



**Gonçalo José  
Almeida Dias**

**Relatório de estágio**



**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. Ivan Miguel Serrano Pires, da Escola Superior de Tecnologia e Gestão de Águeda, Universidade de Aveiro.

**agradecimentos /  
acknowledgements**

Agradeço a todos os colaboradores da Wiseware Solutions pela colaboração e hospitalidade. Agradeço ao Prof. Doutor Ivan Pires pela orientação. Um agradecimento especial ao orientador Gustavo Corrente pela disponibilidade, apoio na resolução de problemas e abertura a novas ideias.

**palavras-chave**

Tecnologias, software, equipa, tarefas, conhecimentos.

**resumo**

O presente relatório comprehende 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 análise das experiências vivenciadas no estágio e as expectativas para o futuro com base nos conhecimentos obtidos.

**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 objectives were defined: to acquire 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.

**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 [insert AI system(s) and link] to [specific use of generative artificial intelligence or other tasks]. I acknowledge the use of [software, codes or platforms] to [specific use software, codes or platforms or to other tasks].

Example 1: I acknowledge the use of ChatGPT 3.5 (Open AI, <https://chat.openai.com>) to summarise the initial notes and to proofread the final draft and the use of Office365 (Microsoft, <https://www.office.com>) for text writing and productivity.

Example 2: No content generated by AI technologies has been used in this Thesis.

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

# Conteúdo

<b>Conteúdo</b>	i
<b>Lista de Figuras</b>	iii
<b>Glossário</b>	v
<b>1 Introdução</b>	1
<b>2 Entidade de acolhimento</b>	2
<b>3 Plano de trabalho</b>	3
<b>4 Trabalho realizado</b>	4
4.1 Descrição do projeto . . . . .	4
4.2 Análise de bibliotecas . . . . .	5
4.2.1 APIs georreferenciação . . . . .	5
4.2.2 APIs de visualização gráfica . . . . .	8
4.3 Interface gráfica . . . . .	10
4.3.1 Svelte 5 . . . . .	10
4.3.2 Tecnologias fundamentais . . . . .	12
4.3.3 Componente Mapa . . . . .	13
4.3.4 Página <i>reset palavra-passe</i> . . . . .	16
4.3.5 Página de dashboard . . . . .	17
4.3.6 Página de status . . . . .	18
4.3.7 Página de utilizadores . . . . .	19
4.3.8 Página de gráficos . . . . .	20
4.3.9 Página de erro . . . . .	20
4.3.10 Rotas sem inicio de sessão . . . . .	21
4.3.11 Guardar filtros . . . . .	21
4.3.12 Alterações no <i>design system</i> . . . . .	22
4.4 Servidor da aplicação . . . . .	23

4.4.1	O que é o <i>backend</i> ? . . . . .	23
4.4.2	Rota para obter número de pedidos por dispositivo . . . . .	24
4.4.3	Sistema de envio de emails . . . . .	26
4.4.4	Dados do <i>acc</i> e de temperatura no sample . . . . .	27
4.4.5	Rotas para aceder a dados sem <i>login</i> . . . . .	29
4.4.6	Correção de um <i>bug</i> que duplicava dados . . . . .	30
<b>5</b>	<b>Analise Resultados</b>	<b>31</b>
<b>6</b>	<b>Conclusões</b>	<b>32</b>
	<b>Referências</b>	<b>33</b>
<b>A</b>	<b>Apendices</b>	<b>33</b>

# Listas de Figuras

2.1	WiseWare Solutions . . . . .	2
4.1	Open Layers . . . . .	6
4.2	Google maps . . . . .	6
4.3	Mapbox . . . . .	7
4.4	Gráfico d3js . . . . .	8
4.5	Gráfico chartJs . . . . .	9
4.6	Gráfico Apache Echarts . . . . .	9
4.7	Diferenças <i>Svelte</i> 4 e 5 . . . . .	10
4.8	Exemplo <i>props svelte</i> . . . . .	11
4.9	<i>\$derived &amp; \$effect</i> . . . . .	11
4.10	Exemplo de interface criada com <i>tailwind</i> . . . . .	12
4.11	Interface com componentes <i>flowbite</i> . . . . .	12
4.12	Camadas . . . . .	13
4.13	Atualizar pontos selecionados no mapa . . . . .	13
4.14	Exemplo de marcadores . . . . .	14
4.15	<i>Map utils</i> . . . . .	14
4.16	Desenhar polígonos . . . . .	15
4.17	<i>Flow</i> de edição de polígonos . . . . .	15
4.18	Páginas <i>reset-pass</i> . . . . .	16
4.19	Redirecionar para o <i>login</i> . . . . .	17
4.20	Página de <i>dashboard</i> e pacotes detalhes . . . . .	17
4.21	Componente <i>Device stats</i> . . . . .	18
4.22	Página <i>device status</i> . . . . .	18
4.23	Página <i>user</i> para <i>owners</i> . . . . .	19
4.24	Página <i>user</i> para <i>tenant admins</i> . . . . .	19
4.25	Erro . . . . .	20
4.26	Páginas <i>app-gaime</i> . . . . .	21
4.27	<i>Svelte persisted store</i> . . . . .	21

4.28	Cores geradas pela tailwind . . . . .	22
4.29	Cores geradas por sombra . . . . .	22
4.30	Antiga <i>query</i> para obter o estado dos dispositivos . . . . .	24
4.31	Nova <i>query</i> para obter o estado dos dispositivos . . . . .	25
4.32	Emails enviados pelo servidor . . . . .	26
4.33	Flow envio de dados dos dispositivos para o servidor . . . . .	27
4.34	Erro de integridade ao guardar dados . . . . .	27
4.35	Parte do código usado para registar uma nova <i>sample</i> . . . . .	28
4.36	Código para criação das rotas <i>/noAuth</i> . . . . .	29
4.37	Flow de envio quando o dispositivo encontra-se <i>offline</i> . . . . .	30
A.1	Plano trabalho . . . . .	34

# 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  
**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 análise 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 inovadoras e de alta qualidade para empresas.

A Wiseware é responsável por produzir soluções em áreas como robótica, micro eletrônica, inteligência artificial, comunicações sem fio e muito mais. Aplicaram os 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 técnico, cortes 2D/3D a laser, 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 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.

# CAPÍTULO 4

## Trabalho realizado

Neste capítulo irei abordar todo o trabalho desenvolvido durante o período de estágio, organizado em três grandes secções. Na primeira secção abordarei o trabalho realizado durante a análise de APIs e bibliotecas, incluindo o porque da escolha final de cada uma. A segunda secção visa comentar o trabalho realizado na interface gráfica do projeto aniposture. A terceira secção comenta 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 animal. O dispositivo regista igualmente o estado da bateria e emite alertas quando o nível se encontra 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 satélite, 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.

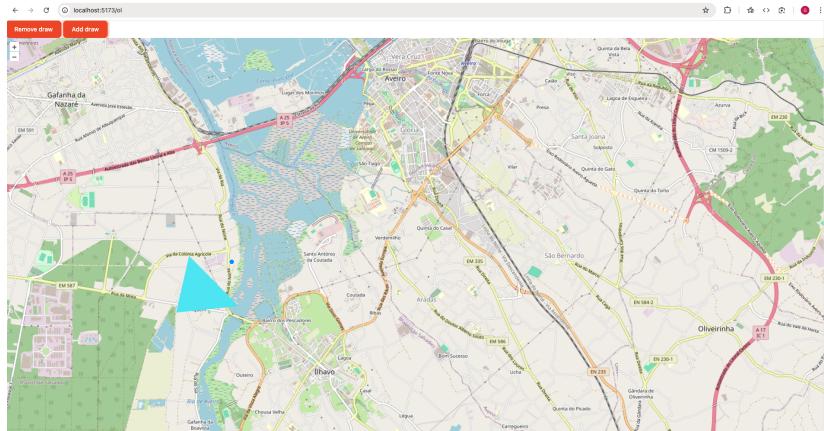
#### *OpenLayers*

*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 contém nenhum *tile* em satélite.

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.



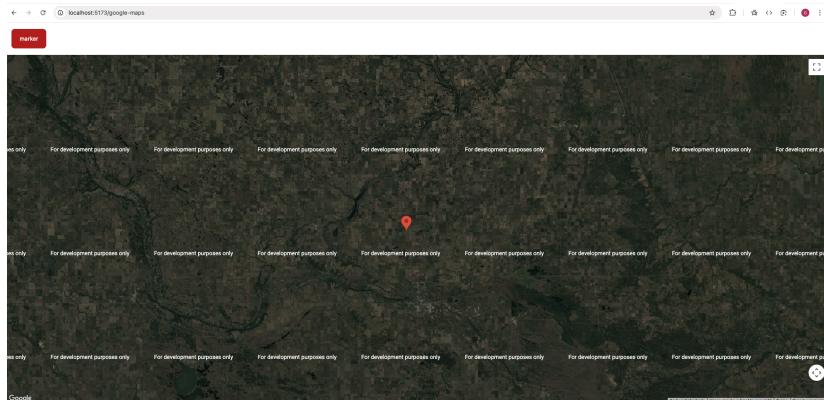
**Figura 4.1:** OpenLayers

### **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.



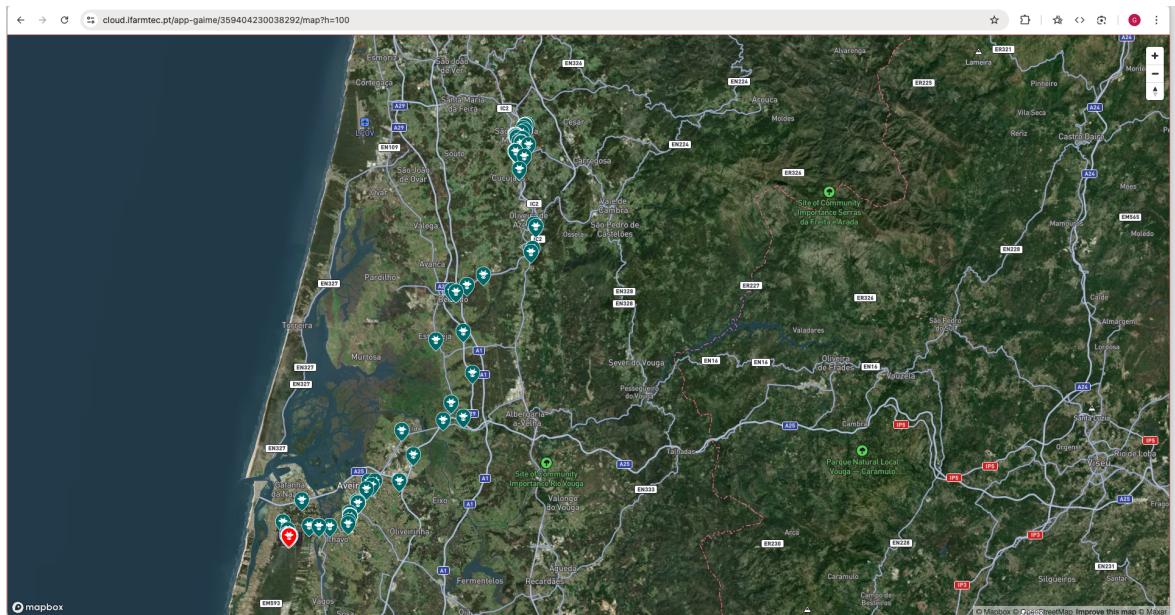
**Figura 4.2:** Google Maps

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, contém 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/económico, 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 4.3:** 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.

#### 4.2.2 APIs de visualização gráfica

Após análise 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 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.



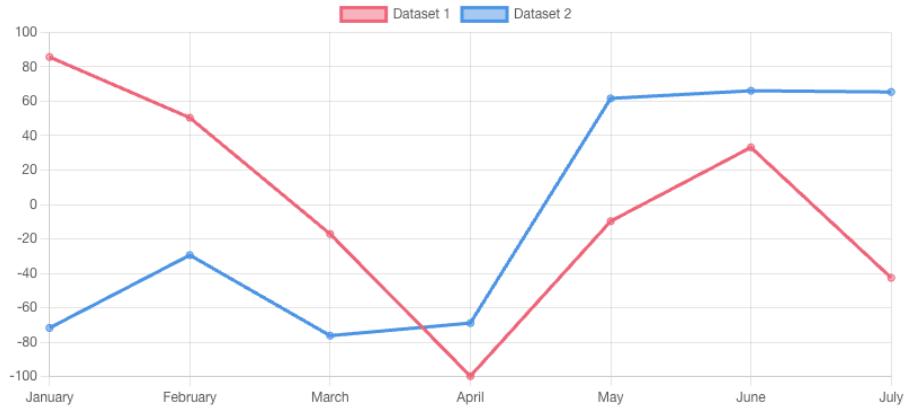
**Figura 4.4:** Exemplo de gráfico criado com o d3js

##### *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 informações.



**Figura 4.5:** Exemplo de gráfico criado com o chartJs

### Echarts

*Apache Echarts* é uma biblioteca gratuita, desenvolvida no âmbito da fundação *Apache*, que apresenta características 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 4.6:** 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 funcionalidades nativas e pela elevada performance a carregar 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 4.7b.

Nas versões anteriores, o *svelte* tornava todos os objetos reativos por padrão, isto dificultava a percepção 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 4.7a, 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 4.7:** 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 4.8.

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

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

**Figura 4.8:** 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 importantes, e os que exigiram mais tempo a compreender foram o *\$derived* e o *\$effect*.

O *\$derived* é uma runa utilizada para definir uma variável derivada. Ou seja, cria-se uma nova variável cujo o valor é automaticamente atualizado com base numa expressão, sempre que ocorre alguma alteração noutra variável, Figura 4.7b.

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 4.7b.

```
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 4.9:** *\$derived* & *\$effect*

Na Figura 4.9, podemos ver um exemplo um pouco mais complexo da utilização das runas *\$derived* e *\$effect*.

No caso do *\$derived*, a variável *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 variável *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 experiência 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.timestampToDateStr(data.requestDate)}
    </span>
  </div>
</div>
```

**Figura 4.10:** 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.

Filters				
<input checked="" type="checkbox"/> Points with GPS	<input checked="" type="checkbox"/> Points without GPS	<input checked="" type="checkbox"/> BootUp		
Points				
DATE	FIX TIME	NO FIX COUNT	TYPE	DETAILS
31/05/2025 15:19:38	0	196		
Info				
Fix time in ms	Device battery	Device status		
0	5			
Network data				
CELL ID	OPERATOR	RSRP	TAC	
313100	26806	-96	48710	

**Figura 4.11:** Interface com componentes *flowbite*

### 4.3.3 Componente Mapa

Um dos trabalhos mais complexos nesta fase da interface foi a implementação do mapa. Trata-se de um elemento com características especiais, que exige funcionalidades como o carregamento de pontos, executar ações ao detetar cliques, carregar *layouts*, atualização automática 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 (ref arquitetura do sistema) *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, símbolos, 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 (ref arquitetura do sistema). 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;
    selectedFeature = selectedFeatures[0];
  }
});
```

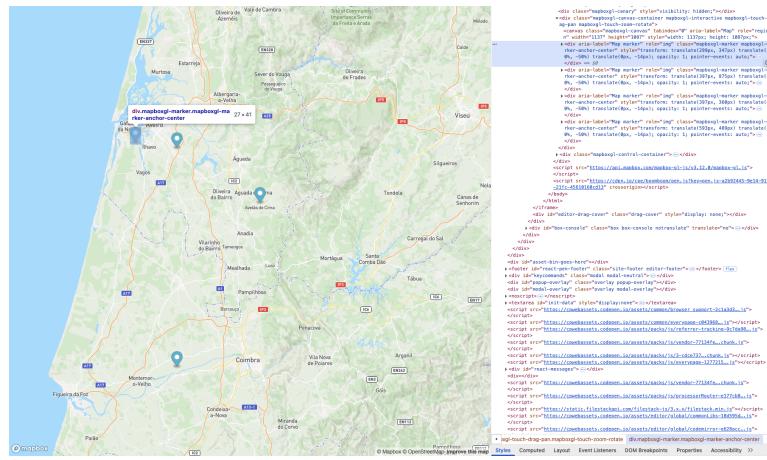
(b) *Map onClick*

**Figura 4.13:** Atualizar pontos selecionados no mapa

## Marcadores

Antes da utilização das camadas vetoriais, descritas na 4.3.3, foram realizados alguns testes com os marcadores nativos do mapbox.

Um marcador é uma representação visual de uma coordenada específica no mapa. Estes marcadores são elementos *html* inseridos no *DOM* da página, Figura 4.14. Devido ao baixo desempenho, esta abordagem revela-se inviável 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 4.14:** 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 4.15.

```
export interface GpsMapProps {
  style: MapStyle;
  center: LngLatLike;
  zoom: number;
  popups?: GeoSample[];
  markerOnClick?: () => void;
  drawFeatures?: 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_v12 = 'satellite-satellite-v12',
  satellite_streets_v12 = 'satellite-streets-v12',
  navigation_day_v1 = 'navigation-day-v1',
  navigation_night_v1 = 'navigation-night-v1'
}
```

(a) *Map Interfaces*

```
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 4.15:** *Map utils*

## Desenho de polígonos

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

Esta funcionalidade foi implementada em cima de um *plugin* criado pela comunidade, *mapbox-gl-draw*[3], 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 polígonos 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 4.16.



Figura 4.16: Desenhar polígonos

O sistema funciona conforme representado na Figura 4.17. Esta abordagem revelou-se a mais prática e eficiente para desenvolver a funcionalidade de desenho e edição de polígonos. Além disso, ao estruturar o sistema desta forma, torna-se possível, no futuro, carregar polígonos a partir de uma base de dados, editá-los diretamente no mapa e, posteriormente, guardar as alterações realizadas.

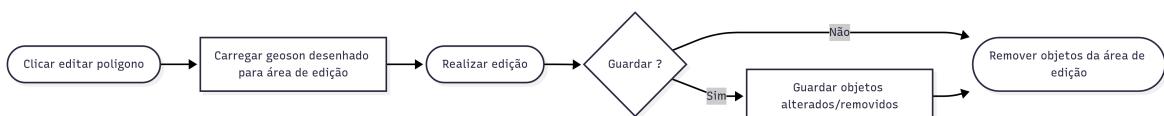
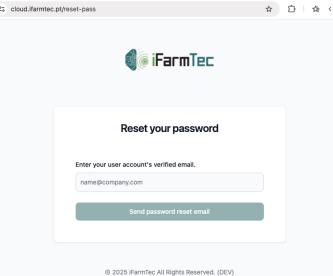


Figura 4.17: Flow de edição de polígonos

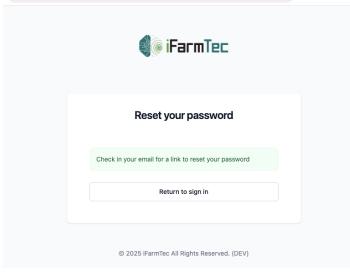
#### 4.3.4 Página *reset palavra-passe*

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.

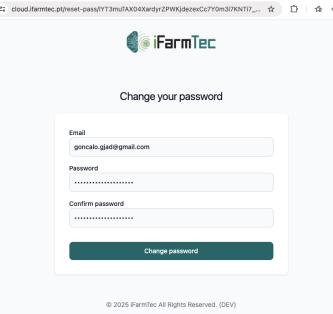
Todo este sistema foi implementado na mesma rota, */reset-pass*. Ao aceder a esta página, o utilizador é redirecionados para um ecrã onde lhe é solicitado a inserir o endereço de *email* associado à sua conta. Após inserir o *email* e clicar no botão "*Send password reset email*", é enviado um *email*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



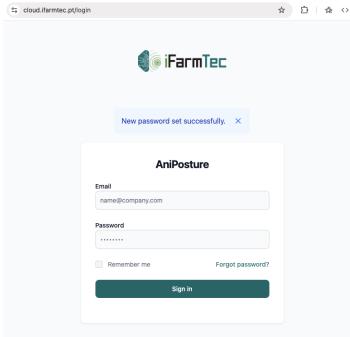
(a) *Reset-pass* enviar email



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



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



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

**Figura 4.18:** 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]*[4]. 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 4.18d.

Para tornar todo este sistema funcional e, simultaneamente, garantir que apenas utili-

zadores 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));

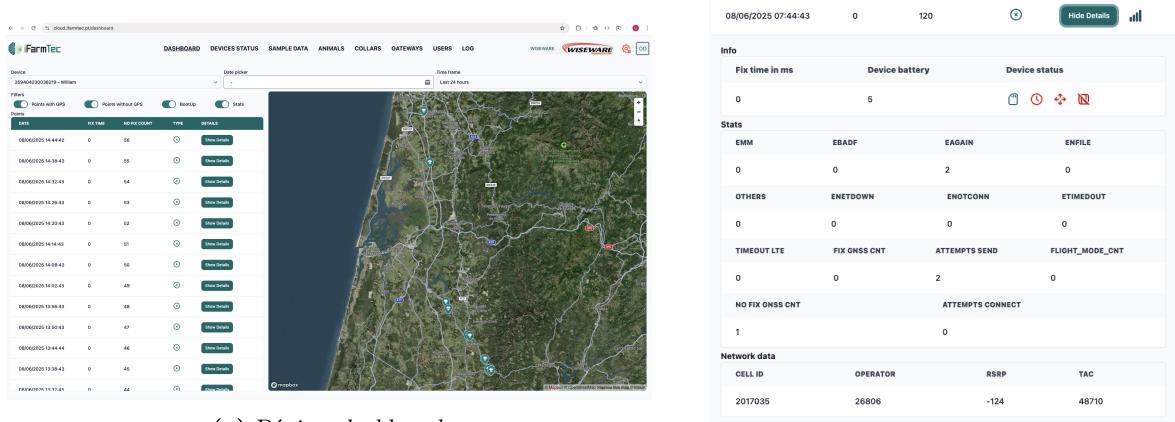
if (sessionService.isAuthenticated() && url_path == '/login') {
  goto('/');
} else if (!sessionService.isAuthenticated() && filter.length === 0) {
  goto('/login');
}
```

**Figura 4.19:** 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 (/).

#### 4.3.5 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.



**Figura 4.20:** Página de *dashboard* e pacotes detalhes

Como se pode observar na Figura 4.20a, 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 4.20b, 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.

#### 4.3.6 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 4.21.

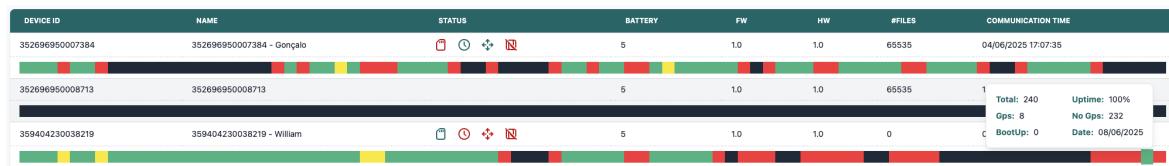


Figura 4.21: Componente *Device stats*

##### *Página device status*

Esta página contém, 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 4.22.

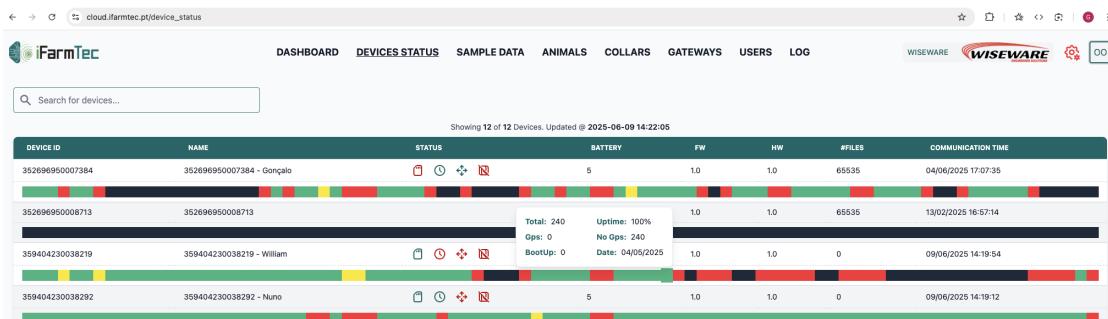


Figura 4.22: Página *device status*

#### 4.3.7 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.

USER	EMAIL	TENANT	ACTIONS
Gustavo Corrente	gcorrente@gmail.com	GAME	
Gonçalo Dias	goncalodias@ua.pt	WISEWARE	
Gustavo Corrente2	gcorrente@gmail.com2		
ANIPOSTURE GATEWAYS	gateways@wisewaresolutions.com		
ANIPOSTURE GATEWAYS	aniposture-gateways@wisewaresolutions.com		
Gustavo Corrente3	gcorrente@gmail.com3		
WISEWARE NELSONZ	nelson.filipe2@wisewaresolutions.com		
OWNER	gustavo.corrente@wisewaresolutions.com		
Gonçalo Dias	goncalo.giad@gmail.com	GAME	
Old Owner	admin@mail.com	All Tenants	

Figura 4.23: Página *user* para *owners*

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.

ENABLED	USER	EMAIL	ACTIONS
	ANIPOSTURE GATEWAYS	aniposture-gateways@wisewaresolutions.com	
	Gonçalo Dias	goncalo.giad@gmail.com	
	Gustavo Corrente	gcorrente@gmail.com	
	Gustavo Corrente2	gcorrente@gmail.com2	
	Gustavo Corrente3	gcorrente@gmail.com3	

Figura 4.24: Página *user* para *tenant admins*

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*.

#### 4.3.8 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 experiência, 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.

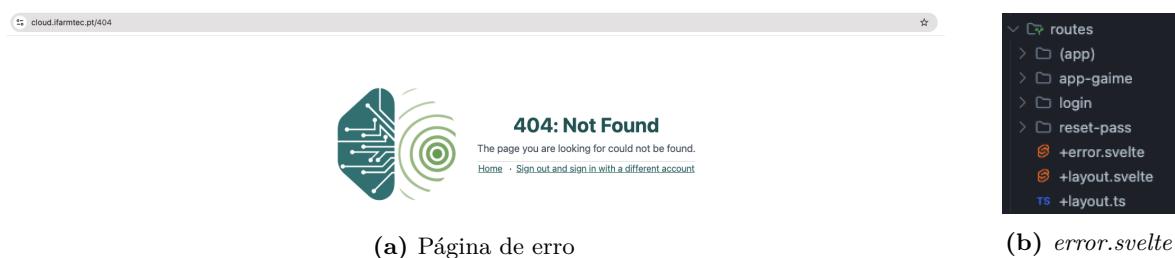
Os dados provenientes do acelerômetro são apresentados de forma normalizada, ao utilizar a seguinte fórmula:  $\sqrt{x^2 + y^2 + z^2}$ .

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.

#### 4.3.9 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 4.25a, foi utilizada uma funcionalidade nativa do *sveltekit*. Ao criar uma página de erro na raiz da aplicação, como demonstrado na Figura 4.25b, todos os erros que ocorram durante a navegação são automaticamente redirecionados para essa página.

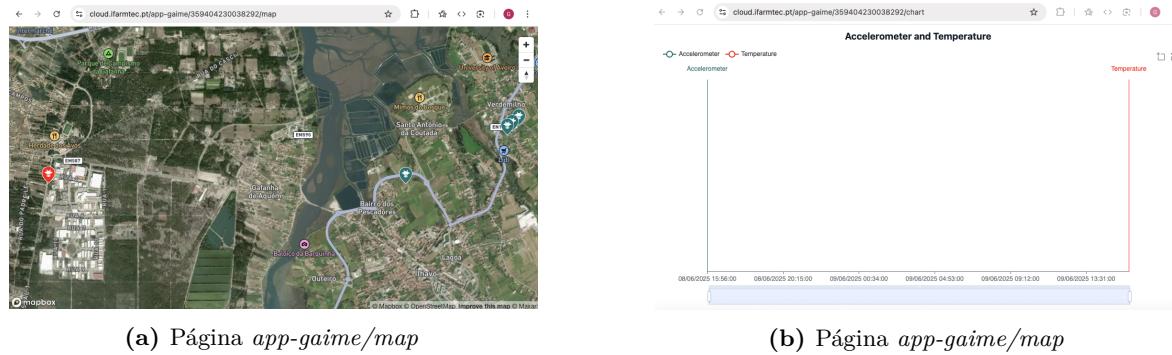
Desta forma, foi possível desenvolver uma página de erro genérica, aplicável a qualquer secção da aplicação, cuja apresentação adapta-se dinamicamente consoante o tipo de erro ocorrido.



**Figura 4.25:** Erro

#### 4.3.10 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 4.26:** Páginas `app-gaime`

Por padrão, a rota do mapa , Figura 4.26a, solicita ao servidor todos os pontos com coordenadas GPS registados nas últimas 24 horas. De forma semelhante, a rota do gráfico, Figura 4.26b, recupera informações relativas ao acelerômetro 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]`.

#### 4.3.11 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* [5].

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.

```
// Dashboard
export const searchDeviceDashboard = persisted('searchTermDeviceDashboard', '');
export const searchTimeFrameDashboard = persisted('searchTimeFrameDashboard', '');
export const searchDateFromDash = persisted('searchDateFromDashboard', null);
export const searchDateToDash = persisted('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 staticPointsStore = persisted<boolean>('staticPointsStore', true);

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

(a) Criação das *stores*

```
<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>
```

(b) Utilização das *stores* num ficheiro `.svelte`

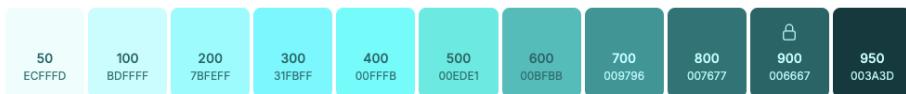
**Figura 4.27:** *Svelte persisted store*

#### 4.3.12 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.

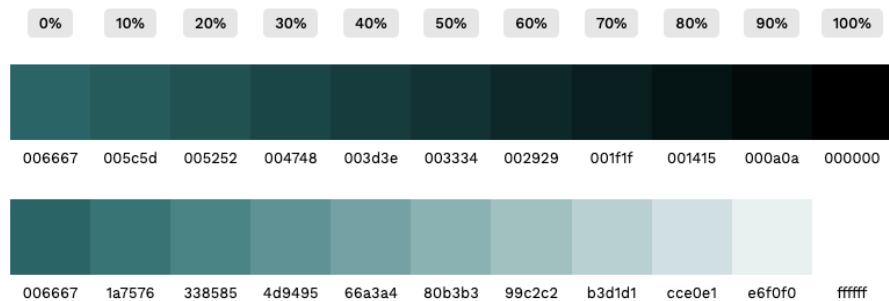
Para tornar essa reformulação possível, alterei o tema principal. O tema anterior, gerado automaticamente pelo *tailwind* a partir de uma única cor, originou uma paleta cromática que se afastava da identidade visual pretendida para a aplicação.

A Figura 4.28 mostra que as tonalidades mais claras da aplicação apresentavam um azul claro que não se combinava 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.



**Figura 4.28:** Cores geradas pela tailwind

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.



**Figura 4.29:** Cores geradas por sombra

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, 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.

## 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* assíncrona 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 esta alteração, 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', timestampptz '$FROM')
      AND date_trunc('day', timestampptz '$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 4.30:** 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.

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 4.31:** 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 4.3.6. Enquanto a rota anterior mostrava apenas informações sobre os pontos, esta exibe dados detalhados sobre os dias e os pontos registados em cada um.

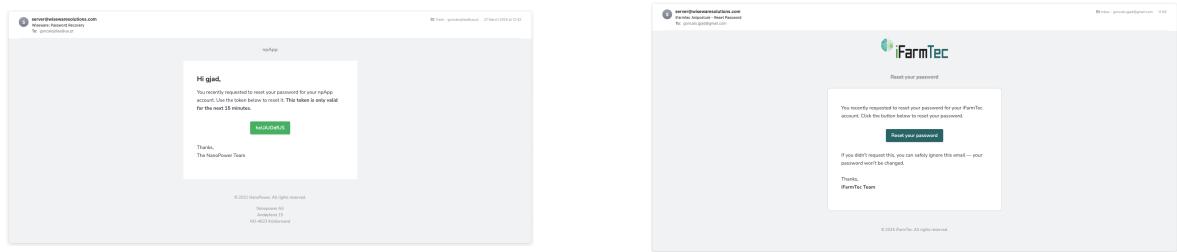
Esta reestruturação não só otimizou a rota em termos de dados, mas também conferiu um valor analítico superior. A capacidade de identificar dias sem comunicação, permite diagnosticar falhas de conexão ou potenciais anomalias nos dispositivos, algo que a versão anterior não proporcionava.

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, e foi aqui que as alterações se focaram, principalmente no *template* do email. O *template* existente não se alinhava com o *design system* da aplicação, nem com o seu propósito.



(a) Antigo template de email

(b) Novo template de email

**Figura 4.32:** Emails enviados pelo servidor

A rota *change-pass*, por sua vez, permite a alteração da palavra-passe. Aqui, foi identificado um problema considerado 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.

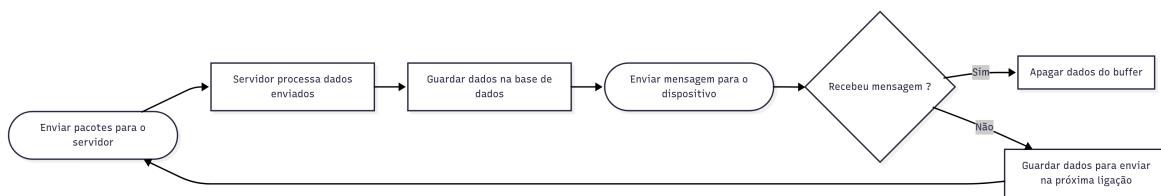
As alterações asseguraram não só a correção de falhas técnicas, mas também a melhoria da experiência do utilizador e a robustez da segurança da aplicação. Adequar os templates de email ao *design system* da aplicação contribuiu para uma comunicação mais clara para com os utilizadores. Simultaneamente, 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 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 4.33:** 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 4.34:** 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 únicos. 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 constraints.
@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 constraints.
    @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 4.35:** Parte do código 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 análise 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 4.35, é essencial para assegurar a continuidade da informação dos pontos GPS, não GPS e de *boot*.

#### 4.4.5 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 4.3.10.

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) {
    GpsSample q = query();
    // Necessary to make query on the correct tenant
    q.setBaseTable(WiseSchema.getSchema() + ".gps_sample");
    if (deviceId != null)
        q.deviceId.eq(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*

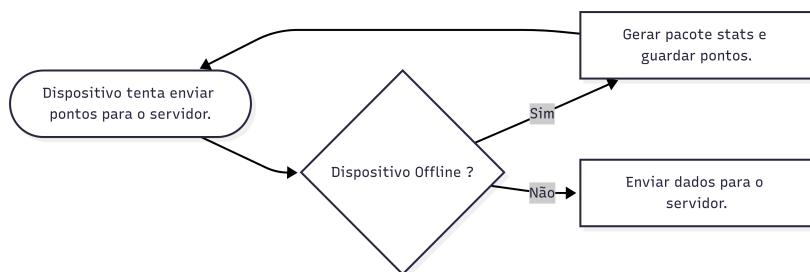
**Figura 4.36:** Código para criação das rotas */noAuth*

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

#### 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.



**Figura 4.37:** Flow de envio quando o dispositivo encontra-se *offline*

Ao ligar-se finalmente à rede, cada dispositivo enviava um pacote de localização, com dados GPS ou sem GPS, caso não conseguisse os primeiros. 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 outros.

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 período 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 terá informações de rede pois são dados registados enquanto o dispositivo encontrava-se *offline*. É ainda importante mencionar que o dispositivo envia todos os dados num único pacote *UDP*, composto por um determinado número de *bytes*. Cada segmento deste bloco corresponde a um tipo específico pacote, o que assegura a sua distinção e o processamento correto dos dados.

# CAPÍTULO 5

## Analise Resultados

# CAPÍTULO 6

## Conclusões

# 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] Mapbox, *Draw a polygon and calculate its area*, "<https://docs.mapbox.com/mapbox-gl-js/example/mapbox-gl-draw/>", 2025.
- [4] S. Routing, *SvelteKit - Core concepts, Routing*, "<https://svelte.dev/docs/kit/routing>", 2025.
- [5] joshnuss, *svelte-persisted-store*, "<https://github.com/joshnuss/svelte-persisted-store>", 2024.

# APÊNDICE A

## Apendices

**ANEXO**  
**PLANO DE TRABALHO DO ESTÁGIO**

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