Universidade de Aveiro
Escola Superior de Tecnologia e Gestão de Águeda

Gonçalo José Almeida Dias

Relatório de estágio

Glossário

R&D	Research and development
SMT	Surface-mount technology
UI	User interface
IT	Instituto de Telecomunicações
UA	Universidade de Aveiro
UM	Universidade do Minho
DOM	Document Object Model
GPS	Global Positioning System
JVM	Java Virtual Machine
API	Application Programming Interface
ORM	Object-Relational Mapping
SQL	Structured Query Language
JPA	Java Persistence API
NFC	Near Field Communication
UDP	User Datagram Protocol

CAPÍTULO 1

Introdução

O presente relatório pretende descrever as atividades desenvolvidas durante o Estágio inserido no 3º ano da Licenciatura em Tecnologias da Informação, lecionado na Escola Superior de Tecnologia e Gestão de Águeda e coordenado pelo Prof. Doutor Ivan Pires.

A realização do estágio tem com objetivo colocar em prática todos os conhecimentos, teóricos e práticos adquiridos e desenvolvidos nas unidades curriculares. O estágio decorreu na Wiseware Solutions, Gafanha da Encarnação, entre 11 de Fevereiro e 23 de Maio de 2025 sobre a orientação de Gustavo Corrente.

Este estágio teve como principais objetivos a obtenção de experiência em contexto de trabalho, adquirir novos conhecimentos e desenvolver competências práticas e teóricas a nível profissional.

O presente relatório encontra-se dividido em quatro partes. Na primeira parte, é apresentada uma breve descrição da entidade de acolhimento. No segundo capítulo, é feito um comentário à cerca do plano de trabalho. A terceira parte contempla uma descrição detalhada de todo o trabalho realizado ao longo do estágio, no âmbito do projeto *aniposture*, nomeadamente ao nível da interface gráfica e do *backend*. Por fim é realizada uma analise com base nos conhecimentos adquiridos durante as 466H de estágio.

Para concluir o relatório são apresentadas considerações finais, uma reflexão sobre as competências adquiridas e um balanço geral do estágio.

CAPÍTULO 2

Entidade de acolhimento

A Wiseware Solutions é uma empresa de R&D localizada na Zona Industrial da Mota, Gafanha da Encarnação, com a missão de desenvolver soluções inovadores e de alta qualidade para empresas.

A Wiseware é responsável por produzir soluções em áreas como robotica, micro eletronica, inteligencia artificial, comunicações sem fio e muito mais. Aplicaram o seus conhecimentos em projetos de saúde e bem-estar, telemetria, controlo de qualidade, inspeção e automação de sistemas.

Figura 2.1: Wiseware Solutions

A Wiseware realiza vários serviços como a criação de PCB, montagem de SMT, inspeção de SMT, desenho tecnico, cortes 2D/3D a lazer, fresagem, etc.

Apresentam um conjunto de parceiros como a *Universidade de Aveiro (UA)*, o *Instituto de Telecomunicações (IT)*, a escolha de engenharia da *Universidade do Minho (UM)*, a *inovaria*, entre outros. Como seus clientes estão nomes como a *Adidas*, *Digital Logic*, *Efcom*, grupo *iPesa*, entre outros.

CAPÍTULO 3

Plano de trabalho

Antes do estágio começar, foi definido um plano de trabalho que estabelecia os principais objetivos a atingir no decorrer do mesmo, Figura A.1. Embora este plano represente a base para a realização do estágio, é expectável que, no final, possam existir variações entre o trabalho inicialmente planeado e o efetivamente realizado.

O documento apresenta os objetivos propostos, as atividades a desenvolver para a sua concretização e as horas prevista por atividade. Estes objetivos encontram-se organizados por áreas de trabalho distintas, conforme ilustrado na referida Figura A.1.

A fase inicial de acolhimento e introdução ao desenvolvimento, com uma previsão de 40 horas, visou a introdução com o ambiente de trabalho e, a aquisição de conhecimentos sobre as tecnologias e ferramentas essenciais usadas pela empresa no processo de desenvolvimento, como *Git*, *Gitlab*, *Jenkins*, o ambiente *Java* e *Svelte*.

Seguiu-se o levantamento de requisitos, também estimado em 40 horas. Este período dedicou-se à aquisição de uma compreensão aprofundada do funcionamento do projeto *aniposture*, nomeadamente dos seus objetivos, estrutura e tecnologias envolvidas.

O estudo e seleção da *API* de Mapas constituiu um objetivo crucial, com 120 horas alocadas. Esta tarefa implicou uma análise comparativa das funcionalidades, custos e relação custo-benefício de diversas APIs de georreferenciação, como *Google Maps* e *Mapbox*, para suportar a tomada de decisão.

Com 120 horas previstas, o desenvolvimento da interface de utilizador, UI, centrou-se na implementação gráfica e na integração da API de georreferenciação previamente selecionada.

Paralelamente, o desenvolvimento de serviços, com igual carga horária de 120 horas, focou-se na criação de serviços *RESTful* e na implementação de *endpoints* essenciais para o funcionamento do *backend* da aplicação.

A realização de testes funcionais, com uma duração estimada de 24 horas, visou validar o correto funcionamento da aplicação e a sua conformidade com os requisitos definidos.

Por fim, dedicou-se tempo à finalização da escrita do relatório, com uma previsão de 2 horas, ainda no decorrer do período de estágio.

Este plano de trabalho serviu, portanto, como um guia para a realização do estágio, ao definir as principais tarefas e as suas respetivas metas. A sua existencia foi fundamental para direcionar os esforços, servir como base de comparação e permitir medir o progresso do estágio.

CAPÍTULO 4

Trabalho realizado

Neste capítulo irei abordar todo o trabalho desenvolvido durante o período de estágio, organizado em quatro grandes secções. Na primeira farei uma pequena descrição do projeto aniposture. Na segunda abordarei o trabalho realizado durante a analise de APIs e bibliotecas, incluindo o porque da escolha final de cada uma. A terceira secção visa comentar o trabalho realizado na interface gráfica do projeto aniposture. E por fim comentarei o trabalho desenvolvido no respetivo backend.

4.1 Descrição do projeto

O projeto *Aniposture* apresenta uma solução inovadora que integra dispositivos inteligentes e tecnologias de análise de dados para monitorar o bem-estar animal e melhorar os processos agropecuários.

O objetivo consiste em proporcionar uma gestão eficiente e precisa do comportamento dos animais, através de coleiras equipadas com sensores que registam o movimento, medem a temperatura, avaliam a postura e determinem a localização via GPS. Os módulos de comunicação das coleiras enviam os dados para uma infraestrutura *cloud*, que centraliza e processa a informação.

Cada coleira incorpora uma *IMU*, Unidade de Medição Inercial, com uma frequência de amostragem ajustável, o que permite a recolha de dados sobre a aceleração e a orientação.Desta forma, é possível identificar padrões de movimento e posturas, como estar deitado, em pé ou em movimento. A coleira integra sensores de temperatura que permitem monitorar tanto a temperatura atmosférica como a do temperatura animal. O dispositivo regista igualmente o estado da bateria e emite alertas quando o nível encontra-se baixo. Inclui ainda uma interface NFC que possibilita a configuração local dos dispositivos.

A plataforma na *cloud* disponibiliza a informação por meio de uma interface *web*. Entre as funcionalidades incluem-se a visualização do estado das coleiras e o acompanhamento em tempo real das localizações GPS. A interface apresenta ainda gráficos e filtros por coleira e por período temporal, o que permite consultar a informação de forma rápida e eficaz.

A combinação da tecnologia integrada nas coleiras com a infraestrutura *cloud* coloca o projeto *aniposture* na vanguarda da inovação no sector agropecuário, o que favorece uma gestão mais eficiente e sustentável dos rebanhos.

4.2 Análise de bibliotecas

Durante este estágio, foram-me atribuídas tarefas de análise e pesquisa, com o objetivo de identificar e selecionar as ferramentas mais adequadas que cumprissem os requisitos necessários para o desenvolvimento da aplicação. Foram, assim, realizadas duas grandes análises: a primeira sobre as *APIs* de georreferenciação e a segunda nas *APIs* de visualização gráfica.

4.2.1 APIs georreferenciação

Esta foi uma das primeiras tarefas que foram-me atribuídas durante o estágio. Antes de começar as comparações, foram definidos alguns requisitos fundamentais, nomeadamente a necessidade de *tiles* em satelite, marcadores, bom desempenho com uma elevada quantidade de pontos e desenho de polígonos. Depois de levantados os requisitos, iniciou-se a análise comparativa entre três *APIs* de georreferenciação o *mapbox*, *openlayers* e o *google maps*.

Apesar de partilharem o mesmo objetivo, criação de mapas interativos, as três bibliotecas são consideravelmente diferentes nas suas implementações.

OpenLayers

Openlayers é o biblioteca *free e Open Source* que permite a criação de mapas interativos. Disponibiliza recursos como *raster tiles*, camadas vetoriais e marcadores. Por ser *Open Source*, contem vários *addons* criados pela comunidade que permite alargar as suas funcionalidades. Contudo, apresenta uma linha de aprendizado um pouco mais complexa do que outras alternativas e, por padrão, não contêm nenhum *tile* em satelite.

Foram ainda realizados alguns testes com opções de desenho vetorial. No entanto, o que levou o *OpenLayers* não ser o escolhido foi a sua, ainda, baixa integração com *WebGL*, a documentação menos completa em comparação com outras alternativas e a ausência de suporte nativo para mapa com satélite.

Embora as capacidades de desenho vetorial fossem positivas, apresentava, na altura da análise, algumas limitações para o desempenho exigido em cenários com um elevado

volume de dados, como os rastros de localização dos animais. Esta limitação, em conjunto com a complexidade na configuração de funcionalidades essenciais, como as camadas de satélite, e da necessidade de realizar otimizações de forma manual, resultou numa curva de aprendizado mais acentuada e num tempo de desenvolvimento maior. Adicionalmente, a documentação, comparada com outras alternativas, mostrou-se menos completa, o que prolongou a resolução de problemas e a implementação de características específicas.

Perante este conjunto de constrangimentos, e ao considerar os requisitos de georreferenciação do projeto, optou-se por bibliotecas que demonstraram maior sinergia em termos de performance, facilidade de uso e suporte nativo às funcionalidades pretendidas.

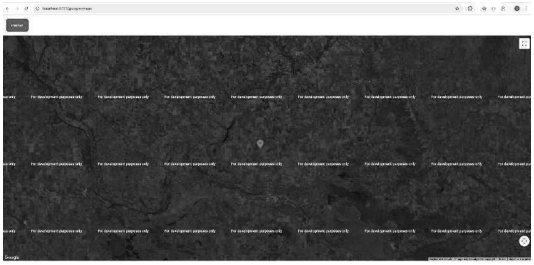
Google Maps

O *google maps* é, muito provavelmente, o sistema de mapas mais utilizado e atualizado a nível mundial. Por essa razão, foi considerada a sua utilização, tendo sido analisada a forma como é implementado e as suas funcionalidades. Inicialmente, esta solução revelou-se a mais promissora, uma vez que permite adicionar camadas vetoriais, marcadores, vista em satélite e possui um bom desempenho.

Ao contrário do *Openlayers*, 4.2.1, esta é uma solução paga, que apresenta, algumas vantagens como mapas satélite mais atualizados e integração com os serviços da *Google*. Contudo, o maior problema é o seu preço, consideravelmente elevado se formos comprar com outras alternativas. A *Google* permite a utilização da *Maps JavaScript API*[1] de forma gratuita até 10.000 pedidos mensais. A partir dessa quantidade começam a ser aplicadas cobranças.

Embora as suas capacidades técnicas fossem adequadas, a sua integração com outros serviços da *Google* e a qualidade das suas vistas de satélite fossem vantajosas, a estrutura de custos e a incerteza associada ao escalonamento da utilização impediram a sua seleção final. A prioridade recaiu sobre bibliotecas que oferecessem um modelo de licenciamento mais previsível e economicamente sustentável.

(a) OpenLayers



(b) Google Maps

Figura 4.1: *OpenLayers & Google Maps*

Mapbox

Chegamos, por fim, à opção escolhida: o *Mapbox*. Esta biblioteca revelou-se a melhor entre os dois mundos, implementação inicial simples, utiliza *WebGL* por padrão e disponibiliza vários tipos de *tiles*, incluindo satélite.

Embora seja uma solução paga, apresenta modelos mais em conta do que a opção da *google*, a versão gratuita, contem um total de 50.000 pedidos por mês. Esta mesma quantidade de pedidos no *google maps* teria um custo de cerca de 280€ mensais. Para atingir um custo parecido no *mapbox*, seria necessário utilizar pelo menos 100.000 pedidos mensais. Para além do fator preço/economico, esta solução, compre todos os requisitos definidos para a nossa aplicação, juntamente de uma documentação bem estruturada.

Tendo em conta todos os requisitos levantados, o *mapbox* cumpre-os de forma exímia. Graças á sua grande comunidade, o *mapbox*, dispõe vários plugins que permitem expandir significativamente as suas funcionalidades, o que acrescenta ainda mais valor à sua escolha como opção para o projeto.

Figura 4.2: Mapbox

A implementação mais simples com o uso nativo de *WebGL* no *Mapbox* garante não só um desenvolvimento mais rápido, mas também a capacidade de processar e renderizar grandes volumes de dados, tais como os trajetos de múltiplos animais ou o desenho de polígonos para delimitar áreas. Esta performance é crucial para oferecer aos utilizadores uma melhor experiência, mais fluida e em tempo real, permitindo-lhes monitorar os animais de forma detalhada e interativa.

Do ponto de vista económico, a estrutura de preços do *Mapbox* oferece uma vantagem estratégica significativa, o que proporciona uma maior previsibilidade e controlo de custos. Ao juntar a isto, a qualidade da documentação técnica do *Mapbox* facilita a integração, a resolução de problemas e a manutenção do código.

Universidade de Aveiro
Escola Superior de Tecnologia e Gestão de Águeda

Gonçalo José Almeida Dias

Relatório de estágio

Relatório de Estágio apresentado à Universidade de Aveiro para obtenção do da Licenciatura em Tecnologia de Informação, realizado sob a orientação do Prof. Doutor Ivan Miguel Serrano Pires, da Escola Superior de Tecnologia e Gestão de Águeda, Universidade de Aveiro.

4.2.2 APIs de visualização gráfica

Após analise e avaliação das bibliotecas de georreferenciação, realizei uma pequena pesquisa e comparação na área dos gráficos.

D3js

Esta foi a biblioteca identificada pelo orientador, como uma possível opção a implementar. O *d3js* não é uma biblioteca de gráficos tradicional, uma vez que não possui o conceito de "gráficos". Quando visualizamos os dados com o *d3js*, estamos a compor uma variedade de objetos primitivos, como linhas, círculos, retângulos, entre outros.

Apesar de apresentar um desempenho elevado e permitir um elevado nível de interatividade, a sua utilização é consideravelmente mais complexa que outras alternativas.

Figura 4.3: Exemplo de gráfico criado com o d3js

ChartJs

O *chartJs* é uma das bibliotecas mais conhecidas e amplamente utilizadas no mundo *JavaScript* para representação de dados.À data atual, conta com mais de **66 mil** estrelas no *github*[2] e é regularmente atualizada pela comunidade e pelos seus contribuidores.

Ao contrário do *d3js*, 4.2.2, o *chartJs* é consideravelmente mais simples de utilizar. Trata-se de uma biblioteca que incorpora o conceito de "gráfico", permite a visualização da informações em 8 estilos diferentes, sendo ainda responsivo.

No entanto, essa mesma simplicidade pode ser vista como uma limitação: os gráficos, por padrão, oferecem poucos funcionalidades além da visualização básica dos informação.

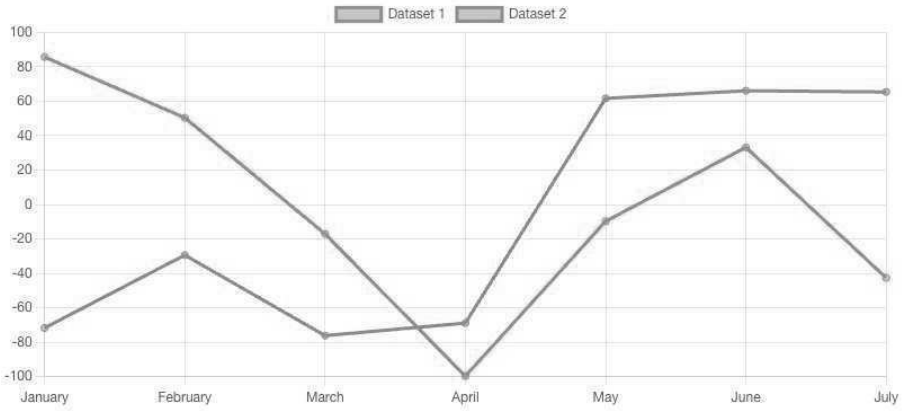


Figura 4.4: Exemplo de gráfico criado com o chartJs

Echarts

Apache Echarts é uma biblioteca gratuita, desenvolvida no âmbito da fundação *Apache*, que apresenta caracteristicas similares ao chartJs. Suporta mais de 20 tipos diferentes de gráficos, é responsivo e permite a renderização de até 10 milhões de dados em tempo real.

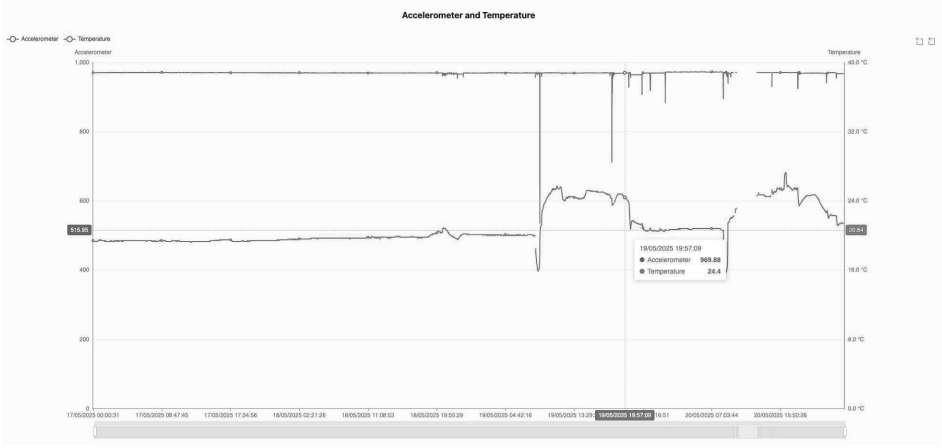


Figura 4.5: Exemplo de gráfico criado com Echarts

O *Apache Echarts* foi a solução escolhida para integrar no projeto, não apenas por apresentar um maior número de opções de visualização de dados, mas também pelas suas pelas suas funcionalidades nativas e pela elevada performance a carregar uma grandes volumes de dados.

4.3 Interface gráfica

Nesta fase, será abordado com detalhes todo o trabalho realizado no *frontend*/interface gráfica da aplicação, incluindo as dificuldades encontradas, conhecimentos adquiridos e as tarefas realizadas durante o desenvolvimento.

4.3.1 Svelte 5

Durante a fase de desenvolvimento, esta foi das primeiras tarefas que foram-me atribuídas. No último trimestre de 2024, a equipa responsável pelo *svelte* lançou uma nova versão, que alterou conceitos fundamentais da *framework*. Fui, então, encaminhado a aprender esta nova versão, testar as novas funcionalidades, compreender as principais alterações e analisar o processo de atualização de um projeto desenvolvido na versão 4 para a versão 5 do *svelte*.

O *svelte* 5 introduziu um novo conceito denominado *runes* (runas), que consistem num mecanismo para declarar estado de reatividade. As runas são símbolos que influenciam o compilador do *svelte*, permitem ao programador definir, de forma explícita, quais objetos são reativos e quais não são, Figura 4.6b.

Nas versões anteriores, o *svelte* tornava todos os objetos reativos por padrão, isto dificultava a perceção de quais objetos podiam, efetivamente, provocar alterações na interface gráfica.

Estas mudanças representam uma alteração significativa na forma como criavamos código em *svelte*. A nova abordagem, removeu a sintaxe do *svelte* 4, Figura 4.6a, que, muitas das vezes, era confusa, já que a mesma expressão podia realizar ações diferentes. Com a introdução da nova sintaxe, cada uma das ações realizas é bem definida, o que torna o código mais previsível e fácil de manter.

```
<script>
  let count = 0;

  $: doubled = count * 2
  $: document.title = `count is ${count}`
</script>
```

(a) Exemplo *Svelte* 4

```
<script>
  let count = $state(0)

  let doubled = $derived(count * 2)

  $effect(() => console.log(`count is ${count}`))
</script>
```

(b) Exemplo *Svelte* 5

Figura 4.6: Diferenças *Svelte* 4 e 5

Uma das alterações introduzidas nesta nova versão do *svelte* foi a forma como passamos elementos para dentro dos componentes.

Na versão anterior, utilizava-se a palavra-chave *export*, o que permitia declarar propriedades em qualquer parte do componente. Essa flexibilidade, embora útil, podia tornar o código menos organizado e mais difícil de compreender. Na nova versão, todos

os elementos recebidos pelo componente são definidos num único local, o que promove uma estrutura mais clara e facilita a leitura e manutenção do código. Ver Figura 4.7.

```
// Svelte 4
export let prop1;
export let prop2;

// Svelte 5
let { prop1, prop2 = $bindable() } = $props();
```

Figura 4.7: Exemplo *props svelte*

Depois de familiarizar-me com os novos conceitos introduzidos no *svelte* 5, iniciei a modificação e criação novos componentes nesta versão. Os dois conceitos mais importes, e os que exigiram mais tempo a compreender foram o *$derived* e o *$effect*.

O ***$derived*** é uma runa utilizada para definir uma variavel derivada. Ou seja, cria-se uma nova variavel cujo o valor é automaticamente atualizado com base numa expressão, sempre que ocorre alguma alteração noutra variavel, Figura 4.6b.

Já o ***$effect*** é uma função que é executada sempre que um determinado estado é atualizado. Esta runa pode ser utilizada, por exemplo, para chamar biblioteca externas, desenhar elementos num canvas ou realizar pedidos à rede, Figura 4.6b.

```
let hasStats = $derived.by(() => {
    const rs = sampleList.filter((sample) => sample.details?.stats);

    if (rs.length > 0) return true;
    return false;
});
```

(a) Exemplo *$derived*

```
$effect(() => {
    if (derivedPoints && derivedPoints.length > 0 && mapLoaded) {
        resetPointsLayers();
        updateMapLayers();
        boundingZoom();
    }
});
```

(b) Exemplo *$effect*

Figura 4.8: *$derived & $effect*

Na Figura 4.8, podemos ver um exemplo um pouco mais complexo da utilização das runas *$derived e $effect*.

No caso do *$derived*, a variavel *hasStats* é atualizada sempre que ocorrem alterações no *array sampleList*. Já o *$effect* executa todas as funções chamadas no seu interior sempre que o mapa está carregado e a variavel *derivedPoints* é alterada.

4.3.2 Tecnologias fundamentais

Nesta secção irei comentar um pouco sobre as outras tecnologias que existem no projeto *aniposture*.

Tailwind

O *tailwind* é uma *framework* css que permite construir interfaces diretamente no *HTML*, através da utilização de classes pré definidas. Esta abordagem promove rapidez e consistência no design, o que reduz necessidade de escrever *CSS* personalizado.

Esta foi a minha primeira experiencia com tailwind. Encontrei algumas dificuldades nas primeiras semanas, sobretudo por ainda não compreender totalmente o seu conceito nem as suas classes pré definidas. No entanto, após ultrapassar esta fase de adaptação, consegui desenvolver interfaces de forma mais rápida e prática.

```
<div class="flex flex-col items-center justify-center gap-2 m-4 mb-2">
    <div class="text-sm ☐text-gray-700 ■dark:text-gray-400">
        Showing
        <span class="font-semibold ☐text-gray-900 ■dark:text-white">{filteredItems.length}</span>
        of
        <span class="font-semibold ☐text-gray-900 ■dark:text-white">{devicesLists.size}</span>
        Devices. Updated @
        <span class="font-semibold ☐text-gray-900 ■dark:text-white">
            {UTILS.timestampToDateTimeStr(data.requestDate)}
        </span>
    </div>
</div>
```

Figura 4.9: Exemplo de interface criada com *tailwind*

Flowbite svelte

O *flowbite* é uma biblioteca de componentes baseada em *tailwind*, que tem como objetivo acelerar e simplificar a criação de interfaces. Esta fornece um vasto conjunto de componentes prontos a usar, como tabelas, botões, modais, entre outros. A utilização destes componentes contribui para uma interface mais uniforme e coerente.

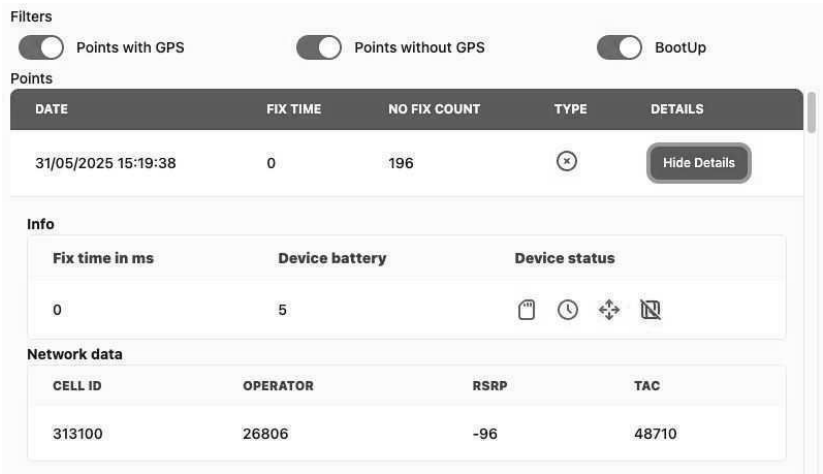


Figura 4.10: Interface com componentes *flowbite*

4.3.3 Alterações no *design system*

Uma das tarefas que executei durante o desenvolvimento deste projeto foi a reformulação da interface, alteração que abrangeu todas as páginas da aplicação.

Para tornar essa reformulação possível, alterei o tema principal. O tema anterior, gerado automaticamente pelo *tailwind* a partir de uma única cor, originou uma paleta cromática que se afastava da identidade visual pretendida para a aplicação.

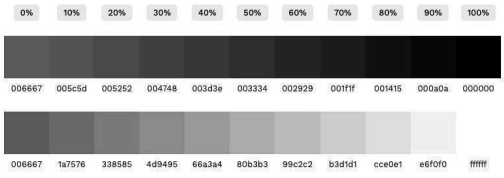
A Figura 4.11b mostra que as tonalidades mais claras da aplicação apresentavam um azul claro que não combinavam com as restantes cores. Para corrigir essa incoerência, alterei a paleta cromática para um sistema de sombras, mantendo a mesma cor base em variações mais claras e mais escuras. O novo esquema confere um aspeto mais agradável à seleção de botões, às colunas das tabelas, aos realces e aos demais elementos.

Esta mudança de paleta assegura que a interface respeite a identidade visual definida. A cor base mantém-se, mas apresenta-se em sombras progressivamente mais claras e mais escuras, o que gera uma hierarquia visual consistente. Os componentes que antes exibiam um azul claro passam a integrar-se no conjunto cromático, o que melhora a leitura e a navegação. O resultado oferece uma experiência mais coesa e agradável para o utilizador.

Foram também efetuadas alterações às bibliotecas de ícones. Anteriormente, a aplicação dispunha de uma biblioteca com um número reduzido de ícones, fator que limitava as possibilidades de desenvolvimento. Instalei uma nova biblioteca, *Tabler*[1], que apresenta uma vasta gama de opções.

Com o objetivo de tornar a interface mais harmoniosa, introduziram-se alterações que permitem a várias páginas reutilizar componentes, quer do *flowbite*, quer desenvolvidos internamente.

Esta abordagem aumenta a consistência visual e reduz o esforço necessário de desenvolvimento, ao evitar duplicação de código e simplificar a manutenção. Como resultado, a aplicação apresenta um aspeto mais uniforme e permite atualizações mais rápidas no futuro.



(a) Cores geradas por sombra

(b) Cores geradas pela tailwind

Figura 4.11: Sistemas de cores

[1]Tabler - https://tabler.io/icons

4.3.4 Componente Mapa

Um dos trabalhos mais complexos nesta fase da interface foi a implementação do mapa. Trata-se de um elemento com caracteristicas especiais, que exige funcionalidades como o carregamento de pontos, executar ações ao detetar cliques, carregar *layouts*, atualização automatica de novas informações, desenho de elementos, entre outros.

Um dos pontos mais importantes durante o desenvolvimento deste componente foi garantir que os dados fossem carregados e atualizados automaticamente. Este aspeto revelou-se particularmente importante devido à arquitetura *multi tenant* do projeto. Ao escolher um novo *tenant*, todas as informações tinham de ser limpas e substituídas por novas. Para atingir este objetivo, foram utilizadas várias ferramentas do *svelte 5*, Subseção 4.3.1.

Layers (Camadas)

As *Layers*, ou camadas, são componentes principais em qualquer sistema de mapas. O *mapbox* permite a adição de diversas *camadas* ao mapa, como elementos vetoriais, simbolos, imagens *raster*, modelos 3D, entre outros. Estas camadas permitem representar diferentes tipos de dados geográficos e visuais, sendo fundamentais para a construção de mapas interativos.

Na criação deste componente, foram adicionadas duas camadas para cada tipo de animal existente. Uma das camadas é responsável por mostrar todos os elementos, enquanto a outra exibe apenas os elementos selecionados.

```
helper.addLayer({
    id: 'sheep-layer',
    type: 'symbol',
    source: 'sheep-source',
    layout: {
        'icon-image': 'sheep',
        'icon-ignore-placement': true
    }
});
```

(a) Camada de elementos

```
helper.addLayer({
    id: 'sheep-selected',
    type: 'symbol',
    source: 'sheep-source',
    layout: {
        'icon-image': 'sheep-selected',
        'icon-ignore-placement': true
    },
    filter: ['in', 'id', '']
});
```

(b) Camadas de elementos selecionados

Figura 4.12: Camadas

Como ilustrado na Figura 4.12a, na camada correspondente a todos os elementos é carregada a informação através da fonte *sheep-source*, sendo os elementos posteriormente carregados no mapa. Já a camada *sheep-selected* apenas apresenta os elementos quando o filtro correspondente está ativado. Essa seleção ocorre através do evento *on click*, que identifica o ponto naquela coordenada. Após seleção, o *svelte* atualiza automaticamente os pontos no mapa. Figura 4.13.

```
$effect(() => {
    if (selectFeature && mapLoaded) {
        if (selectFeature.properties && selectFeature.geometry) {
            map.setFilter('points-selected', ['in', 'id', selectFeature.properties.id]);
            map.setFilter('sheep-selected', ['in', 'id', selectFeature.properties.id]);
            map.setFilter('goat-selected', ['in', 'id', selectFeature.properties.id]);
            map.setFilter('cow-selected', ['in', 'id', selectFeature.properties.id]);
        }
    }
});
```

(a) Selecionar filtros

```
map.on('click', (e) => {
    if (!disableClicking) {
        const selectedFeatures = map.queryRenderedFeatures([e.point.x, e.point.y], {
            layers: ['no-type-layer', 'goat-layer', 'cow-layer', 'sheep-layer']
        });

        if (selectedFeatures.length > 1 || selectedFeatures.length === 0) return;

        selectFeature = selectedFeatures[0];
```

(b) *Map onClick*

Figura 4.13: Atualizar pontos selecionados no mapa

Uma das funcionalidades que tive de implementar, de forma a facilitar a gestão dos polígonos e dos pontos no mapa, foi a tradução de todos os pontos das camadas para *geoJson*[3]. Esta conversão, de pontos para *geoJson*, formata toda a informação num padrão reconhecido por múltiplos sistemas. Desta forma, é possível editar e exibir a informação, independentemente da fonte e do formato de origem dos dados.

```
export function generateFeature(point: GpsSample): Feature | null {
    if (point.longitude && point.latitude) {
        return {
            type: 'Feature',
            properties: {
                id: point.id,
                animal_type: point.animal_type,
                device_status: point.device_status
            },
            geometry: {
                coordinates: [point.longitude, point.latitude],
                type: 'Point'
            }
        };
    }
    return null;
}
```

Figura 4.14: Função *generateFeature*

No nosso caso, extraímos a latitude e longitude armazenadas nos pontos e convertemos-las numa lista de pontos *geoJson*. Uma das vantagens do *geoJson* é a possibilidade de associar propriedades a cada ponto, que podem ser acedidas posteriormente, ao clicar num ponto. Após gerar cada uma das *features* a partir dos pontos e inseri-las nas camadas corretas, Figura 4.15b, procedemos à atualização de todas as *sources* do mapa de forma a carregar os novos dados, Figura 4.15a.

```
// Função do mapa para atualizar todas as sources.
function updateSources() {
    if (!vector_points) return;
    helper.updateSources('points-source', vector_points);
    helper.updateSources('sheep-source', sheep_points);
    helper.updateSources('goat-source', goat_points);
    helper.updateSources('cow-source', cow_points);
    helper.updateSources('no-type-source', no_type_points);
}


// Função do mapUtils responsável por atualizar as minhas sources.
function updateSources(layerId: string, featureCollection: FeatureCollection) {
    const source = this.map.getSource(layerId) as GeoJSONSource | undefined;
    if (!source) return;
    source.setData(featureCollection);
}
```

(a) Atualizar *sources* do mapa

```
        let feature = generateFeature(point);
        if (!feature) return;
        if (!vector_points) return;

        vector_points.features.push(feature);

        switch (point.animal_type) {
            case 1:
                sheep_points.features.push(feature);
                break;
            case 2:
                goat_points.features.push(feature);
                break;
            case 3:
                cow_points.features.push(feature);
                break;
            default:
                no_type_points.features.push(feature);
                break;
        }
    }
});
updateSources();
```

(b) Adicionar features às layers

Figura 4.15: Atualizar e gerar *geoJson features*

Marcadores

Antes da utilização das camadas vetoriais, descritas na 4.3.4, foram realizados alguns testes com os marcadores nativos do mapbox.

Um marcador é uma representação visual de uma coordenada especifica no mapa. Estes marcadores são elementos *html* inseridos no *DOM* da página, Figura 4.16. Devido ao baixo desempenho, esta abordagem revela-se inviavel quando se pretende representar mais do que algumas dezenas de pontos. Por esta razão, decidi abandonar esta solução e adotar as camadas vetoriais para a visualização de grandes volumes de dados.

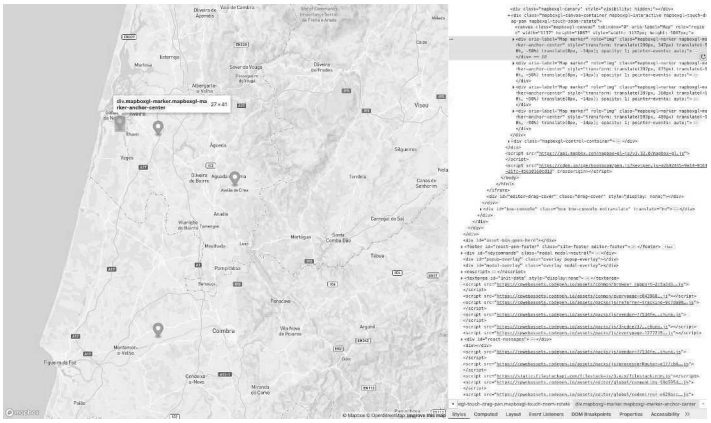


Figura 4.16: Exemplo de marcadores

Map Utils

Para facilitar a criação do mapa e simplificar o desenvolvimento de novos sistemas, foram desenvolvidas algumas classes e interfaces que abstraem e organizam as funcionalidades necessárias. Algumas dessas estruturas podem ser observadas na Figura 4.17.

```ts
export interface GpsMapProps {
    style: mapStyles;
    center: LngLatLike;
    zoom: number;
    points?: GpsSample[];
    markerOnClick?: () => void;
    drawnFeatures?: FeatureCollection;
    editMode?: boolean;
    selectFeature?: Feature | null;
    divClass?: string;
    drawButtons?: boolean;
    heatMapButton?: boolean;
    disableClicking?: boolean;
    mapLoaded?: boolean;
}

export interface MarkerStyle {…
}

/**
 * Possible map styles.
 */
export enum mapStyles {
    standard = 'standard',
    standard_satellite = 'standard-satellite',
    streets_v12 = 'streets-v12',
    outdoors_v12 = 'outdoors-v12',
    light_v11 = 'light-v11',
    dark_v11 = 'dark-v11',
    satellite_v9 = 'satellite-v9',
    satellite_streets_v12 = 'satellite-streets-v12',
    navigation_day_v1 = 'navigation-day-v1',
    navigation_night_v1 = 'navigation-night-v1'
}
```

(a) *Map Interfaces*

```ts
export class mapHelper {
    public constructor(private readonly map: Map) {}

    /**
     * Adds a marker to specified coordinates.
     * Also receives a callback function to execute when the marker is `clicked`
     */
    public addMarker(options: AddMarkerOptions): mapboxgl.Marker {
        const marker = new mapboxgl.Marker(options.style)
            .setLngLat(options.lngLat)
            .addTo(this.map)
            .setPopup(options.popup);

        if (options.callback) {
            marker.getElement().addEventListener('click', options.callback);
        }

        return marker;
    }

    /**
     * Add a source to map.
     * @param source
     */
    public addSource(layerId: string, source: SourceSpecification) {
        this.map.addSource(layerId, source);
    }

    /**
     * updateSources
     */
    public updateSources(layerId: string, featureCollection: FeatureCollection) {
        const source = this.map.getSource(layerId) as GeoJSONSource | undefined;
        if (!source) return;
        source.setData(featureCollection);
    }

    /**
     * Adding new layer. `We need to add a new source before adding a layer.`
     * @param layer layer configuration.
     */
    public addLayer(layer: Layer) {
        this.map.addLayer(layer);
    }
```

(b) *Map classes*

Figura 4.17: *Map utils*

Desenho de poligonos

Uma das funcionalidades mais interessantes que desenvolvi foi a implementação de um sistema de desenho e edição de poligonos sobre o mapa.

Durante a pesquisa de funcionalidades semelhantes, deparei-me com uma ferramenta que aproxima-se com a nossa ideia para esta implementação. Para concretizar esta parte da aplicação, inspirei-me no *geojson.io*[2]. Esta é uma ferramenta *online*, frequentemente utilizada para verificar conteúdos *geojson* e efetuar pequenas alterações.

Esta funcionalidade foi implementada em cima de um *plugin* criado pela comunidade, *mapbox-gl-draw*[4], que permite adicionar funcionalidades de desenho diretamente sobre o mapa. Um dos elementos que deram mais esforço na implementação desta funcionalidade foi a componente de edição.

Este permite ao utilizador remover, mover, aumentar e diminuir poligonos desenhados. Foram igualmente realizadas alterações no *plugin* de desenho, de modo a adaptar toda a sua interface, cores e opções, garantindo a consistência visual e funcional com o restante da aplicação, Figura 4.18.

Figura 4.18: Desenhar polígonos

O sistema funciona conforme representado na Figura 4.19. Esta abordagem revelou-se a mais prática e eficiente para desenvolver a funcionalidade de desenho e edição de poligonos. Além disso, ao estruturar o sistema desta forma, torna-se possível, no futuro, carregar poligonos a partir de uma base de dados, editá-los diretamente no mapa e, posteriormente, guardar as alterações realizadas.

Após definir o fluxo que pretendia para esta funcionalidade desenvolvi os sistemas responsáveis pele funcionamento do sistema. Da mesma forma que foram criadas

[2]geojson.io - https://geojson.io/

agradecimentos /
acknowledgements

Agradeço a todos os colaboradores da Wiseware Solutions pela colaboração e hospitalidade.

Agradeço ao meu orientador Prof. Doutor Ivan Pires por toda a disponibilidade e orientação.

Um agradecimento especial ao orientador Gustavo Corrente pela disponibilidade, apoio na resolução de problemas e abertura a novas ideias.

Por fim quero fazer um agradecimento a toda a Wiseware pela possibilidade de realizar o estágio, que contribiu muito para adquirir competências e conhecimentos ao nível profissional.

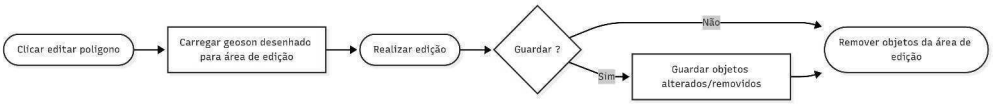


Figura 4.19: *Flow* de edição de poligonos

geoJson featuresCollections para as camadas com pontos, foi também definido uma camada para conter este polígonos.

Ao pressionar o botão de desenhar, altera-se o modo do controlo para desenho. Com este modo ativo, é possível desenhar os polígonos, este modo fica ativado até fechamos o poligono. Ao fechar-se esse polígono, ativa-se um evento, *draw.create*, que obtém a *feature* desenhada, concatena-a com as restantes *features* na camada e desliga o modo de desenho.

Ao pressionar o botão de editar, todas as *features* são carregadas no controlador de desenho e é ativado o modo de edição. Com este modo ativo, o utilizador consegue realizar todo o tipo de alterações, como apagar e redimensionar poligonos.

Durante o modo de edição, não é permitido ao utilizador criar novos poligonos, esta abordagem foi implementada para evitar a ativação do evento *draw.create*, que poderia causar problemas de integridade na nossa camada. Por fim, ao pressionar o botão de guardar, limpamos a antiga camada e guardamos todos os polígonos, tanto os alterados como os que permanecem inalterados. Ao pressionar no cancelar são simplesmente removidos todos os poligonos do controlador de desenho e o modo de edição é desativado. Existe ainda um botão para remover os poligonos selecionados, que remove simplesmente os que se tem selecionado.

```
/**
 * Save all the edit features and updates the source.
 */
function saveEdit() {
    if (!drawnFeatures) return;
    if (singleFeature !== null) {
        let collection = drawControl.getAll();
        if (collection.features.length === 0) {
            drawnFeatures.features.splice(drawnFeatures.features.indexOf(singleFeature), 1);
        } else {
            let feature = collection.features[0];
            drawnFeatures.features[drawnFeatures.features.indexOf(singleFeature)] = feature;
        }
    } else {
        resetFeatureCollection(drawControl.getAll());
    }

    helper.updateSources('edit-source', drawnFeatures);
    map.setLayoutProperty('edit-layer', 'visibility', 'visible');
    drawControl.deleteAll();

    // Updates the ui elements.
    editMode = false;

    singleFeature = null;
}
```

(a) Código guardar *features* editadas

```
/**
 * Function to activate a draw polygon.
 */
export function drawPolygon() {
    if (editMode) return;
    drawControl.changeMode('draw_polygon');
}

/**
 * Cancel any alterations made on the features.
 */
function cancelEdit() {
    map.setLayoutProperty('edit-layer', 'visibility', 'visible');
    drawControl.deleteAll();

    // Updates ui elements.
    editMode = false;
}

/**
 * Function to remove a polygon.
 */
export function removeSelectedPoly() {
    if (drawControl.getSelected().features.length === 0) return;
    drawControl.delete(drawControl.getSelectedIds());
}
```

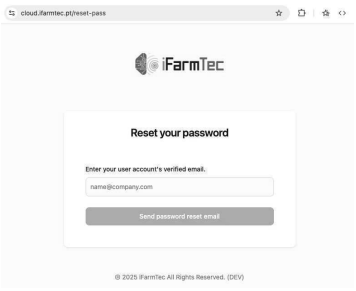
(b) Código desenhar, cancelar e editar

Figura 4.20: Partes do código de edição de poligonos

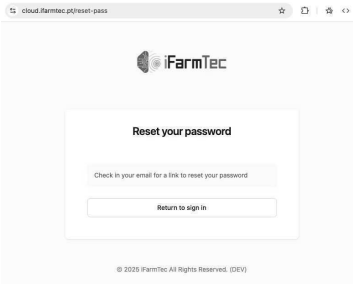
4.3.5 Página *reset palavra-passe*

Uma das tarefas que realizei durante este estágio foi a criação das páginas e do sistema para recuperar palavras-passe. No que diz respeito à interface, foi criada uma nova rota que permite ao utilizador indicar o seu endereço de email para receber um *link* de recuperação, bem como uma página dedicada para a alteração da palavra-passe.

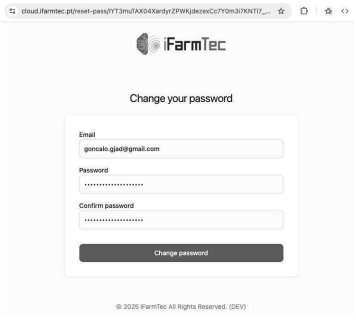
Todo este sistema foi implementado na mesma rota, */reset-pass*. Ao aceder a esta página, o utilizador é redirecionados para um ecrã onde lhe é solicitado a inserir o endereço de *email* associado à sua conta. Após inserir o *email* e clicar no botão *"Send password reset email"*, é enviado um *email* com instruções de recuperação. Caso o envio seja bem-sucedido, é apresentada uma mensagem de sucesso ao utilizador.



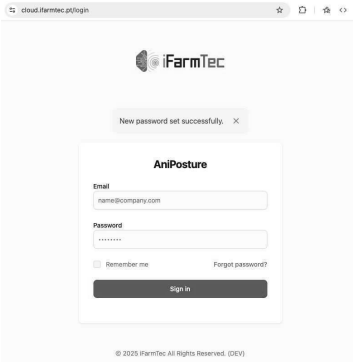
(a) *Reset-pass* enviar email



(b) *Reset-pass* enviar email, sucesso



(c) *Reset-pass* alterar palavra-passe



(d) Página *login*, palavra-passe alterada

Figura 4.21: Páginas *reset-pass*

Após a criação desta primeira tela, foi desenvolvida uma segunda página numa sub-rota, */reset-pass/[token]*. É nesta página que o utilizador pode redefinir a sua palavra-passe. Para tal, deve confirmar o seu endereço de *email* e introduzir a nova palavra-passe.

Foi possível implementar esta funcionalidade graças ao sistema de rotas dinâmicas do *SvelteKit*, ao utilizar o padrão *[slug]*[5]. Este tipo de rota permite criar páginas que recebem parâmetros diretamente na *URL*, o que possibilita o uso da mesma estrutura de rota para diferentes finalidades, basta adicionar o *token* presente no endereço. Após a

alteração bem-sucedida da palavra-passe, o utilizador é automaticamente redirecionado para a página de *login*, conforme ilustrado na Figura 4.21d.

Para tornar todo este sistema funcional e, simultaneamente, garantir que apenas utilizadores autenticados tenham acesso às páginas protegidas, foi implementado um sistema de redirecionamento. Este sistema verifica se o utilizador está autenticado; caso não esteja, é automaticamente redirecionado para a página de *login*. No entanto, existem exceções, nomeadamente as páginas relacionadas com a recuperação de palavra-passe, que permanecem acessíveis mesmo sem autenticação, de forma a permitir que o utilizador consiga redefinir a suas credenciais.

```
let { children } = $props();
const url_path = page.url.pathname;

setContext('url_path', url_path);
const nonLoginRoutes = ['/login', '/reset-pass', 'app-gaime'];

const filter = nonLoginRoutes.filter((str) => url_path.includes(str));

if (sessionService.isAuthenticated() && url_path == '/login') {
    goto('/');
} else if (!sessionService.isAuthenticated() && filter.length === 0) {
    goto('/login');
}
```

Figura 4.22: Redirecionar para o *login*

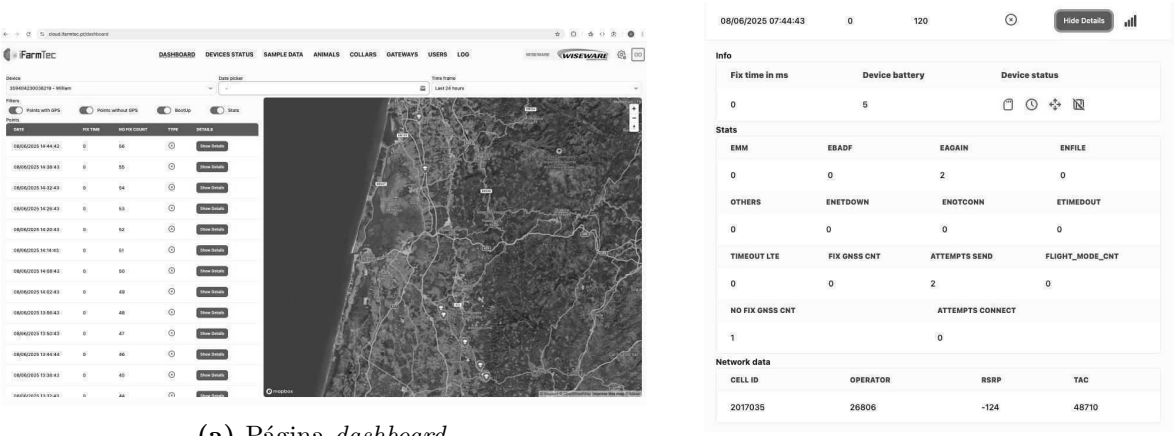
Este código é colocado no ficheiro *layout.svelte* da aplicação, que é responsável por definir uma interface comum a várias páginas. Ao colocá-lo neste ficheiro, garantimos que, sempre que o utilizador tentar aceder a uma página para a qual não tem permissões, ou que não exista, será automaticamente redirecionado para a página de *login* ou para a raiz da aplicação (/).

4.3.6 Página de dashboard

Uma das páginas em que mais trabalhei e ajustei ao longo do estágio foi a página do *dashboard*. Nesta página, são apresentadas várias informações relativas aos pacotes enviados por cada dispositivo.

Como se pode observar na Figura 4.23a, esta página inclui um mapa que representa, através de indicadores, os pontos com coordenadas GPS. À esquerda do mapa, encontra-se uma lista com todos os pontos registados e, na parte superior da página, estão disponíveis opções para seleção do dispositivo e da data pretendida.

Na Figura 4.23b, é possível visualizar o conteúdo apresentado ao clicar no botão de detalhes. Esta secção mostra informações adicionais sobre cada um dos pacotes, incluindo o estado dos periféricos do dispositivo, a data e hora de criação do pacote, dados sobre a rede, entre outros detalhes. Existe também uma secção que permite filtrar os pacotes existentes, o que possibilita ao utilizador visualizar apenas os pacotes com dados GPS, sem GPS, pacotes de *boot* ou aqueles que contenham informações adicionais.



(a) Página *dashboard*

(b) Detalhes de pacotes

Figura 4.23: Página de *dashboard* e pacotes detalhes

4.3.7 Página de status

Esta página é responsável por apresentar os dispositivos, bem como o uptime diário de cada um deles.

Componente device stats

Este componente foi desenvolvido com o propósito de mostrar informações sobre cada um dos dispositivos existentes. Para além de dados gerais, como nível da bateria, versão do *hardware*, versão do *firmware*, o número de ficheiros e a data da ultima comunicação, o componente disponibiliza também informação relativa ao *uptime* do dispositivo.

O *uptime* é calculado com base nos pedidos de GPS realizados pelo dispositivo. Cada dispositivo efetua um pedido de localização a cada 6 minutos, o que resulta num máximo teórico de 240 pedidos por dia. Com base neste valor, é possível determinar a percentagem de tempo em que o dispositivo esteve operacional num determinado dia, Figura 4.24.

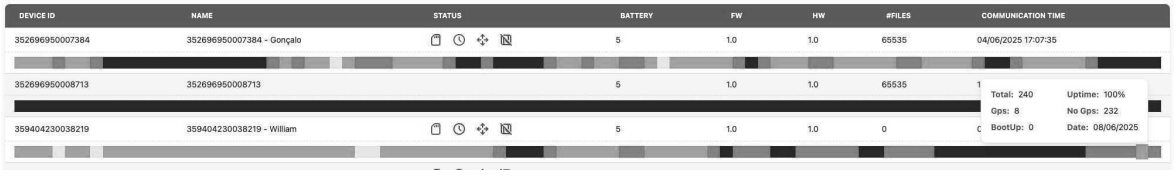


Figura 4.24: Componente *Device stats*

Página device status

Esta página contem, de forma simples, uma tabela composta por múltiplos componentes que apresentam informação relativa a todos os dispositivos registados nesse

tenant. Como em muitas outras páginas, existe uma caixa de entrada que permite filtrar os dispositivos pelo nome, Figura 4.25.

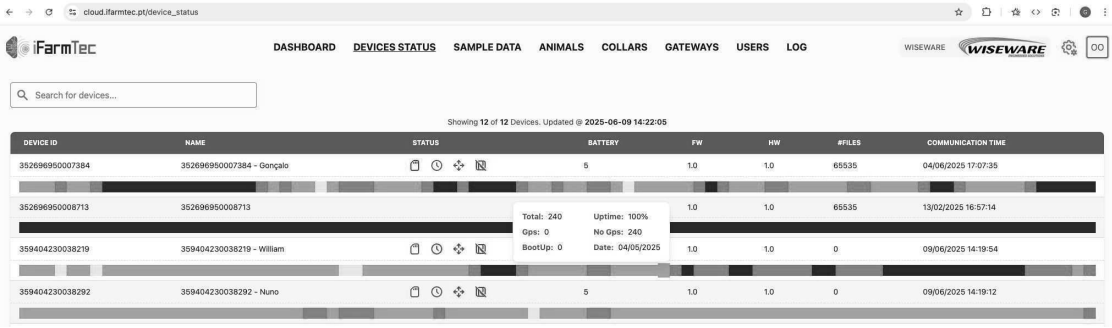
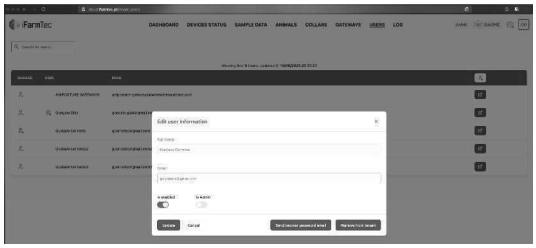


Figura 4.25: Página *device status*

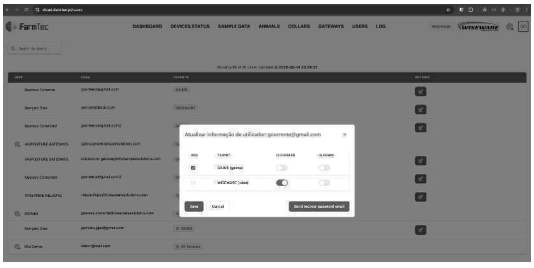
4.3.8 Página de utilizadores

A página foi criada a partir de outra já existente na aplicação. Anteriormente, existia uma página exclusiva para os *Owners* do projeto, onde era possível remover e adicionar utilizadores aos *tenants*, desativá-los, promovê-los a administradores de cada *tenant* e enviar um email para recuperação da palavra-passe. Foram efetuadas alterações nessa página com o intuito de manter um design mais consistente.

A nova página destina-se exclusivamente aos administradores dos *tenants*. Nela, os administradores podem adicionar novos utilizadores ao *tenant*, desativá-lo, promovê-lo a administrador e enviar emails para recuperação da palavra-passe.



(a) Página *user* para *tenant admins*



(b) Página *user* para *owners*

Figura 4.26: Páginas para gerir utilizadores

Nesta fase de desenvolvimento, procedi a uma pequena alteração na *nav bar* para que esta torne-se dinâmica, adapte-se às permissões dos utilizadores. Assim, os utilizadores normais não têm acesso ao link para gestão de utilizadores, os administradores são redirecionados para */tenant_users* e os *owners* para */users*.

4.3.9 Página de gráficos

Nesta secção é apresentado um gráfico com os dados do acelerómetro e da temperatura do dispositivo, ao longo de um determinado intervalo de tempo.

Componente Echarts

Foi desenvolvido um componente para o motor gráfico escolhido, Seção 4.2.2, pela sua flexibilidade e capacidade de visualização interativa. Este componente foi concebido de forma reutilizável, o que permite representar diferentes tipos de dados provenientes dos sensores, ao utilizar gráficos distintos, como linhas ou barras, conforme a necessidade.

O componente foi desenhado com o foco no desempenho e facilidade de utilização, o que garante assim uma boa experiencia, tanto para o utilizador final como para o programador. O gráfico também permite a atualização dinâmica dos dados e a interação com o utilizador através de funcionalidades como zoom e seleção de intervalos.

Os dados provenientes do acelarômetro são apresentados de forma normalizada, ao utilizar a seguinte formula: $\sqrt{x^2 + y^2 + z^2}$.

Esta normalização permite obter um valor que representa a intensidade de movimento do animal, o que facilita uma análise visual mais clara e concisa. Simultaneamente, a variação da temperatura é apresentada no mesmo gráfico, mas num eixo distinto, o que possibilita a correlação entre a temperatura e o movimento do animal.

Surgiu um contratempo durante a realização desta página. Para apresentar os dados de forma mais fidedigna, de modo a perceber o *uptime* e *downtime* dos dispositivos, foi necessário mostrar, propositadamente, lacunas nos dados. Para tal, foi necessário criar uma lista com todas os minutos existentes durante o período de pesquisa, o que gera 1440 registos para um único dia, 24 horas. Posteriormente, verificava-se se cada uma das amostras pertencia a um determinado minuto. Se pertencia, apresentavam-se os valores do acelerómetro e da temperatura com a data, hora, minuto e segundo da sua recolha. Caso contrário, inseria-se um valor nulo, de modo a assinalar a ausência de amostra nesse momento.

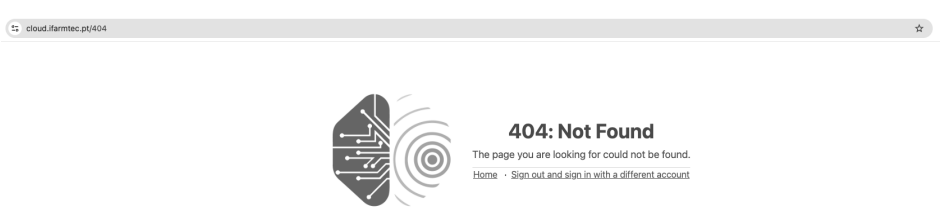
```
// Creates an array between 2 dates, every minute.
while (fromCalcDate <= toCalcDate) {
    evMinute.push(new Date(fromCalcDate.setSeconds(0)).setMilliseconds(0));

    fromCalcDate = new Date(fromCalcDate);
    fromCalcDate.setMinutes(fromCalcDate.getMinutes() + 1);
}

// Map to store all the samples referanced by their date.
let dataByMinute = new Map<number, Sample>();

// Store all the samples, references by date.
sampleList.forEach((item) => {
    if (item.date) dataByMinute.set(new Date(item.date).setSeconds(0), item);
});

// Merge data between evMinute and dataByMinute. Returns a new array referencing every sample date with evMinute date.
const mergedData = evMinute.map((day) => {
    const minuteData = dataByMinute.get(day);
    if (minuteData) {
        let normalizedMean = 0;
        if (minuteData.data) {
            normalizedMean = Math.sqrt(
                Math.pow(minuteData.data.accX.mean, 2) +
                    Math.pow(minuteData.data.accY.mean, 2) +
                    Math.pow(minuteData.data.accZ.mean, 2)
            );
        }

        return {
            timestamp: minuteData.date,
            originalData: minuteData,
            normalizedMean: normalizedMean
        };
    } else {
        // Return null, in originalData and normalizedMean if we can't find the data in the map.
        return {
            timestamp: day,
            originalData: null,
            normalizedMean: null
        };
    }
});
```

Figura 4.27: Codigo verificar acelarômetro e temperatura

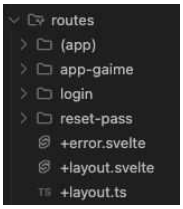
4.3.10 Página de erro

Foi criada uma nova página de erro, mais alinhada com o design da aplicação. Para a sua implementação, conforme ilustrado na Figura 4.28a, foi utilizada uma funcionalidade nativa do *sveltekit*. Ao criar uma página de erro na raiz da aplicação, como demonstrado na Figura 4.28b, todos os erros que ocorram durante a navegação são automaticamente redirecionados para essa página.

Desta forma, foi possível desenvolver uma página de erro genérica, aplicável a qualquer secção da aplicação, cuja apresentação adapta-se dinamicamente consoante o tipo de erro ocorrido.



(a) Página de erro



(b) *error.svelte*

Figura 4.28: Erro

4.3.11 Componente data hora

A biblioteca de componentes, *flowbite*, não contém nenhum componente que permita ao utilizador selecionar uma data e hora para cada dia. Decidi então desenvolver um componente próprio. Para tal, reutilizei o componente de seleção de data do *flowbite* e adicionei um campo que permite a seleção das horas. Neste caso, o primeiro campo de hora corresponde ao primeiro dia, e o segundo ao segundo dia. Assim, conseguimos selecionar de forma mais precisa os intervalos desejados para a pesquisa.

Figura 4.29: Componente *Datepicker*

Tive ainda de criar algumas funcionalidades no componente para definir automaticamente as horas e as datas ao selecionalas.

```
$effect(() => {
    if (dateFrom && dateTo) {
        dateFrom?.setHours(0, 0);
        dateTo?.setHours(23, 59, 59, 999);
    }
});

function handleDateSelect() {
    if (changeValue && dateFrom && dateTo) changeValue();
}
```

Figura 4.30: Selecionar data no componente *Datepicker*

4.3.12 Rotas sem inicio de sessão

Uma das últimas tarefas que foram-me atribuídas durante o estágio foi a criação de duas rotas que permitissem a visualização de dados dos dispositivos sem necessidade de autenticação. A nova rota foi chamada de */app-gaime* e inclui duas sub-rotas: */map* e */chart*.

Figura 4.31: Página *app-gaime/map*

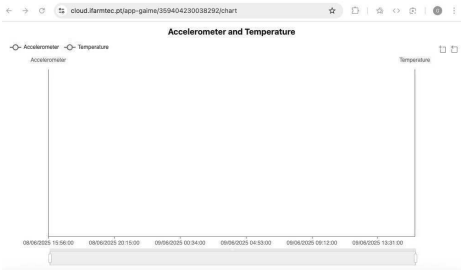


Figura 4.32: Página *app-gaime/chart*

Para selecionar o dispositivo do qual queremos ver os dados, é necessário introduzir o seu *id* na barra de pesquisa, */app-gaime/[device_id]*.

Por padrão, a rota do mapa , Figura 4.31, solicita ao servidor todos os pontos com coordenadas GPS registados nas últimas 24 horas. De forma semelhante, a rota do gráfico, Figura 4.32, recupera informações relativas ao acelarômetro e temperatura no mesmo intervalo de tempo.

A quantidade de horas solicitadas ao servidor podem ser alteradas através de um *query param*, ao utilizar o formato *?h=[número de horas]*.

4.3.13 Guardar filtros

Para melhorar a usabilidade da aplicação, foram implementadas funcionalidades que tornam permanentes os filtros selecionados pelo utilizador. Dessa forma, ao mudar de página ou fechar a aplicação, as opções permanecem previamente selecionadas. Esta funcionalidade foi construída com base numa biblioteca existente, conforme descrito em *svelte-persisted-store* [6].

A referida biblioteca simplifica a utilização das *stores* do *Svelte*, armazena informações no *local storage* do *browser*. Essa abordagem permite ao utilizador navegar entre as páginas sem perder os filtros previamente definidos.

```
<div>
    <form class="flex !gap-0">
        <Search
            size="lg"
            class="py-2.5 rounded border-primary-500 sm:w-md"
            placeholder="Search for devices..."
            bind:value={$searchDeviceStatus}
        />
    </form>
</div>
```

Figura 4.33: Utilização das *stores* num ficheiro *.svelte*

```
// Dashboard
export const searchDeviceDashboard = persisted('searchTermDeviceDashboard', '');
export const searchTimeFrameDashboard = persisted('searchTimeFrameDashboard', '');
export const searchDateFromDash = persisted<Date | null>('searchDateFromDashboard', null);
export const searchDateToDash = persisted<Date | null>('searchDateToDashboard', null);

// Dashboard filter
export const gpsPointsStore = persisted<boolean>('gpsPointsStore', true);
export const noGpsPointsStore = persisted<boolean>('noGpsPointsStore', true);
export const bootUpPointsStore = persisted<boolean>('bootUpPointsStore', true);
export const statsPointsStore = persisted<boolean>('statsPointsStore', true);

// Device status
export const searchDeviceStatus = persisted('searchDeviceStatus', '');
```

Figura 4.34: Criação das *stores*

4.4 Servidor da aplicação

Este projeto utiliza um servidor desenvolvido em *Java*, responsável por armazenar os dados enviados pelos dispositivos, criar e autenticar utilizadores, assim como obter informações através de diferentes rotas, entre outras funcionalidades.

4.4.1 O que é o *backend* ?

Como mencionado anteriormente, o *backend* é desenvolvido em *Java* e adota uma arquitetura do tipo *Controller-Service-Repository*. Esta abordagem divide a aplicação em três camadas: os *Controllers* gerem as requisições e respostas, os *Services* implementam a lógica de negócio e os *Repositories* cuidam do acesso à base de dados. Para implementar esta arquitetura de forma mais eficiente e com melhor desempenho, foram utilizadas diversas bibliotecas de apoio.

Vert.x

O *Vert.x* é uma *biblioteca* assincrona e reativa para o JVM, utilizada no desenvolvimento de aplicações concorrentes, escaláveis e de elevado desempenho, como *APIs* e sistemas baseados em eventos. No nosso caso o *Vert.x* é utilizado para desenvolver a API do projeto, definir rotas, criar serviços, enviar *email*, entre outras funcionalidades.

Ebean ORM

A componente do *Repository* (repositório) é desenvolvida com o auxílio do *Ebean ORM*. Esta *framework* suporta múltiplos níveis de abstração de consultas, incluindo consultas ORM, consultas SQL, entre outros. O *Ebean* é compatível com diversos sistemas de bases de dados, como *Postgres* e *MySQL*. Além disso, utiliza anotações de mapeamento JPA e disponibiliza mapeamentos adicionais, como *@DBJson* e *@DBArray*.

Postgres

O *Postgres* é o sistema de base de dados utilizado neste projeto. Não só permite a utilização de pontos GPS, como destaca-se pela sua rapidez, escalabilidade e robustez. Adicionalmente, garante a alta disponibilidade e integridade dos dados, esta capacidade promove a execução de consultas complexas e a implementação de estratégias de replicação, o que assegura a continuidade do serviço mesmo em ambientes de elevada concorrência e carga.

Funcionamento do projeto

Neste caso, o projeto funciona da seguinte forma. Existem dois "projetos"dentro do mesmo sistema, ou seja, há um projeto base que contém o conjunto de funcionalidades comuns a outros projetos, incluindo a autenticação de utilizadores, o sistema de *tenants*, o envio de *emails*, entre outros.

Dentro deste projeto base, encontra-se um sub-projeto específico, no nosso caso referido por *aniposture*, que reúne as ações específicas, tais como a gestão de dispositivos, gateways, animais, o servidor *UDP* para receção pacotes GPS, entre outros.

4.4.2 Rota para obter número de pedidos por dispositivo

Este foi um dos trabalhos mais interessantes que realizei. Fui "desafiado"a alterar uma das rotas da aplicação para a tornar mais útil e com mais informação.

A rota preexistente exibia a quantidade de pedidos efetuados por cada dispositivo. Permitia inserir as datas, desde e até quando, e o dispositivo a pesquisar. Retornava os dados de cada dispositivo por dia, mas apresentava um senão: os dados do dia atual apenas estavam disponíveis no dia seguinte.

Apesar de cumprir a sua função inicial, que era mostrar como os dispositivos comunicavam, a rota tinha pontos negativos, tais como a consulta dos dados ser apenas possível no dia seguinte ou a necessidade de especificar uma data, entre outros.

Para a melhorar, realizei as seguintes alterações: para cada pedido, enviam-se agora apenas dois dados, o dispositivo e a quantidade de dias a pesquisar. A rota passa a retornar dados detalhados, como a quantidade de pontos GPS, de pontos sem GPS e os registos de *boot*, que são enviados quando o dispositivo liga-se à rede, e também o total de pontos por dia. Com estas alterações, são também reportados os dias em que não houve comunicação. Desta forma, é possível identificar se o dispositivo apresentou alguma anomalia nesse dia.

```
SELECT  date_trunc('day', timestamp) as ts_day, count(*) as count  FROM $SCHEMA.gps_sample
    where
    device_id = '$DEVICE_ID' and
    (
    details->>'type' = 'GPS'
    or
    details->>'type' = 'NOGPS'
    )
    and
    timestamp between date_trunc('day', timestamptz '$FROM')
    AND date_trunc('day', timestamptz '$TO')
    group by device_id , date_trunc('day', timestamp) order by ts_day;
```

(a) SQL para rota */countDaily*, antiga rota.


```
{
    "date": "2025-06-15 00:00:00.0",
    "count": 240
},
{
    "date": "2025-06-16 00:00:00.0",
    "count": 240
},
{
    "date": "2025-06-17 00:00:00.0",
    "count": 240
},
{
    "date": "2025-06-18 00:00:00.0",
    "count": 240
}
```


(b) Resultado da *query* antiga

Figura 4.35: Antiga *query* para obter o estado dos dispositivos

SQL

Para concretizar esta rota, algo mais complexa que outras, optei por utilizar SQL puro em vez da ORM. As ORM são bastante úteis para rotas mais simples, mas, neste caso, o seu uso representaria provavelmente um problema.

palavras-chave Tecnologias, software, equipa, tarefas, conhecimentos.

resumo

O presente relatório compreende todas as atividades realizadas na empresa Wiseware Solutions durante o estágio de 466 horas de duração. Durante o estágio foram definidos certos objetivos: adquirir competências acerca do funcionamento da entidade, integração com as tecnologias utilizadas, desenvolvimento de software e elaboração do relatório final de estágio. Estes objetivos permitiram aplicar os conhecimentos adquiridos no decorrer do curso.

A primeira parte do relatório apresenta uma descrição sucinta da empresa juntamente com uma breve descrição das tarefas nela realizadas. Em seguida são descritas as atividades planeadas, para a realização do estágio. Numa terceira fase descreve-se detalhadamente cada uma das atividades desenvolvidas, as dificuldades sentidas na sua realização e as competências adquiridas com as suas realizações. Para finalizar é feita uma analise das experiências vivenciadas no estágio e as expectativas para o futuro com base nos conhecimentos obtidos.

Esta *query* integra o seguinte: duas relações temporárias, uma para obter todos os dias selecionados no período definido e outra para recolher os dados do dispositivo. Posteriormente, é efetuada uma *query* sobre as duas tabelas temporárias, estabelecendo a relação entre cada dia, da tabela de dias, e os pontos da tabela de pontos. Realiza-se a filtragem por cada tipo de ponto e a soma de todos eles, de modo a obter os dados necessários.

```
with
date_series as (
    select generate_series(
        (timezone('utc', NOW()) - INTERVAL '$TIME_FRAME DAYS')::date,
        'today'::date,
        '1 day'::interval
    )::date as date
),
device_records as (
    select
        timestamp::date as date,
        count(*) filter (where details->>'type' = 'GPS') as gps_count,
        count(*) filter (where details->>'type' = 'NOGPS') as nogps_count,
        count(*) filter (where details->>'type' = 'BOOTUP') as bootup_count
    from $SCHEMA.gps_sample
    where device_id = :device_id
    and timezone('utc', timestamp) between
        date_trunc('day', timezone('utc', NOW()) - INTERVAL '$TIME_FRAME DAYS')
        and date_trunc('minute', timezone('utc', NOW()))
    group by device_id, date
)
select
    ds.date,
    coalesce(dr.gps_count, 0) as gps_count,
    coalesce(dr.nogps_count, 0) as nogps_count,
    coalesce(dr.bootup_count, 0) as bootup_count,
    coalesce(gps_count,0) + coalesce(nogps_count, 0) as total
from date_series ds
left join device_records dr on ds.date = dr.date
order by ds.date asc;
```

(a) SQL para rota */countFrame*, nova rota.


```
{
    "date": "2025-06-17",
    "gps_count": 62,
    "no_gps_count": 178,
    "boot_up_count": 0,
    "total": 240
},
{
    "date": "2025-06-18",
    "gps_count": 54,
    "no_gps_count": 186,
    "boot_up_count": 0,
    "total": 240
},
{
    "date": "2025-06-19",
    "gps_count": 0,
    "no_gps_count": 92,
    "boot_up_count": 0,
    "total": 92
}
```


(b) Resultado da nova *query*

Figura 4.36: Nova *query* para obter o estado dos dispositivos

No caso do dia atual, efetua-se uma pesquisa de todos os pontos registados até às zero horas desse dia.

Esta rota teve como propósito alimentar uma das páginas da interface do projeto, Subseção 4.3.7. Enquanto a rota anterior mostrava apenas informações sobre os pontos, esta exibe dados detalhados sobre os dias e os pontos registados em cada um.

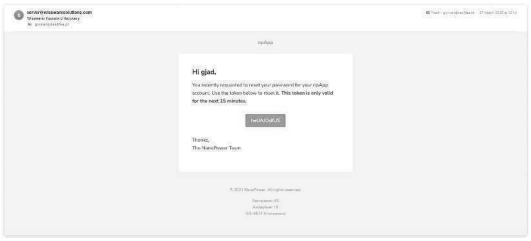
Esta reestruturação não só otimizou a rota em termos de dados, mas também conferiu um valor analítico superior. A capacidade de identificar dias sem comunicação, permite diagnosticar falhas de conexão ou potenciais anomalias nos dispositivos, algo que a versão anterior não proporcionava.

A opção por SQL em detrimento da ORM revelou-se essencial para assegurar a agregação complexa de diversos tipos de pontos, com GPS, sem GPS e de *boot*. A deteção de períodos sem atividade fornece à interface dados mais robustos e detalhados para o utilizador.

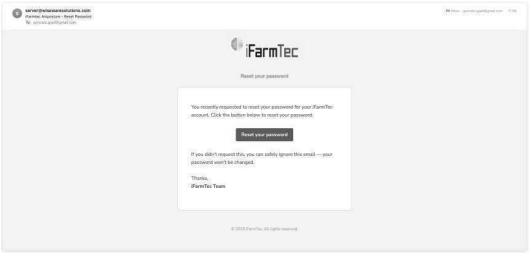
4.4.3 Sistema de envio de emails

O sistema de emails foi outra das alterações que realizei durante o estágio. No próprio *backend* já existia uma configuração de email em funcionamento, mas alguns problemas impediam o seu envio.

Foram corrigidos alguns problemas nas rotas de *reset-pass* e *change-pass*. A rota *reset-pass* é responsável pelo envio do email ao utilizador, aqui foram realizadas alterações no *template* do email. O *template* existente não alinhava-se com o *design system* da aplicação, nem com o seu propósito.



(a) Antigo template de email



(b) Novo template de email

Figura 4.37: Emails enviados pelo servidor

A rota *change-pass*, por sua vez, permite a alteração da palavra-passe. Aqui, foi identificado um problema que considero invulgar: ao criar o *hash* da nova palavra-passe, a função de *hash* era invocada tanto no *controller* como no *service*, o que resultava na criação do *hash* do *hash*. Por consequencia, era impossível aceder com a palavra-passe definida.

Foram ainda implementadas outras alterações nesta rota, que incluem a verificação do *token* e a alteração do seu formato, para que contenha apenas letras minúsculas, maiúsculas, hífens e *underscores*. Não deve conter caracteres especiais, como os símbolos de igual ou de interrogação, dado que estes podem comprometer a estrutura dos *links* no *frontend*. A presença de um ponto de interrogação num URL, por exemplo, pode ser interpretada como o início de parâmetros, o que não corresponde ao formato pretendido para os *tokens*.

No que respeita à verificação do *token*, foram introduzidas alterações na sua forma de validação. Cada *token* tem uma validade de 15 minutos a contar do momento da que é inserido na base de dados. Se a data atual exceder em 15 minutos a data de criação do *token*, este é considerado inválido. Em caso de atualização bem sucedida da palavra-passe, não é necessário efetuar esta verificação, uma vez que o *token* é removido após a atualização da palavra-passe. Assim, se o utilizador tentar reutilizar o mesmo *token*, não o conseguirá fazer.

As alterações asseguraram não só a correção de falhas técnicas, mas também a melhoria da experiência do utilizador e a robustez da segurança da aplicação. Adequar

os templates de email ao *design system* da aplicação contribuiu para uma comunicação mais clara para com os utilizadores. Simultaneamente, as medidas de validação do *token*, tanto no formato como na validade, em conjunto com a correção do erro do *hash do hash*, mitigam riscos de segurança, protegem a integridade dos dados do utilizador e fortalecem a confiança no sistema de autenticação e recuperação de palavra-passe.

4.4.4 Rotas para aceder a dados sem *login*

Foram criadas duas rotas para a aplicação que permitem aceder a algumas informações sem necessidade de *login*. Estas rotas foram desenvolvidas no *backend* para suportar a interface */app-gaime*, conforme detalhado na Subseção 4.3.12.

Criou-se uma rota no *controller* do *gpsSample*, e outra no *controller* da *sample*. Ao contrário das outras rotas, nas quais envia-se no *header* o *tenant* que o utilizador tem selecionado, neste caso pretendemos obter informações de um *tenant* específico. Para tal, tive de sobrepor o *tenant* selecionado no serviço e, em seguida, efetuar o pedido à base de dados através da ORM.

Existe um *bug* no *Ebean* que faz com que a modificação manual do *tenant* não afete o *schema* selecionado. Por isso, foi necessário adicionar o *tenant* manualmente ao pedido.

```
/**
 * Controller responsible for the /noAuth route.
 * @param context
 */
public void noAuth(WiseRoutingContext context) {
    WiseParams params = context.request.queryParams();
    String deviceId = params.getString(deviceIdParam);
    Instant from = params.getInstant(fromParam);
    Instant to = params.getInstant(toParam);
    Integer size = params.getInteger(sizeParam, Consts.PAGE_SIZE_DEFAULT);

    TenantOverride.use(tenant:"wise");
    var list = service.noAuthGpsSamples(deviceId, from, to, size);

    context.response.setOk(list);
}
```

(a) *Controller* gpsSample

```
public List<GpsSample> noAuthGpsSamples(String deviceId, Instant from, Instant to, Integer size) {
    QGpsSample q = query();

    // Workaround to make query on the correct tenant.
    q.setBaseTable(WiseSchema.getSchema() + ".gps_sample");

    if (deviceId != null)
        q.device.id.ieq(deviceId);

    if (from != null)
        q.id.timestamp.ge(from);

    if (to != null)
        q.id.timestamp.le(to);

    if (size != null)
        q.setMaxRows(size);

    q.orderBy().id.timestamp.asc();
    return q.findList();
}
```

(b) *Service* gpsSample

Figura 4.38: Codigo para criação das rotas */noAuth*

A criação destas rotas ampliou as funcionalidades da interface, */app-gaime*, uma vez que que permitem aceder aos dados dos pontos GPS, do acelarômetro e da temperatura sem necessidade de iniciar sessão.

4.4.5 Dados do *acc* e de temperatura no sample

Durante as últimas semanas de estágio, foi-me pedido para alterar a estrutura de um pacote enviado pelos dispositivos, de modo a incluir a informação do acelerómetro e da temperatura.

Para tal, tive de alterar o servidor UDP existente no *backend*. Antes de mais, importa explicar como funciona a comunicação entre os dispositivos e o *backend*.

Cada dispositivo envia os dados por UDP para o servidor. Ao receber esses dados, o servidor processa-os e armazena-os na base de dados. No final, envia uma mensagem de volta para o dispositivo, a confirmar que guardou todos os dados. Ao receber esse pacote, o dispositivo apaga todos os dados do seu *buffer* interno. Esta foi a parte mais calma de todo o processo: o servidor recebia os dados em *bytes*, processava cada *byte* e guardava-os. No entanto, surgiu um problema após termos o servidor em funcionamento por um dia.

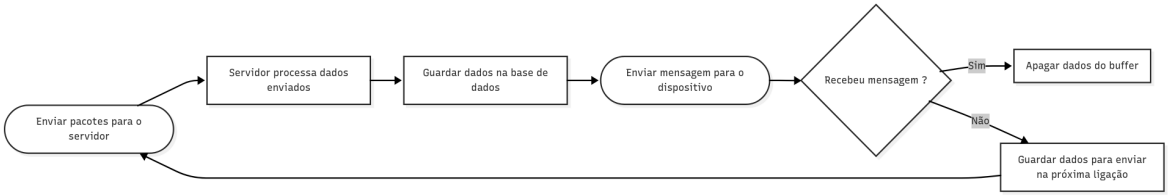


Figura 4.39: Flow envio de dados dos dispositivos para o servidor

Pelas comunicações serem realizadas através de UDP, pode acontecer que alguns pacotes sejam perdidos. Ocorria que, por vezes, os dispositivos não recebiam a mensagem de resposta do servidor. Isto fazia com que não apagassem a informação e tentassem reenviá-la numa próxima ligação. Ao enviar estes dados, que já estavam inseridos na base de dados, resultava num problema de integridade de dados, com a duplicação de tuplos.

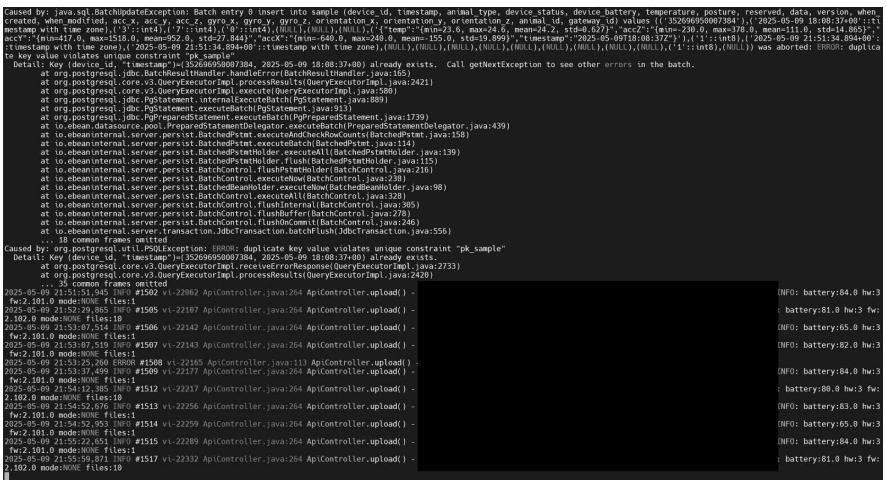


Figura 4.40: Erro de integridade ao guardar dados

Para resolver este problema, temos de adicionar uma *flag* na ORM, que permite ignorar dados já inseridos no momento da nova tentativa de inserção. Apesar de o problema parecer resolvido, existe um *bug* no *Ebean* que o impede de reconhecer os campos unicos. Desta forma, mesmo que se configure a ignorância de dados duplicados, é necessário, primeiramente, modificar o serviço para que este identifique quais os campos que são únicos.

```
// Classe sample, com unique constrains.
@Entity
@UniqueConstraint(columnNames = { "device_id", "timestamp" })
@Index(columnNames = { "timestamp" })
public class Sample extends WiseModel<SampleId> {


// Resolução do bug, onde o Ebean não percebe as unique constrains.
@Service
public class SampleService extends WiseService<Sample, SampleId, QSample> {
    protected SampleService() {
        super(Sample.class, () -> new QSample(), "device_id, timestamp");
    }
}


// Criar um novo registo sample na base de dados.
sampleService.createOne(sample, OnConflict.IGNORE);
```

Figura 4.41: Parte do codigo usado para registar uma nova *sample*

A introdução destas informações na base de dados beneficia não só o utilizador final, que pode avaliar o estado de cada animal, mas também o engenheiro, que consegue identificar problemas e padrões que estejam errados com os sensores.

A perda de integridade da informação, que resulta da perda de pacotes UDP e do consequente reenvio de dados, comprometia a analise da informação sobre o comportamento e o estado dos animais. Com a resolução deste problema e ao garantir a integridade dos dados, reforçamos a fiabilidade do sistema.

Ao contornar o *bug* do *Ebean* através da identificação prévia dos campos únicos no serviço, Figura 4.41, é essencial para assegurar a continuidade da informação dos pontos GPS, não GPS e de *boot*.

4.4.6 Correção de um *bug* que duplicava dados

Durante o estágio, eu e alguns colegas de trabalho reparámos na duplicação de certos dados, que deveriam ser armazenados apenas uma vez. Isto acontecia quando um dispositivo ficava *offline* durante algum tempo. Nesse período, o dispositivo persistia na tentativa de ligação à rede e envio de pacotes para o servidor. Por não conseguir enviá-los, guardava-os num *buffer* interno para a retransmitir quando voltasse a ligar-se. Ao permanecer *offline* durante algum tempo, o dispositivo gera um pacote com informações sobre o período de desconexão, o número de tentativas de ligação efetuadas, entre outros dados.

Ao ligar-se finalmente à rede, cada dispositivo enviava um pacote de localização, com dados GPS ou sem GPS, caso não obtivesse localização. No momento dessa ligação, apenas o pacote criado nesse instante devia incluir as informações da rede. Contudo, todos os outros pacotes, que tinham sido gerados enquanto o dispositivo esteve *offline*, guardavam, indevidamente, essa mesma informação de rede.

Isto constitui um problema, pois pode induzir em erro quem verifica a informação, uma vez que, na realidade, apenas um desses pacotes conseguiu ligar-se à rede, pelo que só esse devia apresentar os dados de rede.

Para resolver este problema efetuaram-se várias alterações do servidor UDP do *backend*. Sempre que chega novos pacotes ao servidor, estes dados são processados da seguinte forma, primeiramente, sempre que um dispositivo estabelece ligação, envia um pacote chamado *DEVICE_INFO*, que inclui informações do estado atual, nível da bateria, o estado dos módulos internos, a versão de *firmware*, entre outras informações.

A ordem de envio de pacotes, permite ao servidor perceber o estado do dispositivo antes da ligação, se esteve *offline* ou *online*. Se em seguida o servidor receber um pacote de *stats*, informações sobre o dispositivo durante o tempo que esteve *offline*, conclui-se que o dispositivo esteve *offline* durante periodo de tempo. Neste cenário, o pacote seguinte, um GPS ou sem GPS, deverá guardar as informações da rede. Já os restantes pacotes, deste mesmo conjunto, não terão informações de rede pois são dados registados enquanto o dispositivo encontrava-se *offline*. É ainda importante mencionar que o dispositivo envia todos os dados núm único pacote UDP, composto por um determinado número de *bytes*. Cada segmento deste bloco corresponde a um tipo especifico pacote, o que assegura a sua distinção e o processamento correto dos dados.

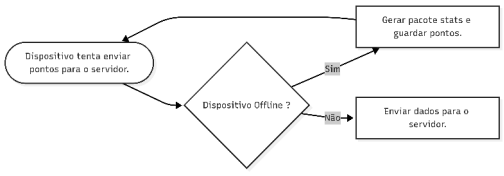


Figura 4.42: Flow de envio quando o dispositivo encontra-se *offline*

CAPÍTULO 5

Analise Resultados

5.1 Análise de objetivos

Nesta secção, farei uma análise a todos os objetivos definidos no plano, às diferenças entre o trabalho realizado e o trabalho definido e o porquê dessas diferenças, conforme os objetivos definidos no plano A.1.

5.1.1 Acolhimento e introdução ao desenvolvimento

Este foi o primeiro objetivo definido e realizado neste estágio. Tinha como finalidade introduzir-me às ferramentas e tecnologias utilizadas na empresa. Esta fase foi consideravelmente mais curta do que o planeado. Muitas destas ferramentas já eram-me conhecidas, e em algumas delas já tinha trabalhado anteriormente em projetos do curso e outras foram lecionadas em unidades curriculares. Refiro-me ao *java*, ao *svelte* e ao *git*. Às restantes tecnologias adaptei-me rapidamente, e ao fim de dois dias já as conseguia utilizar para desenvolver.

5.1.2 Levantamento de requisitos

Esta fase, que totalizou 40 horas, foi fundamental para a minha integração no projeto, o que terminou num entendimento da arquitetura do sistema e das regras do negócio. Primeiro, procurei compreender as tecnologias e ferramentas utilizadas no projeto. Dado que a área do projeto onde trabalhei abrangia partes distintas, *frontend* e *backend*, foi essencial conhecer o código já desenvolvido pelos outros colaboradores da empresa, as diferenças entre as versões das tecnologias e a criação de novas ferramentas.

Apesar de ter sido uma fase demorada, em comparação com a anterior, foi aqui que aprendi como funcionava cada parte do projeto e a forma correta de implementar novas funcionalidades.

5.1.3 Escolher a API de Mapas

Esta foi uma tarefa muito interessante. Nela consegui pesquisar e realizar uma análise sobre as tecnologias lecionadas numa das unidades curriculares de que mais gostei de realizar neste curso, sistemas de informação geográfica. Nesta análise verifiquei as funcionalidades de algumas das tecnologias disponíveis no mercado, comparei-as e selecionei a que se adequava melhor ao nosso projeto, com base nos requisitos definidos pela empresa. Após algumas horas de testes e pesquisa, cheguei a uma conclusão que se mostrou acertada no momento da sua implementação, a escolha do *mapbox*.

Um aspeto que diferenciou-se um pouco foi a quantidade de horas previstas. Apesar de estarem previstas 120 horas, esta pesquisa foi realizada de forma mais rápida, devido à necessidade de apresentar os pontos na aplicação e ao facto de, em equipa, chegarmos a um consenso de que aquela tecnologia era a correta para o projeto.

5.1.4 Desenvolvimento da UI

Uma das etapas mais importantes deste estágio foi o desenvolvimento da *UI*. Neste objetivo, foi-me dada a responsabilidade de alterar a interface, efetuar melhorias de usabilidade e incorporar as funcionalidades dos mapas e gráficos. Estas foram as primeiras tarefas que realizei, a implementação do mapa teve a maior prioridade, onde procurei criar componentes reutilizáveis, elaborar documentação para outros programadores e definir formas de conversão de dados, entre outros aspetos.

Da mesma forma que criei estas funcionalidades, também alterei algumas que já existiam, como a adição de conteúdo às páginas, a modificação de alguns *layouts* e a alteração do *design system*.

Acredito que no fim desta etapa, que decorreu de forma intercalada com outras, as horas previstas foram cumpridas e, provavelmente, até ultrapassadas. Isto aconteceu pela complexidade da criação de alguns componentes e a refatoração da interface existe.

5.1.5 Desenvolvimento dos serviços

Nesta tarefa, desenvolvi alguns serviços *RESTful* para aumentar as funcionalidades da interface e resolver problemas existentes nos mesmos. Em conjunto com os colaboradores da empresa, consegui implementar algumas funcionalidades que não previa implementar, como por exemplo, a adição de dados recebidos via UDP na base de dados.

O plano, em concreto, não continha qualquer tarefa no espectro que me competia, pelo que tive alguma liberdade, concedida pelo orientador, para verificar alguns serviços que não funcionavam, analisar o porquê de não funcionarem e corrigi-los, como por

exemplo, o serviço de email. A forma como abordei a criação de funcionalidades também foi essencial para manter a segurança e o bom funcionamento do servidor.

Nesta fase, uma vez que realizei algumas tarefas que não estavam completamente planeadas, parte do tempo das tarefas iniciais foi realocado a esta.

5.1.6 Testes funcionais

Esta foi a única tarefa que não se realizou durante o estágio. Apesar de os testes serem uma parte importante para verificar as funcionalidades de um sistema, durante este estágio, não houve oportunidade nem o momento certo para os realizar. Apesar de não os ter realizado, outras tarefas foram levadas a cabo durante as horas previstas.

5.1.7 Relatório

Esta foi uma pequena tarefa realizada durante o último dia de estágio. Nesta tarefa, foram criadas as bases onde este relatório foi elaborado.

5.2 Considerações finais

Esta análise revela um percurso de estágio que, embora preso ao plano inicial, demonstrou uma capacidade de adaptação às necessidades reais e aos imprevistos ocorridos no projeto. Verifica-se que, para além das tarefas planeadas, como a análise de *APIs* de georreferenciação e o desenvolvimento da interface, parte do tempo foi dedicada à resolução de problemas e à implementação de melhorias. A decisão de realocar algumas das horas destinadas a outras tarefas demonstrou a necessidade de concretizar funcionalidades para o sistema.

Deste modo, o trabalho realizado ao longo das 466 horas de estágio resultou num contributo mais robusto e alinhado com os desafios práticos encontrados.

CAPÍTULO 6

Conclusões

O estágio curricular ao serviço da Wiseware foi uma experiência única e gratificante que permitiu aprofundar conhecimentos teóricos, técnicos e adquirir competências profissionais.

Ao concluir este estágio, sinto-me capaz de realizar diversas funções que permitem-me ter sucesso dentro de um ambiente empresarial, especialmente em funções dedicadas à criação de interfaces, no desenvolvimento de APIs, encontrar novas tecnologias e trabalhar em equipa.

Durante a realização deste estágio, consegui executar muitas das tarefas que estavam propostas no plano, assim como realizei outras que não constavam no mesmo.

Por vezes, consegui realizar as tarefas num tempo inferior ao proposto. No entanto, houve também tarefas desafiadoras onde dediquei algum tempo à leitura da documentação, na compreensão das suas nuances e na definição da melhor estratégia para a sua implementação.

Algo que se revelou desafiador, tanto a nível profissional como pessoal, foi a alteração de mentalidade que tive de realizar devido às diferenças entre o trabalho académico e o trabalho profissional, onde é necessária maior agilidade e usabilidade na produção, pois não somos os únicos a desenvolver, avaliar e utilizar o código.

O desenvolvimento da interface foi a área de maior aprendizagem, onde o contacto direto com os utilizadores e a incorporação do seu *feedback* se revelaram cruciais para compreender as suas necessidades e entregar um produto com maior valor.

keywords Technology, software, team, tasks, knowledge.

abstract The present report outlines all the activities carried out at the company Wiseware Solutions during the 466 hours of the internship. During the internship, specific objetives where defined: to adquire new skills related to the functioning of the organization, integration with the technologies used, software development and elaboration of internship report. These objectives made it possible to apply the knowledge acquired throughout the course.

The first part of this report presents a concise description of the company including a brief description of the activities carried out there. Next, the activities planned for the internship are described. In the third part, each activity is described in detail, including the difficulties encountered during their execution and the skills acquired through their completion.In the end, an analysis is carried out of the experiences gained during the internship and of future expectations, based on the knowledge acquired.

Juntamente com o desenvolvimento da interface, a criação de serviços *RESTful* permitiu a aplicação de vários conhecimentos lecionados no curso, bem como de outros adquiridos neste estágio, onde tive de os aprender e resolver os problemas que foram surgindo pelo caminho.

Face aos vários desafios encontrados ao longo da realização do estágio, existiram momentos em que foi necessário solicitar ajuda ao orientador e também muitas vezes realizar pesquisas de como resolver determinados problemas e como corrigi-los corretamente.

Por fim é possível dizer que este estágio foi bastante positivo, tanto profissionalmente como pessoalmente, pois deu-me a vontade de aprender sobre novos temas e iniciar novos projetos. Juntamente com os novos conhecimentos, que virão a ser importantes em projetos futuros, aprendi também como é trabalhar em um ambiente empresarial e que pouco tempo nesse ambiente é muito importante para a formação profissional.

Referências

[1] Google, *Maps JavaScript API*, "https://developers.google.com/maps/documentation/javascript", 2025.

[2] Chart.js, *Chart.js*, "https://github.com/chartjs/Chart.js", 2025.

[3] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub e S. Hagen, *The GeoJSON Format*, RFC 7946 (Proposed Standard), Internet Engineering Task Force, ago. de 2016. URL: https://www.ietf.org/rfc/rfc7946.txt.

[4] Mapbox, *Draw a polygon and calculate its area*, "https://docs.mapbox.com/mapbox-gl-js/example/mapbox-gl-draw/", 2025.

[5] S. Routing, *SvelteKit - Core concepts, Routing*, "https://svelte.dev/docs/kit/routing", 2025.

[6] joshnuss, *svelte-persisted-store*, "https://github.com/joshnuss/svelte-persisted-store", 2024.

APÊNDICE A

Apendices

universidade de aveiro estga

ANEXO

PLANO DE TRABALHO DO ESTÁGIO

IDENTIFICAÇÃO DO ESTÁGIO	
Estudante	Gonçalo José de Almeida Dias
Número Mecanográfico	102344
Unidade Curricular	99142-ESTÁGIO / PROJETO EM TECNOLOGIAS DA INFORMAÇÃO
Curso	Licenciatura em Tecnologias de Informação
Data de início	segunda-feira, 10 de fevereiro de 2025
Data de conclusão	sexta-feira, 13 de junho de 2025
Horário a cumprir	8/horas dia
Local (ou Locais) do Estágio	Zona Industrial da Mota, Rua 12, Lote 51, Fração E, 3830-527 Gafanha da Encarnação
Orientador da UA-ESTGA	Ivan Miguel Serrano Pires
Orientador da EA	Gustavo Corrente

Resumo do Plano de Trabalho

OBJETIVOS	**ATIVIDADES A DESENVOLVER**	**CALENDARIZAÇÃO PREVISTA (HORAS OU DIAS)**
Acolhimento e introdução ao desenvolvimento	Acolhimento Introdução às tecnologias e ferramentas: GIT, GITLAB, JENKINS, Ambiente JAVA e SVELTE	**40h**
Levantamento de requisitos	Levantamento de requisitos no projeto aniposture	**40h**
Escolher a API de Mapas	Estudo de funcionalidades entre GoogleMaps e OpenStreatMaps	**120h**
Desenvolvimento da UI	Desenvolvimento de UI de georreferenciação	**120h**
Desenvolvimento dos serviços	Desenvolvimento de serviços RESTful	**120h**
Testes funcionais	Realização de Testes Funcionais	**24h**
Relatório	Finalização da Escrita do Relatório	**2h**

Figura A.1: Plano trabalho

Figura A.2: Dispositivo de monitorização, coleira

Figura A.3: Dispositivo de monitorização, coleira

Figura A.4: Dispositivo de monitorização, coleira

Figura A.5: Dispositivo de monitorização, coleira

acknowledgement of use of AI tools

Recognition of the use of generative Artificial Intelligence technologies and tools, software and other support tools.

I acknowledge the use of Gemini 2.5 Flash (Google, https://gemini.google.com/) for text revision and assisting with streamlining information searches.

I acknowledge the use of GPT-4-turbo (Open AI, https://chat.openai.com) for text revision and assisting with streamlining information searches.

Conteúdo

Lista de Figuras