Instituto Politécnico de Tomar

Escola(s) Superior de Tecnologia de Tomar

Rafael Augusto Abreu Rodrigues

Estágio na Glintt

Projeto eDelivery

Relatório de Estágio

Orientado por:

Ana Cristina Barata Pires Lopes, Instituto Politécnico de Tomar
Pedro Daniel Frazão Correia, Instituto Politécnico de Tomar
Hugo Alexandre Novo, GLINTT Global

Relatório de estágio apresentado ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre em
Engenharia Informática – Internet das Coisas

RESUMO

Este relatório descreve o trabalho desenvolvido durante o estágio no âmbito da Unidade Curricular de Estágio do segundo ano do Mestrado em Engenharia Informática – Internet das Coisas. Este estágio decorreu na empresa GLINTT Global, com participação no programa anual de estágios Academia GLINTT, iniciado a 01/10/2023 e terminado a 31/05/2024.

Devido à necessidade de um sistema para a troca de documentos entre os países membros da União Europeia, foi desenvolvida uma consola de administração e uma *Application Programming Language* (API), através das tecnologias Angular e .NET, respetivamente. No relatório está relatada a contextualização do projeto, assim como as tarefas desenvolvidas durante o estágio para implementar as funcionalidades da aplicação, nas componentes de *frontend*, *backend* e de validação. São também mencionadas as tecnologias utilizadas no projeto para completar as diferentes tarefas.

Além disso, está descrita a metodologia Scrum, que serviu de apoio ao desenvolvimento do projeto, através de um processo iterativo e incremental.

Durante o estágio, foram adquiridas competências sobre várias tecnologias, através de cursos finalizados ao longo do mesmo, sendo explicados alguns dos conceitos aprendidos na sua realização.

Nos momentos finais do estágio, houve a participação num evento com outros países, para a realização de testes, de forma a verificar o funcionamento da aplicação desenvolvida.

Palavras-chave: *API, Angular, .NET, frontend, backend, Scrum*

ABSTRACT

This report describes the work developed during the internship as part of the Internship Curricular Unit in the second year of the Master's degree in Computer Engineering – Internet of Things. This internship took place at GLINTT Global, with participation in the annual internship program Academia GLINTT, starting on 01/10/2023 and ending on 31/05/2024.

Due to the need of a system to exchange documents between European Union member countries, an administration console and an *Application Programming Language* (API) were developed using Angular and .NET technologies, respectively. The report outlines the project background and details the tasks developed during the internship to implement the application's functionalities, including the *frontend*, *backend*, and *validation* components. The technologies used to complete the various tasks are also mentioned.

Additionally, the Scrum methodology, which supported the project's development through an iterative and incremental process, is described.

During the internship, several skills were acquired in various technologies through courses completed during that period, with some of the concepts learned in these courses being explained.

In the final stages of the internship, there was a participation in an event with other countries, where tests were made to verify the functionality of the developed application.

Keywords: *API*, *Angular*, *.NET*, *frontend*, *backend*, *Scrum*

AGRADECIMENTOS

A conclusão deste relatório de estágio teve a ajuda de várias pessoas, que me permitiram seguir motivado durante a participação no estágio e o desenvolvimento do relatório.

À minha família, pelo incentivo que me dão todos os dias para seguir em frente independentemente dos obstáculos, para conseguir alcançar os meus objetivos.

Aos meus coordenadores, a Professora Ana Cristina Barata Pires Lopes e o Professor Pedro Daniel Frazão Correia pela ajuda no decorrer do desenvolvimento do relatório e pelo apoio e partilha de conhecimentos durante todo o meu percurso académico.

Ao meu orientador de estágio na empresa Hugo Alexandre Novo que contribuiu para a minha integração na empresa e que me ajudou com o esclarecimento de quaisquer dúvidas.

Aos meus colegas de equipa, que me ajudaram especialmente ao partilhar conhecimentos.

Aos meus amigos, que me apoiam todos os dias, o me faz manter motivado para conseguir seguir em frente.

O meu muito obrigado!

Índice

Índice de Figuras

Lista de Abreviaturas e Siglas

AP – *Access Point*

API – *Application Programming Language*

CRUD – *Create Read Update Delete*

CSS – *Cascading Style Sheets*

DOM – *Document Object Model*

DTO – *Data Transfer Object*

GIF – *Graphics Interchange Format*

HTML – *Hypertext Markup Language*

HTTP – *Hypertext Transfer Protocol*

IDE – *Integrated Development Environment*

JSON – *JavaScript Object Notation*

JSX – *JavaScript XML*

LINQ – *Language Integrated Query*

MB - *Megabyte*

MVC – *Model-View-Controller*

ORM – *Object-Relational Mapping*

PDF – *Portable Document Format*

SQL – *Structured Query Language*

URL – *Uniform Resource Locator*

XML – *Extensible Markup Language*

1 Introdução

1.1 Contextualização e Motivação

Este relatório descreve o trabalho realizado no âmbito da concretização da Unidade Curricular "Estágio" do segundo ano do Mestrado em Engenharia Informática – Internet das Coisas lecionado no ano letivo 2023/2024, no Instituto Politécnico de Tomar. No essencial este relatório reporta e enquadra as atividades de desenvolvedor de software desenvolvidas durante o estágio na empresa GLINTT Global, representando a participação no programa Academia GLINTT 2023. Para melhor enquadrar o trabalho desenvolvido, este relatório também apresenta algumas normas utilizadas, como a injeção de dependências, e a metodologia de trabalho praticada na equipa na qual este estágio incidiu e as tecnologias utilizadas no projeto em questão.

A GLINTT Global, com sede na Quinta da Beloura em Sintra, é uma empresa de matriz portuguesa, líder tecnológica, com um legado de mais de 30 anos de tecnologias, integradas no tecido da vida quotidiana [1]. Destaca-se por proporcionar transformações digitais, desde os cuidados de saúde até aos mais diversos setores empresariais.

A GLINTT Life é a marca da empresa relacionada à saúde, líder ibérica em tecnologia da saúde e dedicada à criação, desenvolvimento e reinvenção de soluções para apoiar os profissionais de saúde em diversas áreas, através da implementação de *software* para farmácias e hospitais [2][3].

A GLINTT Next, representa a marca ligada à consultoria de serviços tecnológicos. Este departamento desenvolve soluções em tecnologias emergentes que têm por objetivo antecipar as tendências e entregar as novidades tecnológicas que melhor se adaptam a cada negócio e que mais contribuem para a sua evolução [3].

Os valores da GLINTT Global podem ser divididos em quatro tópicos [4]:

- Pessoas – Existe um compromisso firme com as suas pessoas e em melhorar vidas através da tecnologia e as soluções são centradas no ser humano com um impacto positivo tangível;
- Experiência – Vasto conhecimento tecnológico, desde o panorama geral até aos pormenores e soluções informadas para o presente e o futuro.
- Compromisso – Dedicação profunda aos clientes, parceiros e sociedade, e a precisão e fiabilidade no fornecimento de tecnologia para uma mudança positiva.

- Ambição – Ultrapassar continuamente os limites, procurando redefinir o potencial da tecnologia e oferecer um valor excecional, na vanguarda da inovação.

A GLINTT Global conta com mais de 1200 colaboradores, estando cerca de 900 destes concentrados em Portugal, onde existem 5 escritórios e 2 centros de excelência e 300 colaboradores em Espanha, onde também se encontram 5 escritórios e 3 centros de excelência [1].

A Academia GLINTT consiste num programa que recruta 40 pessoas, recém-licenciadas ou recém-mestres nas áreas de Engenharia Informática, Ciências de Dados, Gestão, Economia, Bioquímica, Ciências Farmacêuticas e Biomédicas, entre outras. Este programa tem como objetivo integrar esses 40 estagiários em equipas da empresa, para que estes consigam desenvolver as suas capacidades e, no futuro, possam colaborar com a empresa [5].

O estágio decorreu a tempo inteiro, com horário das 9h às 18h e intervalo para almoço entre as 13h e as 14h, de segunda a sexta-feira. O estágio teve início a 1 de outubro de 2023 e terminou a 31 de maio de 2024.

Durante este período o trabalho desenvolvido foi integrado na equipa de desenvolvimento do projeto "eDelivery" [6], excetuando os meses que corresponderam à formação inicial na tecnologia Angular e a outras formações complementares, finalizadas no decorrer do estágio.

O projeto eDelivery é um projeto internacional de grande dimensão e que surgiu da necessidade da existência de uma plataforma com capacidade de efetuar a transferência de documentos específicos entre os países membros da União Europeia, através de um protocolo seguro e com uma arquitetura pré-estabelecida, sendo o cliente, neste caso, uma organização pública.

A integração na equipa de desenvolvimento como programador de *fullstack* aconteceu quando este projeto já se encontrava numa fase de desenvolvimento avançada. A equipa de desenvolvimento era constituída por dois programadores sénior, com foco principal no *backend* e um programador júnior *fullstack* que participou na edição anterior da Academia GLINTT.

No momento da integração na equipa, muitos dos problemas iniciais, especialmente os relacionados à funcionalidade base do *backend* já tinham sido analisados e solucionados, visto que este projeto já tinha cerca de um ano e meio no momento da integração na equipa. A componente de *frontend*, por sua vez, foi uma funcionalidade extra, sugerida previamente pelos membros da equipa e que consistiu numa consola de administração, com o objetivo de

auxiliar a gestão das diversas áreas da solução base através de uma interface gráfica, tornando essa gestão mais intuitiva, com a disponibilização de várias funcionalidades de gestão e monitorização. De realçar que esta plataforma também já se encontrava numa fase consideravelmente avançada de desenvolvimento.

Houve ainda um período de dois meses dedicado somente a completar formações, devido a alguns fatores externos relacionados com o cliente, visto que, como a aplicação se encontrava praticamente pronta e representava um projeto público, foi submetida a concurso público, o que justificou essa pausa. Sendo assim, nos meses finais, o foco do estágio incidiu na conclusão de formações em tecnologias da Microsoft, que foram recomendadas pela empresa, visto que os colaboradores da GLINTT Global têm acesso completo a todos os cursos disponíveis na plataforma Udemy [7].

1.2 Objetivos

Os principais objetivos do estágio tiveram dois domínios fundamentais, a formação e a participação no projeto internacional eDelivery. Estes objetivos variaram conforme as diferentes fases do projeto, porém podem ser devidamente separados de acordo com o tipo das tarefas desempenhadas.

O primeiro objetivo deste estágio foi completar um curso sobre a *framework* Angular, durante o período inicial de formação que demorou cerca de um mês. Os conhecimentos adquiridos durante a formação em Angular foram imprescindíveis, visto que, a equipa necessitava de elementos com conhecimento específico nesta tecnologia. Salienta-se que a *framework* Angular correspondia a uma tecnologia completamente nova para o aluno, com a qual não havia qualquer contacto prévio, tal como com a linguagem de programação TypeScript [8], relacionada à mesma, e outros conceitos aplicados à *framework* como o *Data Binding* [9].

O próximo objetivo está relacionado ao projeto eDelivery, desempenhado durante o estágio, e incidiu em aplicar os conhecimentos obtidos para implementar novas funcionalidades, como a criação ou modificação de páginas no *frontend*, associadas à formação anterior, e também há implementação de *handlers*, regras de validação e outras tarefas na *Application Programming Language* (API) do *backend*. Associado a isto, outro objetivo foi o de compreender como funcionavam as diferentes componentes do projeto, já desenvolvidas, as boas práticas utilizadas e a forma como a lógica era aplicada e dividida dentro da equipa, em contexto profissional.

Destaca-se ainda o objetivo no domínio da formação que consistiu na realização de alguns cursos durante os dois meses em que o projeto esteve parado. Estes cursos de formação, em diferentes linguagens de programação, *frameworks* e bibliotecas, foram incluídos num caminho de aprendizagem definido pela hierarquia da empresa. A lista de cursos realizados é apresentada de seguida:

- C# Intermediate: Classes, Interfaces and OOP;
- .NET Core MVC – The Complete Guide;
- Entity Framework in depth: The Complete Guide;
- Microsoft SQL for beginners;
- Responsive Web Design Essentials – HTML5 CSS3 Bootstrap;
- Dapper – Getting Started;
- React Native – The Practical Guide;
- C# Advanced Topics – Prepare for Technical Interviews.

Por fim, o último objetivo passou por tentar obter uma boa classificação num evento em que a equipa participou e que estava relacionado com a execução de testes de funcionamento da aplicação desenvolvida, aplicando o conhecimento obtido durante um período de preparação para esse evento, que durou cerca de um mês. Para além disso, houve ainda o objetivo de melhorar a fluência na comunicação com outras pessoas em inglês, incentivada pela necessidade de comunicar com as equipas dos restantes estados-membro da União Europeia para executar esses testes.

1.3 Contribuições

As contribuições estão intrinsecamente relacionadas à aplicação dos conhecimentos adquiridos nos primeiros momentos do estágio e na formação académica que a antecedeu, à respetiva análise do código da solução já efetuada e à realização de validações.

De seguida enumeram-se as contribuições do trabalho realizado:

- Formação em Angular, onde foram aprofundadas várias propriedades dessa *framework*, que foram posteriormente aplicadas em formato de trabalho no decorrer do desenvolvimento da aplicação, especialmente nas tarefas dedicadas ao componente de *frontend*.
- Participação na equipa de desenvolvimento do projeto eDelivery, especialmente na execução de funcionalidades de *frontend*.
- Refinação e alteração de desenvolvimentos, não relacionados ao funcionamento base da aplicação.

- Alteração de um protocolo de comunicação da arquitetura existente, graças à inclusão de uma nova máquina no sistema.
- Formações em tecnologias da Microsoft, recomendadas pela hierarquia aos estagiários da empresa, devido ao acesso completo a todos os cursos disponíveis na plataforma Udemy.
- Participação num evento para a monitorização do ponto de situação do desenvolvimento da aplicação em conjunto com diferentes países, incluindo a execução de testes entre esses países.

2 Fundamentos do projeto

Este capítulo descreve a metodologia de trabalho utilizada pela equipa no decorrer do desenvolvimento, assim como as tecnologias utilizadas, a arquitetura da aplicação e a forma como os diferentes componentes funcionam.

2.1 Metodologia Ágil

Neste projeto utilizou-se uma metodologia ágil para apoiar o desenvolvimento da aplicação. As metodologias ágeis, referidas comumente como *Agile*, são abordagens de gestão de projeto utilizadas para a criação e evolução de produtos e que sustentam a entrega e melhoria contínua, através de um processo iterativo e incremental, suportando-se na colaboração de equipas, que seguem um ciclo de planeamento, execução e avaliação.

Este tipo de abordagem permite confirmar precocemente se o desenvolvimento está a caminhar na direção correta, facilitando a introdução de correções e melhorias, caso necessário [10].

A metodologia que se utilizou durante toda a duração do estágio foi o Scrum que é uma metodologia de desenvolvimento ágil utilizada no desenvolvimento de *software* organizada em iterações curtas conhecidas como *sprints* [11]. O Scrum consiste numa *framework* ágil, adaptável, rápida, flexível e eficaz projetada para oferecer valor ao cliente durante todo o desenvolvimento do projeto. Relativamente ao principal objetivo do Scrum, passa por satisfazer a necessidade do cliente através de um ambiente de transparência na comunicação, responsabilidade coletiva e progresso contínuo. O desenvolvimento parte de uma ideia geral do que é necessário construir, elaborando uma lista de características ordenadas por prioridade chamada *Product Backlog*.

Existem alguns valores associados ao Scrum, como coragem, foco, compromisso, respeito e abertura. Esses valores devem ser adotados pelos membros da equipa de desenvolvimento ao trabalhar em conjunto, pois são essenciais para criar ambientes onde a experimentação é essencial para progredir [12]:

1. Coragem: A equipa de desenvolvimento tem a coragem para fazer a coisa certa e trabalhar em problemas difíceis;
2. Foco: Todos os membros se focam no trabalho da *sprint* e nos objetivos da equipa de desenvolvimento;
3. Compromisso: Os membros comprometem-se pessoalmente para alcançar os objetivos da equipa de desenvolvimento;

4. Respeito: Os membros da equipa de desenvolvimento respeitam-se mutuamente para serem pessoas capazes e independentes.
5. Transparência: A equipa de desenvolvimento e os *stakeholders* concordam em ser transparentes sobre todo o trabalho e os desafios enfrentados durante o desenvolvimento do trabalho.

Os valores referidos anteriormente encontram-se ilustrados na Figura 1, assim como os três pilares do Scrum que são a transparência, a inspeção e a adaptação [12].

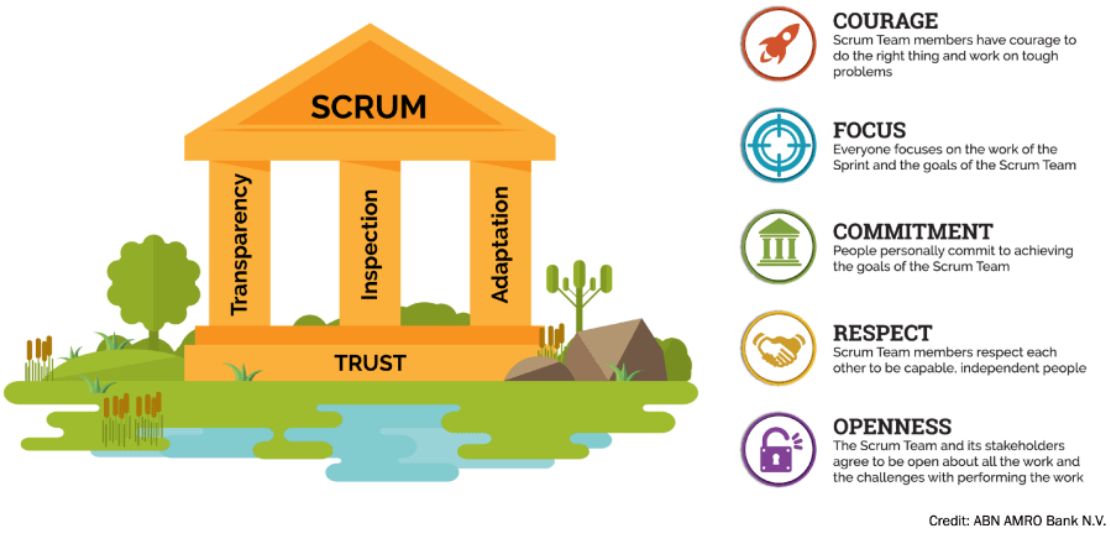

Figura 1 Pilares e valores do Scrum [12].

O Scrum é ainda baseado num conjunto de práticas e funções que devem ser tidas em conta durante o processo de desenvolvimento de software, abrangendo os 12 princípios ágeis [13]:

1. Satisfazer o cliente através da entrega antecipada e contínua;
2. Aceitar mudanças de requisitos;
3. Entregar valor com frequência;
4. Trabalhar em conjunto;
5. Construir projetos em torno de indivíduos motivados;
6. A forma mais eficaz de comunicação é presencialmente;
7. Software funcional é a principal medida de progresso;
8. Manter um ritmo de trabalho sustentável;
9. A excelência contínua aumenta a agilidade;
10. Simplicidade é essencial;
11. Equipas auto-organizadas geram mais valor;
12. Refletir e ajustar regularmente como ser mais eficaz.

As *sprints* normalmente duram entre duas e quatro semanas [11]. Neste projeto foram realizadas *sprints* de duas semanas, nas quais a equipa tem o objetivo de entregar um pequeno incremento do projeto global.

Uma equipa do Scrum integra as seguintes funções principais:

1. *Scrum Master*: É responsável por garantir que a equipa compreenda e siga os princípios e práticas associadas ao Scrum, remover eventuais obstáculos que possam impedir o progresso da equipa e facilitar o agendamento de reuniões, auxiliando a equipa a tornar-se mais eficiente e focada [11].
2. *Product Owner*: É o representante dos interesses do cliente, *stakeholders* e utilizadores finais, responsável por gerir o *Product Backlog*, definir as prioridades dos *items* com base no valor para o cliente, traçar as metas das *sprints* e especificar o que é necessário para atingir essas metas. De notar que no estágio esta gestão de prioridades e metas foi feita pelo *Scrum Master*, em conjunto com a equipa de desenvolvimento [11].
3. Equipa de desenvolvimento: É composta por um grupo de profissionais com os conhecimentos técnicos necessários para transformar os *items* do *Product Backlog*, com os quais se comprometem no início de cada *sprint*, em incrementos do projeto [11].

No que diz respeito aos eventos do Scrum, estes facilitam a adaptação de alguns aspetos do processo, como o desenvolvimento do produto, do progresso e das relações [11]:

1. *Sprint Planning*: É realizado no início de cada *sprint*, tendo por objetivo definir o que será feito na *sprint* e como será realizado. É também nesta fase que se associa quem ficará responsável por cada *item* do *Backlog*.
2. *Daily Scrum*: Consiste numa reunião curta e que se realiza diariamente durante o período da *sprint*. O objetivo é avaliar o progresso efetuado desde a última reunião, sincronizando as atividades e criando um plano de progresso para o próximo dia.
3. *Sprint Review*: O propósito é rever e avaliar o trabalho que se concluiu durante a *sprint*, de forma a receber feedback do *Product Owner*, do cliente ou de eventuais partes interessadas. No desenvolvimento do projeto, foi o *Product Owner* que participou nestas reuniões.
4. *Sprint Retrospective*: Trata-se de um evento onde a equipa revê os objetivos alcançados na *sprint*, apontando o que funcionou e o que pode ser melhorado, de forma a evitar a repetição de erros no futuro.

De notar que, além destes eventos, desempenhou-se um quinto evento durante o estágio, chamado *Done Review*. Este evento foi realizado três vezes por semana, todas as segundas, quartas e sextas-feiras, sempre que alguém concluía novas tarefas no decorrer da *sprint*, com o objetivo de rever essas tarefas entre os membros da equipa.

Na Figura 2, é possível verificar o planeamento dos eventos que foram utilizados na adaptação da metodologia Scrum, aplicados à funcionalidade de calendário do Microsoft Teams.

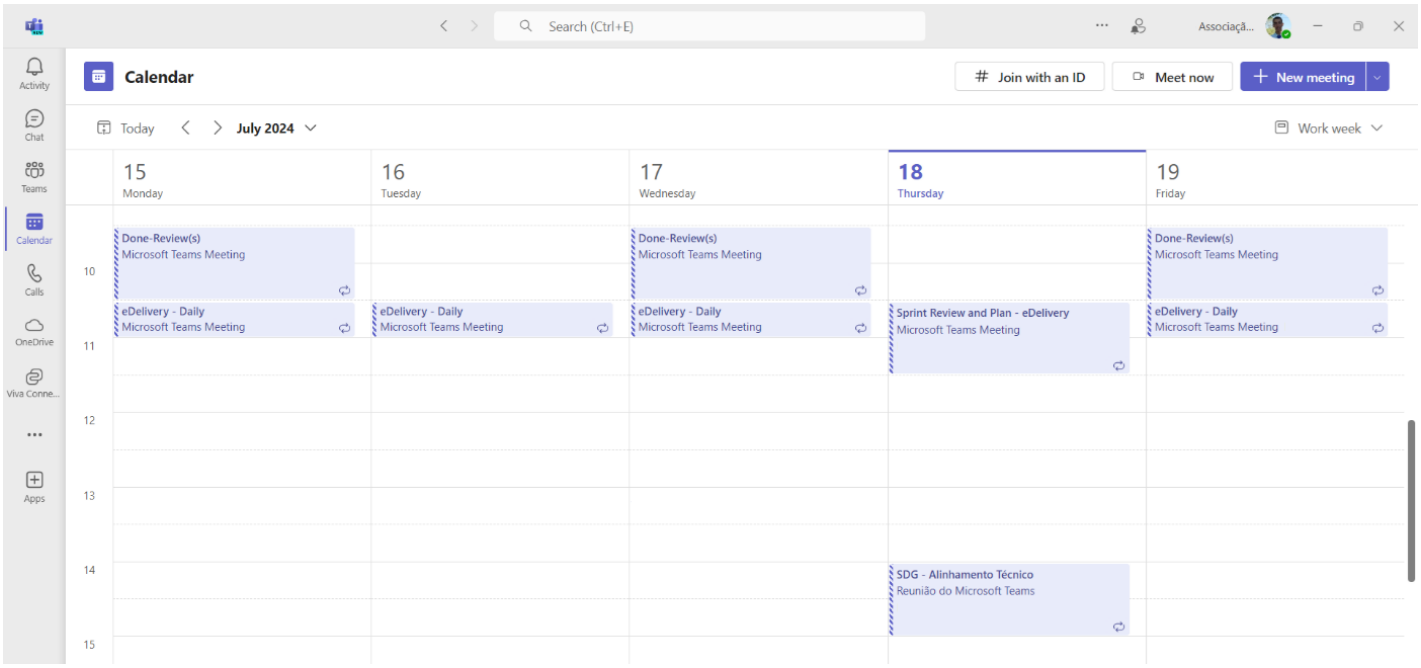

Figura 2 Reuniões Scrum aplicadas ao Microsoft Teams.

Por fim, existem ainda os artefactos do Scrum, que são projetados para garantir a transparência das informações mais importantes na tomada de decisões [11]:

1. *Product Backlog*: É uma lista que reúne tudo o que o produto precisa para satisfazer os clientes, é mantida pelo *Product Owner*, sendo as funções priorizadas de acordo com o grau de importância para o negócio. No estágio foi mantida também pelo Scrum *Master* e por alguns membros da equipa de desenvolvimento.
2. *Sprint Backlog*: É um subconjunto de *items* do *Product Backlog*, selecionados pela equipa de desenvolvimento, para que sejam implementados durante a *sprint*.

Na Figura 3, está representado o ciclo de desenvolvimento do Scrum, no qual se podem observar os eventos, a equipa e os artefactos do Scrum, explicados anteriormente de forma ordenada.

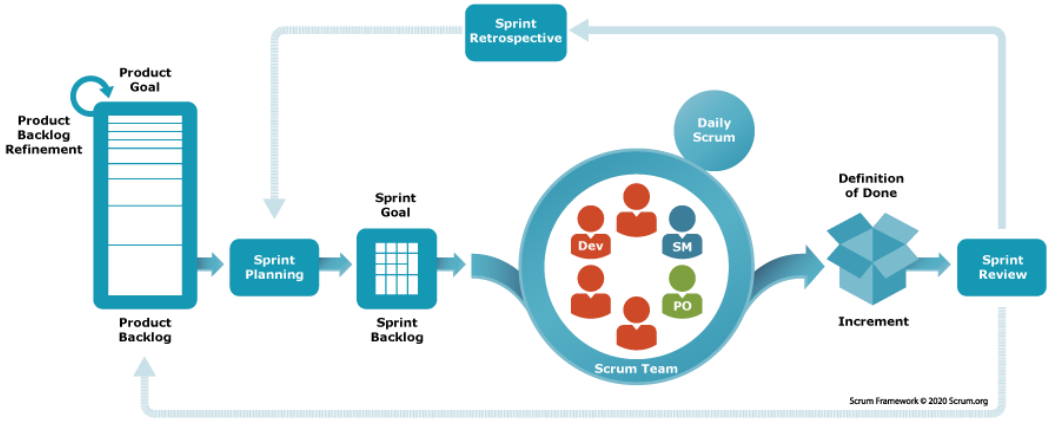

Figura 3 Ciclo de desenvolvimento do Scrum. [12]

2.2 Ambiente de Desenvolvimento

No desenvolvimento da aplicação, para além da utilização da metodologia Scrum para organizar o método como o trabalho é desempenhado, foram utilizadas várias tecnologias, como linguagens de programação relacionadas a *frameworks* de *frontend* e *backend*, algumas ferramentas de gestão de tarefas para apoiar o Scrum, um sistema de gestão de base de dados e um software para gerir os repositórios da aplicação.

2.3 Tecnologias Utilizadas

2.3.1 Microsoft Teams

Relativamente à forma como as reuniões e comunicações foram feitas em contexto de trabalho, utilizou-se o Microsoft Teams, é uma aplicação de comunicação e colaboração desenvolvida pela Microsoft. Esta ferramenta foi projetada para facilitar o trabalho em equipa, oferecendo diversas funcionalidades integradas para aumentar a produtividade e a comunicação dentro das organizações [14].

Uma das funcionalidades mais utilizadas são as mensagens de *chat* [15], nas quais o utilizador pode enviar mensagens de texto instantâneas para um individuo particular ou para um grupo. Essas mensagens são organizadas em tópicos, de forma a facilitar a leitura e acompanhamento da conversa. Além de simplesmente escrever em formato de texto, o *Microsoft Teams* possibilita aos seus utilizadores enviar *emojis*, *Graphics Interchange Formats (GIFs)*, adesivos e ainda reagir a cada mensagem, de forma a tornar a

comunicação mais fluída e expressiva. Por fim, relativamente às mensagens, é possível notificar um colega ou um grupo completo diretamente, de maneira a facilitar que a mensagem seja vista.

Outro recurso são as chamadas de vídeo, que permitem a comunicação por voz e vídeo, ao vivo, entre dois ou mais participantes. Também é possível efetuar o compartilhamento de ecrã, que é maioritariamente utilizado para apresentações ou desenvolvimento monitorizado [16]. Ainda relacionado às apresentações, é possível gravar reuniões para que possam ser revistas mais tarde [16]. O agendamento dessas reuniões pode ser feito diretamente no Microsoft Teams ou através da sincronização com o calendário do Outlook [17].

Uma funcionalidade que também é comumente utilizada a nível organizacional é a modificação de documentos em simultâneo, diretamente no Microsoft Teams, possibilitando verificar as alterações efetuadas em tempo real [18].

Existem também as equipas, que podem ser criadas para diferentes departamentos ou projetos, para as quais os membros devem ser convidados [19]. Dentro de uma equipa, é possível criar *chats* para conversar ou partilhar ficheiros entre membros de equipas [19]. Lembrando que, podem ser definidas diferentes permissões para cada membro, garantindo um nível hierárquico, se necessário [19].

O Microsoft Teams garante a segurança nas comunicações da plataforma através de criptografia, sendo que estas são criptografadas tanto em trânsito como em repouso, garantindo a segurança da informação [20]. Outro fator de segurança é ao nível da autenticação, uma vez que a plataforma suporta autenticação com múltiplos fatores.

Além das funcionalidades descritas anteriormente, o Microsoft Teams contém integrações com outras aplicações, como o mercado de aplicações no qual os utilizadores podem adicionar ferramentas adicionais, como o Trello [21] e o GitHub [22].

Por fim, relativamente à utilização do Microsoft Teams neste projeto, foi uma das ferramentas mais utilizadas, uma vez que todos os dias era exigida a comunicação entre os colegas da equipa. Através do Microsoft Teams, foi possível acompanhar as reuniões da metodologia Scrum com o auxílio da funcionalidade de calendário e ter uma boa coordenação e organização do projeto, através de vários *chats* criados conforme o tópico e os participantes necessários, garantindo também a troca de ficheiros e código.

2.3.2 Source Tree

O SourceTree é uma ferramenta gráfica para facilitar a interação com sistemas distribuídos de controlo de versões [23], como o Git [24], e que foi desenvolvida pela Atlassian [25]. Esta ferramenta oferece uma interface gráfica para a gestão desses sistemas, tornando-a mais intuitiva do que a gestão típica através da linha de comandos.

Uma das principais funcionalidades do *Git*, o *commit*, serve para registar uma instância das alterações num ou mais ficheiros de um projeto no momento atual [26]. Cada *commit* possui um identificador exclusivo para rastrear alterações feitas, o momento exato em que ocorreram e quem as realizou [27]. O SourceTree destaca-se por permitir uma visualização organizada do histórico de *commits*, possibilitando observar em detalhe cada mudança efetuada num *commit* como adições, edições ou remoções de linhas de código ou de ficheiros.

Esta ferramenta facilita também o uso de outras operações, como o *push* e o *pull*, fundamentais no fluxo de trabalho com repositórios remotos. O *push* refere-se à ação de enviar as alterações locais para o repositório remoto, permitindo que outros colaboradores acedam ao trabalho mais atualizado [28]. Isto garante que todos os membros da equipa possam sincronizar as suas alterações. Por outro lado, o *pull* consiste em trazer as alterações do repositório remoto para o ambiente local, garantindo que o programador tenha a versão mais recente do código antes de prosseguir com novas modificações [29]. O SourceTree simplifica estes processos através de botões dedicados para estas operações, evitando a necessidade de usar comandos complexos.

Além disso, existem algumas outras funcionalidades que esta aplicação oferece, como a clonagem de repositórios remotos para o ambiente local e a possibilidade de guardar alterações temporariamente, mesmo quando ainda não foram sincronizadas, sem existir a perda de progresso.

Portanto, o SourceTree foi utilizado para efetuar todas as tarefas relacionadas à gestão do repositório Git do projeto, possibilitando a partilha das alterações efetuadas no código.

2.3.3 JIRA

O processo de organização das tarefas no decorrer do desenvolvimento do projeto, foi realizado utilizando a aplicação JIRA, que é um software para organizar tarefas, gerir projetos e automatizar fluxos de trabalho [25].

No caso do estágio, este software foi utilizado para gerir as tarefas, mostrando-se muito útil durante as *sprints*. Foi a plataforma onde as tarefas foram atribuídas a cada membro da equipa e na qual se definiu as suas prioridades, assim como a *sprint* à qual

estavam associadas. Sendo assim, permitiu organizar as tarefas em vários estados no decorrer das *sprints*, tais como:

- *To Do*: Nesta coluna estavam representadas as tarefas por fazer durante a *sprint*, cada uma com um membro da equipa associado, um título a informar o tipo de implementação e uma breve descrição sobre a mesma.
- *In Progress*: Nesta coluna ficavam as tarefas que cada membro estava a desenvolver no momento.
- *Blocked*: As tarefas que apresentavam algum bloqueio que impedia a conclusão da sua implementação eram colocadas temporariamente nesta coluna, até que esse bloqueio fosse resolvido.
- *In Review*: Neste estado estavam as tarefas concluídas e que ainda não tinham sido revistas e aprovadas por todos os membros da equipa na reunião *Done Review*.
- *Done*: A coluna *Done* representava a última etapa do processo de implementação das tarefas, onde ficavam as tarefas que tinham sido concluídas e, posteriormente, revistas e aprovadas pelos membros da equipa.

2.3.4 SQL Server

A gestão da base de dados do projeto foi realizada através do SQL Server, que é um sistema de gestão de base de dados relacional [30], em conjunto com a biblioteca Entity Framework Core do Visual Studio.

Desta forma, sempre que foi surgiu a necessidade de consultar ou testar determinados registos ou atributos de determinadas tabelas, recorreu-se a esta ferramenta, especialmente para analisar se determinadas alterações efetuadas no código foram aplicadas corretamente.

2.3.5 Visual Studio 2022

O *Integrated Development Environment* (IDE) utilizado para o desenvolvimento das componentes de *backend* e de validação do projeto foi o Visual Studio 2022, que foi desenvolvido pela Microsoft e que permite utilizar várias linguagens de programação, *frameworks* e ferramentas para a resolução de erros. É bastante utilizado no desenvolvimento de software para várias plataformas, como *Desktop*, *Web*, *Mobile* e até na implementação de APIs [31].

Algumas das funcionalidades do Visual Studio 2022 mais utilizadas durante o projeto foram:

- ASP.NET Core: É uma *framework* de código aberto e multiplataforma, aplicada na criação e gestão de APIs e aplicações *web* de alto desempenho.
- Entity Framework: É o *Object-Relational Mapping* (ORM) mais popular do ASP.NET e foi utilizado para a manipulação de dados e operações de base de dados de maneira eficiente.
- Fluent Validation: É uma biblioteca para criar regras de validação de forma intuitiva, disponibilizando métodos para auxiliar nessa implementação.
- NUnit: É uma *framework* para desenvolver testes unitários, com suporte para todas as linguagens de programação da plataforma ASP.NET, que permitiu a automação de testes para confirmar o comportamento de determinadas regras de validação.

2.3.5.1 ASP.NET Core e arquitetura de desenvolvimento

O ASP.NET Core foi fundamental para a criação de APIs que suportam o fluxo de dados entre as componentes da aplicação. Esta *framework* facilita o desenvolvimento de várias aplicações, oferecendo uma grande flexibilidade no suporte a linguagens como o C#, que foi a linguagem de programação utilizada no *backend* deste projeto.

A arquitetura utilizada no desenvolvimento do *backend* seguiu o padrão *Model-View-Controller* (MVC) [32]:

- *Model*: Representa a camada de dados, incluindo a definição de classes para as tabelas da base de dados e a lógica de acesso e manipulação dos dados. Os *models* foram implementados com o Entity Framework para simplificar as operações com os dados.
- *View*: Embora o projeto tenha tido uma camada de *frontend* separada, desenvolvida em Angular, o conceito de *View* representa a lógica para exibir as páginas aos utilizadores.
- *Controller*: Esta é a camada intermediária entre a *View* e o *Model*. No estágio, o *Controller* representou os *handlers* da API desenvolvida, responsável pelo processamento dos pedidos recebidos do *frontend*, executando as operações necessárias e retornando objetos *JavaScript Object Notation* (JSON).

2.3.5.2 Padrão de projeto – Repository pattern

Neste projeto foi utilizado um padrão de projeto chamado *repository pattern* [33] dentro da estrutura da API, desenvolvida no Visual Studio, para manter a organização e

promover a reutilização de código. Seguindo esse padrão, os pedidos *Hypertext Transfer Protocol* (HTTP) foram geridos por *Controllers*, classes que serviram como interface entre o *frontend* e a lógica do *backend*. Cada *Controller* tinha vários *handlers* que utilizavam métodos de classes chamadas *Processors*, onde era feito o processamento dos dados recebidos desses *handlers*, aplicando a lógica de negócio específica.

Por fim, havia os repositórios, que são classes cujos métodos eram chamados nas funções dos *Processors*, também fazendo parte desse padrão de projeto. A função desses métodos era aplicar as operações necessárias à base de dados, permitindo a abstração da interação com os dados e garantindo que a lógica de acesso esteja desacoplada de funcionalidades específicas.

2.3.5.3 Métodos de teste e validações

Para desenvolver regras de validação, utilizou-se uma biblioteca do Visual Studio chamada Fluent Validation. Esta biblioteca permite aos desenvolvedores criar regras de validação de forma fluente e intuitiva, tornando o código legível e fácil de manter [34]. A Fluent Validation disponibiliza alguns métodos principais para auxiliar essas regras, como por exemplo:

- *RuleFor*: Utilizado para definir uma regra para uma propriedade específica de uma classe;
- *Must*: Permite implementar lógica personalizada para testar um comportamento específico, retornando um valor booleano.
- *WithMessage*: Serve para definir uma mensagem de erro personalizada, para ser exibida caso a regra falhe.

Para garantir que as regras de validação se comportavam corretamente, estas foram aplicadas a testes unitários já existentes, utilizando a *framework* NUnit, disponível no ASP.NET, que oferece várias funcionalidades para facilitar esses testes [35][36]:

- Métodos de teste: A cada método pode ser atribuído o atributo “[Test]”, que informa o NUnit que esse método representa um teste unitário.
- *Assertions*: As *assertions* são parte fundamental dos testes no NUnit, permitindo comparar o resultado esperado com o obtido. Alguns exemplos incluem:
  - Assert.IsTrue(valor obtido): testa se o valor obtido é verdadeiro;
  - Assert.IsFalse(valor obtido): testa se o valor obtido é falso;
  - Assert.AreEqual(valor esperado, valor obtido): testa se dois valores são iguais.

A Figura 4 mostra o exemplo de uma regra de validação no construtor da classe *TestValidator*, assim como um método de teste unitário chamado *Name_MustNotBeEmpty*, desenvolvido com a *framework* NUnit. O método utiliza o atributo "[Test]" e retorna o resultado de uma *assertion* para confirmar se o resultado da regra de validação é falso.

```
using NUnit.Framework;
using FluentValidation;
using FluentValidation.Results;

public class TestObject
{
    public string Name { get; set; }
}

public class TestValidator : AbstractValidator<TestObject>
{
    public TestValidator()
    {
        RuleFor(x => x.Nome)
        .Must((object, attribute) =>
        {
            return string.IsNullOrEmpty(attribute);
        })
        .WithMessage("The name must not be empty.");
    }
}

[TestFixture]
public class ValidatorTests
{
    private TestValidator _validator;

    [SetUp]
    public void Setup()
    {
        _validator = new TestValidator();
    }

    [Test]
    public void Name_MustNotBeEmpty()
    {
        var obj = new TestObject { Name = "" };
        ValidationResult result = _validator.Validate(obj);

        Assert.IsFalse(result.IsValid);
    }
}
```

Figura 4 Exemplo de implementação de uma regra de validação e de um teste unitário para verificar o comportamento dessa regra, através da bilbioteca Fluent Validation e da framework NUnit.

2.3.6 Angular

No que se refere à componente de *frontend*, esta foi desenvolvida através da *framework* Angular. Trata-se de uma *framework* de código aberto que utiliza a arquitetura MVC para criar de aplicações *web* dinâmicas de página única [37]. Nestas aplicações, a interação e navegação entre as diferentes secções de uma página ocorrem de forma fluida, sem a necessidade de recarregar a página a cada alteração efetuada. A *framework* Angular é baseada em componentes, permitindo dividir o projeto em partes menores e reutilizáveis, sendo que cada componente é composto por ficheiros *Cascading Style Sheets* (CSS), *Hypertext Markup Language* (HTML) e TypeScript, nos quais se define o comportamento e a apresentação desse componente [38].

Os componentes funcionam como blocos de construção fundamentais para o desenvolvimento das interfaces gráficas e encapsulamento de funcionalidades. Estes promovem a reutilização e modularidade, contribuindo para manter a consistência, reduzir a duplicação de código, e criar interfaces complexas por meio da combinação de vários componentes organizados hierarquicamente. Os ficheiros que compõem esses componentes possuem as seguintes características:

- Modelo HTML: Define a estrutura do conteúdo a ser apresentado ao utilizador. Este ficheiro é crucial, pois descreve como os dados e a lógica se manifestam visualmente.
- Ficheiro TypeScript: Serve para implementar a lógica e o comportamento do componente. Nele, são definidas as propriedades e métodos que controlam a forma como o componente responde às interações do utilizador, eventos e manipulações de dados.
- Ficheiro CSS: Serve para aplicar estilos visuais, determinando a aparência dos elementos do ficheiro HTML. Os estilos definidos num componente são encapsulados, ou seja, aplicam-se apenas ao componente específico a que estão associados, evitando conflitos de estilos entre diferentes componentes.

Um dos recursos centrais do Angular é o *Data Binding* [9], que possibilita a sincronização automática dos dados entre o modelo TypeScript de um componente e a sua exibição no modelo HTML. Este recurso simplifica a manipulação de dados, tornando a interface mais dinâmica e responsiva, existindo dois tipos principais de *Data Binding*.

2.3.6.1 One-Way Binding

No *One-Way-Binding*, os dados fluem numa única direção, seja do componente para o modelo HTML ou do modelo HTML para o componente, permitindo que a interface reflita o estado atual do modelo TypeScript correspondente [39]. Dentro deste conceito, destacam-se três técnicas fundamentais:

- *String Interpolation*: Permite a inserção de valores do modelo TypeScript diretamente nos elementos HTML, utilizando a sintaxe "{{ valor }}". Esta abordagem é frequentemente usada para exibir textos dinâmicos no HTML, atualizando a interface sempre que o valor do modelo TypeScript é alterado. Um exemplo prático pode ser observado na Figura 5, onde o valor da propriedade *name* do modelo "app.component.ts" é inserido diretamente no modelo

"app.component.html" através da sintaxe mencionada, e à direita da imagem pode ver-se o resultado num navegador.

Figura 5 Exemplo de Data Binding, através de String Interpolation.

- *Property Binding*: Liga os valores do modelo TypeScript a propriedades específicas dos elementos HTML, usando a sintaxe '[propriedade] = "valor"' para atualizar dinamicamente atributos dos elementos, fazendo com que a interface reflita automaticamente o valor atualizado [40]. Na Figura 6 e na Figura 7, estão representados dois exemplos desta técnica, onde se usa *Property Binding* para manipular a propriedade *disabled* de um botão, com base no valor do atributo *isDisabled*, no modelo "app.component.ts".

Figura 6 Exemplo de Data Binding através de Property Binding para ativar um botão HTML, baseado no valor de um atributo do modelo TypeScript do componente.

Figura 7 Exemplo de Data Binding através de Property Binding para desativar um botão HTML, baseado no valor de um atributo do modelo TypeScript do componente.

- *Event Binding*: Permite ao modelo HTML do componente reagir a eventos do utilizador, ligando esses eventos a métodos do modelo TypeScript através da sintaxe '(evento) = "método()"'. Isso permite que a interface responda a interações e execute o código do método vinculado [41]. Um exemplo pode ser verificado na Figura 8 e na Figura 9, onde o *Event Binding* é usado para alterar o texto de um botão com base no valor de um atributo do modelo "app.component.ts", através do método *updateMessage*, vinculado ao evento *click*. Na Figura 8, exibe o valor "Click me", enquanto na Figura 9, após o clique, o texto muda para "Clicked!".

Figura 8 Exemplo de Data Binding através de Event Binding para alterar o texto de um botão, com a manipulação de um atributo, por um método, antes da chamada a esse método.

Figura 9 Exemplo de Data Binding através de Event Binding para alterar o texto de um botão, com a manipulação de um atributo, por um método, depois da chamada a esse método.

2.3.6.2 Two-Way Binding

O *Two-Way-Binding* combina as técnicas de *Property Binding* e *Event Binding*, permitindo que os dados fluam em ambas as direções. Isso significa que as alterações feitas no *modelo* HTML atualizam automaticamente o modelo TypeScript e vice-versa [42]. Através da sintaxe '[(ngModel)] = valor', qualquer modificação no modelo HTML reflete-se imediatamente no modelo TypeScript. Como ilustrado na Figura 10 e na Figura 11, é possível observar a alteração do atributo *text* do modelo TypeScript por meio da modificação do *input* no HTML.

Figura 10 Exemplo de Two-Way-Binding, através da alteração de um input no HTML, antes dessa alteração.

Figura 11 Exemplo de Two-Way-Binding, através da alteração de um input no HTML, depois dessa alteração.

Outro recurso essencial são as diretivas, que oferecem a capacidade de manipular o *Document Object Model* (DOM), interagindo diretamente com os elementos HTML. Por meio delas, é possível criar novos elementos, modificar os existentes ou até adicionar atributos ao HTML [43]. Existem dois tipos principais de diretivas no Angular: as diretivas estruturais, como "*ngIf" e "*ngFor", que alteram a estrutura do DOM com base

em condições, e as diretivas de atributos, como “ngClass” e “ngStyle”, que modificam a aparência de elementos já existentes.

Para facilitar a navegação entre secções, o Angular disponibiliza um sistema de encaminhamento, permitindo o mapeamento de *Uniform Resource Locators* (URLs) para os componentes necessários. Este sistema também suporta a funcionalidade de *lazy loading*, otimizando o desempenho ao carregar apenas os recursos necessários, em vez de carregar tudo de uma vez sem necessidade [44].

Por fim, para o desenvolvimento de formulários, o Angular oferece suporte a dois tipos principais: *Template-Driven* e *Reactive Forms*. Enquanto os *Template-Driven* são mais simples de usar, os *Reactive Forms* oferecem um maior controlo na definição da estrutura em TypeScript. Ambos os tipos de formulário permitem validações síncronas e assíncronas em tempo real [45].

3 Fundamentos adquiridos em contexto de formação

Após o período de desenvolvimento do projeto, houve uma pausa na qual o foco alterou para a conclusão de alguns cursos pertencentes a um caminho de aprendizagem, sugerido no início do estágio. Esses cursos fazem parte da plataforma Udemy e apesar do conteúdo da maioria deles não ter sido aplicado em contexto de trabalho, serviram para consolidar o conhecimento em determinadas tecnologias.

3.1 C# Intermediate: Classes, Interfaces and OOP

O primeiro curso completado está relacionado com a aprendizagem e consolidação de alguns conceitos da programação orientada a objetos aplicados à linguagem de programação C#, os quais foram utilizados anteriormente nas tarefas do estágio. Esses conceitos serviram para a revisão de algumas informações, com as quais já existia um certo domínio, como a criação de classes, construtores, métodos, atributos e as relações entre objetos. Porém permitiram também reforçar e adquirir conhecimentos em outras áreas não tão familiares, como as interfaces e a injeção de dependências, que começou a ser utilizada mais proeminentemente durante o estágio, como explicado anteriormente.

3.2 .NET Core MVC – The Complete Guide

O conteúdo deste curso também está relacionado à linguagem C#, visto que geralmente os projetos da *framework* .NET têm como base essa linguagem de programação. No entanto, esse não foi o foco do curso, que incidiu principalmente em explicar o funcionamento de vários aspetos da *framework* de forma mais detalhada.

Por exemplo, o ficheiro "*launchSettings.json*", que costuma estar presente ao iniciar um projeto por defeito, define perfis de desenvolvimento com diversas variáveis de ambiente, essenciais à inicialização da aplicação.

Outra componente que está sempre presente é a pasta *wwwroot*, utilizada para hospedar todo o conteúdo estático do projeto, como ficheiros CSS, JavaScript, pacotes NuGet, bibliotecas, ou até imagens ou *Portable Document Formats* (PDFs). Outro ficheiro abordado e relevante, é o "*appsettings.json*", utilizado para guardar credenciais secretas, como as credenciais da base de dados de cada ambiente. Ao criar diferentes ficheiros *appsettings* para cada ambiente, é possível, nos perfis do ficheiro "*launchSettings.json*", definir uma variável de ambiente para indicar qual desses documentos deve ser utilizado em cada perfil.

Outro ponto interessante abordado durante o curso, e que colaborou para uma melhor compreensão da injeção de dependências, foi a forma como funcionam os diferentes ciclos de vida dos serviços injetados, sendo explicados os seguintes tipos:

- *Scoped* – Este ciclo de vida foi o primeiro com o qual houve contacto ao iniciar a aprendizagem do funcionamento da injeção de dependências. É o tipo de ciclo de vida que geralmente é utilizado por defeito no desenvolvimento de aplicações *web* e, portanto, foi o mais utilizado durante o estágio. O tipo *Scoped* cria um ciclo de vida para o serviço injetado a cada requisição HTTP, garantindo que a mesma instância seja utilizada durante essa requisição, porém será criada uma nova instância para cada nova requisição.
- *Transient* – Neste tipo de ciclo, um novo ciclo de vida do serviço é iniciado sempre que se efetua uma solicitação desse serviço, mesmo dentro da mesma requisição HTTP. Este tipo de ciclo de vida, no entanto, não foi utilizado durante o estágio.
- *Singleton* – Utilizando o tipo *Singleton*, o ciclo de vida de um serviço é criado apenas na primeira vez em que é solicitado. Ou seja, a instância do serviço utilizada é sempre a mesma enquanto a aplicação estiver ativa, independentemente do número de requisições HTTP efetuadas. Um exemplo de injeção de um serviço onde se aplicou este tipo de ciclo de vida durante o estágio foi num serviço de cache, desenvolvido por outro colega de equipa, para melhorar o desempenho da aplicação, ao colocar em cache alguns dados extensos que são frequentemente requisitados, de forma a não sobrecarregar a base de dados.

Outra funcionalidade comum no desenvolvimento *web*, associada também à injeção de dependências, que foi utilizada na aplicação do estágio e demonstrada no curso, foi a utilização da classe *UnitOfWork*. Esta classe reúne as implementações mais utilizadas numa única classe e interface, tornando o código mais limpo e legível, além de descentralizar métodos comuns entre essas implementações.

3.3 Entity Framework in depth: The Complete Guide

O curso seguinte focou-se em explicar o funcionamento do Entity Framework, o ORM mais utilizado em conjunto com a *framework* .NET. O Entity Framework é uma tecnologia que já havia sido utilizada pelo aluno tanto fora como dentro do estágio, porém este curso possibilitou a revisão de alguns conceitos em várias áreas. Assim, foi mostrado de forma

prática como implementar os dois fluxos de trabalho mais utilizados para a criação de base de dados numa aplicação, conhecidos como *database-first* e *code-first*, sendo este último o mais adotado, visto que permite um controlo de versão da base de dados mais intuitivo e fácil. Com o *code-first*, através das migrações, é possível reverter para qualquer versão da base de dados utilizando alguns comandos simples.

Também foi demonstrado o funcionamento da FluentAPI, que é uma forma programática de configurar o mapeamento entre classes e a base de dados, utilizando métodos encadeados, em vez de aplicar anotações diretamente nas propriedades das classes do modelo. Semelhante a essas anotações, a FluentAPI permite definir regras de relações e outras configurações no método *OnModelCreating*.

Outra funcionalidade interessante abordada neste curso foi o *Language Integrated Query* (LINQ), que serve para executar consultas a bases de dados em C# de forma mais coesa e intuitiva, abstraindo a complexidade do *Structured Query Language* (SQL) ou de outras linguagens específicas para manipular bases de dados. Isto permite que os desenvolvedores se concentrem na lógica, em vez desses detalhes, já que os métodos em C# são automaticamente traduzidos para essas linguagens especificas, conforme exemplificado no esquema da Figura 12.

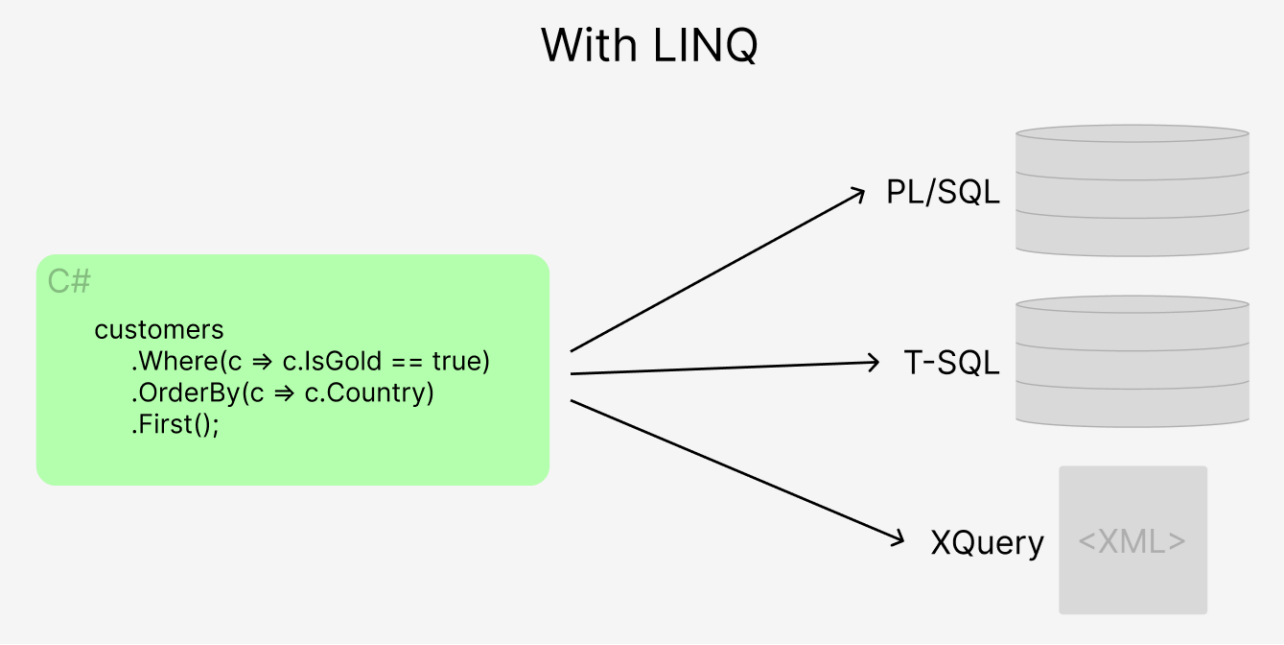

Figura 12 Exemplo de funcionamento de uma query, através do LINQ.

No final do curso, foi ainda explicado o funciona o *repository pattern*, que é um padrão de projeto (ou *design pattern)* na engenharia de software, que divide a lógica de acesso aos dados da lógica de negócio, criando um nível de abstração entre algumas camadas, com o

intuito de tornar a aplicação mais modular. Durante o desenvolvimento da aplicação do estágio, também se utilizou este padrão de projeto, o que representou o primeiro contacto com o *repository pattern*.

Já tinha havido algum estudo prévio sobre deste tipo de padrão durante a fase inicial de integração na equipa, devido à necessidade de analisar o funcionamento da aplicação. No entanto, esta explicação foi de grande utilidade, especialmente por permitir uma compreensão mais consolidada de algumas vantagens, como a minimização da duplicação de lógica para efetuar pesquisas. Por exemplo, se em diferentes partes no código fosse necessário obter uma lista de objetos com base num determinado parâmetro, sem a utilização do *repository pattern* haveria duplicação desta lógica. Na maioria dos casos, é preferível encapsular a lógica dessas pesquisas num repositório e simplesmente chamar um método geral.

Outro benefício é o desacoplamento da aplicação de *frameworks* de persistência, como o Entity Framework, o que facilita a troca para outra *framework* com o mínimo impacto no resto da aplicação, como ilustrado na Figura 13.

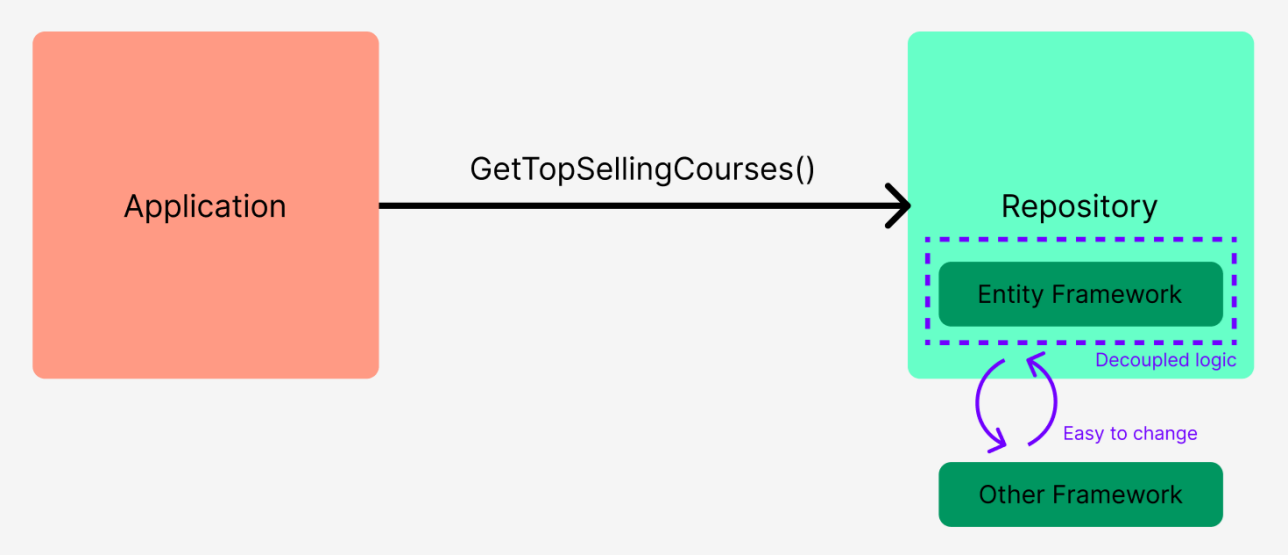

Figura 13 Exemplo de lógica desacoplada, com um método geral GetTopSellingCourses, através do repository pattern.

3.4 Microsoft SQL for beginners

O próximo curso deste período de formação, tal como o nome indica, é destinado a iniciantes em SQL. Apesar de o aluno já ter alguma experiência nesta linguagem de programação, não houve nenhum tema particularmente novo. No entanto, o curso serviu para a revisão de alguns comandos que não costumam ser utilizados com tanta frequência.

O comando *inner join* permite retornar colunas de diferentes tabelas numa única pesquisa, desde que exista uma condição em comum entre elas. Neste caso, apenas os registos que têm correspondência em ambas as tabelas são retornados. O *inner join* compara as

colunas especificadas numa condição de junção, geralmente através de uma relação entra a chave primária de uma tabela e uma chave estrangeira noutra. O resultado dessa pesquisa contém os registos onde a condição de junção é verdadeira, ao contrário do que acontece nos casos em que não há correspondência.

Um exemplo deste comando está representada na Figura 14, onde são apresentadas duas tabelas, *Students* e *Enrollments*. Um *inner join* é efetuado para analisar em que cadeiras os estudantes estão matriculados. A condição utilizada é que o campo *StudentID* de uma tabela deve corresponder ao mesmo valor no campo correspondente da outra tabela. O estudante com o valor de *StudentID* igual a "4" fica fora do resultado, visto que não cumpre esta condição.

É importante destacar que, além do *inner join*, existem outros dois comandos de *join* semelhantes, chamados *left join* e *right join*. A principal diferença é que o *left join* retorna todos os registos da tabela à esquerda, incluindo aqueles que não têm correspondência na tabela da direita, resultando em valores nulos nessas colunas. De maneira semelhante, o *right join* retorna todos os registos da tabela à direita, incluindo aqueles que não têm correspondência na tabela da esquerda, onde os valores também resultam em nulo.

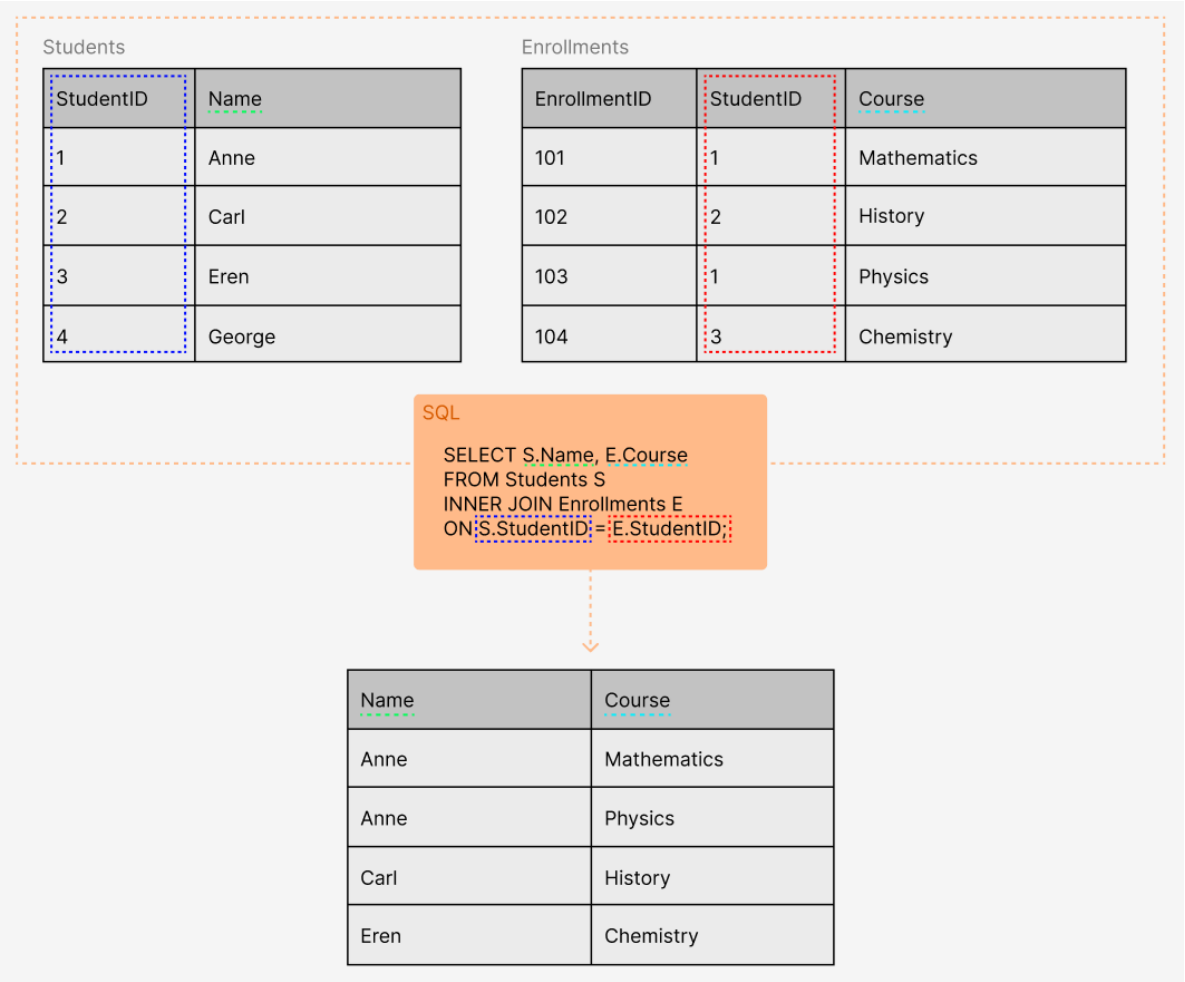

Figura 14 Exemplo de implementação de um inner join.

3.5 Responsive Web Design Essentials – HTML5 CSS3 Bootstrap

O curso seguinte sobre a *framework* Bootstrap, amplamente utilizada no desenvolvimento do *frontend* em aplicações *web*. Já existia alguma familiarização com esta tecnologia por parte do aluno, adquirida tanto em contexto profissional e académico, como em projetos pessoais.

Este curso serviu como revisão de algumas funcionalidades, como o sistema de grelha com 12 colunas, que permite a criação de *layouts* responsivos, ajustando automaticamente o conteúdo do site a diferentes tamanhos de ecrã, apenas através da utilização de classes nativas da *framework*. Na Figura 15, é possível observar essa funcionalidade: num ecrã de tamanho médio, o terceiro *div* ocupará quatro colunas, devido à classe "col-md-4". Da mesma forma, para ecrãs pequenos, o *div* ocupará seis colunas por causa da classe "col-sm-6". Finalmente, o *div* ocupará doze colunas em ecrãs com tamanhos menores que os abrangidos por essas classes.

```
1  <div class="container">
2      <div class="row">
3          <div class="col-md-4 col-sm-6 col-12">
4              <div class="box">Content</div>
5          </div>
6      </div>
7  </div>
8
```

Figura 15 Exemplo de implementação de classes responsivas do Bootstrap.

Além das classes mencionadas, o Bootstrap oferece uma vasta biblioteca de outros componentes que agilizam o processo de desenvolvimento de interfaces, através de classes nativas como botões, menus de navegação, formulários, carrosséis, *modals*, tabelas, entre outros.

3.6 Dapper – Getting Started

O próximo curso, tal como o nome indica, é destinado a iniciantes no Dapper, o micro ORM mais popular na comunidade .NET, desenvolvido pelo StackOverflow. Antes deste curso, o aluno nunca tinha tido qualquer contacto com o Dapper, tendo utilizado sempre o Entity Framework em projetos .NET. Este curso foi bastante útil para compreender os conceitos de micro ORMs.

Um micro ORM foca-se em realizar o mapeamento entre a base de dados e os objetos de uma aplicação de forma eficiente, embora alguns possam executar mais tarefas além desse

mapeamento. O curso explicou algumas diferenças entre um micro ORM, como o Dapper, e um ORM, como o Entity Framework. Entre as principais vantagens dos micro ORMs está o melhor desempenho, pois o foco é o mapeamento entre os dados e o código, gastando menos tempo em outras tarefas. Outra vantagem é que, para projetos mais simples, com poucas tabelas na base de dados, uma *framework* completa pode ser desnecessária. Os micro ORMs deixam o código mais simples e fácil de manter, especialmente se a aplicação utilizar muitos procedimentos armazenados, já que a lógica permanece na base de dados, não sendo necessário acoplar toda a lógica a um ORM tradicional. Além disso, os micro ORMs funcionam com qualquer base de dados, permitindo que o utilizador escreva as próprias pesquisas em SQL, o que é uma grande vantagem para quem tem experiência nesse tipo de escrita.

Entre as principais desvantagens, a primeira está relacionada à última vantagem referida: escrever as próprias pesquisas em SQL pode ser problemático se o utilizador não tiver muita experiência, podendo tornar as pesquisas de um ORM tradicional mais eficientes. Outra desvantagem é que trabalhar com relações entre tabelas pode ser mais complexo em micro ORMs, pois essas relações precisam ser estruturadas de forma específica. Finalmente, se uma aplicação necessitar de validações extensas, os ORMs tradicionais costumam ter métodos dedicados a esse tipo de funcionalidade, enquanto os micro ORMs se focam no mapeamento e desempenho, em vez dessas implementações.

Um exemplo prático de implementação com o Dapper pode ser observado na Figura 16, onde é mostrada a forma de escrever e aplicar *scripts* em SQL. Neste caso, para obter um registo de uma tabela da base de dados, passa-se um *id* por parâmetro. Primeiramente define-se uma *string* com o comando desejado, onde o parâmetro é especificado com um "@" antes do nome. Em seguida, o comando é executado com o método *Query*, no qual se específica o tipo de objeto que será retornado, neste caso, um objeto *Company*. No primeiro parâmetro, coloca-se a *string* com o código SQL e, no próximo, um objeto com os parâmetros utilizados na *string* e o respetivo valor.

```
1   public CompanyRepository (IConfiguration configuration){
2       this.db = new SqlConnection(configuration.GetConnectionString("DefaultConnection"));
3   }
4
5   public Company Find(int id)
6   {
7       var sql = "SELECT * FROM Companies WHERE CompanyId = @CompanyId";
8
9       return db.Query<Company>(sql, new { @CompanyId = id }).Single();
10  }
11
```

Figura 16 Exemplo da utilização de um comando SQL, através do Dapper.

3.7 React Native – The Practical Guide

Outro curso que permitiu aprofundar o conhecimento foi sobre a *framework* React Native, com a qual já havia algum contacto prévio pelo aluno. Esta *framework* é amplamente utilizada para o desenvolvimento do *frontend* em aplicações móveis, ao invés de aplicações *web*, como no caso do React DOM. Enquanto o React DOM é responsável por traduzir a lógica implementada no React para os navegadores, o React Native permite a construção de aplicações móveis nativas, compatíveis com Android e iOS, que podem ser distribuídas através de lojas de aplicações para ambos os tipos de dispositivos.

Para substituir a função do React DOM, o React Native oferece uma coleção de componentes em *JavaScript XML* (JSX) que são traduzidos para elementos de UI nativos, como ilustrado na Figura 17, bem como APIs em JavaScript que permitem interações com o dispositivo. O código JavaScript desenvolvido é executado numa *thread* separada, que comunica com a *thread* do dispositivo, garantindo que as interações e instruções sejam traduzidas para comandos nativos compreendidos pelo sistema operativo.

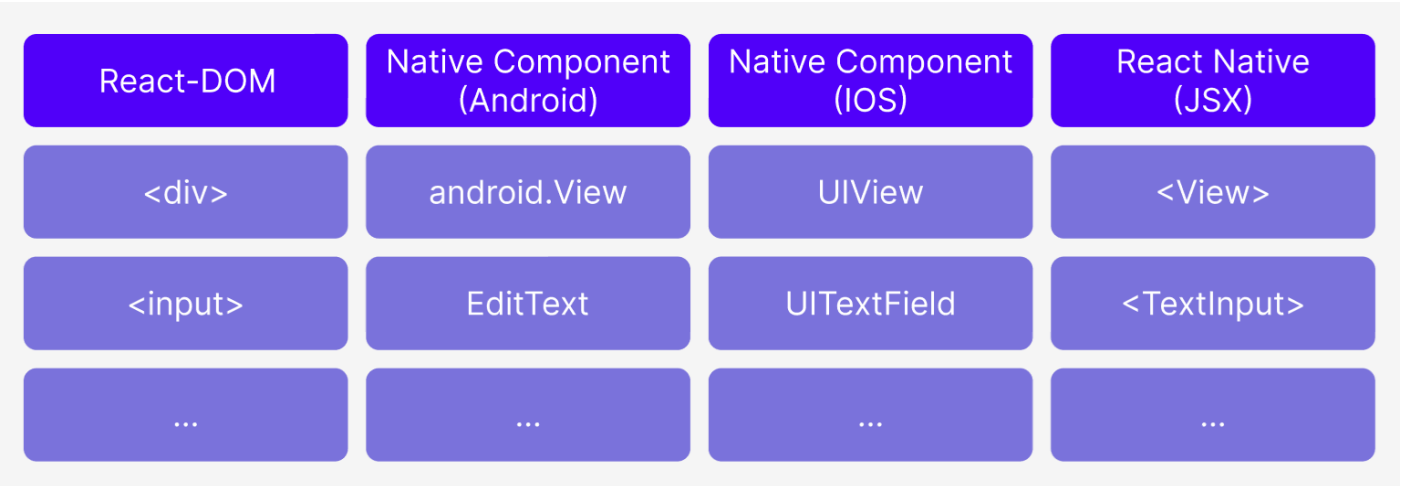

Figura 17 Exemplo de tradução de elementos, entre as diversas tecnologias.

3.8 C# Advanced Topics – Prepare for Technical Interviews

Por fim, o último curso concluído deste período aborda os tópicos avançados da linguagem de programação C#, sendo uma continuação de um dos cursos feitos anteriormente, intitulado "*C# Intermediate: Classes, Interfaces and OOP*".

O primeiro tópico abordado neste curso foi sobre *Generics*, que são um recurso para criar métodos e classes genéricas, reutilizáveis e compatíveis com qualquer tipo de objeto, sem a necessidade de efetuar *casting* desses objetos. Este conceito otimiza o desempenho da aplicação e evita a repetição de código.

Outro tópico abordado foram os *Delegates*, que representam uma referência, ou ponteiro, para uma lista de métodos. São utilizados para invocar métodos diretamente, permitindo a criação de aplicações mais extensíveis e flexíveis.

O tema seguinte foram as *Lambda Expressions*, que representam métodos sem modificador de acesso, sem nome e sem tipo de retorno explícito. A sua principal utilidade é a redução do volume de código, tornando-o mais consigo e claro, mantendo a funcionalidade.

Em seguida, foram explicados os *Events*, um mecanismo de comunicação entre objetos. Quando algo acontece num determinado objeto, este pode notificar outros objetos, facilitando a construção de aplicações com componentes e classes pouco acopladas, fáceis de estender e manter sem modificar funcionalidades existentes. Um exemplo prático seria a definição de uma classe como publicadora e outras como subscritoras, prontas para reagir a eventos gerados pela primeira, sem acoplamento direto entre essas classes.

Foram também abordados os *Extension Methods*, que permitem adicionar novos métodos a tipos já existentes, sem a criação de subclasses. Estes métodos são definidos como métodos estáticos, sendo que o primeiro parâmetro usa a palavra-chave *this* para representar a variável do tipo que se pretende estender. Estes métodos são invocados como se fizessem parte do tipo original.

Outro tópico foi sobre os *Dynamic Types*, que permitem criar variáveis cujo tipo é definido em tempo de execução, ao invés de em tempo de compilação. Com a palavra-chave *dynamic*, trabalha-se com objetos sem um tipo definido, facilitando a interação com APIs externas ou outras fontes de dados dinâmicos.

De seguida, foi explicado o funcionamento do *Exception Handling*, um mecanismo comumente utilizado para lidar com erros durante a execução de código. Utilizam-se as palavras-chave *try*, *catch* e, se necessário, *finally*. O bloco *try* contém o código suscetível de causar exceções, o bloco *catch* captura e trata essas exceções, e o bloco *finally* é executado independentemente da ocorrência de exceções, para libertar recursos utilizados.

Por fim, foi abordado o tema da *Asynchronous Programming*, que permite a execução de operações de forma assíncrona, sem bloquear o fluxo da aplicação enquanto se aguardam tarefas demoradas. Através das palavras-chave *async* e *await*, criam-se métodos assíncronos que libertam a *thread* principal, melhorando a responsividade das aplicações, especialmente em cenários com interfaces gráficas.

4 Projeto eDelivery

4.1 Desenvolvimento

Durante o estágio, foi realizada a participação no projeto internacional “eDelivery” de grande envergadura, que surgiu da necessidade de desenvolver uma plataforma destinada a facilitar a transferência de um determinado tipo de ficheiros entre os países membros da União Europeia. Para isso desenvolveu-se uma solução com a colaboração de outras equipas e organizações públicas que contribuíram com componentes específicos para auxiliar essa comunicação. Devido à confidencialidade de algumas partes do projeto, certos detalhes do estado da arte foram abordados de forma mais superficial.

O projeto eDelivery tem como objetivo central o desenvolvimento de uma plataforma para a troca de ficheiros, como mencionado anteriormente. Além dessa implementação, a equipa do estágio sugeriu a criação de uma consola de administração para gerir e monitorizar essas transferências. Essa consola será útil no ambiente de produção, facilitando a identificação de problemas e também durante eventos periódicos da Comissão Europeia, que analisam o estado de desenvolvimento de cada país.

Este projeto é bastante complexo, pois é necessário seguir um conjunto de normas e protocolos para a comunicação entre as diferentes entidades, como os Estados-Membros e outras organizações, seja para o envio de mensagens de pedido ou para a transferência de ficheiros entre os componentes da infraestrutura.

4.2 Funcionalidades de *FrontEnd*

Em relação ao tipo de desenvolvimento geral efetuado durante o estágio, algumas tarefas incluíram a criação de páginas de acordo com as funcionalidades necessárias, geralmente associadas a tabelas. O objetivo da consola da aplicação estava relacionado com a monitorização e gestão da solução do *backend* e da base de dados, através de uma interface gráfica.

Associadas às páginas dessa interface, existem filtros de pesquisa baseados em vários atributos vinculados às colunas dessas páginas, conforme retratado na Figura 18. Também são utilizados *modals*, que são elementos exibidos numa camada superior ao conteúdo da página, desativando esse conteúdo [46]. Estes *modals* permitem verificar os detalhes de um determinado registo de forma mais aprofundada, como ilustrado na Figura 19.

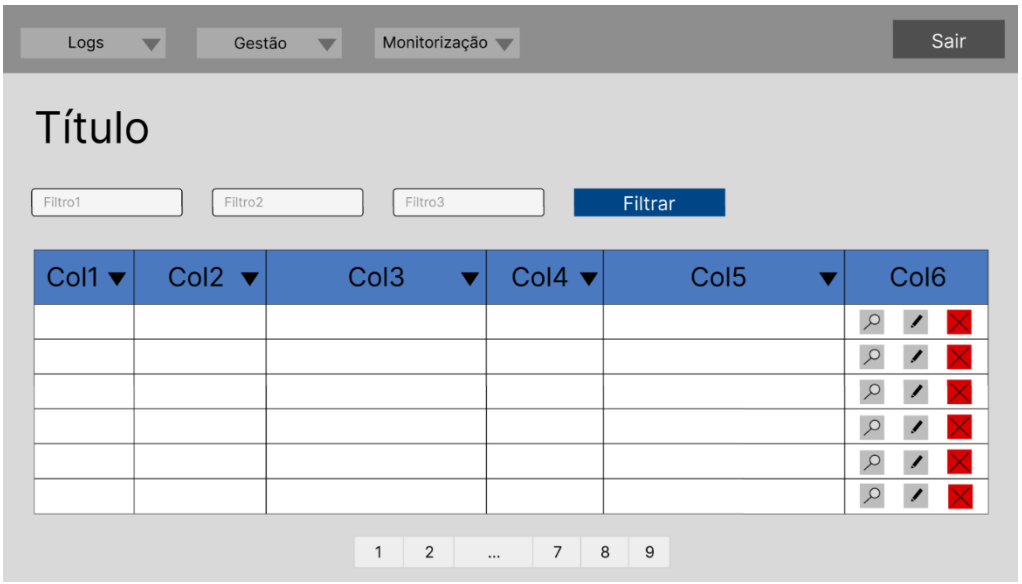

Figura 18 Exemplo de implementação de uma página da aplicação.

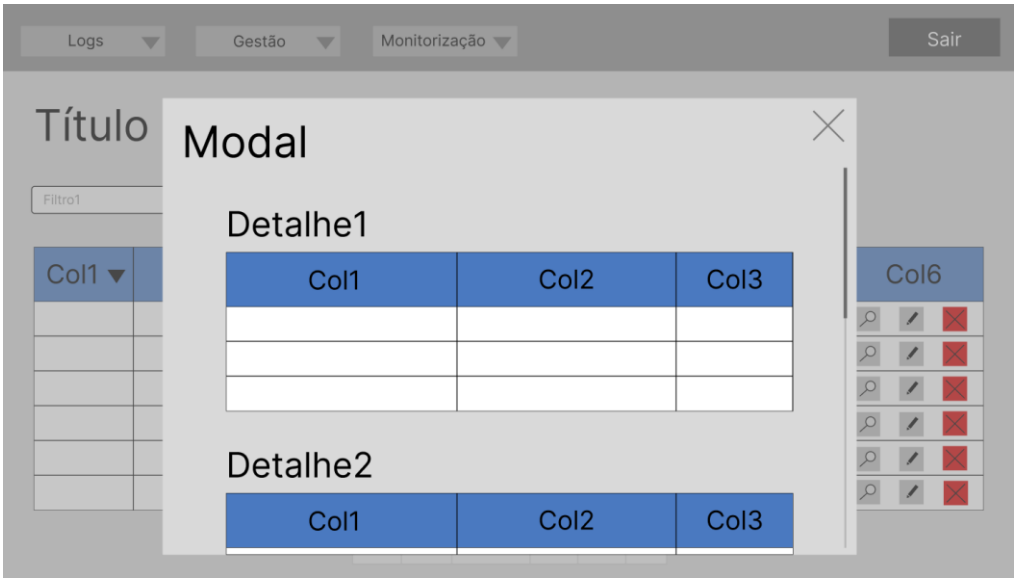

Figura 19 Exemplo de modal de detalhes de uma página da aplicação.

Em conjunto com a criação dessas páginas, vários detalhes de implementação exigidos pela aplicação foram abordados, como a criação de caminhos entre as páginas e de *labels* ou elementos que utilizam esses caminhos para o redireccionamento entre as páginas. No caso desta aplicação, esses elementos estavam organizados em diversos menus, exemplificados na Figura 20. Outro aspeto importante é a utilização de métodos de diversos serviços para consumir ou manipular dados através da API fornecida pelo *backend*.

Além disso, foram implementados métodos nas classes TypeScript dos componentes das páginas criadas, com o intuito de definir determinados comportamentos. Alguns exemplos práticos dessas funções incluem métodos que permitem ordenar cada coluna das tabelas conforme o tipo de atributo, seja por ordem numérica, alfabética ou cronológica. Outra aplicação comum era a alteração da cor de um elemento da página de acordo com o "estado de saúde" atual de cada componente do *backend* que necessita de ser monitorizado, de forma semelhante ao que se encontra na Figura 21. Por fim, cada página que inclui uma tabela possui a respetiva paginação, como é possível verificar na Figura 20 e na Figura 21.

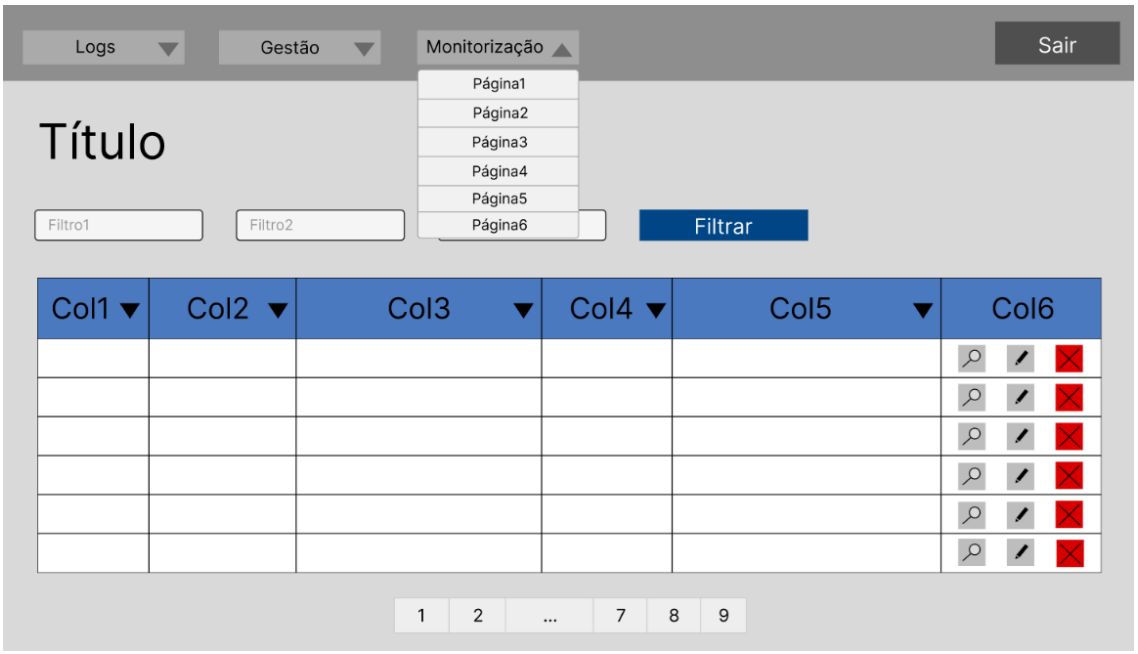

Figura 20 Exemplo de um menu com as páginas da aplicação.

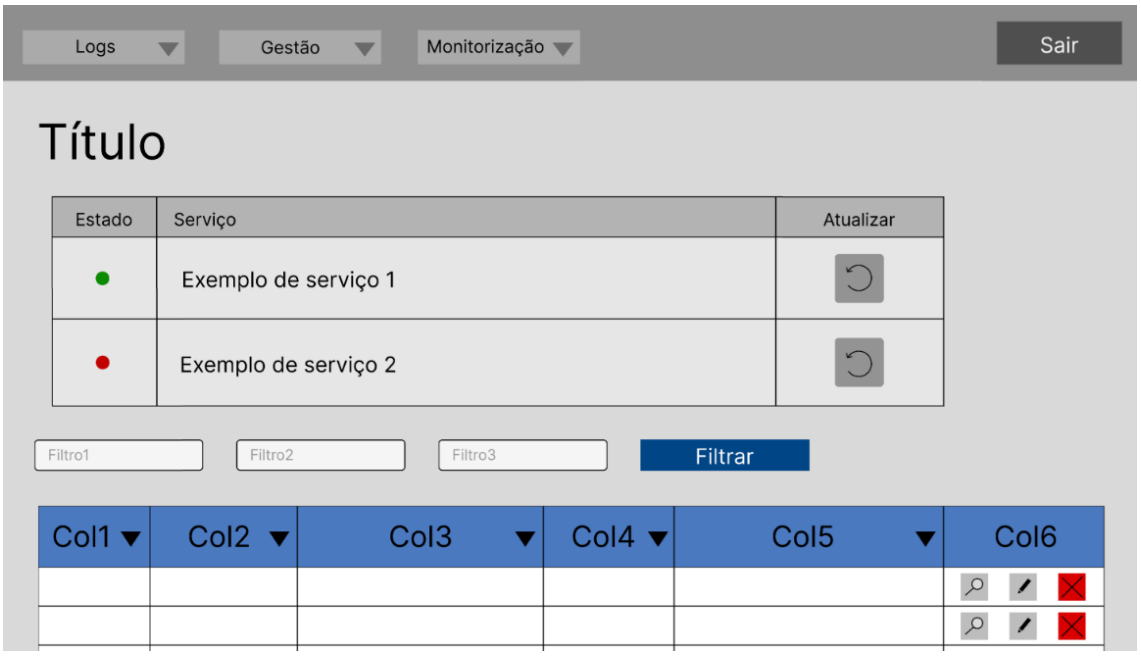

Figura 21 Exemplo de uma página com o estado de serviços.

4.3 Funcionalidades de *BackEnd*

Ainda no que diz respeito à primeira parte do estágio, onde houve um maior desenvolvimento de tarefas, as funções desenvolvidas no *backend*, estão principalmente relacionadas com a correção de certas implementações já existentes, a criação de testes unitários e regras de validação, para a validação dos ficheiros recebidos ao longo dos fluxos da aplicação, e a adição de certos detalhes que faltavam em determinados fluxos ou que podiam ser úteis para o *frontend*. Isso envolve a criação de *endpoints* e, em todas essas tarefas, a organização do código, o uso de boas práticas de programação e de bibliotecas disponibilizadas pela *framework* .NET.

Citando alguns exemplos práticos das implementações realizadas, uma das alterações mais relevantes no que toca à correção de funcionalidades diz respeito a uma mudança na arquitetura, ou seja, à forma como o sistema de outra equipa entrega e como o nosso sistema recolhe os ficheiros num fluxo em que Portugal atua como fornecedor.

Antes desta alteração, como exemplificado na Figura 22, a informação dos ficheiros era enviada diretamente pela entidade da outra equipa, representada como *File Provider*, através de um *handler* de uma API disponibilizada por essa entidade. No entanto, o tamanho de alguns ficheiros nessa transferência podia chegar a cerca de 1 *Megabyte* (MB), especialmente no caso de ficheiros do tipo PDF, o que podia causar atrasos nas comunicações. Para prevenir esse comportamento, foi adicionada uma nova máquina à arquitetura do sistema, representada como *Files Handler* na Figura 23.

Com esta nova arquitetura, a entidade *File Provider* passa a enviar os ficheiros, que são armazenados numa base de dados dessa máquina. Em seguida, o *File Provider* envia apenas a informação necessária para o sistema, representado como *Application*, de modo a que seja possível identificar e recolher o respetivo ficheiro através de um novo *handler* no servidor *Files Handler*. Desta forma, estabelece-se um servidor dedicado ao carregamento, manutenção e transferência dos ficheiros, permitindo reduzir a necessidade de recursos contínuos e a pressão na comunicação direta entre as duas máquinas originais, além de disponibilizar a comunicação de forma assíncrona e aumentar escalabilidade da aplicação.

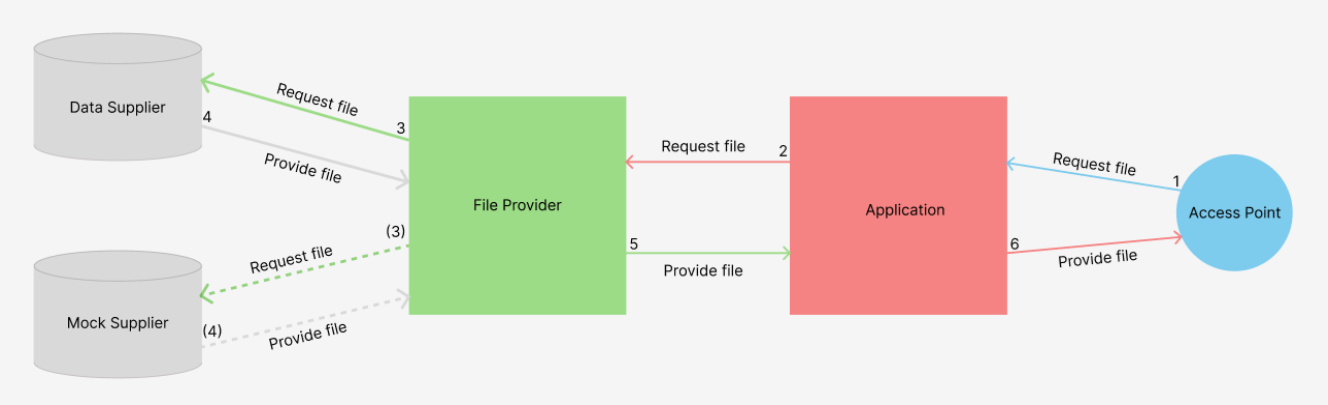

Figura 22 Exemplo de fluxo como fornecedor, sem preview, antes da alteração.

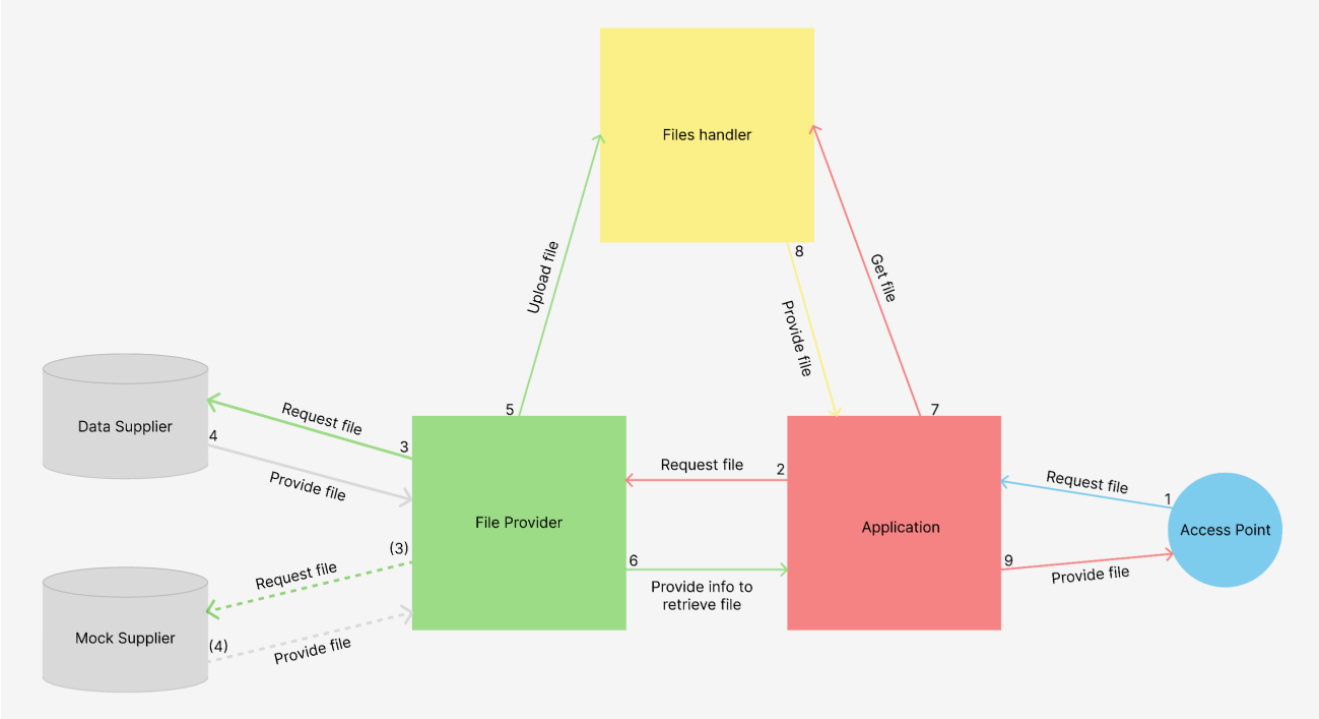

Figura 23 Exemplo de fluxo como fornecedor, sem preview, depois da alteração.

4.4 Regras de validação

As regras de validação foram implementados através da biblioteca Fluent Validation, disponível no .NET, a qual facilita a criação dessas regras. No âmbito deste projeto, esta biblioteca foi utilizada para a criar regras de validação que verificam a conformidade de certos ficheiros, baseando-se na estrutura e conteúdo de ficheiros *Extensible Markup Language* (XML), de modo a assegurar que as mensagens trocadas em determinados fluxos seguem o tipo de solução comum acordada entre os países membros envolvidos. Na Figura 24, pode ser visualizado um exemplo ilustrativo de um ficheiro XML e na Figura 25 uma regra de

validação para demonstrar, de forma simplificada, como as regras foram aplicadas aos ficheiros.

XMLSimples.xml

C: > Users > rafab > Desktop > Mestrado > XMLSimples.xml

```xml
<note exemplo="namespaceExemplo">
    <exemplo:to>Person1</to>
    <exemplo:from>Person2</from>
    <exemplo:heading>Reminder</heading>
    <exemplo:body>SomeInfo</body>
</note>

```

Figura 24 Exemplo de ficheiro XML simples.

```
public void Rule001()
{
    RuleFor(x => x)
    .Must((x, y, context) =>
    {
        return x.GetElement("namespaceExemplo", "to").Value == "Person1";
    })
    .WithErrorCode("Rule001")
    .WithMessage("O valor do elemento 'to' do XMLSimples deve ser igual a 'Person1'")
    .WithSeverity(Severity.Error);
}

```

Figura 25 Exemplo de regra de validação simples.

Adicionalmente, foram criados *handlers*, que representam funções responsáveis por tratar determinados eventos, pedidos ou ações, associados, neste caso, a pedidos HTTP processados pelo *backend* [47]. Esses *handlers* permitiram ao *frontend* realizar determinadas operações, tais como as operações *Create*, *Read*, *Update* e *Delete* (CRUD) de elementos nas tabelas das páginas, a transferência de *logs* em ficheiros de texto com informações sobre possíveis exceções na troca de fluxos e falhas nas regras de validação em fluxos específicos, além de sistemas de monitorização do estado de determinados serviços. Também foram desenvolvidos *handlers* para validar dados de registo de utilizadores, verificando, por exemplo, se a palavra-passe cumpre os parâmetros estabelecidos ou se o endereço de email já existe na aplicação.

Relativamente ao processo de análise do projeto, durante o desenvolvimento das tarefas mencionadas, houve aprendizagens e consolidações importantes em áreas relacionadas com a programação orientada a objetos e a organização da arquitetura da aplicação. Um exemplo foi a injeção de dependências, um conceito abordado pelo aluno durante a licenciatura, mas que foi aprofundado durante o estágio, sendo recorrente na criação e utilização de classes no sistema. A injeção de dependências é um princípio da programação orientada a objetos que consiste em fornecer as dependências necessárias a um objeto

externamente, em vez de o próprio objeto as criar. Existem várias formas de realizar a injeção de dependências, como por exemplo através de construtores, métodos ou propriedades, com o objetivo de reduzir o acoplamento entre classes [48handl].

Este conceito apresenta várias vantagens em comparação com a criação convencional de uma classe para cada implementação, tais como:

- Facilitar testes unitários: A injeção de dependências torna mais simples a execução de testes unitários, permitindo que as classes sejam facilmente isoladas. Como as dependências são injetadas externamente, é possível substituí-las por implementações de teste durante a execução dos mesmos.
- Facilitar a manutenção do código: Ao utilizar a injeção de dependências, torna-se mais fácil substituir dependências em função de alterações nos requisitos. Por exemplo, se for necessário alterar a implementação de um serviço de notificações, que anteriormente funcionava por email, para uma nova implementação via mensagens, isso pode ser feito sem modificar a classe original ou as classes que utilizam esse serviço.

Na Figura 26, está representada uma arquitetura simples, sem a utilização da injeção de dependências, onde a classe consumidora *(Consumer Class*) está fortemente acoplada à classe que fornece a implementação necessária *(Service Implementation 2)*. Neste cenário, caso a implementação sofra alterações, será necessário modificar toda a lógica existente, incluindo os métodos e inicializações na classe consumidora.

Por outro lado, na Figura 27, observa-se o mesmo exemplo com a aplicação da injeção de dependências. Nesta arquitetura, em vez de existir um acoplamento direto entre a classe consumidora *(Consumer Class)* e a implementação específica, uma classe intermediária *(Injector)* é responsável por injetar as dependências na classe consumidora *(Costumer Class)*, indicando a implementação a ser utilizada. Assim, a classe consumidora *(Costumer Class)* apenas necessita de acesso à interface da implementação. Em caso de alterações na lógica ou adição de novas implementações, a reformulação é facilitada, bastando injetar as novas dependências na classe intermediária *(Injector)*.

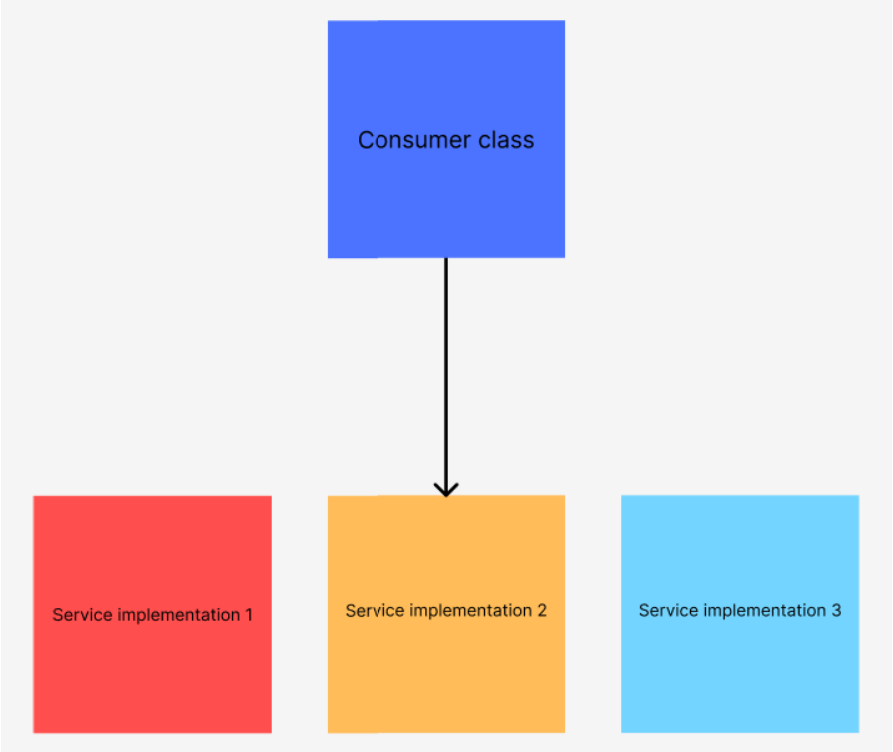

Figura 26 Exemplo de arquitetura sem a utilização de injeção de dependências.

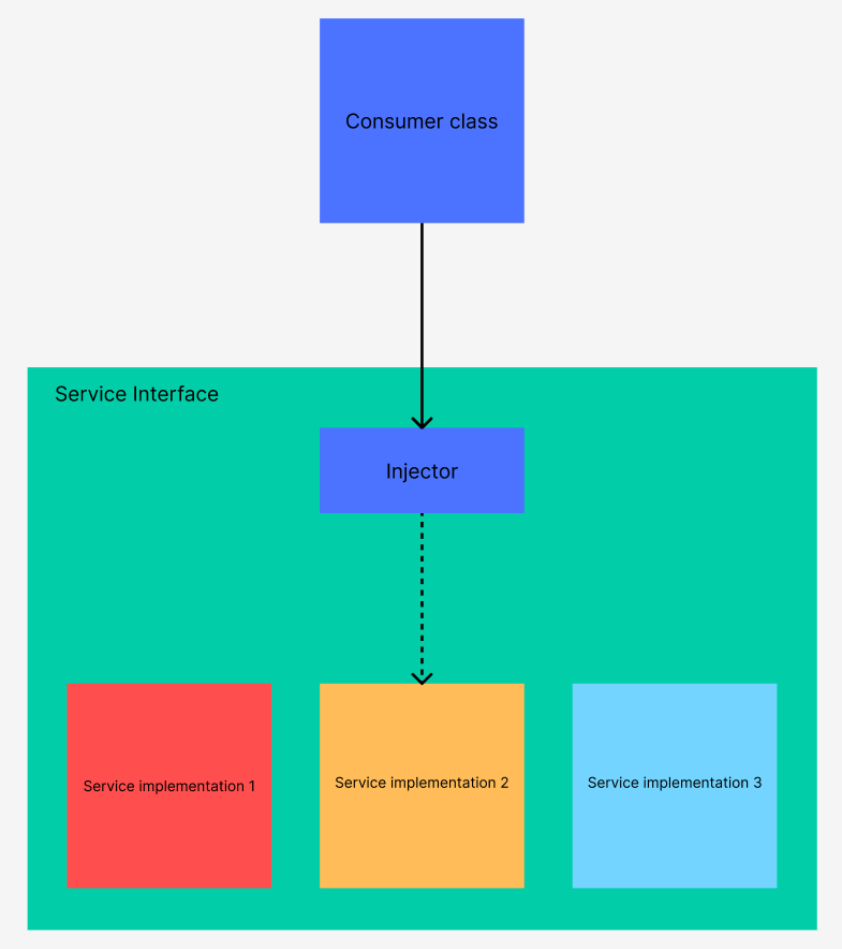

Figura 27 Exemplo de arquitetura com a utilização de injeção de dependências.

4.5 Arquitetura da aplicação

A arquitetura da aplicação é consideravelmente complexa, visto que depende da transferência de informação entre diversas entidades, como se pode observar na Figura 28. Considerando a necessidade de manter a privacidade de algumas informações e o isolamento na forma como as diferentes entidades estão estruturadas, esta figura apresenta uma simplificação da arquitetura implementada.

Algumas entidades desta arquitetura foram, portanto, divididas entre duas camadas diferentes:

- *App Layer*: Inclui as entidades com as quais a *Application* (que representa a aplicação desenvolvida neste estágio) comunica diretamente e que pertencem unicamente à solução portuguesa:
- *Database Layer:* Nesta situação, apenas possui a base de dados da *Application*, uma vez que não há acesso à estrutura das restantes entidades representadas.

Fora dessas duas camadas, estão representadas as restantes entidades, que também participam no fluxo de troca de ficheiros. No entanto, elas não comunicam diretamente com a *Application*, como o caso do *Data Supplier*, ou do *Files Handler*, ou não pertencem exclusivamente à solução nacional, como é o caso do *Info Service*.

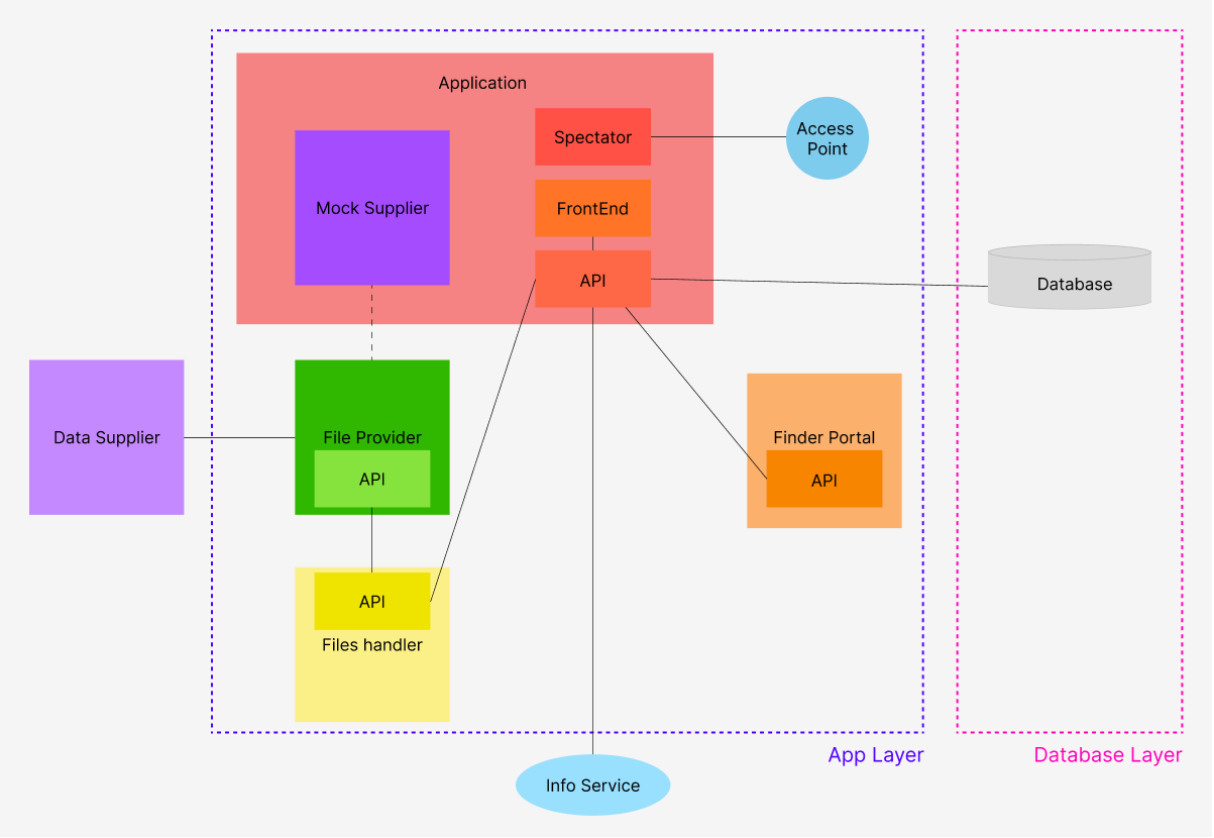

Figura 28 Arquitetura detalhada do sistema.

Passando a descrever cada componente desta arquitetura, podemos observar as seguintes máquinas e as suas funções:

4.5.1 Application

A *Application* é a solução desenvolvida ao longo deste estágio, sendo composta por diversas componentes internas, que desempenham funções cruciais dentro da arquitetura. Entre as principais estão:

4.5.1.1 Spectator

O *Spectator* é um serviço automatizado que atua de forma cíclica. Este, monitoriza o *Access Point* (AP) para verificar a existência de novas mensagens. Quando uma mensagem é recebida, este serviço recolhe a informação e armazena-a na base de dados, além de notificar o sistema para que o processamento dessa mensagem seja iniciado.

4.5.1.2 FrontEnd

O *FrontEnd* representa a interface de administração da aplicação, permitindo a interação do utilizador com o sistema. Este utiliza os *handlers* da API para implementar funcionalidades administrativas, como a manipulação e gestão dos dados na base de dados, facilitando o controlo das operações internas da aplicação.

4.5.1.3 API

A API é uma componente central, utilizada por diversas entidades dentro da arquitetura da aplicação. É responsável por processar, enviar e recolher mensagens nos fluxos de troca de documentos, além de gerir a base de dados. A API também interage com entidades externas, como o *Finder Portal*, para criar *links* de pré-visualização de documentos, ou com o *File Provider* e o *Files handler*, para enviar e solicitar ficheiros respetivamente. Deste modo, é essencial para a comunicação e o fluxo de dados entre os diferentes componentes.

4.5.1.4 MockSupplier

O *MockSupplier* foi criado para simular a função do *DataSupplier*, facilitando os testes durante o desenvolvimento da aplicação. Em cenários onde o sistema de Portugal atua como fornecedor de documentos, o *MockSupplier* fornece dados simulados ao *File Provider*, que posteriormente os envia à *Application*. Assim, ele permite testar fluxos que envolvem o fornecimento de ficheiros, mesmo sem uma

entidade real disponível para fornecer esses documentos. O *MockSupplier* armazena os ficheiros *mock* na base de dados local da *Application*, oferecendo *handlers* para a sua recolha

4.5.2 Access Point

O *Access Point* é responsável por mediar a troca de mensagens entre soluções de diferentes países membros. Este facilita a comunicação ao receber e enviar pedidos de fornecimento de evidências entre as partes envolvidas. São utilizados dois *Access Points* num fluxo de troca de mensagens, localizados nas soluções de cada país, para garantir o envio e receção de mensagens de forma eficiente e segura.

4.5.3 Finder Portal

O *Finder Portal* é o portal onde os utilizadores podem solicitar ficheiros. Comunica diretamente com a *Application* através de uma API, facilitando a interação do utilizador com o sistema.

4.5.3.1 API

A API do *Finder Portal* é utilizada para gerar *links* de pré-visualização de documentos. Quando a *Application* recebe um pedido de visualização, envia uma solicitação para esta API, que cria um *link* de pré-visualização e o devolve à *Application*.

4.5.4 Info Service

O *Info Service* disponibiliza uma API acessível por todos os países membros do sistema. Esta API permite obter informações detalhadas sobre os tipos de ficheiros disponíveis, as entidades fornecedoras, entre outros dados relevantes para os fluxos de troca de documentos. A *Application* pode solicitar estas informações e disponibilizá-las ao *Finder Portal*, garantindo que o utilizador tenha acesso aos detalhes necessários para efetuar os seus pedidos.

4.5.5 File Provider

O *File Provider* desempenha uma função central na arquitetura, sendo responsável por solicitar evidências tanto ao *Data Supplier* quanto ao *Mock Supplier*. Esta assegura que o sistema funcione corretamente quando Portugal atua como fornecedor de documentos e é uma componente central que comunica com diversas máquinas diferentes, inclusive com a *Application*, através de uma API disponibilizada à equipa de desenvolvimento.

4.5.5.1 API

A API do *File Provider* é utilizada pela *Application* para solicitar os pedidos de evidências de outros países membros. Esta fornece os *handlers* necessários para direcionar os pedidos aos *Suppliers* e efetua o carregamento das evidências ao *Files Handlers*, através da comunicação com a API da mesma.

4.5.6 Files Handler

O *Files Handler* é responsável por armazenar temporariamente os ficheiros fornecidos pelo *Data Supplier*. Após ser feito um pedido, os ficheiros são mantidos nesta máquina até que sejam recolhidos pela *Application*, através de um *handler*. A inclusão do *Files Handler* no fluxo visa otimizar o processo de transferência de ficheiros, evitando sobrecarga de memória, que anteriormente ocorria quando os documentos eram transferidos diretamente entre as entidades. Com esta máquina intermediária, a gestão de recursos é mais eficiente e o sistema torna-se mais escalável.

4.5.6.1 API

A API do *Files Handler* é composta por apenas dois *handlers*, um de carregamento, para o qual a API do File Provider envia os ficheiros obtidos dos *Suppliers* e um de transferência que é consumido pela API da *Application* para recolher esses ficheiros, através de alguns identificadores enviados por parâmero.

4.5.7 Database

A *Database* é a única entidade presente na *Database Layer* da arquitetura. Durante o estágio, foi possível compreender apenas a estrutura interna da base de dados utilizada pela *Application*. Esta base de dados foi crucial para o desenvolvimento dos fluxos de transferência de ficheiros, servindo como armazenamento temporário de documentos enquanto aguardam pré-visualização e também para registar todas as mensagens enviadas e recebidas. Além disso, a base de dados suporta funcionalidades do *FrontEnd*, como o sistema de *login* de utilizadores e a exibição de dados administrativos.

4.6 Fluxos

O sistema contempla diversos fluxos de transferência de ficheiros, visto que cada país implementa a sua própria solução, podendo atuar como requisitante ou fornecedor de ficheiros. Para ambas as funções, existem ainda duas modalidades: com ou sem pré-visualização de evidências.

Na secção seguinte serão descritos os detalhes sobre o funcionamento dos fluxos.

4.6.1 Fluxo como fornecedor sem pré-visualização

Neste fluxo, a solução nacional funciona como fornecedora, ou seja, é responsável por recolher e entregar os ficheiros solicitados pelo país membro requisitante. Este processo envolve várias componentes, como ilustrado na Figura 29. Em primeiro, o país requisitante *(Requester)* envia uma mensagem de solicitação de ficheiros para o *Access Point* da aplicação.

De seguida, o *Spectator* monitoriza as mensagens pendentes no AP e a solução implementada *(Application)* obtém essa mensagem, processando as informações contidas nela.

A *Application* então gera uma nova mensagem de solicitação de ficheiros e envia-a à entidade *File Provider*. Esta, ao receber a mensagem, identifica o fornecedor de dados apropriado, processa a mensagem recebida e formula um novo pedido de ficheiros para o fornecedor relevante.

O fornecedor de dados (*DataSupplier*), por sua vez, envia o ficheiro para o *File Provider*, que, através de um *handler*, transfere o ficheiro e informações associadas para a entidade *Files Handler*, onde os ficheiros são armazenados temporariamente, e envia uma mensagem com informações sobre esses ficheiros à *Application*. Posteriormente, a *Application* envia uma mensagem à entidade *Files Handler*, com os detalhes dos ficheiros carregados, permitindo recuperar os ficheiros solicitados.

Após obter os ficheiros, a *Application* gera uma nova mensagem com os

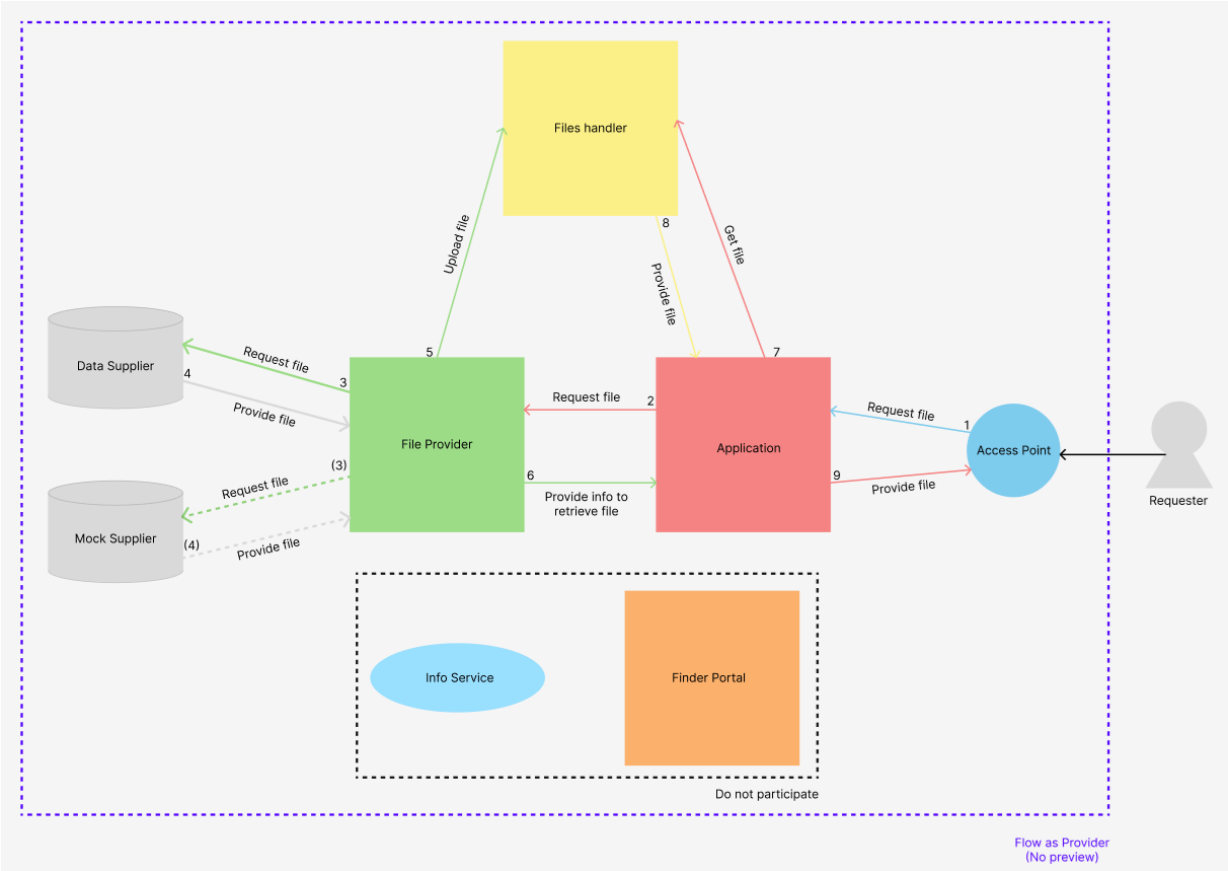

Figura 29 Fluxo de troca de ficheiros como fornecedor, sem pré-visualização.

ficheiros tratados, que é enviada para o *Access Point*. Por fim, este reencaminha a mensagem para o AP do sistema do país requisitante.

4.6.2 Fluxo como fornecedor com pré-visualização

Este fluxo é semelhante ao anterior, com a diferença de incluir a necessidade de disponibilizar um espaço ou página para que o requisitante possa pré-visualizar o documento antes da sua obtenção. Conforme ilustrado na Figura 30, o processo inicia-se com o envio, por parte do requisitante, de uma mensagem para solicitar um espaço de pré-visualização ao AP.

A *Application* processa a mensagem e formula um novo pedido, enviando-o à entidade *File Provider*, que processa o pedido e solicita os ficheiros ao fornecedor de dados *(Data Supplier)*. Estes ficheiros são depois enviados ao *File Provider*, que os transfere para o *Files Handler* e informa a *Application*, enviando informações sobre os ficheiros carregados. A *Application* recolhe os ficheiros e armazena-os temporariamente na base de dados, antes de solicitar um *link* para a pré-visualização desses documentos.

.O *Finder Portal* cria o *link* de pré-visualização e envia-o para a *Application*, que o reencaminha para o requisitante através do *Access Point*. Após a pré-visualização, o requisitante decide se pretende ou não obter os ficheiros. Se a resposta for afirmativa, a *Application* envia os ficheiros previamente armazenados para o AP, que os reencaminha para o requisitante. Caso a resposta seja negativa, a *Application* elimina os dados dos ficheiros da base de dados. Se não houver resposta, os ficheiros são mantidos durante um período definido, ao fim do qual são eliminados.

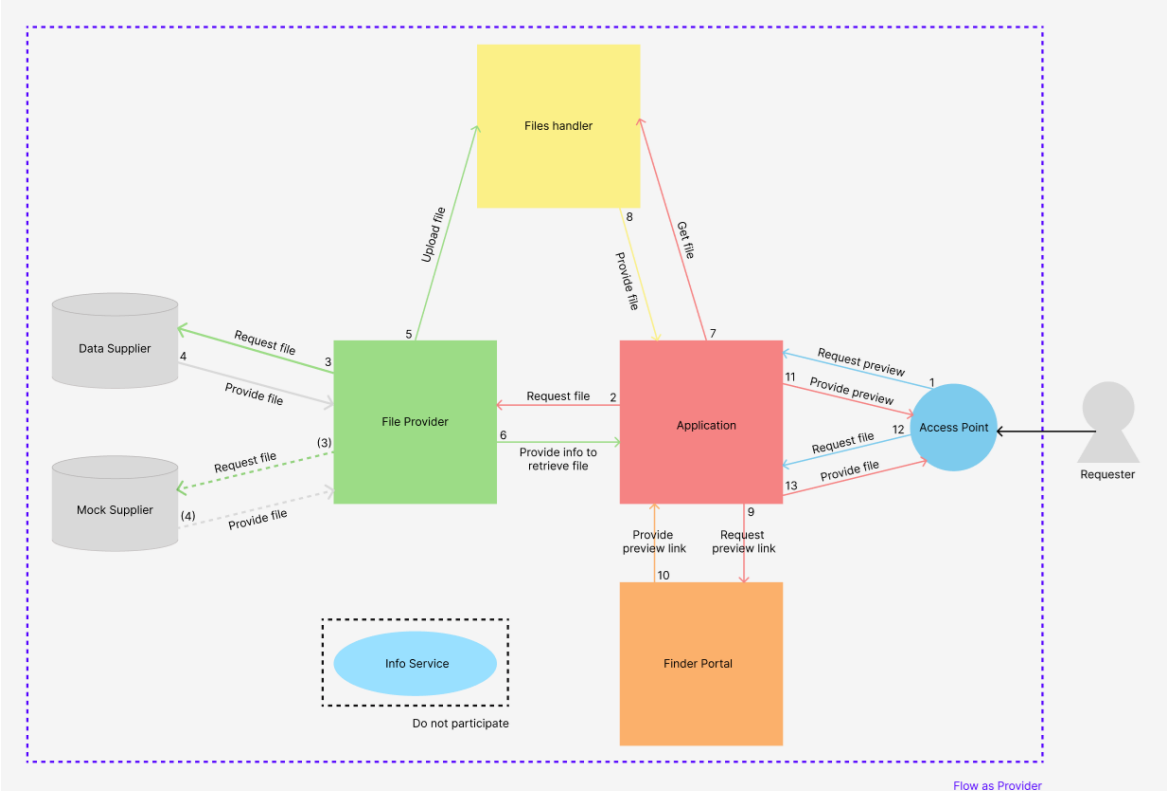

Figura 30 Fluxo de troca de ficheiros como fornecedor, com pré-visualização.

4.6.3 Fluxo como solicitante sem pré-visualização

Neste cenário a *Application* atua como requisitante de evidências, sendo outro país o fornecedor de ficheiros, sem opção de pré-visualização (como o fluxo da Figura 29).

Como descrito na Figura 31, o utilizador começa por aceder ao *Finder Portal* para identificar os documentos que deseja solicitar. A *Application* processa o pedido e solicita ao *Info Services* os dados necessários sobre os documentos e fornecedores disponíveis.

Após receber a resposta do *Info Service*, a *Application* processa a informação e apresenta-a ao utilizador no *Finder Portal*. O utilizador seleciona os documentos e o fornecedor pretendidos, e a *Application* envia o pedido ao *Access Point*, que o transmite para o sistema do país fornecedor.

O país fornecedor processa o pedido e envia os documentos solicitados para a *Application*, que os disponibiliza no *Finder Portal*, permitindo ao utilizador aceder a esses documentos.

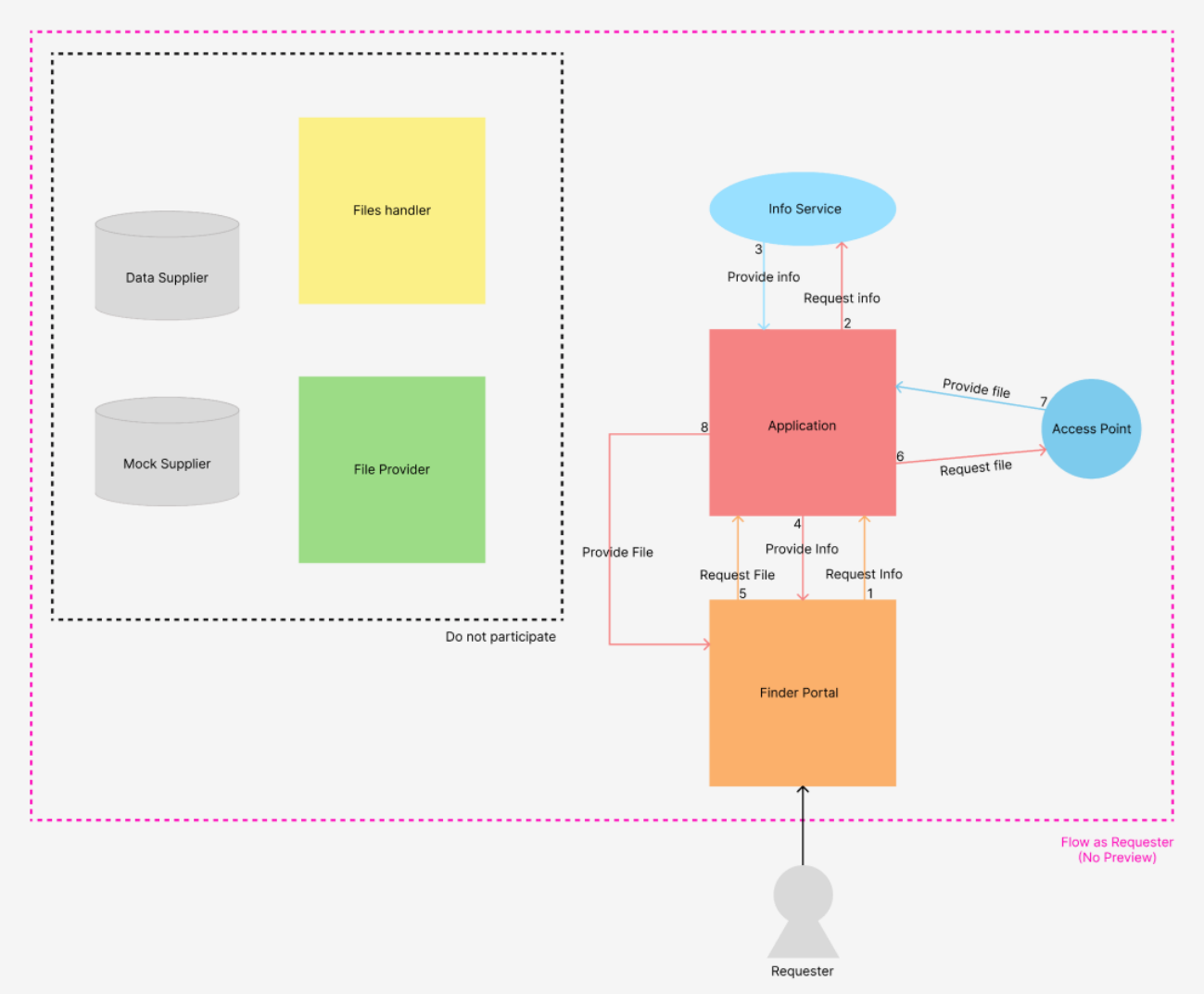

Figura 31 Fluxo de troca de ficheiros como solicitante, sem pré-visualização.

4.6.4 Fluxo como solicitante com pré-visualização

Neste fluxo, semelhante ao anterior, a *Application* também atua como requisitante, mas com a opção de pré-visualização dos documentos antes da sua obtenção.

Conforme ilustrado na Figura 32, o utilizador começa por solicitar ao *Finder Portal* um *link* para pré-visualizar os documentos que pretende obter. A *Application* reencaminha esse pedido para o *Info Service*, que devolve os dados dos fornecedores e documentos disponíveis.

O utilizador, no *Finder Portal*, escolhe os documentos e pede o *link* de pré-visualização. A *Application* envia o pedido ao país fornecedor, que obtém os documentos e gera o *link* de pré-visualização. Este *link* é então enviado para a *Application*, que o reencaminha para o *Finder Portal*, permitindo ao utilizador pré-visualizar os documentos.

Após a pré-visualização, o utilizador pode optar por solicitar os ficheiros. Se a solicitação for confirmada, a *Application* processa o pedido e envia-o ao AP, que o redireciona para o país fornecedor. Os documentos são então transmitidos para a *Application*, que os disponibiliza no *Finder Portal* para o utilizador.

Caso o utilizador decida não solicitar os documentos ou não responda, o pedido expira após um período previamente definido pelo país fornecedor, invalidando o processo

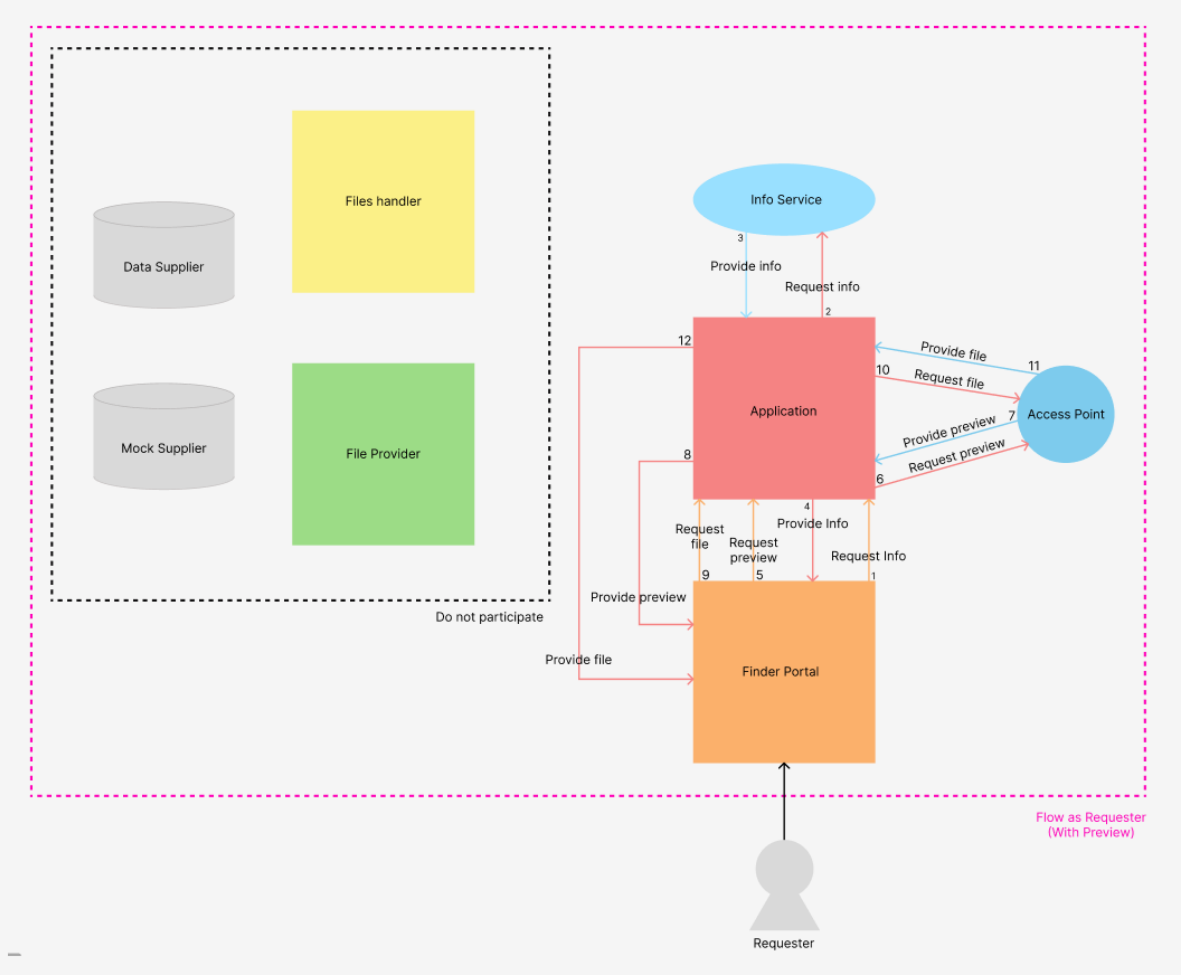

Figura 32 Fluxo de troca de ficheiros como solicitante, com pré-visualização.

5 Trabalho de desenvolvimento

No que diz respeito às atividades desenvolvidas durante o estágio, estas foram organizadas em *sprints*, que, neste caso, representam iterações de 2 semanas. Esta organização obedece à metodologia Scrum utilizada, tendo sindo realizadas tarefas nas *sprints* abordadas a seguir.

5.1 Sprint 1

Este período representa a primeira *sprint*, onde foram efetuados os primeiros contributos para o projeto. Assim, é possível observar que, de forma a facilitar a integração, foram atribuídas tarefas de componentes diferentes, com suporte dos restantes membros da equipa. Isto permitiu explorar a estrutura de cada um desses componentes e realizar as tarefas no período estipulado.

- 1ª Tarefa – Componente de *frontend*:
    - A primeira tarefa atribuída a esta *sprint* estava relacionada com a componente de *frontend*, mais especificamente à reorganização de alguns menus suspensos. Primeiramente, reformularam-se alguns menus, visto que alguns valores associados já não faziam sentido, organizando-se, portanto, esses valores devidamente. Em seguida, removeu-se um menu suspenso que se tornou obsoleto após essas alterações e, por fim, adicionou-se um ícone que herdou o caminho de uma das páginas que pertencia a esses menus.
- 2ª Tarefa – Componente de *backend*:
    - A segunda e última tarefa desta *sprint* apresentou uma maior dificuldade, devido ao seu grau de complexidade em comparação com a anterior, estando associada à componente de *backend*. Inicialmente, a realização da tarefa exigiu consultar documentação, o que trouxe a necessidade de entrar em contacto com membros de outra equipa, para esclarecer algumas dúvidas e auxiliar na compreensão de conceitos. Foram realizadas algumas reuniões online para este efeito. Uma vez compreendida a estrutura, foi desempenhada a implementação inicial da tarefa, que visava desenvolver a lógica inicial de um novo método. O objetivo era substituir, futuramente, a alternativa utilizada naquele momento. Testou-se, por fim, esse novo método, verificando que estava de acordo com o suposto.

5.2 Sprint 2

Esta *sprint* foi bastante trabalhosa. Inicialmente, começou com duas tarefas de *frontend* mais simples, dado que já existia um maior conhecimento sobre esta componente. Contudo, a terceira tarefa, que representou a primeira interação com a componente de validação, revelou-se uma tarefa árdua e repetitiva.

- 1ª Tarefa – Componente de *frontend*:
    - Relativamente à primeira tarefa desta *sprint*, criou-se uma tabela para ser preenchida com informação proveniente do *backend*. De seguida, implementaram-se e testaram-se métodos no *frontend* que possibilitassem obter e tratar esses dados de acordo com o que se pretendia. Implementou-se ainda outro método relacionado à tranferência de ficheiros.
    - Esta tarefa implicou algumas alterações nos métodos que serviam de *handlers* no acesso à informação necessária do *backend*, além de alguns problemas que surgiram no método de transferência, os quais exigiram a ajuda de membros mais experientes da equipa, tendo-se, portanto, efetuado em conjunto, as respetivas depurações em conjunto.
- 2ª Tarefa – Componente de *frontend:*
    - Na segunda tarefa desta *sprint*, que também estava relacionada com o *frontend*, tinha-se como objetivo alterar alguns *endpoints* de um determinado serviço que estavam desatualizados. Alteraram-se esses *endpoints* nos respetivos métodos, atualizou-se o nome desses métodos no serviço e o nome de outros métodos utilizados noutras classes que o referenciavam, e por fim, removeram-se ainda algumas variáveis que se tornaram obsoletas.
    - Esta tarefa foi concluída rapidamente, visto que não houve muita complexidade, nem foi encontrado nenhum obstáculo significativo no seu desenvolvimento.
- 3ª Tarefa – Componente de validação:
    - Em relação à última tarefa desta *sprint*, esta representou a tarefa mais desafiadora e trabalhosa até àquele momento. Tratou-se da concretização de uma lista extensa de regras de validação minuciosas e um pouco abstratas. Estas regras exigiram um longo período para serem realizadas e tiveram de ser reformuladas diversas vezes. Foi necessário o suporte de outros membros da equipa, com os quais se interpretou qual seria a resolução mais correta a implementar. Também se criaram alguns métodos para ajudar essas regras, que fizeram parte dessas discussões. Para além do desenvolvimento das regras,

surgiu a necessidade de as documentar. Anexou-se automaticamente, através de um teste unitário já implementado por outro membro da equipa, informação num ficheiro *Excel* acerca de cada uma dessas regras, agrupadas por tipo de regra, após a validação das mesmas.

- Sendo a tarefa mais extensa até ao momento, ultrapassou o limite da *sprint* e ainda ficou incluída num grande período da seguinte.

5.3 Sprint 3

Na terceira *sprint*, apenas se completou uma tarefa, visto que grande parte do seu tempo foi utilizado para resolver a terceira tarefa da *sprint* anterior. Contudo, devido à experiência adquirida nessa tarefa e a alguns métodos reutilizáveis, foi notavelmente mais simples.

- 1ª Tarefa – Componente de validação:
  - Esta tarefa, semelhante à realizada anteriormente, também fez parte da componente de validação, mas foi muito menos extensa, exigindo uma lista menor de regras. Essas regras encontravam-se também mais bem especificadas, por isso não foi necessário reformulá-las diversas vezes, como anteriormente. Além disso, já existiam os métodos auxiliares necessários, logo, não se precisou de abordar nenhum destes possíveis entraves com outros colegas de equipa. Assim, necessitou-se apenas de desenvolver as respetivas regras, que foram feitas mais rapidamente, devido tanto à experiência obtida na tarefa anterior como à semelhança entre algumas regras dessas tarefas. Em seguida, validou-se essas regras através do mesmo teste unitário.
  - Desta forma, a tarefa foi completada dentro do período da *sprint*, no último dia deste intervalo.

5.4 Sprint 4

A quarta *sprint* foi bastante exigente, mesmo tendo incluído apenas uma tarefa, devido à sua alta complexidade e à necessidade de unificar diversas outras tarefas mais amplas.

- 1ª Tarefa – Componente de validação:
  - Esta foi a primeira e única tarefa desta *sprint*, correspondendo à criação de um projeto único de testes unitários. A tarefa exigiu um elevado grau de detalhe na sua execução.

- Inicialmente, foi necessário reorganizar os métodos de todos os projetos individuais de testes unitários para um novo projeto central, ajustando adequadamente as bibliotecas importadas relacionadas a esses métodos. Também foram adicionados métodos auxiliares para otimizar a manipulação de ficheiros no código, uma vez que a solução anterior exigia a criação de um ficheiro para cada tipo de teste unitário, gerando uma acumulação excessiva de ficheiros.
- Para resolver essa questão, foi implementada uma solução que manipulava os dados de cada regra de validação em memória, através de uma lista de objetos, com atributos a indicar a manipulação necessária para cada regra, armazenada num ficheiro JSON. Assim, foi adicionado um ficheiro JSON por cada regra de validação.
- A execução desta tarefa exigiu bastante tempo, pois a adição de objetos foi feita manualmente para cada teste unitário previamente realizado por todos os membros da equipa em tarefas anteriores.
- Esta tarefa representou a mais extensa até ao momento, já que exigiu o desenvolvimento de uma nova solução, baseada na integração de várias tarefas complexas. A sua conclusão ultrapassou o período inicialmente previsto, ocupando grande parte da próxima *sprint*.

5.5 Sprint 5

Na quinta *sprint*, a maior parte do desenvolvimento concentrou-se na tarefa herdada da *sprint* anterior. Apesar do longo período ocupado pela mesma, conseguiram-se concluir mais duas tarefas relativamente simples.

- 1ª Tarefa – Componente de *backend*:
  - A primeira tarefa associada a esta *sprint* representou uma alteração simples na base de dados, bastando, portanto, remover dois registos de uma tabela. Assim, começou-se por eliminar ambos os *seeds* desses registos no respetivo ficheiro. Em seguida, criou-se uma migração e, por fim, aplicaram-se e testaram-se essas mudanças na base de dados.
  - Devido à simplicidade desta tarefa, foi concluída no próprio dia em que foi iniciada.
- 2ª Tarefa – Componente de *frontend*:

- Relativamente à segunda tarefa, semelhante à anterior, também não exigiu muito tempo, devido à sua baixa dificuldade. Esta relacionou-se à primeira tarefa da segunda *sprint* e incidiu no facto de a tabela associada estar a acumular muitos registos no *frontend*. Assim, implementou-se a paginação dessa tabela. Como já existiam algumas páginas nesta componente que possuíam paginação, foram utilizadas essas implementações como base, o que facilitou a execução da tarefa.
- Assim, graças à sua simplicidade e à existência de referências de soluções similares esta tarefa foi finalizada na mesma *sprint*.

5.6 Sprint 6

Nesta *sprint*, o foco centrou-se na adição e correção de diversas funcionalidades no *backend*, abrangendo várias implementações. Foram completadas um número considerável de tarefas, o que foi fundamental para aprofundar o conhecimento, especialmente sobre a arquitetura da aplicação.

- 1ª Tarefa – Componente de *backend:*
  - O objetivo da primeira tarefa desta *sprint* consistiu na criação de um utilizador específico para a aplicação, designado *SuperAdmin*, que deve ter acesso a todas as permissões.
  - Para implementar esta tarefa, foi criada a classe *UserSeeder* num projeto específico da solução e adicionado um *seed* do tipo *User*, preenchendo todas as propriedades necessárias para o funcionamento desejado.
  - Por fim, adicionou-se uma migração com o nome *AddUserSeeder*, gerou-se o respetivo script em SQL, atualizou-se a base de dados e verificou-se se a alteração foi aplicada corretamente.
- 2ª Tarefa – Componente de *backend*:
  - Esta tarefa está relacionada com a pesquisa e implementação inicial realizada na segunda tarefa da primeira *sprint*, que visava a integração de uma nova componente. O objetivo agora foi completar essa integração.
  - Para implementar a integração, foi necessário alterar a parte da lógica de um método que utilizava uma abordagem diferente. Para isso, foi criada a classe *FilesHttpClient*, juntamente com a interface correspondente, implementando os seguintes métodos:

        - *DownloadFile*: Este método garante a lógica para obter o conteúdo de um ficheiro em formato *base64*, utilizando informações prévias que o identifiquem, através de um *endpoint* específico fornecido por outra equipa relacionada à integração.
        - *UploadFile*: Este método foi projetado para receber o conteúdo de um ficheiro em formato *base64*, manipulá-lo, encriptá-lo, armazená-lo e preparar os cabeçalhos necessários para efetuar o carregamento através de um *endpoint* também fornecido por outra equipa.
    - Seguidamente, foram testadas e adicionadas as novas implementações, utilizando os novos métodos por meio de injeção de dependências. Além disso, foram adicionados novos ambientes na tabela *Environments*, assim como *endpoints* e informações relevantes na tabela *AccessPoints*, incluindo as *seeds* necessárias nas classes *EnvironmentSeeder* e *AccessPointSeeder*.
    - Por fim, na tabela *MiddlewareSettings*, foi adicionado um novo registo associado a esta funcionalidade, com um *seed* que contém as informações necessárias na classe *MiddlewareSettingSeeder*, permitindo a alternância entre a nova e a antiga integração.
- 3ª Tarefa – Componente de *frontend:*
    - O objetivo desta tarefa consistiu em incluir serviços na página de monitorização no *frontend*, especificamente para alguns serviços como o *Finder*.
    - Para tal, inicialmente foram adicionados métodos ao serviço de *healthcheck* para consumir os dados disponibilizados na API de *healthcheck* do *Middleware*, através dos *endpoints* correspondentes a cada serviço.
    - Depois, foram adicionados dois métodos na classe da componente de monitorização para incluir informações sobre os serviços monitorizados e o seu estado, obtidas pelos métodos criados no serviço de *healthcheck*.
    - Por fim, foram incorporados os dois últimos métodos ao método *refresh* da classe da componente de monitorização para possibilitar a atualização do estado dos serviços.
- 4ª Tarefa – Componente de *backend;*
    - Nesta tarefa o objetivo consistia em implementar a possibilidade de alterar o estado de um fluxo através do *frontend*.
    - Para isso, iniciou-se com a adição do *handler SetFlowStatus* à classe *FrontEndController*, assim como à respetiva interface.

  - Seguidamente, foi adicionada uma nova permissão ao *PermissionSeeder*, associada a esse *handler*, uma vez que ainda não existia uma permissão para essa ação.
  - Depois, foi implementado um método no *FrontEndProcessor* e na interface correspondente, que seria utilizado por esse *handler*, com a lógica necessária para alterar o estado do fluxo, utilizando o Entity Framework.
  - Para permitir que o estado dos fluxos no *frontend*, foi também adicionado o *handler GetFlowStatus* ao *FrontEndController*, a respetiva permissão no *PermissionSeeder* e o método chamado por esse *handler* para retornar os estados de todos os fluxos ao *FrontEndProcessor*.
- 5ª Tarefa – Componente de *backend*:
  - Com a adição e remoção de várias permissões ao longo do desenvolvimento, surgiu a necessidade de atualizar as permissões para os utilizadores das funções *Admin* e *User*.
  - Assim, foram adicionadas na classe *RolePermissionSeeder* as *seeds* com as permissões que faltavam, associadas a ambas as funções.
  - Da mesma forma, foram removidas as *seeds* associadas a permissões que já não eram pertinentes a essas funções no *RolePermissionSeeder*, e as que não faziam mais sentido estarem vinculadas a qualquer função foram removidas da classe *PermissionSeeder*.
- 6ª Tarefa – Componente de *backend:*
  - Após a tarefa anterior, foi identificado que era possível especificar algumas permissões da aplicação, o que resultou na obsolescência de outras.
  - Assim, na classe *PermissionSeeder*, foram removidas as permissões que se tornaram obsoletas e foram adicionadas novas permissões mais abrangentes.
  - Por fim, foram identificados todos os *handlers* que utilizavam as permissões anteriores e estas foram substituídas pelas novas adicionadas nesta tarefa.
- 7ª Tarefa – Componente de *backend:*
  - Para permitir o filtro na pesquisa de um histórico de utilizadores numa página do *frontend*, foi necessário alterar parte da lógica no *backend*, possibilitando que as informações retornadas fossem filtradas por nome de utilizador, ação ou intervalo de datas.
  - Para isso, foi criado o *Data Transfer Object* (DTO) *UserHistoryRequest*, com os atributos *Username*, *Action* e *Date*, e adicionado um parâmetro dessa classe

no *handler GetUserHistory*, tanto na classe *FrontEndController* como na respetiva *interface*.

- Finalmente, para filtrar a pesquisa, foi adicionado esse DTO nos parâmetros do método *Get* do repositório *UserHistoryRepository* e, após isso, foi atualizada a forma como a pesquisa é realizada nesse método com os valores obtidos por parâmetro.

- 8ª Tarefa – Componente de *frontend:*
  - O objetivo desta tarefa consistiu em reformular algumas mensagens de sucesso apresentadas no *frontend*.
  - Para tal, o processo passou por corrigir as *strings* definidas no ficheiro JSON de traduções, que contém todas as mensagens utilizadas nas páginas do *frontend*.
  - Por fim, foram testados os métodos responsáveis por disponibilizar essas mensagens e foi verificado se as alterações foram aplicadas corretamente.

5.7 Sprint 7

Semelhante à *sprint* anterior, esta também englobou uma grande quantidade de tarefas, uma vez que, nesta fase, já se tinha alcançado maior familiaridade com a estrutura das diferentes componentes da solução. Foram corrigidas algumas implementações no *backend*, e no *frontend* adicionou-se uma nova página e incluíram-se detalhes adicionais.

- 1ª Tarefa – Componente de *backend:*
  - Nesta tarefa, foi detetado um erro ao alterar um registo do tipo *Party*, através do método *SetParty* da classe *FrontEndController*. O erro ocorria quando um atributo era enviado como nulo, resultando numa falha ao atualizar a tabela.
  - O problema surgiu numa verificação prévia ao validar se as mensagens de sucesso estavam corretamente alteradas, quando um dos *handlers* retornava uma mensagem de erro ao receber o atributo a nulo.
  - A solução consistiu em ajustar a verificação do atributo antes de o atribuir à entidade *Party*. Caso o valor do atributo fosse fornecido, este era atribuído ao *Party*, caso contrário, não se adicionava uma instância com valores a nulo, eliminando assim o erro.

- 2ª Tarefa – Componente de *backend:*
  - O problema nesta tarefa estava relacionado com a remoção de uma permissão de uma determinada função. Um erro ocorria na tabela *RolePermissions*,

devido à incapacidade de localizar o registo correspondente à combinação da função e permissão.

  - Para resolver, foi alterado o retorno da pesquisa, utilizando o método *FirstOrDefault* em vez de retornar um *IEnumberable* com uma sequência de objetos. Adicionou-se também o identificador da permissão como parâmetro de pesquisa, assegurando que o registo correto era encontrado e atualizado.

- 3ª Tarefa – Componente de *frontend:*
  - Nesta tarefa, adicionaram-se campos de pesquisa na página de registo de ações dos utilizadores, na componente *UserHistoryComponent*. Os campos de pesquisa incluíram o nome de utilizador, ação e a data.
  - Em primeiro lugar, os campos de pesquisa foram adicionados ao ficheiro HTML da componente *users-history,* utilizando *divs*, *labels* e *inputs*, com recurso a classes do *Bootstrap* para formar os filtros.
  - A lógica foi implementada no ficheiro TypeScript correspondente, incluindo métodos como *resetFilters* e o envio dos parâmetros de pesquisa para o *backend* através de um pedido. Para tal, foi criado o DTO *UserHistorySearch*, que permitiu a passagem dos parâmetros na consulta ao *backend*.
- 4ª Tarefa – Componente de *backend:*
  - Ao adicionar filtros às páginas, verificou-se que as pesquisas estavam a ser feitas de forma literal, ou seja, apenas retornavam registos com valores exatamente iguais aos dos filtros. O objetivo foi modificar as pesquisas para retornarem também registos com valores parcialmente correspondentes.
  - A solução envolveu a substituição do método *Equals* pelo método *Contains* nas pesquisas, permitindo que fossem obtidos registos com valores parcialmente coincidentes.
- 5ª Tarefa – Componente de *backend:*
  - Esta tarefa envolveu a criação de *handlers* no *backend* para permitir a alteração de informações do utilizador, como o nome, o *email* e *palavra-passe* na página de Dados do Utilizador.
  - Criou-se o DTO *SetUserDataRequest*, e foram implementados os *handlers SetUserProfile* e *SetUserPassword* no *FrontEndController*. Estes métodos receberam o DTO por parâmetro, executaram validações e atualizaram os registos necessários. Também foi implementado um método para verificar a existência de emails duplicados na tabela de utilizadores.
- 6ª Tarefa – Componente de *backend:*

  - Até ao momento, a encriptação/desencriptação de palavras-passe exigia a instância de uma classe específica. Decidiu-se tornar essa classe e os seus métodos estáticos, para facilitar a utilização.
  - Alterou-se a assinatura da classe para *static*, bem como dos seus métodos, e ajustou-se o código onde estes eram chamados, eliminando a necessidade de instanciar a classe.
- 7ª Tarefa – Componente de *frontend:*
  - Com a implementação de uma nova funcionalidade no *backend*, foi necessário criar uma página no *frontend* para gerir essa funcionalidade.
  - Criou-se a componente *code-lists-component* no Angular, juntamente com os ficheiros necessários. Adicionou-se o caminho no menu suspenso, criou-se uma tabela para apresentar os registos e implementaram-se botões para atualizar e eliminar registos, e novas mensagens de sucesso/erro foram incluídas no ficheiro JSON de traduções.
- 8ª Tarefa – Componente de *frontend:*
  - O objetivo desta tarefa foi acrescentar uma coluna com o título "Formato" numa tabela, alimentada pelos valores do campo *ContentType* de uma tabela no *backend*. O *handler* já retornava esses valores, sendo necessário apenas adaptar o *frontend* para utilizá-los.
  - Foi adicionado o HTML necessário para criar a nova coluna na tabela e integrar os valores do campo *ContentType*.
- 9ª Tarefa – Componente de *frontend:*
  - Até então, sempre que um utilizador realizava uma pesquisa numa página, era registada uma ação de utilizador. O objetivo desta tarefa foi garantir que esse registo só acontecia ao aceder à página, e não sempre que se fazia uma pesquisa.
  - Para isso, ajustou-se o código para que o registo da ação fosse feito no método *ngOnInit* das páginas, garantindo que apenas era registada a ação de acesso à página.
- 10ª Tarefa – Componente de *frontend:*
  - Algumas colunas das tabelas no *frontend* já permitiam a inversão da ordem dos valores, mas essa funcionalidade não estava presente em todas as colunas relevantes. O objetivo foi garantir que a inversão da ordenação fosse possível em todas as colunas adequadas.

- Implementou-se a função para utilizar o método *orderBy* do *UtilsService* em todos os ficheiros TypeScript das componentes relevantes. Foram adicionadas variáveis para identificar a coluna e a direção da ordenação, e o HTML foi ajustado para aplicar a ordenação correta.

5.8 Sprint 8

Durante a 8ª *sprint*, diversas tarefas foram realizadas tanto no *backend* quanto no *frontend*, com um foco especial em mudanças estruturais, como a remoção de arquivos obsoletos e a reformulação de classes e atributos.

- 1ª Tarefa – Componente de *backend:*
  - No início do projeto, as credenciais dos ambientes do *access point* eram armazenadas em arquivos *appSettings*. Com a evolução da aplicação, essas credenciais passaram a ser geridas pela base de dados, tornando os atributos dos arquivos *appSettings* obsoletos.
  - Como parte da tarefa, foi verificado que essas credenciais não eram mais utilizadas, e, em seguida, todas as informações obsoletas foram removidas desses arquivos.
- 2ª Tarefa – Componente de *backend:*
  - Devido à criação e alteração de algumas páginas no *frontend*, foi necessário ajustar os nomes das permissões utilizadas. Inicialmente, os nomes foram corrigidos nos objetos do *PermissionSeeder*. Posteriormente, as permissões antigas foram substituídas nos *handlers* das páginas, e uma migração foi criada para refletir essas mudanças na base de dados. O código SQL gerado foi adicionado ao *Script* de atualização da base de dados.
- 3ª Tarefa – Componente de *frontend:*
  - Foi implementada uma verificação do *frontend* para garantir que, ao definir a palavra-passe de um utilizador, a nova palavra-passe seja diferente da atual. Para evitar chamadas desnecessárias ao *backend*, foi adicionado um método *isSamePassword* no TypeScript da componente e integrado ao método *validForm*, que verifica a validade do formulário. Uma nova mensagem de erro foi adicionada ao ficheiro JSON, exibida via *Property Binding* no Angular.
- 4ª Tarefa – Componente de *frontend:*
  - Para manipular a tabela *Tags*, foi criada a página *tags-component*. Os arquivos HTML e TypeScript foram desenvolvidos com base em outras páginas da aplicação, com as colunas necessárias e botões para acionar os *handlers* do

backend. Foi adicionado também um *modal* de confirmação para cada operação e frases de confirmação no ficheiro JSON de mensagens.

- 5ª Tarefa – Componente de *backend:*
    - O *handler* da classe *MessagesController* que não era mais utilizado foi removido. Isso incluiu a exclusão do método correspondente no *MessagesController*, da lógica associada no *FrontEndController* e dos DTOs usados para transporte de dados.
- 6ª Tarefa – Componente de *backend:*
    - A classe *FrontEndController* estava com muitos *handlers*, o que dificultava a organização. Foi realizada a divisão dos *handlers* em controladores específicos para cada funcionalidade.
    - Com isso, o arquivo *FrontEndController* tornou-se obsoleto e foi removido após verificar que os *handlers* foram corretamente transferidos.
- 7ª Tarefa – Componente de *backend:*
    - Foi criado o campo booleano *DefaultPassword* na tabela de utilizadores para permitir ao utilizador efetuar *login* pela primeira vez, através de uma palavra-passe padrão. Este campo foi configurado para ser verdadeiro na criação de novos utilizadores e alterado para falso após a alteração da palavra-passe, na primeira vez que o utilizador efetua o *login*. As modificações foram implementadas no método *AddUser* e *SetUserPassword* da classe *FrontEndProcessor*, e as alterações ao modelo foram aplicadas por meio de uma migração.

5.9 Sprint 9

Nesta *sprint* foram realizadas menos tarefas devido a fatores externos, relacionados com burocracia vinculada ao cliente, mas houve um equilíbrio entre tarefas do *backend e frontend*.

- 1ª Tarefa – Componente de *frontend:*
    - A *string* a indicar que campos são obrigatórios estava codificada em várias componentes. Para centralizar isso, foi adicionado um novo atributo *REQUIRED_FIELDS* no ficheiro JSON de traduções, substituindo a *string* nas componentes onde era utilizada.
- 2ª Tarefa – Componente de *backend:*

    - Algumas condições de *BadRequest* no *UsersController* estavam a bloquear a exibição de mensagens de erro detalhadas. Essas condições foram removidas dos métodos, permitindo que mensagens de erro personalizadas sejam exibidas ao utilizador.

- 3ª Tarefa – Componente de *backend:*
    - O método *AddUser* do *FrontEndProcessor* foi reformulado para validar os campos separadamente, evitando a exibição de uma mensagem de erro genérica. Agora, cada erro de validação gera uma mensagem individual. Para isso, foram adicionadas mensagens de erro personalizadas no ficheiro de mensagens, e, no ficheiro de mensagens, e no serviço *Toast*, de notificações, e do *frontend*, foi habilitada a formatação HTML para exibir múltiplas linhas nas notificações de erro.
- 4ª Tarefa – Componente de *frontend:*
    - Foi necessário permitir ao utilizador aceder a detalhes de erros relacionados a mensagens. Isso envolveu a adição dos campos *ErrorCode* e *ErrorMessage* ao DTO correspondente e a criação de uma nova grelha no *modal* de detalhes de mensagens, com uma tabela e as colunas necessárias. Os títulos da grelha foram adicionados ao ficheiro JSON de traduções.
- 5ª Tarefa – Componente de *backend:*
    - Mensagens *harcoded* ainda presentes em alguns *handlers* da API *FrontEnd* foram transferidas para o ficheiro de traduções. Para cada mensagem fixa foi criado um novo atributo no ficheiro de traduções e, em seguida, essas mensagens foram substituídas pelo valor dos novos atributos nos respetivos *handlers*.

6 Evento de testes

Após o período de formações, apesar de o projeto ainda estar à espera de resultados, houve o planeamento de um evento por parte de uma entidade relacionada ao cliente, com grande relevância para a solução, que ocorre em intervalos de cerca de cinco meses desde o início do projeto. Este evento tem como objetivo verificar o ponto de situação da solução desenvolvida por cada equipa, sendo que as equipas de diferentes países são responsáveis pelas suas respetivas implementações. Estas equipas devem executar um conjunto de testes comuns, em conjunto com os restantes países, de forma a analisar a eficiência de cada funcionalidade da aplicação.

De notar que este evento é tradicionalmente organizado presencialmente em Bruxelas, sendo que membros da equipa deste estágio já tinham viajado para participar em edições anteriores, de forma a facilitar a comunicação com as equipas dos restantes países membros e obter a experiência de representar a solução de Portugal a nível internacional. No entanto, nesta última edição, a equipa participou de forma remota, devido à fase em que o projeto se encontrava.

Este evento representou a primeira participação ativa num evento deste género. Inicialmente, houve cerca de um mês de preparação para o evento, com a orientação de um membro mais experiente da equipa, que demonstrou aos desenvolvedores juniores, através de uma plataforma de testes preparatórios disponibilizada pela entidade promotora, qual a finalidade de cada teste e que protocolo deveria ser seguido para os executar corretamente. Estes testes, mesmo que preparatórios, eram registados na plataforma e recebiam uma avaliação associada. Existem alguns tipos diferentes de avaliações:

- Positiva: No caso de o teste ter sido executado corretamente, demonstrando que a funcionalidade testada estava implementada como suposto, em ambos os países participantes.
- Razoável: Quando a funcionalidade estava parcialmente correta em ambas as soluções.
- Negativa: Quando a funcionalidade estava mal implementada, ou seja, não funcionava na sua totalidade em alguma das aplicações.

Foi necessário um grande trabalho de preparação para garantir que todos os testes recebessem avaliação positiva. Após essa fase preparatória, deu-se início ao evento, que teve a duração de três dias. Durante esse período, foi necessário replicar os testes realizados na fase de preparação, mas desta vez em ambiente de pré-produção, e comunicar com várias equipas

em simultâneo para perceber quais as funcionalidades que já tinham implementado e que deveriam ser testadas.

O evento foi muito positivo, tanto em termos de resultados como em termos de desenvolvimento de competências. Houve sucesso na comunicação com os restantes membros do evento e também no alcance de uma meta inédita em relação ao número de testes efetuados pela equipa de Portugal durante estes eventos. Graças a este desempenho, obtivemos duas distinções diferentes da entidade organizadora:

1. País com a maior quantidade de testes concluídos com sucesso durante a fase preparatória do evento.
2. País com a maior quantidade de testes concluídos com sucesso durante o período do evento.

Estes resultados destacaram o trabalho de equipa e reforçaram o sucesso na preparação e execução dos testes.

7 Conclusões

Para concluir, este estágio foi uma grande oportunidade para adquirir novas competências, tendo-se consolidado conhecimentos em tecnologias que já tinham sido utilizadas e adquirido experiência em ferramentas e *frameworks* com as quais não havia contacto prévio, como o caso da *framework* Angular, utilizada ao longo do projeto, ou em contextos de formação.

O ambiente disponibilizado pela empresa foi também essencial para que houvesse crescimento profissional, seja pelo acompanhamento atencioso do coordenador, pelo espírito de cooperação e entreajuda dos colegas de equipa ou pelas ferramentas fornecidas.

Além disso, a participação num projeto de grande dimensão, mesmo já numa fase avançada de desenvolvimento, permitiu compreender a importância da adoção de boas práticas de programação, essenciais para garantir a escalabilidade e organização do código.

Nos momentos finais do estágio, a participação num evento de testes internacional foi igualmente enriquecedora, não só pela experiência de trabalhar num ambiente competitivo entre equipas de desenvolvimento de diferentes países, mas também pelo aperfeiçoamento da fluência em inglês, uma vez que foi a língua utilizada para a comunicação com as restantes equipas.

Os principais objetivos do estágio concentraram-se no desenvolvimento da aplicação, o que, além do conhecimento adquirido durante o estágio, exigiu a aplicação prática de conceitos aprendidos ao longo do percurso académico em contexto profissional. Deste modo, todos os objetivos foram alcançados, quer na transformação do conhecimento em funcionalidades nos diferentes componentes do sistema, no desenvolvimento profissional, quer na participação com impacto positivo no evento de testes.

Foi também praticada a metodologia Scrum em ambiente profissional, o que permitiu uma adaptação progressiva ao ritmo de trabalho e incentivou a gestão eficiente do tempo, através da categorização das tarefas por componentes e complexidade técnica.

Por fim, a experiência do estágio foi extremamente positiva, constituindo um ponto fundamental para o primeiro contacto com o mundo profissional e criando as bases para uma futura integração numa equipa de forma mais confiante e confortável.

8 Referências Bibliográficas

[1] GLINTTGlobal (2024). Nós somos GLINTT Global. Consultado em setembro de 2024, de GLINTTGlobal: https://www.glinttglobal.com/

[2] GLINTTLife (2024). Care for tomorrow, a visão da GLINTT Life. Consultado em setembro de 2024, de GLINTTLife: https://www.glinttlife.com/care-for-tomorrow/

[3] GLINTTGlobal (2024). We think tech, Tecnologias Inteligentes Globais. Consultado em setembro de 2024, de GLINTTGlobal: https://www.glinttglobal.com/we-think-tech/

[4] GLINTTGlobal (2024). Cultura global, os nossos valores. Consultado em setembro de 2024, de GLINTTGlobal: https://www.glinttglobal.com/cultura-global/

[5] GLINTTGlobal (2024). GLINTT Global Academy. Consultado em setembro de 2024, de GLINTTGlobal: https://www.glinttglobal.com/glintt-global-academy/

[6] Europa (2024). eDelivery . Consultado em setembro de 2024, de Europa: https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/eDelivery

[7] Udemy (2024). About Udemy. Consultado em setembro de 2024, de Udemy: https://about.udemy.com/

[8] TypeScript (2024). What is TypeScript. Consultado em setembro de 2024, de TypeScript: https://www.typescriptlang.org/

[9] Angular. (28 de fevereiro de 2024). Informações sobre Data Binding. Consultado em junho de 2024, de Angular: https://v17.angular.io/guide/binding-syntax

[10] Mosaico. (24 de junho de 2024). Metodologias Ágeis. Consultado em setembro de 2024, de Mosaico: https://mosaico.gov.pt/areas-tecnicas/metodologias-ageis

[11] NimbleWork. (11 de setembro de 2024). Scrum Methodology Explained: An Introduction for Agile Team. Consultado em dezembro de 2023, de NimbleWork: https://www.nimblework.com/agile/scrum-methodology/

[12] Scrum. (2024). Ciclo de vida da metodologia Scrum. Consultado em dezembro de 2023, de Scrum: https://www.scrum.org/resources/what-scrum-module

[13] BusinessMap. (2024). 12 Principios ágeis. Consultado em dezembro de 2023, de BusinessMap: https://businessmap.io/pt/metodologia-agil/principios-manifesto-agil

[14] Microsoft. (14 de março de 2017). Lançamento mundial do Microsoft Teams para o Office 356. Consultado em dezembro de 2023, de Microsoft: https://news.microsoft.com/2017/03/14/microsoft-teams-rolls-out-to-office-365-customers-worldwide/

[15] Microsoft. (2024). Funcionalidade de chat no Microsoft Teams. Consultado em dezembro de 2023, de Microsoft: https://support.microsoft.com/en-us/office/chat-with-others-in-microsoft-teams-0c71b32b-c050-4930-a887-5afbe742b3d8

[16] Microsoft (2024). Principais funcionalidades do software de videoconferências. Consultado em setembro de 2024, de Microsoft: https://www.microsoft.com/pt-pt/microsoft-teams/video-conferencing

[17] Microsoft (2024). Agendar uma reunião do Microsoft Teams a partir do Outlook. Consultado em setembro de 2024, de Microsoft: https://support.microsoft.com/pt-pt/office/agendar-uma-reuni%C3%A3o-do-microsoft-teams-a-partir-do-outlook-883cc15c-580f-441a-92ea-0992c00a9b0f

[18] Microsoft (2024). Trabalhar em conjunto no Microsoft Teams. Consultado em setembro de 2024, de Microsoft: https://support.microsoft.com/pt-pt/office/trabalhar-em-conjunto-no-microsoft-teams-8066077a-e2a7-45b7-a8d1-adfd0a944b67

[19] Microsoft (22 de maio de 2024). Overview of teams and channels in Microsoft Teams. Consultado em setembro de 2024, de Microsoft: https://learn.microsoft.com/en-us/microsoftteams/teams-channels-overview

[20] Microsoft (2024). Utilizar a encriptação ponto a ponto para chamadas do Microsoft Teams. Consultado em setembro de 2024, de Microsoft: https://support.microsoft.com/pt-

pt/office/utilizar-a-encripta%C3%A7%C3%A3o-ponto-a-ponto-para-chamadas-do-microsoft-teams-1274b4d2-b5c5-4b24-a376-606fa6728a90

[21] Trello (2023). Informação sobre o Trello. Consultado em setembro de 2024, de Trello: https://trello.com/

[22] GitHub (2024). Informação sobre o GitHub. Consultado em setembro de 2024, de GitHub: https://github.com/

[23] SourceTree. (2024). Funcionalidades do Source Tree. Consultado em dezembro de 2023, de SourceTree: https://www.sourcetreeapp.com/

[24] Atlassian (2024). What is Git. Consultado em setembro de 2024, de Atlassian: https://www.atlassian.com/git/tutorials/what-is-git

[25] Atlassian. (2024). Funcionalidades do JIRA. Consultado em dezembro de 2023, de Atlassian: https://www.atlassian.com/software/jira

[26] Atlassian (2024). Git commit. Consultado em setembro de 2024, de Atlassian: https://www.atlassian.com/git/tutorials/saving-changes/git-commit

[27] GitHub (2024). Commiting and reviewing changes to your project in GitHub Desktop. Consultado em setembro de 2024, de GitHub: https://docs.github.com/en/desktop/making-changes-in-a-branch/committing-and-reviewing-changes-to-your-project-in-github-desktop

[28] Atlassian (2024). Git push. Consultado em setembro de 2024, de Atlassian: https://www.atlassian.com/git/tutorials/syncing/git-push

[29] Atlassian (2024). Git pull. Consultado em setembro de 2024, de Atlassian: https://www.atlassian.com/git/tutorials/syncing/git-pull

[30] Microsoft. (14 de fevereiro de 2024). What is SQL Server. Consultado em fevereiro de 2024, de Microsoft: https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16

[31] Microsoft. (2024). Visual Studio 2022. Consultado em fevereiro de 2024, de Microsoft: https://visualstudio.microsoft.com/vs/

[32] Microsoft. (2024). Explicação da arquitetura MVC. Consultado em fevereiro de 2024, de Microsoft: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc

[33] Medium (20 de abril de 2017). Repository Design Pattern. Consultado em setembro de 2024, de Medium: https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30

[34] Medium (21 de dezembro de 2023). Validate with Fluent Validations in .NET. Consultado em setembro de 2024, de Medium: https://medium.com/codenx/code-with-fluent-validations-in-net-c-da2fb517566d

[35] Microsoft. (5 de abril de 2024). Utilização do NUnit. Consultado em abril de 2024, de Microsoft: https://learn.microsoft.com/pt-pt/dotnet/core/testing/unit-testing-with-nunit

[36] CodeProject (16 de fevereiro de 2024). Understanding ASP.NET Core Testing. Consultado em setembro de 2024, de CodeProject: https://www.codeproject.com/Articles/5377649/NUnit-in-ASP-NET-Core-What-You-Need-to-Get-Started

[37] Udemy. (agosto de 2024). Curso sobre o Angular. Consultado em junho de 2024, de Udemy: https://www.udemy.com/course/the-complete-guide-to-angular-2

[38] Medium (17 de dezembro de 2023). Introdução aos componentes no Angular. Consultado em setembro de 2024, de Medium: https://medium.com/@f.marinho162001/angular-parte-1-componentes-ea21520b1f37

[39] Infragistics. (2024). Informações sobre o One-Way-Binding. Consultado em junho de 2024, de Infragistics: https://www.infragistics.com/products/ignite-ui-angular/angular/components/general/wpf-to-angular-guide/one-way-binding

[40] Angular. (1 de setembro de 2023). Informações sobre Property Binding. Consultado em junho de 2024, de Angular: https://v17.angular.io/guide/property-binding

[41] Angular. (2020). Informações sobre Event Binding. Consultado em junho de 2024, de Angular: https://docs.angular.lat/guide/event-binding

[42] AngularMinds. (21 de abril de 2024). Implementação de Two-Way-Binding. Consultado em junho de 2024, de AngularMinds: https://www.angularminds.com/blog/how-to-implement-two-way-data-binding-in-angular

[43] Angular. (28 de fevereiro de 2022). Informações sobre directives. Consultado em julho de 2024, de Angular: https://angular.io/guide/built-in-directives

[44] Angular. (24 de outubro de 2023). Sistema de encaminhamento e lazy loading. Consultado em julho de 2024, de Angular: https://v17.angular.io/guide/router

[45] Angular. (7 de setembro de 2023). Introdução aos formulários do Angular. Consultado em julho de 2024, de Angular: https://v17.angular.io/guide/forms-overview

[46] WebFX (31 de agosto de 2023). Informação sobre os *modals*. Consultado em setembro de 2024, de WebFX: https://www.webfx.com/blog/web-design/examples-of-modal-windows/

[47] Medium (30 de setembro de 2023). What does handlers function mean. Consultado em setembro de 2024, de Medium: https://medium.com/@sultanbasitkhan830/what-does-handlers-function-mean-e27af3fcb488

[48] Medium (8 de abril de 2024). Injeção de Dependência: O que é e como usar. Consultado em setembro de 2024, de Medium: https://medium.com/@annarafadev/inje%C3%A7%C3%A3o-de-depend%C3%AAncia-o-que-%C3%A9-e-como-usar-7ec564b11b60