

2015

Instituto Politécnico de Coimbra

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

iNavigator

Sistema móvel de suporte a tarefas diárias

MESTRADO EM INFORMÁTICA E SISTEMAS

AUTOR | Ricardo Jorge Lopes Cabete

ORIENTADOR | Doutor Simão Pedro Mendes Cruz Reis Paredes

Coimbra, dezembro 2015



Departamento de Engenharia Informática e de Sistemas

***iNavigator* – Sistema móvel de suporte a tarefas diárias**

Relatório de projeto apresentado para obtenção do grau de

Mestre em Informática e Sistemas

Autor

Ricardo Jorge Lopes Cabete

Orientador

Professor Doutor Simão Pedro Mendes Cruz Reis Paredes

Professor adjunto do ISEC

Coimbra, dezembro de 2015

DEDICATÓRIA

À minha família, pelo apoio incondicional desde o primeiro momento e
à Diana, por me inspirar sempre à transcendência.

Ricardo Cabete

AGRADECIMENTOS

A elaboração de um relatório de projeto representa, tipicamente, o final de uma etapa importante na vida de qualquer pessoa. O presente relatório não é exceção e testemunha o término de um curso de Mestrado iniciado há dois anos. Apesar do mérito da conclusão de um curso recair tendencialmente na figura de quem o termina, não é exatável que alguém consiga atingi-lo sozinho pelo que se torna impreterível agradecer àqueles que mais contribuíram para que tal acontecesse.

O espaço necessariamente limitado da secção de agradecimentos impede-me de expressar, de forma exaustiva, o meu agradecimento a quem, direta ou indiretamente me apoiou durante o curso e me impeliu sempre a continuar. Por este motivo foi imperativo encontrar um critério de seleção que limitasse o número de pessoas citadas, cientes que, qualquer que fosse o critério adotado, não é possível evitar a injustiça a que serão sujeitos os que ficaram de fora. O critério que se revelou mais adequado foi o de citar apenas as pessoas diretamente relacionadas com a realização deste projeto na sua dimensão pessoal e institucional. Assim, o meu sincero agradecimento:

Ao Professor Doutor Simão Pedro Mendes Cruz Reis Paredes pela orientação do projeto desde o primeiro momento. O meu obrigado pelo entusiasmo e profissionalismo na investigação de soluções *out of the box* e na total disponibilidade que sempre demonstrou.

Ao Professor Doutor Jorge Manuel Oliveira Henriques, uma voz sempre calma na tempestade que é a investigação e desenvolvimento de soluções não triviais para problemas só aparentemente simples.

A um grupo selecionado de Professores do ISEC, felizmente demasiado grande para listar individualmente mas que foram o espelho da competência, do profissionalismo e do zelo durante o tempo em que frequentei esta instituição.

Por fim, o meu agradecimento à minha família direta, pai, mãe e irmã que, desde o primeiro momento apoiaram incondicionalmente esta etapa de reinvenção profissional e, consequentemente, também pessoal. Foram 5 anos de esforço concertado e concentrado que implicaram uma longa permanência ininterrupta no estrangeiro e que agora terminam. A família indireta, tios, primos, avós e bisavós também merecem uma palavra pelas vidas tão ricas, interessantes e cheias de bravura que eu espero, mais do que tudo, conseguir honrar. Por fim, um agradecimento especial à Diana que, com a sua presença tranquilizadora me manteve motivado nas centenas de horas de estudo e não me deixou desesperar em momentos particularmente aborrecidos e desmotivadores. Sem ti teria acabado esta etapa com média inferior, em período temporal eventualmente superior mas, sobretudo, sem grande parte da diversão!

RESUMO

As tecnologias móveis podem desempenhar um papel particularmente relevante no apoio à realização de um conjunto de tarefas de complexidade variável por utilizadores com necessidades especiais. O objetivo deste trabalho prende-se com o desenvolvimento de um sistema, tendo por base um dispositivo móvel (*smartphone* ou *tablet*), para apoio a utilizadores com défices cognitivos ligeiros diretamente relacionados com memória e dificuldades de orientação.

Com efeito, os problemas cognitivos podem estar relacionados com perda de memória, atenção, raciocínio, orientação, fala, sendo que a incapacidade na realização de tarefas pode manifestar-se de forma muito diversa, desde o simples esquecimento na realização de tarefas até situações que podem colocar em risco a sua segurança como desorientação, falha na toma de medicação.

O projeto *iNavigator* teve por objetivo sistema capaz de apoiar o cidadão com deficiência cognitiva ligeira, tipicamente ligada à dificuldade de memória e de atenção ou às dificuldades de orientação, recorrendo para isso a uma aplicação móvel capaz de definir alarmes e localizar o utilizador.

O sistema pretende, através da definição de alarmes e da monitorização da localização, ser útil num contexto em que o utilizador revela problemas cognitivos que dificultam a realização de um conjunto de tarefas da sua vida quotidiana.

Este relatório descreve o trabalho realizado durante as várias fases de desenvolvimento do projeto *iNavigator* e que permitiram atingir plenamente os objetivos propostos, nomeadamente a análise do estado da arte e do carácter diferenciador desta

solução, a análise de requisitos, a justificação da seleção das tecnologias e ferramentas adotadas, a discussão da arquitetura e da implementação propriamente dita e, por fim, da verificação e validação do sistema.

Palavras-chave: déficit cognitivo, *iNavigator*, alarme, localização, aplicação web, aplicação móvel *Android*.

ABSTRACT

Mobile technologies may be exceptionally helpful in assisting day-to-day activities of users with special needs. This report focuses on the development of a mobile system capable of helping users with mild mental impairment that specifically affects memory and orientation and impedes the fulfilment of ordinary tasks that would easily be forgotten or executed in the wrong order or at the wrong time.

By defining alarms and monitoring the localization of users, the system aims to help users that struggle with everyday chores. This can be helpful in users that show signs of memory loss, attention deficit disorder, difficulty in reasoning or orientation and any sort of speech impairment. The consequences of these conditions can be variably adverse from simple forgetfulness to complete a minor task to incredibly catastrophic when the users put themselves in danger by getting disoriented or by forgetting life-keeping medication.

This report presents all the tasks done during the development of *iNavigator* that allowed the fulfilment of all project goals, namely state of the art analysis, requirements analysis, architecture and implementation discussion and, finally, the verification and validation of the system.

Keywords: mental impairment, *iNavigator*, alarm, localization, web application, Android mobile application.

DECLARAÇÃO DO AUTOR

O autor declara que o presente trabalho foi levado a cabo de acordo com os regulamentos do Departamento de Engenharia Informática e de Sistemas do Instituto Superior de Engenharia de Coimbra do Instituto Politécnico de Coimbra e que é original, exceto onde explicitamente indicado. Declara, ainda que este trabalho, no todo ou em parte, não foi apresentado para avaliação noutra instituição de ensino superior portuguesas ou estrangeiras. Quaisquer visões expressas são as do autor e não representam de modo nenhum as visões do Departamento de Engenharia Informática e de Sistemas do Instituto Superior de Engenharia de Coimbra ou do Instituto Politécnico de Coimbra.

Ricardo Jorge Lopes Cabete (a21210617)
15 de dezembro de 2015

ÍNDICE

1 Introdução	1
1.1 O iNavigator	1
1.2 Objetivos.....	2
1.3 Planeamento de trabalhos.....	4
1.4 Estrutura do relatório.....	4
2 Estado da arte	7
2.1 Aplicações de alarmes	8
2.2 Sistemas de localização.....	12
2.3 O iNavigator	14
3 Análise de requisitos.....	17
3.1 Documento de visão e âmbito.....	17
3.2 Requisitos funcionais.....	18
3.2.1 Aplicação web.....	18
3.2.2 Aplicação móvel <i>Android</i>	19
3.3 Requisitos não funcionais.....	20
3.4 Casos de uso.....	21
3.5 Regras de negócio.....	22
4 Tecnologias e ferramentas adotadas	25
4.1 A aplicação web do servidor.....	25
4.1.1 O stack <i>Apache</i> , <i>MySQL</i> e <i>php</i>	25
4.1.2 <i>php</i> e MVC.....	26
4.1.3 A <i>framework Laravel</i>	29
4.2 A aplicação móvel.....	31
4.2.1 <i>Xamarin</i>	32
4.2.2 <i>Android</i>	35
4.2.3 <i>Android</i> SDK e <i>Android Studio</i>	38
4.2.4 MQTT	38
5 Arquitetura	41
5.1 A aplicação web.....	41
5.1.1 Modelo	42
Especificação e manutenção da base de dados.....	42
O ficheiro do modelo.....	44
<i>Seeding</i> dos dados.....	45
O acesso aos dados.....	46
5.1.2 Controlador	48
Rotas	48

Controladores.....	49
Fachadas da <i>framework</i>	50
O <i>middleware</i> de HTTP.....	51
5.1.3 Vista.....	52
Layouts e master page.....	52
O motor de templating Blade.....	54
5.2 A aplicação móvel.....	56
5.2.1 Atividades.....	57
5.2.2 Serviços e <i>broadcast receivers</i>	58
5.3 A interação entre as aplicações.....	60
5.3.1 Comunicação Servidor -> Cliente e Cliente->Cliente.....	60
5.3.2 Comunicação Cliente -> Servidor e Cliente -> Cliente.....	63
6 Implementação.....	65
6.1 Metodologia.....	65
6.2 Funcionalidades da aplicação web.....	67
6.2.1 Autenticação, criação de conta e idiomas.....	67
6.2.2 A <i>dashboard</i>	69
6.2.3 Gestão de elementos.....	71
CRUD de pessoas.....	71
CRUD de categorias de alarmes.....	72
CRUD de alarmes.....	73
CRUD de dispositivos.....	75
6.2.4 Alarmes.....	76
Associação de alarmes a dispositivos.....	76
Publicação de alarmes.....	77
6.2.5 Dados do perfil.....	78
6.3 Funcionalidades da aplicação <i>Android</i>	79
6.3.1 Ambos os perfis – Autenticação.....	80
6.3.2 Perfil utente.....	81
Interação com alarmes.....	81
Menu do utente.....	82
Submenu SOS.....	83
6.3.3 Perfil vigilante – Menu vigilante.....	86
7 Verificação e validação.....	89
7.1 Testes.....	89
7.2 Validação pelo cliente.....	92
8 Conclusões e trabalho futuro.....	95
Bibliografia.....	99
Anexo I – Documento de visão e âmbito.....	105
1.1 Requisitos de negócio.....	105

1.1.1	Enquadramento	105
1.1.2	Objetivos de negócio e critérios de sucesso	106
1.1.3	Riscos do negócio	106
1.2	Visão da solução	107
1.2.1	Visão	107
1.2.2	Principais funcionalidades	107
1.2.3	Pressupostos e dependências	108
1.3	Âmbito e limitações	109
1.3.1	Âmbito das versões inicial e subsequentes	109
1.3.2	Limitações e exclusões	109
1.4	Contexto de negócio	110
1.4.1	Perfis dos stakeholders	110
1.4.2	Prioridades do projeto	110
Anexo II - Diagramas de casos de uso		111
1.1	Aplicação web – utilizador não autenticado	111
1.2	Aplicação web – utilizador autenticado	112
1.2.1	<i>Dashboard, logout</i> e edição de perfil	112
1.2.2	Alarmes e categorias de alarmes	113
1.2.3	Dispositivos e utilizadores	114
1.3	Aplicação móvel <i>Android</i> – perfil utente	115
1.4	Aplicação móvel <i>Android</i> – perfil vigilante	116
Anexo III - Casos de uso		117
1.1	Tabela de casos de uso da aplicação web	117
1.2	Tabela de casos de uso da aplicação cliente	118
1.3	Casos de uso da aplicação web do servidor	119
1.3.1	Fazer login	119
1.3.2	Criar conta de utilizador	119
1.3.3	Apresentar a <i>dashboard</i>	120
1.3.4	Publicar alarmes	120
1.3.5	Pedir relatório	121
1.4	Casos de uso da aplicação para dispositivos móveis	122
1.4.1	Definir configuração inicial	122
1.4.2	Definir configuração	122
1.4.3	Autenticar aplicação	123
1.4.4	Mostrar mensagem de erro	123
1.4.5	Mostrar alarme	124
1.4.6	Enviar estatísticas	124
1.4.7	Aceitar alarme	125
1.4.8	Adiar alarme	125
1.4.9	Aceder ao menu SOS	126
1.4.10	Enviar mensagem SMS ao vigilante	126

1.4.11	Obter localização GPS	127
1.4.12	Enviar coordenadas GPS ao vigilante	127
1.4.13	Visualizar página inicial do utente	127
1.4.14	Aceder ao menu notas	128
1.4.15	Registar nota fotográfica	128
1.4.16	Registar nota em vídeo	129
1.4.17	Registar nota sonora	129
1.4.18	Enviar mensagem SMS pré-definida	130
1.4.19	Enviar mensagem SMS personalizada	130
1.4.20	Telefonar ao vigilante	131
1.4.21	Visualizar página inicial do vigilante	131
1.4.22	Obter localização do cliente	132
1.4.23	Obter relatório de atividades do cliente	132
1.4.24	Telefonar ao utente	133
1.4.25	Visualizar localização do utente	133
1.4.26	Visualizar relatório de atividades do utente	134
Anexo IV – Modelo da base de dados		135
1.1	Modelo físico	135
Anexo V – Regras de negócio		137
Anexo VI – Maquetes de baixa fidelidade		139

ÍNDICE DE FIGURAS

Figura 1 - Diagrama geral do <i>iNavigator</i>	2
Figura 2 - Calendarização apresentada na proposta de projeto.....	4
Figura 3 - A lista de alarmes e os detalhes dos alarmes na aplicação <i>To Do Reminder</i>	10
Figura 4 - Lista de alarmes ativos na aplicação <i>Just Reminder</i>	11
Figura 5 - Lista de alarmes na aplicação <i>Simplest Reminder</i>	12
Figura 6 - Ecrã principal da aplicação Life360 num dispositivo Apple.....	13
Figura 7 - Ecrã com o mapa do percurso realizado no <i>Runtastic</i>	14
Figura 8 - Especificação do caso de uso fazer login.....	22
Figura 9 - Excerto da tabela de regras de negócio do <i>iNavigator</i>	23
Figura 10 - Diagrama do modelo MVC.....	28
Figura 11 - Gráfico da distribuição da cota de mercado das <i>frameworks php</i>	30
Figura 12 - Diagrama da divisão do código fonte em <i>Xamarin</i>	34
Figura 13 - Diagrama da arquitetura do sistema operativo <i>Android</i>	37
Figura 14 - Diagrama da comunicação no protocolo MQTT.....	39
Figura 15 - <i>Migration</i> responsável pela criação e eliminação de uma tabela.....	42
Figura 16 - Modelo responsável pela obtenção e recuperação de dados.....	44
Figura 17 - Um <i>seeder</i> com a especificação de um objeto do tipo Alarme.....	45
Figura 18 - Uma <i>query</i> em ORM com indicação de modelos adicionais a carregar.....	46
Figura 19 - Exemplo de criação, edição e destruição de um objeto do tipo Alarme.....	47
Figura 20 - Acesso a Alarmes condicionado a restrições aplicadas à tabela de Eventos.....	48
Figura 21 - Duas rotas iguais, resolvidas em função do método GET ou POST.....	48
Figura 22 - Uma rota que responde a um pedido DELETE.....	48
Figura 23 - Uma rota que responde a um pedido DELETE.....	49
Figura 24 - Excerto do controlador de Alarmes.....	50
Figura 25 - Exemplo de utilização da fachada <i>Auth</i>	51
Figura 26 - Excerto do <i>middleware</i> responsável pela autenticação do utilizador.....	52
Figura 27 - Excerto de um ficheiro da vista enriquecido com diretivas de <i>Blade</i>	53
Figura 28 - Explosão das subvistas que constituem uma vista final.....	54
Figura 29 - Anatomia de uma vista com as diretivas do motor de <i>templating Blade</i>	55
Figura 30 - Iteração de uma coleção com a diretiva <i>@foreach</i>	56
Figura 31 - Ciclo de vida das atividades no sistema operativo <i>Android</i>	58
Figura 32 - Figura 5 - Ciclo de vida dos serviços no sistema operativo <i>Android</i>	59
Figura 33 - Diagrama da cadeia de comunicação entre servidor e dispositivo móvel.....	62
Figura 34 - Vista do <i>Kanban</i> na ferramenta <i>Trello</i> da aplicação web do <i>iNavigator</i>	66
Figura 35 - Ecrã de login da aplicação web do <i>iNavigator</i>	68
Figura 36 - Vista geral da <i>dashboard</i> do <i>iNavigator</i>	69
Figura 37 - Vista do relatório de atividade.....	70
Figura 38 - Secção estatística da <i>dashboard</i>	70
Figura 39 - CRUD de pessoas do <i>iNavigator</i>	71
Figura 40 - Janela modal de confirmação à operação de eliminação de uma pessoa.....	72
Figura 41 - Janela modal de edição dos dados de uma pessoa.....	72
Figura 42 - CRUD das categorias de alarmes do <i>iNavigator</i>	73
Figura 43 - CRUD de alarmes do <i>iNavigator</i>	74

Figura 44 - Ecrã de edição dos atributos de um alarme.....	75
Figura 45 - CRUD de dispositivos do <i>iNavigator</i>	75
Figura 46 - Tabela onde se visualizam as associações entre alarmes e dispositivos.....	76
Figura 47 - Ecrã se define a associação entre alarmes e dispositivos.....	76
Figura 48 – Janela modal que aguarda a confirmação da publicação dos alarmes.....	77
Figura 49 – Ecrã de visualização e adição dos dados do perfil.....	78
Figura 50 – Ecrã de autenticação da aplicação móvel <i>Android</i>	80
Figura 51 – Ecrã de alarme do <i>iNavigator</i>	81
Figura 52 – Menu do <i>iNavigator</i> no perfil utente.....	82
Figura 53 – Menu SOS do <i>iNavigator</i> no perfil utente.....	83
Figura 54 – Submenu de envio de mensagens SMS.....	84
Figura 55 - Menu do <i>iNavigator</i> no perfil vigilante.....	86
Figura 56 – Ecrã de localização do dispositivo.....	87
Figura 57 - Primeira página do plano de testes do <i>iNavigator</i>	91
Figura 58 - Página dos testes à <i>dashboard</i> da aplicação web.....	92
Figura 59 - Diagrama de casos de uso da aplicação web.....	111
Figura 60 – Casos de uso respeitantes à <i>dashboard</i> , <i>logout</i> e perfil.....	112
Figura 61 - Casos de uso respeitantes a alarmes e categorias de alarmes.....	113
Figura 62 - Casos de uso respeitantes a dispositivos e utilizadores.....	114
Figura 63 – Casos de uso da aplicação <i>Android</i> – perfil utente.....	115
Figura 64 - Casos de uso da aplicação <i>Android</i> – perfil vigilante.....	116
Figura 65 - Ecrã de autenticação do servidor.....	139
Figura 66 – <i>Dashboard</i> do servidor.....	140
Figura 67 - Lista de utilizadores no servidor.....	140
Figura 68 – Criação de um utilizador no servidor.....	141
Figura 69 - Lista de alarmes no servidor.....	141
Figura 70 – Interação com um alarme no cliente.....	142
Figura 71 – Ecrã do cliente quando corrido <i>on-demand</i>	142
Figura 72 – Ecrã de seleção de mensagem SMS.....	143
Figura 73 – Menu de captura de notas multimédia.....	143

ABREVIATURAS, ACRÓNIMOS E SIGLAS

ACK	<i>Acknowledgement.</i>
AJAX	<i>Asynchronous JavaScript and XML.</i>
AMP	<i>Apache, MySQL, php.</i>
API	<i>Application Programming Interface.</i>
APP	<i>Aplicação.</i>
CSS	<i>Cascading Style Sheets.</i>
CRUD	<i>Create, Read, Update and Delete.</i>
DBMS	<i>Database Management System.</i>
GPS	<i>Global Positioning System.</i>
GRASP	<i>General Responsibility Assignment Software Patters (or Principles).</i>
GSM	<i>Global System for Mobile communications.</i>
HTML	<i>HyperText Markup Language.</i>
HTTP	<i>Hypertext Transfer Protocol.</i>
IDE	<i>Integrated Development Environment.</i>
IMEI	<i>International Mobile Equipment Identity.</i>
ISEC	<i>Instituto Superior de Engenharia de Coimbra.</i>
IoT	<i>Internet of Things.</i>
JIT	<i>Just-in-time.</i>
JS	<i>JavaScript.</i>
JSON	<i>JavaScript Object Notation.</i>
M2M	<i>Machine to machine.</i>
MQTT	<i>Message Queue Telemetry Transport.</i>
MVC	<i>Model-View-Controller.</i>
ORM	<i>Object-Relational Mapping.</i>
PDA	<i>Personal Digital Assistant.</i>
POJO	<i>Plain Old Java Object.</i>
SDK	<i>Software Development Kit.</i>
SMS	<i>Short Message Service.</i>
SVN	<i>Apache Subversion.</i>
UI	<i>User Interface.</i>
XML	<i>eXtensible Markup Language.</i>

1 INTRODUÇÃO

No âmbito do segundo ano do curso de Mestrado em Informática e Sistemas na especialização de Desenvolvimento de Software do Departamento de Engenharia Informática e de Sistemas do Instituto Superior de Engenharia de Coimbra do Instituto Politécnico de Coimbra está prevista, optativamente, a realização de estágio curricular ou o desenvolvimento de projeto industrial.

A opção entre as duas hipóteses é da responsabilidade do aluno e, no meu caso, optei pela realização de um projeto apresentado pelo professor doutor Simão Paredes e que visava dar resposta a um problema concreto ao mesmo tempo que permitiria aprofundar os conhecimentos abordados nas várias unidades curriculares frequentadas durante o curso.

1.1 O *iNAVIGATOR*

O projeto *iNavigator* consistiu no desenvolvimento de um sistema capaz de apoiar o cidadão com deficiência cognitiva ligeira, tipicamente ligada à dificuldade de memória e de atenção ou às dificuldades de orientação, recorrendo para isso a uma aplicação móvel capaz de definir alarmes e localizar o utilizador.

O *iNavigator* consiste em duas aplicações de tipologia diferente: uma aplicação web, usada como servidor e que permite criar configurações de alarmes e uma aplicação para dispositivos móveis, que comunica com a aplicação web para receber uma configuração e que depois funciona autonomamente.

A divisão do sistema em duas tipologias de aplicações tem a vantagem que uma terceira entidade, designada neste relatório por vigilante, possa programar os alarmes que devem ser seguidos e, posteriormente, seguir o seu cumprimento.

A imagem seguinte ilustra o funcionamento geral do sistema. Note-se que, apesar da representação de um cliente *Android* para o vigilante e outro para o utente, eles fazem uso da mesma aplicação, conforme será descrito nos capítulos dedicados à arquitetura e à implementação.

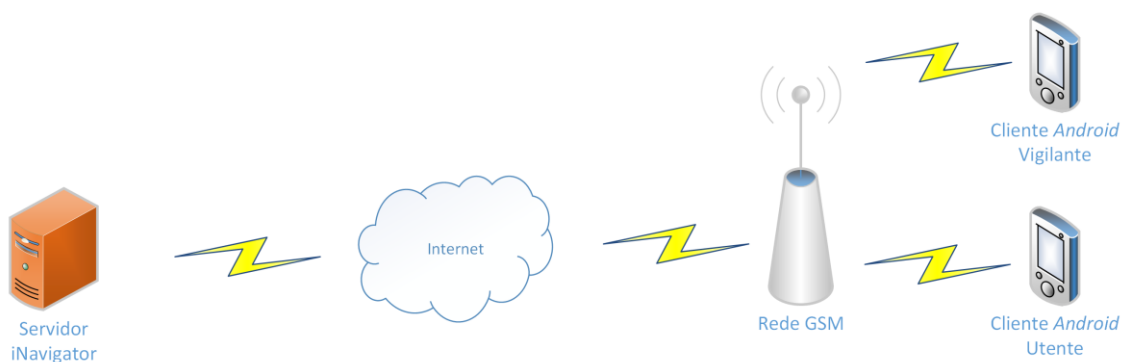


Figura 1 - Diagrama geral do *iNavigator*.

Da análise do diagrama pode verificar-se que cada componente do sistema é dependente da ligação à Internet ou à rede GSM. Esta dependência, porém, só acontece no momento de configuração dos clientes: durante a utilização das aplicações móveis não há necessidade de ligação a qualquer rede exceto quando é necessária comunicação.

1.2 OBJETIVOS

Na proposta do projeto *iNavigator* foi definido como objetivo a implementação de um sistema modular que verificasse as melhores práticas de interação pessoa máquina e que fosse composto por:

- Um módulo de gestão centralizada do sistema implementado numa aplicação web.
- Um módulo de acompanhamento e apoio do doente implementado num dispositivo móvel que permita ao prestador de cuidados (designado genericamente por vigilante):
 - a definição e gestão de lembretes para o utente;
 - o acompanhamento do histórico de atividades realizadas;
 - a localização em tempo real do utente.
- Um módulo implementado num dispositivo móvel que permita ao utilizador (designado genericamente por utente) reconhecer e interagir com os lembretes definidos:
 - o utilizador deve ter a possibilidade de ouvir o lembrete e interagir de forma simples e intuitiva.
 - o módulo deve explorar os sensores do equipamento.
- O sistema deve ser modular de forma a permitir a sua expansão futura.

Apesar de não estar explicitamente definido na proposta como objetivo, durante as primeiras reuniões de projeto surgiu a ideia de tentar criar uma aplicação móvel unificada, capaz de ser executada em sistema operativo *Android*, *Apple iOS* e *Windows Phone*.

Esta aplicação universal seria uma mais-valia significativa uma vez que tornaria o *iNavigator* num sistema muito mais abrangente mas implicaria a exploração de tecnologia inovadora. A exploração desta tecnologia e as razões para o seu eventual abandono são apresentadas no capítulo 4.

1.3 PLANEAMENTO INICIAL

Aquando da proposta inicial do projeto foram identificadas 7 tarefas principais:

- **T1.** Elaboração do estado da arte relativo a sistemas móveis de apoio a doentes com défice cognitivo ligeiro.
- **T2.** Análise de requisitos.
- **T3.** Estudo de tecnologias.
- **T4.** Design da arquitetura do sistema e especificação de testes.
- **T5.** Desenvolvimento e teste do sistema.
- **T6.** Testes em ambiente simulado e real.
- **T7.** Escrita da dissertação.

As tarefas identificadas foram calendarizadas de acordo com a tabela seguinte.

Part-time						Full-time				
	Mês 1	Mês 2	Mês 3	Mês 4	Mês 5	Mês 6	Mês 7	Mês 8	Mês 9	Mês 10
T1										
T2										
T3										
T4										
T5										
T6										
T7										

Figura 2 - Calendarização apresentada na proposta de projeto.

1.4 ESTRUTURA DO DOCUMENTO

Este relatório apresenta o trabalho desenvolvido e está organizado da seguinte forma:

O segundo capítulo compreende a análise do estado da arte e apresenta aplicações concorrentes que se propõem resolver partes do problema que deu origem ao *iNavigator* sem, no entanto, descurar a apresentação do seu carácter diferenciador.

O terceiro capítulo será dedicado à análise de requisitos. Será apresentado o documento de visão e âmbito e identificados os requisitos funcionais e não funcionais quer da aplicação web, quer da aplicação móvel *Android*. São também especificados os atores do sistema e os respetivos casos de uso e, por fim, as regras de negócio do *iNavigator*. Por restrições de espaço estes documentos foram remetidos para anexo.

No capítulo quatro abordam-se as tecnologias e ferramentas adotadas para a implementação da aplicação web e da aplicação móvel. Do lado da aplicação web justifica-se a opção pela linguagem de programação e pela *framework* adotada. Do lado da aplicação móvel dá-se conta da tentativa de construção de uma aplicação móvel universal e justifica-se o motivo do seu abandono e a opção pela plataforma *Android* como destino da aplicação móvel. Justifica-se ainda a opção pelo protocolo de comunicação MQTT usado não só na comunicação entre dispositivos mas também na comunicação entre o servidor e o cliente.

O capítulo cinco foca-se na aplicação web, na aplicação móvel e na interação entre estas duas aplicações. É dada especial relevância ao modo de funcionamento da *framework Laravel* e são incluídos excertos de código ilustrativos dos conceitos apresentados.

O capítulo seis aborda a metodologia adotada para a implementação do sistema e apresenta as principais funcionalidades da aplicação.

No sétimo capítulo apresentam-se as atividades levadas a cabo para as fases de verificação e validação, dando conta dos testes a que foi submetido o sistema e ao processo de validação levado a cabo pelo cliente.

No último capítulo são tecidas conclusões e uma breve reflexão daquilo que pode constituir trabalho futuro para a melhoria do *iNavigator*.

Por fim, considera-se necessário apresentar esclarecimento para a grande quantidade de estrangeirismos, nomeadamente anglicismos presentes no texto. Como estes vocábulos, por serem ubíquos, fazem sentido e despertam associações semânticas automáticas quando grafados no original inglês optou-se por os deixar no original inglês todos os termos considerados comuns, alterando apenas o corte da sua tipografia para itálico.

2 ESTADO DA ARTE

As tecnologias móveis podem desempenhar um papel particularmente relevante no apoio à realização de um conjunto de tarefas de complexidade variável por utilizadores com necessidades especiais. Estes utilizadores compreendem não só os indivíduos que tenham défice cognitivo ligeiro mas também a população em progressivo envelhecimento dos países ocidentais [1], de onde se estima que a maioria sofra de perturbações na memória [2].

Um indivíduo com défice cognitivo enfrenta problemas na execução da sua rotina diária, de que são exemplos o controlo da medicação, a toma das refeições, na higiene pessoal e a seleção de vestuário assim como na orientação espacial [3].

Os indivíduos com défices cognitivos ou de memória tendem a usar lembretes para melhor seguirem as atividades da sua rotina diária [4] e, para os ajudar, existe no mercado grande número de sistemas de sofisticação variada.

Hsu et al [5] conceberam um sistema de lembretes que implementaram em PDAs, os dispositivos tipicamente considerados como os antecessores dos *smartphones* atuais. O sistema de *Hsu* baseava-se em RFID para detetar a saída da habitação e, nesse momento, lançava um alerta com uma lista de objetos que o indivíduo devia carregar consigo.

Wang et al [6] propuseram um sistema de alertas hospitalar baseado em Bluetooth. Este sistema consistia na informação medicamentosa de cada paciente e na transmissão da informação entre a prescrição do médico, as caixas de medicamentos “inteligentes” a cargo das enfermeiras e um sistema de mensagens que encaminhava as instruções dos médicos para as enfermeiras.

Asai et al [7] conceberam um sistema de alertas de medicação *context-aware* que funcionava com vários tipos de sensores, nomeadamente sensores usados em casas inteligentes. O sistema consistia em quatro módulos: um módulo de sensores (que consistia numa tabela de medicação com leitores RFID), um módulo de tomadas de decisão, um módulo de geração de regras e um módulo de atuação (que consistia num terminal dedicado instalado na casa do utente e num telemóvel).

No caso específico de problemas cognitivos ligeiros, a adoção de tecnologias móveis pode ser particularmente útil nomeadamente nos problemas com perda de memória, atenção, raciocínio, orientação, fala, sendo que a incapacidade na realização de tarefas pode manifestar-se de forma muito diversa, desde o simples esquecimento na realização de tarefas até situações que podem colocar em risco a sua segurança como desorientação, falha na toma de medicação.

O envelhecimento da população agrava esta situação, uma vez que potencia o surgimento de um conjunto de patologias (demência, *Alzheimer*, ...) em que este tipo de incapacidade é preponderante [8].

Existem ainda outras patologias em que estes défices cognitivos são particularmente relevantes, como por exemplo síndrome de *Down* e de *Asperger* [9] [10].

2.1 APLICAÇÕES DE ALARMES

As aplicações móveis capazes de gerir lembretes são inúmeras, porém, com uma ou outra diferença de pormenor, acabam por funcionar de forma análoga: a aplicação é autónoma e nela se definem um conjunto de alarmes.

Esta abordagem tem a vantagem significativa de autonomizar a aplicação e de permitir a definição de alarmes mesmo que o dispositivo não tenha acesso à Internet. A desvantagem é que não permite a centralização da gestão dos alarmes nem o seguimento do seu cumprimento.

A *To Do Reminder* é uma aplicação disponibilizada em modelo *freemium*¹ pela *App Innovation* e uma das mais usadas no sistema operativo *Android*. A caminho dos 5 milhões de instalações, a aplicação é popular e extremamente completa: há opções para gerir múltiplas categorias de alarmes, para controlar os alarmes vindouros (e também os já passados) e ainda a integração com redes sociais através, por exemplo, da publicação de uma mensagem no *facebook* ou *twitter* no aniversário de alguém.

Em função do grande número de funcionalidades, a aplicação é particularmente complexa. Há imensas opções para configurar alarmes, categorias de alarmes e ligações a sistemas de terceiros. Não há, porém, nenhum modo de partilhar ou mesmo de fazer uma cópia de segurança dos alarmes com o objetivo de os migrar para outro dispositivo e também não é possível que outrem vigie o seu cumprimento efetivo.

A imagem seguinte mostra em plano de fundo, o ecrã de visualização de alarmes por dia e, em primeiro plano, os detalhes dos alarmes de cada dia.

¹ O modelo *freemium* disponibiliza a aplicação gratuitamente mas requer o pagamento para obter acesso a funcionalidades adicionais ou bens virtuais. Este modelo tem-se tornado extremamente popular porque elimina a barreira do pagamento no acesso à aplicação, escolhendo o utilizador se pretende pagar por funcionalidades ou bens *premium* durante a utilização.

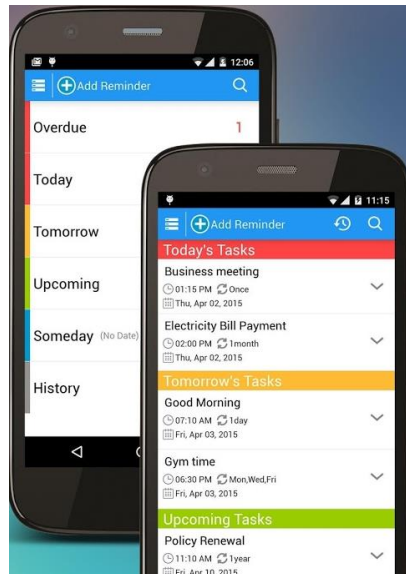


Figura 3 - A lista de alarmes e os detalhes dos alarmes na aplicação *To Do Reminder*.

A *Just Reminder* é uma aplicação similar, igualmente disponibilizada em modelo *freemium* e que também permite definir alarmes e categorizá-los de forma simples. À semelhança da *To Do Reminder*, a definição de alarmes é feita também no dispositivo e é possível sincronizar a aplicação com a lista telefónica do *Android* para definir automaticamente alarmes para o aniversário das pessoas na lista de contactos.

A aplicação oferece alguns extras interessantes como a possibilidade de definir alarmes sem a necessidade de digitar qualquer texto, usando técnicas de *speech-to-text* que convertem a voz humana em texto. Há também a possibilidade de fazer backup de todos os dados e de os migrar para a mesma aplicação noutro dispositivo. Continuam, porém, ausentes as opções de definição de alarmes e do seu controlo por terceiros, seja através de uma aplicação móvel, seja através de um servidor web.

A imagem seguinte mostra a lista de alarmes ativos na aplicação. Cada alarme é descrito por um texto e pode ser acompanhado por uma imagem ou ícone.

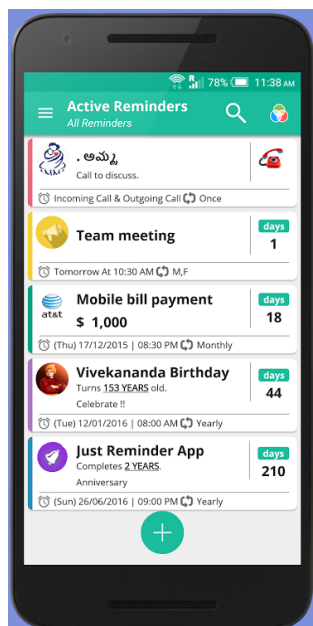


Figura 4 – Lista de alarmes ativos na aplicação *Just Reminder*.

Apesar do grande número de aplicações capazes de gerir alarmes, as aplicações verdadeiramente gratuitas são menos numerosas e, como esperado, muito menos completas.

A aplicação *Simplest Reminder* da *GadgetJudge* é grátis e permite gerir alarmes de forma simples. A aplicação tem uma versão comercial que permite definir alarmes periódicos mas a versão gratuita é suficientemente funcional para merecer ser considerada.

Nesta aplicação é possível definir alarmes que depois serão mostrados no ecrã sob a forma de notificações com opcional despertar do dispositivo de *deep sleep*¹. A imagem seguinte mostra a lista de alarmes ativos.

¹ *Deep sleep* é o nome informal de um modo de gestão de energia no sistema operativo *Android*. Neste modo a velocidade do processador principal é reduzida e os sensores não usados são desligados

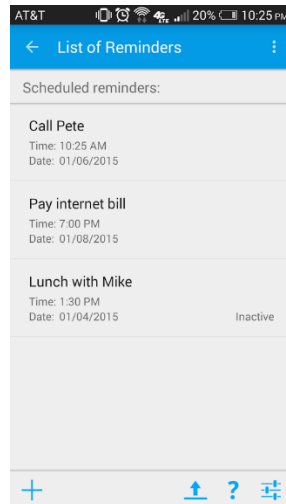


Figura 5 – Lista de alarmes na aplicação *Simplest Reminder*.

2.2 SISTEMAS DE LOCALIZAÇÃO

Para além da gestão de alarmes, a monitorização da localização do indivíduo foi alvo de atenção por parte da indústria e há no mercado grande número de soluções de localização de complexidade variável.

A *Life360* [11] lançou um serviço de localização que usa dispositivos móveis *Android*, *iOS* ou *Windows Phone* para monitorizar indivíduos. Este produto foi claramente orientado à família e permite criar conta num servidor e, após a instalação da aplicação nos dispositivos iniciar a monitorização. A monitorização não é em tempo real e exige que o dispositivo tenha não só acesso permanente à Internet mas também disponibilidade GPS. A imagem seguinte, retirada do site oficial do produto mostra, num dispositivo móvel, a localização dos vários elementos de uma família.



Figura 6 – Ecrã principal da aplicação Life360 num dispositivo Apple.

A grande vantagem do sistema é permitir agrupar pessoas em famílias (designados círculos na aplicação) e seguir os seus movimentos, que ficam guardados no histórico. É também possível comunicar através de chat privado rudimentar ou por chamada de voz. O sistema é seguro porque a agregação de um dispositivo a uma conta não é automática e exige confirmação por parte do seu proprietário. O produto base é grátis mas existem duas ofertas comerciais com renda mensal e que incluem funcionalidades adicionais.

A *Runtastic* [12] é uma empresa especializada em aplicações móveis para acompanhamento de atletas. A sua gama de aplicações inclui soluções para corrida, ciclismo e caminhada e, uma das opções disponíveis inclui a opção de se fazer o acompanhamento em tempo real do circuito realizado pelo dispositivo móvel onde a aplicação está instalada. O serviço base é grátis mas não inclui a capacidade de acompanhamento, que implica o pagamento de uma mensalidade. A imagem abaixo mostra o mapa disponibilizado no dispositivo (que deverá ter obrigatoriamente ligação à Internet) – o mapa disponibilizado online aquando da subscrição do serviço é similar.

- A aplicação cliente assumirá dois perfis de utilização e haverá comunicação entre elas.
- A criação da figura do vigilante, que tem por missão controlar as atividades quotidianas de um conjunto de utentes. O controlo pode ser feito através da sua própria aplicação móvel ou através de uma aplicação web.
- A localização é feita apenas *on-demand* e não em contínuo para respeitar a privacidade do utente e para maximizar a vida útil da bateria dos equipamentos terminais.

3 ANÁLISE DE REQUISITOS

A ideia para o *iNavigator* surgiu na sequência de várias conversas informais entre o Dr. Sérgio Vicente, o professor doutor Jorge Henriques, da Universidade de Coimbra e o professor doutor Simão Paredes, orientador do projeto e professor no ISEC.

Dessas conversas entendeu-se organizar um conjunto de reuniões para identificar as dificuldades e necessidades dos utilizadores que constituiriam o público-alvo e, gradualmente, foram-se identificando formalmente requisitos e, a partir deles, extraindo os casos de uso do sistema.

Neste capítulo apresentar-se-ão o documento de visão e âmbito, os requisitos funcionais e não funcionais, os atores do sistema, os casos de uso, as regras de negócio e, finalmente as maquetes de baixa fidelidade da interface do sistema.

3.1 DOCUMENTO DE VISÃO E ÂMBITO

As primeiras reuniões de *stakeholders* permitiram elaborar o documento de visão e âmbito que, pela sua extensão, não é reproduzido neste capítulo mas pode ser consultado integralmente no anexo I. Este documento identifica os requisitos do negócio, nomeadamente o enquadramento, os objetivos, os critérios de sucesso e os riscos; a visão da solução que nos propusemos construir, com ênfase para as principais funcionalidades, pressupostos e dependências; o âmbito do projeto e as suas limitações e exclusões; e, por fim, o contexto em que o projeto decorrerá, sem esquecer os perfis dos vários *stakeholders* e as prioridades do projeto. A partir do documento de visão e âmbito foi possível identificar os requisitos funcionais e não funcionais do sistema.

3.2 REQUISITOS FUNCIONAIS

O *iNavigator* é um sistema capaz de apoiar o cidadão com deficiência cognitiva ligeira, tipicamente ligada à dificuldade de memória e de atenção ou às dificuldades de orientação, na execução das suas atividades quotidianas, recorrendo para isso a uma aplicação móvel capaz de definir alarmes e localizar o utilizador e a uma aplicação web.

3.2.1 Aplicação web

- Implementar uma aplicação web de utilização intuitiva e design responsivo que permita gerir e monitorizar utilizadores, alarmes e dispositivos móveis. Assim, deve ser possível:
 - criar uma conta de utilizador que permite usar a aplicação visualizando apenas os dados relevantes à conta do utilizador autenticado.
 - criar, editar e eliminar utilizadores e associá-los ao perfil de utilização.
 - criar, editar e eliminar categorias de alarme e associar-lhe uma designação, uma imagem e um som.
 - criar, editar e eliminar alarmes com campos textuais e associar-lhe uma categoria.
 - criar eventos para cada alarme com periodicidades predefinidas.
 - associar uma imagem e um som únicos a cada alarme.
 - criar, editar e eliminar dispositivos móveis e associá-los a pessoas.
 - associar alarmes a dispositivos móveis.
 - publicar alarmes e solicitar relatórios ao dispositivo móvel.
 - gerir a conta de utilizador e mudar a palavra-passe de acesso à aplicação.

- Deve ainda ser criada uma *dashboard* que permita visualizar relatórios e estatísticas relevantes acerca dos alarmes no sistema e do seu estado.

3.2.2 **Aplicação móvel *Android***

- Implementar uma aplicação móvel *Android* de utilização intuitiva que assuma os perfis de utente e vigilante, permita interagir com os alarmes e que comunique com o servidor.
 - Deve ser possível configurar automaticamente e de forma transparente o perfil que a aplicação deve assumir aquando da autenticação do utilizador.
 - Deve haver menus diferentes para o utente e para o vigilante.
 - Deve haver funcionalidades para o utente capturar fotos, vídeos e som.
 - Deve haver um menu SOS para o utente poder comunicar diretamente com o vigilante via voz ou SMS, fazendo uso de textos predefinidos ou escrevendo o seu próprio texto.
 - Deve ser possível ao utente localizar-se no mapa.
 - Deve haver um ecrã de interação com o alarme que reproduza o som ao mesmo tempo que mostra a imagem e o texto e que disponibilize controlos para que o alarme possa ser aceite ou adiado.
 - Os alarmes adiados devem ser ativados novamente passados alguns segundos ou minutos.
 - A aplicação do utente deve responder a pedidos de localização enviados pela aplicação do vigilante.
 - A aplicação do utente deve responder a pedidos de relatório dos alarmes enviados pelo servidor ou pela aplicação do vigilante.

3.3 REQUISITOS NÃO FUNCIONAIS

Paralelamente à identificação dos requisitos funcionais foram também identificados requisitos não funcionais que limitam o modo de utilização da aplicação:

- **Interface** – a aplicação deve ser totalmente usável através da combinação teclado e rato ou, em alternativa, interface tátil com teclado virtual.
- **Desempenho** – a aplicação web deve suportar no mínimo 100 utilizadores simultâneos.
- **Desempenho** – as páginas geradas pela aplicação web devem ser descarregadas em menos de 5 segundos numa ligação de 10Mbps.
- **Desempenho** – as operações que envolvam o acesso ao modelo persistente de dados devem estar concluídas em menos de 4 segundos.
- **Segurança** – as aplicações só devem poder ser usadas por utilizadores autenticados.
- **Segurança** – a aplicação web não deve permitir a contaminação dos dados entre as diferentes contas de utilizador.
- **Usabilidade** – as aplicações devem assegurar navegação intuitiva.
- **Disponibilidade** – a aplicação web deve estar disponível 99,9% das vezes, 24 horas por dia, 7 dias por semana, exceto nos momentos afetos à manutenção do sistema.
- **Robustez** – a aplicação web não deve recuperar nenhuma das operações não efetivadas se for perdida a ligação entre o *browser* cliente e o servidor.

3.4 CASOS DE USO

A especificação dos casos de uso do sistema é um processo sistemático que se inicia com a identificação dos atores do sistema. Assim, para a aplicação web do servidor foram identificados quatro atores:

- **Utilizador não autenticado** – este ator não pode aceder a qualquer funcionalidade da aplicação web exceto criar uma conta de utilizador ou autenticar-se.
- **Utilizador autenticado** – este ator pode aceder a todas as funcionalidades disponibilizadas pela aplicação web exceto criar uma conta e autenticar-se.
- **Internet** – este ator permite a comunicação do servidor com o exterior.
- **Servidor de email** – este ator é responsável por enviar uma mensagem de correio eletrónico aquando da criação da conta de utilizador.

Para a aplicação móvel *Android* foram identificados quatro atores:

- **Utente** – este ator pode interagir com os alarmes, aceder ao menu SOS para comunicar com o vigilante, registar fotografias, vídeos e sons e localizar-se no mapa.
- **Vigilante** – este ator pode localizar o utente no mapa e pedir o relatório dos alarmes.
- **Relógio do sistema** – este ator é responsável por tornar os alarmes visíveis nos momentos definidos.
- **Rede GSM** – este ator permite a comunicação entre dispositivos e também entre o dispositivo e o servidor.

Identificados os atores especificaram-se os vários casos de uso e que foram divididos dois grupos: os casos de uso da aplicação web e os casos de uso da aplicação móvel *Android*. Como, ainda assim, os diagramas ficavam demasiado complexos, dividiram-se os casos de uso da aplicação web em 3 diagramas e os casos de uso da aplicação móvel em dois diagramas, um por cada perfil de utilizador. Os diagramas dos casos de uso estão integralmente reproduzidos no anexo II.

Na figura seguinte está descrito o caso de uso fazer login no servidor. Este caso de uso é executado pelo vigilante e é fundamental para que as funcionalidades do servidor estejam disponíveis. No anexo III encontra-se a especificação de alguns casos de uso do servidor e de todos os casos de uso da aplicação móvel em formato tabular.

ID do caso de uso: 1	
Nome do caso de uso:	Fazer login
Atores:	Utilizador não autenticado
Descrição:	Sempre que um utilizador acede à aplicação web tem que fazer login para ser reconhecido pelo servidor.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Fazer login. 1. O utilizador não autenticado faz login introduzindo o nome de utilizador e a palavra passe e pressionando o botão Login. 2. Se as credenciais forem reconhecidas o utilizador passa a estar autenticado. 3. O sistema apresenta a página inicial da aplicação web.
Fluxos alternativos:	2. Se as credenciais não forem reconhecidas é mostrada uma mensagem de erro.

Figura 8 - Especificação do caso de uso fazer login.

3.5 REGRAS DE NEGÓCIO

Paralelamente à especificação dos casos de uso foram também definidas as regras de negócio do *iNavigator*. Cada regra de negócio foi classificada em função do seu tipo ao mesmo tempo que se identificava a sua natureza e fonte.

Na figura seguinte reproduzem-se as cinco primeiras regras de negócio. A tabela com a reprodução integral das regras de negócio do *iNavigator* identificadas pode ser encontrada no anexo V.

Identificação	Definição da regra	Tipo de regra	Estática ou dinâmica	Fonte
BR-1	As chamadas telefónicas têm que ser pagas pelos utentes.	Restrição	Dinâmica	Utente
BR-2	As mensagens SMS têm que ser pagas pelos utentes ou pelos vigilantes.	Restrição	Dinâmica	Vigilantes Utente
BR-3	O vigilante pode obter a localização do utente via mensagem SMS.	Facto	Estático	Vigilante
BR-4	O vigilante pode obter um relatório das atividades realizadas pelo utente durante o dia em curso.	Facto	Estático	Vigilante
BR-5	O utente pode aceitar ou adiar um alarme.	Facto	Estático	Utente

Figura 9 – Excerto da tabela de regras de negócio do *iNavigator*.

4 TECNOLOGIAS E FERRAMENTAS ADOTADAS

O *iNavigator* consiste em duas aplicações de tipologia diferente: uma aplicação web, usada como servidor e que permite criar configurações de alarmes e uma aplicação para dispositivos móveis, que comunica com a aplicação web para receber uma configuração e que depois funciona autonomamente.

Neste capítulo será feita uma breve apresentação das ferramentas, tecnologias e linguagens de programação adotadas, assim como a justificação para a sua adoção nos casos em que existia alternativa.

4.1 A APLICAÇÃO WEB DO SERVIDOR

A massificação do acesso à Internet na primeira década do séc. XXI tornou a codificação de servidores num campo da engenharia de software onde as novidades surgem constantemente. Grande parte dos servidores atuais são codificados em quatro tecnologias concorrentes: *Microsoft ASP.NET*, *Java* (normalmente tendo por base uma *framework* como por exemplo *Spring* ou *Apache Tapestry*), *php* (recorrendo a uma das muitas *frameworks* disponíveis, de que são exemplo *Codeigniter* ou *Laravel*) e *node.js*.

4.1.1 O stack *Apache*, *MySQL* e *php*

Para a codificação do servidor do *iNavigator* foi selecionada a linguagem *php*, uma linguagem *open-source*, simples de usar e com uma comunidade muito forte. Atualmente o *php* conta com níveis de adoção muito relevantes [13], o que se reflete diretamente na quantidade de ajuda disponível na Internet. A linguagem possui inúmeras

bibliotecas de funções – na sua maioria desenvolvidas por terceiros – muita documentação e é, no seu estado atual, uma linguagem orientada a objetos.

As razões para a adoção desta linguagem foram várias: em primeiro lugar, a linguagem é multiplataforma e, como tal, não tem dependências de um sistema operativo em particular. Estão disponíveis no mercado, de forma gratuita, soluções AMP para os principais sistemas operativos que facilitam grandemente a instalação e configuração inicial da pilha formada pelo servidor *Apache HTTP Server*, a base de dados *MySQL* e a linguagem de programação *php*.

Outro fator determinante para a adoção desta linguagem foi o seu licenciamento: *php* tem uma licença própria, baseada na licença BSD e pode ser usada, sem restrições, mesmo em projetos comerciais. Considerando que o *Apache HTTP Server* e a base de dados *MySQL* têm também licenças similares percebe-se a razão para que as ofertas comerciais de alojamento serem muito mais económicas do que ofertas similares baseadas noutras tecnologias.

Apesar da sua popularidade, a linguagem não é isenta de críticas e historicamente foram-lhe apontados reparos à sua falta de consistência interna [14] [15], pela promoção de um estilo de programação pouco seguro [16] e por ser exclusivamente interpretada. A versão 7 da linguagem, ainda em fase beta, inclui profunda otimização [17], praticamente comparável à inclusão de compilação JIT [18] [19] e promete incrementos substanciais na velocidade de execução.

4.1.2 ***php* e MVC**

Tradicionalmente a programação em *php* não é particularmente estruturada. Uma aplicação codificada em *php* típica consistia assim num conjunto de ficheiros

relativamente díspares, com cada um deles a executar uma tarefa específica e independente, sem grandes preocupações de coesão ou visão do conjunto. Com o amadurecimento da linguagem e o aparecimento de *frameworks* iniciou-se um esforço significativo de sistematização e implementação de boas práticas de programação, nomeadamente pela introdução dos princípios GRASP [20] e, por inferência, dos padrões de design popularizados pelo coloquialmente chamado *Gang of Four* (*Erich Gamma, Richard Helm, Ralph Johnson e John Vissides*), de onde se destacam os padrões *Builder, Command, Facade, Factory, Iterator, Observer, Repository, Singleton* e *Strategy* [21].

Uma razão que pesou grandemente na opção por esta linguagem prendeu-se assim com a grande quantidade de *frameworks* existentes e com a sua maturidade. Uma *framework* é uma *estrutura* – um guia – para o programador desenvolver a sua aplicação. Uma boa *framework* deve promover boas práticas de programação e a implementação, ainda que transparente, de padrões de design de software que permitam não só economizar tempo mas também criar aplicações mais sólidas e seguras, além de reduzir a quantidade de código repetido [22] [23] [24].

As *frameworks* modernas implementam, de forma praticamente universal, o padrão arquitetural MVC. A filosofia que lhe está subjacente é o isolamento de cada uma das responsabilidades principais aquando da implementação de interfaces de utilizador. Neste padrão, as interações com os dados estão estratificadas em componentes, cada um com uma responsabilidade específica e desacoplada dos restantes.

A maioria das *frameworks* PHP segue o padrão arquitetural MVC (*Model-View-Controller*). O MVC separa assim a informação e as regras de negócio da interface com a qual o utilizador interage. Em termos gerais, o MVC é uma forma de estruturar a aplicação

que permite que a interface de interação, a que se chama *View*, esteja separada do controlo da informação em si (*Model*), separação essa que é intermediada por outra camada controladora a que se dá o nome de *Controller*.

O diagrama abaixo [25] ilustra o modelo MVC e enfatiza as interações entre os vários componentes e o utilizador. Da sua análise é fácil constatar que o utilizador apenas interage com elementos do controlador e não lhe é dado acesso direto aos dados do modelo. Em função das operações do utilizador o controlador provoca alterações de estado no modelo que, por sua vez, provocam atualização das vistas e que são a única forma do utilizador poder consultar os dados.

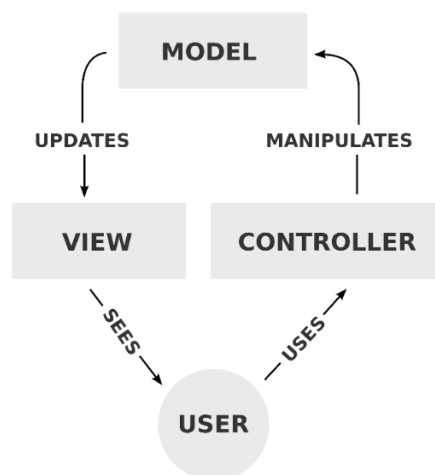


Figura 10 - Diagrama do modelo MVC.

O modelo é o componente que representa os dados e que gere o seu acesso (escrita, validação e leitura), em função dos comandos que lhe são enviados pelo controlador. O controlador é responsável por toda a lógica da aplicação. É ele que decide que modelo utilizar, que pedidos lhe devem ser feitos e que combinação de vistas será usada para mostrar os dados. A vista representa a interação entre o sistema e o utilizador.

Tudo o que o utilizador vê, seja *html*, *xml*, *csv*, etc. deve ser gerado e exibido através deste componente.

4.1.3 **A *framework* *Laravel***

As *frameworks* modernas implementam, de forma praticamente universal, o padrão arquitetural MVC.

A escolha de uma *framework* não é simples dado o seu elevado número e a volatilidade da sua adoção. Historicamente *CodeIgniter*, *Symphony* e *Zend* foram as *frameworks* mais usadas na linguagem *php* mas, nos últimos anos, *Laravel* tem conquistado uma grande quantidade de programadores e é, neste momento, a *framework* mais usada.

Como seria espectável, *Laravel* também implementa o padrão MVC mas onde cada componente é apoiado por um conjunto de bibliotecas adicionais com o objetivo duplo de facilitar a vida ao programador ao mesmo tempo que assegura a implementação quase transparente de boas práticas de arquitetura.

O gráfico seguinte [26] mostra a distribuição da quota de mercado das principais *frameworks* de *php*. Como se pode verificar as quatro *frameworks* mais usadas são *Laravel*, *Phalcon*, *Symfony2* e *CodeIgniter*.

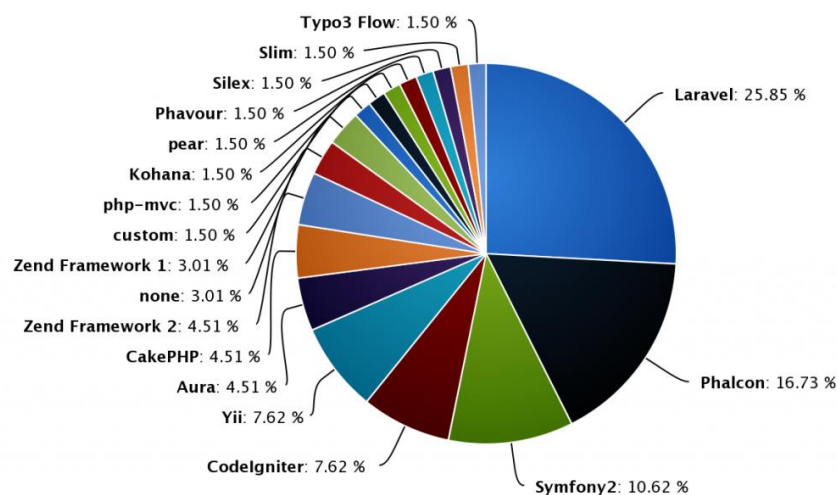


Figura 11 - Gráfico da distribuição da cota de mercado das *frameworks* php.

Como pode observar-se, *Laravel* apresenta grande disparidade sobre as restantes e consegue reclamar mais de 25% da quota de mercado, o que é tanto mais significativo quanto considerarmos que é das *frameworks* mais recentes. *Phalcon* está a ganhar aceitação por resolver uma das maiores limitações da linguagem *php*: a velocidade. Esta *framework* tem a particularidade de ser codificada em C e compilada pelo que os seus tempos de execução são significativamente mais rápidos que as *frameworks* concorrentes.

O sítio web da *framework* *Laravel* está bem estruturado e possui vasta documentação de qualidade, fácil de seguir para quem pretender iniciar o seu uso. Contudo – e à semelhança de qualquer outra *framework* – a documentação nem sempre é sistemática e abrangente e há que ter em conta a natural curva de aprendizagem inerente ao domínio de uma nova técnica de programação.

O *Laravel* faz uso de um instalador de dependências, o *Composer*, que facilita a instalação e atualização de pacotes. Para além do gestor de dependências está ainda disponível uma interface de linha de comandos, designada *Artisan*, que fornece uma série

de comandos úteis para o desenvolvimento e gestão das aplicações web como a criação de controladores ou a gestão da base de dados.

4.2 A APLICAÇÃO MÓVEL

Atualmente o mercado de dispositivos móveis divide-se em 3 sistemas operativos principais: o *iOS* da *Apple*, o *Android* da *Google* e o *Windows Phone* da *Microsoft*, recentemente renomeado *Windows 10 Mobile*.

Como é natural, cada sistema operativo tem a sua *framework* própria, com APIs que diferem não só em filosofia de operação mas até nas linguagens de programação utilizadas para as programar.

O *iOS* da *Apple* foi inicialmente programado em *Objective-C* mas com o lançamento da linguagem *Swift* em Junho de 2014 os programadores têm-se progressivamente dividido entre as duas linguagens, com *Swift* a ganhar aceitação considerável ao longo do último ano [27]. O *Android* pode ser programado em *Java* ou em *C*, apesar desta última linguagem ser muito pouco usada e exigir até um SDK diferente, conhecido por *NDK* (*Native Development Kit*). Finalmente o *Windows Phone*, como aliás grande parte do ecossistema *Microsoft*, é programado em *C#*.

Esta diversidade de APIs e linguagens significa que o programador que deseje oferecer a sua aplicação de forma universal tem à sua frente um esforço considerável e que passa pela aprendizagem de várias linguagens de programação e *frameworks* e pela reescrita da aplicação em cada um destes sistemas operativos.

Para simplificar vida do programador foram surgindo, ao longo dos últimos anos, várias *frameworks* que visam unificar a base de código da aplicação e abstrair as APIs

nativas de cada sistema operativo, recuperando o velho slogan com que a *Sun* publicitava a linguagem *Java*: “*write once, run anywhere*”. Esta promessa é sedutora mas é preciso ter em conta que as várias soluções encontradas são em termos técnicos e de abordagem bastante diferentes para a construção de aplicações *cross-platform*.

O primeiro grupo de soluções para a construção de aplicações móveis universais foi o *PhoneGap*, entretanto renomeado *Apache Cordova*. A abordagem passa por codificar a aplicação em *HTML*, *JS* e *CSS* para que ela possa ser executada não só em *iOS*, *Android* e *Windows Phone* mas também em sistemas operativos menos usados: *Bada*, *BlackBerry*, *Tizen*, *Ubuntu Touch*, *Firefox OS*, *Symbian* e *Palm WebOS*. A vantagem desta abordagem é permitir o reaproveitamento dos conhecimentos destas linguagens de programação que, em maior ou menor grau, é transversal à maioria dos programadores [28]. A desvantagem é que a aplicação resultante não é uma aplicação nativa e, como tal, a sua execução é fortemente penalizada o que resulta forçosamente em pior experiência de utilização [29] [30]. Apesar disso existem inúmeras *frameworks* para a escrita de aplicações *cross-platform* baseadas nesta abordagem, de que são exemplo *AppFurnace*, *AppGyver Steroids*, *Application Craft*, *Appspresso*, *Monaca*, *Enyo* e *Sencha Touch*.

4.2.1 **Xamarin**

Cientes das limitações da abordagem apresentada desenvolveram-se posteriormente *frameworks* de desenvolvimento de aplicações *cross-platform* baseadas numa abordagem diferente. Estas *frameworks* são baseadas numa linguagem de programação tradicional (*C++*, *C#*, *Java*, etc.), que pode ser compilada e a adaptação está do lado do SDK. A ideia é que ao invés de se invocarem os métodos do SDK diretamente

se invoquem métodos de uma API de terceiros que acaba por servir de fachada ao SDK nativo de cada sistema [31].

A *framework* mais usada e baseada nesta abordagem é *Xamarin*, criada pelos autores do projeto Mono que, em 2004 iniciaram o esforço de recriação em ambiente *Linux* da *framework .NET*, com compilador de *C#* e da *Common Language Runtime* compatíveis com as versões oficiais da *Microsoft*. Esta experiência permitiu desenvolver um produto comercial capaz de produzir aplicações nativas para *Android*, *iOS*, *Windows* e *Apple OS X*.

A partir de 2014 a empresa investiu esforço considerável para abstrair os componentes da UI mais comuns a que chamou *Xamarin.Forms*. Com esta API é possível programar a UI com recurso a métodos abstratos que, durante a compilação, são convertidos nas invocações aos métodos correspondentes nas respetivas APIs nativas.

A imagem seguinte, retirada do site oficial da *framework Xamarin* [32], ilustra a sua filosofia de utilização: a maior parte do volume de código de uma aplicação está na codificação das regras de negócio e, como tal, pode ser facilmente partilhado se for codificado numa linguagem que seja compilável noutro sistema. Com *Xamarin.Forms*, o código da UI também passa a poder ser partilhado. O remanescente, ilustrado por cores diferentes por baixo dos logótipos de cada sistema operativo, são as invocações a métodos que, por alguma razão, sejam únicos a cada plataforma.

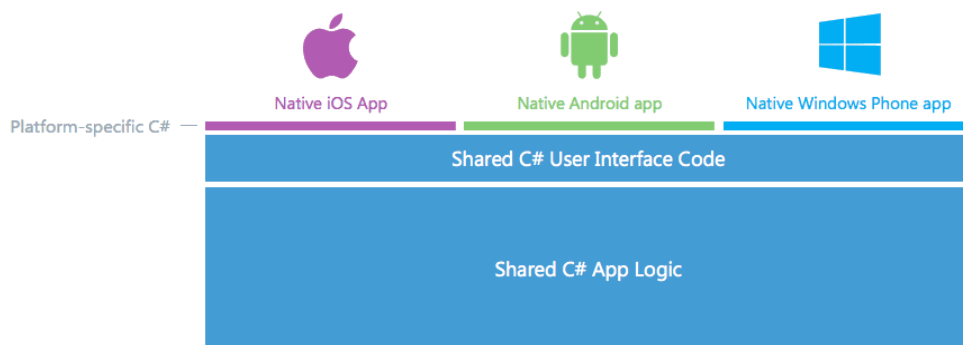


Figura 12 – Diagrama da divisão do código fonte em *Xamarin*.

A promessa de *Xamarin* pareceu interessantíssima e foi ativamente explorada durante várias semanas, resultando num protótipo de alguns casos de uso do *iNavigator*. Foi, aliás, na codificação do protótipo que se identificaram algumas das importantes limitações atuais de *Xamarin* e que obrigaram ao seu abandono mas não sem antes se ter despendido um período de tempo não insignificante na sua elaboração.

O caso de uso principal da aplicação *iNavigator* é a definição programática de alarmes. Para isso, era expectável que *Xamarin* abstraísse a API de alarmes de cada uma das diferentes plataformas móveis, passando a usar-se uma API personalizada que serviria de fachada e adaptador às APIs nativas de cada sistema operativo. Esta abordagem, aliás, é usada por outras *frameworks* como *Unity 3D* ou *Unreal Engine*, vocacionadas para a codificação de jogos mas que cumprem integralmente a promessa da codificação unificada qualquer que seja o dispositivo em que o jogo venha a ser executado.

Em *Xamarin* a abordagem é diferente: aquando da criação do projeto o editor cria, na realidade, quatro projetos: um para *iOS*, outro para *Android* e um último para *Windows Phone*, complementados por um projeto comum com código partilhado por todos. A grande diferença é que as operações sobre as APIs nativas de cada sistema têm que ser codificadas recorrendo aos métodos próprios de cada sistema operativo o que implica, na prática, que o programador tem que conhecer profundamente 3 APIs

diferentes para cada operação. Neste cenário (e no caso específico do *iNavigator*) o único código partilhável era o do acesso à base de dados, à Internet e eventualmente algum do código da UI: o restante código que suporta os casos de uso principais da aplicação (definição de alarmes, envio de SMS, etc.) teria de ser codificado de forma independente para cada plataforma.

Depois da codificação do protótipo considerou-se que o custo do esforço de aprendizagem de três APIs completamente diferentes era demasiado grande para compensar o benefício de disponibilizar a aplicação móvel nos três sistemas operativos, tanto mais que a quota de mercado de cada um destes sistemas favorece de forma clara um deles em detrimento dos demais. Assim, decidiu-se avançar com o sistema operativo *Android*, por ser o mais adotado.

4.2.2 **Android**

O sistema operativo para dispositivos móveis *Android* foi originalmente desenvolvido pela *Android, Inc.* em *Palo Alto*, Califórnia, nos Estados Unidos da América em 2003, por *Andy Rubin*, *Rich Miner*, *Nick Sears* e *Chris White* [33].

O plano original da *Android, Inc.* era o desenvolvimento de um sistema operativo avançado para câmaras digitais mas rapidamente se tornou evidente que o mercado para esse tipo de dispositivos não seria suficientemente grande para justificar o investimento convertendo-se o projeto num sistema operativo para telemóveis capaz de ombrear com os sistemas *Symbian* e o *Windows Mobile*, os maiores sistemas operativos móveis de então (o *iPhone* da *Apple* só viria a ser lançado em 2007) [34].

Em 2005 a primeira versão do *Android* estava praticamente concluída e a *Google* comprou a *Android Inc.*, transformando-a em sua subsidiária e mantendo ao seu serviço

três dos quatro sócios fundadores. Entre 2005 e 2008 o sistema foi sendo progressivamente melhorado e aumentado mas só a 22 de Outubro de 2008 foi lançado o *HTC Dream*, o primeiro telemóvel com sistema operativo *Android*.

A partir de 2008 o *Android* tem crescido exponencialmente. Hoje em dia estima-se que mais de mil e seiscentos milhões de dispositivos *Android* tenham sido ativados e é, sem qualquer dúvida, o sistema operativo móvel mais usado no mundo, com mais de 80% de quota de mercado.

Em termos técnicos o *Android* é baseado no *kernel Linux* e está organizado em camadas, construídas umas sobre as outras. Assim a camada "base" é formada pelo *kernel* do sistema e pelos drivers dos componentes de hardware, por "cima" do qual funcionam um conjunto de *middleware*, bibliotecas e APIs escritos em C. Por cima destas há um conjunto de *software* de aplicação que correm numa *framework* de aplicação que inclui as bibliotecas compatíveis com a linguagem *Java* baseadas no extinto projeto *Harmony* da *Apache Software Foundation*.

Na imagem seguinte pode ver-se uma representação das várias camadas que constituem o *Android* [35]. Note-se que apenas as aplicações de sistema, que constituem a camada superior, podem ser facilmente substituídas por software de outros fabricantes.

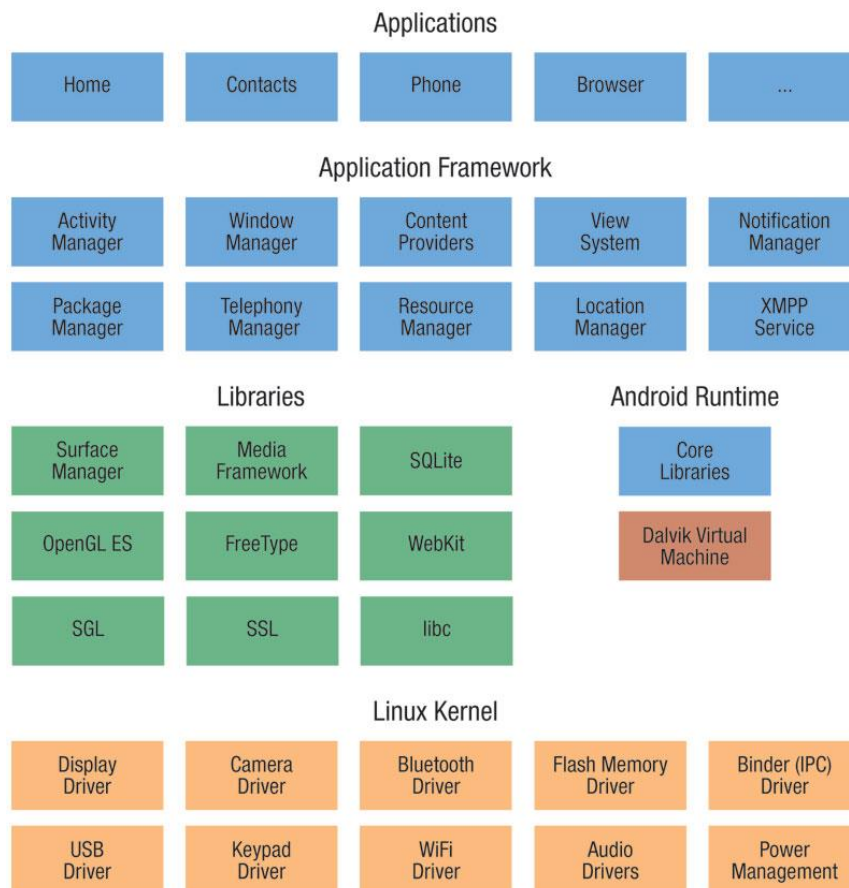


Figura 13 – Diagrama da arquitetura do sistema operativo *Android*.

O *Android* usava (até à versão 4.4 KitKat) a máquina virtual *Dalvik*, originalmente desenvolvida por *Dan Bornstein* e faz uso de um compilador *JIT* para correr código *dex-code*, uma versão traduzida do *bytecode* de *Java*. Esta máquina virtual estava originalmente disponível apenas para arquitetura *ARM* mas as versões mais atuais ganharam suporte para a arquitetura *x86* da Intel e para arquitetura *i.MX* nas versões mais modernas.

A partir da versão 5.0 *Marshmallow* do *Android* a máquina virtual *Dalvik* foi substituída pelo ART (*Android Runtime*). Esta máquina faz uso de compilação *ahead-of-time*, compilando as aplicações aquando da sua instalação, com contributos significativos quer no desempenho quer no consumo de energia.

4.2.3 **Android SDK e Android Studio**

O desenvolvimento para a plataforma móvel *Android* é feito, maioritariamente, em linguagem *Java* complementada com o *Android SDK*, disponibilizado pela *Google* [36]. Tradicionalmente o ambiente de desenvolvimento preferido consistia no IDE Eclipse, no qual se instalavam as extensões *Eclipse Android Development Tools* (ADT) sob a forma de *plug-in* capazes de comunicar com os dispositivos virtuais e físicos necessários para executar e depurar aplicações.

Em Maio de 2013 a *Google* lançou o *Android Studio*, uma versão fortemente personalizada do *IDE IntelliJ IDEA* da *JetBrains*, que rapidamente se popularizou e hoje é usado pela maioria das empresas, tanto mais que o ADT foi entretanto abandonado pela *Google* [37]. Similarmente ao Eclipse, o *Android Studio* suporta os sistemas operativos Windows, OS X e Linux.

4.2.4 **MQTT**

A comunicação entre o servidor e aplicação cliente foi feito através do protocolo *Message Queue Telemetry Transport* (MQTT). Este protocolo de mensagens foi desenvolvido para comunicações assíncronas *M2M* (*Machine-to-Machine*) e *IoT* (*Internet of Things*) desenvolvido por *Andy Stanford-Clark*, da *IBM*, e por *Arlen Nipper*, da *Cirrus Link Solutions* em 1999 [38]. O protocolo destinava-se à transferência de dados de telemetria sob a forma de mensagens em redes de baixo débito e normalmente de alta latência, congestionadas ou constrangidas por qualquer outro método. Este protocolo é tipicamente usado em sensores e atuadores e tem visto recentemente a sua adoção abranger não só telefones móveis, sistemas embutidos em veículos e computadores portáteis mas também *workstations* e servidores [39].

A ideia subjacente ao funcionamento é a da subscrição de um tópico num servidor, genericamente designado *broker* [40]. Quando um cliente pretende publicar uma mensagem tem que subscrever um tópico novo ou juntar-se a um tópico existente no *broker*. Um tópico pode ser subscrito por um grande número clientes e as mensagens novas são automaticamente propagadas por todos os subscritores desse tópico.

No diagrama abaixo [41] pode ver-se um exemplo de comunicação com vários subscribers e publishers num broker MQTT. Cada nó representa uma aplicação cliente e subscreveu o tópico que lhe era relevante. O broker é responsável pela difusão das mensagens entre todos os clientes que subscrevam o mesmo tópico.

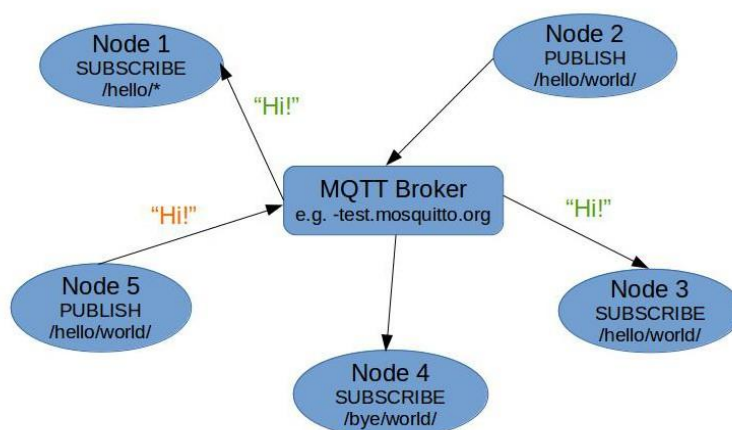


Figura 14 - Diagrama da comunicação no protocolo MQTT.

Um dos objetivos essenciais deste protocolo é o baixo *overhead* no envio das mensagens e, para o controlar, o MQTT define 3 níveis de QoS (*Quality of Service*). Num primeiro nível, as mensagens são enviadas "*at most once*", no modo também comumente designado "*fire and forget*". Este nível de funcionamento assegura que o tamanho da mensagem, a complexidade da comunicação e a ocupação do canal de transporte são os menores possível mas não assegura que a mensagem seja entregue se houver algum problema na rede. O nível seguinte é designado "*at least once*" e obriga a

que o *broker* responda a todas as mensagens publicadas com uma mensagem de *ACK*. O inconveniente deste método é que a mesma mensagem pode ser enviada muitas vezes se, por qualquer razão, o *broker* não enviar a mensagem de *ACK* num período que o cliente esteja configurado para aguardar. O último nível é designado “*exactly once*” e sacrifica a simplicidade, a rapidez e o tamanho de envio em troca da garantia que a mensagem é publicada uma e uma só vez através de um sofisticado mecanismo de *handshacking*¹ e sequenciação das mensagens de *ACK*.

Dada a sua qualidade, resiliência e baixo *overhead* no envio de mensagens, o protocolo foi rapidamente adotado pela indústria e implementado num grande número de *APIs*. No iNavigator foi usada a *API Eclipse Paho* por ser a que mais facilmente se integrava no ambiente *Android*, para o qual dispõe de bibliotecas nativas [42]. Do lado do servidor foi usada a *API phpMQTT* desenvolvida pela *Blue Rhinos* para o envio das mensagens aos clientes. Como o php não suporta *callbacks*, não foi possível usar MQTT para a receção de pedidos que passaram a ser atendidos por HTTP. O *broker* escolhido foi o *Mosquitto* [43] por ser *open source* e se ter mostrado fácil de instalar e configurar e, sobretudo, altamente fiável ao longo do desenvolvimento.

¹ *Handshaking* é o processo automático de negociação de parâmetros de um canal de comunicação estabelecido por duas entidades antes da comunicação se poder iniciar.

5 ARQUITETURA

O *iNavigator* é constituído por duas aplicações: uma aplicação web, usada como servidor e uma aplicação para dispositivos móveis *Android*, que comunica pontualmente com a aplicação web mas que, normalmente, funciona em regime autónomo.

Neste capítulo será feita uma breve apresentação da arquitetura de cada uma dessas aplicações.

5.1 A APLICAÇÃO WEB

A aplicação web que serve de servidor ao *iNavigator* foi codificada em *php* com recurso à *framework Laravel*. Conforme apresentado no capítulo anterior, a programação em *php* com *Laravel* é orientada a objetos e encaminha o programador para o uso de padrões de design e boas práticas de programação.

As aplicações web organizam-se em torno de dados e das operações que, de alguma forma os modificam. Em termos gerais os utilizadores navegam nas rotas da aplicação que ativam controladores que, por sua vez, modificam o estado dos dados dos modelos e que, por fim, atualizam as vistas para que o utilizador possa verificar se as suas operações produziram o efeito esperado. A este modelo de funcionamento chama-se MVC e o seu funcionamento interno já foi apresentado no capítulo anterior.

Na primeira parte deste capítulo será apresentado o modo como *Laravel* implementa os vários intervenientes nesta cadeia fazendo uso, sempre que relevante, de excertos de código que exemplifiquem os conceitos apresentados.

5.1.1 Modelo

Especificação e manutenção da base de dados

Os DBMS, coloquialmente designados por bases de dados, são aplicações informáticas complexas e independentes responsáveis pelo armazenamento organizado de dados. A abordagem típica ao uso de DBMS em aplicações implica a especificação das várias tabelas de dados tidas como necessárias, assim como as respetivas relações entre si. Esta especificação pode ser feita diretamente na UI do DBMS (tipicamente uma consola) ou num programa próprio para o efeito, de que são exemplo o *HeidiSQL*, *Navicat*, *phpMyAdmin*, etc.

A *framework Laravel* introduz uma nova abordagem e traz a especificação das tabelas da base de dados para o código fonte da aplicação. Através da criação de *migrations* é possível definir o esquema de uma tabela da base de dados, conforme se pode verificar no excerto seguinte.

```
class CriarTabelaPerfis extends Migration {  
    public function up() {  
        Schema::create('perfis', function(Blueprint $table) {  
            $table->increments('perfis_id');  
            $table->integer('contas_id')  
                ->unsigned()  
                ->nullable();  
            $table->foreign('contas_id')  
                ->references('contas_id')->on('contas')  
                ->onUpdate('cascade')  
                ->onDelete('cascade');  
            $table->string('perfis_descricao', 63);  
            $table->timestamps();  
        });  
    }  
  
    public function down() {  
        DB::statement('SET FOREIGN_KEY_CHECKS=0;');  
        Schema::dropIfExists('perfis');  
    }  
}
```

Figura 15 - *Migration* responsável pela criação e eliminação de uma tabela.

O excerto acima, apesar da sua dimensão, é particularmente completo: em primeiro lugar é visível a implementação do padrão de design *Command*, completo com um método *up()*, para quando a migração é corrida, e um método *down()*, para quando a migração é desfeita. A presença deste padrão de design é interessante porque permite reiniciar a base de dados facilmente sempre que se fazem alterações à estrutura dos dados numa ou várias tabelas.

Também é visível o nome da tabela criada e desfeita, neste caso a tabela *perfis*. Na criação da tabela especifica-se a estrutura da tabela. Há um campo *increments*, que representa uma chave primária que se auto incrementa, uma chave estrangeira *contas_id* que admite valores nulos através da diretiva *->nullable()*, uma *string* e dois campos de *timestamp*, um que registará o momento de criação de um registo e outro que registará o seu *update* mais recente.

A especificação dos campos da tabela é feita através dos tipos de variáveis próximos das linguagens de programação comuns (*string*, *integer*, etc.) mas é sabido que os DBMS usam outros tipo de dados (*TINYINT*, *INT*, *VARCHAR*, etc.). A conversão é feita através de um filtro selecionável no ficheiro de configuração */app/config/database.php*. Neste ficheiro é possível especificar o driver para o DBMS utilizado entre quatro disponíveis: *SQLite*, *MySQL* (também compatível com *MariaDB*), *PostgreSQL* e *Microsoft SQL Server*. Como seria esperado, as migrações são agnósticas em relação ao DBMS usado e é possível mudar o driver a qualquer momento e usar um DBMS diferente. O *Laravel* assegura a conversão dos comandos SQL gerados para o dialeto SQL usado por cada um.

O ficheiro do modelo

Feitas as várias migrações é necessário especificar os ficheiros de modelos que mapearão a relação entre os objetos *php* e as tabelas do DBMS. O código abaixo reproduz o modelo Evento, que representa uma instância de um alarme.

```
namespace iNavigator\Models;
use Illuminate\Database\Eloquent\Model;

class Evento extends Model {

    protected $table = 'eventos';
    public $primaryKey = 'eventos_id';

    // Estabelece a relação 1:N com a tabela de periodicidades
    public function periodicidade() {
        return $this->belongsTo('iNavigator\Models\Periodicidade', 'periodicidades_id');
    }

    // Estabelece a relação 1:N com a tabela de alarmes
    public function alarme() {
        return $this->belongsTo('iNavigator\Models\Alarme', 'alarmes_id');
    }

    // Estabelece a relação 1:N com a tabela de dias
    public function dia() {
        return $this->belongsTo('iNavigator\Models\Dia', 'dias_id');
    }
}
```

Figura 16 - Modelo responsável pela obtenção e recuperação de dados.

O ORM da *framework Laravel* assume que o nome da tabela é o plural do nome do ficheiro do modelo correspondente grafado em caixa baixa. Assim, como o modelo se chama Evento, o nome da tabela chamar-se-ia eventos. O problema desta regra é que os plurais portugueses nem sempre são regulares e pode ser vantajoso especificar outro nome. Por outro lado, assume-se que a chave primária se chama sempre id.

No exemplo anterior é visível a especificação explícita da tabela usada (eventos) e do nome da respetiva chave primária (eventos_id). É também visível o estabelecer de três relações com outros modelos, no caso, relações de 1:N (especificadas aqui no lado N) definidas através do método *->belongsTo()*.

O ORM da *framework Laravel* traz uma diferença substancial em relação a outros ORM, de que o *Hibernate* é exemplo: não é necessário construir o POJO. As variáveis do modelo são inferidas a partir da *query* à tabela ou tabelas em questão.

Seeding dos dados

Quando se programa uma aplicação web é confortável ter dados de teste, de forma a ir depurando as operações que se vão implementando. Tradicionalmente estes dados de teste são introduzidos de forma *ad-hoc*, diretamente na consola do DBMS ou através de uma aplicação auxiliar. O *Laravel* implementa o conceito de *seeder*, que mais não é que uma classe responsável por criar um objeto e, por ORM, inseri-lo no DBMS. Os vários *seeders* são então invocados e recheiam o DBMS de dados.

O excerto seguinte reproduz uma secção do *seeder* de alarmes. Note-se que para a execução do *seeder* é necessário que os ficheiros que controlam os modelos já estejam especificados uma vez que a inserção de dados é feita por ORM. Esta é, aliás, uma boa maneira de validar os modelos e as relações entre eles.

```
use Illuminate\Models\Alarme;
use Illuminate\Database\Seeder;

class SeederAlarmes extends Seeder {

    public function run() {

        // Apaga todos os registos da tabela e reseta o contador auto_increment
        DB::table('alarmes')->truncate();

        Alarme::create([
            'contas_id' => 1,
            'categorias_id' => 1,
            'periodicidades_id' => 2,
            'alarmes_titulo' => 'Introdução à Programação',
            'alarmes_subtitulo' => 'Sala L1.1',
            'alarmes_urlSom' => 'Alarm_Beep_01.ogg',
            'alarmes_urlImagem' => 'classroom.jpg'
        ]);
    }
}
```

Figura 17 - Um *seeder* com a especificação de um objeto do tipo Alarme.

O acesso aos dados

O acesso aos dados armazenados em bases de dados é tipicamente feito através da escrita e envio de comandos SQL. Este modelo, apesar de confortável e próximo das competências que a maioria dos programadores já tem, é desvantajoso quando se programa numa linguagem orientada a objetos porque nem sempre há relação direta entre os objetos usados nos modelos e as tabelas especificadas no DBMS.

Para ultrapassar esta limitação (e também, não menos importante, para poupar ao programador o trabalho de escrever os comandos SQL) o *Laravel* vem equipado com um ORM particularmente poderoso designado *Eloquent*.

A solicitação e carregamento de um modelo ao DBMS é feito com a especificação do modelo, conforme exemplificado no excerto seguinte.

```
use iNavigator\Models\Dispositivo;

/* ... */

$dispositivo = Dispositivo::where('dispositivos_codigo_identificacao', $idCliente)
    ->orWhere('dispositivos_numeroDoCartao', $idCliente)
    ->with('pessoa')
    ->first();
```

Figura 18 - Uma *query* em ORM com indicação de modelos adicionais a carregar.

Este excerto é interessante porque mostra, em primeiro lugar, que com a *framework Laravel*, a linguagem *php* adotou a organização em *namespaces* usada noutras linguagens. A primeira linha, que ficará no topo do ficheiro, especifica que o modelo *Dispositivo* está armazenado na diretoria *Models* do *namespace iNavigator*. A segunda linha do excerto mostra o carregamento em si através da invocação de métodos estáticos que implementam o padrão *builder*. Cada restrição é especificada através do método adequado (*where*, *orWhere*, etc.) e é visível o carregamento adicional através do método *->with()* do modelo *Pessoa*.

As operações básicas no acesso aos dados são as designadas CRUD (por vezes designado SCRUD, se considerarmos o *S* de *Search*). O CRUD é um acrónimo que representa as operações de criação (*create*), leitura (*read*), atualização (*update*) e eliminação (*delete*) de dados. Numa aplicação web a maioria das entidades são configuráveis pelo utilizador pelo que é necessário dar-lhe acesso às operações típicas do CRUD.

Como seria de esperar todas estas operações estão disponíveis em *Laravel*. O excerto abaixo é artificial e mostra uma operação de criação (*INSERT*) de um alarme, seguida de uma de atualização (*UPDATE*) e outra de eliminação (*DELETE*) de um registo na base de dados.

```
$alarme = new Alarme();  
  
$alarme->alarmes_titulo = Request::input('titulo');  
$alarme->alarmes_subtitulo = Request::input('subtitulo');  
$alarme->save();  
  
$alarme->delete();
```

Figura 19 - Exemplo de criação, edição e destruição de um objeto do tipo Alarme.

As operações de leitura (*SELECT*) já foram referidas mas os exemplos apresentados eram particularmente simples. O excerto abaixo é artificial e mostra uma *query* não trivial onde é carregado um alarme através da ordenação de uma coluna (*eventos_data*) de uma das suas tabelas relacionadas. A novidade está na criação de uma função anónima responsável por passar ao builder da query os campos a partir dos quais deve ser feita a ordenação, uma vez que a ordenação é feita numa tabela que não a do principal.

```

$alarme = Alarme::where('alarmes_id', $idAlarme)
    ->with('dispositivo')
    ->with('categoria')
    ->with(array('eventos' => function($query) {
        return $query->where('contas_id', '<>', Auth::user()->contas_id)
        ->orderBy('eventos_data', 'ASC');
    }))
    ->first();

```

Figura 20 – Acesso a Alarmes condicionado a restrições aplicadas à tabela de Eventos.

5.1.2 Controlador

Rotas

Como acontece com a maioria das *frameworks* web, o acesso à lógica é invocado através de rotas. Em *Laravel* as rotas podem ser reativas a pedidos *GET* e *POST*. No excerto abaixo podemos constatar que a rota */pessoas* está associada a métodos diferentes do controlador *Pessoas* consoante o pedido seja feito por *GET* ou *POST*. A rota apresentada é relativa mas a *framework* transforma-a em absoluta automaticamente.

```

Route::get('/pessoas', 'ControladorPessoas@listarPessoas');
Route::post('/pessoas', 'ControladorPessoas@criarPessoa');

```

Figura 21 - Duas rotas iguais, resolvidas em função do método GET ou POST.

Apesar dos pedidos HTTP 1.0 só poderem ser do tipo *GET* ou *POST*, *Laravel* suporta HTTP 1.1 e os novos métodos, nomeadamente *PUT* e *DELETE*. Para assegurar a retrocompatibilidade estes métodos são enviados como um campo escondido num pedido *GET* mas, aquando da especificação das rotas, é possível usar o nome verdadeiro do método, conforme ilustrado abaixo.

```

Route::delete('/pessoas/{idPessoa}', 'ControladorPessoas@eliminarPessoa');

```

Figura 22 - Uma rota que responde a um pedido DELETE.

Note-se que é ainda possível especificar argumentos (*idPessoa*, no exemplo acima) que serão também passados ao método correspondente de forma automática desde que o nome seja o mesmo.

```
public function eliminarPessoa($idPessoa) { /* ... */ }
```

Figura 23 - Uma rota que responde a um pedido DELETE.

Controladores

Um controlador é normalmente uma classe que implementa as regras de negócio de cada operação. Os métodos que o constituem são invocados a partir das rotas que, por sua vez, refletem os casos de uso pelos quais o utilizador vai passando. Os controladores são normalmente classes relativamente extensas, sobretudo se assegurarem o funcionamento de um caso de uso fulcral para a aplicação.

O exemplo abaixo mostra um excerto adaptado do controlador *Alarmes*. Conforme ilustrado, o controlador principia com a definição do *namespace* em que se insere, ao qual se segue a importação dos vários modelos definidos pelo programador e fachadas da *framework* utilizadas. Finda a importação segue-se um construtor que invoca o *middleware* de autenticação. Este *middleware* é executado imediatamente antes de qualquer método deste controlador ser invocado. Seguidamente ilustram-se dois métodos simples, um responsável por listar os alarmes presentes na base de dados e outro responsável por criar um alarme. Ambos os métodos recolhem dados da base de dados que são passados para a vista correspondente através da instrução *compact()*.

```

namespace iNavigator\Http\Controllers;

use iNavigator\Models\Alarme;
use iNavigator\Models\Categoria;
use iNavigator\Models\Periodicidade;
/* ... */
use Auth;
/* ... */

class ControladorAlarmes extends Controller {

    public function __construct() {
        $this->middleware('auth');
    }

    public function listarAlarmes() {
        $alarmes = Alarme::with('periodicidade')
            ->with('categoria')
            ->with('conta')
            ->with('eventos.dia') // Chamada à boss de um nested object
            ->where('contas_id', '=', Auth::user()->contas_id)
            ->get();
        $categorias = Categoria::where('contas_id', '=', Auth::user()->contas_id)->get();
        $periodicidades = Periodicidade::all();

        return View::make('alarmes.listarAlarmes', compact('alarmes', 'categorias'));
    }

    public function criarAlarme() {
        $alarme = new Alarme();
        $alarme->contas_id = Auth::user()->contas_id;
        $alarme->save();

        return View::make('alarmes.criarAlarme', compact('alarme'));
    }
}

```

Figura 24 - Excerto do controlador de Alarmes.

Fachadas da *framework*

No exemplo do ponto anterior era visível a importação e utilização de uma das fachadas da *framework*, identificada como *Auth*. Esta fachada guarda informação sobre o utilizador autenticado na aplicação e, naturalmente, já trás implementada a lógica suficiente para distinguir utilizadores concorrentemente autenticados em função da sua sessão. O *Laravel* tem pré-programadas um grande número de fachadas que podem ser acedidas de forma estática e que executam as funções mais comuns com o objetivo de poupar ao programador a sua codificação ao mesmo tempo que ajudam a eliminar bugs neste tipo de código tipicamente repetitivo.

Na sua versão atual (5.1) a *framework* conta com 31 fachadas que gerem componentes tão distintos como a autenticação, a cache, a configuração da aplicação, as cookies, a encriptação, a base de dados, os ficheiros, os registos (*logs*), as passwords, o envio de emails, as sessões e as vistas, entre muitas outras.

O seu uso é muito simples, conforme ilustrado no exemplo abaixo:

```
use Auth;
/* ... */

echo (Auth::user()->personas_nome);
```

Figura 25 - Exemplo de utilização da fachada *Auth*.

O *middleware* de HTTP

O *middleware* de HTTP fornece um mecanismo particularmente conveniente para filtrar os pedidos HTTP que entram na aplicação. Por exemplo, é possível especificar um *middleware* que assegure que o utilizador está autenticado na aplicação. O *middleware* assegurar-se-á então que, em caso negativo, o utilizador seja reencaminhado para o ecrã de autenticação. Se, pelo contrário, o *middleware* determinar que o utilizador está autenticado, permite que o pedido HTTP seja encaminhado para as camadas mais profundas da aplicação.

Como seria de esperar, é possível especificar *middleware* personalizado se for vantajoso executar tarefas específicas a todos os pedidos HTTP. No caso do *iNavigator* foi especificado um *middleware* que define o idioma da aplicação em função do valor especificado na variável de sessão.

O exemplo seguinte é cópia integral do *override* do método *.handle()* do *middleware Authenticate*, que garante que o utilizador atual tem que estar autenticado para poder usar a aplicação. Repare-se que o controlador verifica se o pedido HTTP

provém de um utilizador não autenticado e, caso tenha sido feito por AJAX para, em caso positivo, responder com um erro 401 – *Unauthorized*. Se o pedido não tiver sido feito por AJAX o utilizador não autenticado é redirecionado para a página de autenticação. Se o utilizador estiver autenticado o *middleware* simplesmente passa o pedido para a camada seguinte na pilha de responsável pelo tratamento de pedidos HTTP.

```
/**
 * Handle an incoming request.
 *
 * @param \Illuminate\Http\Request $request
 * @param \Closure $next
 * @return mixed
 */
public function handle($request, Closure $next) {
    if ($this->auth->guest()) {
        if ($request->ajax()) {
            return response('Unauthorized.', 401);
        } else {
            return redirect()->guest('autenticacao/login');
        }
    }
    return $next($request);
}
```

Figura 26 - Excerto do *middleware* responsável pela autenticação do utilizador.

5.1.3 Vista

Layouts e master page

As vistas em aplicações web consistem em ficheiros HTML, complementado por CSS, JS e qualquer outra tecnologia adicional que se entenda necessário. Tipicamente construídas de forma independente, também em *Laravel* se notou um esforço considerável para estratificar e sistematizar componentes que depois podem ser montados para gerar dinamicamente a vista final.

Uma das boas práticas a este nível é uso de uma *master page*. Nesta página é possível especificar a estrutura geral da página, importar os ficheiros CSS e JS comuns e agregar os elementos individuais que irão fazer parte da vista final a enviar ao browser

cliente. A vantagem desta abordagem é a eliminação de repetições que resulta na menos provável introdução de bugs e em tempo de revisão e manutenção forçosamente mais curto.

O exemplo seguinte mostra uma compressão da *master page* às suas secções fundamentais. Assim, dentro da tag `<head>` são incluídos todos os ficheiros de CSS usados na aplicação. Na tag `<body>` são incluídas as várias subvistas usadas para criar a vista final mostrada ao utilizador. Note-se que a secção principal está identificada como conteúdo e é incluída pela diretiva `@yield('conteudo')`. Por fim o ficheiro inclui todos os ficheiros JS usados na aplicação.

```
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>iNavigator</title>
    @section('head')
    <!-- Bootstrap 3.3.4 -->
    {!! Html::style('css/bootstrap.min.css'); !!}
    /* ... */
  </head>
  <body class="skin-blue sidebar-mini fixed">
    <div class="wrapper">
      <header class="main-header">
        @include('layouts.barraSuperior')
      </header>

      @include('layouts.menuLateral')

      <div class="content-wrapper">
        <section class="content-header">
          @yield('headerPage')
        </section>
        <section class="content">
          @yield('conteudo')
        </section>
      </div>
      @section('footer')
      @include('layouts.footer')
      @show
    </div>
    {!! Html::script('js/api/jquery-2.1.4.min.js'); !!}
    @yield('scripts')
    /* ... */
  </body>
</html>
```

Figura 27 - Excerto de um ficheiro da vista enriquecido com diretivas de *Blade*.

Na imagem seguinte é possível ver a explosão do layout usado na *master page* evidenciado cada uma das subvistas incorporada no código anterior com a diretivas `@include()` e `@yield()`. As subvistas têm nomes evidentes e são, pela ordem em que são referidas no exemplo de código, *barraSuperior*, *menuLateral*, *headerPage*, *conteúdo* e *footer*.

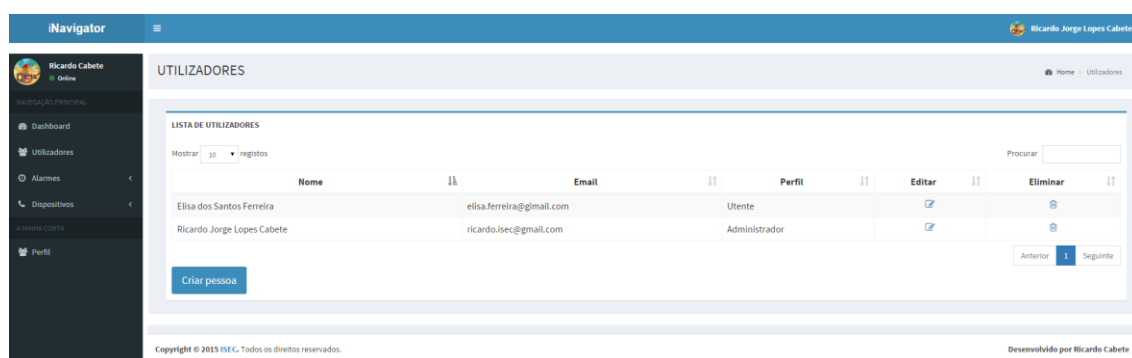


Figura 28 - Explosão das subvistas que constituem uma vista final.

O motor de templating Blade

O *Blade* é o motor de *templating* incluído na *framework Laravel*. Ao contrário de outros motores de *templating*, o *Blade* não impede o utilizador de usar código *php* nos ficheiros das vistas mas fornece um conjunto alargado (e que até pode ser expandido pelo utilizador) de diretivas para que o código fique mais simples de ler, depurar e manter.

Na secção anterior já foram apresentadas as diretivas `@include()` e `@yield()`. Aquando da construção do conteúdo de cada subvista usam-se, para a construção do layout, as diretivas `@extends()` e `@section()`.

No exemplo seguinte reproduz-se, de forma simplificada, a anatomia de uma vista. A diretiva `@extends()` indica ao *Blade* que esta vista deve usar a *master page* definida para a aplicação. Em seguida, a diretiva `@section('css')` define todo o CSS que é exclusivo a esta vista. Note-se que a diretiva `@section()` deve ser terminada com a diretiva `@stop`. Após

a inclusão do CSS especifica-se a secção *headerPage* que inclui o título da página e as *breadcrumbs*, a seguir à qual se especifica a secção conteúdo, onde estará o grosso do código da vista. Finalmente especifica-se a secção scripts onde se incluem os ficheiros JS que suportam a vista.

```
@extends('layouts.master')

@section('css')
/* ... */
@stop

@section('headerPage')
<h1>
    UTILIZADORES
</h1>
<ol class="breadcrumb">
    <li><a href="#"><i class="fa fa-dashboard"></i> Home</a></li>
    <li class="active">Utilizadores</li>
</ol>
@stop

@section('conteudo')
/* ... */
@stop

@section('scripts')
<script src="{{ URL::asset('js/pessoas.js')}}" type="text/javascript"></script>
@stop
```

Figura 29 – Anatomia de uma vista com as diretivas do motor de *templating Blade*.

Para além das diretivas usadas para especificar o layout da página o motor de *templating Blade* também pode ser usado para iterar coleções de dados mantendo o código da vista claro e simples de ler e manter, conforme se pode verificar no exemplo seguinte.

```

<table id="tabelaPessoas" class="table table-striped table-bordered table-hover">
  <thead>
    <tr>
      <th style="text-align: center;">Nome</th>
      <th style="text-align: center;">Email</th>
      <th style="text-align: center;">Perfil</th>
      <th style="text-align: center;">Editar</th>
      <th style="text-align: center;">Eliminar</th>
    </tr>
  </thead>
  <tbody>
    @foreach($pessoas as $key => $value)
      <tr>
        <td>{{ $value->pessoas_nome_completo }}</td>
        <td>{{ $value->pessoas_email }}</td>
        @if ($value->perfil->perfis_descricao)
          <td>{{ $value->perfil->perfis_descricao }}</td>
        @else
          <td>Sem perfil associado</td>
        @endif
        <td style="text-align: center;"><i class="fa fa-pencil-square-o"></i></td>
        <td style="text-align: center;"><i class="fa fa-trash-o "></i></td>
      </tr>
    @endforeach
  </tbody>
</table>

```

Figura 30 – Iteração de uma coleção com a diretiva `@foreach`.

O exemplo anterior ilustra o preenchimento de uma *datatable* e faz uso das diretivas `@foreach()`, `@if()` e `{{ }}`, estas últimas usadas para escapar para o ecrã o conteúdo de variáveis. Se o conteúdo das variáveis contiver *tags* HTML devem ser usadas as diretivas `{!! !!}` uma vez que as diretivas `{{ }}` não permitem o escape destas *tags*.

5.2 A APLICAÇÃO MÓVEL

A aplicação móvel para dispositivos *Android* foi codificada em Java com recurso ao SDK oficial. A programação em Java já é, pela natureza da linguagem, orientada a objetos e os padrões de design surgem de forma natural.

As aplicações *Android* organizam-se em torno de *Activity* (atividade). Uma atividade é, em termos gerais, um ecrã no qual se processa a interação com o utilizador. Daqui se infere que cada atividade tem uma vista e uma classe, que funciona como suporte aos elementos dispostos na vista.

5.2.1 Atividades

As vistas em *Android* são configuradas com recurso a ficheiros XML. Alternativamente pode ser usado o editor gráfico disponibilizado pelo IDE. Como é comum neste tipo de programação existem vários gestores de layouts que permitem dispor os elementos de forma a assegurar a sua correta adaptação aos vários tamanhos de ecrã dos dispositivos.

Cada vista tem associada uma classe que *extends Activity* e onde é disponibilizado o código que gere o comportamento quer dos elementos gráficos da aplicação (*callbacks* dos botões, por exemplo) quer do que deve ser feito em reação ao ciclo de vida.

As atividades *Android* seguem um ciclo de vida bem definido. Cada fase do ciclo de vida invoca um método que pode ser reescrito (*@Override*) para modificar o comportamento definido por omissão.

O ciclo de vida das aplicações está ilustrado no diagrama seguinte [44]. Os métodos mais importantes são o *.onCreate()*, invocado quando a atividade é criada e que acaba por servir como construtor informal; o método *.onStart()* invocado aquando da execução da atividade; o método *.onPause()*, invocado quando a atividade perde *focus*; o método *.onStop()*, invocado quando a atividade deixa de estar visível e o método *.onDestroy()*, invocado quando a atividade é destruída e que serve de destrutor informal (a linguagem Java usa *garbage collector* e, como tal, não tem destrutor formal para as suas classes).

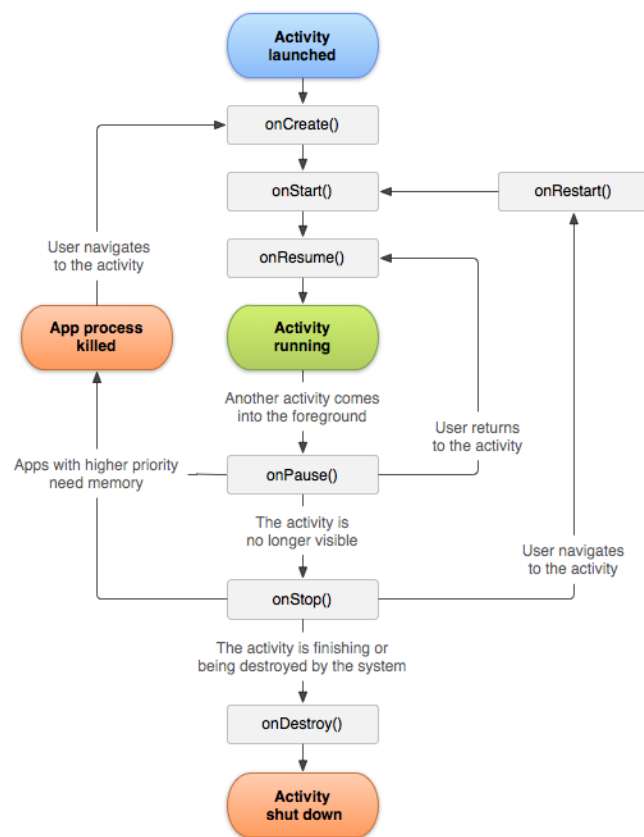


Figura 31 - Ciclo de vida das atividades no sistema operativo *Android*.

5.2.2 Serviços e *broadcast receivers*

Para além das atividades, executadas a pedido do utilizador, o *Android* permite criar serviços que ficam em execução permanente no sistema e são invocados em reação a eventos.

À semelhança das atividades os serviços também seguem ciclos de vida bem definidos, ainda que mais simples que os das atividades. O diagrama seguinte [45] mostra o ciclo de vida dos dois tipos de serviço disponibilizados.

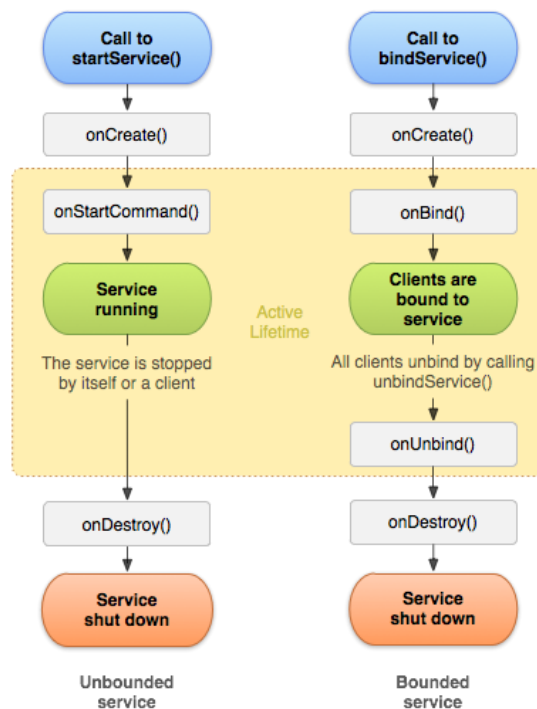


Figura 32 - Figura 5 - Ciclo de vida dos serviços no sistema operativo *Android*.

O *iNavigator* faz uso de um serviço responsável por manter ativa a ligação MQTT, processar a configuração enviada pelo servidor e definir e eliminar alarmes.

Para além dos serviços o *Android* permite a criação de *broadcast receivers* que são, em termos gerais, *callbacks* que reagem a eventos específicos. Por exemplo, aquando da receção de uma SMS o sistema lança um *broadcast* que pode ser lido por qualquer aplicação que tenha criado um *broadcast receiver* para o efeito.

O *iNavigator* define vários *broadcast receivers*: um para controlar o *boot* do sistema, assegurando que a aplicação é executada quando o equipamento arranca; outro para controlar a receção de SMS; outro para controlar o momento de ativação dos alarmes e um último para controlar o estado da Internet e que funciona como *watchdog* do serviço, avisando-o nos momentos de perda ou restauro de conectividade à rede.

5.3 A INTERAÇÃO ENTRE AS APLICAÇÕES

As duas aplicações que constituem o *iNavigator* não seriam tão úteis nem compreenderiam tantos casos de uso se não comunicassem entre si. A comunicação entre aplicações pode fazer-se por vários meios e padrões. Nesta seção descreve-se a forma como essa comunicação se processa quer na comunicação entre servidor e cliente, quer também na comunicação entre clientes.

5.3.1 Comunicação Servidor -> Cliente e Cliente->Cliente

A comunicação entre o servidor e o cliente não é tão trivial quanto parece. Os dispositivos móveis têm restrições inerentes à sua natureza, sendo a principal o consumo de energia e consequentes repercussões na autonomia.

O protocolo HTTP é particularmente agressivo no consumo de bateria. Esta característica não é particularmente preocupante em dispositivos permanentemente ligados à corrente elétrica mas para os dispositivos da chamada IoT o consumo exagerado de energia pode ser catastrófico. O problema prende-se com o peso dos pedidos HTTP: um pedido HTTP necessita no mínimo de nove pacotes TCP, mas se considerarmos falhas de transmissão e que os *headers* são enviados em texto em claro pode dizer-se que o protocolo HTTP é demasiado verboso para ser uma opção viável [46] [47]. O tamanho dos pedidos HTTP não é um problema apenas pela energia que gasta: a quantidade acrescida de pacotes TCP significa que os custos da ligação, normalmente taxados ao megabyte, serão elevados.

Para colmatar esta dificuldade surgiram nos últimos anos vários protocolos de comunicação adequados à IoT de que são exemplo o MQTT, o *ActiveMQ*, o *STOMP* e o

XMPP. Estes protocolos foram concebidos com a comunicação M2M em mente e apresentam vantagens em pontos fulcrais a este tipo de comunicação:

- Latência – O uso de menos pacotes TCP significa que a comunicação tem menos *overhead* (o MQTT acrescenta apenas 2 bytes de *overhead*) e é mais rica em dados que efetivamente interessam o que permite que dispositivos com capacidade limitada de processamento consigam proceder à sua análise de forma mais rápida.
- Bateria – Quanto menos pacotes circularem na rede, menos bateria se gasta a enviá-los e a recebê-los.
- Largura de banda – Naturalmente, quanto menos pacotes forem enviados e recebidos menos largura de banda é consumida.

Stephen Nicholas comparou de forma relativamente informal os protocolos HTTPS e MQTT e concluiu que MQTT era 93 vezes mais rápido, tinha 12 vezes menos consumo energético para enviar dados, 170 vezes menos consumo energético para os receber, metade do consumo energético para manter a ligação aberta e 8 vezes menos *overhead* [48]. Uma das mais recentes vitórias do MQTT foi a sua adoção pelo *facebook* para a sua aplicação de chat.

A comunicação entre o servidor e o cliente do *iNavigator* recorre ao protocolo MQTT. A única desvantagem deste tipo de comunicação é que exige a presença de um broker, cuja instalação não é suportada pelos serviços de alojamento mais comuns. Na implementação atual, o *iNavigator* faz uso do broker de teste do *Mosquitto*, um broker *open source* que implementa o protocolo MQTT.

A adoção deste broker de teste significa que o *iNavigator* tem uma fragilidade evidente. Esta opção, porém, prendeu-se com a possibilidade contratar um serviço de alojamento para a aplicação web de baixo custo. A comunicação segue então o caminho Servidor -> Broker -> Cliente e, nos muitos testes efetuados, funcionou sempre sem problemas: o MQTT é rápido, fiável e o broker *Mosquitto* muito simples de configurar. O diagrama abaixo ilustra a cadeia de comunicação entre servidor e dispositivo móvel e torna evidente a fragilidade identificada.

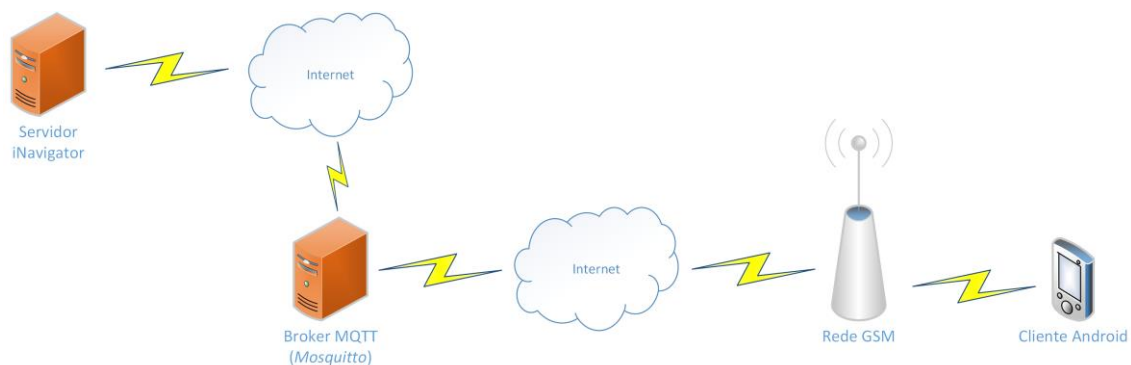


Figura 33 - Diagrama da cadeia de comunicação entre servidor e dispositivo móvel.

Outra vantagem importante deste tipo de comunicação é o facto de, no cliente, ser possível definir um *callback* que reaja à publicação de novas mensagens no broker o que evita soluções menos elegantes de *polling* ou a configuração de *sockets* que deverão estar continuamente à espera de receber informação. Desta forma é possível fazer *push* de novas configurações para o cliente, desde que esteja disponível uma ligação à Internet.

A comunicação entre clientes também fez uso do MQTT. Apesar de originalmente ter sido considerada a comunicação via SMS, rapidamente se percebeu que as SMS não têm fiabilidade para poderem constituir ferramenta de comunicação.

5.3.2 Comunicação Cliente -> Servidor e Cliente -> Cliente

Contrariando o exposto na secção anterior, a comunicação entre o cliente e o servidor é feita por HTTP ao invés de MQTT mas a razão para assim ser não se prende com o mérito do HTTP nem com o demérito do MQTT: a razão é mais simples e prende-se com uma limitação natural no *modus operandis* do *stack Apache - php*.

O fluxo seguido na execução de código em *php* é o seguinte: o servidor *Apache* (ou outro equivalente) compilam o código a executar e enviam-no para o processador *php* que o interpreta e o executa. Ora, em termos de ciclo de vida, o código é dado como terminado assim que é executado. Isto significa que não é possível definir um *callback* que reaja à publicação de uma mensagem no broker MQTT.

Sem um *callback* as opções de comunicação limitam-se: ou se cria um *poller* que, a intervalos regulares questionasse o broker acerca das novas mensagens ou se procurava outra solução de comunicação.

O modelo de comunicação adotado foi o modelo cliente servidor comum fazendo uso de pedidos HTTP. Como a comunicação entre o cliente e o servidor é pontual e reduzida não se fazem sentir de forma intensa as limitações deste protocolo.

6 IMPLEMENTAÇÃO

O *iNavigator*, por consistir numa aplicação web e numa aplicação móvel *Android*. Este capítulo apresentará a metodologia de trabalho adotada e uma vista geral das funcionalidades implementadas.

6.1 METODOLOGIA

O *iNavigator* foi desenvolvida por um único programador e, quando assim acontece, a tentação de não usar uma metodologia de trabalho definida é grande e deve ser combatida. Hoje em dias as metodologias mais utilizadas serão a *Waterfall* (com maiores ou menores adaptações) e o *Scrum* (também ele eventualmente adaptado) [49]. Apesar de ambas as metodologias terem as suas virtudes e defeitos [50] [51], optou-se por *Scrum*, por ser aquela que, dado o carácter experimental e de investigação de um trabalho académico desta natureza, permitiria entregar produtos mais rapidamente ao mesmo que deu oportunidade ao cliente de rever e modificar requisitos e prioridades de forma contínua durante o desenvolvimento [52].

A metodologia *Scrum* assenta no conceito de sprints e de *backlogs*. Uma sprint é um período de desenvolvimento, que pode ir tipicamente de 1 semana a 1 mês. Um *backlog* é uma lista com as funcionalidades: em *Scrum* existe o *product backlog*, que contém as funcionalidades desejadas para o produto e o *sprint backlog*, que contém a lista de funcionalidades a implementar numa sprint.

Para a gestão dos *backlogs* de *Scrum* foi usado o *Trello*, uma ferramenta online que implementa uma versão adaptada do sistema *Kanban*, um sistema de organização de

tarefas inicialmente pensado para linhas de montagem mas que pode ser combinado com o *Scrum* para a gestão do estado de cada tarefa [53].

Na imagem seguinte é apresentado o *Trello* durante os estádios iniciais de desenvolvimento da aplicação web. Na ferramenta criaram-se colunas para o *backlog*, para a identificação das funcionalidades desejadas; para a implementação, onde constam as funcionalidades que estão a ser implementadas; para os *bugs* identificados, para onde transitam as atividades onde tenham sido identificados bugs; para o controlo de qualidade, onde estão as atividades que em teste e, finalmente, uma coluna para as atividades já concluídas.

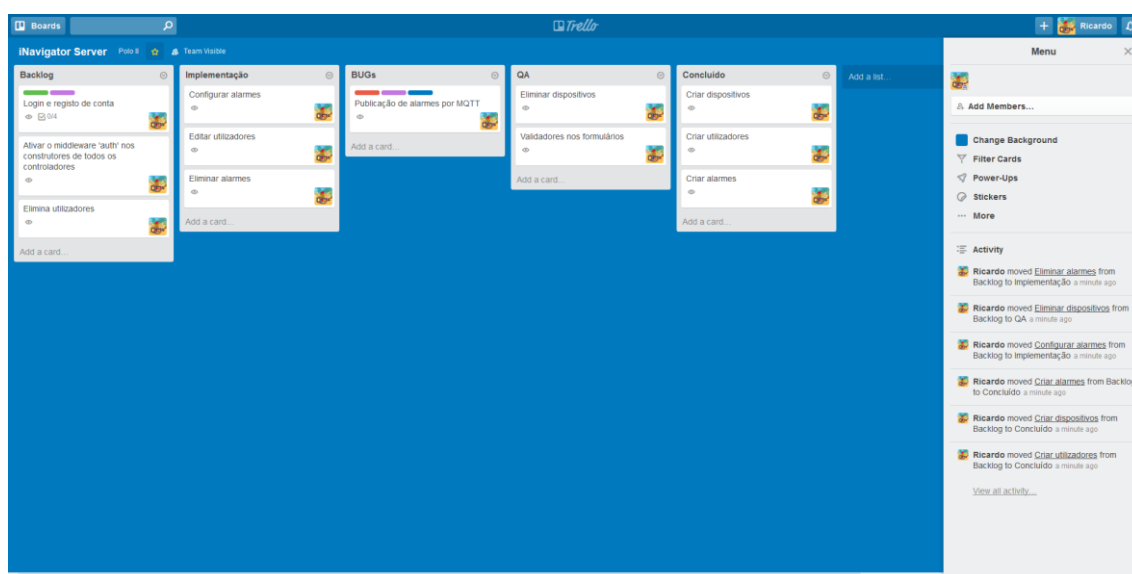


Figura 34 - Vista do *Kanban* na ferramenta *Trello* da aplicação web do *iNavigator*.

Após a configuração das várias colunas criam-se os vários cartões. Cada cartão corresponde a uma funcionalidade que vai transitando de coluna em coluna, em função do estado em que se encontra.

A ferramenta não exige fluxo linear, isto é, um cartão pode transitar de e para qualquer coluna, consoante a necessidade. Para além disso, podem acrescentar-se

marcadores coloridos aos cartões cuja legenda é definida pelos programadores (ex. marcador vermelho para funcionalidades que exijam atenção urgente).

O nível de detalhe de cada cartão é variável e, a qualquer momento, um cartão pode ser dividido em vários outros que descrevam determinada funcionalidade de forma mais discriminada. Também é possível atribuir programadores a determinadas atividades, definir prazos limite de conclusão e verificar o histórico de atividade de cada cartão.

Para além da adoção de uma metodologia de desenvolvimento de software foi usada também uma ferramenta de controlo de versões. Este tipo de ferramentas serve de repositório do código fonte de forma estruturada [54]: no final de cada etapa de desenvolvimento (normalmente ao final do dia) é possível submeter versões novas do código fonte que substituem as existentes que, no entanto, continuam a pertencer ao histórico do projeto e, por esse motivo, poder continuar ser acedidas.

A ferramenta adotada foi o *Apache Subversion*, coloquialmente designado apenas por SVN, por ser simples de configurar e utilizar e por haver com esta ferramenta alguma experiência prévia noutros projetos.

6.2 FUNCIONALIDADES DA APLICAÇÃO WEB

Nesta secção serão apresentadas as funcionalidades implementadas na aplicação web do servidor, acompanhadas de capturas de ecrã que permitam contextualizar cada funcionalidade na aplicação.

6.2.1 Autenticação, criação de conta e idiomas

À semelhança da maioria das aplicações web, o *iNavigator* exige que o utilizador se autentique com um nome de utilizador e uma palavra-passe. Esta exigência prende-se

com a necessidade de manter os conceitos de conta e de sessão que permitem ao servidor conhecer o utilizador e mostrar-lhe apenas a informação que lhe é relevante.

A imagem abaixo mostra o ecrã de login do *iNavigator*. Repare-se nas bandeiras da república portuguesa e do reino unido, cujo clique provoca a alternância de idioma entre português e inglês. Este ecrã dá também acesso a outro, muito similar, que permite registar uma nova conta que, como é hábito nestas aplicações, deve ser confirmada via correio eletrónico antes de ser usada.



The image shows a web application login screen for 'iNAVIGATOR'. At the top, the logo 'iNAVIGATOR' is displayed next to the flags of Portugal and the United Kingdom. Below this is a white box with the title 'AUTENTICAÇÃO'. Inside the box, there are two input fields: 'Endereço de correio eletrónico' with an envelope icon and 'Palavra-passe' with a lock icon. Below these fields is a checkbox labeled 'Manter a sessão iniciada'. A prominent blue button labeled 'Entrar' is positioned below the checkbox. At the bottom of the white box, there are two links: 'Esqueceu-se da palavra-passe?' and 'Criar uma conta'.

Figura 35 - Ecrã de login da aplicação web do *iNavigator*.

Se o utilizador se esquecer da sua palavra-passe há acesso a um ecrã onde é possível voltar a introduzir o endereço de correio eletrónico e pedir que seja enviada outra palavra-passe.

6.2.2 A dashboard

Após a autenticação no sistema a aplicação mostra a *dashboard* onde é apresentado o relatório de atividade e alguma informação estatística, conforme ilustrado na imagem seguinte, que representa uma vista geral da *dashboard*.

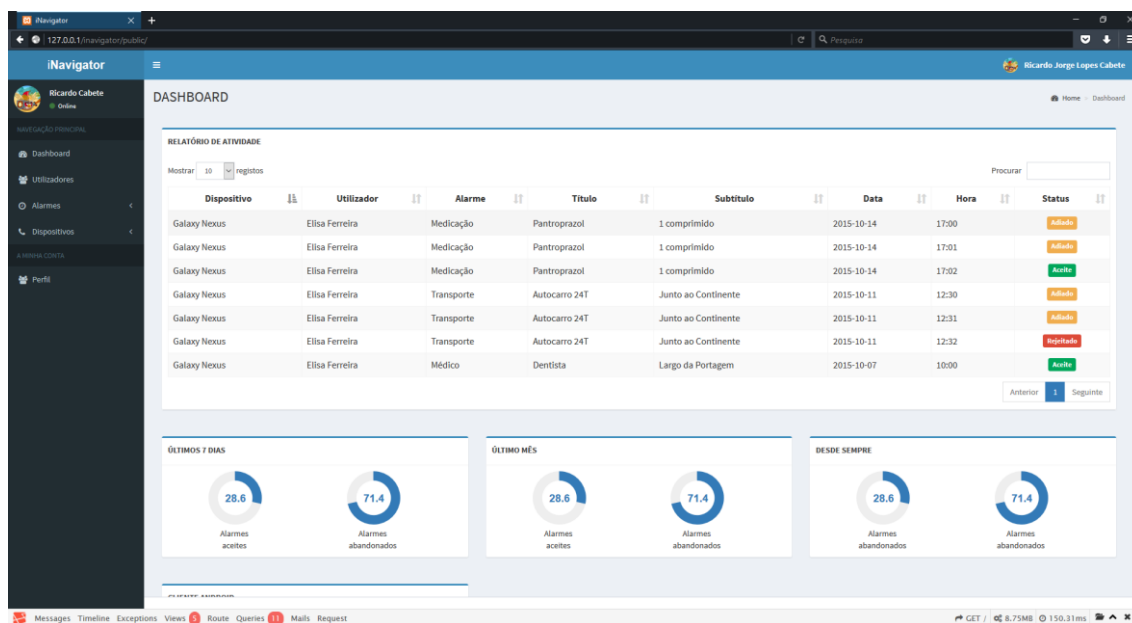


Figura 36 - Vista geral da *dashboard* do iNavigator.

O lado esquerdo da imagem anterior mostra o menu de navegação. É através dele que o utilizador executa as operações disponibilizadas pelo sistema: gerir os vários elementos (utilizadores, alarmes e respetivas categorias e dispositivos) e os associa entre si.

No top da *dashboard* surge relatório de atividade onde é possível seguir o estado de cada alarme: é mostrada a identificação do dispositivo, do utilizador que lhe está associado e os detalhes de cada alarme com a informação do seu estado que pode variar entre aceite, adiado ou rejeitado. Os relatórios mostrados na *dashboard* não são obtidos automaticamente: é necessário que o servidor solicite ao cliente o envio do relatório para

que estes dados sejam atualizados. A imagem seguinte mostra a tabela de relatório de atividade.

DASHBOARD

Home Dashboard

RELATÓRIO DE ATIVIDADE

Mostrar 10 registos

Procurar

Dispositivo	Utilizador	Alarme	Título	Subtítulo	Data	Hora	Status
Galaxy Nexus	Elisa Ferreira	Medicação	Pantoprazol	1 comprimido	2015-10-14	17:00	Aceite
Galaxy Nexus	Elisa Ferreira	Medicação	Pantoprazol	1 comprimido	2015-10-14	17:01	Aceite
Galaxy Nexus	Elisa Ferreira	Medicação	Pantoprazol	1 comprimido	2015-10-14	17:02	Aceite
Galaxy Nexus	Elisa Ferreira	Transporte	Autocarro 24T	Junto ao Continente	2015-10-11	12:30	Aceite
Galaxy Nexus	Elisa Ferreira	Transporte	Autocarro 24T	Junto ao Continente	2015-10-11	12:31	Aceite
Galaxy Nexus	Elisa Ferreira	Transporte	Autocarro 24T	Junto ao Continente	2015-10-11	12:32	Rejeitado
Galaxy Nexus	Elisa Ferreira	Médico	Dentista	Largo da Portagem	2015-10-07	10:00	Aceite

Anterior 1 Seguinte

Figura 37 - Vista do relatório de atividade.

Por baixo do relatório de atividade é também mostrada alguma informação estatística, nomeadamente a percentagem de alarmes aceites e abandonados nos últimos 7 dias, no último mês (30 dias) e desde sempre. A imagem seguinte mostra a secção estatística da *dashboard* preenchida com dados de teste.



Figura 38 - Secção estatística da *dashboard*.

Por fim existe ainda uma hiperligação para a loja de aplicações da Google onde é possível descarregar a aplicação para dispositivos móveis.

6.2.3 Gestão de elementos

A apresentação das funcionalidades ao longo deste capítulo teve o cuidado de seguir os passos lógicos que o utilizador deverá seguir na aplicação: apesar de não ser obrigatório, sugere-se que primeiro sejam criadas as pessoas, seguidas das categorias de alarmes, dos alarmes e dos dispositivos.

CRUD de pessoas

O CRUD é um acrónimo que representa as operações de criação (*create*), leitura (*read*), atualização (*update*) e eliminação (*delete*) de dados. Numa aplicação web a maioria das entidades são configuráveis pelo utilizador pelo que é necessário dar-lhe acesso às operações típicas do CRUD.

Um dos formatos típicos de um CRUD web é uma tabela onde, com recurso a hiperligações ou botões, é possível criar, atualizar ou eliminar instâncias de determinada entidade. Na imagem seguinte pode ver-se o CRUD de pessoas. Repare-se nos botões editar e eliminar, presente para cada registo na tabela. Por baixo da tabela há um botão que permite criar novas pessoas.

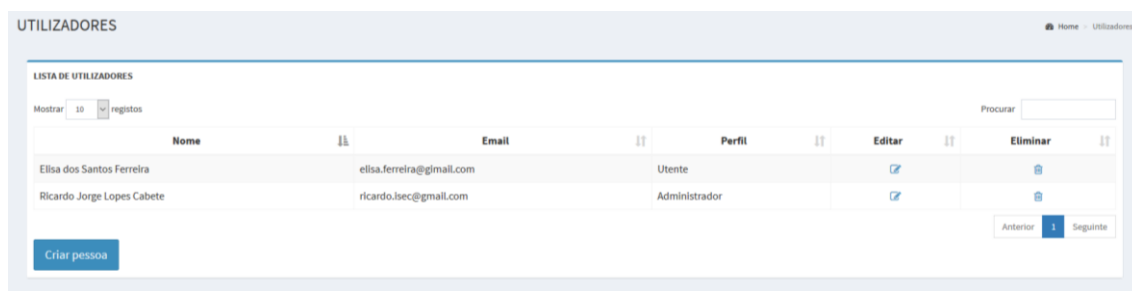


Figura 39 - CRUD de pessoas do *iNavigator*.

Num CRUD há uma funcionalidade que é particularmente destrutiva: a eliminação de instâncias de uma entidade. No *iNavigator* houve a preocupação de

confirmar todas as opções destrutivas para evitar que o utilizador perdesse informação na sequência de um clique accidental. Na imagem seguinte ilustra-se a janela modal (uma janela subordinada que cria um modo de interação capaz de impedir a utilização do ecrã principal) que serve de confirmação à operação de eliminação de uma pessoa.

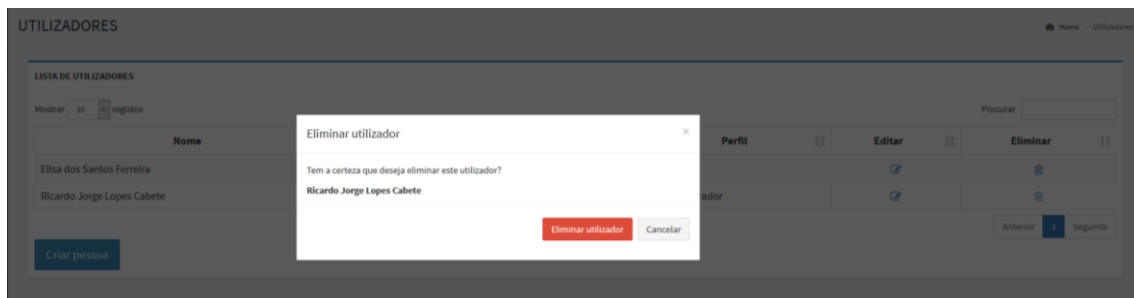


Figura 40 - Janela modal de confirmação à operação de eliminação de uma pessoa.

Na figura seguinte pode ver-se a janela modal que permite a edição dos dados de uma pessoa. O perfil não é modificável por se tratar do administrador da conta.

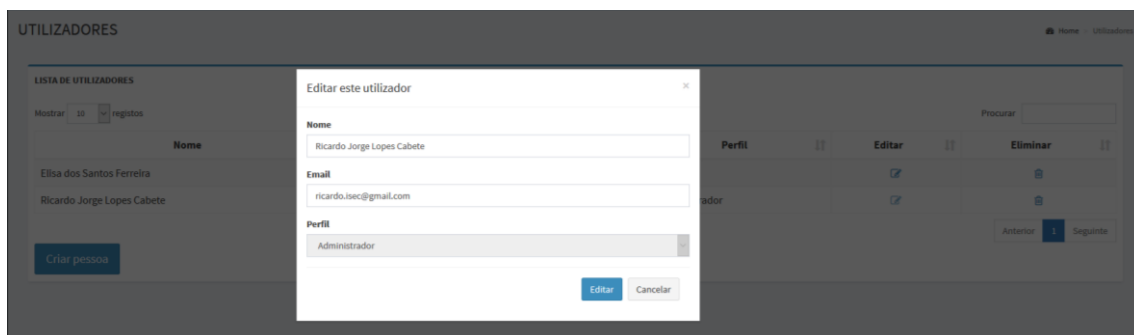


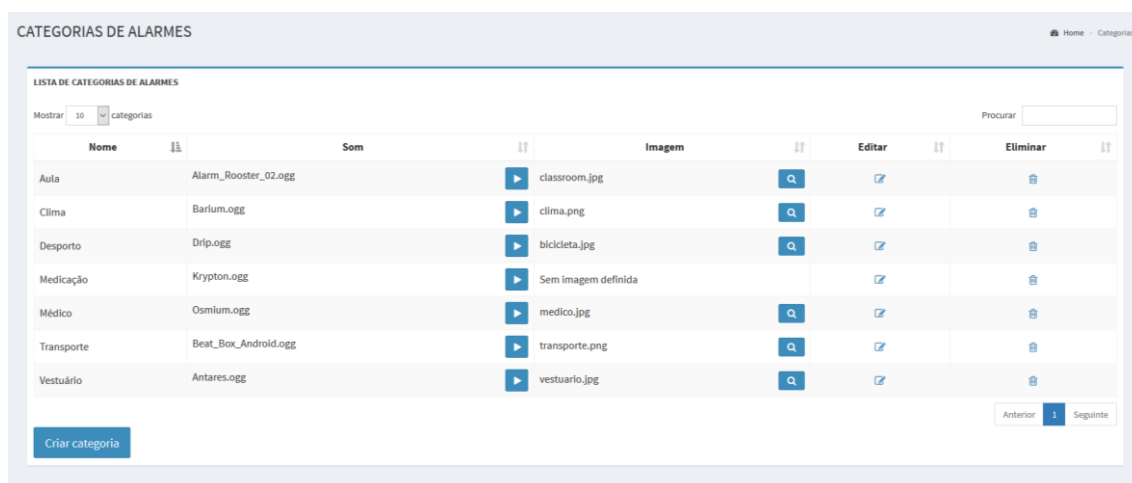
Figura 41 - Janela modal de edição dos dados de uma pessoa.

CRUD de categorias de alarmes

O conceito de CRUD e o *modus operandis* das janelas modais associadas às operações de edição e eliminação de instâncias de entidades é similar em toda a aplicação. Por esse motivo não faz sentido abordar estes tópicos em cada CRUD restante. Assim sendo, optou-se por mostrar apenas o que é diferente em cada um deles.

As categorias de alarmes permitem criar um sistema de classificação capaz de agrupar alarmes. Assim, é possível indicar ao *iNavigator* que um conjunto de alarmes pertence à categoria Aula ou Clima, por exemplo e, por esse motivo, partilharão a mesma imagem e som aquando da sua visualização na aplicação móvel.

A imagem abaixo mostra o CRUD das categorias de alarmes. Cada categoria é configurada com uma imagem e com um som que pode ser personalizado e que serão automaticamente aplicados a todos os alarmes da mesma categoria. Na tabela há ainda botões que permitem ouvir o som ou visualizar a imagem recorrendo a uma janela modal.



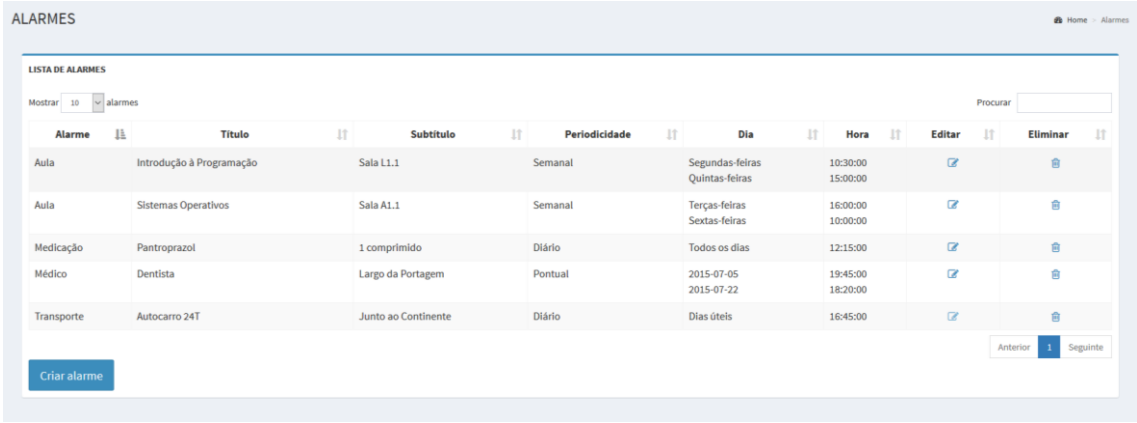
Nome	Som	Imagem	Editar	Eliminar
Aula	Alarm_Rooster_02.ogg	classroom.jpg		
Clima	Barium.ogg	clima.png		
Desporto	Drip.ogg	bicicleta.jpg		
Medicação	Krypton.ogg	Sem imagem definida		
Médico	Osmium.ogg	medico.jpg		
Transporte	Beat_Box_Android.ogg	transporte.png		
Vestuário	Antares.ogg	vestuario.jpg		

Figura 42 - CRUD das categorias de alarmes do *iNavigator*.

CRUD de alarmes

Os alarmes são as entidades principais do *iNavigator*. Um alarme tem um título e um subtítulo e pertence obrigatoriamente a uma categoria herdando, por omissão, os seus som e imagem. Cada alarme pode ter mais do que uma instância, isto é, o mesmo alarme pode ocorrer mais do que uma vez (por exemplo, uma aula pode ocorrer às segundas, quintas e sextas, a horas diferentes, sem que para isso seja necessário criar três alarmes distintos).

A imagem seguinte ilustra o CRUD de alarmes. A coluna dia é particularmente rica porque permite verificar a periodicidade (no fundo, em que dias ocorrerá) de cada alarme. As periodicidades podem incluir dias isolados ou grupos de dias, identificados como dias úteis, fins de semana e todos os dias.



ALARMES

LISTA DE ALARMES

Mostrar 10 alarmes

Procurar

Alarme	Título	Subtítulo	Periodicidade	Dia	Hora	Editar	Eliminar
Aula	Introdução à Programação	Sala L1.1	Semanal	Segundas-feiras Quintas-feiras	10:30:00 15:00:00		
Aula	Sistemas Operativos	Sala A1.1	Semanal	Terças-feiras Sextas-feiras	16:00:00 10:00:00		
Medicação	Pantoprazol	1 comprimido	Diário	Todos os dias	12:15:00		
Médico	Dentista	Largo da Portagem	Pontual	2015-07-05 2015-07-22	19:45:00 18:20:00		
Transporte	Autocarro 24T	Junto ao Continente	Diário	Dias úteis	16:45:00		

Anterior 1 Seguinte

Criar alarme

Figura 43 - CRUD de alarmes do *iNavigator*.

Para que a programação de alarmes não fosse demasiado complexa entendeu-se limitar as instâncias de cada alarme a um único tipo de periodicidade. Assim, se um alarme for de periodicidade pontual (os alarmes pontuais ocorrem num determinado instante e não se repetem), por exemplo, não pode ser definido cumulativamente como semanal, que implicaria a sua repetição.

A imagem seguinte mostra o ecrã de configuração de um alarme. Note-se que, apesar deste alarme pertencer à categoria medicação e, por esse motivo, herdar os respetivos som e imagem, é possível especificar som e imagens específicos para este alarme em particular que substituirão os da sua categoria. A parte inferior da imagem mostra a criação de instâncias ao alarme, de periodicidade pontual e que, neste caso, ocorrerá em dois momentos temporais.

ALARMES Home > Alarmes > Editar alarme

Editar Alarma

Categoria
Médico

Título
Dentista

Subtítulo
Largo da Portagem

Som
Drag & drop files here ...
Escolher ficheiro

Imagem
Drag & drop files here ...
Escolher ficheiro

Periodicidades
Pontual

Data
[Calendar icon]

Hora
20:39
Adicionar à tabela

Dia	Hora	Eliminar
2015-07-05	19:45:00	[Trash icon]
2015-07-22	18:20:00	[Trash icon]

Voltar Gravar alterações

Figura 44 - Ecrã de edição dos atributos de um alarme.

CRUD de dispositivos

Os dispositivos representam os equipamentos *Android* onde a aplicação móvel será instalada. O *iNavigator* providencia também um CRUD de dispositivos que tem a dupla função de gerir estas entidades e de os associar a pessoas.

A imagem seguinte mostra o CRUD de dispositivos do *iNavigator*. Na tabela é visível o ID do dispositivo que será usado para a identificação do canal de comunicação MQTT entre o servidor e o cliente. O número do cartão é também fundamental porque as aplicações cliente enviam SMS entre si.

DISPOSITIVOS Home > Dispositivos

LISTA DE DISPOSITIVOS

Mostrar 10 registos Procurar

Descrição	Utilizador	Tipo	SO	ID Dispositivo	Cartão	Editar	Eliminar
Galaxy Nexus	Elisa Ferreira	Smartphone	Android	351554059153545	914717733	[Edit icon]	[Trash icon]
Huawei G300	Ricardo Cabete	Tablet	Android	861229016049630	913363969	[Edit icon]	[Trash icon]

Anterior 1 Seguinte

Criar dispositivo

Figura 45 - CRUD de dispositivos do *iNavigator*.

6.2.4 Alarmes

Associação de alarmes a dispositivos

Após a criação de todas as entidades é necessário fazer as suas associações: indicar a que pessoa pertence um dispositivo, a que categorias pertence um alarme e que alarmes deverão ficar associados a que dispositivo. Todas estas associações já foram resolvidas nos respetivos CRUDs exceto a associação entre alarmes e dispositivos.

A imagem seguinte mostra o ecrã de associação entre alarmes e dispositivos.

ALARMES DOS DISPOSITIVOS

Mostrar 10 registos

Procurar

Descrição	Utilizador	Alarmes	Periodicidade	Dia	Hora	Associar alarmes
Galaxy Nexus	Elisa Ferreira	Medicação: Pantoprazol	Pontual	2015-11-25	20:30:00	
		Médico: Dentista	Pontual	2015-07-05	19:45:00	
		Médico: Dentista	Pontual	2015-07-22	18:20:00	
		Transporte: Autocarro 24T	Diário	Dias úteis	16:45:00	
		Aula: Sistemas Operativos	Semanal	Terças-feiras	16:00:00	
		Aula: Sistemas Operativos	Semanal	Sextas-feiras	10:00:00	
Huawei G300	Ricardo Cabete	Sem alarmes associados				

Anterior 1 Seguinte

Figura 46 - Tabela onde se visualizam as associações entre alarmes e dispositivos.

A imagem seguinte mostra a associação entre alarmes e dispositivos.

ASSOCIAR ALARMES AO DISPOSITIVO

Home Dispositivos Associar alarmes

Associar alarme

Descrição
Galaxy Nexus

Tipo
Smartphone

SO
Android

ID do dispositivo
351554059153545

Número do cartão
914717733

Utilizador
Elisa Ferreira

Alarmes
Selecione um alarme

Associar alarme

Mostrar 10 registos

Procurar

Alarme	Periodicidade	Dia	Hora	Eliminar
Autocarro 24T	Diário	Dias úteis	16:45:00	
Dentista	Pontual	2015-07-05 2015-07-22	19:45:00 18:20:00	
Pantoprazol	Pontual	2015-11-25	20:30:00	
Sistemas Operativos	Semanal	Terças-feiras Sextas-feiras	16:00:00 10:00:00	

Anterior 1 Seguinte

Voltar

Figura 47 - Ecrã se define a associação entre alarmes e dispositivos

Publicação de alarmes

Quando a associação entre alarmes e dispositivos está terminada falta apenas a sua publicação. A publicação de alarmes no dispositivo é considerada uma operação destrutiva porque o dispositivo elimina todos os alarmes já existentes quando recebe uma publicação.

Como a comunicação MQTT é *best effort* e, muitas vezes, os dados móveis e o Wi-Fi são desativados automaticamente pelo dispositivo para poupar bateria, não há maneira de saber se a publicação foi bem sucedida. Para ultrapassar esta dificuldade a aplicação cliente comunica com o servidor para um *url* definido como *callback* para confirmar a publicação dos alarmes.

A imagem abaixo mostra o servidor a aguardar a confirmação por parte do cliente. Esta operação é feita com recurso a uma janela modal e tem um *timeout* de 30 segundos, ao fim dos quais a publicação é dada como não confirmada.

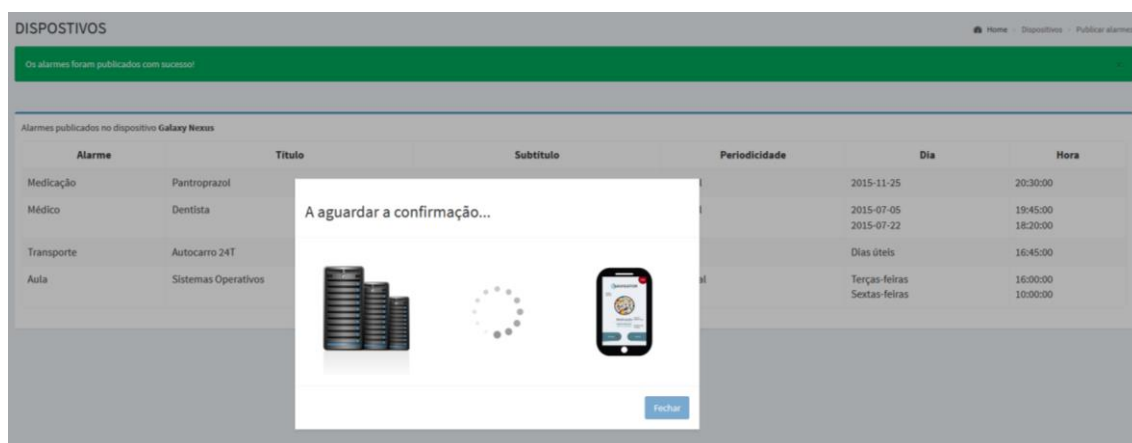


Figura 48 – Janela modal que aguarda a confirmação da publicação dos alarmes.

6.2.5 Dados do perfil

Os dados da conta do utilizador podem ser visualizados e editados num ecrã criado para o efeito. Este ecrã, ilustrado na imagem seguinte, está dividido em zonas identificadas como informação básica, informação adicional e imagem de perfil. Neste ecrã é ainda possível alterar a palavra-passe de acesso à aplicação.

The screenshot displays the 'EDITAR PERFIL' (Edit Profile) interface of the iNavigator application. The interface is structured as follows:

- Header:** Shows the user's name 'Ricardo Jorge Lopes Cabete' and a 'Home - Editar Perfil' link.
- Left Sidebar:** Contains navigation links: Dashboard, Utilizadores, Alarmes, Dispositivos, and Perfil.
- Main Content Area:**
 - Informação básica (Basic Information):** Fields for Name (Ricardo Jorge Lopes Cabete), Endereço de correio eletrónico (ricardo.isec@gmail.com), Telefone (212 345 678), and Telemóvel (914 737 733). An 'Atualizar' (Update) button is at the bottom right.
 - Informação adicional (Additional Information):** Fields for Morada (Rua das Flores, 145, 9/C Dto.), Código postal (3030-123), Freguesia (Sto António dos Olivais), Concelho (Coimbra), and Distrito (Coimbra). An 'Atualizar' (Update) button is at the bottom right.
 - Alteração da palavra-passe (Password Change):** Fields for Palavra-passe atual (masked with dots), Nova palavra-passe (masked with dots), and Confirmação da palavra-passe (masked with dots). An 'Atualizar' (Update) button is at the bottom right.
 - Imagem de perfil (Profile Image):** A section for uploading a profile picture, showing a preview of a cartoon bird wearing sunglasses. A file selection area below shows '1 file selected' and an 'Escolher imagem' (Choose image) button.
- Footer:** Copyright © 2015 iSEC. Todos os direitos reservados. Desenvolvido por Ricardo Cabete.

Figura 49 – Ecrã de visualização e adição dos dados do perfil.

A zona identificada como informação básica comporta o nome, endereço de correio eletrónico, telefone e telemóvel. A zona dedicada à informação adicional inclui a morada, o código postal, a freguesia, o concelho e, por fim, o distrito de residência do utilizador autenticado. É ainda possível submeter uma imagem de perfil e alterar a palavra-passe, digitando a palavra-passe atual e, em seguida, a palavra-passe pretendida duas vezes.

6.3 FUNCIONALIDADES DA APLICAÇÃO *ANDROID*

A aplicação móvel *Android* pode ser encontrada no site oficial de aplicações da Google. O download e instalação da aplicação são o primeiro passo para começar a usar o *iNavigator* mas, à semelhança da aplicação web, também aqui é necessária a autenticação para que o sistema saiba em que perfil deve configurar a aplicação.

A aplicação pode assumir dois perfis distintos: utente ou vigilante. O utente é aquele em cujo dispositivo ocorrerão os alarmes. Em oposição, o vigilante é aquele em cujo dispositivo se fará o controlo do sistema. O utente conta ainda com funcionalidades adicionais como a recolha de imagens, vídeos e sons.

Na maioria dos casos, a interface de utilização do *iNavigator* está escondida respeitando o estado de repouso dos dispositivos (*deep sleep*). No caso do utente, o caso de uso principal diz respeito à aceitação ou adiamento dos alarmes que estejam ativos em determinado momento. No caso do utente os casos de uso respeitantes à interação com alarmes não existem porque este perfil de utilização não admite a definição de alarmes.

A aplicação também pode ser executada de forma explícita. Neste caso é mostrado um menu contextual cujas opções mudam em função do perfil em que a aplicação esteja configurada. No caso do utente é possível encontrar funcionalidades que aceleram a comunicação com o vigilante; no caso do vigilante encontram-se funcionalidades que permitem monitorizar o utente.

O remanescente deste capítulo dá conta das funcionalidades implementadas na aplicação móvel *Android*, acompanhadas de capturas de ecrã que permitam contextualizar cada funcionalidade na aplicação.

6.3.1 Ambos os perfis – Autenticação

Assim que a aplicação é instalada é exigida a autenticação. Esta autenticação deve ser feita com os mesmos dados usados para a autenticação do utilizador na aplicação web do servidor. A imagem seguinte mostra o ecrã de autenticação com os dados (nome de utilizador e palavra-passe) já preenchidos.



Figura 50 – Ecrã de autenticação da aplicação móvel *Android*.

No fundo da imagem anterior, por cima do botão de validação de credenciais, é visível o ID do dispositivo. Este ID é calculado do seguinte modo: se for possível ler o número do cartão GSM usa-se esse valor. Se não for possível é usado o IMEI do equipamento. Se não for possível ler o número do cartão GSM nem o IMEI do equipamento (num *tablet*, por exemplo) é gerada uma *string* aleatória que deve ser usada aquando da configuração do dispositivo na aplicação web.

6.3.2 Perfil utente

Interação com alarmes

Os alarmes da aplicação móvel disparam à data e hora definidas. Nesse momento o dispositivo é despertado do *deep sleep* e é mostrado um ecrã similar à imagem seguinte que dá conta da categoria do alarme, do seu título e subtítulo. No centro do ecrã é mostrada a imagem da categoria ou do alarme e o dispositivo reproduz o som da categoria ou do alarme ao mesmo tempo que executa um padrão de vibração.



Figura 51 – Ecrã de alarme do *iNavigator*.

Quando este ecrã é mostrado o utilizador tem duas opções: aceitar ou adiar o alarme. A aceitação do alarme implica que ele parará de tocar imediatamente e não voltará

a tocar senão no agendamento previsto. Se o alarme for adiado ele ficará agendado para daí a um minuto. Se o alarme for adiado duas vezes é marcado como rejeitado. Estas ocorrências são registadas e comunicadas ao servidor aquando do envio do relatório.

Menu do utente

Se a aplicação for executada a pedido é mostrado um menu de botões cujo conteúdo depende do perfil do utilizador: utente ou vigilante. Nesta secção serão apresentados os vários menus visíveis na perspetiva do utente.

A imagem abaixo mostra o ecrã inicial com um menu de três botões. A legenda destes botões é autoexplicativa e, a partir deles, é possível tirar uma fotografia, gravar um vídeo ou gravar apenas som.



Figura 52 – Menu do *iNavigator* no perfil utente.

Estas funcionalidades fazem uso dos componentes oficiais do *Android*. A opção por estes componentes foi tomada no sentido de aproveitar a familiaridade que o utilizador já tem com eles, evitando assim a aprendizagem de componentes proprietários para realizar ações comuns.

Submenu SOS

No topo direito da imagem anterior era visível um botão vermelho redondo com a legenda SOS. Este botão dá acesso a um menu só disponível no perfil utente e que dá acesso a um submenu que facilita a comunicação com o vigilante. Na imagem seguinte pode ver-se os botões do menu utente.



Figura 53 – Menu SOS do *iNavigator* no perfil utente.

As opções disponíveis neste menu permitem ligar ou enviar uma SMS ao vigilante e solicitar a posição atual num mapa. Esta última opção exige que a localização GPS esteja ativada no dispositivo e que esteja disponível uma ligação à Internet.

A opção de ligar ao vigilante usa a componente oficial de chamadas de voz do *Android*, novamente para aproveitar a familiaridade que o utilizador já possa ter com o sistema operativo.

A opção de envio de SMS abre um submenu com alguns modelos de texto predefinidos. Se nenhum dos textos estiver ajustado ao contexto desejado, é possível escrever uma SMS com texto personalizado, também aqui fazendo uso da componente oficial de envio de SMS do *Android*. A imagem seguinte mostra o submenu de envio das SMS.

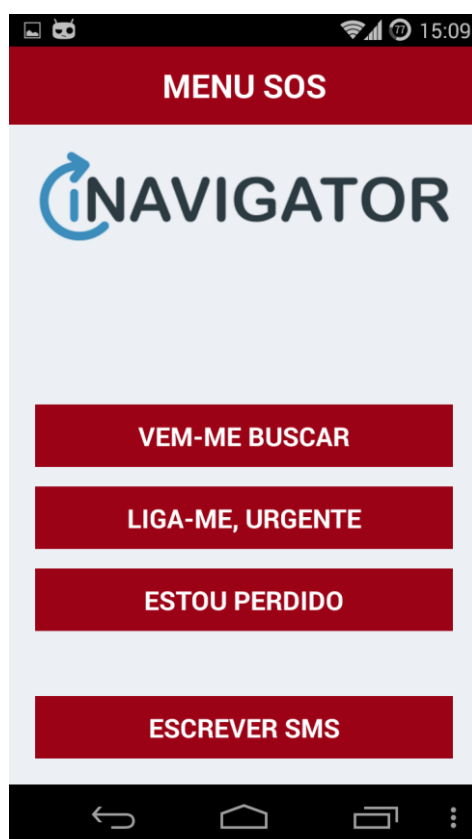


Figura 54 – Submenu de envio de mensagens SMS.

Os três botões superiores correspondem a textos predefinidos e o seu envio é automático, bastando para isso carregar no botão pretendido. O botão inferior permite o envio de SMS com texto personalizado.

A última opção do menu SOS permite a localização do dispositivo. Esta funcionalidade exige que o dispositivo consiga obter dados de posicionamento via GPS e ligação à Internet.

6.3.3 Perfil vigilante – Menu vigilante

Se a aplicação for executada a pedido e estiver configurada com o perfil vigilante é mostrado um menu que permite localizar o utente e pedir o relatório de estados de cada alarme. A imagem abaixo mostra o ecrã inicial com um menu de dois botões que permitem localizar o dispositivo do utente e solicitar o relatório de alarmes, respetivamente.

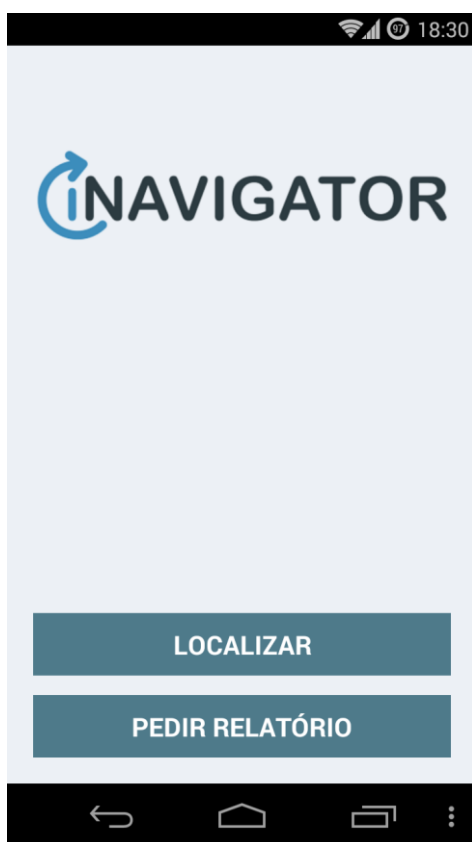


Figura 55 - Menu do *iNavigator* no perfil vigilante.

Para que a localização possa funcionar é necessário que o dispositivo do utente consiga obter dados de posicionamento via GPS. Como a visualização é feita do lado do dispositivo do vigilante, não é necessário que a aplicação do utente tenha ligação à Internet pelo que a comunicação entre os dispositivos é feita por SMS: a aplicação do vigilante solicita o envio das coordenadas GPS e aguarda a resposta do utente durante 60

segundos. Se o envio das coordenadas for bem sucedido é mostrado o ecrã com o posicionamento, conforme ilustrado na imagem seguinte.

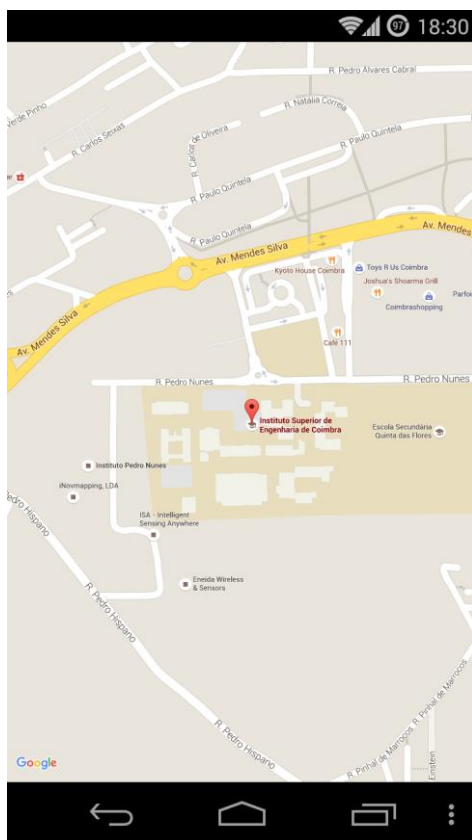


Figura 56 – Ecrã de localização do dispositivo.

7 VERIFICAÇÃO E VALIDAÇÃO

A verificação e validação (por vezes representada pela sigla V&V) de um projeto de engenharia consiste em levar a cabo procedimentos que têm por objetivo verificar se o sistema resultante cumpre os requisitos e a especificação apresentados e se cumpre o propósito para o qual foi construído. A verificação pretende aferir se o sistema cumpre os regulamentos e as especificações originalmente definidas e é, por definição, um processo interno. A validação, por seu lado, assegura que o sistema cumpre as necessidades do cliente ou de quaisquer outros *stakeholders*.

A verificação, validação e, eventualmente, a certificação são fases essenciais do ciclo de vida de qualquer sistema informático, cujo desenvolvimento não está completo sem que seja sujeito a testes rigorosos [55].

7.1 TESTES

A complexidade e extensão desta fase é suficiente para ser o foco de muitas teses de mestrado e doutoramento pelo que pareceu razoável adaptá-la à dimensão do projeto. Por esse motivo decidiu-se realizar apenas testes de aceitação por serem executados tipicamente pelo cliente ou outro *stakeholder* aquando da entrega do sistema e por terem por objetivo estabelecer confiança no sistema [56]. Os testes de aceitação são testes de caixa preta¹ e são executados aquando da entrega ao cliente do sistema. Dado o contexto académico do *iNavigator* entendeu-se que os testes deveriam ser executados pelo

¹ Por caixa preta entende-se o método de teste de software que analisa as funcionalidades de um sistema sem analisar o seu funcionamento interno.

programador que implementou o sistema mas, no mundo industrial, não é costume que assim seja.

O plano de testes do *iNavigator* foi especificado em folha de cálculo Microsoft Excel e dividido em seções, agrupando casos de uso de funcionalidade similar quer na aplicação web quer na aplicação móvel. Dada a natureza distinta das duas aplicações criadas foram especificados testes para a aplicação web e para a aplicação móvel *Android*.

Os testes da aplicação web incidiram sobretudo sobre cada uma das suas páginas, nomeadamente sobre a *dashboard*, os vários CRUDs, a associação entre alarmes e dispositivos, a publicação de alarmes, a obtenção de relatórios e a edição do perfil de utilizador.

Os testes à aplicação móvel *Android* incidiram sobre as várias *activity* que constituem a aplicação, nomeadamente os alarmes, os menus do utente e do vigilante, o menu SOS e também sobre a comunicação entre dispositivos e com o servidor.

No total foram considerados 220 testes que devem ser repetidos sempre que houver atualização significativa ao sistema. No caso de correção de bugs ou atualização menor podem ser repetidos apenas os testes dos casos de uso envolvidos. Sempre que o sistema for aumentado com novas funcionalidades o plano de testes deve ser revisto e, tipicamente, são-lhe acrescentados novos testes.

A imagem seguinte mostra a página inicial do plano de testes do *iNavigator*. Esta página serve de *dashboard* e permite aferir rapidamente quais os testes bem-sucedidos e os que falharam por secção de teste.

	A	B	C	D	E	F	G	H
1	iNavigator - Plano de testes de aceitação				Ricardo Cabete			
2								
3								
4	Página	Testes aprovados	Testes reprovados	Total de testes	Percentagem de sucesso			
5	WEB - autenticação	8	1	9	88,89%			
6	WEB - dashboard	5	3	8	62,50%			
7	WEB - CRUD pessoas	7	2	9	77,78%			
8	WEB - CRUD categorias	9	0	9	100,00%			
9	WEB - CRUD alarmes	5	0	5	100,00%			
10	WEB - CRUD dispositivos	6	1	7	85,71%			
11	WEB - Associar alarmes-dispositivos	4	3	7	57,14%			
12	WEB - Publicar alarmes	11	2	13	84,62%			
13	WEB - Obter relatório	8	5	13	61,54%			
14	WEB - Editar perfil	15	4	19	78,95%			
15	ANDROID - autenticação	8	3	11	72,73%			
16	ANDROID - menu utente	11	1	12	91,67%			
17	ANDROID - menu vigilante	15	2	17	88,24%			
18	ANDROID - menu SOS	14	6	20	70,00%			
19	ANDROID - localização	16	5	21	76,19%			
20	ANDROID - comunicação servidor	9	3	12	75,00%			
21	ANDROID - comunicação cliente	8	2	10	80,00%			
22	ANDROID - alarmes	12	6	18	66,67%			
23								
24	TOTAL	171	49	220	78,76%			
25								
26								
27								

Figura 57 - Primeira página do plano de testes do iNavigator.

Como se pode constatar, à data de captura do ecrã acima havia ainda 49 testes que apresentavam falhas, em praticamente todas as secções testadas. Como os testes de aceitação são executados aquando da entrega, é normal que as falhas estejam distribuídas de forma relativamente uniforme pela aplicação.

Na imagem seguinte pode ver-se a página com os detalhes dos testes à *dashboard* da aplicação web. Como os testes se destinavam a ser executados pelo mesmo programador que implementou a aplicação foi possível especificar os testes de forma particularmente lacónica. Se a equipa de testes fosse externa os testes teriam que ser especificados de forma mais exaustiva e, quando relevante, acompanhados dos valores esperados para que mais facilmente se aferisse o seu sucesso.

	A	B	C	D	E
1	Elemento	Descrição	Avaliação	Comentário	Data
2	Renderização	A página é corretamente renderizada, sem elementos desformatados.	OK	Os indicadores de percentagem aparecem desformatados no Google Chrome.	15/10/2015
3	Tabela de relatórios	A tabela de relatórios mostra os dados constantes na base de dados.	OK		15/10/2015
4	Estatística 7 dias	A percentagem de alarmes aceites é correta.	OK		15/10/2015
5	Estatística 7 dias	A percentagem de alarmes abandonados é correta.	OK		15/10/2015
6	Estatística último mês	A percentagem de alarmes aceites é correta.	OK		15/10/2015
7	Estatística último mês	A percentagem de alarmes abandonados é correta.	OK		15/10/2015
8	Estatística desde sempre	A percentagem de alarmes aceites é correta.	OK		15/10/2015
9	Estatística desde sempre	A percentagem de alarmes abandonados é correta.	OK		15/10/2015
10	Cliente Android	O link para a aplicação Android está correto.	OK		15/10/2015
11					

Figura 58 - Página dos testes à *dashboard* da aplicação web.

7.2 VALIDAÇÃO PELO CLIENTE

Apesar do *iNavigator* ter sido criado em contexto académico, a ideia do desenvolvimento desta aplicação surgiu na sequência de várias conversas informais entre o Dr. Sérgio Vicente, o professor doutor Jorge Henriques, da Universidade de Coimbra e o professor doutor Simão Paredes, orientador do projeto.

Assim, quando o projeto foi considerado implementado, em finais de setembro de 2015, iniciou-se uma fase importante de validação levada a cabo pelo Dr. Sérgio Vicente e pelo seu filho Francisco, portador de deficiência.

Originalmente prevista para durar 8 semanas, esta fase de validação foi substancialmente reduzida, devido a circunstâncias não controláveis na vida do utente que originaram uma interrupção abrupta no processo de validação que estava a ser desenvolvido. Em todo o caso, nas semanas subsequentes à entrega, a aplicação foi sendo usada e sujeita à avaliação por *stakeholders* externos ao desenvolvimento.

A aceitação da aplicação pelo utente foi um dos riscos identificados do sistema¹ mas, no caso do Francisco, o *iNavigator* foi muito bem aceite. O jovem seguiu sempre as

¹ Anexo I, documento de visão e âmbito, secção de riscos de negócio.

indicações do dispositivo móvel e respeitou as indicações que lhe eram dadas pelos vários alarmes o que constitui, de forma imediata, uma melhoria significativa no controlo da sua rotina diária.

Foi obtido feedback que consistiu essencialmente na identificação de falhas no sistema e sugestões para que o processo de gestão dos alarmes fosse mais fluido e intuitivo. Todas as falhas foram prontamente corrigidas e algumas das sugestões foram implementadas.

As que implicariam alterações substanciais no sistema foram registadas e constituirão trabalho futuro.

8 CONCLUSÕES E TRABALHO FUTURO

No final do projeto *iNavigator* é tempo de refletir no trabalho realizado e dele extrair ilações e aprendizagens para o futuro, tanto mais que este foi um projeto extremamente bem-sucedido: durante o tempo previsto foi possível conceber um sistema, especificá-lo, implementá-lo e entregá-lo a um cliente para validação.

Conforme previsto, foi criado um servidor na forma de uma aplicação web capaz de gerir de forma centralizada o sistema. Este servidor destina-se a ser usado apenas pelo vigilante e constitui o principal elemento diferenciador em relação a aplicações similares.

Foi também criada uma aplicação móvel unificada capaz de assumir os perfis utente e vigilante. A unificação desta aplicação representa uma vantagem técnica para o programador porque exige a gestão de uma única base de código mas também para os utilizadores que não têm que configurar múltiplas aplicações de interface potencialmente disruptivo entre si.

É ainda importante salientar que todos os casos de uso identificados e especificados durante a fase de análise de requisitos foram efetivamente implementados e que, no que diz respeito à relação entre a especificação e a implementação, o sistema está, para todos os efeitos, completo.

Por fim, uma nota final às tecnologias e arquiteturas adotadas que permitiram conceber um sistema verdadeiramente modular, baseado não só em soluções *well-known* mas também em boas práticas da indústria, o que torna simples a sua manutenção e expansão futura.

A experiência de desenvolvimento de um projeto facultou-me uma oportunidade privilegiada de pensar o desenvolvimento de forma integrada e contínua, de aplicar conhecimentos obtidos em unidades curriculares diversas e de explorar tecnologias de ponta, na sua maioria desconhecidas.

Esta experiência foi tão mais rica quanto se considera que me foi dada a oportunidade de concretizar as várias fases de desenvolvimento de um projeto, nomeadamente a especificação de requisitos, a conceção da arquitetura, a implementação, os testes e, finalmente, a entrega ao cliente.

Os projetos de cariz académico são espaços privilegiados para a experimentação e a investigação e o *iNavigator* não foi exceção. Os três meses dedicados à exploração de *Xamarin*, apesar de infrutíferos na perspetiva do produto final, foram talvez os mais ricos em termos pessoais não só pelos desafios de engenharia mas também por implicarem o caminhar para fora do trilho comumente percorrido.

A maturidade que se obtém no desenvolvimento completo de um projeto não tem paralelo com o tempo passado no contexto empresarial, onde cada programador tem a seu cargo um problema selecionado do sistema.

O *iNavigator* é um projeto de inovação e, por esse motivo, nunca estará verdadeiramente concluído. É sempre possível reiterar e acrescentar funcionalidades, fundamentais ou complementares que melhorem a experiência dos utilizadores, seja melhorando as aplicações envolvidas seja adaptando-as novos equipamentos.

A meu ver, a evolução a médio prazo poderia ser direcionada para:

- Efetivo suporte a vários idiomas com vista à internacionalização. A implementação atual suporta tecnicamente qualquer número de

idiomas mas, compreensivelmente, não se investiu na tradução exaustiva de todas as *strings*, tanto mais que essa tradução deve ser feita por profissionais.

- Adaptação da aplicação móvel *Android* para os *form factors* dos novos dispositivos móveis, nomeadamente *smartwatches* e *wearables*, que começam a ganhar a ubiquidade que o *smartphone* conquistou.
- Alarmes definidos por *context awareness*. Com a panóplia de sensores disponíveis nos dispositivos atuais pode dar-se autonomia ao dispositivo para que crie os alarmes que entenda adequados, nomeadamente em função da luminosidade, temperatura, altitude, intensidade sonora ou outros.
- Suporte a múltiplos vigilantes criando efetivamente uma relação muitos para muitos entre os vigilantes e os utentes. Esta funcionalidade permitiria que a aplicação fosse usada em instituições onde grupos de vigilantes (professores, enfermeiros, auxiliares, etc.) tivessem a seu cargo grupos de utentes (alunos, pacientes, idosos, etc.). Os casos de uso típicos poderiam ser, por exemplo, os lares de idosos, os hospitais e centros de saúde ou as escolas.
- A aceitação ou rejeição do alarme pode não ser feita com a simples pressão de um botão mas através de um elemento gráfico deslizante que tivesse que chegar ao destino. O caminho até lá poderia ser tão simples ou tão complexo quanto pretendido. O objetivo seria quebrar a aceitação ou rejeição automática que pode manifestar-se depois de os utentes terem familiaridade com o sistema.

- Possibilidade de definição de alarmes na própria aplicação, eventualmente através de um menu escondido para não aumentar a complexidade aparente da aplicação e acedível através de comandos especiais.
- Codificação de aplicações para outros sistemas operativos móveis. Infelizmente o esforço para que tal aconteça é muito significativo porque, apesar de em termos lógicos e arquiteturais a aplicação existente poder ser adaptada para outros sistemas operativos, é necessário recodificar não só as chamadas às APIs de sistema mas também reescrever todo o código existente porque cada sistema operativo é codificado com uma linguagem de programação diferente.

BIBLIOGRAFIA

- [1] S. Zhang, S. McClean, B. Scotney, P. Chaurasia e C. Nugent, "Using duration to learn activities of daily living in a smart home environment," *Pervasive Computing Technologies for Healthcare*, pp. 1-8, 2010.
- [2] H. T. Chaminda, V. Klyuev e K. Naruse, "A smart reminder system for complex human activities," 14th International Conference on Advanced Communication Technology (ICACT), pp. 235-240, 2012.
- [3] N. K. F. Khan, "Context base reminder system, supporting persons using smart phone accelerometer data," School of computing, Blekinge Institute of Technology, Karlskrona, Suécia, 2013.
- [4] D. Ferreira, "Monitorização de bio-sinais utilizando dispositivos móveis," Faculdade de engenharia da universidade do Porto, Porto, 2012.
- [5] H.-H. Hsu e C.-N. a. C. Y.-F. Lee, "An RFID-Based Reminder System for Smart Home," IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 264-269, 2011.
- [6] H. Wang, Y. Shi, D. Zhao e A. Y. C. Liu, "Automatic Reminder System of Medical Orders Based on Bluetooth," 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), pp. 1-4, 2011.
- [7] D. Asai, J. Orszulak, R. Myrick, C. Lee, J. F. Coughlin e O. L. d. Weck, "Contextaware reminder system to support medication compliance," IEEE International Conference on Systems, Man, and Cybernetics (SMC), p. 3213–3218, 2011.
- [8] A. Ginsberg, "Mental and Behavioral Health and Older Americans," [Online]. Available: <http://www.apa.org/about/gr/issues/aging/mental-health.aspx>.
- [9] A. Woods, E. Mahdavi e J. Ryan, "Treating clients with Asperger's syndrome and autism," *Child and Adolescent Psychiatry and Mental Health*, pp. 7-32, 2013.
- [10] D. Fidler, "The Emerging Down Syndrome Behavioral Phenotype in Early Childhood," *Infants & Young Children*, vol. 18 número 2, pp. 86-103, 2005.
- [11] Life360, "Life360 - The New Family Circle," [Online]. Available: <https://www.life360.com/>.
- [12] Runtastic, "Runtastic: Running, Cycling & Fitness GPS Tracker," [Online]. Available: <https://www.runtastic.com/>.

- [13] TIOBE, "TIOBE Index for November 2015," Novembro 2015. [Online]. Available: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.
- [14] D. Embry, "Why PHP Sucks," [Online]. Available: <http://webonastick.com/php.html>.
- [15] MichaelT, "Why is PHP so hated?," [Online]. Available: <http://programmers.stackexchange.com/questions/263389/why-is-php-so-hated>.
- [16] A. Kaseorg, "Why is PHP hated by so many developers?," [Online]. Available: <https://www.quora.com/Why-is-PHP-hated-by-so-many-developers>.
- [17] D. Stogov e Z. Suraski, "PHP RFC: Move the phpng branch into master," [Online]. Available: <https://wiki.php.net/rfc/phpng>.
- [18] O. Habib, "Comparing PHP 7 vs. HHVM," [Online]. Available: <https://blog.appdynamics.com/php/comparing-php-7-vs-hhvm/>.
- [19] P. Bissonnette, "Lockdown Results and HHVM Performance," 2015 Junho 9. [Online]. Available: <http://hhvm.com/blog/9293/lockdown-results-and-hhvm-performance>.
- [20] nikic, "Don't be STUPID: GRASP SOLID!," 27 Dezembro 2011. [Online]. Available: <https://nikic.github.io/2011/12/27/Dont-be-STUPID-GRASP-SOLID.html>.
- [21] P. Sparks, "Software Design Patterns in Laravel," 31 Agosto 2013. [Online]. Available: <http://pt.slideshare.net/sparksphill/software-design-patterns-in-laravel-by-phill-sparks>.
- [22] B. Skvorc, "What are the explicit advantages using a php framework versus building a complex web app from scratch?," 2012 Maio 16. [Online]. Available: <https://www.quora.com/What-are-the-explicit-advantages-using-a-php-framework-versus-building-a-complex-web-app-from-scratch>.
- [23] "Advantages and Disadvantages of Frameworks," [Online]. Available: <http://www.vizteams.com/blog/advantages-and-disadvantages-of-frameworks/>.
- [24] Prava-MFS, "Why to use Framework in PHP," 7 Outubro 2014. [Online]. Available: <http://www.codeproject.com/Articles/826642/Why-to-use-Framework-in-PHP>.
- [25] C. d. Wikipedia. [Online]. Available: <https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/2000px-MVC-Process.svg.png>.
- [26] B. Skvroc, "Best PHP Frameworks for 2014," 28 Dezembro 2013. [Online]. Available: <http://www.sitepoint.com/best-php-frameworks-2014/>.

- [27] N. Kolakowski, "Apple's Swift Is Killing Objective-C," 10 Outubro 2015. [Online]. Available: <http://insights.dice.com/2015/10/13/apples-swift-is-killing-objective-c/>.
- [28] Htmlpanda, "PhoneGap App Development's Advantages & Disadvantages: Let's Explore Thoroughly," 28 Julho 2014. [Online]. Available: <http://www.htmlpanda.com/blog/phonegap-app-developments-advantages-disadvantages-lets-explore-thoroughly/>.
- [29] agup, "What all mobile web devs should know about PhoneGap," 1 Janeiro 2014. [Online]. Available: <http://www.andygup.net/what-all-mobile-web-devs-should-know-about-phonegap/>.
- [30] RandyFreak, "Limitations of Cordova PhoneGap," 23 Agosto 2012. [Online]. Available: <https://cordovablogs.wordpress.com/2015/07/10/limitations-of-cordova-phonegap/>.
- [31] B. Kohan, "Native vs Hybrid / PhoneGap App Development Comparison," [Online]. Available: <http://www.comentum.com/phonegap-vs-native-app-development.html>.
- [32] "Create native iOS, Android,," [Online]. Available: <http://xamarin.com/platform>. [Acedido em 30 Novembro 2015].
- [33] Wikipedia, "Android (operating system)," [Online]. Available: http://en.wikipedia.org/wiki/Android_%28operating_system%29. [Acedido em 29 de Junho de 2013].
- [34] Google Android, "Página inicial," [Online]. Available: <http://www.android.com/>. [Acedido em 17 de Abril de 2013].
- [35] T. Williams, "Android Poised to Move from Phones and Tablets to Wider Embedded Applications," Fevereiro 2012. [Online]. Available: <http://www.rtcmagazine.com/articles/view/102484>.
- [36] Google Android, "Developer Tools," [Online]. Available: <http://developer.android.com/tools/index.html>. [Acedido em 30 de Abril de 2013].
- [37] C. Duckett, "Google releases Android Studio, kills off Eclipse ADT plugin," 9 Dezembro 2014. [Online]. Available: <http://www.zdnet.com/article/google-releases-android-studio-kills-off-eclipse-adt-plugin/>.
- [38] mqtt.org, "Frequently Asked Questions," [Online]. Available: <http://mqtt.org/faq>.
- [39] Wikipedia, "MQ Telemetry Transport," [Online]. Available: <http://en.wikipedia.org/wiki/Mqtt>. [Acedido em 28 de Agosto de 2013].
- [40] MQTT.org, "MQ Telemetry Transport," [Online]. Available: <http://mqtt.org/>. [Acedido em 15 de Maio de 2013].

- [41] "Simple MQTT client in Python," 11 Agosto 2014. [Online]. Available: <https://sakshambhatla.wordpress.com/2014/08/11/simple-mqtt-broker-and-client-in-python/>.
- [42] paho, "What is paho?," [Online]. Available: <http://www.eclipse.org/paho/>. [Acedido em 12 de Junho de 2013].
- [43] Mosquitto.org, "Mosquitto," [Online]. Available: <http://mosquitto.org/>. [Acedido em de Maio de 29 2013].
- [44] "Android Activity Lifecycle," [Online]. Available: <http://www.javatpoint.com/android-life-cycle-of-activity>.
- [45] "Android Service Tutorial," [Online]. Available: <http://www.javatpoint.com/android-service-tutorial>.
- [46] T. Kellog, "Why HTTP won't work for IoT," 7 Novembro 2014. [Online]. Available: <http://www.edn.com/electronics-blogs/eye-on-iot-/4437056/Why-HTTP-Won-t-Work-for-IoT>.
- [47] J. Speed, "REST is for sleeping. MQTT is for mobile," 3 Maio 2013. [Online]. Available: <https://mobilebit.wordpress.com/2013/05/03/rest-is-for-sleeping-mqtt-is-for-mobile/>.
- [48] S. Nicholas, "Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android," 31 Maio 2012. [Online]. Available: <http://stephendnicholas.com/archives/1217>.
- [49] G. Lister, "Project Management Roots and Understanding the PMBOK Guide," 6 Novembro 2014. [Online]. Available: <http://www.ftpress.com/articles/article.aspx?p=2261742&seqNum=12>.
- [50] M. Heusser, "Perils and Pitfalls of Agile Adoption," 3 Fevereiro 2006. [Online]. Available: <http://www.informit.com/articles/article.aspx?p=441115>.
- [51] ISTQB, "What is Waterfall model- advantages, disadvantages and when to use it?," [Online]. Available: <http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/>.
- [52] b. c. group, "Benefits Of Scrum," [Online]. Available: <https://www.braintrustgroup.com/scrum/benefits-of-scrum/>.
- [53] H. Kniberg e M. Skarin, "Kanban and Scrum - Making the Most of Both," 21 Dezembro 2009. [Online]. Available: <http://www.infoq.com/minibooks/kanban-scrum-minibook>.
- [54] S. Yeates, "What is version control? Why is it important for due diligence?," 1 Janeiro 2005. [Online]. Available: <http://oss-watch.ac.uk/resources/versioncontrol>.

- [55] E. Tran, "Verification/Validation/Certification," Primavera 1999. [Online]. Available:
http://users.ece.cmu.edu/~koopman/des_s99/verification/index.html.
- [56] ISTQB, "What is acceptance testing?," [Online]. Available:
<http://istqbexamcertification.com/what-is-acceptance-testing/>.

ANEXO I – DOCUMENTO DE VISÃO E ÂMBITO

1.1 REQUISITOS DE NEGÓCIO

1.1.1 Enquadramento

As tecnologias móveis podem desempenhar um papel particularmente relevante no apoio à realização de um conjunto de tarefas de complexidade variável por utilizadores com necessidades especiais. O objetivo deste trabalho prende-se com o desenvolvimento de um sistema móvel tendo por base um dispositivo móvel (*smartphone* ou *tablet*) para apoio a utilizadores com défices cognitivos ligeiros diretamente relacionados com memória e dificuldades de orientação.

As tecnologias móveis podem ser particularmente úteis num contexto em que o utilizador revela problemas cognitivos que dificultam a realização de um conjunto de tarefas da sua vida quotidiana. Com efeito, os problemas cognitivos podem estar relacionados com perda de memória, atenção, raciocínio, orientação, fala, sendo que a incapacidade na realização de tarefas pode manifestar-se de forma muito diversa, desde o simples esquecimento na realização de tarefas até situações que podem colocar em risco a sua segurança como desorientação, falha na toma de medicação. O envelhecimento da população agrava esta situação, uma vez que potencia o surgimento de um conjunto de patologias (demência, *Alzheimer*, ...) em que este tipo de incapacidade é preponderante. Existem ainda outras patologias em que estes défices cognitivos são particularmente relevantes, como por exemplo síndrome de *Down* e de *Asperger*.

1.1.2 **Objetivos de negócio e critérios de sucesso**

ON-1: Possibilitar ao prestador de cuidados a definição de alarmes (tarefas, medicação, etc.) para o(s) utente(s) entregues ao seu cuidado.

ON-2: Permitir ao prestador de cuidados conhecer a localização do(s) utente(s) que de si dependem.

ON-3: Permitir ao prestador de cuidados a consulta do histórico de atividades realizadas para o(s) utente(s) entregues ao seu cuidado.

ON-4: Aproximar a tecnologia da pessoa com idade avançada ou portadora de deficiência.

CS-1: Taxa de utilização semanal efetiva do serviço superior a 80% dos utentes.

CS-2: Conseguir a apreciação de Bom por parte de 75% dos utilizadores.

1.1.3 **Riscos do negócio**

RN-1: O serviço pode ser percecionado como pouco importante e não ter a adesão desejada por parte dos utentes. (Probabilidade = 0.2; Impacto = 9).

RN-2: Pela sua natureza, o produto pode levantar questões de privacidade que afugente potenciais utilizadores. (Probabilidade = 0.1; Impacto = 2).

1.2 VISÃO DA SOLUÇÃO

1.2.1 Visão

O iNavigator disponibiliza aos prestadores de cuidados uma ferramenta centralizada e divulgação para a criação, edição e eliminação de alarmes para o(s) utente(s) entregue(s) ao seu cuidado. Os alarmes podem ser criados manualmente ou de forma automática, com base em *context awareness*. A componente do sistema que acompanha o prestador de cuidados permite-lhe solicitar a localização e o relatório de atividades do utente; a componente do sistema que acompanha o utente permite ainda a recolha de notas em fotografia, vídeo e áudio e aceder a um menu SOS.

1.2.2 Principais funcionalidades

FU-1: Criação, edição ou eliminação manual ou automática de um alarme.

FU-2: Categorização do alarme e associação de um som e/ou a uma categoria de alarmes ou a um alarme em particular.

FU-3: Associação de um alarme a um utente e dispositivo.

FU-4: Consulta da localização do utente por parte do prestador de cuidados.

FU-5: Consulta do relatório de atividades do utente por parte do prestador de cuidados.

FU-6: Gravação de uma nota em fotografia, vídeo ou áudio por parte do utente.

FU-7: Aceitar um alarme ou pedir o seu adiamento por 1 minuto.

FU-8: Definição do comportamento do botão de pânico: chamadas telefónicas para números pré-definidos, envio de mensagens SMS pré-definidas ou personalizadas e consulta da localização atual.

1.2.3 Pressupostos e dependências

PR-1: O sistema exige um dispositivo móvel para cada utente e, opcionalmente, para o prestador de cuidados, com ligação à Internet para comunicação com o servidor.

PR-2: O sistema exige que os dispositivos móveis dos utentes e dos prestadores de cuidados tenham ligação à rede GSM para envio e receção de mensagens SMS.

PR-3: O sistema exige que a aplicação web do servidor seja acedida com um dispositivo capaz de mostrar páginas web modernas.

DE-1: É necessária ligação à Internet para configurar todos os dispositivos terminais.

DE-2: É necessária ligação à rede GSM para que os dispositivos dos utentes e prestadores de cuidados possam comunicar.

1.3 ÂMBITO E LIMITAÇÕES

1.3.1 Âmbito das versões inicial e subsequentes

Funcionalidade	Versão inicial	Versão 2
FU-1	Totalmente implementada	
FU-2	Totalmente implementada	
FU-3	Totalmente implementada	
FU-4	Totalmente implementada	
FU-5	Totalmente implementada	
FU-6	Pode ser remetida para versões futuras	Totalmente implementada
FU-7	Totalmente implementada	
FU-8	Totalmente implementada	

1.3.2 Limitações e exclusões

LI-1: O *iNavigator* não terá certificação médica ou similar.

LI-2: O *iNavigator* só será usado para gerir o sistema de alertas e controlar a sua aceitação.

1.4 CONTEXTO DE NEGÓCIO

1.4.1 Perfis dos stakeholders

Stakeholder	Valores principais	Atitudes	Interesse principal	Constrangimentos
Prestador de cuidados	Possibilidade de obtenção da localização do utente. Possibilidade de obtenção de relatório de atividades do utente.	Curiosidade. Entusiasmado com as possibilidades do produto.	Simplicidade. Máxima adesão por parte do utente. Fiabilidade e robustez do software	Preocupações com a privacidade.
Utente	Possibilidade de tirar notas multimédia. Botão SOS de comportamento personalizável. Alarmes de atividades ou eventos à hora certa, com som e imagem.	Curiosidade. Entusiasmado com as possibilidades do produto.	Simplicidade. Fiabilidade e robustez do software. Ganhos na gestão da rotina diária.	O sistema pode ser percecionado como demasiado intrusivo.

1.4.2 Prioridades do projeto

Dimensão	Driver	Constrangimentos	Grau de liberdade
Agenda			O lançamento da primeira versão está planeada para a data da defesa da tese de Mestrado em que o projeto está inserido.
Funcionalidades		Todas as funcionalidades previstas para a versão 1.0 têm que estar completamente operacionais.	
Qualidade		100% de aprovação nos testes de aceitação e de segurança.	
Equipa	Um programador. Um orientador que desempenha o papel de gestor de projeto e cliente.		
Custo			Não está prevista derrapagem.

ANEXO II - DIAGRAMAS DE CASOS DE USO

1.1 APLICAÇÃO WEB – UTILIZADOR NÃO AUTENTICADO

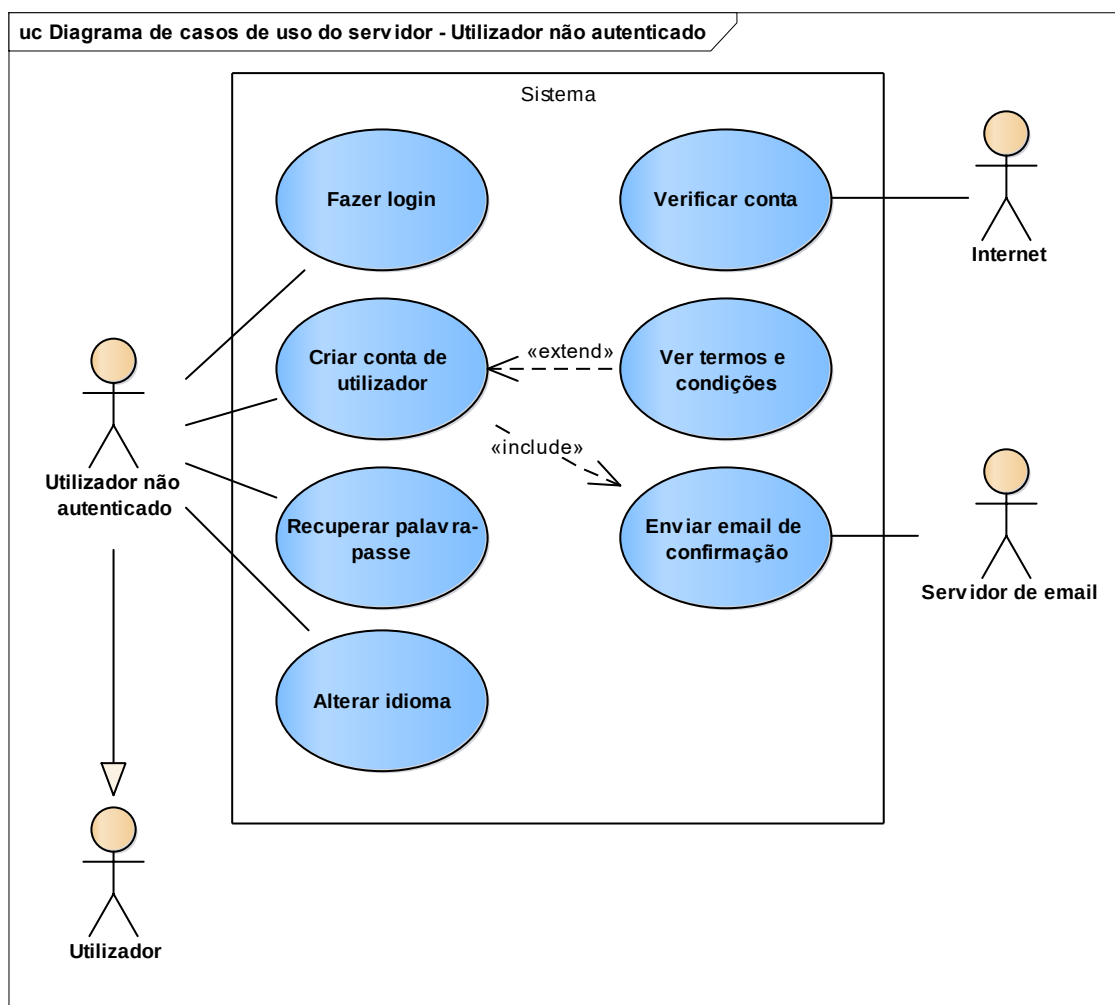


Figura 59 - Diagrama de casos de uso da aplicação web.

1.2 APLICAÇÃO WEB – UTILIZADOR AUTENTICADO

1.2.1 *Dashboard, logout* e edição de perfil

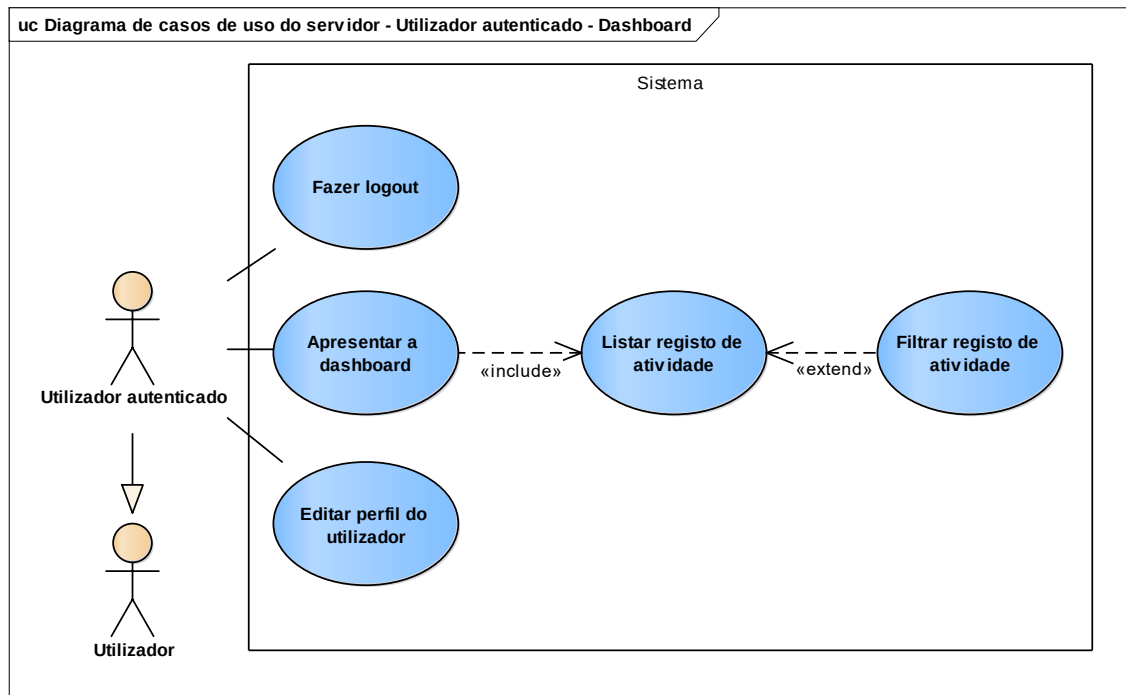


Figura 60 – Casos de uso respeitantes à *dashboard, logout* e perfil.

1.2.2 Alarmes e categorias de alarmes

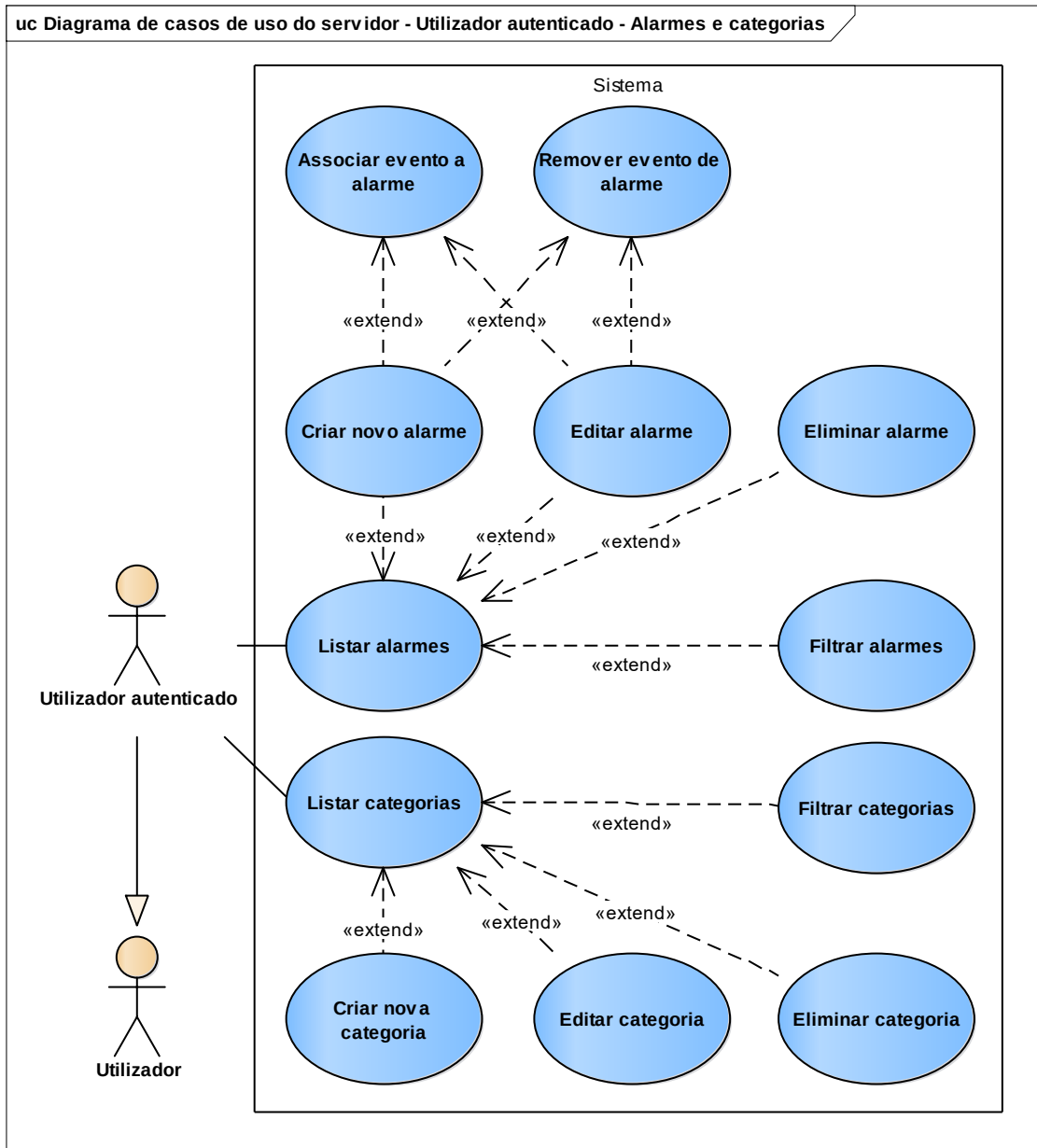


Figura 61 - Casos de uso respeitantes a alarmes e categorias de alarmes

1.2.3 Dispositivos e utilizadores

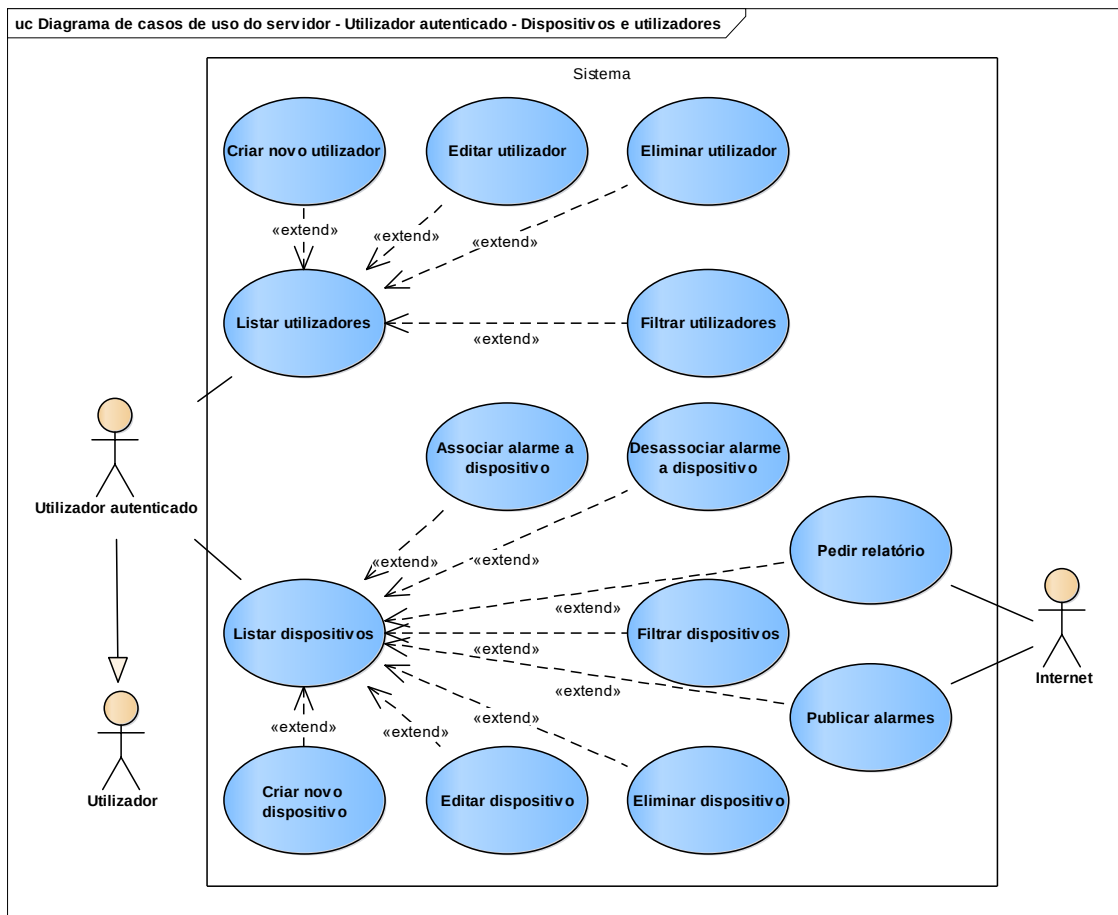


Figura 62 - Casos de uso respeitantes a dispositivos e utilizadores.

1.3 APLICAÇÃO MÓVEL ANDROID – PERFIL UTENTE.

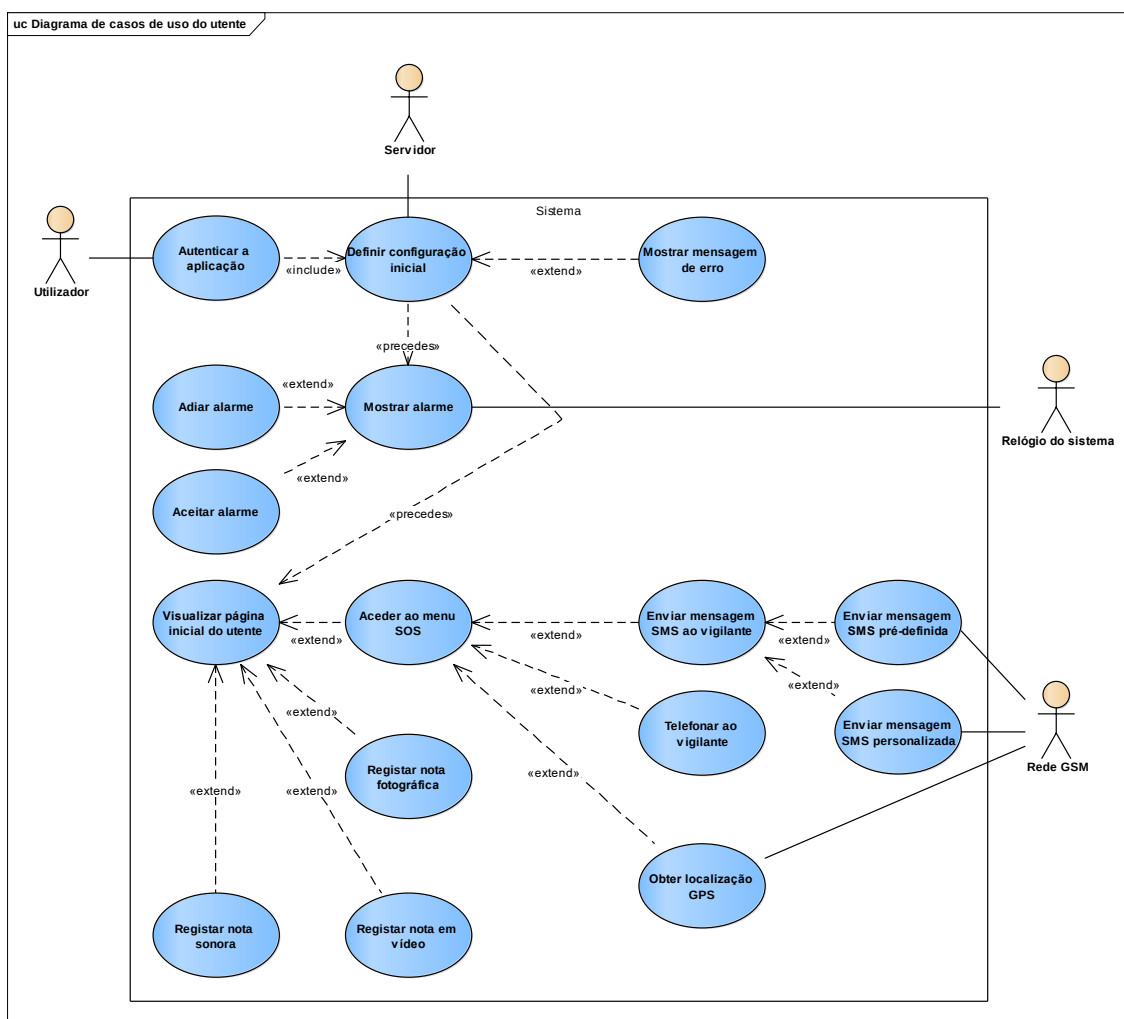


Figura 63 – Casos de uso da aplicação *Android* – perfil utente.

1.4 APLICAÇÃO MÓVEL *ANDROID* – PERFIL VIGILANTE.

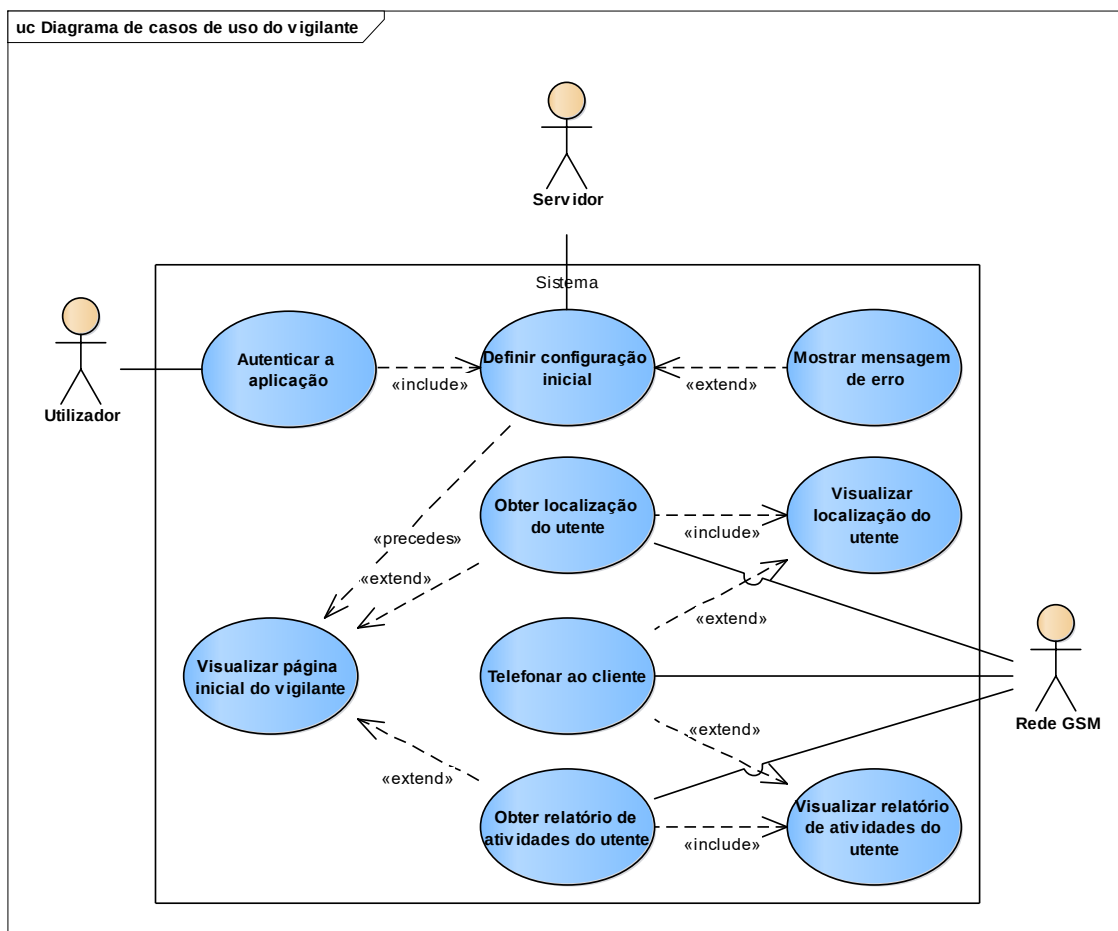


Figura 64 - Casos de uso da aplicação *Android* – perfil vigilante

ANEXO III - CASOS DE USO

1.1 TABELA DE CASOS DE USO DA APLICAÇÃO WEB

A tabela seguinte lista os casos de uso da aplicação web, divididos em função dos atores que neles intervêm.

Ator primário	Caso de uso
Utilizador não autenticado	1. Fazer <i>login</i>. 2. Criar conta de utilizador. 3. Recuperar palavra passe. 4. Alterar idioma.
Utilizador autenticado	5. Apresentar a <i>dashboard</i>. 6. Listar registo de atividade. 7. Listar utilizadores. 8. Criar novo utilizador. 9. Editar utilizador. 10. Eliminar utilizador. 11. Listar dispositivos. 12. Criar novo dispositivo. 13. Selecionar dispositivo. 14. Editar dispositivo. 15. Eliminar dispositivo. 16. Associar utilizador a dispositivo. 17. Associar alarme a dispositivo. 18. Desassociar alarme a dispositivo. 19. Listar alarmes. 20. Selecionar alarme. 21. Editar alarme. 22. Eliminar alarme. 23. Criar novo alarme. 24. Associar evento a alarme. 25. Remover evento de alarme. 26. Listar categorias. 27. Criar nova categoria. 28. Editar categoria. 29. Eliminar categoria. 30. Publicar alarmes. 31. Pedir relatório. 32. Fazer <i>logout</i>.

1.2 TABELA DE CASOS DE USO DA APLICAÇÃO CLIENTE

A tabela seguinte lista os casos de uso da aplicação instalada nos dispositivos terminais, divididos em função do perfil da aplicação e em função dos atores que neles intervêm.

Perfil	Ator primário	Caso de uso		
Utente Vigilante	Servidor	1. a) Definir configuração inicial. b) Definir configuração.		
	Servidor Utilizador	2. Autenticar a aplicação. 3. Mostrar mensagem de erro.		
Utente	Relógio do sistema	4. Mostrar alarme.		
	Rede GSM	5. Enviar estatísticas.		
	Utilizador	6. Aceitar alarme. 7. Adiar alarme. 8. Aceder ao menu SOS. 9. Enviar mensagem SMS ao vigilante. 10. Obter localização GPS. 11. Enviar coordenadas GPS ao vigilante. 12. Visualizar página inicial do utente. 13. Aceder ao menu notas. 14. Registar nota fotográfica. 15. Registar nota em vídeo. 16. Registar nota sonora.		
		Utilizador Rede GSM	17. Enviar mensagem SMS pré-definida. 18. Enviar mensagem SMS personalizada. 19. Telefonar ao vigilante.	
		Vigilante	Utilizador	20. Mostrar página inicial. 21. Obter localização do cliente. 22. Obter relatório de atividades.
			Utilizador Rede GSM	23. Telefonar ao utente. 24. Visualizar localização do utente. 25. Visualizar relatório de atividades do utente

1.3 CASOS DE USO DA APLICAÇÃO WEB DO SERVIDOR

Dado o elevado número de casos de uso e a sua simplicidade, são apresentados apenas os casos de uso mais relevante do servidor. Os casos de uso da aplicação para dispositivos móveis são apresentados integralmente.

1.3.1 Fazer login

ID do caso de uso:	1
Nome do caso de uso:	Fazer login
Atores:	Utilizador não autenticado
Descrição:	Sempre que um utilizador acede à aplicação web tem que fazer login para ser reconhecido pelo servidor.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Fazer login. 1. O utilizador não autenticado faz login introduzindo o nome de utilizador e a palavra passe e pressionando o botão Login. 2. Se as credenciais forem reconhecidas o utilizador passa a estar autenticado. 3. O sistema apresenta a página inicial da aplicação web.
Fluxos alternativos:	1. Se as credenciais não forem reconhecidas é mostrada uma mensagem de erro.

1.3.2 Criar conta de utilizador

ID do caso de uso:	2
Nome do caso de uso:	Criar conta de utilizador
Atores:	Utilizador não autenticado
Descrição:	Para aceder à aplicação web é necessário criar uma conta.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Criar conta. 1. O utilizador não autenticado cria uma conta introduzindo o nome de utilizador, o endereço de correio eletrónico e a palavra passe e pressionando o botão Criar conta. 2. É enviado um email para o endereço de correio eletrónico especificado. 3. O sistema apresenta uma mensagem dando conta que é necessário fazer a validação do endereço de correio eletrónico indicado.
Fluxos alternativos:	1. Não existem.

1.3.3 Apresentar a *dashboard*

ID do caso de uso:	5
Nome do caso de uso:	Apresentar a <i>dashboard</i>
Atores:	Utilizador autenticado Sistema
Descrição:	O sistema mostra a <i>dashboard</i> ao utilizador.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
	1.0 Apresentar a <i>dashboard</i>.
Fluxo normal:	1. O sistema mostra a <i>dashboard</i> ao utilizador aquando da sua autenticação no sistema.
Fluxos alternativos:	1. Nenhum.
Prioridade:	Alta.
Frequência de uso:	Aproximadamente uma utilização diária por utilizador.

1.3.4 Publicar alarmes

ID do caso de uso:	30
Nome do caso de uso:	Publicar alarmes
Atores:	Utilizador autenticado.
Descrição:	O utilizador publica os alarmes no dispositivo.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
	1.0 Publicar alarmes.
Fluxo normal:	1. O utilizador solicita ao sistema a publicação de alarmes do dispositivo. 2. O sistema publica os alarmes. 3. O sistema aguarda a confirmação da publicação por parte do dispositivo. 4. O sistema recebe a confirmação da publicação por parte do dispositivo. 5. O sistema mostra a confirmação da publicação dos alarmes.
Fluxos alternativos:	1. Se o dispositivo não tiver qualquer alarme associado é mostrada uma mensagem de erro. 4. O sistema não recebe a confirmação da publicação dos alarmes. 5. O sistema não mostra a confirmação da publicação dos alarmes.
Prioridade:	Alta.
Frequência de uso:	Aproximadamente uma utilização diária por utilizador.

1.3.5 Pedir relatório

ID do caso de uso:	31
Nome do caso de uso:	Pedir relatório
Atores:	Utilizador autenticado.
Descrição:	O utilizador pede o relatório ao dispositivo.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Pedir relatório. 1. O utilizador solicita ao sistema o pedido de relatório 2. O sistema pede o relatório. 3. O sistema aguarda a confirmação do relatório por parte do dispositivo. 4. O sistema recebe a confirmação do relatório por parte do dispositivo. 5. O sistema mostra a confirmação do relatório.
Fluxos alternativos:	4. O sistema não recebe a confirmação do relatório. 5. O sistema não mostra a confirmação do relatório.
Prioridade:	Alta.
Frequência de uso:	Aproximadamente uma utilização diária por utilizador.

1.4 CASOS DE USO DA APLICAÇÃO PARA DISPOSITIVOS MÓVEIS

1.4.1 Definir configuração inicial

ID do caso de uso:	1a
Nome do caso de uso:	Definir configuração inicial.
Perfil da aplicação:	Sem perfil.
Atores:	Servidor.
Descrição:	Na primeira execução da aplicação o sistema interroga o servidor acerca da configuração que deverá adotar.
Pré-condições:	1. O sistema tem que estar ligado à Internet.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Definir configuração inicial. 1. Na primeira execução o sistema solicita ao servidor a configuração inicial. 2. O servidor envia ao sistema a configuração inicial. 3. O sistema adota o perfil cliente ou vigilante em função da resposta do servidor.
Fluxos alternativos:	1.1 Definir configuração inicial. 1. Na primeira execução o sistema solicita ao servidor a configuração inicial. 2. O servidor envia ao sistema a configuração inicial. 3. O sistema não obtém uma resposta do servidor. 4. O sistema executa o caso de uso mostrar mensagem de erro.

1.4.2 Definir configuração

ID do caso de uso:	1b
Nome do caso de uso:	Definir configuração.
Perfil da aplicação:	Utente.
Atores:	Servidor.
Descrição:	O servidor envia ao sistema uma nova configuração de alarmes que substitui todos os anteriores.
Pré-condições:	1. O sistema tem que estar ligado à Internet.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Definir configuração. 1. O servidor envia ao sistema uma nova configuração de alarmes. 2. O sistema executa o fluxo alternativo 1.1 do caso de uso enviar estatísticas. 3. O sistema substitui a sua configuração de alarmes pela recebida pelo servidor.
Fluxos alternativos:	1. Não existem.
Pressupostos:	1. Nenhum.
Notas e questões:	1. Nenhumas.

1.4.3 Autenticar aplicação

ID do caso de uso: 2	
Nome do caso de uso:	Autenticar aplicação.
Perfil da aplicação:	Sem perfil.
Atores:	Servidor.
Descrição:	Em resposta a um pedido, o servidor envia a configuração ao sistema.
Pré-condições:	1. O sistema tem que estar ligado à Internet.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Definir configuração inicial. 1. O servidor envia ao sistema a configuração inicial. 2. O sistema adota o perfil cliente ou vigilante em função da resposta do servidor.
Fluxos alternativos:	1.1 Definir configuração inicial. 1. O servidor não envia ao sistema a configuração inicial ou envia uma mensagem dando conta que a configuração é inválida. 2. O sistema não obtém uma resposta do servidor. 3. O sistema executa o caso de uso mostrar mensagem de erro.

1.4.4 Mostrar mensagem de erro

ID do caso de uso: 3	
Nome do caso de uso:	Definir configuração inicial.
Perfil da aplicação:	Todos os perfis.
Atores:	Cliente, utilizador.
Descrição:	O sistema mostra uma mensagem de erro que impede a continuação da execução.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.1 Mostrar mensagem de erro. 1. O sistema mostra uma mensagem de erro ao utilizador. 2. O utilizador clica no botão de reconhecimento da mensagem de erro. 3. O sistema termina a sua execução.
Fluxos alternativos:	1. Não existem.

1.4.5 Mostrar alarme

ID do caso de uso:	4
Nome do caso de uso:	Mostrar alarme.
Perfil da aplicação:	Utente.
Atores:	Relógio do sistema.
Descrição:	O relógio do sistema torna visível um alarme.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Mostrar alarme. 1. O relógio do sistema torna visível um alarme, com a imagem e som associados.
Fluxos alternativos:	1. Não existem.

1.4.6 Enviar estatísticas

ID do caso de uso:	5
Nome do caso de uso:	Enviar estatísticas.
Perfil da aplicação:	Utente.
Atores:	Relógio do sistema. Rede GSM.
Descrição:	O sistema gera estatísticas de utilização e envia-as ao servidor ou à rede GSM.
Pré-condições:	1. O sistema tem que estar ligado à rede GSM (Fluxo normal) 2. O sistema tem que estar ligado à Internet (Fluxos alternativos).
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Enviar estatísticas. 1. A rede GSM pede estatísticas de utilização ao sistema. 2. O sistema gera estatísticas de utilização. 3. O sistema envia as estatísticas à rede GSM.
Fluxos alternativos:	1.1 Enviar estatísticas (pedido do servidor). 1. O servidor pede estatísticas de utilização ao sistema. 2. O sistema gera estatísticas de utilização. 3. O sistema envia as estatísticas ao servidor. 1.2 Enviar estatísticas (inclusão do CU Definir configuração). 1. O servidor envia ao sistema uma nova configuração de alarmes. 2. O sistema gera estatísticas de utilização. 3. O sistema envia as estatísticas ao servidor.

1.4.7 Aceitar alarme

ID do caso de uso: 6	
Nome do caso de uso:	Aceitar alarme.
Perfil da aplicação:	Utente.
Atores:	Utente.
Descrição:	O utente aceita o alarme mostrado pelo sistema.
Pré-condições:	1. O sistema executou o caso de uso Mostrar alarme.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Aceitar alarme. 1. O utilizador aceita o alarme. 2. O sistema regista a aceitação. 3. O sistema esconde o alarme.
Fluxos alternativos:	1. Não existem.

1.4.8 Adiar alarme

ID do caso de uso: 7	
Nome do caso de uso:	Adiar alarme.
Perfil da aplicação:	Utente.
Atores:	Utente.
Descrição:	O utente adia o alarme mostrado pelo sistema.
Pré-condições:	1. O sistema executou o caso de uso Mostrar alarme.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Adiar alarme. 1. O utilizador adia o alarme. 2. O sistema regista o pedido de adiamento. 3. O sistema define novo alarme para daí a 1 minuto. 4. O sistema esconde o alarme.
Fluxos alternativos:	1.1 Adiar alarme. 1. O utilizador adia o alarme. 2. O sistema regista o pedido de adiamento. 3. O sistema não define novo alarme por já ter sido adiado três vezes. 4. O sistema esconde o alarme.

1.4.9 Aceder ao menu SOS

ID do caso de uso:	8
Nome do caso de uso:	Aceder ao menu SOS.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	Na página inicial ou durante um alarme o utente pode aceder ao menu SOS.
Pré-condições:	1. O sistema executou o caso de uso Mostrar alarme. 2. Alternativamente o sistema executou o caso de uso Visualizar página inicial do cliente.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Aceder ao menu SOS. 1. O utilizador prime o botão que dá acesso ao menu SOS. 2. O sistema mostra as opções do menu SOS.
Fluxos alternativos:	1. Não existem.

1.4.10 Enviar mensagem SMS ao vigilante

ID do caso de uso:	9
Nome do caso de uso:	Enviar mensagem SMS ao vigilante.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	O utilizador envia uma mensagem SMS ao vigilante.
Pré-condições:	1. O sistema executou o caso de uso Aceder ao menu SOS.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Enviar mensagem SMS a número pré-definido. 1. O utilizador seleciona a opção que permite enviar uma mensagem SMS. 2. O sistema mostra um menu com várias mensagens pré-definidas ao mesmo tempo que possibilita criar uma mensagem personalizada.
Fluxos alternativos:	1. Não existem.

1.4.11 Obter localização GPS

ID do caso de uso:	10
Nome do caso de uso:	Obter localização GPS.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	O utilizador solicita ao sistema as coordenadas GPS onde se encontra e envia-as por mensagem SMS a um número pré-definido.
Pré-condições:	1. O sistema executou o caso de um Aceder ao menu SOS.
Pós-condições:	1. O sistema executa o caso de uso Enviar coordenadas GPS para número pré-definido.
Fluxo normal:	1.0 Obter localização GPS. 1. O utilizador solicita ao sistema as coordenadas GPS onde se encontra. 2. O sistema mostra um mapa e assinala a posição atual.
Fluxos alternativos:	1. Não existem.

1.4.12 Enviar coordenadas GPS ao vigilante

ID do caso de uso:	11
Nome do caso de uso:	Enviar coordenadas GPS ao vigilante.
Perfil da aplicação:	Utente.
Atores:	Utente. Rede GSM.
Descrição:	O cliente do utente envia coordenadas GPS ao cliente do vigilante.
Pré-condições:	1. O sistema executou o caso de uso Obter localização GPS.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Enviar coordenadas GPS para número pré-definido. 1. O vigilante solicita o envio das coordenadas GPS do utente. 2. O sistema envia as coordenadas ao vigilante.
Fluxos alternativos:	1. Não existem.

1.4.13 Visualizar página inicial do utente

ID do caso de uso:	12
Nome do caso de uso:	Visualizar página inicial do cliente.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	O utilizador inicia o arranque manual do sistema.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Visualizar página inicial do cliente. 1. O utilizador inicia o arranque manual do sistema. 2. O sistema mostra a página inicial do perfil cliente.
Fluxos alternativos:	1. Não existem.

1.4.14 Aceder ao menu notas

ID do caso de uso:	13
Nome do caso de uso:	Aceder ao menu notas.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	Na página inicial o utilizador pode aceder ao menu notas.
Pré-condições:	1. O sistema executou o caso de uso Visualizar página inicial do cliente.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Aceder ao menu notas. 1. O utilizador prime o botão que dá acesso ao menu notas. 2. O sistema mostra as opções do menu SOS.
Fluxos alternativos:	1. Não existem.

1.4.15 Registar nota fotográfica

ID do caso de uso:	14
Nome do caso de uso:	Registar nota fotográfica.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	O utilizador pretende capturar uma fotografia.
Pré-condições:	1. O sistema executou o caso de uso Aceder ao menu notas.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Registar nota fotográfica. 1. O utilizador seleciona a opção nota fotográfica. 2. O sistema executa a aplicação do sistema operativo para a captura de fotografias.
Fluxos alternativos:	1. Não existem.

1.4.16 Registar nota em vídeo

ID do caso de uso:	15
Nome do caso de uso:	Registar nota em vídeo.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	O utilizador pretende capturar um vídeo.
Pré-condições:	1. O sistema executou o caso de uso Aceder ao menu notas.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Registar nota em vídeo. 1. O utilizador seleciona a opção nota em vídeo. 2. O sistema executa a aplicação do sistema operativo para a captura de vídeos.
Fluxos alternativos:	1. Não existem.

1.4.17 Registar nota sonora

ID do caso de uso:	16
Nome do caso de uso:	Registar nota sonora.
Perfil da aplicação:	Utente.
Atores:	Utilizador.
Descrição:	O utilizador pretende capturar áudio.
Pré-condições:	1. O sistema executou o caso de uso Aceder ao menu notas.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Registar nota sonora. 1. O utilizador seleciona a opção nota sonora. 2. O sistema executa a aplicação do sistema operativo para a captura de áudio.
Fluxos alternativos:	1. Não existem.

1.4.18 Enviar mensagem SMS pré-definida

ID do caso de uso:	17
Nome do caso de uso:	Enviar mensagem SMS pré-definida.
Perfil da aplicação:	Utente.
Atores:	Utilizador. Rede GSM.
Descrição:	O utilizador deseja enviar uma mensagem SMS pré-definida a um número igualmente pré-definido.
Pré-condições:	1. O sistema executou o caso de uso Enviar mensagem SMS a número pré-definido.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Enviar mensagem SMS pré-definida. 1. O utilizador seleciona uma mensagem pré-definida. 2. O sistema envia a mensagem SMS à rede GSM.
Fluxos alternativos:	1. Não existem.

1.4.19 Enviar mensagem SMS personalizada

ID do caso de uso:	18
Nome do caso de uso:	Enviar mensagem SMS personalizada ao vigilante.
Perfil da aplicação:	Utente.
Atores:	Utilizador. Rede GSM.
Descrição:	O utilizador deseja enviar uma mensagem SMS personalizada ao vigilante.
Pré-condições:	1. O sistema executou o caso de uso Enviar mensagem SMS ao vigilante.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Enviar mensagem SMS personalizada. 1. O utilizador compõe o texto da mensagem. 2. O sistema envia a mensagem SMS à rede GSM.
Fluxos alternativos:	1. Não existem.

1.4.20 **Telefonar ao vigilante**

ID do caso de uso:	19
Nome do caso de uso:	Telefonar a número pré-definido.
Perfil da aplicação:	Utente.
Atores:	Utilizador. Rede GSM.
Descrição:	O utilizador telefona ao vigilante.
Pré-condições:	1. O sistema executou o caso de uso Aceder ao menu SOS.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Telefonar a número pré-definido. 1. O utilizador seleciona o número do vigilante. 2. O sistema solicita à rede GSM que estabeleça a ligação com o cliente. 3. A rede GSM estabelece a ligação com o número pré-definido.
Fluxos alternativos:	1.1 Telefonar a número pré-definido. 3. A rede GSM não estabelece a ligação com o cliente. 4. O sistema mostra uma mensagem de erro.

1.4.21 **Visualizar página inicial do vigilante**

ID do caso de uso:	20
Nome do caso de uso:	Visualizar página inicial do vigilante.
Perfil da aplicação:	Vigilante.
Atores:	Utilizador.
Descrição:	O utilizador inicia o arranque manual do sistema.
Pré-condições:	1. Não existem.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Mostrar página inicial. 1. O utilizador inicia o arranque manual do sistema. 2. O sistema mostra a página inicial do perfil vigilante.
Fluxos alternativos:	1. Não existem.

1.4.22 Obter localização do cliente

ID do caso de uso:	21
Nome do caso de uso:	Obter localização do cliente.
Perfil da aplicação:	Vigilante.
Atores:	Utilizador. Rede GSM.
Descrição:	O utilizador pretende saber a localização do cliente.
Pré-condições:	1. O sistema executou o caso de uso Visualizar página inicial do vigilante.
Pós-condições:	1. O sistema executa o caso de uso Visualizar localização do cliente.
Fluxo normal:	1.0 Obter localização do cliente. 1. O utilizador seleciona a opção que permite obter a localização do cliente. 2. O sistema encaminha o pedido para a rede GSM. 3. O sistema aguarda a resposta da rede GSM.
Fluxos alternativos:	1. Não existem.

1.4.23 Obter relatório de atividades do cliente

ID do caso de uso:	22
Nome do caso de uso:	Obter relatório de atividades.
Perfil da aplicação:	Vigilante.
Atores:	Utilizador. Rede GSM
Descrição:	O utilizador pretende obter o relatório de atividades do cliente.
Pré-condições:	1. O sistema executou o caso de uso Visualizar página inicial do vigilante.
Pós-condições:	1. O sistema executa o caso de uso Visualizar relatório do cliente.
Fluxo normal:	1.0 Obter relatório de atividades. 1. O utilizador seleciona a opção que permite obter a localização do cliente. 2. O sistema encaminha o pedido para a rede GSM. 3. O sistema aguarda a resposta da rede GSM.
Fluxos alternativos:	1. Não existem.

1.4.24 Telefonar ao utente

ID do caso de uso:	23
Nome do caso de uso:	Telefonar ao cliente.
Perfil da aplicação:	Vigilante.
Atores:	Utilizador. Rede GSM.
Descrição:	Na sequência da visualização da posição GPS ou do relatório de atividades, o utilizador pretende ligar ao utente.
Pré-condições:	1. O sistema executou o caso de uso Visualizar localização do utente. 2. Alternativamente o sistema executou o caso de uso Visualizar o relatório do utente.
Pós-condições:	1. Não existem.
Fluxo normal:	1.2 Telefonar ao utente. 1. O utilizador seleciona a opção para ligar ao utente. 2. O sistema solicita à rede GSM que estabeleça a ligação com o utente. 3. A rede GSM estabelece a ligação com o utente.
Fluxos alternativos:	1.3 Telefonar ao utente. 5. A rede GSM não estabelece a ligação com o utente. 6. O sistema mostra uma mensagem de erro.

1.4.25 Visualizar localização do utente

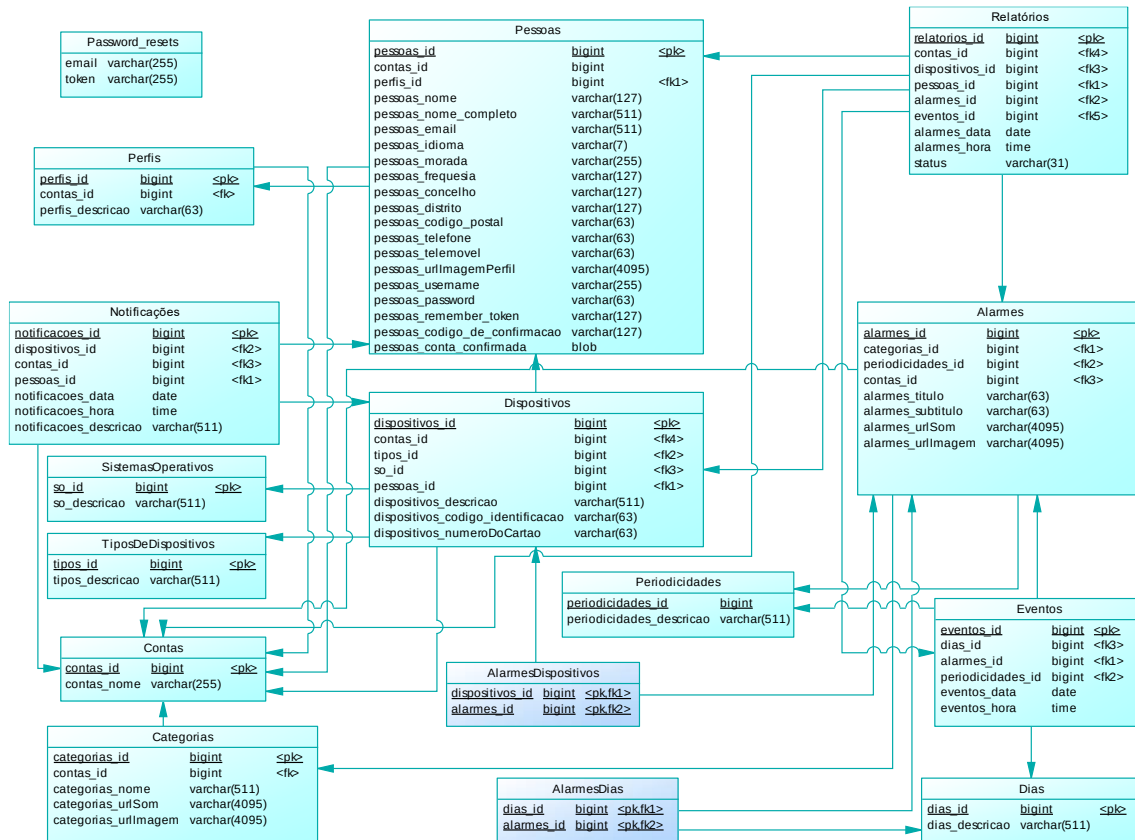
ID do caso de uso:	24
Nome do caso de uso:	Visualizar localização do utente.
Perfil da aplicação:	Vigilante.
Atores:	Utilizador. Rede GSM. Relógio do sistema.
Descrição:	A resposta chega ao sistema e o sistema mostra um mapa e assinala a localização do cliente. Se a resposta não tiver chegado passado 1 minuto o sistema mostra uma mensagem de erro.
Pré-condições:	1. O sistema executou o caso de uso Obter localização do cliente.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Visualizar localização do cliente. 1. A rede GSM envia as coordenadas GPS do cliente ao sistema. 2. O sistema mostra um mapa e assinala a localização.
Fluxos alternativos:	1.1 Visualizar localização do cliente. 1. O relógio do sistema alerta o sistema para a passagem do intervalo de tempo definido (1 minuto). 2. A rede GSM não enviou resposta ao sistema no intervalo de tempo definido. 3. O sistema mostra uma mensagem de erro.

1.4.26 Visualizar relatório de atividades do utente

ID do caso de uso:	25
Nome do caso de uso:	Visualizar relatório de atividades do utente.
Perfil da aplicação:	Vigilante.
Atores:	Utilizador. Rede GSM. Relógio do sistema.
Descrição:	A resposta chega ao sistema e o sistema mostra o relatório de atividades do cliente. Se a resposta não tiver chegado passado 1 minuto o sistema mostra uma mensagem de erro.
Pré-condições:	1. O sistema executou o caso de uso Obter relatório de atividades do cliente.
Pós-condições:	1. Não existem.
Fluxo normal:	1.0 Visualizar relatório do cliente. 1. A rede GSM envia o relatório de atividades do cliente ao sistema. 2. O sistema mostra o relatório de atividades do cliente.
Fluxos alternativos:	1.1 Visualizar relatório do cliente. 1. O relógio do sistema alerta o sistema para a passagem do intervalo de tempo definido (1 minuto). 2. A rede GSM não enviou resposta ao sistema no intervalo de tempo definido. 3. O sistema mostra uma mensagem de erro.

ANEXO IV – MODELO DA BASE DE DADOS

1.1 MODELO FÍSICO



ANEXO V – REGRAS DE NEGÓCIO

Identificação	Definição da regra	Tipo de regra	Estática ou dinâmica	Fonte
BR-1	As chamadas telefónicas têm que ser pagas pelos utentes.	Restrição	Dinâmica	Utente
BR-2	As mensagens SMS têm que ser pagas pelos utentes ou pelos vigilantes.	Restrição	Dinâmica	Vigilantes Utente
BR-3	O vigilante pode obter a localização do utente via mensagem SMS.	Facto	Estático	Vigilante
BR-4	O vigilante pode obter um relatório das atividades realizadas pelo utente durante o dia em curso.	Facto	Estático	Vigilante
BR-5	O utente pode aceitar ou adiar um alarme.	Facto	Estático	Utente
BR-6	Um alarme só pode ser adiado 3 vezes.	Restrição	Dinâmica	Utente
BR-7	O utente pode telefonar para números pré-definidos.	Facto	Estático	Utente
BR-8	O utente pode enviar mensagem SMS para números pré-definidos usando mensagens pré-definidas ou personalizadas.	Facto	Estático	Utente
BR-9	O utente pode recolher uma nota multimédia.	Facto	Estático	Utente
BR-10	Os dispositivos têm que ter acesso pontual ou contínuo à Internet.	Restrição	Estático	Vigilante Utente
BR-11	O acesso à aplicação web do servidor está condicionado pela criação de uma conta e posterior autenticação.	Restrição	Dinâmica	Vigilante
BR-12	A configuração dos dispositivos é feita na aplicação web do servidor.	Facto	Estático	Vigilante
BR-13	Os alarmes podem ser agrupados em categorias.	Facto	Estático	Vigilante
BR-14	Cada categoria tem o seu próprio som e/ou imagem de fundo.	Facto	Estático	Vigilante
BR-15	É possível definir um som e/ou imagem de fundo para um alarme particular	Facto	Estático	Vigilante
BR-16	O vigilante pode visualizar estatísticas de utilização na aplicação web do servidor.	Facto	Estático	Vigilante

ANEXO VI – MAQUETES DE BAIXA FIDELIDADE

Neste anexo incluem-se algumas maquetes de baixa fidelidade do sistema. As maquetas apresentadas não pretendem ser exaustivas mas apenas testemunhar o trabalho de conceção subjacente à afinação da interface do utilizador.

The image shows a low-fidelity wireframe of a web application's login screen, titled "iNavigator v0.4". The browser's address bar displays "http://www.inavigator.pt". In the top right corner, there is a language selector set to "Português (Portugal)". The main heading "iNavigator" is centered. Below it is a login form containing a user icon, a text input for "Email ou nome de utilizador", a text input for "Palavra-passe", a link "Esqueci a minha palavra-passe", a "Login" button, and a checkbox for "Manter sessão iniciada". At the bottom of the form is a link "Criar uma conta".

Figura 65 - Ecrã de autenticação do servidor.

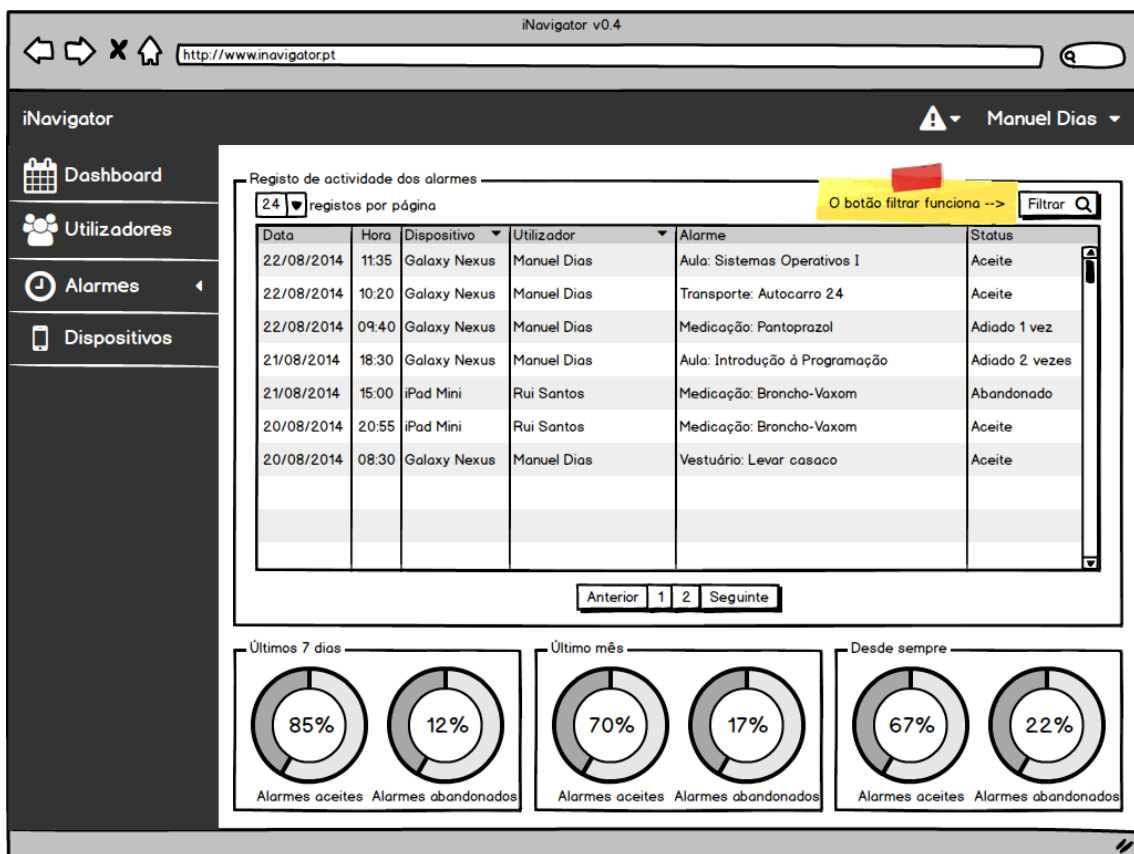


Figura 66 – Dashboard do servidor.

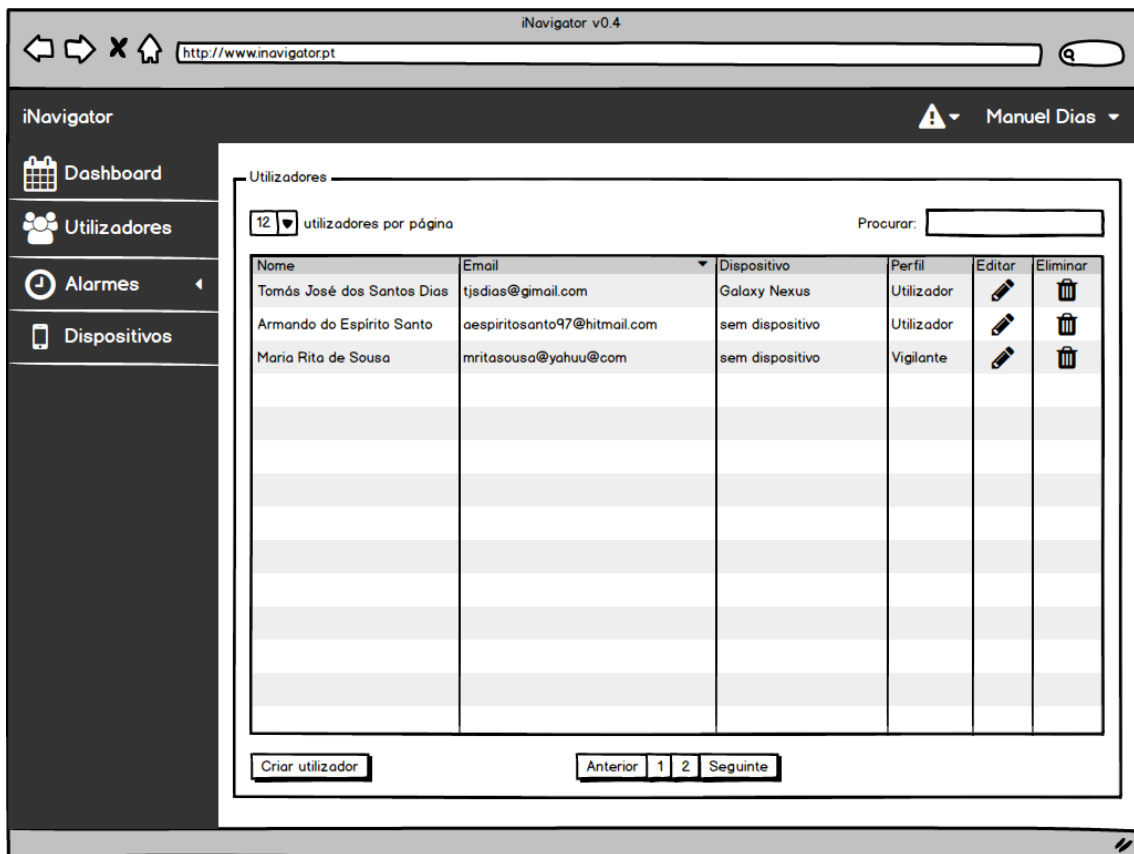


Figura 67 - Lista de utilizadores no servidor.

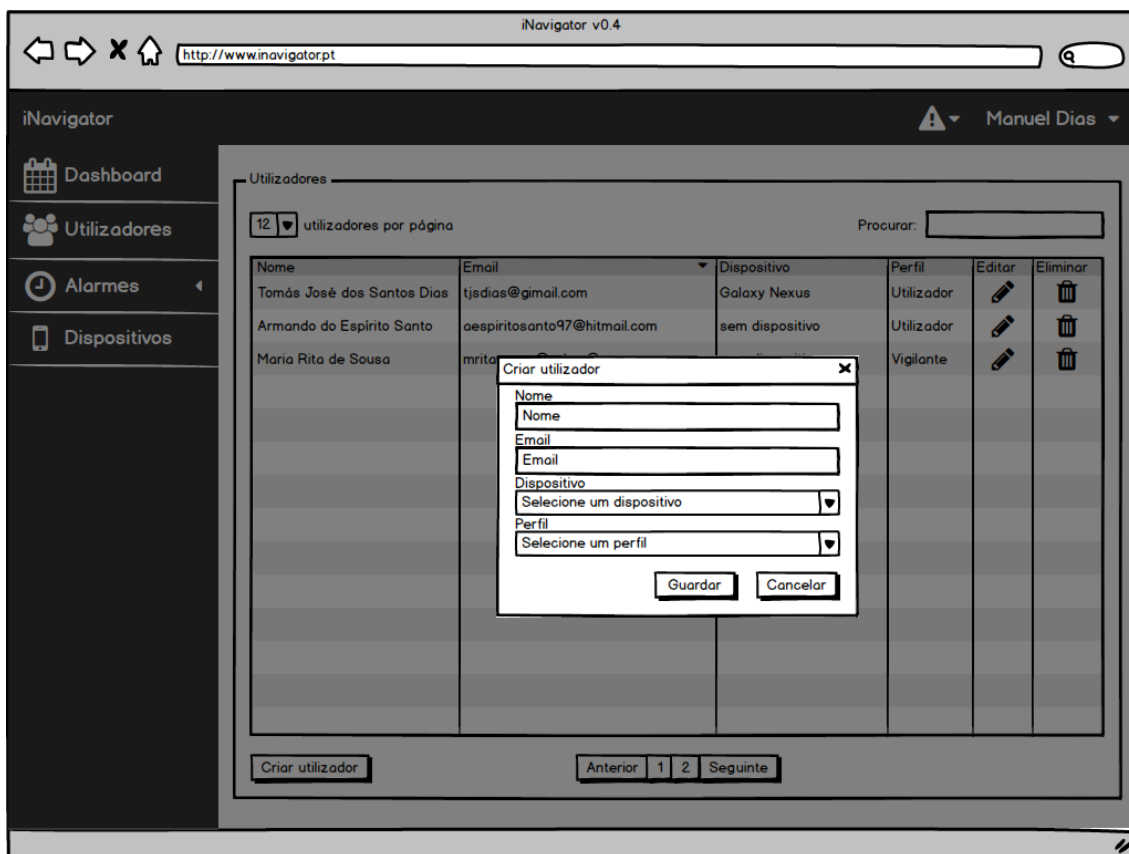


Figura 68 – Criação de um utilizador no servidor.

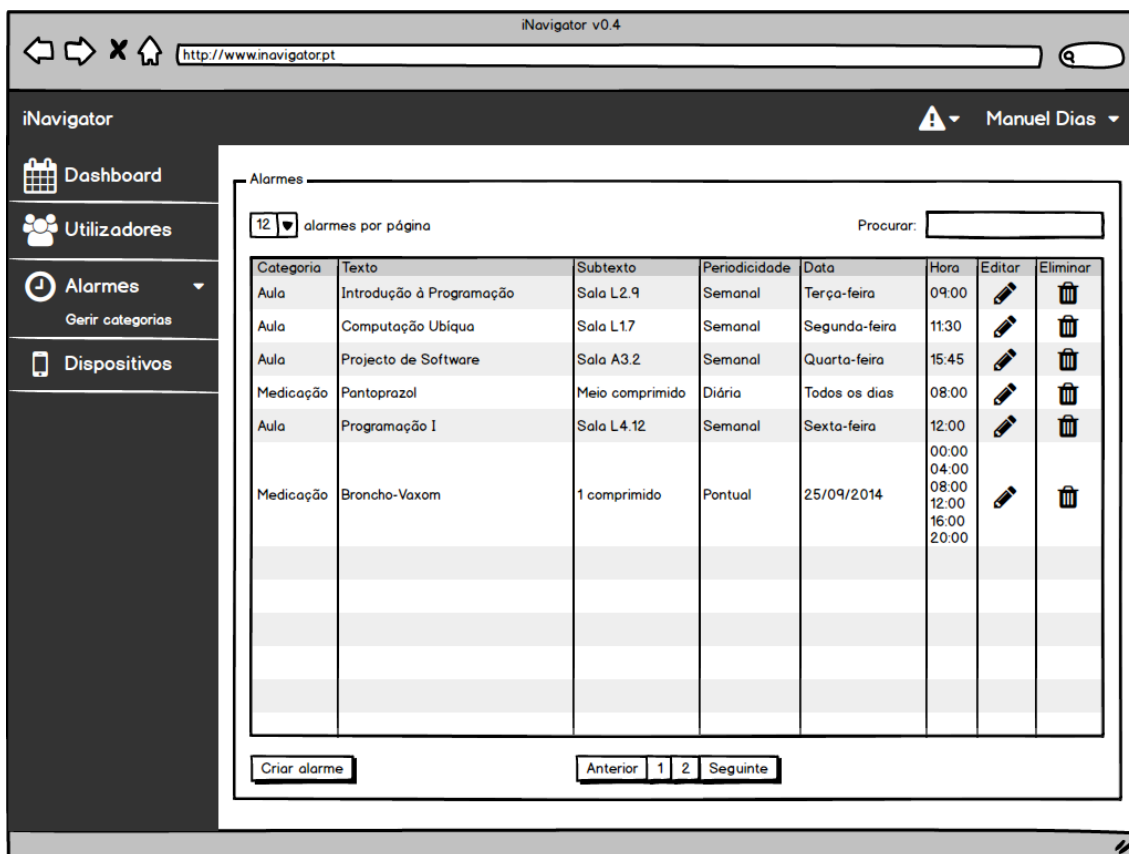


Figura 69 - Lista de alarmes no servidor.



Figura 70 – Interação com um alarme no cliente.



Figura 71 – Ecrã do cliente quando corrido *on-demand*.

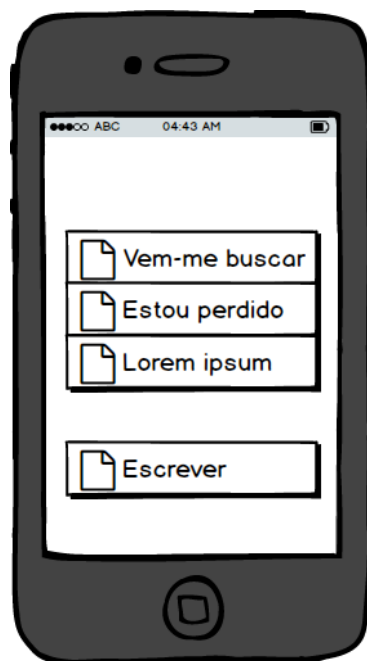


Figura 72 – Ecrã de seleção de mensagem SMS.



Figura 73 – Menu de captura de notas multimédia.