ANGULAR ATOMIC COMPONENTS ARCHITECTURE

(Versão final após defesa)

Nuno Rodrigo Lopes Pena

Relatório de Estágio para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Nuno Gonçalo Coelho Costa Pombo
Orientador da Emvenci: Alexandre Miguel Coelho Aniceto

julho de 2024

Declaração de Integridade

Eu, Nuno Rodrigo Lopes Pena, que abaixo assino, estudante com o número de inscrição M12018 de/o Mestrado em Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 10/07/2024

Assinado por: **Nuno Rodrigo Lopes Pena**
Num. de Identificação: 15078792
Data: 2024.07.10 20:44:42+01'00'

CHAVE MÓVEL

Agradecimentos

Após todo o trabalho aplicado neste relatório queria em primeiro lugar dar um agradecimento especial à minha família pelo apoio que me prestou ao longo deste caminho, e por sempre me proporcionar o tempo e o espaço necessário para poder trabalhar. Queria também agradecer a todos os meus amigos que me apoiaram, não só na realização deste relatório com as suas dicas, mas também a todos os outros que passaram neste meu ciclo de vida.
Um agradecimento à empresa EMVENCI por me permitir realizar o estágio nas suas instalações, a todos os elementos da empresa por me receberem de braços abertos, e em especial ao Engenheiro Alexandre Aniceto por toda a ajuda prestada durante esta etapa. Em adição, queria também agradecer ao meu orientador, Professor Doutor Nuno Pombo, por toda a ajuda e disponibilidade ao longo deste projeto.
Por fim, queria agradecer à Universidade da Beira Interior por tudo o que permitiu viver durante estes anos e a todos aqueles que estiveram envolvidos, de algum modo, neste projeto.

Resumo

O principal objetivo deste relatório passa por expor todo o trabalho realizado durante o estágio na empresa Emvenci. Este incide na realização de uma reformulação/criação de diversas páginas da aplicação web da empresa utilizando a metodologia de *Atomic Design*. Através desta reestruturação, um dos principais objetivos é permitir uma melhoria de performance do website, realizando otimizações ao nível do tempo de carregamento de conteúdo, número de linhas de código e tamanho que cada ficheiro ocupa. Desta forma, o documento começa por expor o universo de trabalho da empresa e os seus respetivos objetivos. De seguida, é feita uma descrição da plataforma utilizada pela empresa e uma explicação da metodologia a ser aplicada. No terceiro capítulo, são apresentadas as ferramentas utilizadas, bem como uma explicação das mesmas. Por fim, é retratado o desenvolvimento do estágio, referindo os aspetos mais importantes que foram realizados.

Palavras-chave

Design Atómico;Metodologia Front-end;Angular

Abstract

The primary objective of this report is to expose all the work done during the internship in the company Emvenci. This insides in the fulfillment of a reformulation/creation of several webpages of the company's web application, using the methodology Atomic Design. Using this restructuring, one of the main objectives is to allow an improvement in the website's performance, making optimizations in the content loading time, number of lines used to program and the size of each individual file. Therefore, this document starts by exposing the work environment inside the company and its objectives. Thereafter, a description of the platform's architecture is made, alongside a deep explanation of the new methodology that is going to be applied. In the following chapters, all the tools used during the internship are spread out, alongside a small explanation of them. To conclude, the development of the internship during these months is portrayed, referring all the major details that happened during the time in the company.

Keywords

Atomic Design;Front-end Methodology;Angular

Índice

Lista de Figuras

Lista de Acrónimos

API	Application Programming Interface
CSS	Cascading Style Sheets
GDPR	General Data Protection Regulation
HTML	HyperText Markup Language
SaaS	Software as a Service
UI	User Interface
UML	Unified Modeling Language
UX	User Experience
VSCode	Visual Studio Code

Capítulo 1

Introdução

O seguinte relatório de estágio descreve a realização de um projeto proposto pela empresa EMVENCI no âmbito da cadeira de Projeto de Dissertação ou de Estágio em Engenharia Informática. Neste primeiro capítulo vai ser descrita a empresa, bem como os vários motivos que levaram à realização deste projeto, os objetivos a atingir com o seu desenvolvimento e, por fim, a estrutura do relatório.

1.1 Empresa e Motivação

A Emvenci[2] é uma empresa que opera na área de segurança informática a nível global fundada em 2013 com o objetivo de ajudar os seus clientes a diminuir diversos riscos de segurança a que possam estar expostos. A plataforma desenvolvida pela empresa permite ajudar a sensibilizar os funcionários de empresas clientes em diversos temas de cibersegurança. O desenvolvimento de front-end para aplicações web pode rapidamente tornar-se algo repetitivo e *time-consuming*. Por este motivo, a utilização de novos e mais eficazes métodos de programação tem-se vindo a tornar uma necessidade no mundo do desenvolvimento web. Através da utilização de uma ou mais metodologias, é possível aumentar a produtividade bem como, a facilidade de aprendizagem para um novo programador. Para além das razões mencionadas anteriormente, a utilização de uma metodologia, permite uma escrita de código mais coerente, facilitando não só a leitura de código, mas também permitindo em certos casos a sua reutilização. Devido aos motivos apresentados anteriormente, concluí por bem que seria não só interessante aprender e implementar uma metodologia, mas também que o uso da mesma traz benefícios não só às empresas, mas também aos programadores front-end presentes e futuros.

1.2 Objetivos

Todo o software desenvolvido durante o presente estágio teve como objetivo o aprimoramento da aplicação web utilizada pela empresa Emvenci[2], não só a nível visual, mas também a nível de código e performance. A reformulação das diversas páginas web foi feita utilizando a metodologia do *Atomic Design* aplicada à programação em Angular. A utilização desta metodologia vai permitir a reutilização de diversos componentes na criação das diversas páginas web da aplicação.

1.3 Organização do Documento

O presente documento encontra-se organizado da seguinte forma:

1. O primeiro capítulo - **Introdução** - apresenta a empresa, a motivação, os objetivos e a organização do documento;

2. O segundo capítulo - **Estado da Arte** – descreve a plataforma e a sua antiga arquitetura, bem como o que é o *Atomic Design* e vantagens/desvantagens;

3. O terceiro capítulo – **Tecnologias e Ferramentas Utilizadas** – demonstra as tecnologias utilizadas durante o desenvolvimento do estágio;

4. O quarto capítulo - **Engenharia de Software** – apresenta a arquitetura utilizada e requisitos funcionais não funcionais;

5. O quinto capítulo - **Implementação** – retrata o início na empresa, bem como a introdução e desenvolvimento de páginas utilizando o *Atomic Design*;

6. O sexto capítulo - **Desenvolvimento** – explica de forma detalhada as páginas construídas durante o estágio;

7. O sétimo capítulo - **Conclusões e Trabalho Futuro** - apresenta as conclusões sobre o estágio e trabalho futuro.

Capítulo 2

Estado de Arte

2.1 Introdução

Neste capítulo vai ser feita uma apresentação da metodologia do Atomic Design, bem como uma comparação com outras metodologias utilizadas no desenvolvimento web front-end.

2.2 Plataforma

A plataforma criada pela empresa funciona como um SaaS que oferece diferentes opções para ajudar os seus clientes em diversas questões de cibersegurança. Um SaaS que significa *"software as a service"*, é uma forma de disponibilizar *software* através da *cloud* em que os utilizadores precisam de ter acesso à internet para aceder ao serviço disponibilizado. No caso da EMVENCI, é disponibilizada aos seus clientes uma aplicação web. Em adição, esta aplicação funciona com uma arquitetura *multitenancy*, o que significa que apenas existe uma instância do *software* que permite o acesso a diversos *"tenants"* (clientes) ao mesmo tempo sem que o acesso e ações de um dos *tenants* influencie outros. De seguida, são apresentadas algumas funcionalidades apresentadas pela aplicação web da empresa:

- **Policy Manager -** Permite aos utilizadores gerir as suas políticas, armazenadas na plataforma.
- **Phish Simulator -** Permite criar campanhas de phishing seguras, de modo a testar e educar os utilizadores.
- **Privacy Manager -** Possibilita aos utilizadores guardar e partilhar informação de forma segura segundo as novas normas do GDPR.
- **eLearning -** Fornece conteúdo online capaz de informar os utilizadores acerca de problemas de segurança.
- **Log Analytics -** Permite ao utilizador ver informação que foi processada de modo a ser facilmente processada e realizadas ações.
- **Threat & Vulnerability Manager -** Fornece a possibilidade de identificar vulnerabilidades através de uma análise ao cliente.

2.3 *Atomic Design*

Foi decidido pela empresa que seria utilizada a metodologia do *Atomic Design*[3] para realizar uma transformação a nível de programação e design da aplicação web já existente. Desta

forma, o *Atomic Design* é uma metodologia criada em 2013 por Brad Frost que procurava providenciar uma maneira de construir sistemas de design de interfaces de uma forma mais explicita e hierárquica. Pela própria palavra ***Atomic*** podemos perceber que esta metodologia se baseia no estudo da matéria e dos elementos da área da química. Desta forma, a ideia central por trás do *Atomic Design* é que uma interface do utilizador possa ser dividida em partes mais pequenas e indivisíveis, denominadas de "**átomos**". Esses átomos podem então ser combinados para formar "**moléculas**" mais complexas, que por sua vez podem ser combinadas para formar "**organismos**" ainda mais complexos e por fim juntando diversos organismos criamos "***templates***", que ao adicionar dados criamos o nível mais complexo, uma "**página**". Esta abordagem hierárquica de design permite a criação de interfaces de utilizador consistentes, escaláveis e modulares. Um dos principais benefícios do uso do *Atomic Design* é que ele permite que os designers e desenvolvedores criem e reutilizem elementos de interface de maneira sistemática e eficiente. Ao dividir uma interface de utilizador nos seus componentes atómicos, os designers podem criar uma biblioteca de elementos reutilizáveis, que podem ser organizados de diferentes maneiras de forma a criar uma variedade de *layouts* e designs diferentes. Esta abordagem modular de design permite economizar tempo e reduzir a redundância, uma vez que os designers e desenvolvedores não precisam de recomeçar do zero cada vez que criam uma nova interface. Em adição, o *Atomic Design* promove a criação de interfaces consistentes e coesas. Isto porque, ao estabelecer um conjunto de átomos e moléculas padronizados, os designers e desenvolvedores podem garantir que os designs são visualmente consistentes e atendem todos os padrões de design estabelecidos. Por fim, isto permite aos utilizadores uma melhor experiência, uma vez que faculta uma navegação mais facilitada devido a estarem familiarizados com a estrutura.

2.3.1 Componentes do *Atomic Design*

Como foi referido anteriormente, o *atomic design*[4] é divido em cinco fases sendo que a fase seguinte usa de alguma maneira a anterior. As fases são denominadas por:
átomos, sendo que um conjunto destes forma **moléculas**, um conjunto de moléculas formam **organismos**, que aquando vários, formam ***templates*** e quando os *templates* são preenchidos com dados, obtemos as **páginas**.

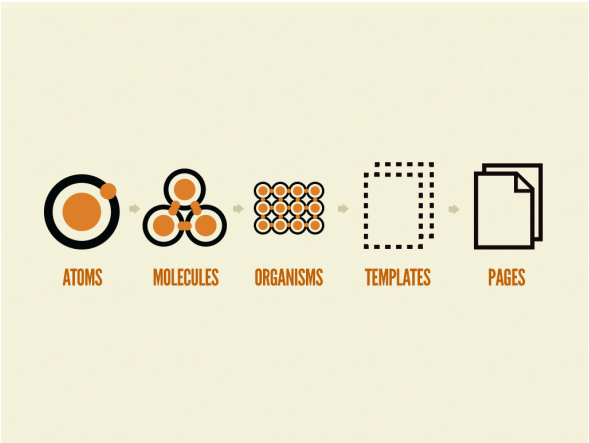

Figura 2.1: - Fases do *Atomic Design*[1]

2.3.1.1 Átomos

Os átomos, da mesma forma que na química, vão representar os blocos de construção fundamentais do nosso sistema. Estes incluem elementos *html* básicos tais como *labels*, *inputs*, botões entre outros, que não possam ser reduzidos sem perderem as suas funcionalidades. Em adição, cada átomo presente numa interface tem as suas características próprias, tais como dimensões, tamanho do texto e cor, o que permite adicionar o átomo a diferentes *templates* com pequenas alterações.[5]

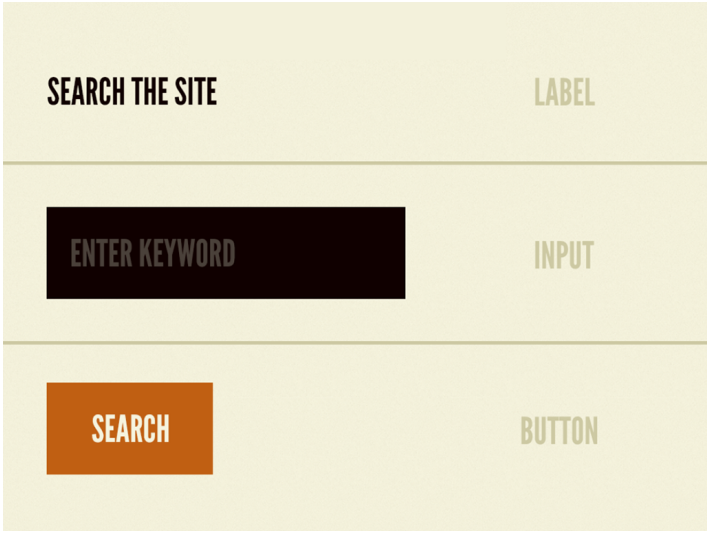

Figura 2.2: - Exemplo de átomos em *Atomic Design*[1]

2.3.1.2 Moléculas

As moléculas representam grupos simples de elementos de Design que funcionam em conjunto como uma unidade. Por exemplo, quando juntamos uma *label*, um botão e um *search input*, criamos desta forma um *search form*. Da mesma forma da química, onde podemos gerar diversas moléculas a partir de diversos átomos, no *Atomic Design* podemos fazer o mesmo. Ou seja, ao utilizar diversos átomos de formas diferentes conseguimos gerar uma enorme quantidade de moléculas diferentes. Uma vez criados, estes componentes podem ser reutilizados em qualquer parte de uma página que sejam necessários. A criação de diversos componentes simples, permite aos desenvolvedores e aos designers seguirem um dos principais princípios da computação, o princípio da responsabilidade única, que encoraja a fazer algo e fazê-lo bem. Ou seja, adicionar demasiada complexidade a um componente, pode tornar este disfuncional. Desta forma, a melhor abordagem é a criação de moléculas simples que encorajem a sua reusabilidade e promovam uma consistência na interface.[5]

2.3.1.3 Organismos

Organismos são componentes de UI relativamente complexos e compostos por uma junção de átomos e/ou moléculas e/ou outros organismos. Utilizando o exemplo anterior de um *search form*, ao adicionar mais alguns átomos ou moléculas, conseguimos criar um organismo que representa um cabeçalho de uma página. Enquanto que um organismo pode consistir de

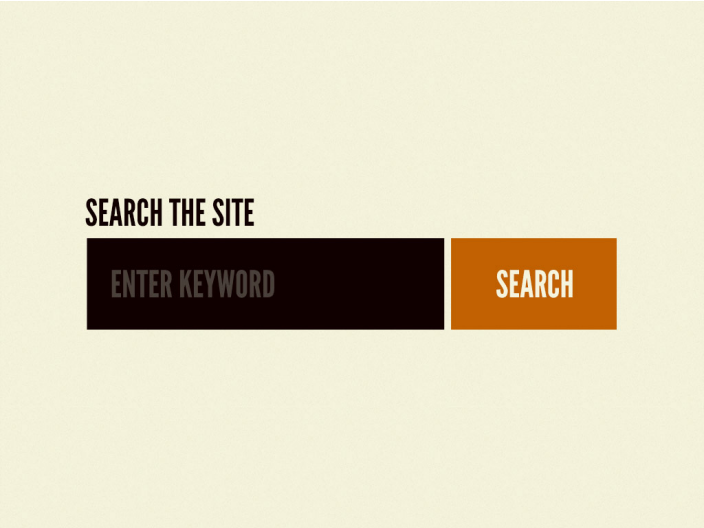

Figura 2.3: - Exemplo de uma molécula em *Atomic Design*[1]

diferentes tipos de moléculas, estes podem também consistir de apenas uma molécula repetida várias vezes, como por exemplo um componente de uma página de vendas que demonstra os produtos disponíveis em diversas linhas, este utiliza a mesma molécula repetida diversas vezes.[5]

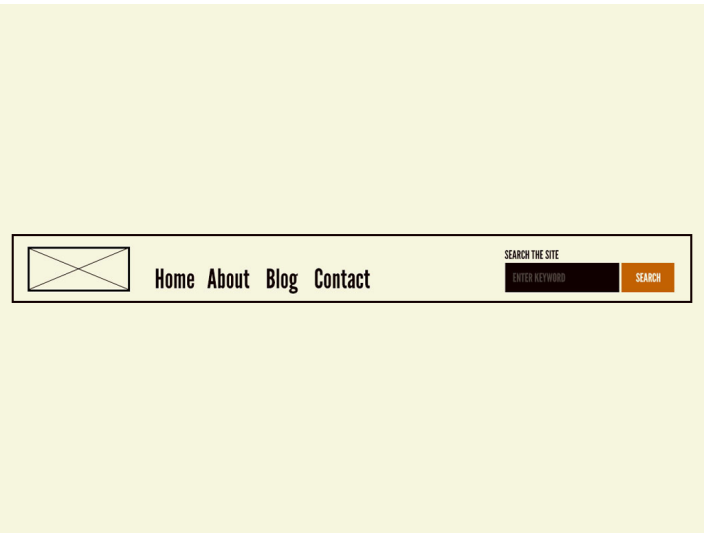

Figura 2.4: - Exemplo de um organismo em *Atomic Design*[1]

2.3.1.4 *Templates*

Os *templates* são objetos ao nível de uma página onde são implementados os componentes dentro de um *layout* para criar uma estrutura de design coesa. Ao pegarmos no exemplo do *header* definido anteriormente e aplicando-o a uma *home page*, estamos a providenciar um conteúdo aos organismos e moléculas abstratos. Isto porque, quando criamos um sistema de design é crucial demonstrar como os componentes se parecem e funcionam em conjunto num específico *layout*. Desta forma, podemos ver os *templates* como o diagrama do design final da página, sendo que os diferentes elementos ainda não contêm qualquer tipo de dados.[5]

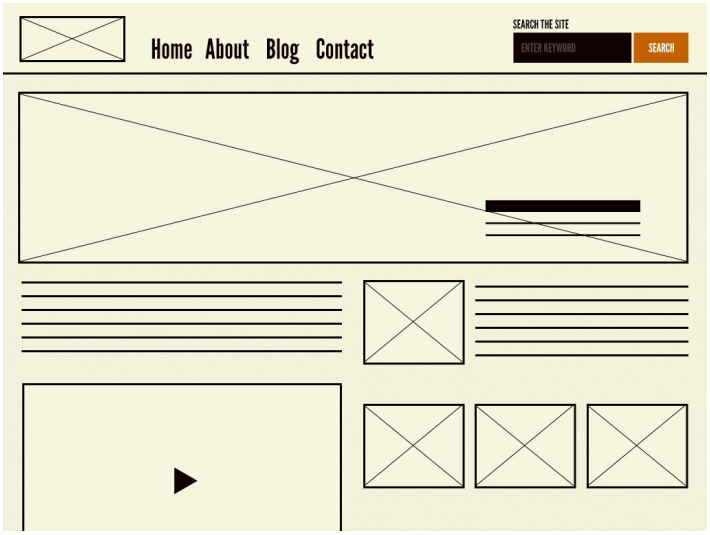

Figura 2.5: - Exemplo de um *template* em *Atomic Design*[1]

2.3.1.5 Páginas

Por fim, as páginas representam o estágio final da metodologia do *Atomic Design*. Nesta fase são adicionados dados aos templates, e onde grande parte do tempo é gasto pelos desenvolvedores. É também neste estágio que os testes de funcionalidades e design são realizados.[5]

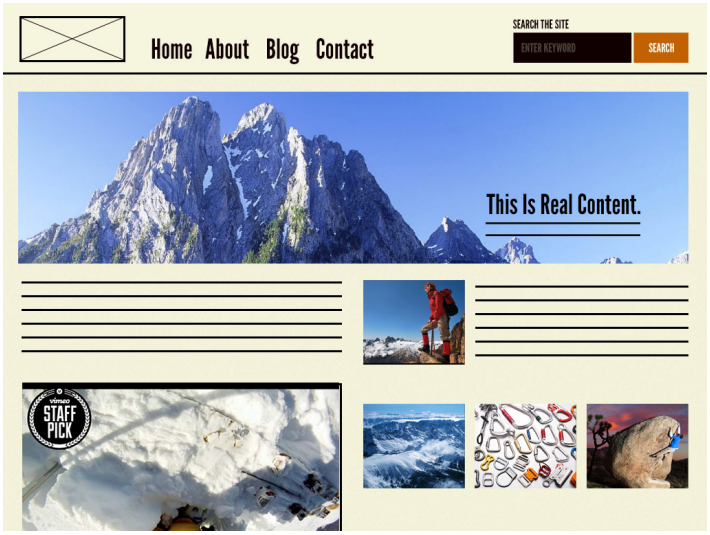

Figura 2.6: - Exemplo de uma página em *Atomic Design*[1]

2.3.2 Vantagens e desvantagens do *Atomic Design*

2.3.2.1 Vantagens

- Os componentes podem ser reutilizados e facilmente alterados para serem utilizados na situação especifica em que são precisos;[6]
- A consistência e a compreensão do código são maiores;
- A reutilização de código permite uma maior consistência ao nível de Design ao longo

do website;

- Facilidade de melhorar o código e remover certas partes que já não sejam necessárias;
- Uma maior organização dos ficheiros;
- Menos código é escrito, uma vez que grande parte pode ser reutilizada;
- A criação de um protótipo é mais rápida, isto porque ter vários átomos antes da criação da página facilita a criação de modelos.

2.3.2.2 Desvantagens

- Pode demorar algum tempo para um desenvolvedor se habituar a esta metodologia;
- Caso não exista uma documentação, pode haver a criação de componentes desnecessários ou até mesmo repetidos;
- Uma pequena alteração em algum dos componentes que pode fazer sentido numa página, pode destruir esse componente que esteja implementado noutra página;
- A alteração de CSS de um componente numa página específica, é por vezes bastante difícil de realizar, sendo necessário recorrer a práticas não recomendadas, tal como o ”ng-deep”.[7]

2.4 Outras Metodologias *Front-End*

Para além da metodologia já referida e escolhida pela empresa, *atomic design*, existem diversas outras que permitem obter resultados semelhantes, mas com as suas próprias vantagens e desvantagens. De seguida vou exemplificar com duas metodologias que considero melhores e mais similares à que implementei durante o estágio na empresa.

2.4.1 *Progressive Enhancement*

O *progressive enhancement*[8] é uma metodologia muito utilizada para o desenvolvimento de aplicações web. A ênfase desta metodologia passa por desenvolver o essencial da página em primeiro lugar e só depois adicionar progressivamente mais camadas técnicas e de recursos de forma a melhorar a experiência do utilizador. Desta forma, permite a todos os utilizadores o acesso ao conteúdo básico e essencial, independente do *browser* utilizado e da sua capacidade de Internet.
O *progressive enhancement* é divido em 6 pilares principais:

- O conteúdo básico deve ser acessível em todos os navegadores web;
- As funcionalidades básicas devem ser acessíveis em todos os navegadores web;
- Todo o conteúdo deve ter uma marcação semântica;
- O *layout* melhorado deve ser providenciado por um CSS externo;

- As funcionalidades melhoradas são providenciadas por um não obstrutivo ficheiro de *JavaScript* externo;
- As preferências do navegador do utilizador devem ser respeitadas.

Desta forma, o principal objetivo do desenvolvimento de uma aplicação com *progressive enhancement* é providenciar funcionalidades básicas, estabilidade em todos os navegadores e dispositivos e uma experiência fluida ao utilizador antes de introduzir complexidade na página web.
Alguns dos principais motivos para o uso desta metodologia passam pela sua forte fundação, uma vez que o seu principal objetivo é o desenvolvimento do *core* da aplicação, e pela estabilidade nos diversos navegadores.

2.4.2 *BEM*

O BEM[9] é uma metodologia utilizada para nomear e organizar classes em *CSS*. O seu nome deriva dos seus 3 principais pilares: ***Block***, ***Element*** e ***Modifier***.
A implementação desta metodologia é relativamente simples, os *blocks* vão conter conjuntos de elementos que vai encapsular os estilos. Os *elements* vão funcionar como partes especificas de um componente. Por fim, os *elements* vão adicionar estilos a componentes específicos.

Um exemplo relativamente fácil de entender é o do seguinte excerto de código:

```
<div class="head">
  <div class="head__eye head__eye--left">(o)</div>
  <div class="head__eye head__eye--right">(o)</div>
</div>
```

Neste pequeno código, temos um *block* denominado *"head"* que contém dois *"elements"* chamados *"head__eye"* e cada um dos elementos tem um *"modifier"*, sendo este, por ordem, *"head__eye–left"* e *"head__eye–right"*. Como podemos ver, da maneira que o nome dos estilos estão organizados, é fácil entender qual pertence a qual, sendo que cada *"element"* começa por ter o nome do *"block"*, seguido de um nome específico para o *"element"*. E que cada *"modifier"* tem o nome do *"head"*, seguido do nome do *"element"* e, por fim, um nome específico para o *"modifier"*.

2.4.3 Comparação entre as 3 metodologias

De seguida é apresentada uma tabela que realiza uma comparação entre as duas metodologias apresentadas anteriormente e o *atomic design*, comparando os aspetos mais importantes.

Aspeto	***Atomic Design***	***Progressive Enhancement***	***BEM***
Foco	Criar um sistema de design estruturado com componentes reutilizáveis	Construir *websites* que melhoram progressivamente a experiência do utilizador	Criar uma convenção de nomenclatura que facilita a manutenção e escalabilidade do código CSS
Consistência	Assegura consistência de design em toda a interface	Assegura consistência funcional em diferentes ambientes	Assegura consistência na nomenclatura e estrutura do *CSS*
Escalabilidade	Altamente escalável com componentes modulares	Escalável através da adição progressiva de funcionalidades	Facilita a escalabilidade do CSS através de uma nomenclatura clara
Acessibilidade	Suporta, indiretamente, acessibilidade através da consistência	Foca-se diretamente na acessibilidade desde o início	Não se foca diretamente na acessibilidade, mas a estrutura clara pode ajudar
Implementação	Requer um entendimento profundo da metodologia	Requer um planeamento cuidadoso para assegurar que as funcionalidades básicas funcionam primeiro	Requer seguir rigorosamente a convenção de nomenclatura BEM
Manutenção	Manutenção mais fácil através da divisão dos componentes das páginas em cada uma das fases do *atomic design*	A manutenção pode ser complexa devido a múltiplas camadas de funcionalidade	Facilita a manutenção do CSS devido à nomenclatura clara e consistente

Tabela 2.1: Comparação entre *Atomic Design*, *Progressive Enhancement* e *BEM*

Através da tabela anterior podemos concluir que:
Atomic Design: Ideal para projetos que requerem um sistema de design consistente e escalável, tais como aplicações de grande escala e *websites* com interfaces de UI/UX complexas.
Progressive Enhancement: Melhor para projetos que precisam manter uma funcionalidade central e acessibilidade em diversos dispositivos e navegadores, tais como *websites* de serviços públicos e portais de informação.
BEM: Perfeito para projetos que precisam de uma estrutura de CSS clara e escalável, tais como grandes *websites* que exigem manutenção frequente.

2.5 Conclusão

Este capítulo começa por explicar a plataforma à qual vamos aplicar a metodologia de *Atomic Design*, bem como a sua arquitetura. De seguida é explicada a metodologia a ser utilizada neste projeto, bem como o que a define. São também explicadas duas metodologias que são bastante utilizadas em *front-end* para além da utilizada no decorrer deste estágio. Por fim, é apresentada uma comparação entre as 3 metodologias e quando devem ser utilizadas. Em termo de conclusão, apenas referir que embora o *Atomic Design* tenha sido criado para páginas web, este pode ser aplicado a qualquer tipo de interfaces, não só as da web.

Capítulo 3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

Neste capítulo vão ser referidas todas as ferramentas e diferentes tecnologias utilizadas durante a primeira parte do estágio. Foram utilizadas ferramentas que vão desde a plataforma que permite a testers reportarem bugs e a sua posterior atribuição a um desenvolvedor para serem corrigidos, ao editor e framework para escrever código e por fim, a ferramenta que permite o envio das alterações para o repositório a fim de serem verificadas e posteriormente adicionadas ao ambiente de produção.

3.2 Angular

Angular[10] é uma framework moderna, mantida pela google, que permite construir *Single-page Web Applications* utilizando typescript e html. Esta *framework* contem diversas características tais como *two-way binding, dependency injection e easy testing.* Tais características tornam o angular numa *framework* perfeita para construir aplicações web poderosas e em grande escala. Em adição, a sua arquitetura dividida em componentes e *templates* é ideal para implementar a metodologia de *Atomic Design.*

3.3 Visual Studio Code

O Visual Studio Code[11] é um editor de código desenvolvido pela Microsoft. Este editor contém suport para *debugging*, controlo de git embutido, *intelligent code completion*, *snippets* e *code refactoring.* O editor é também customizável a diversos níveis, permitindo alterar temas, mudar diversos atalhos de teclado e adicionar extensões que facilitam o trabalho no editor. O Visual Studio Code é grátis, open-source e utilizado para construir aplicações numa grande variedade de linguagens de programação tais como python, go e javascript.

3.4 Bitbucket

O Bitbucket[12] é uma aplicação web que oferece serviços de hospedagem de controlo de projetos. Faz parte dos serviços fornecidos pela Atlassian, e é utilizado em projetos que utilizam Mercurial ou Git. O seu uso principal é para permitir aos desenvolvedores hospedar e gerir os repositórios de código, no entanto, pode também guardar mais tipos de documentos.

3.5 Jira

O Jira[13] é outra plataforma desenvolvida pela Altassian que permite gerir diversos projetos, sendo utilizada para planear, monitorizar e lançar software. Algumas das principais características desta plataforma incluem:

- Monitorização de bugs, novas funcionalidades e outras tarefas;
- Dashboard para acompanhar o desenvolvimento do projeto;
- Workflows personalizáveis;
- Ferramentas de revisão de código. O Jira foi utilizado para planear sprints e distribuir tarefas pelas diferentes equipas de desenvolvedores presentes na empresa.

3.6 Conclusão

Neste capítulo foram apresentadas todas as ferramentas utilizadas durante o estágio. Embora algumas destas tecnologias me fossem desconhecidas, foram de fácil aprendizagem e aquisição. Tais conhecimentos serão benéficos para todo o trabalho futuro.

Capítulo 4

Engenharia de Software

4.1 Introdução

Neste capitulo vai ser apresentada uma simples arquitetura da aplicação web da empresa, bem como alguns requisitos funcionais e não funcionais das páginas web desenvolvidas durante o decorrer do estágio.

4.2 Engenharia de Requisitos

Uma vez que foram desenvolvidas diversas página, sendo todas do mesmo módulo, apenas estão apresentados requisitos que sejam comuns a todas as páginas criadas.

4.2.1 Requisitos Funcionais

Tabela 4.1: Requisitos Funcionais da Aplicação

Requisitos Funcionais	
RF	**Descrição**
1	A aplicação deve ter uma navegação clara e intuitiva.
2	O utilizador deve conseguir navegar entre diferentes secções e páginas facilmente.
3	O conteúdo deve ser organizado e apresentado de forma visualmente apelativa.
4	A validação de formulários deve ser implementada para garantir que os utilizadores fornecem dados válidos.
5	Deve ser fornecido *feedback* ao utilizador sobre o sucesso do envio de formulários ou mensagens de erro.
6	As mensagens de erro apresentadas devem ser provenientes do pedido de submissão por parte do *backend*.
7	Em caso de o pedido não ter uma mensagem, de erro ou sucesso, uma de *default* deve ser apresentada.
8	Os elementos nas diversas páginas devem ser interativos.
9	Os dados devem ser apresentados de forma significativa e compreensível, usando gráficos e tabelas.
10	As páginas devem comunicar corretamente com o *backend* em todas as ações necessárias.
11	Cada ação realizada pelo utilizador que necessite de dados do *backend* deve ser apresentado um ícone de carregamento.
12	A aplicação deve ser responsiva para os três tamanhos de tela mais comuns.
13	Campos de preenchimento obrigatório devem ser facilmente visíveis.
14	As diferentes páginas devem seguir o design criado pela equipa de *UI/UX*.
15	Se não houver design disponível para a página, deve ser similar à antiga, utilizando os novos componentes.

4.2.2 Requisitos Não Funcionais

Tabela 4.2: Requisitos Não Funcionais da Aplicação

Requisitos Não Funcionais	
RNF	**Descrição**
Compatibilidade	A aplicação deve executar em navegadores à base de *chromium*, incluindo *Google Chrome*, *Microsoft Edge* e outros navegadores derivados.
Tolerância a falhas	A aplicação deve ser capaz de lidar com falhas no *backend*, exibindo mensagens de erro adequadas.
Responsividade	O *layout* da aplicação deve-se adaptar a diferentes tamanhos de ecrã.
Usabilidade	A plataforma deve ser de fácil utilização pelo utilizador.
Manutenibilidade	O código deve ser bem comentado e seguir padrões do *atomic design*, bem como as melhores normas de programação em *Angular*.

4.3 Arquitetura do Sistema

Como já foi referido, a aplicação web da EMVENCI utiliza uma arquitetura *multitenancy*, ou seja, dentro da mesma aplicação existem dados guardados para cada *tenant*, representado por um *"id"* único. A arquitetura da plataforma pode ser divida nas seguintes partes:

- **UPI:**
  A plataforma principal, utilizada por cada um dos *tenants*. Representa o *frontend*.

- **eAPI:**
  Aqui o tráfico é filtrado de modo a prevenir ataques, sendo que todos os pedidos aos eGRC passam obrigatoriamente por aqui.

- **eGRC:**
  Local onde existe toda a lógica da aplicação, representando desta forma o *backend*. Única parte da arquitetura com acesso à base de dados

- **UPI-Public:**
  A plataforma que permite aos utilizadores responder a questionários e aceder a diversos conteúdos de *phishing*/*eLearning*. Encontra-se fora da plataforma principal

- **MariaDB:**
  Base de dados utilizada pela empresa. Local onde os dados são guardados, sendo que apenas o egrc comunica com a mesma para enviar esses dados para o frontend.

De seguida é apresentada uma imagem que representa simplificadamente como funciona a aplicação.

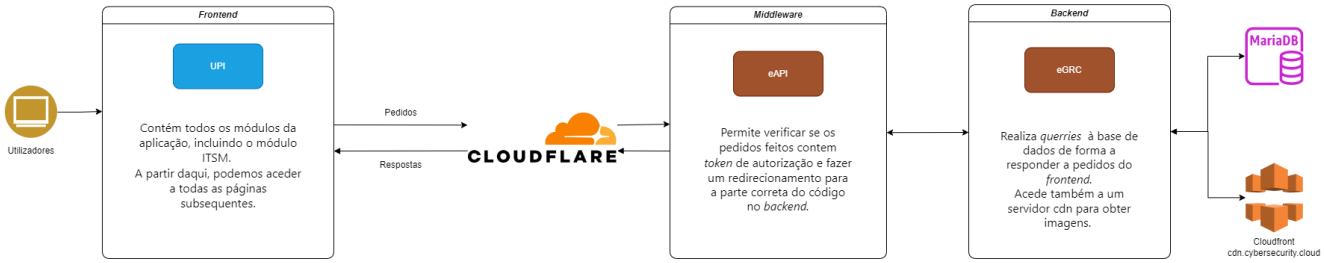

Figura 4.1: - Arquitetura da Aplicação

4.4 Modelação

Os seguintes diagramas *UML* apresentados vão ser referentes apenas a uma das páginas criadas durante o decorrer do estágio, uma vez que grande parte do funcionamento base é semelhante entre todas. Desta forma, a página escolhida foi a dos *Services* 6.2.2.

4.4.1 Diagrama de Caso de Uso

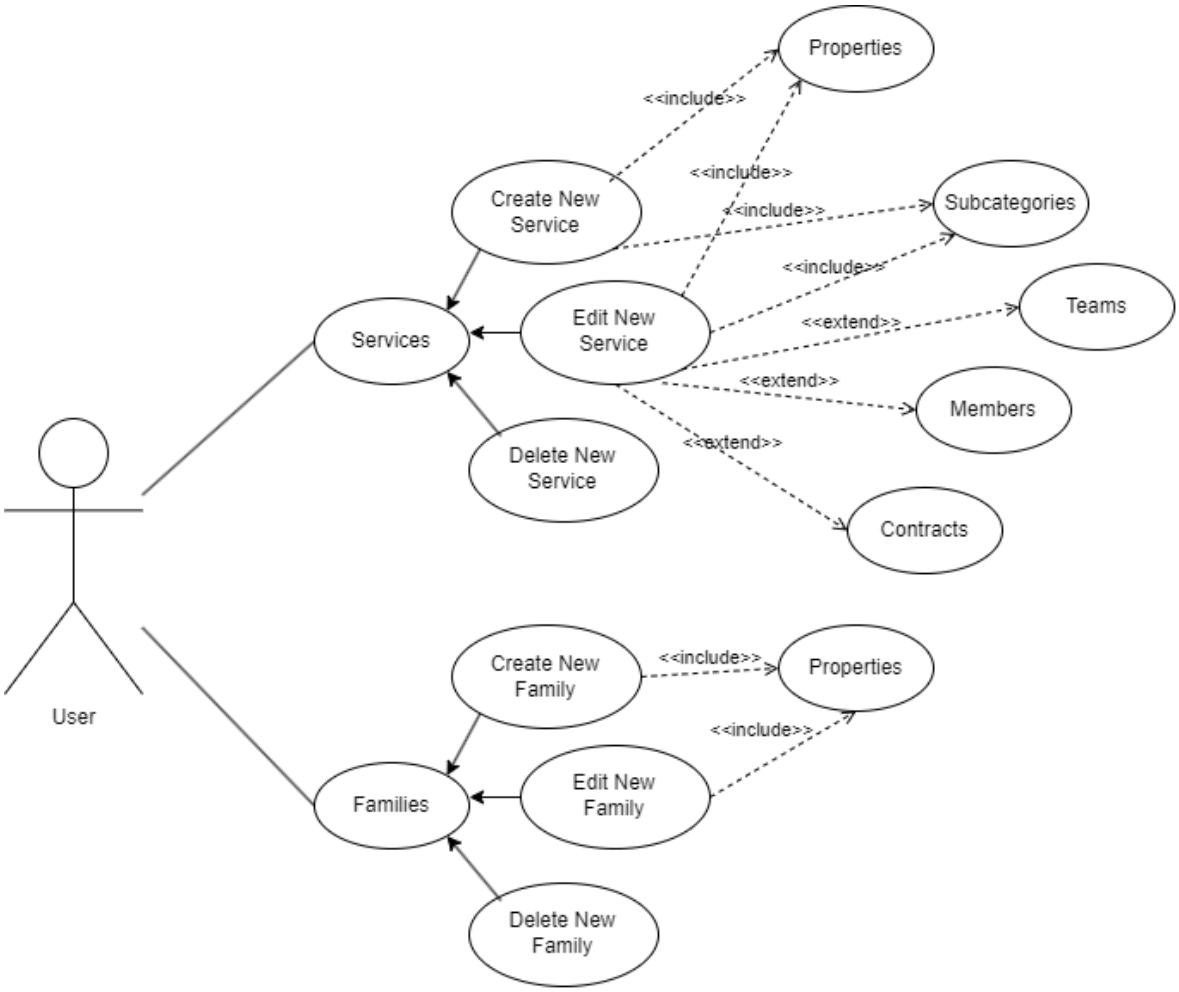

Figura 4.2: - Diagrama de Caso de Uso na página *Services*

Ator: Utilizador que acede à pagina dos *Services* na plataforma.
Descrição: O diagrama demonstra a interação entre o utilizador e duas areas dentro da página dos *services*, ***services*** e ***families***. Cada uma das areas apresenta ações específicas que o utilizador pode realizar

4.4.2 Diagrama de Atividade

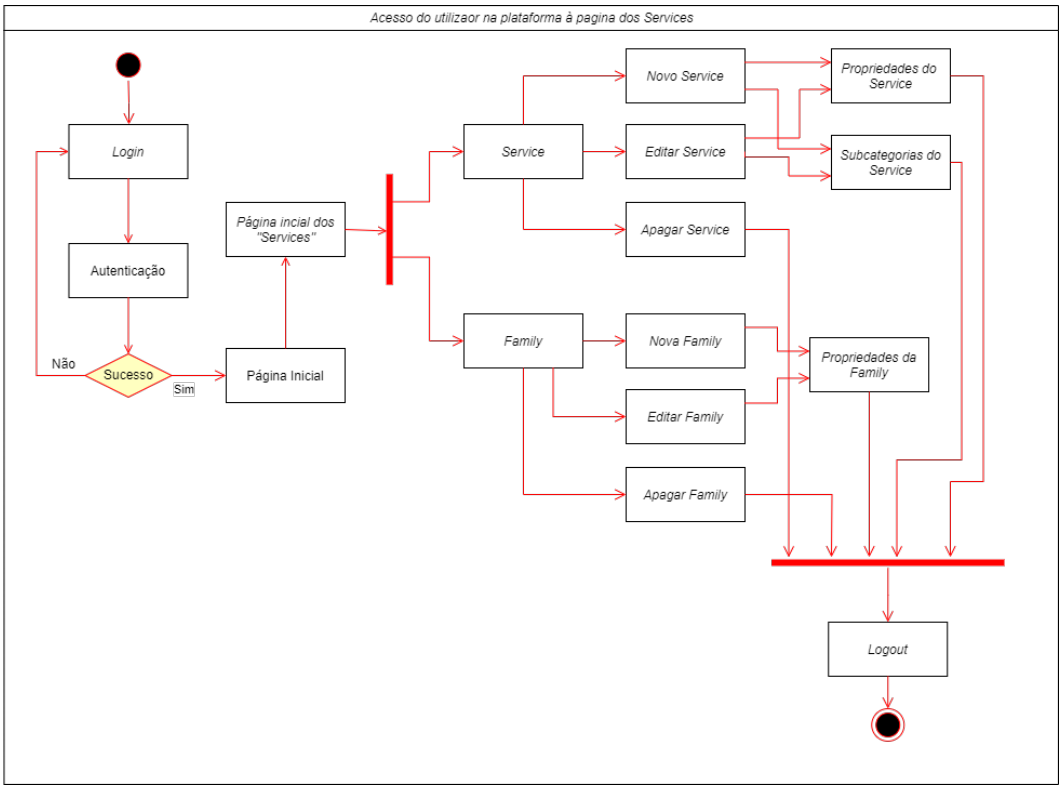

Figura 4.3: - Diagrama de Atividade na página *Services*

O anterior diagrama de atividade fornece uma representação visual do fluxo de interação do utilizador na página dos *services*. O processo começa com o utilizador a fazer *login* e a ser autenticado. Após autenticação bem-sucedida, o utilizador é direcionado para a página inicial e a partir daí acede à pagina dos *services*. Na página de serviços, o utilizador pode realizar várias ações relacionadas com *services* ou *families*, que vão desde criar um objeto novo, edita-lo ou mesmo apaga-lo. Por fim, durante qualquer momento de navegação na página o utilizador pode fazer *logout*.

4.4.3 Diagrama de Sequência

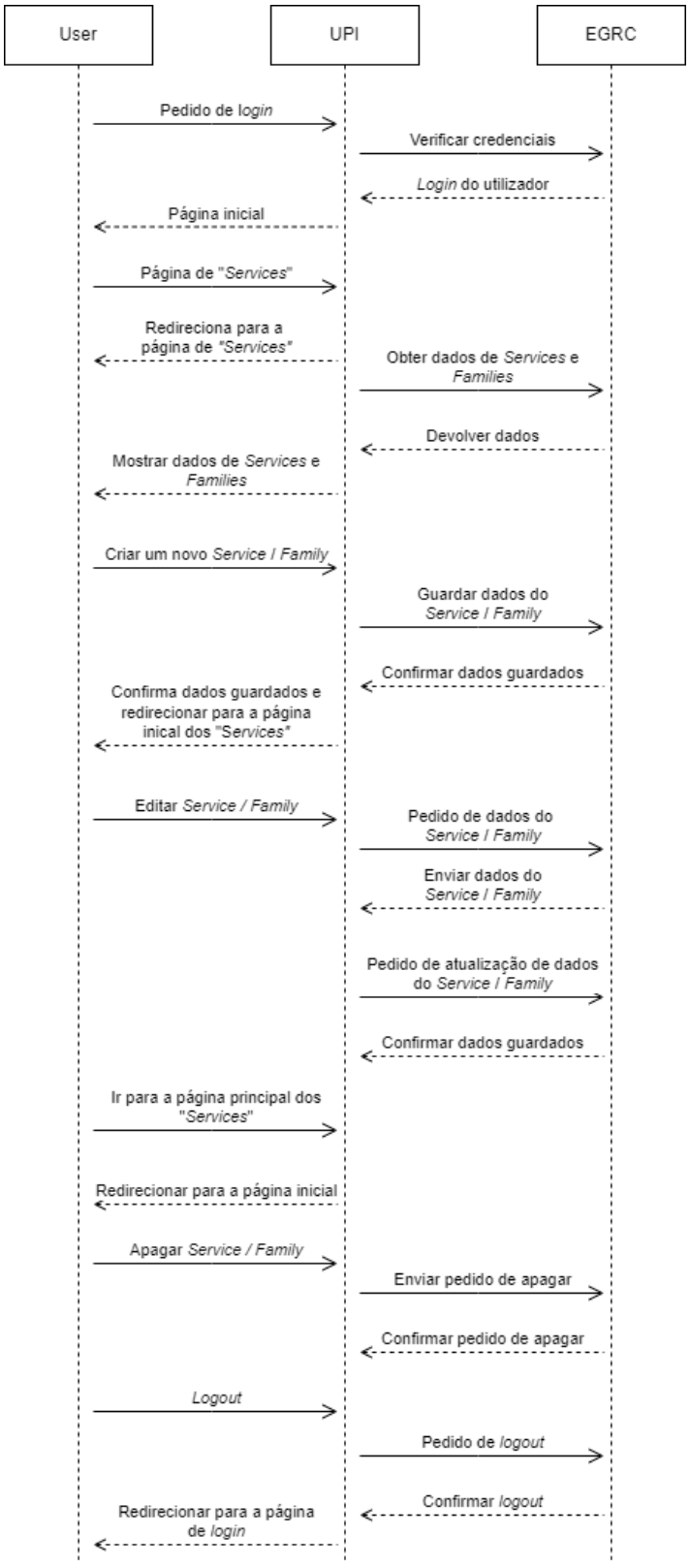

Figura 4.4: - Diagrama de Sequência na página *Services*

No diagrama de sequência são apresentadas as interações entre o utilizador (User) com o *frontend* da aplicação (UPI) e com o *backend* (EGRC). As interações representam o *login* do

utilizador que o leva à pagina dos *services* onde pode cria um novo objeto do tipo *services* ou *family*. Depois do objeto ser criado e guardado, o utilizador escolhe editar o mesmo, sendo que depois o apaga. Por fim, faz *logout* da aplicação.

4.5 Conclusão

Neste capítulo foram apresentados diversos requisitos funcionais e não funcionais referentes às páginas reconstruidas durante o estágio. Em adição, foi também apresentada uma arquitetura do sistema utilizado pela empresa de forma simplificada.

Capítulo 5

Implementação

5.1 Introdução

Neste capítulo vão ser referidas as diferentes tarefas que foram realizadas durante todo o estágio, bem como uma explicação da sua execução.

5.2 Roadmap de Tarefas

As tarefas apresentadas no mapa de Gantt apresentam os diferentes objetivos realizados na empresa pelo estagiário durante os quase nove meses de estágio, sendo que esta teve início dia 26 de setembro de 2022 e finalizou dia 15 de Junho de 2023. O mapa encontra-se cronologicamente organizado, sendo que no início do estágio foi realizado um *onboarding* juntamente com uma familiarização das várias plataformas utilizadas pela empresa. Após o término do *onboarding* foram realizados alguns *bug fixings*, sendo que após uma certa familiarização com o código e arquitetura em *Atomic Design* já existente, existiu um avanço para o estudo, bem como construção de algumas páginas utilizando a metodologia de *Atomic Design*. Em fins de dezembro e início de janeiro, foi realizada a escrita do relatório preliminar e a continuação do desenvolvimento de mais algumas páginas da aplicação web. Em fevereiro, foi-me revelado pela empresa o modulo da aplicação que seria realizado apenas por mim para efeitos deste relatório de estágio. O modo em causa denomina-se ITSM, sigla para "*IT Service Management*". Este modulo é composto por diversas páginas, sendo que foram feitas, ao longo de 4 meses e meio, um total de 5 páginas principais. Destas páginas quase todas incluíam página para mostrar dados, página para criar e editar. Por fim, foi-me dado pela empresa uma página final fora do modulo do *ITSM*, mais complexa do que as que tinha criado anteriormente de modo a testar o meu aprendizado e habilidade com o *Atomic Design* ao longo do meu tempo de estágio na empresa.

Figura 5.1: - Tarefas realizadas durante o estágio

5.3 Implementação das Diferentes Tarefas

5.3.1 Tarefa 1 – *Onboarding*

O principal objetivo desta tarefa foi de introduzir e integrar o estagiário na empresa e na metodologia de trabalho. Para tal foi realizada a construção de um pequeno website utilizando já a metodologia do *Atomic Design*. Para ajudar o estagiário a concluir esta tarefa inicial, foi providenciado um documento para servir de guia e permitir que não só a construção do website saísse correta bem como, ajudar na integração do estagiário nas plataformas do Bitbucket e do Jira.

5.3.2 Tarefa 2– Familiarização com a plataforma

Nesta tarefa, que decorreu ao mesmo tempo da tarefa 1, o objetivo foi perceber o funcionamento de todas as plataformas da empresa, necessárias para o trabalho do estagiário. Dentre estas, inclui as plataformas Jira, Bitbucket, VSCode e a ferramenta Git. Esta tarefa foi concluída devido tanto ao trabalho inicial na tarefa 1, bem como o trabalho de correção de bugs na tarefa 3.

5.3.3 Tarefa 3 – Correção de Bugs

Nesta tarefa foram realizadas diversas correções de bugs no front-end da aplicação web da empresa EMVENCI. Os bugs foram relativamente fáceis no início, de forma a que o estagiário

se entrosasse com as diversas plataformas necessárias, até a bugs mais complicados numa fase final. Esta fase serviu também para o estagiário visualizar e perceber a implementação de alguma metodologia de *Atomic Design*, organização do código e diferenças entre a arquitetura antiga e a mais recente.

5.3.4 Tarefa 4 – Estudo da Metodologia *Atomic Design*

Esta tarefa foi iniciada aquando do contacto do estagiário com algum código já realizado em *Atomic Design*. Durante esta tarefa foi realizado um estudo das diversas características da metodologia e as suas implementações em código real.

5.3.5 Tarefa 5 – Construção de Páginas utilizando *Atomic Design*

A quinta tarefa está relacionada com a construção de algumas páginas usando a metodologia do *Atomic Design*. Para tal, houve um estudo das funcionalidades presentes na arquitetura antiga para serem implementadas no código novo. Em adição, para além da remodelação de código utilizando a nova metodologia, existiu uma alteração ao nível do design, já criado pela equipa de UI/UX da empresa para ser, não só mais apelativo, mas também para permitir uma das características do *Atomic Design* que é a reutilização de código.

5.3.6 Tarefa 6 – Escrita do Relatório do Projeto de Estágio

Esta tarefa, remete para a escrita do relatório preliminar que embora tenha sido apenas começado a escrever em dezembro, foi realizado ao longo de todo o estágio a toma de notas que auxiliaram na escrita final do mesmo.

5.3.7 Tarefa 7 – Estudo da Modulo *ITSM*

Esta tarefa foi realizada para permitir entender como funciona o modulo do *ITSM*, as interações entre as diferentes páginas, os componentes necessários criar e os necessários a utilizar já criados. Após esta avaliação, foi escrito um pequeno relatório a descrever as interações, bem como a ordem pela qual achava que as páginas deviam ser construidas, com base nas suas diferentes interações e dificuldade. Por exemplo, existe uma página denominada "*Services*" que precisava de dados obtidos através de outra página, "*Teams*". Deste modo, não fazia sentido começar na página dos "*Services*" que precisava dessa página. Desta forma, foi determinada a ordem das páginas a serem criadas, sendo esta:

- *Teams - Search*
- *Teams - New/Edit*
- *Services - Search*
- *Services - New*
- *Services - Edit*
- *Services - Family New/Edit*

- *Requests - Search*
- *Requests - Dashboard*
- *Requests - New*
- *Time Tracker - Search*
- *Time Tracker - Modal*

5.3.8 Tarefa 8 – Reestruturação de páginas dentro do modulo *ITSM*

A reestruturação das páginas do modulo *ITSM* incluiu a criação de componentes atómicos, a reutilização de alguns já criados e, por vezes, a renovação de alguns componentes para atender a certas mudanças de design na plataforma. De certo modo, algo comum a todas as páginas foi a criação de um *template* e de um ficheiro *typescript* próprio onde se exportava uma classe com *setters* e *getters* referente à página a ser criada, bem como, métodos com chamadas à API de forma a obter dados. Um exemplo de uma dessas classes é a seguinte:

```
export class Teams{
    protected _teamProps = {} as TeamProps;
    protected _apiService = APIHttpClientService.instance;

    constructor( private teamProps?: TeamProps){
        this._init(this.teamProps)
    }

    public get id(): number {
        return this._teamProps.id;
    }

    public set id(value: number){
        this._teamProps.id = value;
    }

    public get name(): string {
        return this._teamProps.name;
    }

    public set name(value: string){
        this._teamProps.name = value;
    }

    //rest of the setters and getters
    .....

    private _init(teamProps?: TeamProps): void{
        if(teamProps == null) return;
```

```
30
31          this.id = teamProps.id;
32          this.name = teamProps.name;
33          ...
34      }
35
36      public static getTeams<T extends Teams>(this: new (props?: any) => T):
        Promisse<Array<T>> {
37          return new Promise((resolve, reject) => {
38          const context = new this();
39          context._apiService.get(`/itsm/teams`)
40          .then((data:any) =>{
41              const teams = data.data.map((props) => new this(props));
42              resolve(teams);
43          })
44          .catch((error) =>{
45              reject(error);
46          });
47      });
48      }
49      //rest of the api calls
50      ....
51  }
```

Listing 5.1: Exemplo de uma class do objecto teams com getters, setters e cahamdas à API

Através destas classes conseguia criar objetos que melhoravam significativamente a leitura e maneabilidade do código. Em adição, a estruturação ficava muito melhor e mais legível para quem venha posteriormente modificar/ver o código.

Por fim, algo comum a quase todas as classes criadas é um ficheiro *typescript* com a interface a ser usada na classe. Dentro deste ficheiro existe sempre uma interface e por vezes *enums*. O seguinte código representa um simples exemplo referente ao código apresentado anteriormente:

```
1   export interface TeamProps{
2       id?: number,
3       name?: string,
4       ...
5   }
6
7   export enum TeamStatus{
8       Disabled = 'disabled',
9       Enabled = 'enabled'
10  }
```

Listing 5.2: Exemplo de uma class do objecto teams com getters, setters e cahamdas à API

5.3.9 Tarefa 9 – Construção da página final dentro do estágio na empresa

Meio que como teste final, foi-me atribuído pela empresa a reconstrução de uma página final, já fora do módulo do *ITSM*, mais complexa. Esta página foi a "Templates", que tinha uma página de entrada tipo "*search*" e depois bifurcava em duas páginas de criação/edição. Uma dessas páginas era o "*email/sms*" que permitia criar ou editar *templates* de email ou sms para os utilizadores poderem utilizar noutra página. Do mesmo modo, a outra página, denominada "*landing pages*", permitia fazer o mesmo, só que em vez de ser para "*emails/sms* é para páginas web. Esta página foi a que me permitiu demonstrar tudo o que aprendi de *Atomic Design* e programação *front-end* ao longo do estágio, mas também me permitiu melhorar os meus conhecimentos e habilidade como programador.

5.4 Conclusão

Neste capítulo foram apresentadas as diferentes tarefas realizadas durante o decorrer de todo o estágio, essencial para compreender o trajeto percorrido ao longo dos 9 meses na empresa. A conclusão com sucesso de todas as tarefas foi essencial para o bom decorrer do estágio, bem como para a escrita deste presente relatório.

Capítulo 6

Desenvolvimento

6.1 Introdução

Neste capítulo, vai ser explicado como foi feito o desenvolvimento das diferentes páginas web criadas, com uma breve explicação e uma imagem com o resultado final. Em adição, alguns dos exemplos vão incluir uma imagem com a página antiga para termos de comparação.

6.2 Desenvolvimento das diferentes páginas web

Durante a segunda fase do estágio, como já foi referido, foi desenvolvido um modulo da aplicação web definido pela empresa, ITSM. Este módulo é composto por diversas páginas que interagem entre si, ou seja, o dados criados em certas páginas vão ser necessários para o bom funcionamento e criação de objetos em outras páginas. Como exemplo, o objeto "*service*" criado na página com o mesmo nome, na tab "*services*", é obrigatório para a criação de um objeto "*request*". Deste modo, a ordem da construção das páginas teve em consideração não só o que seria necessário nas páginas, mas também o nível de dificuldade. Por estes motivos segui a ordem de construção de páginas mais simples e que não necessitassem de outras páginas primeiro, e as mais complexas e com dependências a outras páginas posteriormente.

6.2.1 Teams

A primeira página criada foi a referente ao objeto "*teams*", que conta com uma página onde são apresentadas as "*teams*" já criadas e uma página que permite criar um novo objeto ou editar um já existente. O acesso à criação e edição de uma "*team*" acontece dentro da página inicial com um botão que leva para a página de criação e uma tabela onde mostra os objetos já criados com 3 opções. Uma que permite editar, que nos leva para a página de criação mas com os dados da "*team*" e vai apenas alterar esse objeto sem criar um novo. Outra opção para eliminar um team que vai abrir um "*modal*" que permite a confirmação de apagar para sempre o objeto selecionado. Por fim, existe também a opção de ver as relações que uma certa "*team*" tem, ou seja, em que outros objetos é utilizada aquela equipa. No caso do objeto ter *relations*, ao clicar nessa opção, é aberto um modal com uma tabela que revela informações sobre onde o objeto é utilizado. Por regra, um objeto com relações não pode ser apagado e por isso apenas uma desses duas opções é apresentada ao utilizador.

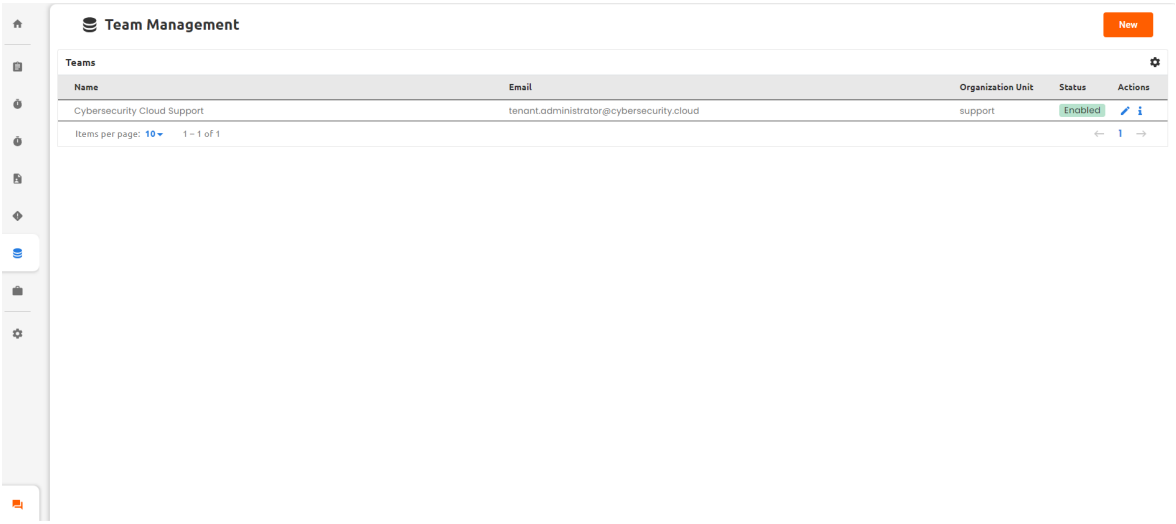

Figura 6.1: - Nova Primeira Página "*Teams*"

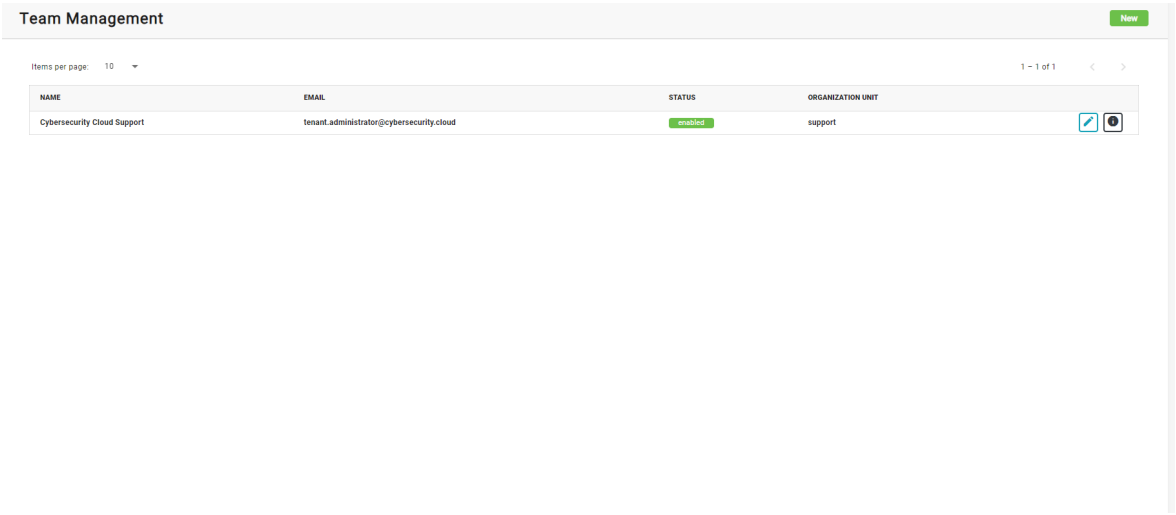

Figura 6.2: - Antiga Primeira Página "*Teams*"

Na pagina de criação/edição, encontramos alguns componentes atómicos já criados previamente na empresa, um de input de texto/números, um de selecionar entre várias opções e outro componente com um *search* que permite abrir um *modal* com um tabela que mostra diversas opções para o utilizador poder escolher uma. Estas opções são referentes à área das "*Properties*". Na área das "*Persons*" existe uma tabela que demonstra as pessoas adicionadas à *team* que está a ser criada/editada e um botão de adicionar que abre um modal que permite adicionar *persons* à dita tabela. Nas opções da tabela é apenas apresentada a de remover pessoas. Nos componentes apresentados, apenas os com asterisco a vermelho são obrigatórios de serem preenchidos pelo utilizador, sendo que os campos de *Organization Unit* e *Status* vêm já preenchido por *default* na criação de uma nova "*team*" com a primeira *organization*, criada anteriormente noutra página pelo utilizador, e o campo de *status* vem já defendido como "*Enabled*". No caso dos campos obrigatórios não estarem preenchidos, o botão de guardar fica desativado até serem preenchidos. No fim, o utilizador carrega no botão de *save* para criar/editar o objeto. Se estiver a criar o objeto, é redirecionado para a página

inicial e existe uma pequena *snackbar* a indicar que o objeto foi criado. Se estiver a editar, a *snackbar* também aparece mas permanece na mesma página. No caso de ocorrer algum erro durante o *save* da *team*, é apresentado o erro na *snackbar* e o utilizador permanece na mesma página para o poder corrigir e assim gravar corretamente.

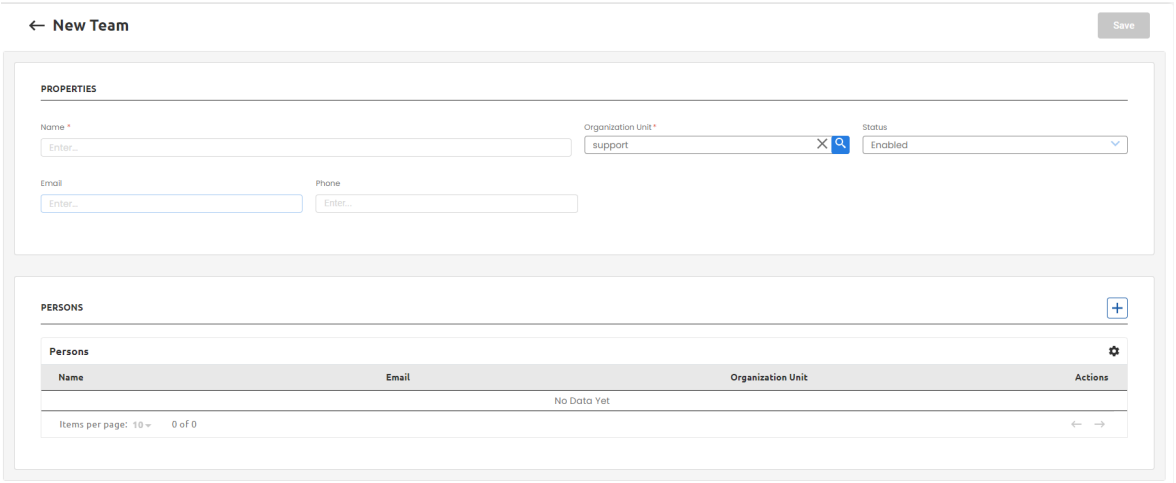

Figura 6.3: - Nova Página de Criação/Edição "*Teams*"

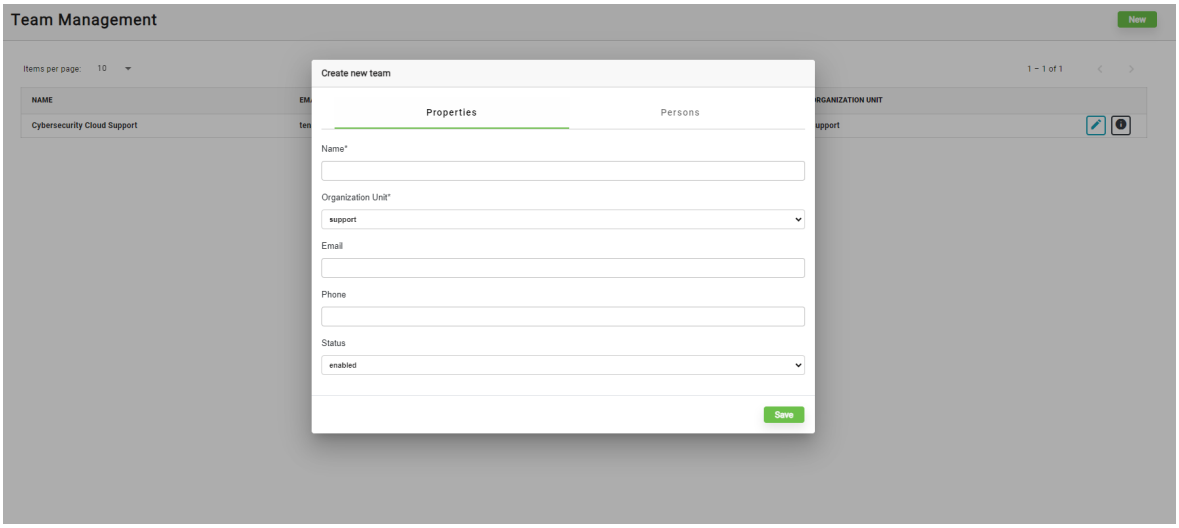

Figura 6.4: - Antiga Página de Criação/Edição "*Teams*"

6.2.2 Services

A pagina inicial dos "*services*" corresponde a um componente "*navigation*" com duas tabs que permitem ver duas tabelas distintas com valores de *services* ou *families*. Em adição, existe um botão que permite criar um dos dois objetos dependendo da tab selecionada. Em ambas as tabelas, as ações permitem eliminar um objeto criado ou ir para uma página que o permite editar.

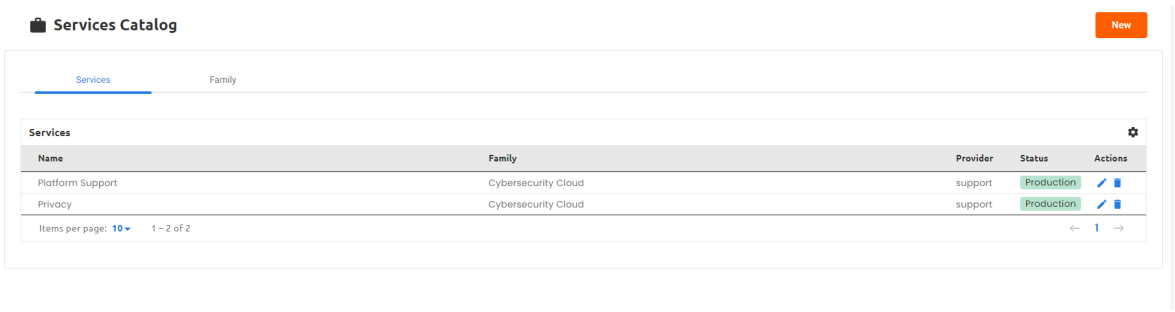

Figura 6.5: - Nova Primeira Página "*Services*"

De seguida, a página de criação e de edição de um *service* são similares, sendo que durante a criação temos acesso a apenas duas tabs ("*Properties*" e "*Subcategories*") com apenas dois campos obrigatórios. *Name*, que é um simples campo de inserção de texto e *provider*, que é um componente que nos permite selecionar um objeto dentro de vários apresentados. Na *tab* das "*Subcategories*" temos uma tabela onde são apresentadas as subcategorias criadas, bem como um botão que abre um *modal* que nos permite criar um objeto que vai ser adicionado à tabela. Durante a edição de um objeto "*service*" temos acesso a mais três *tabs*, sendo que são todas compostas por uma tabela e um botão que permite selecionar dados, previamente criados noutras páginas respetivas a cada objeto. Ou seja, na tab das *Teams* podemos adicionar um objeto criado na página referida anteriormente neste relatório, na tab dos *Members*, podemos adicionar objetos "*persons*", criados na sua página específica. Por fim, na tab dos *Contracts*, podemos adicionar contratos, que são também criados na sua própria página.

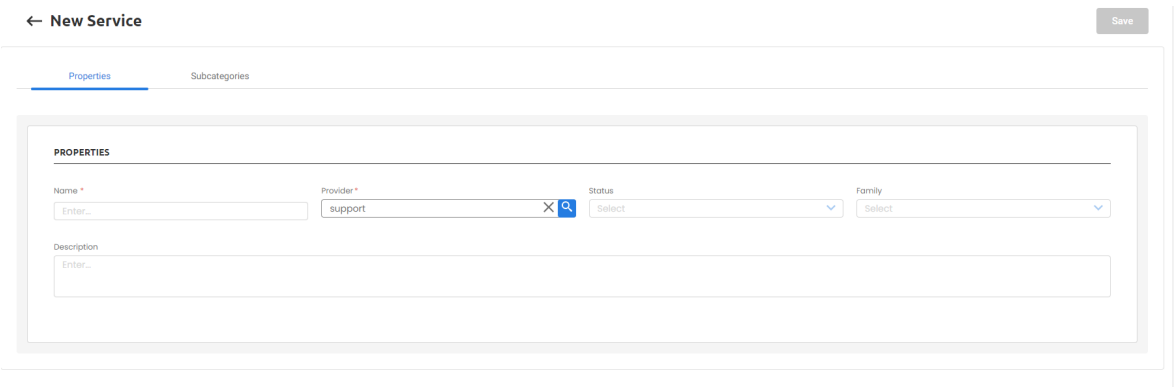

Figura 6.6: - Nova Página de Criação "*Services*"

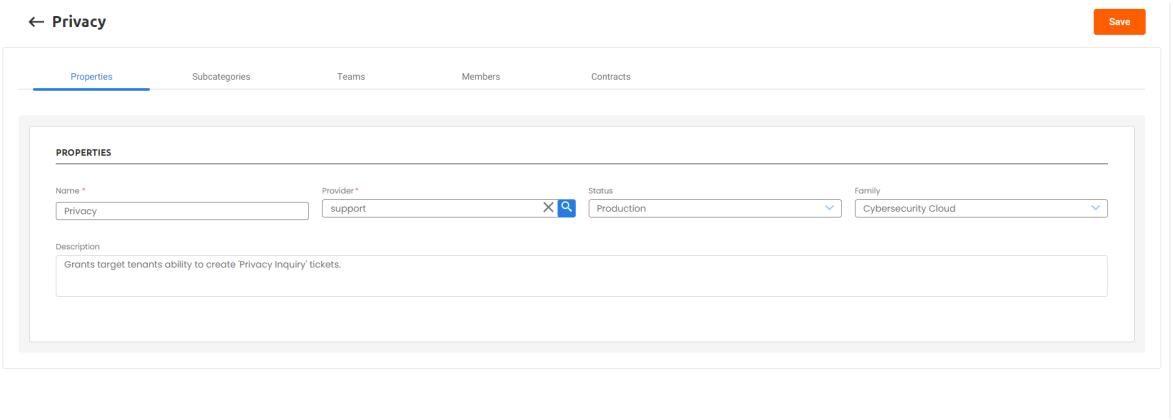

Figura 6.7: - Nova Página de Edição "*Services*"

Por fim, através da página inicial dos *Serives*, podemos também criar um objeto "*family*". A página de criação é relativamente simples, com um campo de inserção de texto obrigatório, um não obrigatório e uma inserção de imagem que vai servir de *icon*. Durante a edição de um objeto deste tipo, podemos aceder a uma nova tab, *Serives*, que apenas contem um tabela com todos os serviços diferentes a que a "*family*" criada está associada.

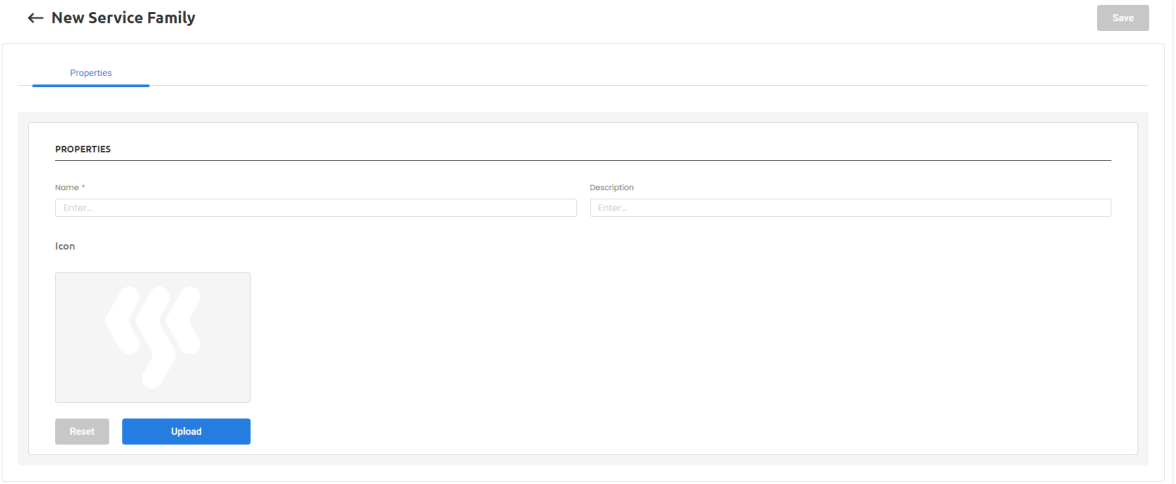

Figura 6.8: - Nova Página de Criação "*Family*"

6.2.3 Requests

A primeira página dos *requests* corresponde ao "*search*", ou seja, a página onde vamos poder ver/procurar entre todos os *requests* criados. Nesta página temos uma tabela com os diversos dados relativos a um objeto do tipo *request*, com apenas as opções de editar ou apagar o objeto criado. Em adição, no lado direito da página temos um conjunto de filtros que ajudam o utilizador a encontrar o *request* específico. Em adição com um tamanho de tela menor (1366px), os filtros deixam de ocupar o lado direito e passam a botão que cria um *dropdown* com os mesmo filtros. Por fim, temos um botão que nos leva para a página de criação de um novo *request*.

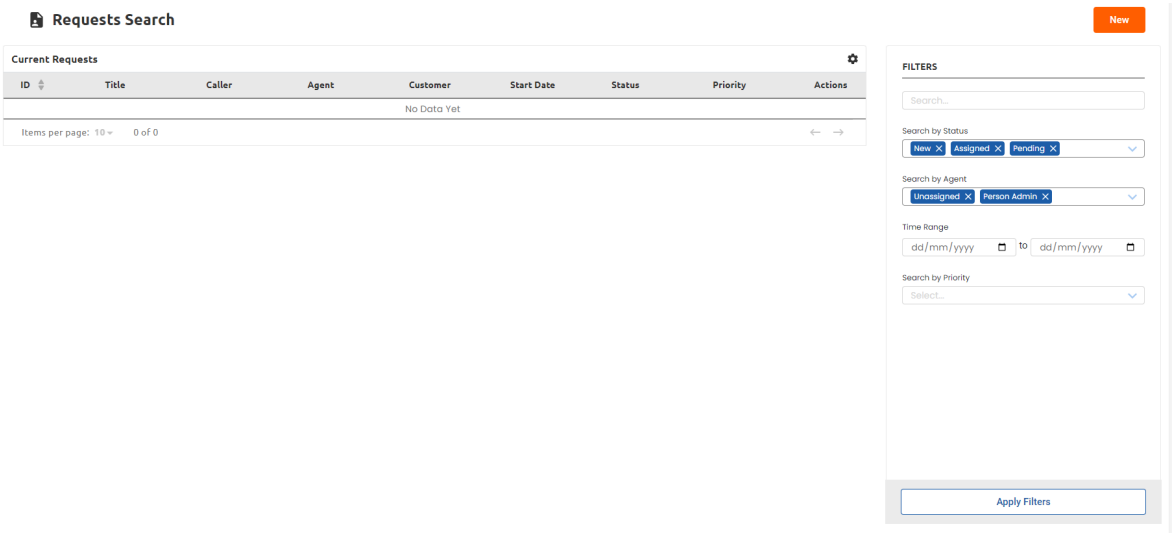

Figura 6.9: - Nova Página de Visualização de "*Requests*"

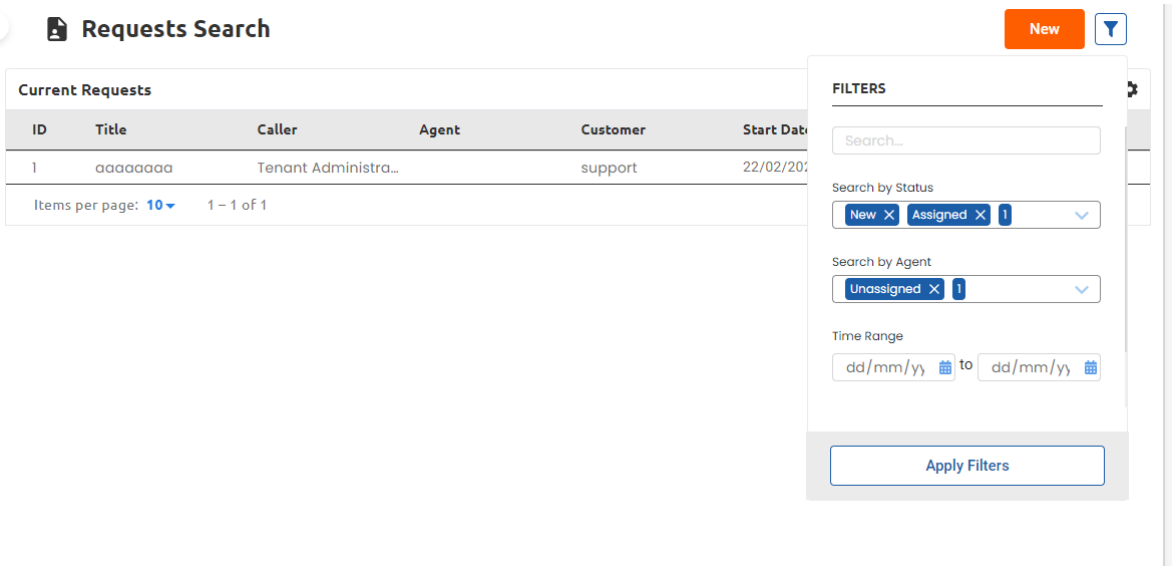

Figura 6.10: - Nova Página de Visualização de "*Requests*" (filtros através do botão)

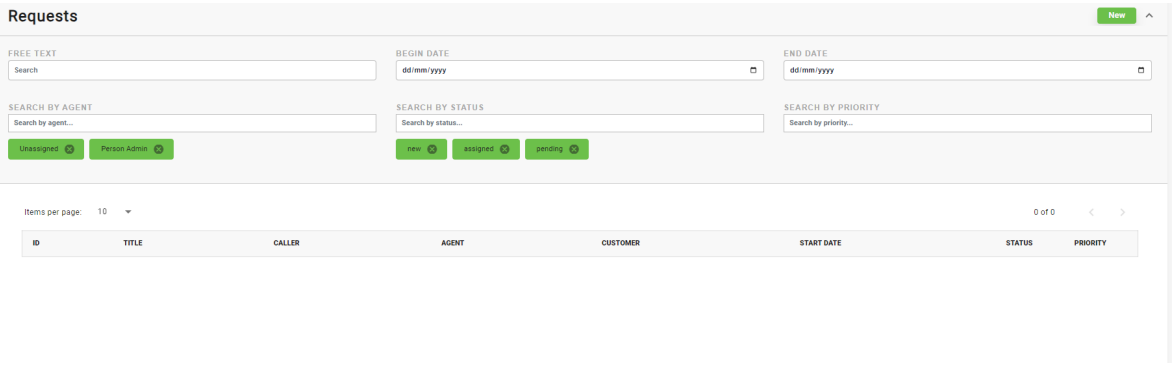

Figura 6.11: - Antiga Página de Visualização de "*Requests*"

Outra página criada em relação ao objeto *requests* foi a do *"Dashboard"*, uma simples página com diversas informações acerca de todos os *requests* criados. Esta página é composta por

um contador do número de *requests* abertos e mais quatro gráficos com diversas informações. Como podemos ver na imagem seguinte 6.12, quando não temos dados, a página é apresentada desta forma ao utilizador. Em adição, esta página contem ainda mais um botão que redireciona o utilizador para a página de criação de um novo *request*.

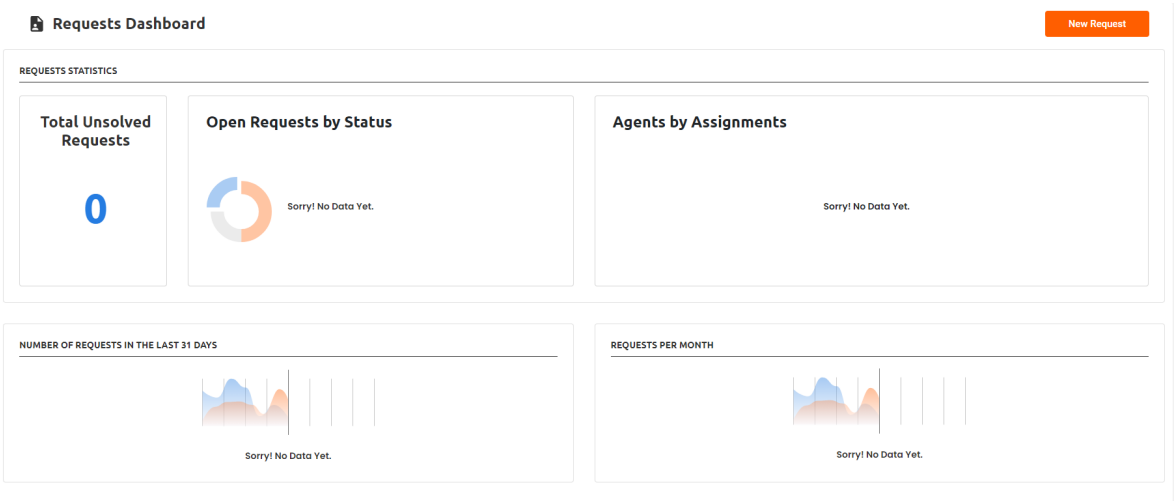

Figura 6.12: - Nova Página de *Dashboard* de "*Requests*"

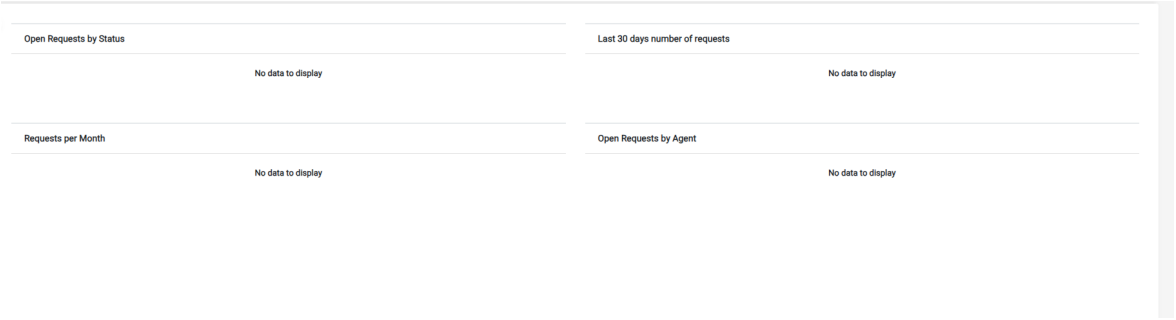

Figura 6.13: - Antiga Página de *Dashboard* de "*Requests*"

A terceira e última página referente ao objeto *request*, é a criação do mesmo. Esta página é composta por uma navegação com duas partes, uma com acesso às propriedades e outra, opcional, com a possibilidade de adicionar entidades diversas. Os únicos dados obrigatórios são, a inserção de texto com o título do *request*, um selecionador de uma *organization unit*, outro selecionador com a pessoa que reportou o *request* e uma descrição do mesmo. Por fim, é também necessário a seleção de um serviço, criado na página dos *services*6.6, bem como de uma subcategoria relacionada ao serviço criado. Os restantes campos, bem como as entidades, são opcionais na criação de um objeto *request*.

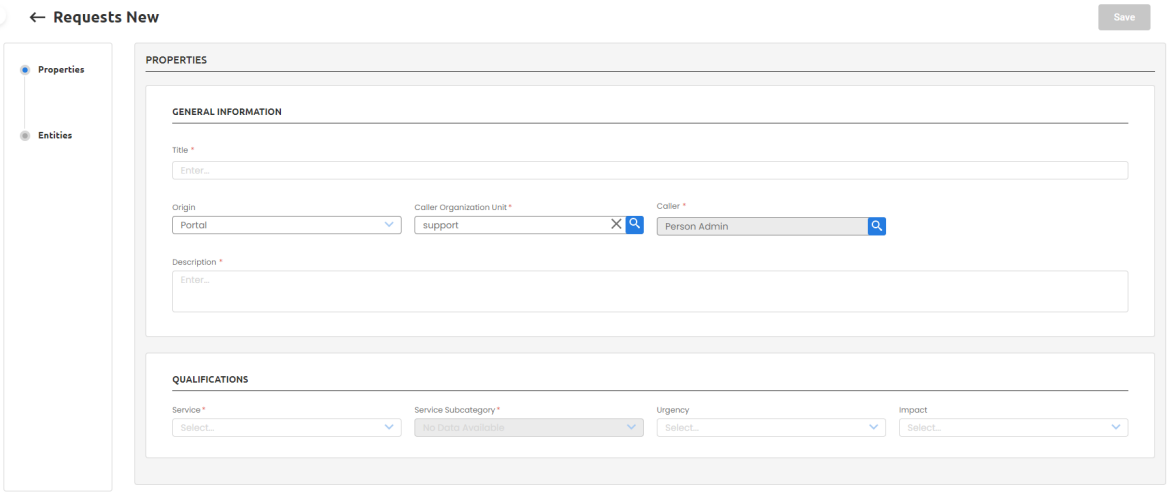

Figura 6.14: - Nova Página de Criação de "*Requests*"

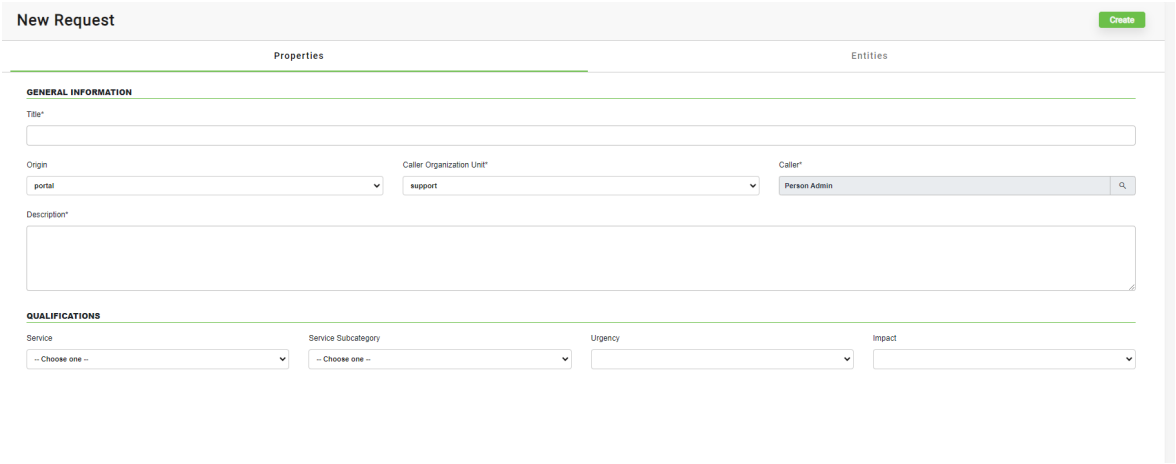

Figura 6.15: - Antiga Página de Criação de "*Requests*"

6.2.4 Time Tracker

O *"time tracker"* é composto por uma página de visualização e um modal que permite a sua criação. Este objeto é relativamente complexo, uma vez que depende de uma variedade de outros para a sua criação, Por exemplo, para criar um *time tracker* do tipo *request*, é necessário adicionar na edição de um *request* texto num dos campos disponíveis. Isto vai fazer com que este seja adicionado à tabela, apresentada na página de visualização de *time trackers*. Pelo contrário, um *time tracker* do tipo *contracts*, tem que ser feito no modal referido anteriormente, selecionando um contrato previamente criado, bem como o preenchimento de todos os campos obrigatórios. Após a criação, no modal, o *timetracker* é apresentado na tabela na página de visualização.

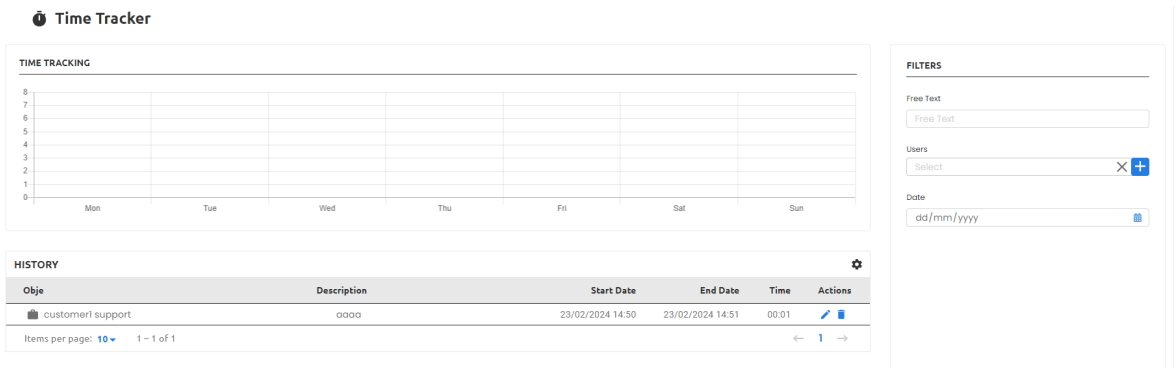

Figura 6.16: - Nova Página de Visualização de um "*Time Tracker*"

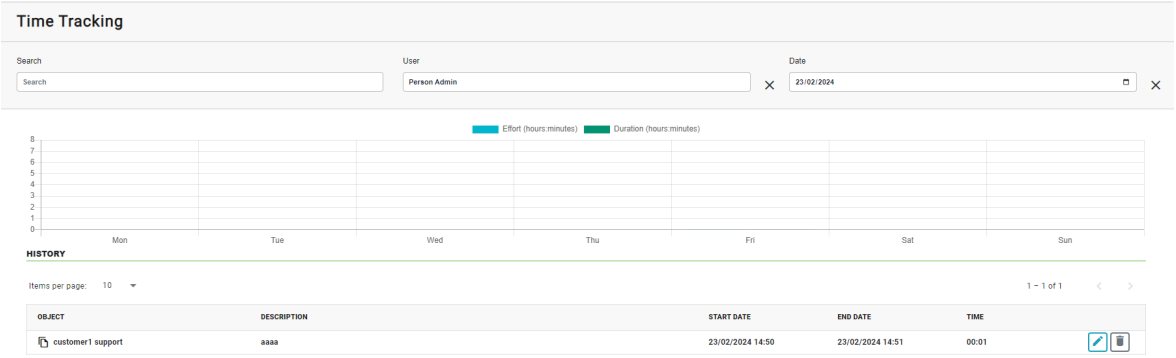

Figura 6.17: - Antiga Página de Visualização de um "*Time Tracker*"

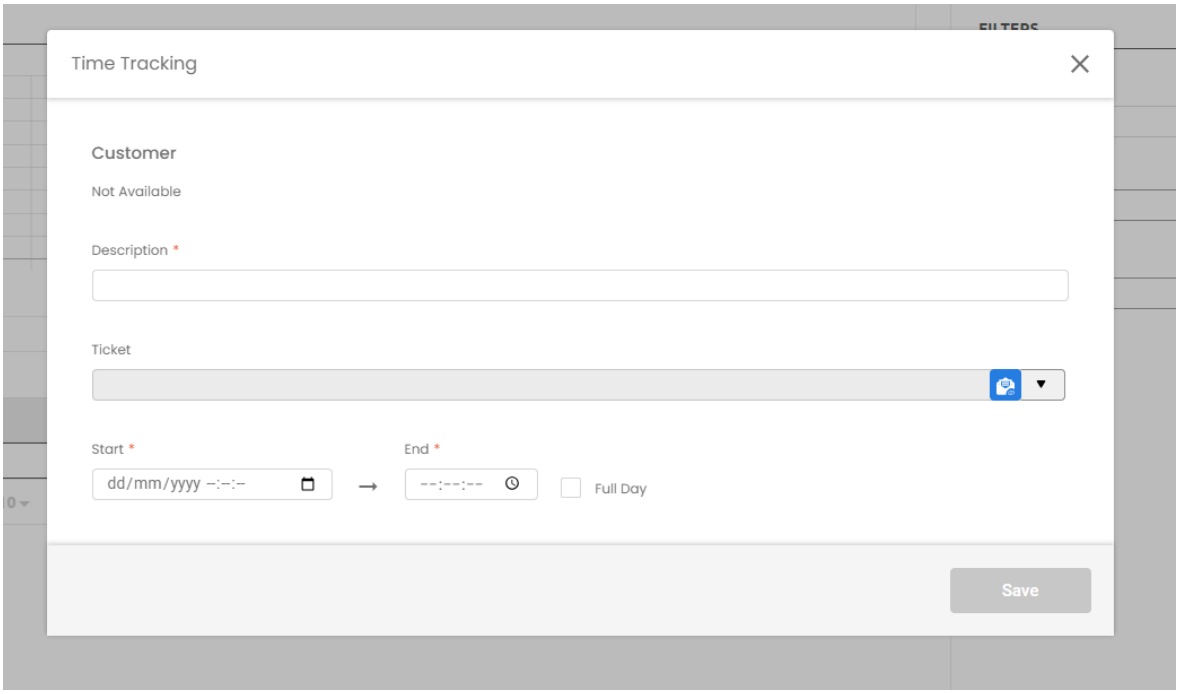

Figura 6.18: - Novo Modal de Criação de um "*Time Tracker*"

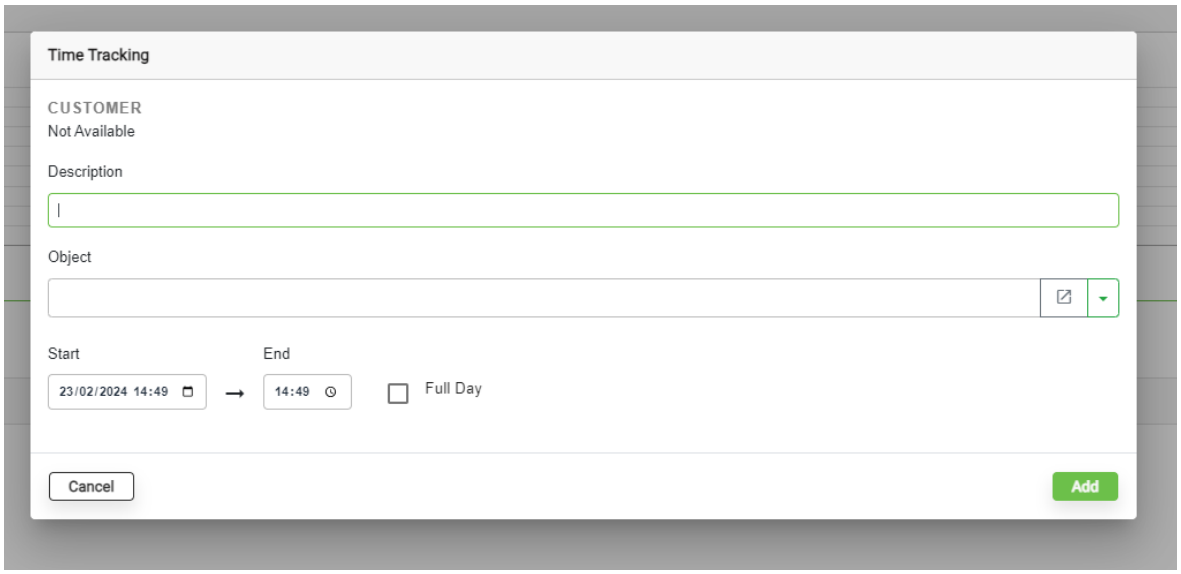

Figura 6.19: - Antigo Modal de Criação de um "*Time Tracker*"

6.2.5 Templates

A página dos *templates* foi a última realizada durante o estágio, fazendo este parte do modulo de *"Phishing"* e não de *"ITSM"*. Esta página é complexa, uma vez que é composta por uma página de visualização com opção de filtragem, meia página que abre quando se clica num dos objetos presentes na tabela e que apresenta diversas informações acerca desse objeto ao utilizador. Em adição, existem duas páginas de criação/edição de objetos diferentes mas que fazem ambos parte do objeto *templates*, *"Email/SMS"* e *"Landing Page"*.

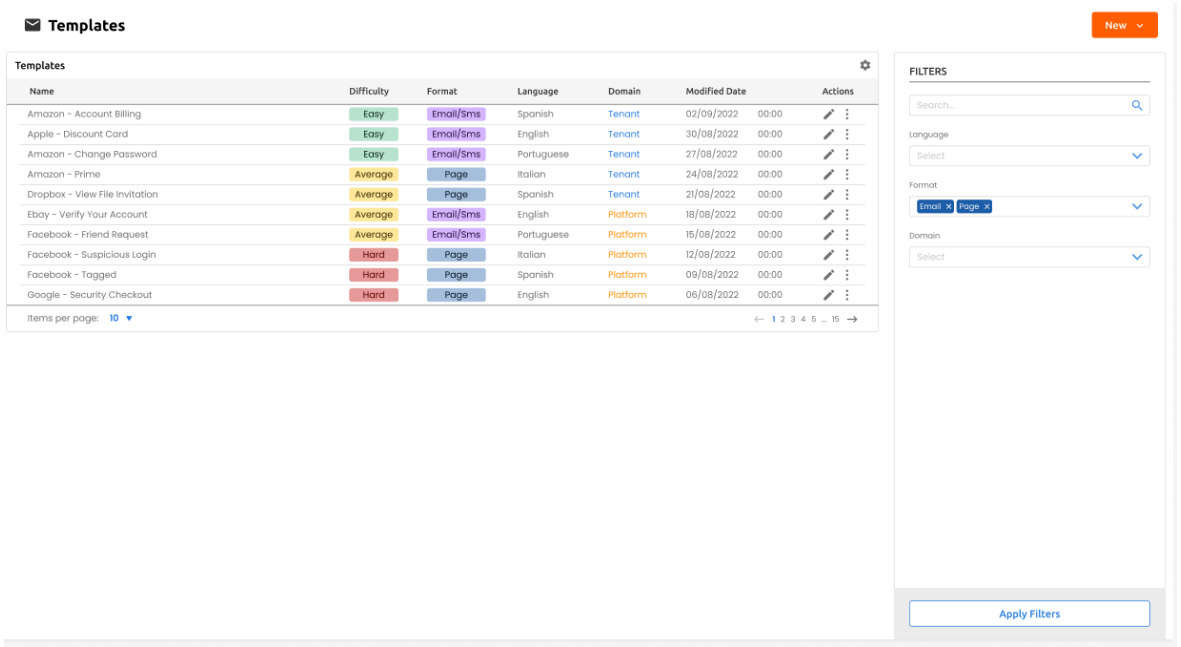

Figura 6.20: - Nova Página de Visualização de um "*Template*"

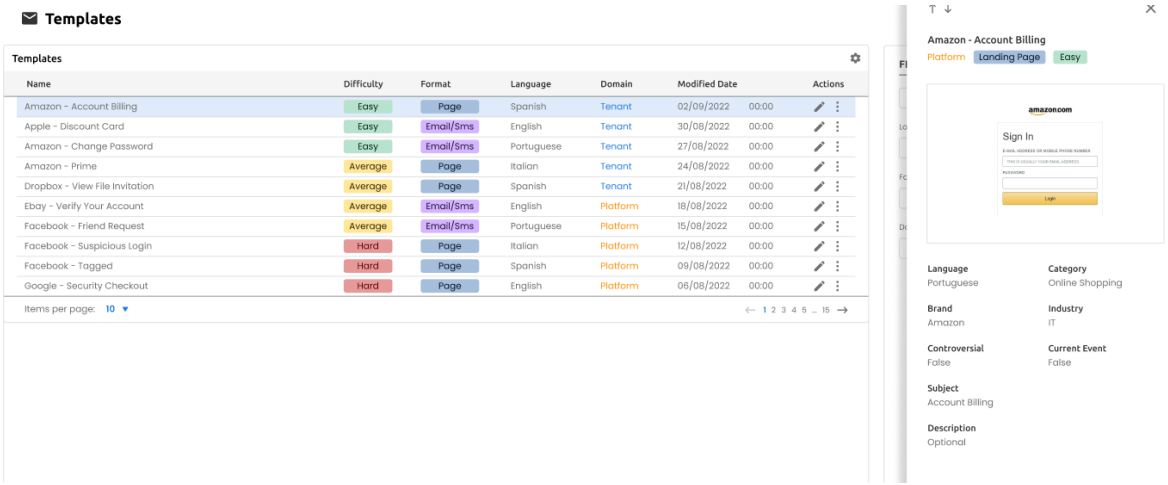

Figura 6.21: - Novo Painel Lateral de um "*Template*"

Desta forma, a primeira página de criação de um *template* do tipo *email/sms* é composta por cinco passos, sendo que o terceiro será apenas implementado no futuro. A primeira etapa contem diversas inserções de texto e selecionadores, dos quais apenas o nome, linguagem e assunto são obrigatórios de serem preenchidos. A segunda etapa corresponde ao conteúdo do *template*, sendo que uma ou ambas as *tabs*, email ou sms, têm de ser preenchidas. Em adição existe uma caixa de seleção que pode ser selecionada dependendo da vontade do utilizador em usufruir da sua aplicação. Na terceira etapa, é dado ao utilizador a opção de adicionar dois tipos de imagens para identificar mais facilmente o *template* criado. Por fim, na última etapa, o utilizador pode associar o *template* que está a criar a outro já criado previamente, desde que não seja da mesma língua. No caso de dois ou mais *templates* serem associados, é criado um *bundle* entre eles, que por regra não podem ter a mesma linguagem. Ou seja, se dentro de um *bundle* existir um *template* com uma certa linguagem, outro *template* com essa linguagem não pode ser associado a esse *bundle* já existente.

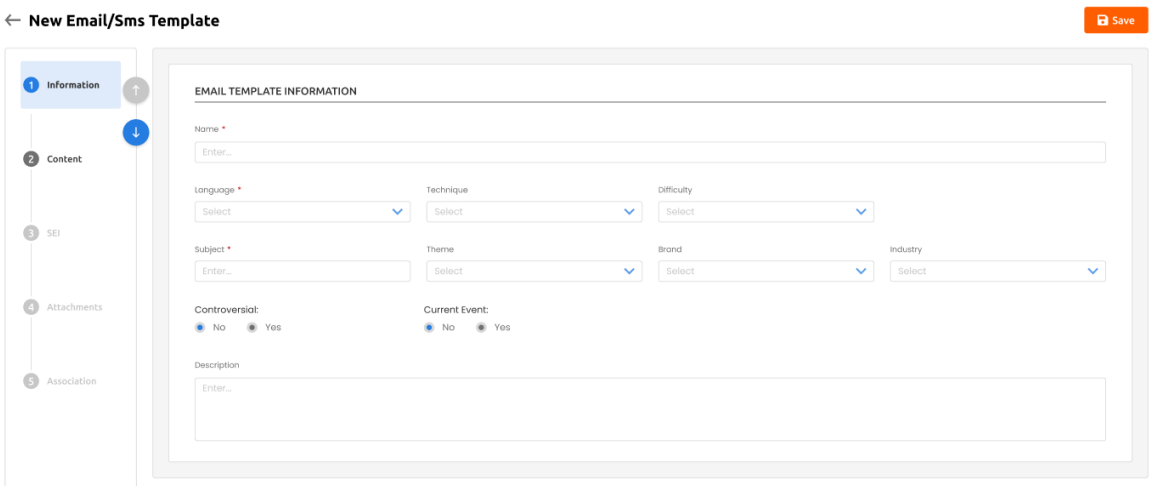

Figura 6.22: - Primeira etapa de criação/edição de um "*Email/SMS*"

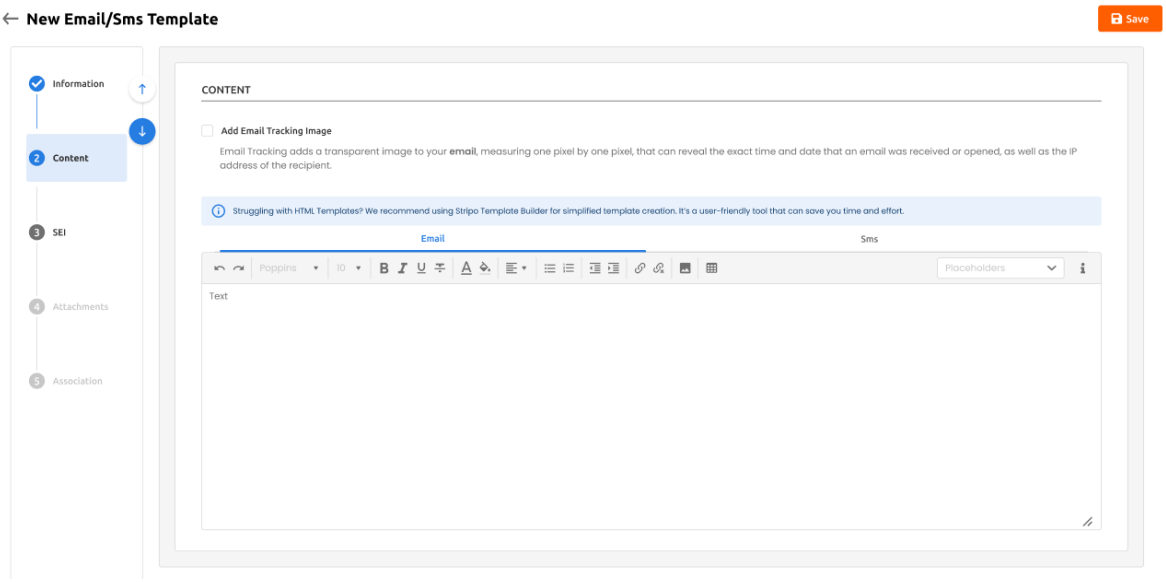

Figura 6.23: - Segunda etapa de criação/edição de um "*Email/SMS*"

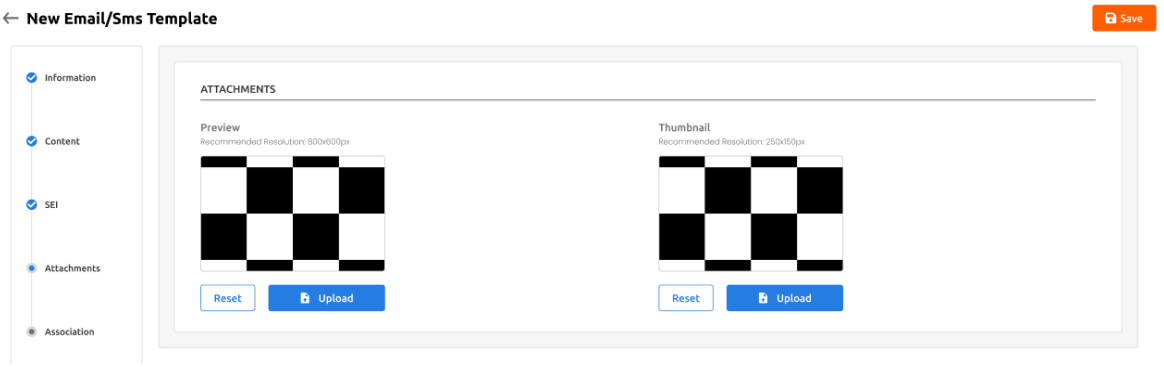

Figura 6.24: - Quarta etapa de criação/edição de um "*Email/SMS*"

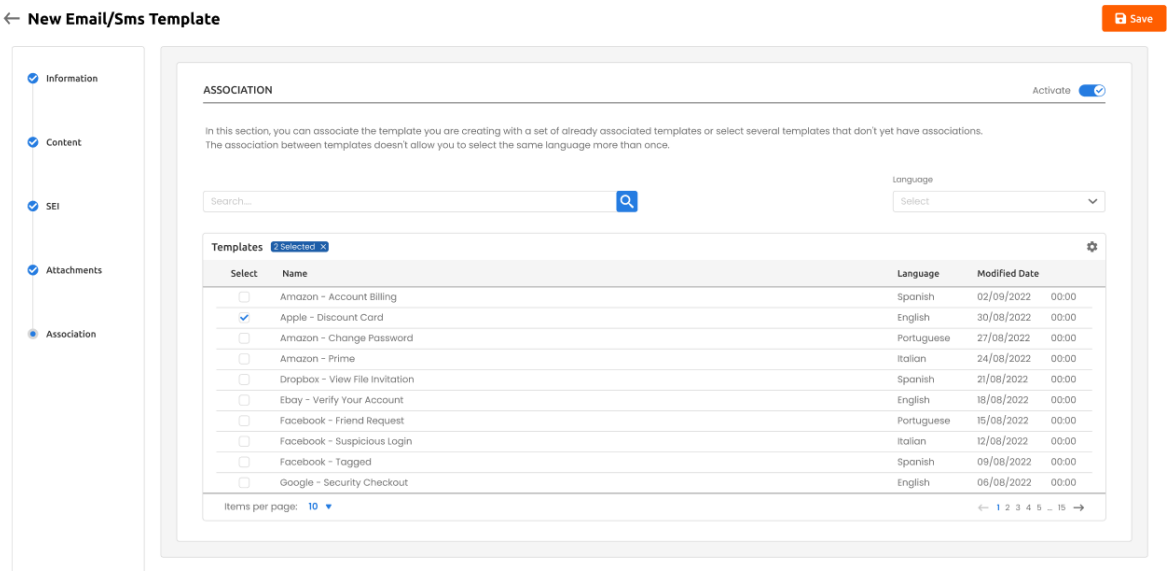

Figura 6.25: - Quinta etapa de criação/edição de um "*Email/SMS*"

Por outro lado, para criar um *template* do tipo *"landing page"*, temos apenas 4 etapas.

A primeira etapa é composta por uma inserção de texto obrigatória e dois selecionadores, um de linguagem e outro opcional que permite ao utilizador escolher um *template* do tipo *"email/sms"* para ficar associado à *"landing page"* que estamos a criar. Os restantes passos são iguais aos apresentados no tipo *"email/sms"*, à exceção do conteúdo, que neste caso apenas permite a introdução de um "*html*" que vai representar a página.

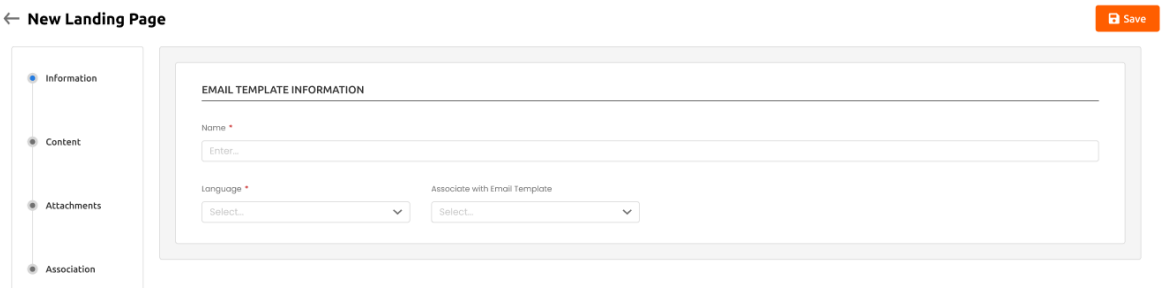

Figura 6.26: - Primeira etapa de criação/edição de uma "*Landing Page*"

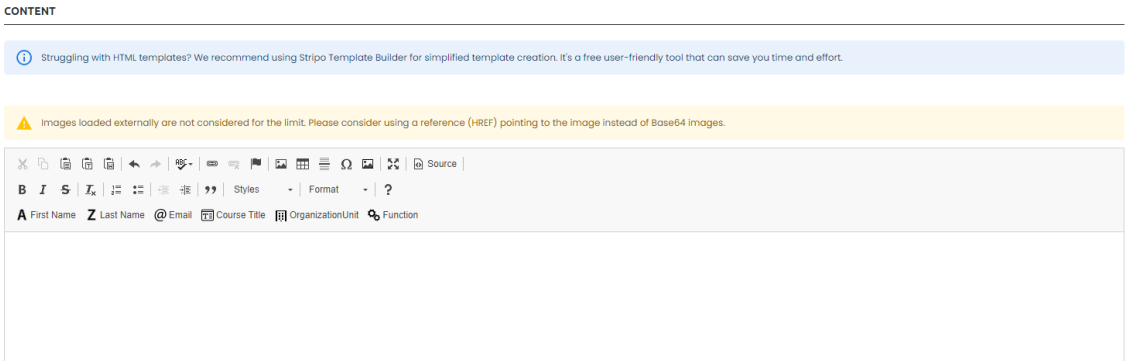

Figura 6.27: - Segunda etapa de criação/edição de um "*Landing Page*"

6.3 Conclusão

Neste capítulo foram apresentadas todas as página desenvolvidas durante a segunda parte deste estágio. No entanto, para além destas páginas houve também a correção de diversos *bugs* pela plataforma entre a mudança da criação de uma página para outra. Por fim, apenas acrescentar que o design da página *"Teams"* e *"Services"* foi criado por mim baseado no design anterior com os novos componentes. Em relação às restantes páginas, o seu design foi criado pela própria equipa de design da EMVENCI.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusões Principais

Com este relatório foi pretendido dar-se a conhecer a empresa e o trabalho realizando na mesma por parte do estagiário ao longo destes meses. A parte inicial do estágio permitiu a integração na empresa, domínio na utilização das diversas plataformas utilizadas na mesma e uma perceção sobre a metodologia do *Atomic Design*. Em adição, a segunda parte do estágio teve como objetivo a construção de diversas páginas web utilizando a metodologia do *Atomic Design*, aprimorando o meu conhecimento no tópico e as minhas capacidades como desenvolvedor. Em termo de conclusão, a realização deste estágio permitiu-me evoluir não só como um *front-end developer*, mas também como pessoa e conhecer diversas pessoas maravilhosas que trabalham na Emvenci.

7.2 Trabalho Futuro

O trabalho futuro vai incluir o término da construção das restantes páginas referentes ao modulo do *ITSM*, bem como a criação dos componentes atómicos necessários para o seu melhor desenvolvimento de acordo com as escolhas e criações da equipa de design. Para além disso, a melhoria das páginas já criadas é também um ponto que pode vir a acontecer de modo a homogeneizar com o resto da aplicação. Outra implementação para o futuro é a criação e utilização de testes *front-end*, algo que não foi feito durante o decorrer do estágio.

Webgrafia

[1] Brad Frost. (2023) Atomic design methodology. [Online]. Available: https://atomicdesign.bradfrost.com/chapter-2/ xiii, 4, 5, 6, 7

[2] Emvenci. [Online]. Available: https://www.emvenci.com/ 1

[3] Rebeka Costa. (2021) The guide to atomic design. [Online]. Available: https://www.justinmind.com/blog/atomic-design/ 3

[4] Brad Frost. Atomic design methodology. [Online]. Available: https://atomicdesign.bradfrost.com/chapter-2/ 4

[5] Matt Rae. (2020) Atomic design principles & methodology 101. [Online]. Available: https://xd.adobe.com/ideas/process/ui-design/atomic-design-principles-methodology-101/ 5, 6, 7

[6] Richard Bray. (2020) Atomic design: 10 reasons you should be using it. [Online]. Available: https://www.creativebloq.com/web-design/10-reasons-you-should-be-using-atomic-design-61620771 7

[7] Wade. (2020) The dangers of ng-deep bleeding. [Online]. Available: https://upmostly.com/angular/the-dangers-of-ng-deep-bleeding 8

[8] Praveen Dubey. (2018) What is progressive enhancement, and why it matters. [Online]. Available: https://www.freecodecamp.org/news/what-is-progressive-enhancement-and-why-it-matters-e80c7aaf834a/ 8

[9] Mantas Kunsmanas. (2020) Creating clean css with bem methodology. [Online]. Available: https://www.devbridge.com/articles/implementing-clean-css-bem-method/ 9

[10] Angular. [Online]. Available: https://angular.io/ 11

[11] Visual studio code. [Online]. Available: https://code.visualstudio.com/ 11

[12] Bitbucket. [Online]. Available: https://bitbucket.org/ 11

[13] Jira. [Online]. Available: https://www.atlassian.com/software/jira 12