizar uma aplicação protótipo com uns pedidos e umas listagens das respostas tendo por base um layout pré-definido? Avalie numa escala de (0 – 100)? Também pode dar uma justificação mais alargada ao valor respondido!

5. Após toda a documentação fornecida e respostas dadas, qual o seu à vontade para realizar uma aplicação para um cliente, recebendo requisitos e implementando (0 – 100)? Também pode dar uma justificação mais alargada ao valor respondido!

Ionic Questões

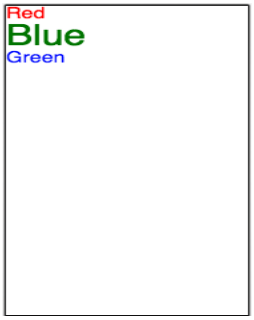
Teste ___

Questões de caraterização

1. Alguma vez desenvolveu alguma aplicação em Ionic ou já fez alguma autoaprendizagem sobre Ionic?

1. **Criação de vistas**

1.1. **<u>Indique</u>** qual o código associado ao seguinte output?



```
<ion-pane>
  <ion-content style="background-color: #FFF;">
    <div style="display: flex; flex-direction: column;">
      <span style="color: red">Red</span>
      <span style="color: green; font-size: 30px;">Blue</span>
      <span style="color: blue">Green</span>
    </div>
  </ion-content>
</ion-pane>
```

1.2. Para obter o output da seguinte imagem **implemente** o seguinte código que está em comentário na imagem abaixo:

Código HMTL

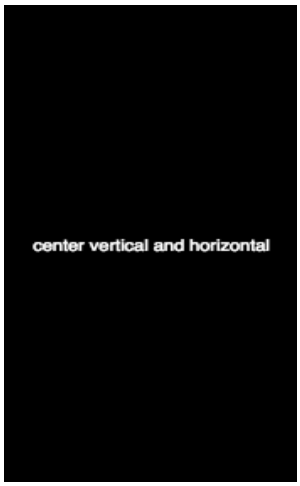
```
<ion-pane>
  <ion-content style="background-color: #FFF;">
    <div class="container">
      <p class="big-big-blue ----- a) -----">Blue</p>
      <p class="big-big-blue">Big Blue</span>
    </div>
  </ion-content>
</ion-pane>
```

Código CSS

```
.container {
    padding-top: 20;
    padding-left: 10;
}

.big-big-blue {
    color: blue;
    font-size: 20;
}

.green-back {
    padding-top: 20;
    margin-top: 5;
    background-color: green;
    font-weight: bold;
}
```

a)

1.3. Complete o código de forma a **gerar** o seguinte output:



Código HMTL

```
<ion-pane>
  <ion-content ------ a) ------->
      <p ----- b) ------>center vertical and horizontal</p>
  </ion-content>
</ion-pane>
```

Código CSS

```
.container {
    ------ c) ----
}

.big-big-blue {
    ------ d) ------
}
```

a)

b)

c)

d)

2. Alteração do estado de uma aplicação

2.1. Para mudar integrar o controlador com a vista para aceder a varáveis ou fazer alterações na mesma através do JavaScript, **selecione** o atributo que permite fazer essa mesma ligação?

5 ng-init
6 ng-state
7 ng-controller
8 ng-set

2.2. Tendo em conta os exemplos do documento de tutorial associados a este capítulo, **implemente** um controlador denominado **indexCounter**, em que tem um estado **total** com o valor 0.

2.3. Tendo em conta os exemplos fornecidos no documento de tutorial associados a este capítulo, **produza** uma vista que mostra dois botões, um com o o texto de + e outro com o de -, que chamem no JavaScript uma função **changeTotal** para alterar o valor de total no viewModel e consequentemente na vista para +1 ou -1. A vista também deve conter um texto com o valor atual do contador. Deve ser **produzido** código para vista como também no controlador de todas as funções.

Nota: Pode usar o controlador anterior

3. Listas de dados

3.1. O código da seguinte figura mudou algumas propriedades e a <ion-list> não está a ser renderizada de forma correta. Imaginemos que queremos renderizar a <ion-list> com o nome das pessoas do array **exampleData** de forma correta. **Indique** qual a **linha de código e ficheiro** que tem de ser alterada e pelo que substituía?

Código Vista

```
38      <ion-pane>
39        <ion-content ng-controller="indexController">
40            <ion-list>
41              <ion-item ng-repeat="item in exampleData">
42                <p>{{ item.text }}</p>
43              </ion-item>
44            </ion-list>
45        </ion-content>
46      </ion-pane>
```

Código Controlador

```
3   .controller('indexController', function($scope, $http) {
4       $scope.exampleData = [
5           {personName: 'Hugo'},
6           {personName: 'Ricardo'}
7       ]
8   })
9
```

Resposta:

3.2. Existe uma lista que renderiza nomes de cores tendo por base, os dados dessa mesma lista. Que propriedades teria de ter o array finalData obrigatoriamente para o componente ListView se comportar da forma correta? Complete com um objeto que inseria no array finalData que se encontra associado às duas seguintes imagens, com essas propriedades para o código ser funcional.

Código Vista

```
<ion-pane>
  <ion-content ng-controller="indexController">
    <ion-list>
      <ion-item ng-repeat="item in finalData">
        <p>{{ item.colorName }}</p>
      </ion-item>
    </ion-list>
  </ion-content>
</ion-pane>
```

Código Controlador

```
.controller('indexController', function($scope) {
    $scope.finalData = [
        /*
            a)
        */
    ]
})
```

a)

3.3. Caso no exemplo anterior (figuras da pergunta 3.2) tivesse um evento que obtinha o clique sobre um item da lista e que chamava uma função **countIncrement** que aumentava uma nova variável **total** do controlador da figura da pergunta 3.2, em + 1 sempre que **qualquer um** dos elementos da lista obtivesse o evento de clique.

Produza o código necessário para fazer uma função que incrementa o valor de total e qual o código necessário para obter o evento clique sobre um item de uma <ion-list>?

Resposta:

4. NetWorking

4.1. Caso tenha de enviar como Body do POST da seguinte imagem apenas e só a propriedade size com o valor de 40, **indique** o que removia e o que inseria?

```
$scope.postExample = function () {

    $http({
        method: 'POST',
        url: 'https://movies.org',
        data: {
            firstParam: 'Example 1',
            secondParam: 'Example 2'
        },
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        }
    })
    .success(function(response) {
        // handle success things
        console.log(response);
    })
    .error(function(data, status, headers, config) {
        // handle error things
    })
```

Resposta:

4.2. Tendo em conta o **request**, que está na seguinte imagem que realiza um get a um determinado url, e que na resposta com sucesso ao pedido recebe um atributo **codeTest** na raiz do JSON da resposta**. Demonstre** como faria uma verificação caso **codeTest** fosse igual ao valor 401 que igualasse uma variável **stateVariable** do mesmo controlador com a string '**Unauthorized**'.

```
$scope.getMovies = function () {

    $http.get('https://movies.org')
    .success(function (data) {

    })
    .error( function (err) {
        console.log(err);
    });


}
```

Resposta:

4.3. Tendo em conta o código das seguintes imagens, e sabendo que existe uma função que obtém os filmes de um determinado servidor chamada de **getMoviesFromAPI**, **gere** uma função denominada por **getMovies** no **indexController**, que obtém um array de filmes através da função **getMoviesFromAPI** já existente e que após receber o array de filmes de **getMoviesFromAPI** atualiza o estado de **moviesList.**

Código Vista

```
<ion-pane>
  <ion-content ng-controller="indexController">
      <ion-list>
          <ion-item ng-repeat="item in moviesList">
            <p>{{ item.title }} | {{ item.releaseYear}}</p>
          </ion-item>
      </ion-list>
      <button class="button button-dark" ng-click="getMovies()">Get Movies From API</button>
  </ion-content>
</ion-pane>
```

Código Controlador

```
.controller('indexController', function($scope, $http) {

    $scope.moviesList = [];

    $scope.getMovies = function () {


    }
})
```

Resposta:

5. Diferenciação entre Plataformas

5.1. Tendo por base a seguinte imagem, faça a **distinção** entre as duas plataformas móveis (Android e iOS) através de uma condição numa função criada denominada de changePlatform. Caso a plataforma seja do tipo Android, a variável **stateVariable** assume o valor de 'android' senão assume o valor de 'ios'.

Nota: Assuma que o modulo PlatformApp do Ionic já se encontra importado.

Código JavaScript

```
.controller('indexController', function($scope, $http) {
    $scope.stateVariable = null;


}
```

Resposta:

5.2. Assuma que tem um ficheiro **main.html** que necessita de ter um simples **Paragrafo** em ios e um **Button** em Android. **Demonstre** como realizaria a vista de main.html de forma a cumprir os seguintes requisitos.

Nota: Assuma que já existe um <ion-pane><ion-content>.

Resposta:

5.3. Assuma que tem de criar um componente genérico que permita ter resultados diferentes em Android e iOS. Em iOS é apresentado o texto Hello World iOS com o fundo em azul e em Android é apresentado o texto Hello World Android com o fundo verde. Assumindo que esses dois ficheiros já existem respetivamente com o nome hello.ios.html e hello.android.html. **Produza** o código necessário para criar um template de uma vista chamada genericLabel.html que permita consoante a plataforma em que está a ser executada a aplicação mostrar os componentes anteriormente descritos de forma a poder inserir mais componentes de vista nesse mesmo template usando os ficheiros html diferentes para cada uma das plataformas.

Nota: Defina todos os ficheiros que achar necessário utilizar tanto .html como .js se necessário. Não deve alterar os ficheiros hello.ios.html nem hello.android.html.

Resposta:

Questões finais

1. Quais as principais dificuldades que teve na aprendizagem dos conceitos de Ionic?

2. Quais os conceitos que para si foram fáceis de entender no Ionic?

3. Que vantagens e desvantagens encontrou no Ionic?

4. Após toda a documentação fornecida e respostas dadas, qual o seu à vontade para realizar uma aplicação protótipo com uns pedidos e umas listagens das respostas tendo por base um layout pré-definido? Avalie numa escala de (0 – 100)? Também pode dar uma justificação mais alargada ao valor respondido!

5. Após toda a documentação fornecida e respostas dadas, qual o seu à vontade para realizar uma aplicação para um cliente, recebendo requisitos e implementando (0 – 100)? Também pode dar uma justificação mais alargada ao valor respondido!

Anexo D – Tutoriais de aprendizagem para os programadores

React Native Regras

São fornecidos dois documentos, um de questões para serem resolvidas após a leitura e interiorização dos conceitos de cada um dos capítulos e o presente que têm todas as instruções de realização do teste e os conceitos associados a cada um dos temas.

A parte de tutorial deste documento é dividida em 5 capítulos relacionados com o desenvolvimento de aplicações móveis em React Native.

Cada capítulo tem uma descrição inicial de apresentação, onde são mostrados os conceitos principais associados ao capítulo para ser possível responder às questões do outro documento de questões.

Para além de toda a explicação dos conceitos são apresentadas outras referências onde é possível saber mais sobre o capítulo em questão.

Não conseguir responder a uma pergunta não inviabiliza que não seja possível responder às perguntas teoricamente mais difíceis do mesmo capítulo.

As descrições de cada capítulo abordam o essencial sobre o conceito, sendo propositalmente vagas principalmente para a resposta de nível mais avançado. Para isso em caso de dúvidas é possível fazer todas as pesquisas necessárias para poder responder às questões.

Todos os testes serão feitos de forma presencial onde serão retiradas algumas informações ao longo da observação da pessoa testada como tempo em cada pergunta, hora de inicio e de término de cada questão bem como tempo dedicado à interiorização dos conceitos de cada um dos capítulos, etc.

Todas as perguntas não são feitas com o intuito de avaliar os conhecimentos de quem as vai realizar, não existindo qualquer tipo de nota no final.

1. Criação de vistas

O React Native utiliza JSX para criação de vistas utilizando JavaScript.

Todos os componentes da vista aceitam uma propriedade chamada de *style* igual ao modo como funciona o CSS na Web excepto as propriedade que estão escritos no formato de CamelCase, como por exemplo é usado *backgroundColor* invés de *background-color*.

O componente utilizado na web é substituído pelo componente <View> nativo do React Native.

Um componente pode especificar o layout de seus filhos usando o algoritmo **flexbox**. O **Flexbox** foi projetado para fornecer um layout consistente em diferentes tamanhos de tela.

Você normalmente usará uma combinação de **flexDirection**, **alignItems** e **justifyContent** para alcançar o layout certo.

À medida que um componente cresce em complexidade, geralmente é mais limpo usar StyleSheet.create para definir vários estilos em um só lugar e podem ser definidos através de um array de vários estilos em conjunto como por exemplo:

```
render() {
    return (
        <View>
            <Text style={styles.bigblue}>Big Blue</Text>
            <Text style={[styles.bigblue, styles.bold]}>Blue</Text>
        </View>
    );
}
```

```
const styles = StyleSheet.create({
    bigblue: {
        color: 'blue',
        fontSize: 30,
    },
    bold: {
        fontWeight: 'bold',
    },
});
```

Para obter mais informação sobre este capítulo pode aceder ao seguinte link:

- https://facebook.github.io/react-native/docs/native-components-ios

2. Alteração de estado de uma aplicação

Existem dois tipos de dados que controlam um componente: **props** e **state**. **props** são definidos pelo pai e são fixados durante todo o tempo de vida de um componente. Para os dados que vão mudar, temos que usar **state**.

Em geral, você deve inicializar **state** no construtor e, em seguida, chamar **setState** quando quiser alterá-lo.

Desta forma é possível aceder ao **state** da classe através da declaração dessa mesma variável dentro do render da vista.

```
class Example extends Component {
  constructor(props) {
    super(props);
    this.state = {
      stateVariable: true
    };
  };
};
```

```
class Example extends Component {
  constructor(props) {
    super(props);
    this.state = {
      stateVariable: true
    };
  };

  render() {
    const { stateVariable } = this.state;

    return (
      <Text>State Value {stateVariable}</Text>
    );
  }
};
```

Neste exemplo é declarada uma variável *stateVariable* com o valor *true* e é acedido na vista ao valor dessa variável, sendo o resultado esperado **State Value First** State.

Para alterar um estado é necessário chamar a função setState como por exemplo:

```
functionExample = () => {
  const { stateVariable } = this.state;
  if (stateVariable == 'First State') {
    this.setState({ stateVariable: 'Stated Changed' })
  }
  else {
    this.setState({ stateVariable: 'First State' })
  }
}

render() {
  const { stateVariable } = this.state;

  return (
    <View>
      <Text>State Value { stateVariable }</Text>
      <Button
        title={'Change State'}
        onPress={() => { this.functionExample() }}
      />
    </View>
  );
}
```

Neste caso foi criada uma função chamada functionExample que é chamada sempre que um botão é pressionado. A functionExample verifica qual o estado da variável e altera o mesmo consoante isso.

Para obter mais informação deste capítulo pode aceder aos seguintes links:

- https://medium.com/@takeoffandroid/props-and-state-in-react-native-c61589bf4cda
- https://facebook.github.io/react-native/docs/state

3. Listas de Dados

Em Março de 2017, o Facebook introduziu o FlatList, que é uma maneira mais fácil e mais eficaz de implementar listas simples e com alto desempenho. Sendo mais fácil de entender no contexto do React do que a ListView. Por isso dessa forma será sempre utilizado o componente FlatList, recomendado para React Native atualmente.

É aconselhado na renderização de FlatList a associação do item renderizado a uma key específica e única de forma a obter todos os ganhos de desempenho da FlatList em relaçao à ListView.

As principais propriadades de uma FlatList são, data onde recebe um array de dados, KeyExtractor que para cada item que renderiza recebe um identificador único do valor e renderItem que é onde é mostrada a vista associado ao item da lista.

```
export default class ListExample extends Component {
  constructor(){
    super();
    this.state = {
      total: 0,
      exampleData: [
        {
          key: 'a',
          text: 'Example 1'
        },
        {
          key: 'b',
          text: 'Example 2'
        }
      ]
    }
  }

  render() {
    const {exampleData} = this.state;

    return (
      <View style={styles.container}>
        <FlatList
            data={exampleData}
            keyExtractor={(item) => item.key}
            renderItem={({item}) => <Text>{item.text}</Text>}
        />
      </View>
    );
  }
}
```

No exemplo de código da figura anterior é possivel ver um exemplo de um componente do tipo FlatList, em que recebe um array, inicializado no estado do componente, e por cada iteração de renderização do array associa um identificador único ao KeyExtractor e na propriedade renderItem mostra a vista associada a esse valor.

Para obter mais informação deste capítulo pode aceder ao seguinte link:

- https://facebook.github.io/react-native/docs/flatlist

4. NetWorking

Muitas aplicações para dispositivos móveis carregam dados através de um URL remoto que tem todos esses dados associados ao utilizador ou a outra circunstância, de forma a ter as aplicações sempre dinâmicas.

Para isso poder ser necessário fazer pedidos através de uma API REST.

O React Native usa o fetch para ir buscar conteúdo arbitário a um certo URL:

```
fetch('https://mywebsite.com/mydata.json');
```

Normalmente o objetivo passa por fazer o pedido para posteriormente utilizar a resposta para alguma coisa. Dessa forma o tratamento da resposta deve ser feita de forma assíncrona. Como por exemplo, na imagem da esquerda com promisses e na da direita com async/await:

```
function getMoviesFromApiAsync() {
  return fetch('https://facebook.github.io/react-native/movies.json')
    .then((response) => response.json())
    .then((responseJson) => {
      return responseJson.movies;
    })
    .catch((error) => {
      console.error(error);
    });
}
```

```
async function getMoviesFromApi() {
  try {
    let response = await fetch(
      'https://facebook.github.io/react-native/movies.json'
    );
    let responseJson = await response.json();
    return responseJson.movies;
  } catch (error) {
    console.error(error);
  }
}
```

Nunca esquecer de que os erros devem ser tratados, senão caso contrário eles serão ignorados no fetch.

Para fazer POST, ou enviar dados na query é necessário configurar um objecto no segundo argumento do fetch com todos esses dados:

Por exemplo, num POST, em que são enviados o **firstParam** e **secondParam** para o mesmo URL do exemplos anteriores.

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    firstParam: 'yourValue',
    secondParam: 'yourOtherValue',
  }),
});
```

Para obter mais informação deste capítulo pode aceder aos seguintes links:

- https://medium.com/@yoniweisbrod/interacting-with-apis-using-react-native-fetch-9733f28566bbhttps://facebook.github.io/react-native/docs/network
- https://facebook.github.io/react-native/docs/network

5. Diferenciação entre Plataformas

Ao criar uma aplicação em React Native pode existir a necessidade de implementar código separado para diferentes plataformas, ou utilizar mesmo código nativo de uma das plataformas para uma determinada funcionalidade.

O React Native fornece um módulo Platform que permite fazer toda essa distinção, sendo apenas necessário obter a valor de Platform.OS e comparar com 'ios' ou 'android' consoante a plataforma que se queira implementar uma lógica diferente.

Também existe a possibilidade de criar um componente com um certo nome, e aplicar diferentes componentes tanto para Android como para iOS como na figura seguinte mostra:

```
const Dialog = Platform.select({
   ios: () => require('DialogIos'),
   android: () => require('DialogAndroid'),
 })();

 <Dialog />;
```

É possível desta forma ter diferentes ficheiros com a extensão .android.js ou .ios.js que permitem fazer essa diferenciação aquando da importação.

```
BigButton.ios.js
BigButton.android.js
```

Para obter mais informações deste capítulo basta aceder ao seguinte link:

- https://facebook.github.io/react-native/docs/platform-specific-code
- https://medium.com/maestral-solutions/react-native-platform-specific-code-e217db5778f

NativeScript Regras

São fornecidos dois documentos, um de questões para serem resolvidas após a leitura e interiorização dos conceitos de cada um dos capítulos e o presente que tem todas as instruções de realização do teste e os conceitos associados a cada um dos temas.

A parte de tutorial deste documento é dividida em 5 capítulos relacionados com o desenvolvimento de aplicações móveis em NativeScript.

Cada capítulo tem uma descrição inicial de apresentação, onde são mostrados os conceitos principais associados ao capítulo para ser possível responder às questões do outro documento de questões.

Para além de toda a explicação dos conceitos são apresentadas outras referências onde é possível saber mais sobre o capítulo em questão.

Não conseguir responder a uma pergunta não inviabiliza que não seja possível responder às perguntas subsequentes do mesmo capítulo.

As descrições de cada capítulo abordam o essencial sobre o conceito, sendo em parte vagas, principalmente para a resposta de nível mais avançado. Para isso, em caso de dúvidas, é possível fazer todas as pesquisas necessárias para poder responder às questões.

Sempre que numa questão sentir necessidade de aceder ao link fornecido ou a documentação externa à fornecida deve registar junto à questão e se possível qual o link.

Todos os testes serão feitos de forma presencial sendo registadas algumas informações ao longo da observação sobre a realização do teste

Todas as perguntas não são feitas com o intuito de avaliar os conhecimentos de quem as vai realizar, não existindo qualquer tipo de classificação no final, mas antes com a intenção de avaliar este tipo de abordagem à aprendizagem do NativeScript.

1. Criação de vistas

O NativeScript na criação de vistas funciona da mesma forma como funciona o desenvolvimento web com CSS.

Os estilos de CSS podem ser definidos em três formas diferentes:

CSS para toda a aplicação: aplica-se em todas as páginas da aplicação.

CSS especifico de uma página: aplica-se às exibições de interface relacionadas apenas com a página associada.

CSS inline: aplica-se diretamente da vista do utilizador.

Umas das formas mais utilizadas para criação de vistas para ficarem com comportamentos nativos é através de GridLayouts que organiza os elementos filhos em uma estrutura de tabela de linhas e colunas. Uma célula posteriormente pode conter vários elementos filhos, podendo sobrepor-se uns aos outros.

O GridLayout tem uma coluna e uma linha com valores por defeito, ou seja para adicionar linhas e colunas é necessário adicionar esse atributo no XML correspondente.

Alguns atributos são:

columns: uma valor em string separado por ; que tem três atributos: number, auto ou *. O atributo number indica uma largura de coluna absoluta, auto torna a coluna tão larga quanto o filho e * faz com que a coluna ocupe todo o espaço horizontal disponível.

rows: Aplica-se com os mesmo atributos da colunas.

O código da imagem seguinte demonstra como posicionar componentes na vista através de GridLayout. Dando o seguinte output da imagem esquematizada:

```
1
2   <Page xmlns="http://schemas.nativescript.org/tns.xsd">
3     <GridLayout columns="50, auto, *" rows="50, auto, *" width="210" height="210" backgroundColor="lightgray" >
4       <Label text="Label 1" row="0" col="0" backgroundColor="red"/>
5       <Label text="Label 2" row="0" col="1" colSpan="2" backgroundColor="green"/>
6       <Label text="Label 3" row="1" col="0" rowSpan="2" backgroundColor="blue"/>
7       <Label text="Label 4" row="1" col="1" backgroundColor="yellow"/>
8       <Label text="Label 5" row="1" col="2" backgroundColor="orange"/>
9       <Label text="Label 6" row="2" col="1" backgroundColor="pink"/>
10      <Label text="Label 7" row="2" col="2" backgroundColor="purple"/>
11    </GridLayout>
12  </Page>
13
```

Para ter acesso a todas os atributos que podem ser utilizados através do GridLayout basta aceder ao seguinte link:

- https://docs.nativescript.org/api-reference/classes/_ui_layouts_grid_layout_.gridlayout

2. Alteração do estado de uma aplicação

O MVVM (Model-View-ViewModel) é o padrão base no qual a estrutura do NativeScript é construído. O MVVM facilita a separação do desenvolvimento da interface gráfica do utilizador do desenvolvimento de negócio ou lógica de back-end (modelo de dados).

Para criação desse mesmo objeto de dados o NativeScript fornece o módulo Observable que permite controlar a alteração de dados entre o JavaScript e a vista em XML.

Para criar um objeto do tipo Observable é usado o método fromObject que cria uma instância Observable onde são definidas as suas propriedades de acordo com o objeto JavaScript. O exemplo seguinte mostra como criar uma viewModel do tipo Observable.

```
const newViewModel = fromObject({ "myColor": "Lightgray" });
```

No exemplo seguinte é possível ver como declarar um viewModel com estados de variáveis dentro do mesmo. Assim no XML é possível aceder aos dados do viewModel associado.

No exemplo seguinte foi criada uma variável denominada stateVariable sendo o valor na mesma state declarado no JavaScript.

Para alterar o estado de uma variável através do XML, deve ser criada uma função auxiliar que altera o valor do viewModel para o comportamento esperado por essa mesmo função.

Código XML

```
<Page navigatingTo="onNavigatingTo" xmlns="http://schemas.nativescript.org/tns.xsd">
  <GridLayout rows="*">
    <GridLayout columns="auto, *">
      <Label text="{{ stateVariable }}" col="0" />
      <Button text="Change State" tap="{{ exampleFunction }}" />
    </GridLayout>
  </GridLayout>
</Page>
```

Código JavaScript

```
var fromObject = require("data/observable").fromObject;
var viewModel;

function onNavigatingTo(args) {
  var page = args.object;

  viewModel = fromObject({
    stateVariable: "state"
  });

  page.bindingContext = viewModel;
}

function exampleFunction() {
  viewModel.set('stateVariable', "otheValue");
}

exports.exampleFunction = exampleFunction;
exports.onNavigatingTo = onNavigatingTo;
```

Neste exemplo foi criada uma função chamada **functionExample** que é chamada sempre que um botão é pressionado e que altera o estado de **stateVariable** para **otheValue**.

Para obter mais informação deste capítulo pode aceder aos seguintes links:

- https://docs.nativescript.org/ns-framework-modules/observable - observable
- https://discourse.nativescript.org/t/how-to-call-function-of-view-model/5471

3. Listas de Dados

Usar um controlo de ListView requer uma atenção especial devido à complexidade de implementação com modelos de itens personalizados, desta forma o NativeScript fornece um módulo de interface utilizador denominado por listView que simplifica a maneira como o ListView nativo é usado.

Na utilização da ListView é aconselhado a associação de alguns atributos importantes como items que o atributo que recebe o array de dado, e ListView.itemTemplate que permite ter os dados de cada item da ListView.

Código XML

```
<Page navigatingTo="onNavigatingTo" xmlns="http://schemas.nativescript.org/tns.xsd">
  <ListView
    items="{{ exampleData }}"
    itemTap="onItemTap"
    loaded="{{ onListViewLoaded }}"
    separatorColor="orangered"
    rowHeight="50"
    class="list-group"
    id="listView"
    row="2">
      <ListView.itemTemplate>
        <StackLayout class="list-group-item">
          <Label text="{{ text || 'Downloading...' }}" textWrap="true" />
        </StackLayout>
      </ListView.itemTemplate>
    </ListView>
</Page>
```

Código JavaScript

```
const fromObject = require("tns-core-modules/data/observable").fromObject;

function onNavigatingTo(args) {
  const page = args.object;
  const vm = fromObject({
    // Setting the listview binding source
    exampleData: [
      { text: "Example 1" },
      { text: "Example 2" }
    ]
  });
  page.bindingContext = vm;
}
exports.onNavigatingTo = onNavigatingTo;

function onListViewLoaded(args) {
  const listView = args.object;
}
exports.onListViewLoaded = onListViewLoaded;

function onItemTap(args) {
  const index = args.index;
  console.log(`Second ListView item tap ${index}`);
}
exports.onItemTap = onItemTap;
```

No exemplo de código da figura anterior é possível ver um exemplo de um componente do tipo ListView que recebe um array inicializado no JavaScript que por cada item mostra uma label texto que está associada ao valor do array.

Para obter mais informação deste capítulo pode aceder ao seguinte link:

- https://docs.nativescript.org/ui/ns-ui-widgets/list-view

4.NetWorking

Muitas aplicações para dispositivos móveis carregam dados através de um URL remoto que tem todos esses dados associados ao utilizador ou a outra circunstância, de forma a ter as aplicações sempre dinâmicas.

Para isso pode ser necessário fazer pedidos através de uma API REST.

O NativeScript fornece o módulo http que permite ir buscar conteúdo a um determinado URL solicitado.

```
const httpModule = require("http");
```

Normalmente o objetivo passa por fazer o pedido para posteriormente utilizar a resposta para alguma coisa. Dessa forma o tratamento da resposta deve ser feita de forma assíncrona como o exemplo seguinte mostra com a ajuda do módulo http do NativeScript.

```
httpModule.request({
    url: "https://movies.org/get",
    method: "GET"
}).then((response) => {

    var obj = response.toJSON();

}, (e) => {
    console.error(e);
});
```

O exemplo seguinte mostra um request ao URL movies.org do tipo GET em que a resposta é obtida e transformada para JSON.

Para fazer um POST em NativeScript, ou enviar dados na query, é necessário configurar um objeto denominado de content em que são colocados todos os dados a enviar no método de POST.

```
httpModule.request({
    url: "https://movies.org/post",
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    content: JSON.stringify({
        firstParam: "Example 1",
        secondParam: "Example 2"
    })
}).then((response) => {
    const result = response.content.toJSON();
}, (e) => {
});
```

Para obter mais informações deste capítulo pode aceder aos seguintes links:

- https://docs.nativescript.org/ns-framework-modules/http

5.Diferenciação entre Plataformas

Ao criar uma aplicação em NativeScript pode existir a necessidade de implementar código separado para diferentes plataformas ou utilizar mesmo código nativo de uma das plataformas para uma determinada funcionalidade.

O NativeScript fornece um módulo Platform que permite fazer toda essa distinção, sendo apenas necessário importar do módulo platform do nativeScript a função isAndroid e isIos que retorna true ou false para cada uma das condições da função.

```
const platformModule = require("tns-core-modules/platform");
```

O NativeScript por defeito não tem um módulo especifico para criar um componente que tem comportamentos diferentes consoante a plataforma.

Contudo existem possibilidades de fazer isso, como diferenciar a interface de utilizador com blocos de marcação para cada plataforma.

Como mostra a imagem seguinte:

```
<Page>
    <ios>
        <Label .... />
    </ios>
    <android>
        <Label .... />
    </android>
</Page>
```

Ao usar as diferentes tags android e ios permite ao compilador que incluía ou remova blocos de marcação em XML consoante a plataforma do dispositivo.

Para usar estilos consoante o tipo de plataforma basta ter o nome de ficheiro de estilos igual ao ficheiro XML com a extensão .android ou .ios como demonstra o exemplo:

```
<Page>
    <Label class="color-me" text="Hello World!">
</Page>

.ios .color-me { color: blue; }
.android .color-me { color: green; }
```

Para obter mais informações deste capítulo basta aceder ao seguinte link:

- https://docs.nativescript.org/api-reference/modules/_platform_

Ionic Regras

São fornecidos dois documentos, um de questões para serem resolvidas após a leitura e interiorização dos conceitos de cada um dos capítulos e o presente que tem todas as instruções de realização do teste e os conceitos associados a cada um dos temas.

A parte de tutorial deste documento é dividida em 5 capítulos relacionados com o desenvolvimento de aplicações móveis em Ionic.

Cada capítulo tem uma descrição inicial de apresentação, onde são mostrados os conceitos principais associados ao capítulo para ser possível responder às questões do outro documento de questões.

Para além de toda a explicação dos conceitos são apresentadas outras referências onde é possível saber mais sobre o capítulo em questão.

Não conseguir responder a uma pergunta não inviabiliza que não seja possível responder às perguntas subsequentes do mesmo capítulo.

As descrições de cada capítulo abordam o essencial sobre o conceito, sendo em parte vagas, principalmente para a resposta de nível mais avançado. Para isso, em caso de dúvidas, é possível fazer todas as pesquisas necessárias para poder responder às questões.

Sempre que numa questão sentir necessidade de aceder ao link fornecido ou a documentação externa à fornecida deve registar junto à questão e se possível qual o link.

Todos os testes serão feitos de forma presencial sendo registadas algumas informações ao longo da observação sobre a realização do teste

Todas as perguntas não são feitas com o intuito de avaliar os conhecimentos de quem as vai realizar, não existindo qualquer tipo de classificação no final, mas antes com a intenção de avaliar este tipo de abordagem à aprendizagem de Ionic.

1.Criação de vistas

O Ionic é uma das frameworks de desenvolvimento móvel com comportamento mais idêntico ao desenvolvimento web, por ter uma webview intermédia que faz a transformação do código para as plataformas móveis.

Desta forma na criação de vistas funciona exatamente como o desenvolvimento web com CSS.

Os estilos de CSS podem ser definidos em três formas diferentes:

CSS para toda a aplicação: aplica-se em todas as páginas da aplicação.

CSS especifico de uma página: aplica-se às exibições de interface relacionadas apenas com a página associada.

CSS inline: aplica-se diretamente da vista do utilizador.

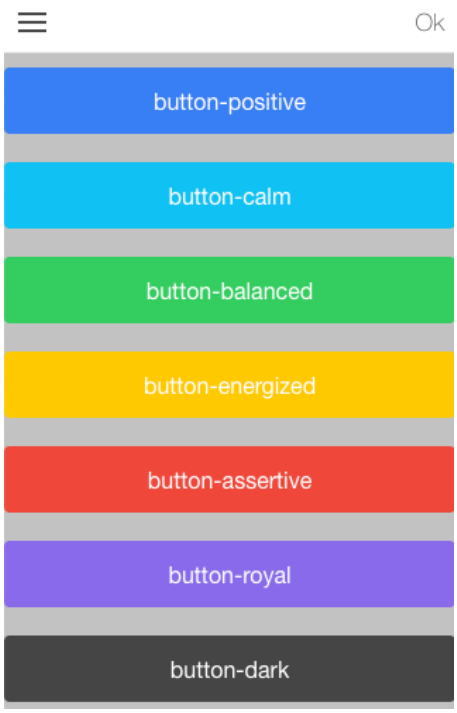
A Framework Ionic é composta por alguns atributos que facilitam a passagem para código nativo dos componentes como por exemplo <ion-pane> equivalente a um container principal no html tipo <body>, <ion-header-bar> que permite a criação de header ou <ion-content> que é onde aparece o conteúdo principal geral da aplicação.

Dentro do <ion-content> é possível utilizar html e css normal como se utilizada no desenvolvimento web.

De seguida enconta-se um exemplo de código com a criação de um header e de botões com recurso a html + css.

Sendo o output do código da imagem o seguinte:

```
<ion-pane>
  <ion-header-bar class="bar-stable">
      <div class="bar bar-header">
          <button class="button button-clear icon ion-navicon"></button>
          <div class="title">Header</div>
          <button class="button button-clear">Ok</button>
        </div>
  </ion-header-bar>
  <ion-content style="background-color: #c2c2c2;">
    <div style="display: flex; flex-direction: column;">

        <button class="button button-block button-positive">
          button-positive
        </button>

        <button class="button button-block  button-calm">
          button-calm
        </button>

        <button class="button button-block  button-balanced">
          button-balanced
        </button>

        <button class="button button-block  button-energized">
          button-energized
        </button>

        <button class="button button-block  button-assertive">
          button-assertive
        </button>

        <button class="button button-block  button-royal">
          button-royal
        </button>

        <button class="button button-block  button-dark">
          button-dark
        </button>
      </div>
  </ion-content>
</ion-pane>
```



Para ter acesso a todas os atributos que podem ser utilizados através do GridLayout basta aceder ao seguinte link:

- https://ionicframework.com/docs/v1/components

2.Alteração do estado de uma aplicação

O MVC é o padrão na qual a estrutura é contruída a arquitetura do Ionic, e baseado em AngularJS. O MVC facilita a separação do desenvolvimento da interface gráfica (V) do utilizador do desenvolvimento de lógica de negócio (C) e com o modelo de dados (M).

Desta forma apenas é necessário utilizar o atributo **ng-controller** associando todos os filhos da vista o controlador declarado.

Na imagem seguinte é possível verificar a associação ao controlador **indexController**, sendo que desta forma todo o conteúdo dentro de **<ion-content>**, pode trabalhar diretamente com esse mesmo controlador declarado.

```
<ion-pane>
  <ion-content ng-controller="indexController">
    <div>
      <button class="button"></button>
    </div>
  </ion-content>
</ion-pane>
```

No exemplo seguinte é possível verificar como declarar um controlador chamado **indexController** com uma variável chamada **stateVariable** que pode ser utilizada na vista.

```
.controller('indexController', function($scope) {
    $scope.stateVariable = 'Example 1';
})
```

Para alterar o estado da variável através da vista, deve ser criada uma função no controlador que altera o valor de **stateVariable** para o valor pretendido, que posteriormente só necessita de ser chamada na vista através de no caso de um botão com **ng-click**.

Código da vista

```
<ion-pane>
  <ion-content ng-controller="indexController">
    <div>
      <button class="button" ng-click="changeState()">Click Me!</button>
      <span>{{ stateVariable }}</span>
    </div>

  </ion-content>
</ion-pane>
```

Código Controlador

```
.controller('indexController', function($scope) {
    $scope.stateVariable = 'Example 1';

    $scope.changeState = function () {
        $scope.stateVariable = 'Change Example 1';
    }
})
```

Nas imagens anteriores foi criada um função chamada **changeState** que é chamada sempre que o botão é pressionado e que altera o valor da variável **stateVariable** para **'Change Example 1'.**

Para obter mais informação deste capítulo pode aceder aos seguintes links:

- http://mcgivery.com/controllers-ionicangular/

3. Listas de Dados

Para listar itens de forma iterativa o Ionic utiliza o controlo <ion-list>, que simplica a forma de como utilizar uma lista de dados numa aplicação móvel.

Na utilização de uma <ion-list> em Ionic está implícita a utilização de <ion-item> que vai representar cada um dos itens da lista.

Nas seguintes figuras é possível ver um exemplo de um componente do tipo <ion-list> que recebe um array inicializado no JavaScript que por cada item mostra uma label com o valor da variável text associada a cada objeto do array.

Código da Vista

```
<ion-pane>
  <ion-content ng-controller="indexController">
      <ion-list>
        <ion-item ng-repeat="item in exampleData">
          <p>{{ item.text }}</p>
        </ion-item>
      </ion-list>
  </ion-content>
</ion-pane>
```

Código do Controlador

```
.controller('indexController', function($scope) {
    $scope.exampleData = [
        { text: 'Example 1' },
        { text: 'Example 2' },
    ]
})
```

Para obter mais informação deste capítulo pode aceder ao seguinte link:

- https://ionicframework.com/docs/v1/guide/building.html

4.NetWorking

Muitas aplicações para dispositivos móveis carregam dados através de um URL remoto que tem todos esses dados associados ao utilizador ou a outra circunstância, de forma a ter as aplicações sempre dinâmicas.

Para isso pode ser necessário fazer pedidos através de uma API REST.

O AngularJS tem o módulo http que permite ir buscar conteúdo a um determinado URL solicitado, bastando passar o mesmo na função de declaração do controlador.

```
.controller('indexController', function($scope, $http) {
```

Normalmente o objectivo passa por fazer o pedido para posteriormente utilizar a resposta para alguma coisa. Dessa forma o tratamento da resposta deve ser feita de forma assíncrona como o exemplo seguinte mostra com a ajuda do módulo **$http** e de uma função de **callback** que aguarda o resultado da função.

```
$http.get('https://movies.org')
.success(function (data) {
    console.log(data);
})
.error( function (err) {
    console.log(err);
});
```

O exemplo seguinte mostra um request ao URL movies.org do tipo GET em que a resposta é obtida de forma assíncrona.

Para fazer um POST em Ionic ou enviar dados na query, é necessário configurar o objecto do request como mostra o exemplo da seguinte figura.

```
$http({
    method: 'POST',
    url: 'https://movies.org',
    data: { id: 25 },
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
    }
})
.success(function(response) {
    // handle success things
    console.log(response);
})
.error(function(data, status, headers, config) {
    // handle error things
})
```

Para obter mais informações deste capítulo pode aceder aos seguintes links:

- https://docs.angularjs.org/api/ng/service/$http

5. Diferenciação entre Plataformas

Ao criar uma aplicação em Ionic pode existir a necessidade de implementar código separado para diferentes plataformas ou utilizar mesmo código nativo de uma das plataformas para uma determinada funcionalidade.

O Ionic fornece um módulo **PlatformApp** que permite fazer toda essa distinção e obter mesmo algumas informações sobre o dispositivo, sendo apenas necessário incluir o módulo **PlatformApp** no módulo do controlador do Angular.

Após isso é possível aceder a todas as funcionalidades do módulo através de ionic.Platform. Para o caso de perceber qual a plataforma basta usar o atributo .platform a seguir ao ionic.Platform, sendo retornado a string em lowerCase com o nome da plataforma do dispositivo que está a executar a aplicação.

O Ionic por defeito não tem um módulo especifico para criar um componente que tem comportamentos diferentes consoante a plataforma, contudo é possível criar nomes de ficheiros iguais mas com a extensão .android.js e .ios.js que permite a criação de lógica de negócio diferente ou .android.html e .ios.html que permite a declaração de componentes diferentes ou de .android.css e .ios.css que permite diferentes estilos consoante o tipo de plataforma. Aliando as potencialidades do AngularJS à possibilidade de customização de diferentes ficheiros através da sua extensão é possível diferenciar todas as plataformas em todos os pontos da aplicação.

Para obter mais informações deste capítulo basta aceder ao seguinte link:

- https://ionicframework.com/docs/v1/api/utility/ionic.Platform

Anexo E – Artigo Análise de frameworks móveis JavaScript

Análise de aprendizagem de frameworks móveis JavaScript

Learning analysis of mobile JavaScript frameworks

Hugo Brito, Álvaro Santos, Jorge Bernardino, Anabela Gomes
Coimbra Polytechnic - ISEC
Coimbra, Portugal
a21220118@isec.pt, ans@isec.pt, jorge@isec.pt, anabela@isec.pt

Resumo — **O JavaScript é atualmente uma linguagem popular principalmente na criação de código do lado do cliente tanto para o desenvolvimento *web* como, cada vez mais, para o desenvolvimento de aplicações móveis. Neste artigo é efetuada uma análise da aprendizagem de programadores em *frameworks* de desenvolvimento de aplicações móveis JavaScript, nomeadamente React Native, NativeScript e Ionic. Esta análise foi realizada com base numa taxonomia adaptada à de Bloom, criada neste trabalho para a avaliação de programadores. Para esta taxonomia foram selecionados critérios de avaliação que permitissem uma análise dos aspetos mais relevantes no processo de aprendizagem, de uma nova *framework* de desenvolvimento móvel. Os resultados mostram que para os programadores analisados o NativeScript apresentou-se como a *framework* mais difícil de aprender e com *feedback* mais negativo, ao contrário do Ionic e do React Native que apresentaram resultados globais idênticos, contudo com ligeiro ascendente nos resultados médios globais para React Native em relação ao Ionic.**

Palavras Chave – JavaScript, React Native, NativeScript, Ionic.

Abstract — **JavaScript is currently a popular language mainly in the creation of client-side code for both web development and, more and more, for the mobile applications development. In this paper, we analyze learning for programmers in frameworks for the development of mobile JavaScript applications, namely React Native, NativeScript and Ionic. This analysis was carried out based on a taxonomy adapted from Bloom, created in this work for the evaluation of programmers, with criteria initially selected for evaluation and with the application of questions based on the taxonomy in order to obtain the most relevant analysis criteria. The results show that NativeScript was the most difficult to learn framework with the most negative feedback, unlike Ionic and React Native, which presented identical global results, but with a slight upward trend in global mean results for React Native in relation to Ionic.**

Keywords - JavaScript, React Native, NativeScript, Ionic.

I. INTRODUÇÃO

Atualmente, os dispositivos móveis são cada vez mais usados [1] e as aplicações móveis têm cada vez um papel mais importante no dia a dia de qualquer utilizador. Esta situação leva a que, nos últimos tempos, tenham aparecido cada vez mais soluções para fazer o desenvolvimento das aplicações móveis. O JavaScript é uma linguagem bastante popular, sendo a linguagem com mais repositórios associados no GitHub [2]. Com esta popularidade surgiram também *frameworks* de desenvolvimento móvel em JavaScript que permitem realizar aplicações tanto para Android como para iOS.

Recentemente foi realizada uma análise comparativa em [3], entre as soluções de desenvolvimento móvel nativo e as híbridas baseadas em JavaScript, nomeadamente React Native, NativeScript e Ionic. Esta análise foi baseada em sete critérios considerados mais relevantes para o desenvolvimento de aplicações móveis, concluindo-se que o React Native se apresentava como a solução com mais vantagens para desenvolvimento de aplicações móveis [1].

Neste trabalho, após os testes preliminares mencionados, foi necessário fazer uma validação dos resultados em termos empíricos, aplicando testes práticos a programadores, de forma a entender se o *feedback*, tempos de desenvolvimento e resultado desse desenvolvimento ia ao encontro dos resultados da análise comparativa previamente realizada. Isto porque, apesar de a avaliação comparativa ter sido feita com rigor, podem surgir aspetos não considerados ou factos novos oriundos da aprendizagem associada a cada uma das *frameworks*. Por exemplo, o problema de os programadores não se adaptarem a uma certa *framework*, por a sintaxe se desviar demasiado da linguagem original pura que não tem recurso a *frameworks*, por as arquiteturas não serem bem interpretadas ou pela dificuldade inerente a cada *framework*. De forma a dar continuidade à analise comparativa realizada [3], foram utilizadas as mesmas frameworks de JavaScript dessa mesma análise (React Native, Ionic e NativeScript).

Assim, estes testes foram planeados tendo em consideração profissionais com conhecimentos em linguagem JavaScript e sem conhecimento nas plataformas testadas. Por um lado, por a empresa onde foi aplicado o estudo ter todo o seu desenvolvimento, exceto o móvel, assente em JavaScript, e por outro porque, como outros estudos referem, o desenvolvimento móvel através de desenvolvimento individual nativo, Java (Android) e Swift (iOS), apenas é vantajoso para empresas estabelecidas e em que exista um produto bem definido e sem alterações constantes [3]–[5].

Este artigo apresenta o estudo referido, estando estruturado da seguinte forma. Na seção II é apresentada a metodologia de avaliação prática apresentada. Na seção III é apresentada a

taxonomia educacional utilizada como base para os testes. Na seção IV é apresentada uma adaptação feita à taxonomia original. Na seção V, são apresentados os itens de avaliação e os critérios recolhidos. Na seção VI é feita uma caraterização da amostra de programadores. Na seção VII, é apresentada a forma como foram validados os resultados. Na seção VIII é feita uma discussão sobre os resultados comparando as *frameworks* consideradas de acordo com os vários critérios analisados. Finalmente, a seção IX apresenta as conclusões.

II. METODOLOGIA

Tal como referido, os testes foram planeados tendo em mente profissionais em JavaScript, de forma a poder incluir algumas questões com alguma complexidade sem que a linguagem de implementação constituísse um entrave para o resultado final dos testes. Este cuidado permite fazer com que as particularidades associadas a cada uma das *frameworks* sejam o foco da avaliação.

Desta forma, foram criados 3 testes diferentes para cada uma das *frameworks*: React Native, Ionic e NativeScript com 15 questões divididas em 5 capítulos, que incidem sobre alguns dos principais conceitos considerados importantes no desenvolvimento de aplicações em *cross-platform* nomeadamente:

- **Capítulo 1** - Desenvolvimento de vistas: para ser possível desenvolver uma aplicação móvel, a interface de utilizador é quase sempre obrigatória, sendo o que permite uma melhor aceitação da aplicação por parte dos utilizadores;
- **Capítulo 2** - Comunicação entre vistas e lógica de negócio: para existir algum tipo de interação entre a vista e a lógica de negócio ou funcionalidades da aplicação tem de haver uma comunicação entre a interface e a lógica de negócio;
- **Capítulo 3** - Listagem de dados: grande parte das aplicações apresentam dados de formas dinâmica e iterativa, sendo as listas de dados a principal forma de disponibilizar dados dinâmicos em desenvolvimento móvel;
- **Capítulo 4** - *Networking*: quase todas as aplicações têm funcionamento reduzido ou praticamente inexistente em modo *offline*. Para evitar essa situação é necessário existir algum tipo de comunicação entre o cliente (aplicação móvel) e o servidor ou servidores de dados;
- **Capítulo 5** - Diferenciação entre plataformas: mesmo quando o desenvolvimento é multiplataforma, ou seja, o mesmo código é usado para Android e iOS, é necessário ter muita atenção em como organizar o código, uma vez que por vezes existem situações onde têm que se executar diferentes lógicas de negócio ou usar diferentes componentes nativos dos dois sistemas operativos.

De referir que, para além do documento de questões fornecido aos programadores, foi fornecido um tutorial associado a cada um dos capítulos de forma a que o programador tivesse uma contextualização inicial dos conceitos principais necessários para poder resolver os exercícios propostos.

III. TAXONOMIA DE BLOOM ORIGINAL

Foi considerado que a realização de atividades de avaliação devesse seguir um referencial que tivesse em conta algumas considerações pedagógicas. Desta forma, a realização das atividades de teste foi pensada com base numa taxonomia de objetivos educacionais. Existem várias taxonomias de objetivos educacionais que permitem planear os conteúdos e avaliar conhecimentos em termos de níveis de dificuldade crescente e gradual. A Taxonomia de Bloom é uma das mais antigas (tendo sido a primeira taxonomia descrita como um modelo hierárquico para o domínio cognitivo) e que continua a ser amplamente usada [6].

A taxonomia de Bloom é representada em três domínios: cognitivo, afetivo e psicomotor [6]. No entanto, para os objetivos da análise de aprendizagem das *frameworks* de desenvolvimento de aplicações móveis considerámos de interesse que os testes fossem baseados no domínio cognitivo.

O domínio cognitivo da taxonomia é dividido em seis níveis: Conhecimento, Compreensão, Aplicação, Análise, Síntese e Avaliação. A Tabela 1 sintetiza os objetivos de todos os níveis da Taxonomia de Bloom [7]. O primeiro é o nível de conhecimento representando níveis mais simples de aprendizagem, enquanto o último nível é o mais complexo e abstrato. Sendo assim, num teste implementado segundo a taxonomia de Bloom é geralmente esperado um sucesso de resposta maior nos primeiros níveis em relação aos últimos [8].

Considera-se que cada nível ou categoria são cumulativos, ou seja, depende dos conhecimentos anteriores para dar suporte à seguinte. Os processos caracterizados pela taxonomia devem resultar de algum tipo de aprendizagem, ou seja, têm sempre em conta que o individuo adquiriu algum tipo de conhecimento e não que já dominava previamente o assunto.

TABELA 1 - NÍVEIS DE TAXONOMIA DE BLOOM E SEUS OBJETIVOS

Nível da Taxonomia de Bloom	***Objetivos***	***Ações***
Conhecimento	Relembrar processos ou informação específica da atividade que aprendeu.	Definir, Nomear, Recordar
Compreensão	Demonstrar compreensão por uma certa informação, sendo capaz de reproduzi-la com mecanismos próprios.	Identificar, Localizar, Transcrever
Aplicação	Recolher e aplicar informação em situações ou problemas concretos.	Aplicar, Demonstrar, Usar
Análise	Estruturar informação, separando as partes e estabelecer relações, explicando-as, entre as partes constituintes.	Classificar, Comparar, Diferenciar,
Síntese	Projetar uma solução a partir da recolha e relacionamento de várias fontes ou partes.	Construir, Criar, Formular
Avaliação	Julgar uma solução, analisá-la, otimiza-la, identificar e/ou corrigir eventuais erros.	Avaliar, Gerar, Validar

IV. Taxonomia de Bloom Adaptada

Neste trabalho, pensou-se no enquadramento das perguntas criadas para os testes em três contextos principais: aquisição dos conhecimentos, utilização dos conhecimentos e generalização ou transferência dos conhecimentos. Desta forma, entendeu-se como conveniente fazer uma adaptação e simplificação da taxonomia de Bloom aos testes realizados. Assim, foram considerados os seguintes níveis:

- **Iniciante:** integra essencialmente os níveis de Conhecimento e Compreensão da taxonomia de Bloom e tem como objetivos principais que o indivíduo consiga reconhecer, recordar e compreender informação adquirida ao longo da aprendizagem. As perguntas devem-se cingir apenas e só a conceitos abordados ao longo da aprendizagem e de fácil interpretação por parte do indivíduo;
- **Intermédio:** integra os níveis de Aplicação e Análise da taxonomia de Bloom e tem como objetivos principais o indivíduo conseguir resolver um problema que difira dos exemplos de aprendizagem, mas em que a pergunta explicite bem ao indivíduo como deve aplicar e de que forma o deve fazer;
- **Avançado:** integra os níveis de Síntese e Compreensão da taxonomia de Bloom e tem como objetivos principais que o indivíduo consiga resolver um problema não trivial, implicando as capacidades de abstração e generalização em relação aos exemplos fornecidos.

Os formatos das questões são variados e considerou-se que, no âmbito do desenvolvimento de software, fosse útil a existência de documentação de ajuda como incentivo à pesquisa individual, pois num contexto empresarial de desenvolvimento de software essa mesma pesquisa e leitura de documentação faz parte do próprio processo de desenvolvimento.

V. Estruturação dos Itens da Prova

Foram então criados, para posteriormente serem fornecidos, dois documentos. Um documento continha questões para serem resolvidas após a leitura e interiorização de conceitos explanados num outro documento de tutorial fornecido inicialmente. O documento de tutorial continha a explicação de todos os conceitos cuja aprendizagem se queria testar e ligações para *websites*, para possibilitar o acesso a mais documentação sobre o capítulo. Não só o tutorial como também as questões foram divididas nos 5 capítulos já referenciados (vistas, controlo de estado da aplicação, listas de dados, *networking* e diferenciação de plataformas). Cada prova fornecida ao programador, continha sempre 3 questões associadas a cada capítulo, uma para cada nível de dificuldade (Iniciante, Intermédio, Avançado). É importante referir que nos documentos de perguntas das três *frameworks* em análise React Native, Ionic e NativeScript, foi usada sempre a mesma linguagem nas perguntas e tipo de funcionalidade a criar, para cada pergunta de cada capítulo do respetivo nível, de forma a não existirem incoerências no tipo de perguntas.

Todos os testes foram realizados de forma presencial e individualmente, de forma a que ao longo dos testes fosse possível recolher *feedback* individual bem como o tempo despendido em cada capítulo. As variáveis obtidas para análise foram as seguintes:

- **Tempos das respostas no Capítulo 1, 2, 3, 4 e 5:** Tempo despendido por cada programador a resolver todos os problemas propostos associados a cada um dos capítulos;
- **Resultados do Capítulo 1, 2, 3, 4 e 5:** Resultados das respostas dos programadores avaliados numa escala de 0-100%, para cada capítulo de cada uma das *frameworks*;
- **Resultado global do teste:** Resultado global, em percentagem, obtido através da média de todas as questões propostas no teste;
- **Resultado por níveis da taxonomia de Bloom adaptada:** Resultado da média de todas as questões que se enquadram no nível iniciante, nível intermédio e nível avançado;
- ***Feedback* livre sobre cada capítulo:** No fim da realização das perguntas de cada capítulo todos os programadores puderam, de forma livre, dar uma opinião relativamente ao capítulo em questão para essa mesma *framework*;
- ***Feedback* pessoal sobre facilidades e dificuldades na *Framework* em geral:** No final de cada um dos testes, o programador foi questionado sobre as principais facilidades e dificuldades que teve com a *framework*;
- **Avaliação pessoal sobre realização de uma aplicação simples:** Numa escala entre 0 – 100, o programador avaliou o à-vontade em desenvolver uma aplicação simples com uma lista de dados com estilos já definidos;
- **Avaliação pessoal sobre a realização de uma aplicação para um cliente:** Numa escala entre 0 – 100, o programador avaliou o à-vontade em integrar uma equipa de desenvolvimento para um cliente em React Native, Ionic ou NativeScript.

VI. Caraterização da Amostra

Os testes foram realizados a programadores profissionais que apresentassem no mínimo mais de 6 meses de experiência profissional em desenvolvimento de *software* com JavaScript. De realçar também que não foram selecionados para integrar a amostra programadores com conhecimento prévio em qualquer das *frameworks* de desenvolvimento móvel que se pretendiam testar de forma a não influenciar os resultados. Os programadores que realizaram os testes práticos apresentavam a seguinte experiência:

- **Programador 1:** programador com um ano de experiência em JavaScript, essencialmente, *backend* em NodeJS com Express;
- **Programador 2:** programador com quatro anos de experiência em desenvolvimento de software, dos quais três anos e meio em JavaScript, mais propriamente em *frontend web*;
- **Programador 3:** programador com dois anos de experiência em JavaScript, mais propriamente com JavaScript puro (sem recurso a *frameworks*) em *frontend web*;

- **Programador 4:** programador com três anos de experiência em *backend* em PHP, mas também com cerca de seis meses de experiência em JavaScript, *frontend web* em AngularJS.
- **Programador 5:** programador com seis meses de experiência em desenvolvimento de software JavaScript em *full stack* NodeJS (Express) em *backend* e AngularJS em *frontend web.*
- **Programador 6:** programador com cinco anos de experiência em desenvolvimento de *software*, dos quais três anos e meio em JavaScript, *frontend* em AngularJS e *backend* em NodeJS (Express).

VII. Validação dos resultados

Antes de uma análise dos resultados, foi feita uma validação das questões segundo a taxonomia proposta. Todos os programadores testados sentiram, em quase todos os casos, um aumento no nível de dificuldade das perguntas à medida que avançavam em cada teste, o que era expectável face à aplicação de uma taxonomia com esses pressupostos. Desta forma, e apesar de poucas exceções, na generalidade os programadores apresentaram piores resultados em percentagem no nível Avançado e melhores resultados no nível Iniciante da taxonomia adotada. A dificuldade crescente das perguntas foi confirmada através do tempo despendido por cada programador. Assim, de todos os 90 capítulos e 270 respostas realizadas, em apenas 12 respostas não se comprovou um aumento de tempo despendido pelo programador em relação à questão anterior. Destas 12 respostas, 8 eram referentes à passagem entre a pergunta 1 e 2 do capítulo 1 de estilos. Esta situação aconteceu, provavelmente, porque a 1ª pergunta era de escolha múltipla com 4 opções de código, que envolvia uma leitura mais extensa da pergunta, sendo necessário reconhecer qual das opções não continha erro. Pensa-se que a 2ª pergunta, apesar de ser já de aplicação ao contexto do capítulo, como implicava resposta mais direta a um problema, levou a que em alguns casos o tempo de resolução do problema fosse menor para alguns programadores, não inviabilizando a maior dificuldade inerente à pergunta 2 em relação à 1.

VIII. Discussão dos resultados

De forma a obter resultados globais relacionados com todos os programadores avaliados, mas sem fazer médias de resultados, foram atribuídos valores absolutos às variáveis em estudo consoante o desempenho em cada uma das variáveis para cada um dos programadores testados. Esta forma de avaliação foi utilizada em todas as seguintes tabelas apresentadas neste trabalho e baseiam-se numa escala de valores de 0 a 2 pontos, em que os 2 pontos eram atribuídos à *framework* com melhores resultados numa determinada variável enumerada na secção 5. Um ponto à *framework* com resultado intermédio e 0 pontos à *framework* com pior resultado. Em caso de igualdade foi dada a mesma classificação, 2 pontos no caso da igualdade se verificar como melhor resultado e 1 ponto no caso da igualdade ser verificada no pior resultado.

Na Tabela 2, é possível aferir que grande parte dos capítulos apresentaram uma resolução mais rápida em Ionic, exceto na diferenciação de plataformas (Capítulo 5) em que todos os programadores apresentaram dificuldades generalizadas em implementar código distinto para cada uma das plataformas (Android e iOS). O React Native apresentou-se como a *framework* mais equilibrada e sem grandes desvios, de tal forma que o resultado global acumulado foi melhor em React Native do que em Ionic. Já no NativeScript houve dificuldade em compreender o fluxo de construção e arquitetura da *framework*.

O resumo final dos resultados dos tempos de respostas, segundo a escala de acumulados da avaliação de 0 a 2 dos vários critérios explanados anteriormente, é apresentado na Tabela 2.

Tabela 2 – Resumo dos tempos de respostas dos programadores

Critérios	*React Native*	*Native Script*	*Ionic*
Tempo de resposta Capítulo 1	8	0	11
Tempo de resposta Capítulo 2	7	0	11
Tempo de resposta Capítulo 3	7	1	10
Tempo de resposta Capítulo 4	7	4	7
Tempo de resposta Capítulo 5	12	6	0
Tempo de resposta ao teste	10	0	8
Total	**51**	**11**	**47**

Devido ao facto de todos os programadores testados terem bases sólidas de JavaScript, todas as respostas foram geralmente positivas, havendo poucas em que o programador obteve 0% no resultado de uma pergunta. Contudo, e apesar disso, apenas um único programador conseguiu obter um resultado de 100% em todas as perguntas dos 18 testes realizados, no caso concreto, na *framework* de Ionic.

Em termos de resultados quase ideais, existiram mais 2 testes em React Native e 1 outro em Ionic em que o resultado também ultrapassou os 97%. De referir que nenhum teste de NativeScript conseguiu atingir percentagens tão altas.

O React Native foi a *framework* com melhores resultados gerais para todos os programadores, tendo existido melhores resultados apenas em Ionic no capítulo 3, indo totalmente ao encontro do *feedback* reportado pelos programadores que afirmam que o *ng-repeat* (função de iteração em Ionic) torna a criação de listas de dados num procedimento quase trivial.

O NativeScript, de acordo com o *feedback* obtido pelos programadores, foi a *framework* cujos resultados ficaram totalmente abaixo das expetativas. Isto porque, a maior parte dos programadores, apesar de entenderem o procedimento necessário para a resolução do problema, durante a resolução perdiam, por vezes, o controlo do fluxo de desenvolvimento desencadeando alguns erros em todas as respostas. Novamente, o *feedback* dos programadores em relação à grande dificuldade de construção de vistas em NativeScript, que consiste numa mistura de XML com CSS, foi ao encontro dos resultados globais finais onde o NativeScript apresentou um rácio de

respostas corretas mais baixo, comparando com as outras *frameworks*.

O resumo final dos resultados em percentagem das questões respondidas, segundo a escala de acumulados da avaliação de 0 a 2 dos vários critérios explanados anteriormente, é apresentado na Tabela 3.

TABELA 3 – RESUMO DOS RESULTADOS ÀS QUESTÕES DOS PROGRAMADORES

Critérios	*React Native*	*Native Script*	*Ionic*
Resultados Capítulo 1	10	2	9
Resultados Capítulo 2	9	3	9
Resultados Capítulo 3	11	4	12
Resultados Capítulo 4	10	7	10
Resultados Capítulo 5	11	4	4
Resultados total do teste	11	1	7
Total	**62**	**21**	**51**

Após a validação das perguntas indiciar que as perguntas foram bem ordenadas e priorizadas por grau de dificuldade, foi feita uma avaliação aos resultados que as *frameworks* obtiveram consoante o grau de dificuldade da taxonomia aplicada.

O React Native apresentou melhores resultados em todos os níveis de dificuldade da taxonomia adaptada, demonstrando que, apesar do nível de dificuldade, os programadores conseguiram atingir os resultados corretos mais facilmente em React Native. Já em NativeScript e em Ionic obtiveram os piores resultados duas e uma vez, respetivamente.

O resumo final dos resultados para resultados por taxonomia adotada, segundo a escala de acumulados da avaliação de 0 a 2 dos vários critérios explanados anteriormente, é apresentado na Tabela 4.

TABELA 4 – RESULTADOS POR TAXONOMIA DE BLOOM PARA TODOS OS PROGRAMADORES

Critérios	*React Native*	*Native Script*	*Ionic*
Resultados Nível Iniciante	12	3	9
Resultados Nível Intermédio	10	8	5
Resultados Nível Avançado	10	3	7
Total	**32**	**14**	**21**

Todos os programadores fizeram ainda uma avaliação subjetiva atribuindo um valor entre 0 – 100 para cada uma das duas variáveis referentes ao à-vontade que teriam no desenvolvimento de uma aplicação num contexto real. Quase todos os programadores elegeram o React Native como a *framework* na qual se sentiriam mais confiantes no que concerne ao desenvolvimento de uma aplicação protótipo ou de um futuro cliente. A maior parte dos programadores aproximou o Ionic de avaliações semelhantes ao React Native. Porém, o NativeScript apresentou, para quase todos os programadores, um *feedback* mais desfavorável em comparação com o React Native e o Ionic, indo ao encontro de todo o *feedback* obtido.

O resumo final dos resultados do à-vontade sentido para integrar um projeto com a *framework,* segundo a escala de acumulados da avaliação de 0 a 2 dos vários critérios explanados anteriormente, é apresentado na Tabela 5.

TABELA 5 – RESUMO GLOBAL DE À-VONTADE SENTIDO COM CADA FRAMEWORK

Critérios	*React Native*	*Native Script*	*Ionic*
Avaliação de à vontade de desenvolvimento de uma aplicação simples	10	8	1
Avaliação de à vontade de desenvolvimento de aplicação para um cliente	11	6	2
Total	**21**	**14**	**3**

De acordo com o *feedback* dos programadores foi obtido um resumo final, com agrupamento das críticas positivas e negativas mais relevantes enumeradas por todos os programadores. As críticas positivas e negativas encontram-se ordenadas pelo número de vezes que foram enumeradas, sendo possível concluir que grande parte dos programadores teceram as mesmas críticas para cada uma das *frameworks* sem existir muitas incoerências no *feedback* dado por cada um.

No React Native foi bastante referida a facilidade de utilização dos estilos e das listas de dados baseados em *arrays*, mas também o facto do desenvolvimento da *framework* ser totalmente baseado em componentes, que é um dos focos principais do React Native. Por outro lado, o facto da sua sintaxe já ser baseada no padrão JavaScript 2015 ou ES6, levou a que alguns programadores a criticassem, eventualmente pelo fato de alguns não trabalharem com esse mesmo padrão.

No Ionic foram enumeradas as vantagens de ser um desenvolvimento quase replicado da *web*, com CSS puro, e *ng-repeat* típico de AngularJS da Web. O conceito de *binding* do Angular também foi um dos pontos referidos como fáceis de entender. Por ser um desenvolvimento quase igual a AngularJS para *web*, apareceram dificuldades em controlar o código consoante a plataforma móvel, tendo essa dificuldade sido identificada por todos os programadores testados. A separação da lógica e da vista, tal e qual como a maior parte das *frameworks web* trabalham, foi criticada no caso do Ionic.

O NativeScript foi bastante criticado negativamente por todos os participantes. No entanto obteve-se *feedback* positivo da *framework* relativamente aos *requests http* serem feitos tal e qual como no JavaScript ES5, ao contrário do React Native que já utilizava ES6. No NativeScript quase todas as opiniões incidiram sobre os aspetos negativos da *framework*. Destaca-se o facto da criação de vistas em XML com recurso a *Grids* ser descrito como um tipo de implementação muito pobre de tão confusa que se torna. Quase todos referiram que, mesmo depois do teste, não sabiam como controlar bem a lógica entre uma vista e um controlador nem como chamar funções entre vistas e controladores.

Os resumos gerais do *feedback* de todos os programadores encontram-se na Tabela 6 para React Native, na Tabela 7 para

NativeScript e na Tabela 8 para Ionic, em que o número indicado antes de cada declaração de *feedback* se refere ao número de vezes que o mesmo *feedback* foi enumerado.

TABELA 6 – FEEDBACK GERAL ACUMULADO DE TODOS OS PROGRAMADORES PARA REACT NATIVE

Positivo	***Negativo***
6 - Estilos são muito intuitivos e igual ao CSS convencional	**4** - Sintaxe de ES6 é recente e ainda difícil de entender para alguns programadores
5 - Listas através de arrays com o componente nativo FlatList torna a criação de listas intuitiva	
4 - Desenvolvimento baseado em pequenos componentes é fácil	

TABELA 7 – FEEDBACK GERAL ACUMULADO DE TODOS OS PROGRAMADORES PARA NATIVESCRIPT

Positivo	***Negativo***
3 – *http* request ter a mesma lógica que o JavaScript puro	**6** – Criação de vistas com XML e com recurso a Grids é demasiado complexa e difícil de entender
2 – Listas de dados através de um ciclo na vista, faz com que seja fácil a iteração de dados	**5** – Difícil de entender como se controla o estado de uma variável no código
2 – Intuitivo criar código para cada plataforma apenas com uma tag Android e iOS na vista	**5** – Não se entende como se passa/chama uma função entre a vista e a lógica de negócio

TABELA 8 – FEEDBACK GERAL ACUMULADO DE TODOS OS PROGRAMADORES PARA IONIC

Positivo	***Negativo***
6 – CSS igual à *web*	**6** – Dificuldade em separar código específico para cada plataforma
6 – Listas de dados com o *ng-repeat*	**5** – Separação de ficheiros de lógica até em pequenos componentes
3 – Conceito de *binding* da vista	

IX. CONCLUSÕES E TRABALHO FUTURO

Neste trabalho foi apresentada uma análise de aprendizagem de *frameworks* de desenvolvimento móvel JavaScript, segundo uma taxonomia, adaptada à de Bloom, criada neste trabalho para perceber a adaptação prática de programadores profissionais a cada uma das *frameworks* analisadas.

Após a aplicação e análise dos testes é facilmente deduzível que o NativeScript não é uma escolha popular entre os programadores, não apenas pela dificuldade demonstrada nas atividades realizadas pelos programadores, mas também pelo seu *feedback* ser em regra geral sempre negativo em relação à sua lógica de implementação. Esta conclusão contraria a análise comparativa inicial que não envolveu testes a programadores [3], em que o NativeScript se apresentava como uma boa solução, quase ao nível do React Native.

Neste trabalho o Ionic revelou-se como a *framework* mais próxima do React Native em alguns resultados, contudo, e ainda assim, quase sempre com resultados inferiores em relação ao React Native. Logo, o Ionic poderia também ser uma solução para programadores que sabem AngularJS e para uma aplicação que não envolvesse demasiadas vistas com animações, lógica de negócio complexa ou demasiada interação com o *hardware*. Isto porque o Ionic ainda demonstra algumas debilidades a nível de desempenho em relação ao React Native e NativeScript, por ter uma base assente sobre uma *webview* [3], [4], [9], [10].

O React Native em quase todos os pontos apresentou-se como a *framework* mais estável, nunca tendo qualquer resultado baixo em qualquer um dos capítulos, não apenas em tempo de desenvolvimento como também em percentagem de acerto das respostas às perguntas. Para além disso, o *feedback* relativamente ao React Native obtido dos programadores foi muito bom, os quais acharam o desenvolvimento através dessa mesma *framework* bastante intuitivo. Este resultado dos testes práticos é totalmente coincidente com as conclusões obtidas em trabalho prévio [3]. Neste e nos testes práticos aplicados aos programadores, também se concluiu que o React Native era a *framework* JavaScript, de entre a avaliação comparativa realizada, com melhores resultados globais, apresentando-se assim como melhor alternativa.

Como trabalho futuro propomos um estudo aprofundado do React Native, por ter sido a *framework* com melhores resultados, relativamente ao ReactJS, *framework* de desenvolvimento *web* que serviu de base ao React Native, para perceber quais os eventuais ganhos em tempo e percentagem de código que pode ser reaproveitado entre o desenvolvimento *frontend mobile* e *web*.

REFERÊNCIAS BIBLIOGRÁFICA

[1] Statista, "Number of smartphone users worldwide from 2014 to 2020," 2017. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed: 27-Dec-2018].

[2] GitHub, "The State of the Octoverse 2017," 2017. [Online]. Available: https://octoverse.github.com/. [Accessed: 14-Jan-2018].

[3] Brito Hugo, Gomes Anabela, Santos Álvaro, Jorge Bernardino, "JavaScript em aplicações móveis: React Native vs Ionic vs NativeScript vs desenvolvimento nativo," *CISTI*, 2018.

[4] Optimus Information, "Native , Hybrid or Mobile Web Application Development Develop a Mobile Application Strategy for Your Business," 2015.

[5] N. Singh, "An comparative analysis of Cordova Mobile Applications V/S Native Mobile Application," *Int. J. Recent Innov. Trends Comput. Commun.*, pp. 3777–3782.

[6] K. Goals, "Bloom ' s Taxonomy of Educational Objectives."

[7] K. Words, "Bloom ' s Taxonomy : A New Look at an Old Standby," pp. 1–6, 2012.

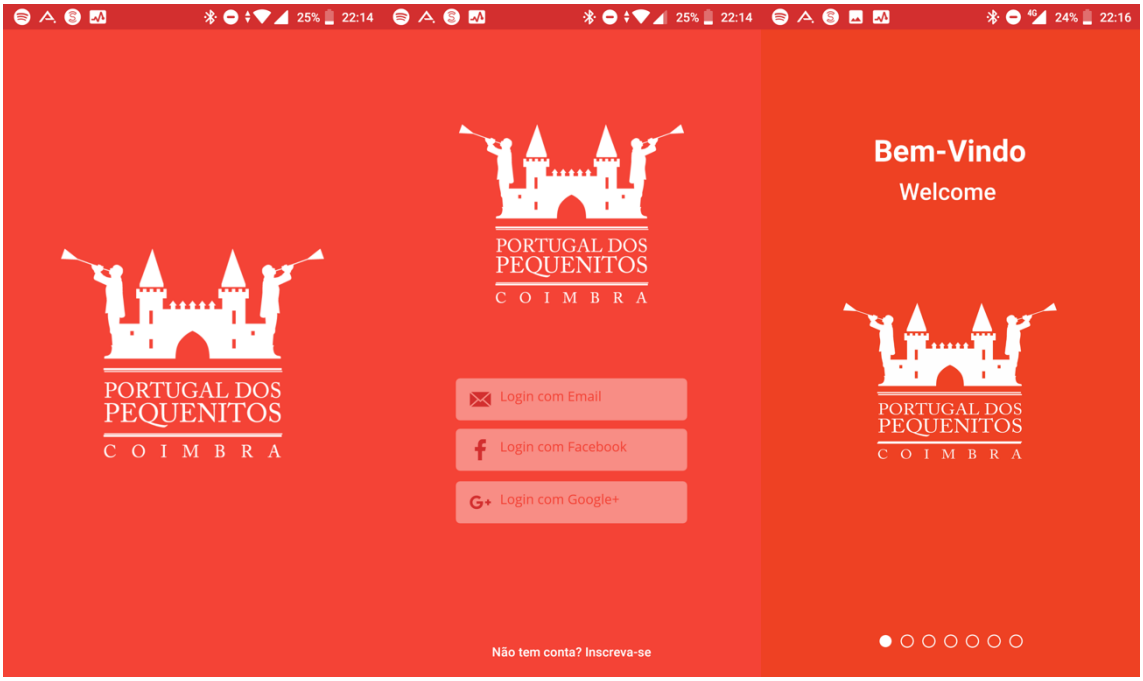
[8] D. R. Krathwohl, "A Revision of Bloom ' s Taxonomy :," vol. 41, no. 4, pp. 212–219, 2002.

[9] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," *Proc. 6th Balk. Conf. Informatics - BCI '13*, p. 213, 2013.

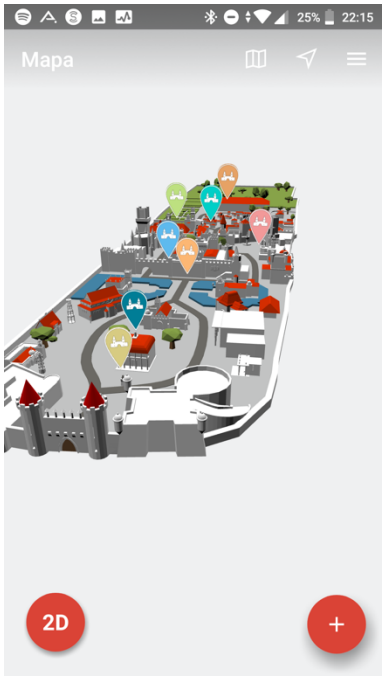
[10] F. Asp Handledare, A. Palanisamy, O. Karlsson Examinator, and K. Sandahl, "A comparison of Ionic 2 versus React Native and Android in terms of performance, by comparing the performance of applications," 2017.

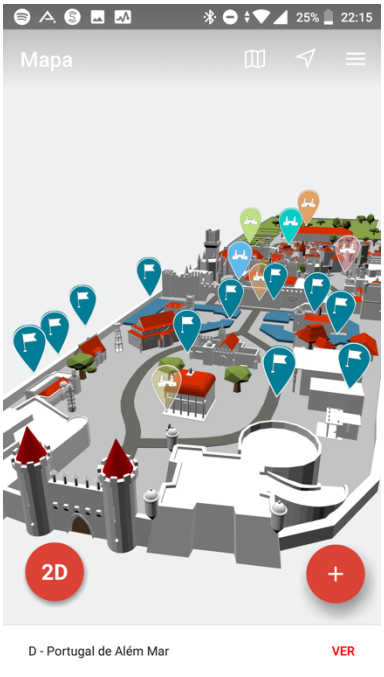
Anexo F – Resultado final da aplicação Portugal dos Pequenitos

O resultado final da aplicação Portugal dos pequnitos foi o seguinte:



Splash Screen *Ecrã de Login* *Tutorial da Aplicação*

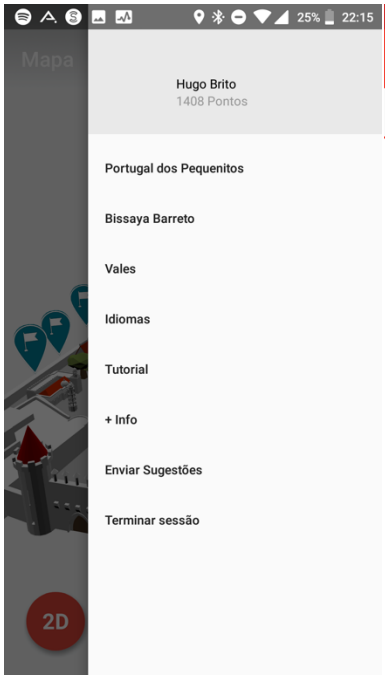




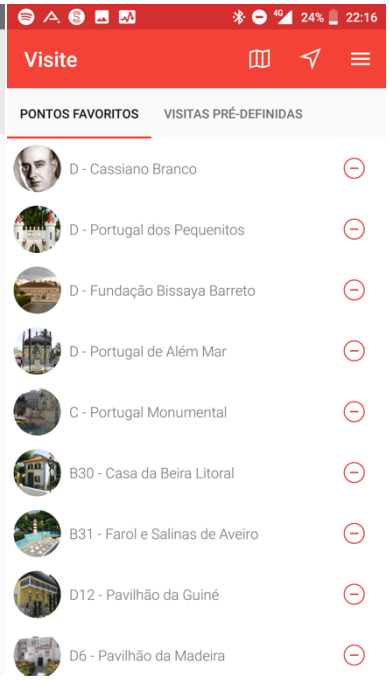
Mapa 3D com pontos principais

Opções da vista principal

Comportamento após clique sobre um dos pontos



Menu lateral (Drawer)



Funcionalidade de favoritos

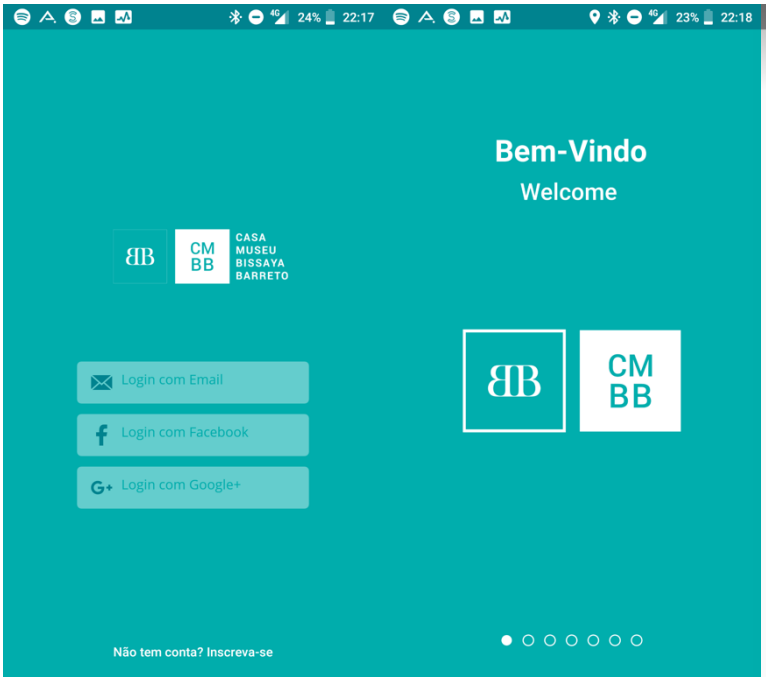
Descrição de ponto de interesse

Descrição de ponto de interesse, com pontos filhos associados

Outros detalhes associados ao ponto de interesse, galleria e quiz

Anexo G – Resultado final da aplicação Casa Museu Bissaya Barreto

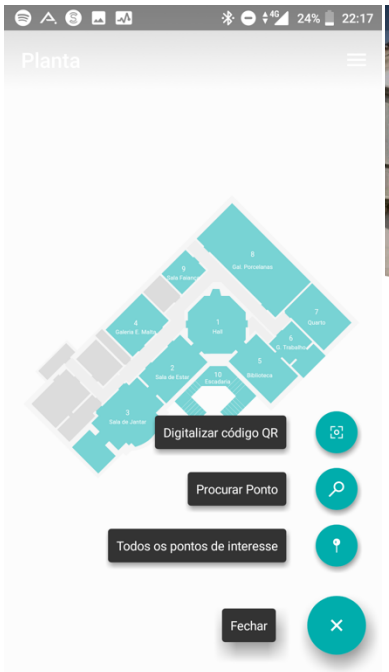
O resultado final da aplicação Casa Museu Bissaya Barreto foi o seguinte:



Ecrã de Login

Tutorial

Mapa da Casa Museu em 2D



Opções principais do ecrã principal

Ponto de interesse

Anexo H – Documento do boilerplate

React-native boilerplate

Welcome to Crossing's react-native boilerplate.

Introduction to react-native

Build native mobile apps using JavaScript and React React Native lets you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components.

Text Editor

Text Editor is a programmer's choice but we prefer to use either Visual Studio Code.

Getting Started

In order to start making applications in React Native it is necessary to learn React in general, and all concepts associated with React, before proceeding to any prototype of test application, should be performed at 100% and without any jump the following tutorials / exercises about the concept **React**.

1 - https://www.codecademy.com/learn/react-101
2 - https://www.codecademy.com/learn/react-102

Have you finished both tutorials completely? Then you are ready to start small example of React Native.

For this it is necessary to read all the documentation with the rules of development in React Native.

Sample / prototype apps are usually created with Create React Native App. It is faster to setup and needs no cofiguration.

When creating a big project we usually start from scratch, using the react-native CLI.
Different configurations are needed for different platforms (OSX / Linux / Windows).

Important resources

- React-native docs
- Learn React Native through interactive examples with RN Express
- List of Awesome React Native components, news, tools, and learning material
- Redux Docs - a predictable state container for JavaScript apps
- React Navigation
- Jest

Folder structure

```
ProjectName
├──  assets
│    ├── logo.svg
│    └── icons.ttf
├── src
│    ├── components
│    │   ├── colors
│    │   │    └── index.js
│    │   ├── styles
│    │   │    └── index.js
│    │   ├── container.js
│    │   ├── button.js
│    │   ├── input.js
│    │   └── index.js
│    ├── containers
│    │   ├── login
│    │   │   └── index.js
│    │   ├── helpers.js
│    │   └── index.js
│    ├── i18n
│    │   ├── locales
│    │   │   ├── en
│    │   │   │   ├── dynamic.js
│    │   │   │   └── static.js
│    │   │   └── pt
│    │   │       ├── dynamic.js
│    │   │       └── static.js
│    │   └── index.js
│    ├── modules
│    │   ├── user
│    │   │   ├── actions
│    │   │   │   ├── helpers.js
│    │   │   │   └── index.js
│    │   │   └── reducers
│    │   │       └── index.js
│    │   ├── errors
│    │   │   ├── actions
```

```
│   │   │   │   └── index.js
│   │   │   └── reducers
│   │   │       └── index.js
│   │   ├── constants.js
│   │   ├── helpers.js
│   │   └── index.js
│   ├── router
│   │   ├── drawer.js
│   │   ├── dashboard.js
│   │   ├── profile.js
│   │   └── index.js
│   ├── utils
│   │   ├── api
│   │   │   └── index.js
│   │   ├── permissions
│   │   │   └── index.js
│   │   └── config.js
│   ├── configureStore.js
│   └── index.js
├── android
├── ios
├── package.json
├── README.md
└── index.js
```

Folder description

- **Assests**: Includes all external and static contents related to the project such as images, videos or others
- **Components**: They are view contents that are reused for different types of view and also different confirations of styles related to the view
    - **colors**: All colors in constants used in the application
    - **styles**: All styles that do not belong to the components
    - **container**: Example of a component created to be used in different views
    - **index.js**: Main file where all the associated components are included in the folder, so that when included in views it is possible to include all in the same declaration
- **Containers**: They are all views of the application, which are may contain different components and data coming from various modules
    - **login**: Example of a login container, which links the view and the data associated with the login
    - **helpers**: All the functions used in the containers and that can be reused in other containers
- **i18n**: Module of management of the different types of languages that the application can assume
- **modules**: The architecture used is Redux, this same architecture is based on actions that according to the type of action send update of information to the respective reduce
    - **actions**: They serve to control the logic between the view and the data storage in Redux, being applied the logic associated with the new state that will be updated by the action done
    - **reducers**: They serve to store the data of the application, with an initial state always defined
- **router**: All navigation is done with react-navigation, which allows you to create stacks, drawers or simple views by default, facilitating all navigation between views and controlling the entire stack control process in both operating systems
- **utils**: Files used in all projects, being as global and reusable as possible

    - **api**: Has all methods and exception handling associated with a REST API type communication
    - **permissions**: It has global functions that allow the control of the main permissions normally used such as location or camera
    - **config.js**: File that allows you to have the various global settings of the application
- **index.js**: Used to do the Redux configuration and data persistence
- **android**: It has all the project for Android, having as main files the build.gradle, and the mainActivity of Android that are important to configure some new package necessary in the creation of an application
- **ios**: It has all the project for iOS, as the executable for XCode and all the librarias used and that can be introduced through the installation of a new package
- **index.js**: Just only register the application in JavaScript for both Android and iOS

Package.json

Required:

- Redux
- Redux Thunk
- React Redux
- Redux Logger
- React Navigation
- React Native Vector Icons
- Redux Persist
- Redux Form
- React Native i18n
- Moment
- Device Info

Optional:

- React Native Debugger
- React Native Fetch BLOB
- React Native FS
- React Native Contacts
- React Native Image Picker
- React Native Fast Image
- React Native Android Dialogs
- React Native Snap Carousel
- React Native Maps

Initializing a new Project

If you already installed the necessary tooling to build your React Native app you should be good to init your project and install the packages we usually use.

```
react-native init ProjectName
npm i

npm install --save redux
```

```
    npm install --save react-redux
    npm install --save redux-logger
    npm install --save redux-thunk
    npm install --save redux-persist
    npm install --save react-navigation
    npm install --save react-native-i18n
    npm install --save react-native-vector-icons
    npm install --save react-native-device-info
    npm install --save react-native-offline
    npm install --save moment

    react-native link

    npm install --save-dev babel-eslint
    npm install --save-dev eslint-config-prettier
    npm install --save-dev eslint-plugin-import
    npm install --save-dev eslint-plugin-jest
    npm install --save-dev eslint-plugin-jsx-a11y
    npm install --save-dev eslint-plugin-prettier
    npm install --save-dev eslint-plugin-react
    npm install --save-dev eslint-plugin-react-native
    npm install --save-dev react-native-debugger-open
    npm install --save-dev remote-redux-devtools
```

Package.json scripts

Add the following scripts in your package.json (stuff related to ESLint and react-native-debugger).

```
    "scripts": {
        "start": "node node_modules/react-native/local-cli/cli.js start",
        "test": "jest",
        "postinstall": "rndebugger-open",
        "eslint-check": "eslint --print-config .eslintrc.js | eslint-
config-prettier-check"
    },
    "jest": {
        "preset": "react-native",
        "transformIgnorePatterns": [
            "/node_modules/(?!react-native|react-navigation)/"
        ]
    }
```

ESLint configuration

ESLint is an open source JavaScript linting utility.

Code linting is a type of static analysis that is frequently used to find problematic patterns or code that doesn't adhere to certain style guidelines.

JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error. Without the benefit of a compilation process, JavaScript code is typically executed in order to find syntax or other errors. Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it.

ESLint is written using Node.js to provide a fast runtime environment and easy installation via npm.

The rules are mainly based on the airbnb-base, because it is the most complete lint extensions and that most meets the way of programming from Crossing Answers.

Some rules are especially adapted to React Native and to mobile development.

Its use is obligatory.

Install and configure ESLint on your project

```
install extension for Visual Studio Code - ESLint by Dirk Baeumer

npm install -g eslint //(if you don't have eslint globally installed)

eslint --init


    > Choose JSON file
    > Paste the following code in .eslintrc.json file

{
    "extends": [
        "airbnb-base",
        "plugin:react/recommended",
        "plugin:react-native/all"
    ],
    "plugins": [
        "react",
        "react-native"
    ],
    "env": {
        "react-native/react-native": true
    },
    "parser": "babel-eslint",
    "parserOptions": {
        "ecmaVersion": 7,
        "sourceType": "module",
        "allowImportExportEverywhere": false,
        "codeFrame": false
    },
    "rules": {
        "max-len": ["warn", 150],
        "no-console": "off",
        "no-useless-concat": "warn",
        "no-unused-vars": "warn",
        "func-names": ["error", "never"],
        "comma-dangle": ["error", "never"],
        "prefer-template": "off",
```

```
            "brace-style": ["error", "stroustrup"],
            "prefer-destructuring": "off",
            "wrap-iife": "off",
            "import/prefer-default-export": "off",
            "no-useless-constructor": "off",
            "no-underscore-dangle": "off",
            "arrow-body-style": "off",
            "no-lonely-if": "off",
            "function-paren-newline": "off",
            "no-use-before-define": ["error", {
                "functions": false
            }],
            "no-param-reassign": ["error", {
                "props": false
            }],
            "object-curly-newline": "off",
            "indent": [
                "error",
                4
            ],
            "linebreak-style": "off",
            "quotes": [
                "error",
                "single"
            ],
            "semi": [
                "error",
                "always"
            ],
            "react-native/no-unused-styles": "error",
            "react-native/split-platform-components": "error",
            "react-native/no-inline-styles": "error",
            "react/jsx-closing-bracket-location": "warn",
            "class-methods-use-this": ["error", {
                "exceptMethods": [
                    "render",
                    "getInitialState",
                    "getDefaultProps",
                    "getChildContext",
                    "componentWillMount",
                    "componentDidMount",
                    "componentWillReceiveProps",
                    "shouldComponentUpdate",
                    "componentWillUpdate",
                    "componentDidUpdate",
                    "componentWillUnmount"
                ]
            }]
        }
    }
```

Deployment

To deploy the applications, it is necessary to take into account that Android and iOS have totally different processes to generate new releases.

It is important to note that for all applications the same tests must be performed for both operating systems, because although the code is used for different operating systems, strange and unexpected behavior can happen for some of them for a particular action.

Android apps

Android requires that all apps be digitally signed with a certificate before they can be installed, so to distribute your Android application via Google Play store, you'll need to generate a signed release APK.

The Signing Your Applications page on Android Developers documentation describes the topic in detail. This guide covers the process in brief, as well as lists the steps required to package the JavaScript bundle.

All Existing Project Keys are in Google Drive from crossing.dev@gmail.com.

You can generate a private signing key using keytool.

```
$ keytool -genkey -v -keystore my-release-key.keystore -alias my-key-alias
-keyalg RSA -keysize 2048 -validity 10000
```

1. Place the my-release-key.keystore file under the android/app directory in your project folder.
2. Edit the file ~/.gradle/gradle.properties or android/gradle.properties and add the following (replace ***** with the correct keystore password, alias and key password).

```
MYAPP_RELEASE_STORE_FILE=my-release-key.keystore
MYAPP_RELEASE_KEY_ALIAS=my-key-alias
MYAPP_RELEASE_STORE_PASSWORD=*****
MYAPP_RELEASE_KEY_PASSWORD=*****
```

3. Edit the file android/app/build.gradle in your project folder and add the signing config

```
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            if (project.hasProperty('MYAPP_RELEASE_STORE_FILE')) {
                storeFile file(MYAPP_RELEASE_STORE_FILE)
                storePassword MYAPP_RELEASE_STORE_PASSWORD
                keyAlias MYAPP_RELEASE_KEY_ALIAS
                keyPassword MYAPP_RELEASE_KEY_PASSWORD
            }
```

```
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
}
```

4. Generating the release APK

```
$ cd android
$ ./gradlew assembleRelease
```

5. The generated APK can be found under android/app/build/outputs/apk/app-release.apk

Beta

The Crossing Answers for unpublished Beta uses only the generated .apk to share. The standard used for describing .apk is' DD-MM-YYY'.apk.

Play store

To publish apps to the Play Store, you need to access the Google Play Console at https://play.google.com/ and sign in with email francovingadas@gmail.com.

Ios apps

The entire production process can be done through XCode.

1. Create a project on https://itunesconnect.apple.com.
2. Select your project in the Xcode Project Navigator, then select your main target. Look for the “General” tab. Go to “Signing” and make sure your Apple developer account team is Crossing Answers.
3. Select the device as Generic IOS Device .
4. Product->Clean and build your application.
5. Then select Product->archive,it will create an archive file in Organiser->archives and open it up for you after archive finished.
6. Once archive is successfully completed it will open in Organizer window. Then select the export function from the right side section.
7. Upload archive to iTunes.

TestFlight

After the new version of the application is sent to https://itunesconnect.apple.com just go into the application and go to the TestFlight option, automatically showing the new version in the application. After that, just select the test group to whom you want to send the application, and iTunes notifies all users that the new version of the application is ready for testing.

App store

After the new version of the application is sent to https://itunesconnect.apple.com you only have to enter the application and do a process identical to the one in TestFlight yet in the App Store option. It is necessary to fill in more information about the application, later the application goes to an acceptance process that will at least always take 24 hours.

Anexo I – Criação de Bridge de comunicação de código entre React Native – Java e C ou C++

BUILDING A REACT NATIVE BRIDGE LIBRARY TO ANDROID WITH C/C++

Introduction

The present document explains step by step how to create a library/package for React Native. This “bridge” will allow the programmer to write native code in Java for Android and access it within the React Native app that is being developed.

After this explanation, there’s also a step by step guide on how to write C/C++ code and use it on your Android apps which will lead to the possibility of accessing C/C++ code in your React Native app through this bridge.

1. Bridge React - Android

In this first point, we’ll explain how you can create an Android app and use the routines of it in your React Native app.

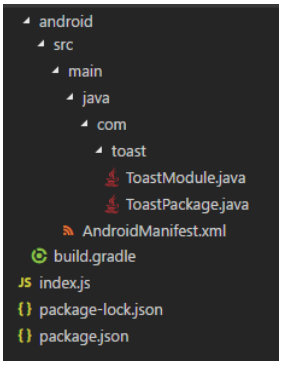
Preparation

Using the Command Prompt run the following commands:

1. Create the package directory:
   ```
   mkdir react-native-android-custompackage
   ```
2. Initiate the package
   ```
   cd react-native-android-custompackage
   npm init
   ```
3. The **package.json** should look like this unless you configure the package differently using the prompt:

```
{
  "name": "react-native-android-custompackage",
  "version": "1.0.0",
  "description": "Android Custom Package",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Android",
    "custompackage",
    "react-native"
  ],
  "author": "Reis",
  "license": "ISC",
  "peerDependencies": {
    "react-native": "^0.41.2"
  }
}
```

4. After this, you should create the following folders and files inside the package folder (in this example we'll the Toast Package):



Folders:

- android/src/main/java/com/**toast**/

Files:

- **ToastModule.java** (inside the folder ./<package_folder>/android/src/main/java/com/toast/);
- **ToastPackage.java** (inside the folder ./<package_folder>/android/src/main/java/com/toast/);
- **AndroidManifest.xml** (inside the folder ./<package_folder>/android/src/main/java/);
- **build.gradle** (inside the folder ./<package_folder>/android/);
- **package-lock.json** (inside the ./<package_folder>/);

5. You'll now be ready to start coding.

Coding

The content of the created files for this example will be:

ToastModule.java (always pay attention to the imports!)

```
package com.toast;

import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactContext;
import com.facebook.react.bridge.ReactMethod;
import com.facebook.react.bridge.Callback;
import android.widget.Toast;

public class ToastModule extends ReactContextBaseJavaModule {

 private final ReactApplicationContext reactContext;

 public ToastModule(ReactApplicationContext reactContext) {
   super(reactContext);
   this.reactContext = reactContext;
 }
 @Override
 public String getName() {
   return "ToastModule";
 }
 @ReactMethod
 public void somaValores(Integer v1, Integer v2, Callback callback) {
   Integer result = v1 + v2;
   callback.invoke(result);
 }
}
```

ToastPackage.java (always pay attention to the imports!)

```
package com.toast;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import com.facebook.react.ReactPackage;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.uimanager.ViewManager;
import com.facebook.react.bridge.JavaScriptModule;

public class ToastPackage implements ReactPackage {
   @Override
   public List<NativeModule> createNativeModules(ReactApplicationContext reactContext) {
     return Arrays.<NativeModule>asList(new ToastModule(reactContext));
   }

   // Deprecated from RN 0.47
   public List<Class<? extends JavaScriptModule>> createJSModules() {
     return Collections.emptyList();
   }

   @Override
   public List<ViewManager> createViewManagers(ReactApplicationContext reactContext) {
     return Collections.emptyList();
   }
}
```

AndroidManifest.xml (pay attention to the package name!)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.toast">
</manifest>
```

build.gradle

```
buildscript {
   repositories {
       jcenter()
   }
   dependencies {
       classpath 'com.android.tools.build:gradle:1.3.1'
   }
}
apply plugin: 'com.android.library'
android {
   compileSdkVersion 26
   buildToolsVersion "26.0.2"
   defaultConfig {
       minSdkVersion 16
       targetSdkVersion 26
       versionCode 1
       versionName "1.0"
   }
   lintOptions {
       abortOnError false
   }
}
repositories {
   mavenCentral()
}
dependencies {
   compile 'com.facebook.react:react-native:+'
}
```

index.js

```
import { NativeModules } from 'react-native';
const { ToastModule } = NativeModules;
export default ToastModule;
```

package-lock.json

```
{
   "name": "react-native-android-toast",
   "version": "1.0.0",
   "lockfileVersion": 1
}
```

Import package and usage

Now that you have the files all set up to start working on your project, you only need to link the package to your app, on the Command Prompt run the following commands:

1. `npm install <path_to_custom_package> --save`
2. `react-native link <package_name>`

Now that your package is linked to your app, you can import it by adding the following code to your JavaScript files (**pay attention to the package name!**):

```
import ToastModule from 'react-native-android-toast';
```

Here's an example to get the sum of two values by using a function from your package:

```
import React, { Component } from 'react';
import { Text } from 'react-native';
import ToastModule from 'react-native-android-toast';
import { styles, Container } from '../../components';

export default class AndroidSpike extends Component {
   constructor(props) {
       super(props);
       this.state = {
           value: 0
       };
   }

   componentDidMount() {
       this.sum();
   }

   sum = () => {
       return ToastModule.somaValores(1, 3, (result) => {
           this.setState({ value: result });
       });
   }

   render() {
       const { value } = this.state;

       return (
           <Container>
               <Text style={[styles.itemsCenter, { fontSize: 30 }]}>{value}</Text>
           </Container>
       );
   }
}
```

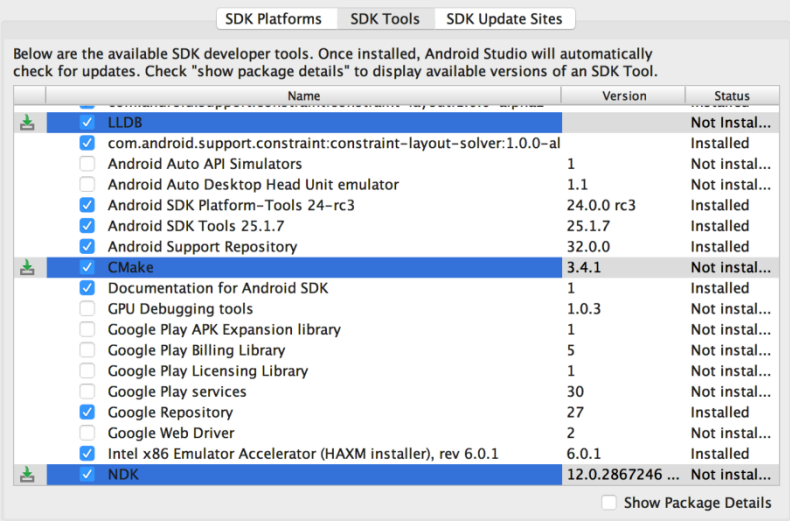
2. Bridge Android - C/C++

Now that you can link your Java Android code to your React Native app, let's see how you can import and use C/C++ code in your Android app.

Preparation

1. On Android Studio, select **Tools > Android > SDK Manager** from the menu bar;

2. Click the SDK Tools tab and check the boxes next to LLDB, CMake, and NDK;



3. On the Command Prompt, check if the PATH of ANDROID_NDK_HOME is set up correctly by running:

```
echo %ANDROID_NDK_HOME%
```

The path should be something like this:

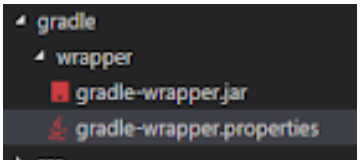
C:\Users\Crossing\AppData\Local\Android\android-sdk\ndk-bundle

4. You can now start setting up your project to use C/C++ functions.

Coding

Before start coding, the are some changes that should be made to your package files:

1. On the android folder of your project, add the gradle wrapper folder with the 4.1 distribution



2. Add a **local.properties** file to your android folder with the following content

```
# Header note:
#Wed Aug 29 19:11:16 BST 2018
ndk.dir=C\:\\Users\\Crossing\\AppData\\Local\\Android\\android-sdk\\ndk-bundle
sdk.dir=C\:\\Users\\Crossing\\AppData\\Local\\Android\\android-sdk
```

3. Change your **build.gradle** file to this;

```
buildscript {
   repositories {
       jcenter()
       maven {
           url 'https://maven.google.com'
       }
   }

   dependencies {
       classpath 'com.android.tools.build:gradle:3.0.1'
   }
}

apply plugin: 'com.android.library'

android {
   compileSdkVersion 26
   buildToolsVersion "26.0.2"

   defaultConfig {
       minSdkVersion 16
       targetSdkVersion 26
       versionCode 1
       versionName "1.0"
       externalNativeBuild {
           cmake {
               cppFlags ""
           }
       }
       ndk {
           abiFilters 'x86', 'armeabi-v7a', 'x86_64', 'arm64-v8a'
       }
   }
   lintOptions {
       abortOnError false
   }
   externalNativeBuild {
       cmake {
           path "CMakeLists.txt"
       }
   }
}

repositories {
   mavenCentral()
   maven {
       url 'https://maven.google.com'
   }
   maven { url "https://jitpack.io" }
}

dependencies {
   compile 'com.facebook.react:react-native:+'
}
```

Create a folder called cpp with your C/C++ files on the ./android/src/main folder

Our .cpp file will be called **native-lib.cpp** and the content of it will be:

```
#include <jni.h>
#include <iostream>
#include <algorithm>

using namespace std;

const int SIZE = 7;


int sum(int v1, int v2){
   return v1 + v2;
}

extern "C"{
   int Java_com_toast_ToastModule_sumTwoValues(
            JNIEnv *env,
            jobject obj,
            int v1,
            int v2) {

        return sum(v1, v2);
   }

   jintArray Java_com_toast_ToastModule_orderMyArray(
        JNIEnv *env,
        jobject obj,
        jintArray arr) {

        jsize len = env->GetArrayLength(arr); // get size of array
        jint *body = env->GetIntArrayElements(arr, 0); // get elements of array

        sort(body, body + len); // function to sort (this would be the call to the function we wanted to use)

        env->ReleaseIntArrayElements(arr, body, 0); // set array to return
        return arr;
   }
}
```

Pay special attention to the names of the functions and the types of the functions!

Create a **CMakeLists.txt** on the android folder with the imports of the c/cpp files:

```
cmake_minimum_required(VERSION 3.4.1)
# set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -Wall -UNDEBUG")

add_library( # Sets the name of the library.
        native-lib
        SHARED
        src/main/cpp/native-lib.cpp )

find_library(log-lib,  log)

target_link_libraries(native-lib
              ${log-lib} )
```

Make the following changes on your **ToastModule.java** file so it can now use the C/C++ functions you've imported:

```
package com.toast;

// … IMPORTS …

public class ToastModule extends ReactContextBaseJavaModule {

 // Used to load the 'native-lib' library on application startup.
 static {
   System.loadLibrary("native-lib");
 }

 private final ReactApplicationContext reactContext;
 public ToastModule(ReactApplicationContext reactContext) {
   super(reactContext);
   this.reactContext = reactContext;
 }
 @Override
 public String getName() {
   return "ToastModule";
 }

 @ReactMethod
 public void sumValues(int v1, int v2, Callback callback) {
   int result = sumTwoValues(v1, v2);
   callback.invoke(result);
 }

 @ReactMethod
 public void orderArrayOfIntegers(ReadableArray values, Callback callback) {
   List<Integer> deconstructedList = new ArrayList<>();
   int [] ints = new int[values.size()];

   for (int i = 0; i < values.size(); i++) {
     deconstructedList.add(values.getInt(i));
     ints[i] = values.getInt(i);
   }

   int [] test2 = orderMyArray(ints);
   List<Integer> newInteger = new ArrayList<>();
   for (int i = 0; i < test2.length; i++) {
     newInteger.add(test2[i]);
   }
   callback.invoke(newInteger.toString());
 }

   // A native method that is implemented by the 'native-lib' native library,
   // which is packaged with this application.
   public native int sumTwoValues(int v1, int v2);
   public native int [] orderMyArray(int [] v);
}
```

You can now build and run your project with the Android package that uses the C++ code,

References

[1]https://blog.botreetechnologies.com/how-to-build-a-react-native-android-bridge-4166e114a990

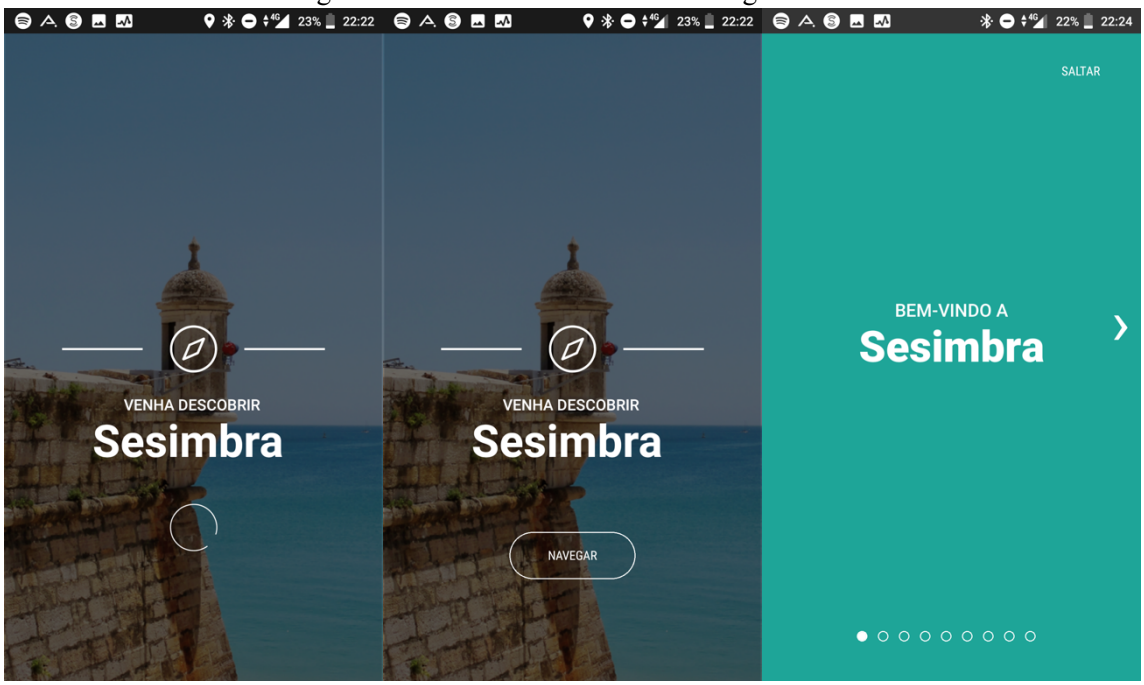
[2] https://shellmonger.com/2018/01/19/creating-a-react-native-bridge-library/

[3] https://docs.npmjs.com/cli/install

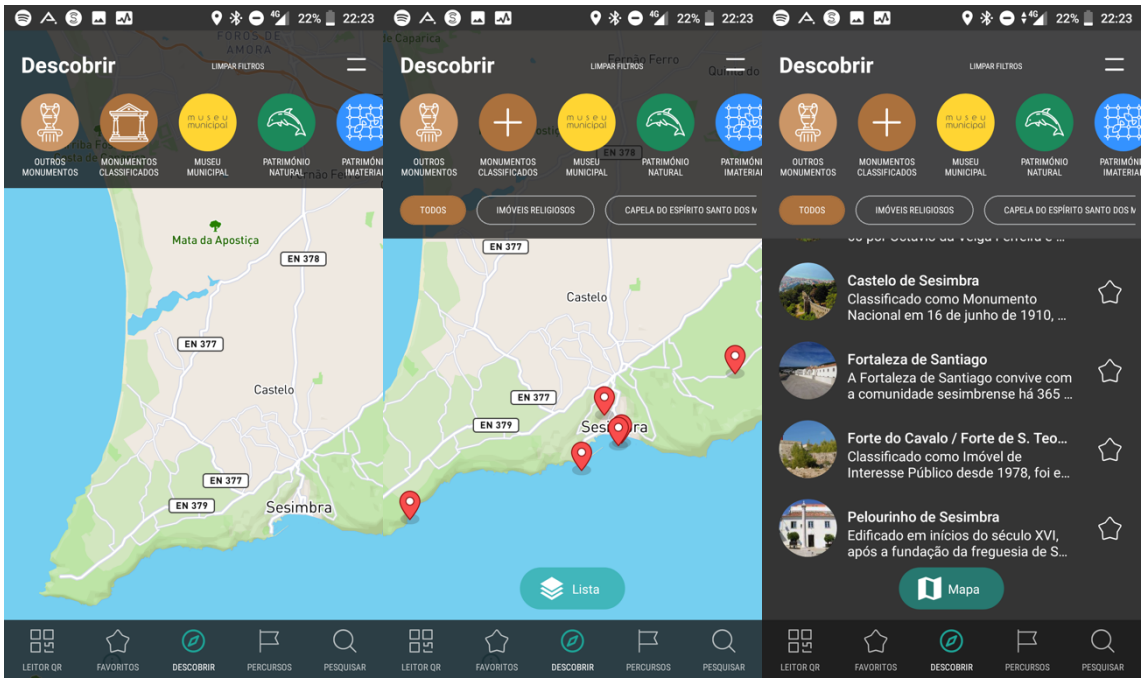
[4] https://developer.android.com/studio/projects/add-native-code

Anexo J – Resultado final da aplicação audioguia de Sesimbra

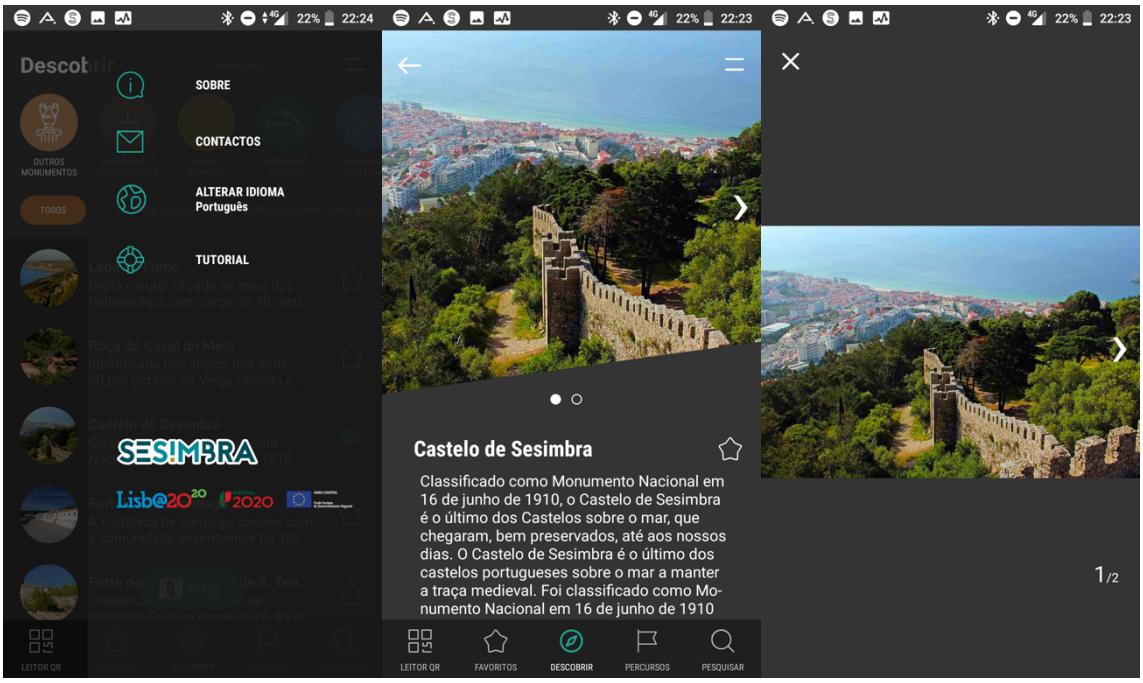
O resultado final da áudio guia criada em React Native foi o seguinte:



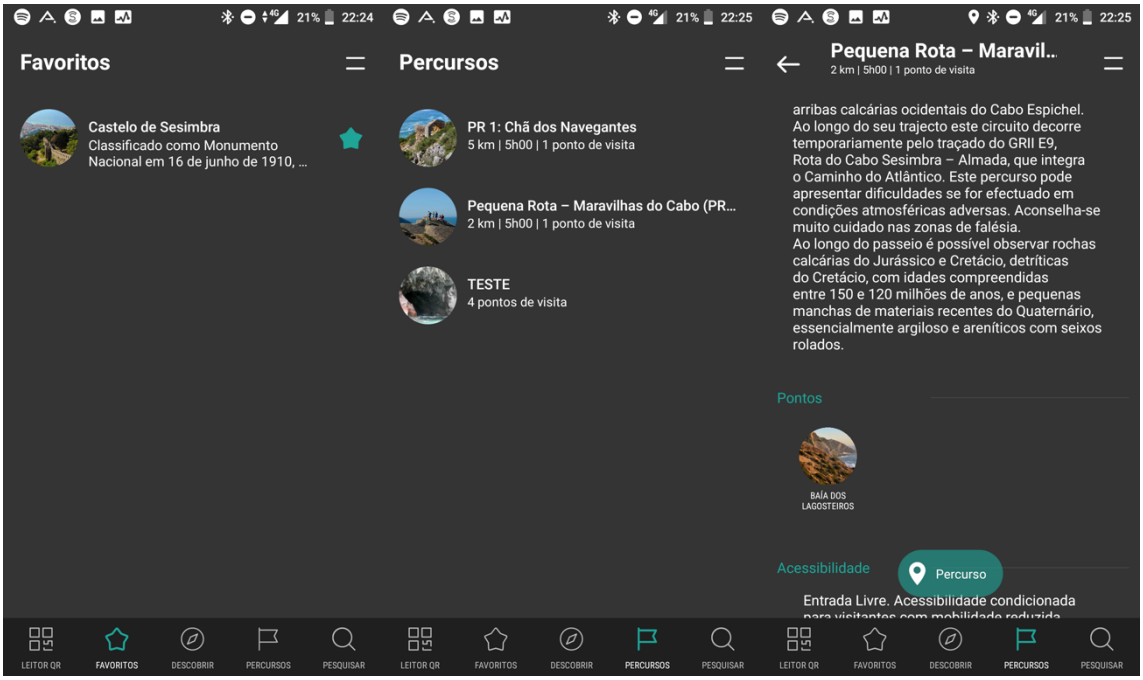
Splash Screen *Ecrã de entrada* *Tutorial da Aplicação*



Mapa inicial da aplicação *Apresentação de pontos de interesse no mapa* *Apresentação dos pontos de interesse em lista*



Menu lateral (Drawer) *Perfil de ponto de interesse* *Galeria de imagens*



Lista de favoritos *Lista de rotas* *Detalhes da rota*

Anexo K – Artigo Desenvolvimento móvel em Swift, Java e React Native: uma avaliação experimental em audioguias

Desenvolvimento móvel em Swift, Java e React Native: uma avaliação experimental em áudioguias

Mobile development in Swift, Java and React Native: an experimental evaluation in audioguides

Hugo Brito, Álvaro Santos, Jorge Bernardino, Anabela Gomes
Coimbra Polytechnic - ISEC
Coimbra, Portugal
a21220118@isec.pt, ans@isec.pt, jorge@isec.pt, anabela@isec.pt

Resumo — **O desenvolvimento móvel está cada vez mais integrado nas empresas de desenvolvimento de *software*. Atualmente os clientes esperam aplicações com bom desempenho em menos tempo e com menor custo. Já não é novidade a procura por uma solução híbrida que permita o desenvolvimento de aplicações para ambas as plataformas com o mesmo código. O React Native é uma *framework* recente que surgiu com o intuito de colmatar as lacunas apresentadas por soluções anteriores, aliando o desenvolvimento em JavaScript, que neste momento é considerada a melhor solução para desenvolvimento *frontend*. Neste artigo é feita uma análise comparativa do tempo de desenvolvimento das principais funcionalidades de uma aplicação do tipo áudioguia, para Java (Android), Swift (iOS) e React Native (Android e iOS). Este trabalho indica que o Swift é a solução que apresenta o tempo de desenvolvimento mais baixo, enquanto o React Native e o Java apresentam tempos de desenvolvimento totais muito idênticos, contudo com a particularidade do React Native dar suporte aos dois principais sistemas operativos móveis.**

Palavras Chave – JavaScript, Android, iOS, React Native.

Abstract — **Mobile development is increasingly integrated into software development companies. Customers are currently expecting applications with good performance in less time and at lower cost. The demand for a hybrid solution which allows the development of applications for both platforms with the same code is no longer new. React Native is a recent framework that has emerged to bridge the gaps presented by previous solutions, combining JavaScript development, which is now considered the best solution for frontend development. In this article, a comparative analysis of the development time of the main functionalities of an audio-guide application for Java (Android), Swift (iOS) and React Native (Android and iOS) is made. This work shows that Swift is the solution with the lowest development time, while React Native and Java have very similar overall development times, however with the particularity that React Native supports both the two main mobile operating systems.**

Keywords - JavaScript, Android, iOS, React Native.

I. INTRODUÇÃO

Uma aplicação móvel é um software projetado para ser executado em *smartphones*, *tablets* ou outros dispositivos móveis [1]. No entanto, para que seja bem sucedida, uma aplicação deverá permitir um envolvimento interativo com o utilizador de forma o mais expedita, intuitiva e portátil possível [2]. Originalmente, a maior parte das aplicações móveis foi concebida para fins informativos e de produtividade como, por exemplo, *e-mail*, calendário, contatos ou calculadora. Com a rápida evolução da tecnologia, o desenvolvimento das aplicações móveis foi expandido para outras categorias, tais como, jogos, localização (GPS) ou redes sociais. Atualmente é possível realizar quase todas as operações do dia-a-dia através de aplicações móveis, as quais oferecem funcionalidades idênticas ou superiores às aplicações tradicionais usadas nos PC, mas podendo ser acedidas em qualquer lugar, tirando ou não partido de uma ligação à Internet.

O número de aplicações publicadas vai aumentando a cada semestre [3] e, devido ao crescimento do mercado das aplicações móveis, as empresas querem construir aplicações mais eficientes e com mais funcionalidades, mas ao mesmo tempo querem manter a mesma sensação de desempenho de qualquer outra aplicação atualmente no mercado.

O grande problema surge quando se inicia o desenvolvimento e se percebe que é necessário configurar vários ambientes de desenvolvimento e aprender a programar duas ou mais linguagens totalmente distintas, Java ou Kotlin para Android e Objective-C ou Swift para iOS. Isto leva a que muitas empresas tenham equipas distintas de desenvolvimento Android e iOS, sendo o conhecimento dos membros normalmente limitado a uma plataforma. Embora o nível de especialização que se obtém nessas equipas seja benéfico em muitas situações, se fosse possível desenvolver aplicações Android e iOS numa única equipa e num único projeto isso poderia reduzir substancialmente o tempo e custos de desenvolvimento. Numa tentativa de mitigar estes problemas e agilizar o processo de desenvolvimento, tem-se assistido ao aparecimento de soluções híbridas e *cross-platform* que permitem atenuar alguns destes problemas. Em alguns casos ajudando na agregação de equipas para desenvolvimento conjunto num único projeto, o qual permitirá ter a aplicação disponível em vários sistemas operativos de plataformas móveis.

Neste contexto, já não são novidade as *frameworks* híbridas baseadas nas tecnologias *web* que permitem desenvolver

aplicações para sistemas diferentes, mas utilizando o mesmo código-fonte. O Phone Gap, Appcelerator ou Ionic, já existem há algum tempo, mas normalmente ficam aquém do exigido para os programadores, quer em termos de desempenho quer de experiência com o utilizador [4], [5], [6]. Esta situação surge porque existe uma *webview* intermédia que faz uma adaptação do código em HTML e CSS para executar no dispositivo móvel.

Num estudo anterior [6] foi realizada uma análise comparativa entre soluções híbridas JavaScript, nomeadamente React Native, NativeScript e Ionic, baseada em sete critérios considerados mais relevantes para o desenvolvimento de aplicações móveis. Este estudo concluiu que o React Native apresentava-se como a solução mais vantajosa para o desenvolvimento de aplicações móveis [6]. Assim sendo, a *framework* escolhida para esta avaliação experimental recaiu sobre o React Native. Devido à *framework* ser recente, não foram encontrados estudos relacionados que comparem o tempo de desenvolvimento em React Native relativamente às duas linguagens atualmente mais utilizadas para desenvolvimento nativo (Java em Android e Swift em iOS).

O React Native [7], criado pelo Facebook, tem como objetivo fornecer uma experiência de utilização similar à das aplicações nativas, tornando a execução possível em Android e iOS, sem ter de recorrer a uma *webview* intermédia para fazer toda a adaptação do código-fonte. Esta *framework* é bastante recente, mas vista como a solução de desenvolvimento híbrido mais emergente atualmente.

Antes de aparecer o React Native foi criado o ReactJS, que é uma *framework* de JavaScript construída pelo Facebook, a qual foi lançada no ano de 2013 [8]. O ReactJS é uma *framework* que vai ao encontro da visão principal de desenvolvimento do Facebook [9], que é a de reduzir o tempo de desenvolvimento, optando pela criação de código reutilizável e eficiente. Desta forma o ReactJS tem por objetivo principal a divisão de todas as vistas em diferentes componentes para ser possível a reutilização de código para outras vistas do projeto ou até mesmo em projetos totalmente diferentes, levando a um ciclo de desenvolvimento mais rápido. Em 2015 e após dois anos do lançamento do ReactJS, na React.js Conf 2015, Tom Occhino [10] anunciou que, com o sucesso que o ReactJS estava a ter no desenvolvimento de aplicações para a *web*, iria ser preparada uma solução para desenvolvimento de aplicações móveis, o React Native. A ideia era a de acabar com a desconexão total no desenvolvimento entre Android e iOS, mas sem a necessidade de recorrer à utilização de *webviews* [10]. O React Native teve a sua primeira aparição em 2015, contudo, em versões com algumas lacunas. Esta situação deve-se ao facto de existir uma maior dificuldade em criar uma *framework* de desenvolvimento híbrido sem recorrer às já mencionadas *webviews*. Atualmente ainda não existe uma versão estável da *framework*, contudo a maior parte dos problemas já foi ultrapassada, estando a *framework* a entrar numa fase de estabilização e com poucas alterações. A lógica da aplicação é gravada em JavaScript enquanto a vista é totalmente nativa, havendo sempre a possibilidade de implementação de código nativo para cada uma das plataformas.

Atualmente, com o grande ecossistema que existe de JavaScript em *frontend web* [11], são cada vez mais procuradas soluções para *backend*, *mobile* ou *desktop* para JavaScript, daí a terem surgido novas alternativas para desenvolvimento móvel de forma híbrida em JavaScript como é o caso deste estudo que compara o React Native com as linguagens as duas principais de desenvolvimento nativo tanto para Android como para iOS.

O estudo realizado é reportado neste documento da seguinte forma. Na seção II é apresentado o tipo de aplicação em que se vai basear a comparação e a metodologia utilizada para a comparação entre propostas de soluções. Na seção III são apresentados os resultados do desenvolvimento das funcionalidades e tarefas para cada uma das plataformas de desenvolvimento e na seção IV é feita uma discussão dos resultados. Por fim na seção V são apresentadas as conclusões e propostas algumas ideias para trabalho futuro.

II. METODOLOGIA DA AVALIAÇÃO EXPERIMENTAL

Para ser feita uma comparação das vantagens e desvantagens do desenvolvimento híbrido em React Native com o desenvolvimento nativo em Java e Swift foram criados três projetos distintos para a mesma solução, neste caso concreto a criação de áudioguias.

No caso do desenvolvimento nativo foi criada uma aplicação em Swift, para o sistema operativo iOS, e a mesma aplicação em Java, para executar no sistema operativo Android. Na criação das aplicações esteve sempre presente o objetivo de replicar ao máximo as aplicações para oferecer as mesmas funcionalidades, independentemente do sistema operativo usado pelo utilizador final da aplicação. Este é um cuidado normalmente associado ao desenvolvimento nativo que no desenvolvimento híbrido não acontece. Posteriormente, foi desenvolvida em React Native uma aplicação para o mesmo modelo de negócio, com as mesmas funcionalidades das anteriores. A escolha do tipo de aplicação, áudioguias, justifica-se pelo facto de já estarem a ser trabalhadas num contexto empresarial. Justifica-se também pelo facto de estar a ser realizada uma mudança de paradigma do desenvolvimento da própria empresa, de soluções nativas para uma solução híbrida. Assim, a escolha das áudioguias para projeto inicial permite fazer uma comparação das vantagens e desvantagens de implementação nas diferentes linguagens/*frameworks*.

Uma áudioguia tem como principal objetivo auxiliar um turista de uma região ou visitante de museus/exposições a ter uma visita guiada de forma autónoma. Estes serviços podem ser totalmente gratuitos ou, em alguns casos, com custos mais baixos em relação a um guia turístico tradicional. As funcionalidades mais comuns de uma áudioguia são:

- **Mapa**: serve para localizar e/ou dar orientações gerais ao utilizador;
- **Pontos de interesse (POI)**: permite identificar zonas de interesse para o utilizador, recorrendo a descrições, galeria de imagens ou áudio;
- **Visitas ou orientações**: permite a obtenção de orientações de navegação até um determinado POI;

- **Favoritos**: possibilita o regresso a um determinado POI de forma mais simples e direta;
- **Pesquisa**: permite efetuar pesquisas através de texto ou informações contextuais (p. ex., QRCode), dando a possibilidade de aceder aos POI mais rapidamente.

Para realizar a comparação entre desenvolvimento nativo e híbrido em React Native foram selecionadas algumas das funcionalidades mais importantes na implementação, para o modelo de negócio em questão. Posteriormente, foi feito o desenvolvimento em todas as linguagens/*frameworks* consideradas neste estudo de forma a serem comparados os tempos de desenvolvimento e as vantagens e desvantagens associadas ao desenvolvimento de cada uma das aplicações. As funcionalidades selecionadas foram divididas em tarefas mais pequenas. Assim, as funcionalidades e tarefas analisadas para a comparação foram:

- **Galeria e *swipe* de imagens**: os aspetos analisados nesta funcionalidade são os relativos à manipulação de imagens, desde o *download* a partir de um servidor, até à criação da vista de galeria de um conjunto de imagens em lista. As tarefas associadas a esta funcionalidade são:
  - Obtenção de uma lista de imagens através de uma REST API;
  - Armazenamento local de imagens para posterior utilização em modo *offline*;
  - Criação de galeria de imagens que permita a navegação através de gestos do tipo *swipe*.

- **Leitura de QRCodes e navegação:** nesta funcionalidade são analisados os aspetos relativos à gestão de permissões da câmara, para leitura e descodificação de QRCodes para texto, e navegação para uma vista relacionada com os dados lidos através do QRCode, sendo esta funcionalidade dividida nas seguintes tarefas:
  - Configuração de permissões associadas à câmara;
  - Leitura de QRCode para formato texto;
  - Navegação para a vista associada ao QRCode identificado.

- **Gestão de favoritos:** com a implementação dos favoritos é possível analisar a facilidade de mudança de uma vista consoante a ação do utilizador (tirar ou remover dos favoritos) e, também, a facilidade de implementar essas repercussões de alteração de dados globais a outras vistas que utilizam os mesmos dados, sendo que as duas tarefas analisadas associadas a esta funcionalidade são:
  - Atualização da vista após mudança de estado de favoritos;
  - Atualização de vistas globais após mudança de estado numa vista particular.

- **Carregamento e execução de áudio:** nesta funcionalidade é observada a forma de manipulação de ficheiros de áudio, desde o seu *download* até à sua execução e controlo, dando origem à análise das seguintes tarefas:
  - Obtenção de ficheiro de áudio através de uma REST API;
  - Execução de ficheiro de áudio;
  - Controlo da execução de áudio;
  - Vista associada ao estado de reprodução do ficheiro de áudio.

- **Lista dinâmica de dados:** com esta funcionalidade é possível verificar a facilidade de implementação de listas de dados, que incluam uma imagem alusiva de um ponto e texto com o nome do ponto. As tarefas analisadas foram:
  - Criação de lista com itens de imagem e texto;
  - Reutilização da mesma lista para implementação numa outra vista distinta.

- **Obtenção de dados e armazenamento:** nesta funcionalidade é investigada a interação com uma REST API, para obtenção de dados e facilidade de armazenamento dos dados para posterior utilização em modo *offline*, dando origem à análise das seguintes tarefas:
  - Preparação do ambiente para suportar armazenamento de modelos de dados;
  - Criação de um primeiro modelo de dados referente a pontos de interesse;
  - Obtenção de dados através de uma REST API.

III. Resultados

O desenvolvimento das aplicações foi realizado seguindo a metodologia SCRUM, que é uma estrutura que permite abordar problemas adaptativos de forma iterativa e incremental [12]. Durante o desenvolvimento, todos os tempos associados às várias tarefas foram registados logo após o seu término. Assim, foi possível obter os tempos para todas as soluções analisadas na comparação. Os resultados em tempo, vantagens e desvantagens para cada uma das soluções foram os seguintes:

- **Galeria e *swipe* de imagens**: na criação de uma galeria de imagens o React Native obteve vantagens na configuração do método de obtenção de ficheiros via REST API, devido a todo o desenvolvimento em React Native ser baseado em objetos JSON, ao contrário das soluções nativas que apresentam maior dificuldade no mapeamento de dados em JSON. Enquanto as soluções nativas apresentam facilidade em realizar *caching* de imagens o React Native necessita de uma biblioteca externa (react-native-fs) que complementa o *caching* de imagens de forma a ser compatível com as duas plataformas (Android e iOS). Na criação da vista de lista de imagens, com possibilidade de navegação através de gestos de *swipe*, os tempos obtidos foram aproximados, tal como demonstra a Tabela 1. Para as três soluções consideradas existe bastante documentação associada ao componente de *swipe* de imagens.

TABELA 1.TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE GALERIA E SWIPE DE IMAGENS

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Obtenção de uma lista de imagens através de uma REST API	4h	5h	1h
Armazenamento local de imagens para posterior utilização	0h 30m	0h 30m	3h
Criação de vista dinâmica de galeria de imagens com funcionalidade de *swipe*	3h	3h 45m	4h

- **Leitura de QRCodes e navegação:** na gestão das permissões para acesso à câmara fotográfica do dispositivo, no Swift basta configurar/disponibilizar uma *string* com o pedido a ser apresentado ao utilizador e o sistema operativo gere as permissões diretamente em *runtime*. O Android a nível de permissões exige um processo mais complicado, sendo tratadas de diferentes formas consoante a versão. No caso de configuração de permissões, o desenvolvimento híbrido em React Native não apresenta vantagens em relação ao desenvolvimento nativo devido a ter de tratar os dois sistemas operativos de forma separada conforme as suas particularidades. A descodificação de um QRCode para formato de texto e a navegação para a vista associada ao QRCode revelaram tempos de desenvolvimento idênticos para cada uma das soluções. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 2.

TABELA 2.TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE LEITURA DE QRCODES E NAVEGAÇÃO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Configuração de permissões associadas à câmara	0h 15m	2h	3h 30m
Leitura e descodificação de QRCode para formato texto	3h	4h	3h 30m
Navegação para a vista associada ao QRCode identificado	2h	0h 30m	0h 45m

- **Gestão de favoritos:** o React Native apresenta maior rapidez na criação de métodos que utilizam diretamente o modelo de dados, pois armazena diretamente os dados em JSON. As linguagens nativas, após uma boa definição da arquitetura do projeto, permitem que a atualização global das vistas após a mudança num modelo de dados seja feita de forma agilizada e automática. No caso do React Native é necessário utilizar o ciclo de vida do estado do componente para comparação dos dados que vêm do modelo (*store*). O React Native ao basear-se numa visão de desenvolvimento de máxima reutilização, apresenta a desvantagem de ser difícil em certos momentos controlar a imutabilidade dos dados entre modelo e vistas que já foram instanciadas. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 3.

TABELA 3.TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE GESTÃO DE FAVORITOS

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Atualização da vista após mudança de estado de favoritos	2h	2h 30m	1h 30m
Atualização global após mudança de estado numa vista particular	0h	0h	2h 30m

- **Carregamento e execução de áudio:** o desenvolvimento nativo apresentou maior lentidão na obtenção de ficheiros de áudio devido a algumas extensões de áudio. Por exemplo, os ficheiros ".wav" obrigam à realização de configurações extras dos metadados da API, dificuldade essa que o React Native não apresentou. A manipulação de áudio apresentou-se mais rápida em Swift e em React Native, devido a permitirem trabalhar de modo simplificado com a *Event Based API* (arquitetura de cliente/servidor com interações baseadas em eventos), algo que foi mais lento de implementar em Java. A nível de controlo da vista consoante o progresso do áudio, o desenvolvimento em React Native foi mais demorado devido a uma maior dependência de bibliotecas externas, relativamente às plataformas nativas, e também pela menor documentação fornecida. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 4.

TABELA 4.TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE CARREGAMENTO E EXECUÇÃO DE ÁUDIO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Obtenção de ficheiros de aúdio através de uma REST API	8h	6h	3h 30m
Execução de ficheiro de aúdio	1h	2h	1h 30m
Controlo da execução do ficheiro de áudio	2h	2h 30m	2h 30m
Vista associada ao estado de reprodução do ficheiro de aúdio	6h	7h	9h

- **Lista dinâmica de dados:** o sistema operativo iOS com o Swift permite reutilização e criação de *rows* (item de uma lista de dados iterativa), sendo rápido implementar listas dinâmicas. No Java, através dos adaptadores, também é rápido implementar a lista, contudo para o mesmo tipo de lista tornou-se um processo ligeiramente mais demorado em comparação com os tempos do Swift. O React Native apresentou o desenvolvimento de listas dinâmicas mais rápido devido a utilizar um componente de *flatlist* que executa o mesmo item da lista (*row*) como um componente independente, fazendo com que seja apenas necessário fornecer um JSON como propriedade da *flatlist* e o componente que a mesma vai instanciar. A nível de reutilização, todas as soluções permitiram uma reutilização da mesma lista noutra vista, de forma rápida e eficaz. Os

tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 5.

TABELA 5. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE LISTAS DINÂMICAS DE DADOS

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Criação de lista com itens de imagens e texto	1h	1h 30m	0h 45m
Reutilização da mesma lista para implementação em outra vista distinta	0h 15m	0h 15m	0h 15m

- **Obtenção de dados e armazenamento:** o Swift e o Java apresentaram tempos de desenvolvimento idênticos na preparação de ambiente de persistência e obtenção de dados através das bibliotecas já utilizadas, existindo bastante documentação associada. A principal dificuldade, não só no Swift como também no Java, foi o mapeamento dos dados de uma REST API em JSON para as devidas estruturas. Pelo contrário, o React Native apresentou tempo de desenvolvimento inverso, pois o React é focado no desenvolvimento de vistas e isso leva a ter de existir instalações e integrações para persistência de dados, como por exemplo o Redux, levando a um maior tempo de preparação do ambiente de armazenamento de dados. A obtenção de dados através de uma REST API e o seu mapeamento são feitos de forma rápida, pois os modelos de dados (*reducers*) em React Native são armazenados no formato de JSON diretamente. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 6.

TABELA 6. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE OBTENÇÃO DE DADOS E ARMAZENAMENTO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Preparação do ambiente para suportar armazenamento de modelos de dados	6h	5h 30m	9h
Criação de um primeiro modelo de dados referente a pontos de interesse	0h 15m	0h 30m	1h 15m
Obtenção de dados referentes através de uma REST API	3h 30m	4h	1h 30m

IV. DISCUSSÃO DOS RESULTADOS

Neste trabalho fornecemos a comparação em termos de tempo de implementação de três diferentes soluções para programação de aplicações móveis. As soluções foram analisadas através do tempo de implementação do mesmo tipo de projeto (áudioguias).

Os resultados obtidos neste estudo estão de acordo com as conclusões alcançadas em outros estudos comparativos realizados entre as soluções nativas, Java e Swift, em que relatam que o desenvolvimento em Java para Android pode ser mais lento entre 20% a 30% em relação ao Swift para iOS [13]. Na análise realizada o Java apresentou-se cerca de 11.1% mais lento do que o Swift.

Como foi constatado, a solução que apresentou menor tempo de desenvolvimento foi o Swift, notando-se também, a par com o React Native, que a quantidade de código feito era consideravelmente menor comparando com o Java, levando a maior facilidade de manuseamento em classes com complexidade de lógica maior. Esta situação justifica em parte a pretensão da Google em transferir o desenvolvimento Android para Kotlin, uma linguagem muito mais próxima em termos de sintaxe com o JavaScript do React Native e o Swift do iOS.

Na Tabela 7 são apresentados os tempos totais das funcionalidades desenvolvidas para todas as soluções.

TABELA 7. TEMPOS DE DESENVOLVIMENTO TOTAL DE TODAS AS FUNCIONALIDADES PARA CADA SOLUÇÃO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Total	42h 45m	47h 30m	48h

V. CONCLUSÕES E TRABALHO FUTURO

O desenvolvimento de aplicações móveis encontra-se em enorme evolução e atrai um grande interesse de várias organizações a par do que tem acontecido com todo o mercado dos *smartphones*.

No estudo realizado para comparar o desenvolvimento nativo (Swift e Java) relativamente ao desenvolvimento híbrido (React Native) concluiu-se que o desenvolvimento híbrido em React Native apresentou o pior tempo de desenvolvimento, mas, contudo, similar ao Java. O resultado final do desenvolvimento em React Native foi uma aplicação multiplataforma, para Android e iOS, enquanto no Swift e no Java apenas foi uma aplicação para cada um dos sistemas operativos, iOS e Android respetivamente. Daí também a implementação em React Native ter um tempo maior. Neste desenvolvimento com o React Native não foi necessário recorrer à implementação em código nativo, embora tenham surgido configurações individuais para cada sistema operativo, como por exemplo no que concerne às permissões. Na instalação de bibliotecas no React Native também foi necessário realizar configurações diferentes para cada um dos sistemas operativos (não apenas as configurações do *gradle* no Android em Java ou nos *pods* no iOS para Swift).

Para pequenas e médias empresas, onde a existência de equipas individuais especializadas no desenvolvimento nativo para iOS e Android é financeiramente inviável, o React Native pode ser uma solução vantajosa para o desenvolvimento móvel, já que é possível fazer aplicações para ambos os sistemas operativos em quase metade do tempo. O maior tempo de desenvolvimento que poderá ser constatado nos primeiros projetos realizados por programadores com experiência apenas em *mobile* (Swift e Java) é rapidamente ultrapassado após os programadores perceberem a arquitetura associada ao React

Native, principalmente se já tiverem experiência em Swift pois é uma linguagem com uma sintaxe mais próxima do JavaScript.

De referir que, através da arquitetura utilizada pelo React Native, as diferenças de desempenho e de utilização são quase impercetíveis entre as aplicações híbridas e as nativas, daí a vasta quantidade de aplicações já criadas em React Native como, por exemplo, Instagram, Skype, UberEats, Tesla [14]–[17].

Para empresas que pretendam mudar o seu desenvolvimento móvel para uma solução de desenvolvimento móvel JavaScript, o React Native apresenta-se como uma das melhores soluções atuais [6]. Esta opção permite criar uma equipa mais homogénea com experiência em desenvolvimento móvel, possivelmente também com experiência em desenvolvimento *frontend* JavaScript, caso este seja realizado em ReactJS. A construção de uma equipa com estas valências tornará o desenvolvimento mais eficiente e com maior qualidade, pois o conhecimento dos aspetos do desenvolvimento móvel aliado ao conhecimento de todas as arquiteturas e boas práticas utilizadas no React Native dará uma curva de aprendizagem melhor à equipa de desenvolvimento. Esta foi uma das técnicas usadas na constituição da equipa de desenvolvimento móvel hibrido realizada pelo Airbnb [18].

Como trabalho futuro, propomos uma avaliação experimental a utilizadores comuns de aplicações móveis, em que são apresentadas quatro versões para uma mesma aplicação, duas para cada plataforma (Android e iOS): uma versão nativa em Java (Android), uma versão nativa em Swift (iOS), uma versão em React Native para Android e outra desenvolvida em React Native para iOS (as versões React Native para Android e iOS tem por base o mesmo projeto). A ideia será a de aferir a satisfação do utilizador tanto a nível de utilização como de resposta à utilização para cada uma das versões, de forma a perceber se são encontradas diferenças entre uma implementação híbrida e uma implementação nativa.

Referências Bibliográficas

[1] V. N Inukollu, D. D. Keshamon, T. Kang, and M. Inukollu, "Factors Influncing Quality of Mobile Apps: Role of Mobile App Development Life Cycle," *Int. J. Softw. Eng. Appl.*, vol. 5, no. 5, pp. 15–34, 2014.

[2] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation," *2015 2nd ACM Int. Conf. Mob. Softw. Eng. Syst.*, pp. 56–59, 2015.

[3] Statista, "App Download and Usage Statistics 2017," 2017. [Online]. Available: http://www.businessofapps.com/data/app-statistics/. [Accessed: 14-Jan-2018].

[4] M. Palmieri and A. Cicchetti, "Comparison of Cross-Platform Mobile Development Tools," in *16th International Conference on Intelligence in Next Generation Networks*, 2012, pp. 179–186.

[5] N. C. Zakas, "The Evolution of Web Development for Mobile Devices," *ACM Digit. Libr.*, pp. 1–10, 2017.

[6] Brito Hugo, Gomes Anabela, Santos Álvaro, Jorge Bernardino, "JavaScript em aplicações móveis: React Native vs Ionic vs NativeScript vs desenvolvimento nativo," *CISTI*, 2018.

[7] Facebook, "React Native Documentation," 2019. [Online]. Available: https://facebook.github.io/react-native/docs/getting-started.html. [Accessed: 15-Dec-2019].

[8] Facebook, "React," 2018. [Online]. Available: https://reactjs.org/. [Accessed: 15-May-2018].

[9] I. Suzdalnitski, "React.js: a better introduction to the most powerful UI library ever created," 2018. .

[10] T. Occhino, "React JS conf," 2015. [Online]. Available: https://code.facebook.com/videos/786462671439502/react-js-conf-2015-keynote-introducing-react-native-/. [Accessed: 15-May-2018].

[11] AlexSoft, "JavaScript Ecosystem," 2018. [Online]. Available: https://www.altexsoft.com/blog/engineering/javascript-ecosystem-38-tools-for-front-and-back-end-development/. [Accessed: 10-Feb-2019].

[12] SCRUM, "What is SCRUM?," 2018. [Online]. Available: www.scrum.org/. [Accessed: 10-Jun-2018].

[13] TomislavCar, "Android development is 30% more expensive than iOS. And we have the numbers to prove it!," 2015. [Online]. Available: https://infinum.co/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios. [Accessed: 31-Dec-2017].

[14] N. Singh, "An comparative analysis of Cordova Mobile Applications V/S Native Mobile Application," *Int. J. Recent Innov. Trends Comput. Commun.*, pp. 3777–3782.

[15] H. Heitk, S. Hanschke, and T. A. Majchrzak, "Evaluating Cross-Platform Development Approaches for Mobile Applications," *Lect. Notes Bus. Inf. Process.*, vol. 140, pp. 120–138, 2013.

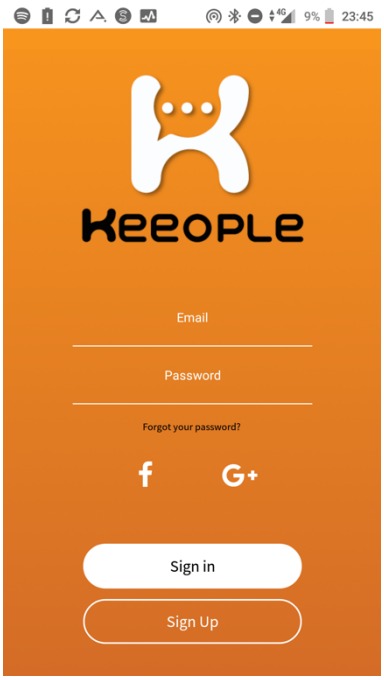
[16] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," *Proc. 6th Balk. Conf. Informatics - BCI '13*, p. 213, 2013.

[17] R. Native, "Who's using React Native?," 2017. [Online]. Available: http://facebook.github.io/react-native/showcase.html. [Accessed: 30-Dec-2017].

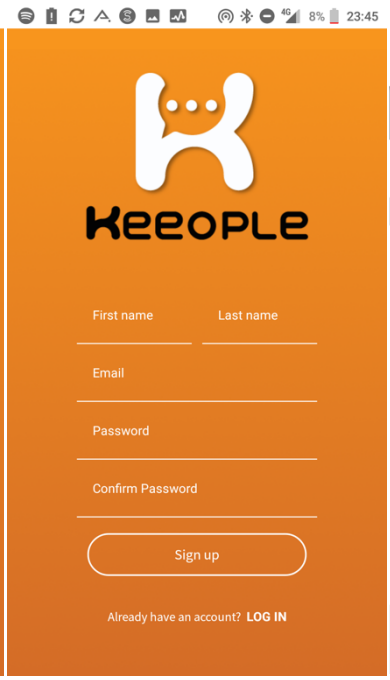
[18] G. Peal, "Building a Cross-Platform Mobile Team," 2018. [Online]. Available: https://medium.com/airbnb-engineering/building-a-cross-platform-mobile-team-3e1837b40a88. [Accessed: 10-Feb-2019].

Anexo L – Resultado final da aplicação Connects

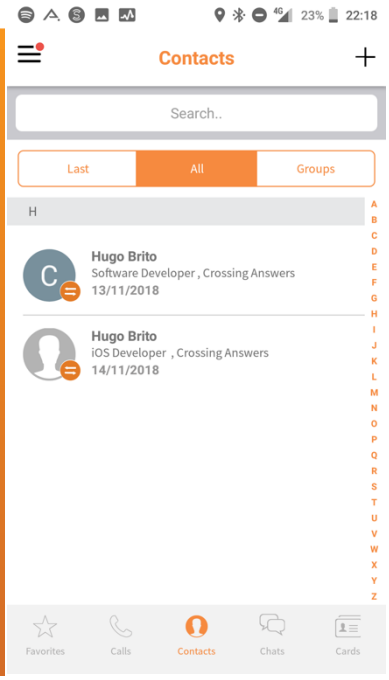
O resultado final das aplicações Connects criada em React Native foi o seguinte:



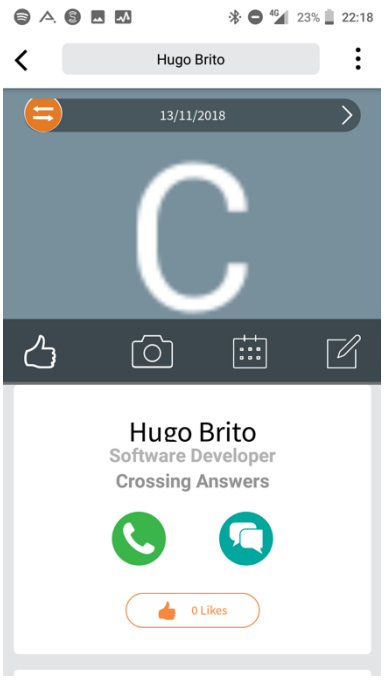
Ecrã de login da aplicação



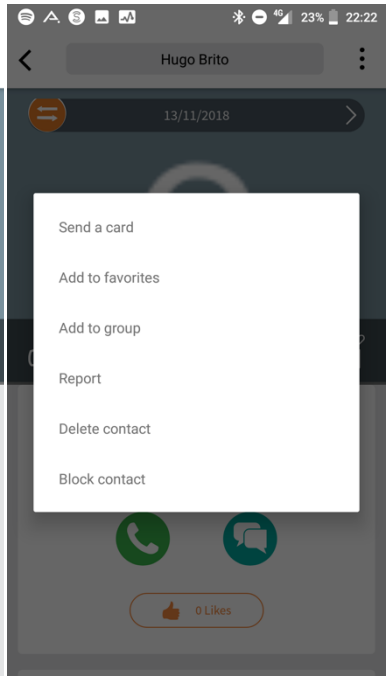
Ecrã de registo na aplicação



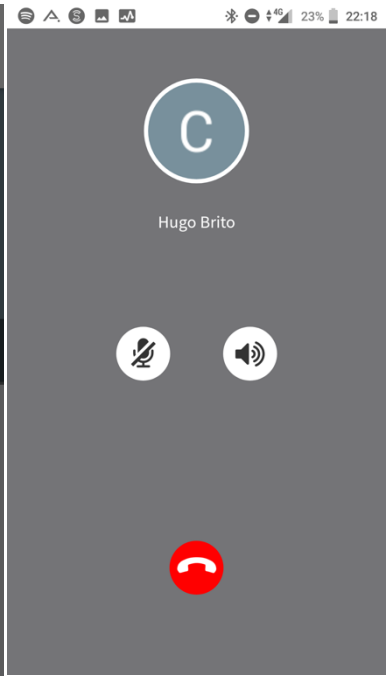
Ecrã principal da aplicação



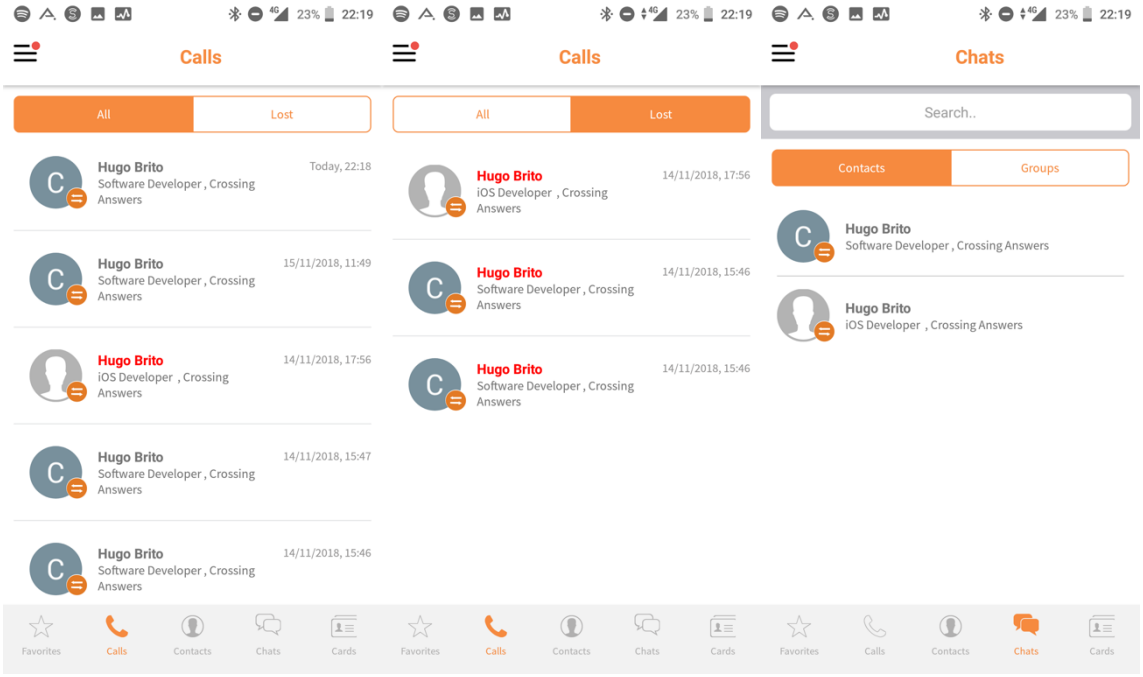
Ecrã de perfil de outro contato



Opções a realizar no contato



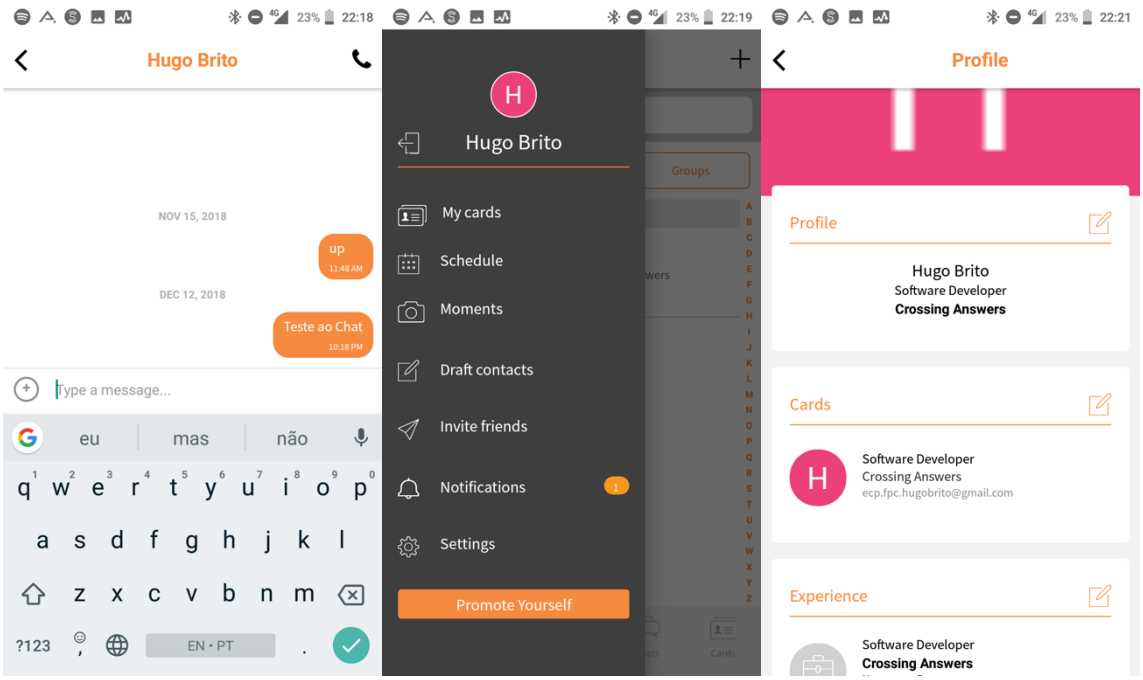
Ecrã de chamada



Ecrã de histórico de chamadas

Ecrã de chamadas perdidas

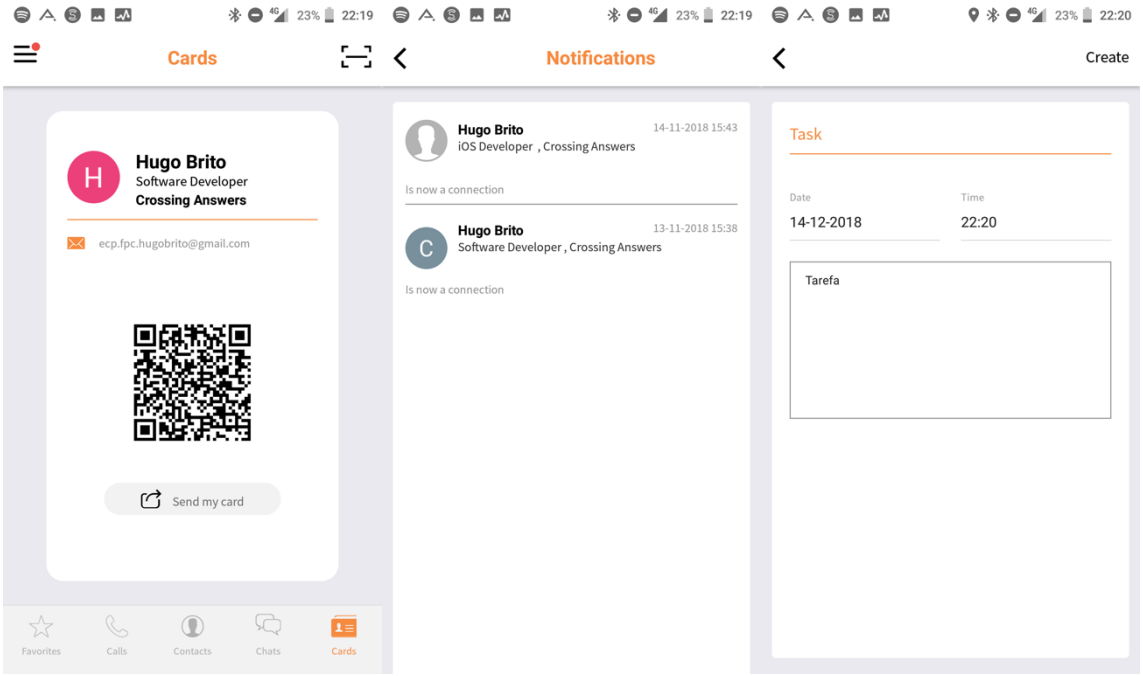
Ecrã lista de conversas no chat



Ecrã de chat

Menu lateral com opções principais (Drawer)

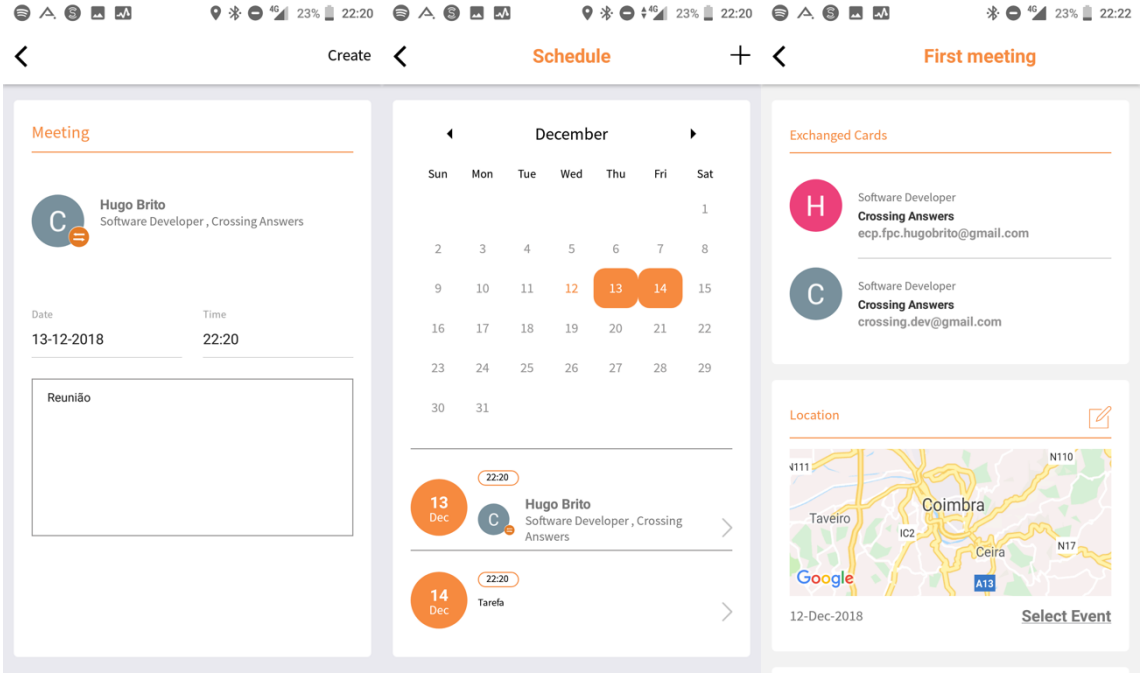
Perfil pessoal de utilizador



Cartão de informação pública

Notificações

Criação de tarefas



Criação de uma reunião

Calendário com eventos

Ecrã de informação de primeiro encontro entre dois contatos

Anexo M – Implementação de navegação através de mapbox offline

Implementação de Navegação Offline com Mapbox

1. Exemplos

1.1. Maps Me

Aplicação com a melhor solução a nível de mapas offline dando alguns exemplos no seguinte repositório (https://github.com/mapsme/omim).

2. Soluções

2.1. OSRM

A melhor solução passará sempre pela utilização dos mapas do projecto OSRM, que podem ser obtidos através http://download.geofabrik.de/

Para obter o mapa em formato XML, é necessário aceder ao site https://www.openstreetmap.org/#map=7/39.453/-9.789 (link para Portugal), selecionar a opção EXPORT e novamente EXPORT após selecionar a área de download do mapa.

2.2. Bibliotecas

2.2.1. Dijkstra's

É um algoritmo estabelecido que permite o cálculo de rotas obtendo a opção mais eficaz a partir de um ficheiro de nodes de uma certa região.

Assim uma das opções seria a implementação deste algoritmo adaptado às plataformas móveis, de forma a comunicar obter sempre os dados do recálculo das rotas.

2.2.2. Graphhopper

Através de uma implementação Java segundo a documentação (https://github.com/graphhopper/graphhoppe) , é neccessário fazer download do mapa em formato osm.pbf, executar a importação e instalação do mapa para posteriormente ser utilizado numa aplicação à medida

2.2.3. OSRM-Backend

Baseado em C++, é uma simulação de um servidor que pode segundo a documentação ser adaptado a mobile, contudo, certos dispositivos podem não suportar a execução de um serviço paralelo.

A ideia principal passaria pela passagem de um ficheiro do tipo OSRM para ARMv7 e depois uma compilação para uma biblioteca do tipo libosrm em C++ que permitisse integrar com os dois sistemas operativos móveis (Android/iOS)

3. Limitações e dificuldades

3.1. Integração de biblioteca de mapas

Uma das principais dificuldades passa por depois de ter a biblioteca pronta a utilizar, perceber como utilizar a mesma tanto no Android como no iOS, e como utilizar mais a alto nível no React Native.

3.2. Suporte iOS

Nunca foi testado num ambiente iOS, e a maioria da documentação existente é para Android.