Breno Tartaroni Soares Garcia

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA VISUALIZAÇÃO DE PARAMETROS AMBIENTAIS

UNIVERSIDADE DO ALGARVE
Instituto Superior de Engenharia
2023

Breno Tartaroni Soares Garcia

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA VISUALIZAÇÃO DE PARAMETROS AMBIENTAIS

Mestrado em Engenharia Eletrotécnica e de Computadores

(Especialidade em Tecnologias de Informação e Telecomunicações)

Trabalho realizado sob a orientação de:

Professor Doutor João Miguel Fernandes Rodrigues

UNIVERSIDADE DO ALGARVE
Instituto Superior de Engenharia
2023

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA VISUALIZAÇÃO DE PARAMETROS AMBIENTAIS

Declaração de autoria da obra

Declaro ser o autor desta obra, que é original e inédita. Os autores e obras consultados são devidamente citados no texto e constam da listagem de referências incluídas.

_____

(Breno Tartaroni Soares Garcia)

Aos meus familiares
e
amigos

AGRADECIMENTOS

À minha família que mesmo longe me apoiou a continuar nessa jornada e me manter firme.

Ao meu orientador João Rodrigues que me guiou durante todo o processo de elaboração. Suas orientações foram essenciais para que eu pudesse desenvolver um trabalho consistente e de qualidade.

Aos professores que me acompanharam ao longo do curso, fornecendo conhecimentos teóricos e práticos que foram fundamentais para o meu crescimento acadêmico e pessoal.

Aos meus amigos, que me apoiaram em todos os momentos e me encorajaram a seguir em frente mesmo diante dos desafios.

RESUMO

Entende-se por aplicações web ou *web apps* uma solução que é executada diretamente no navegador (*browser*), não sendo preciso fazer uma instalação na máquina do utilizador. Este relatório de estágio descreve o desenvolvimento de uma *web app* para a visualização de parâmetros ambientais usando tecnologias e técnicas modernas de desenvolvimento web, como implementação do CI/CD (*Continuous Integration*/*Continuous Delivery*). O relatório descreve o que é CI/CD e suas melhores práticas, bem como algumas medidas de segurança que devem ser levadas em consideração. Analisa e descreve as ferramentas usadas, nomeadamente a *framework* Scrum, Angular, Spring Boot, MySQL, Keycloak e ActiveMQ. Apresenta um estado da arte sobre aplicações semelhantes no mercado e descreve em detalhe o desenvolvimento da *web app*. Quanto a aplicação, é capaz de fornecer informações precisas sobre o tempo atual e previsões futuras para várias cidades em toda a Espanha. A interface do utilizador é projetada de forma intuitiva e interativa, embora o *design* seja muito parecido com outros existentes, permitindo que os utilizadores visualizem facilmente as informações meteorológicas relevantes. A aplicação também fornece recursos adicionais, como autenticação (*login/logout*) e lista de favoritos. Os testes e resultados efetuados pelo cliente ("orientador industrial") mostram que a aplicação é eficaz e fácil de usar fornecendo informações úteis e precisas para os utilizadores.

Palavras Chave: *Web App, Continuous Integration, Continuous Delivery*, Ambiente.

ABSTRACT

A solution that runs directly in the browser (browser) and doesn't need to be installed on the user's computer is known as a web application or web app. The creation of a web application for the display of environmental parameters utilizing cutting-edge web development tools and methods, such as the use of CI/CD (Continuous Integration/Continuous Delivery), is detailed in this internship report. The study explains CI/CD, its best practices, and some security precautions to take. In particular, the Scrum framework, Angular, Spring Boot, MySQL, Keycloak, and ActiveMQ are examined and described. It provides an overview of comparable apps currently available on the market and goes into depth on how the web app was created. Regarding the application, it can provide precise details on the present weather and upcoming forecasts for several cities around Spain. Despite being fairly similar to others already on the market, the user interface is interactive and has an intuitive design that makes it simple for users to access the necessary meteorological data. The program also offers extra features like favorites list and authentication (login/logout). The customer's ("industrial advisor") tests and findings reveal that the application is efficient and simple to use, giving users access to accurate and relevant information.

Keywords: Web App, Continuous Integration, Continuous Delivery, Environment.

ÍNDICE

LISTA DE FIGURAS

LISTA DE ACRÓNIMOS

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
BD	Base de dados
CAD	*Computer-Aided Design*
CI/CD	*Continuous Integration/Continuous Delivery*
CPU	*Central Process Unit*
CSS	*Cascading Style Sheet*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
JMS	*Java Message Service*
JVM	*Java Virtual Machine*
JWT	JSON Web *Tokens*
MQTT	*Message Queuing Telemetry Transport*
NoSQL	*Not Only* SQL
PO	*Product Owner*
RDBMS	*Relational Database Management System*
REST	*Representational State Transfer*
SMS	*Short Message Service*
SPA	*Single Page Application*
SSL	*Secure Sockets Layer*
STOMP	*Streaming Text Oriented Messaging Protocol*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
TLS	*Transport Layer Security*
UV	Ultravioleta
VPN	*Virtual Private Network*

1 INTRODUÇÃO

1.1 ENQUADRAMENTO DO ESTÁGIO

Entende-se por aplicações web ou *web apps* uma solução que é executada diretamente no navegador (*browser*), não sendo preciso fazer uma instalação na máquina do utilizador. Pode-se, também, utilizar como definição "tudo aquilo que é processado num servidor terceiro". As plataformas de *e-commerce* e as redes sociais são alguns dos exemplos que se enquadram nesse perfil (Noleto, 2020; StrongNet, n.d.).

Muitas vezes os utilizadores confundem a *web app* com *website*, que é geralmente usado para fornecer informações estáticas ou conteúdo informativo, como páginas de negócios, blogs, notícias, portfólios, etc., sendo principalmente voltado para a exibição de informações e interação limitada com o utilizador. Um *website* embora possa conter algum nível de interatividade, como formulários de contato ou comentários, seu objetivo principal é fornecer conteúdo e informações. Sintetizando, uma *web app* quando executada num servidor remoto pode ser acedida a partir de qualquer lugar e/ou máquina, desde que haja uma conexão com a internet, é também uma solução que depende da interação do utilizador, com funcionalidades avançadas e interação "complexas" com o utilizador (STH, 2023).

As *web apps* podem ser executados em qualquer navegador (moderno), estes geralmente são projetados para serem acessíveis a partir de vários dispositivos, incluindo desktops, laptops, smartphones e tablets. As principais linguagens para desenvolvimento são CSS (*Cascading Style Sheet*), JavaScript e HTML (*Hyper Text Markup Language*), podendo também serem desenvolvidos utilizando outras tecnologias, como C#, Java, Go, Ruby, entre outras (Codebit, 2022). Essas tecnologias proporcionam maior poder e controle ao programador, permitindo o desenvolvimento de aplicações web mais sofisticadas.

Continuous Integration (CI) e *Continuous Delivery* (CD) são práticas de engenharia de software que visam a otimização do fluxo de trabalho de desenvolvimento de software, a fim de aumentar a eficiência, a qualidade e a rapidez do processo de entrega (Bobrovskis & Jurenoks, 2018; Singh *et al.*, 2022; Bello *et al.*, 2022).

Este estágio do Mestrado em Eng. Eletrotécnica e de Computadores será realizado na empresa Atos, que tem um núcleo sediado no UAlg TEC CAMPUS, no Campus da Penha da Universidade do Algarve (v.d. seção 1.2). Neste estágio serão identificadas, estudadas e aprofundadas algumas das tecnologias e *frameworks* usadas no seio da Atos para o desenvolvimento de software – *web apps*, nomeadamente: Angular, SpringBoot, MySQL, Keycloak e ActiveMQ, que serão comparadas com outras tecnologias existentes no mercado fazendo uma análise SWOT. Será também estudado e identificadas as melhores práticas de *Continuous Integration* e *Continuous Delivery*, bem como da metodologia Scrum (Morandini *et al.*, 2021; Dantas, 2021) usada no processo de desenvolvimento de aplicações. Será ainda feito um estado da arte sobre aplicações para visualização de parâmetros ambientais, como por exemplo as referidas em Beeharry *et al.* (2019) ou em Kox *et al.* (2020).

Como componente prática do estágio será desenvolvido uma *web app* para visualização de parâmetros ambientais, esta aplicação será descrita em detalhe (v.d. capítulo 4) e usará as melhores práticas de CI/CD, bem como será descrito o uso do *framework* Scrum para a gestão do projeto.

1.2 OBJETIVOS

Este estágio tem como objetivo principal o **desenvolvimento de uma *web app* para visualização de parâmetros ambientais**, tem os seguintes sub-objetivos:

a) Analisar e descrever:
    a. as melhores práticas de CI/CD aplicadas ao desenvolvimento de uma *web app*;
    b. ferramenta de gestão de produto, nomeadamente *framework* Scrum;
    c. ferramenta(s) de desenvolvimento de produto, nomeadamente Angular, Spring Boot;
    d. outras ferramentas, nomeadamente MySQL, Keycloak e ActiveMQ;
b) Fazer o estado da arte sobre aplicações existentes no mercado que permitam a visualização de parâmetros ambientais;
c) Desenvolvimento e descrição em detalhe do desenvolvimento da aplicação web;
d) Escrita do relatório.

Resumindo e reforçando, o principal objetivo deste estágio é aprofundar as mais recentes tecnologias para desenvolver *web apps*, i.e., aprender todo o processo de desenvolvimento de uma *web app* do zero. Para isso serão utilizadas ferramentas como Angular, SpringBoot, MySQL, Keycloak e ActiveMQ. A *web app* apresentada será realizada para um cliente fictício, seguindo à risca a metodologia Scrum[1]. Todas funcionalidades e formalidades serão as de um projeto real apesar de não ser entregue para uma empresa contratante o produto final. A *web app* conta com funcionalidades como: *login/logout*, pesquisa por localidades, opção de adicionar localidades a sua lista de favoritos para fácil manuseamento (*user experience* - UX) que serão explicadas em mais detalhes.

1.3 CONTEXTO DO TRABALHO

Atos é uma empresa francesa de serviços de TI (Tecnologia da Informação), está sediada em Bezons-França, é líder global em Transformação Digital, com 112.000 colaboradores e uma faturação anual de 11.000 milhões de euros. A Atos lidera na Europa várias áreas do conhecimento em TI, como a cibersegurança, *cloud* e *high performance computing* e fornece soluções sob medida para os mais diferentes tipos de indústrias em 71 países. A Atos é pioneira em produtos e serviços de descarbonização, está comprometida com a tecnologia digital segura e descarbonizada para os seus clientes, sendo uma SE (*Societas Europaea*) presente no Euronext Paris e incluída nos índices de ações CAC 40, ESG e Next 20 Paris [https://atos.net/pt-pt/portugal].

A Atos atualmente conta com dois escritórios em Portugal, o mais antigo em Lisboa e o mais recente localizado na Universidade do Algarve no novo edifício UAlg TEC CAMPUS (Campus da Penha da UAlg), onde está a ser desenvolvido esse estágio.

Os novos estagiários passam por um processo de formação, onde são apresentados os valores e costumes da empresa, como também, algumas das tecnologias utilizadas.

[1] Scrum é uma metodologia ágil de gerenciamento de projetos que é amplamente utilizada no desenvolvimento de software. Foi originalmente desenvolvida na década de 1990 e tem se tornado cada vez mais popular devido à sua abordagem flexível e iterativa.

Normalmente são submetidos a vídeos curtos focados nas tecnologias que serão utilizadas pelo estagiário, além de *daily meetings* para o acompanhamento do progresso.

Durante o processo de formação seguem a metodologia Scrum (Patrucco *et al.*, 2022) que é amplamente utilizada em projetos reais e por outras empresas de tecnologia. Separados em pequenas equipas, os novos estagiários têm de desenvolver uma *web app* para previsão do tempo com as tecnologias apresentadas durante a formação, que variam dependendo do pedido dos contratantes.

A pequena equipa de estagiários formam a equipa de desenvolvedores, e os *Team Leaders* fazem o papel de *Scrum Master* e *Product Owner* (Hidayati *et al.*, 2022). Apesar de seguir a metodologia Scrum o "contratante" do projeto pede apenas uma *web app* para "previsão do tempo" e determina algumas funcionalidades que são imprescindíveis ficando assim à critério da equipa de desenvolvedores o design da aplicação.

Neste relatório irei apresentar uma evolução da *web app* de "previsão do tempo" para "visualização de parâmetros ambientais".

1.4 ORGANIZAÇÃO DO DOCUMENTO

O presente capítulo apresentou um resumo a definição de *web app*, traçou os objetivos do estágio e fez um enquadramento deste, introduzindo a Atos e o enquadramento do estágio na empresa. O segundo capítulo descreve o que é CI/CD e suas melhores práticas, bem como algumas medidas de segurança que devem ser levadas em consideração. Analisa e descreve ferramenta usadas, nomeadamente a *framework* Scrum, Angular, Spring Boot, MySQL, Keycloak e ActiveMQ.

O terceiro capítulo apresenta um estado da arte sobre aplicações semelhantes no mercado e no quarto descreve em detalhe o desenvolvimento da *web app*. O último capítulo apresenta as conclusões.

2 PRÁTICAS DE DESENVOLVIMENTO DE SOFTWARE E FERRAMENTAS

2.1 INTRODUÇÃO

Este capítulo reflete sobre as práticas de desenvolvimento de software, realçando algumas das melhores práticas em uso. Introduz e faz análise SWOT de algumas das tecnologias e *frameworks* existentes que permitem o desenvolvimento de *web apps* para visualização de parâmetros ambientais. São também apresentadas as ferramentas para trabalhar com base de dados, sistemas de autenticação e mensagens (*message broker*). É efetuado um pequeno enquadramento ao Scrum (Patrucco *et al.*, 2022), que é uma estrutura ágil de gestão de projetos que é usada com frequência por equipas de desenvolvimento de software. O capítulo termina com um sumário de tecnologias/*frameworks* e metodologias a usar no desenvolvimento da *web app*.

2.2 *CONTINUOUS INTEGRATION* E *CONTINUOUS DELIVERY*

Continuous Integration e *Continuous Delivery* são práticas de engenharia de software que visam a otimização do fluxo de trabalho de desenvolvimento de software, a fim de aumentar a eficiência, a qualidade e a rapidez do processo de entrega (Bobrovskis & Jurenoks, 2018; Singh *et al.*, 2022; Bello *et al.*, 2022). Com o aumento da procura por soluções digitais, é cada vez mais importante que as equipas de desenvolvimento de software possam entregar novas funcionalidades do produto rapidamente e com qualidade. A implementação de CI/CD permite que as equipas colaborem de forma eficiente, automatizem o processo de compilação e testes, e entreguem funcionalidades de forma contínua.

Continuous Integration é uma abordagem que visa integrar constantemente o código desenvolvido por diferentes membros da equipa em um repositório central, geralmente usando ferramentas de automação (Liao, 2020), como o Jenkins (Jenkins, 2023) ou o Travis CI (Travis, 2023). O uso destas ferramentas permite a deteção precoce de conflitos de integração, correções rápidas de erros e um aumento na eficiência do processo de desenvolvimento. A automatização do processo de compilação e testes é crucial para garantir a qualidade do software e evitar erros humanos, gerando um *feedback* rápido sobre o estado da integração e da entrega, para que os problemas possam ser resolvidos de forma ágil.

Continuous Delivery, por sua vez, é uma prática que visa automatizar todo o processo de entrega do software, desde a compilação até a produção (*deploy*[2]). Isso é alcançado por meio de uma *pipeline* de entrega contínua, que consiste em etapas como teste automatizado, verificação de qualidade e produção. Esse processo garante a entrega constante de novas funcionalidades aos utilizadores, aumentando a agilidade e a qualidade do software.

É importante monitorar e medir o processo de CI/CD para identificar pontos de melhoria e garantir a eficiência da implementação (Sabharwal *et al.*, 2020; Nogueira & Zenha-Rela, 2021), assim como implementar medidas de segurança ao longo do processo para proteger o código e os dados sensíveis. Em resumo, o uso combinado de *Continuous Integration* e *Continuous Delivery* permite uma integração constante do código, um processo de teste automático e uma entrega confiável de software, o que resulta em um ciclo de feedback mais rápido e eficiente, aumentando a eficiência e a qualidade do processo de desenvolvimento de software.

A Fig. 1 apresenta o fluxograma de desenvolvimento de uma aplicação e onde é empregue o CI e CD, dividindo o fluxograma em duas partes. Sendo a primeira codificar, compilar e fazer testes. E a segunda a partir dos testes, gerar uma versão nova da aplicação e partir para o *deploy* que é a última etapa antes da *web app* estar operacional.

[2] *Deploy* de uma aplicação significa implantar ou disponibilizar essa aplicação para uso em um ambiente específico, como um servidor ou plataforma de hospedagem. É o processo de tornar a aplicação acessível e funcional para os usuários finais.

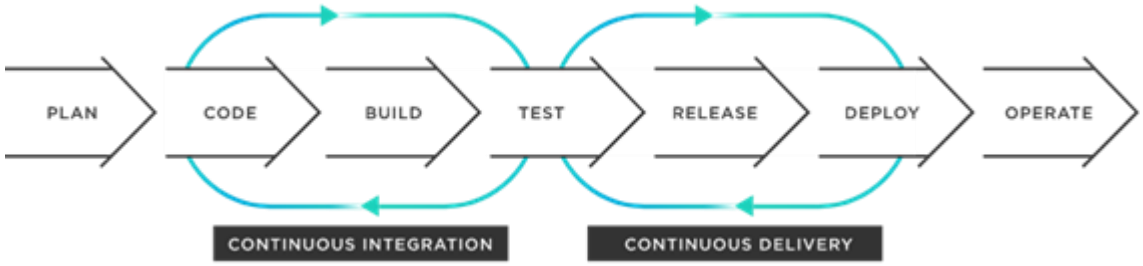

Figura 1 - Continuous Integration e Continuous Delivery, adaptado de TICBO (2023).

2.2.1 MELHORES PRÁTICAS

É importante lembrar que a implementação efetiva de CI/CD depende de muitos fatores, incluindo o tamanho e complexidade do projeto, as necessidades do cliente e a cultura da equipa de desenvolvimento de software. Portanto, é importante personalizar a implementação de CI/CD para atender às necessidades específicas do projeto e da equipa, seguindo as melhores práticas (Micco, 2021).

A primeira boa prática é a ***automação***, que envolve a automação do processo de compilação, teste e entrega de software. Isso garante a consistência do processo e possibilita a execução de tarefas repetidamente. Também é recomendável a ***utilização de ferramentas de integração contínua*** (CI), como o Jenkins, que permite automatizar o processo de compilação e teste do código. Isso garante que o código seja compilado e testado de forma consistente em todos os estágios do desenvolvimento. É importante reforçar que o Jenkins é uma plataforma de automação de código aberto amplamente utilizada para facilitar o CI/CD, permitindo automatizar o processo de desenvolvimento (compilação, preparação de recursos, gerenciamento de dependências), teste e *deploy* de aplicações, melhorando a eficiência, qualidade e velocidade do desenvolvimento de software.

É ainda importante ressaltar que testes automatizados são fundamentais para garantir a qualidade do software. Testes adequados devem ser implementados em todos os estágios do processo de desenvolvimento para garantir que os erros (*bugs*) sejam identificados e corrigidos precocemente.

Para facilitar a entrega de software, recomenda-se a implementação de um ambiente de entrega contínua (CD). Esse ambiente automatiza o processo de entrega de software e garante que o código seja entregue de forma confiável e consistente em todos os ambientes de desenvolvimento. Outra prática importante é o uso de ***ferramentas de monitorização de desempenho*** para acompanhar o desempenho do software em produção. Isso permite a identificação e correção de problemas rapidamente.

Um processo de *feedback* eficiente é outra prática recomendada. Isso permite que os desenvolvedores recebam feedback rápido sobre o código que estão a escrever e ajuda a garantir que as alterações sejam incorporadas ao código principal de forma rápida e eficiente. Por fim, a colaboração entre as equipas de desenvolvimento e operação garante que as alterações sejam implementadas de forma consistente em todos os ambientes.

Resumindo, a implementação dessas práticas pode aumentar a eficiência e a qualidade do processo de desenvolvimento de software, além de garantir a entrega rápida e consistente de software de alta qualidade (GKE, n.d.; JETBRAINS, 2022; Ziolkowski, 2022).

2.2.2 SEGURANÇA

Segurança é um fator essencial nos o processo de CI/CD (Rangnau *et al.*, 2020; Pan *et al.*, 2023), estas são algumas medidas de segurança importantes que devem ser implementadas para proteger o código e os dados sensíveis (durante o processo de CI/CD):

1. **Autenticação e autorização**: Medidas de autenticação forte e autorização para garantir que apenas utilizadores autorizados tenham acesso ao código e aos dados sensíveis.
2. **Criptografia de dados**: Criptografar todos os dados sensíveis, incluindo o código fonte, para garantir a privacidade e segurança.
3. **Segurança de rede**: Implementar medidas de segurança de rede, como *firewalls* e *Virtual Private Networks* (VPNs), para proteger o tráfego de rede e garantir a segurança dos dados sensíveis.
4. **Monitorização de segurança**: Monitorar o processo de CI/CD para identificar ameaças de segurança e tomar medidas para proteger o código e os dados sensíveis.
5. **Treino de segurança**: Treinar a equipa de desenvolvimento de software sobre práticas de segurança seguras, a fim de garantir que eles entendam a importância de proteger o código e os dados sensíveis.
6. **Atualizações de software**: Manter todas as ferramentas e tecnologias usadas no processo de CI/CD atualizadas e seguras, para garantir a segurança do código e dos dados sensíveis.
7. **Armazenamento seguro**: Armazenar o código e os dados sensíveis em servidores seguros, protegidos por medidas de segurança robustas.

A segurança deve ser uma preocupação constante durante todo o processo de CI/CD, é importante monitorar e atualizar as medidas de segurança periodicamente para garantir que elas acompanhem as mudanças e ameaças de segurança

2.3 FRAMEWORKS

Poderão ser utilizados diferentes softwares e *frameworks* para o desenvolvimento da uma *web app* (para visualização de parâmetros ambientais). Nesta seção serão apresentadas e analisadas as ferramentas mais usadas na empresa Atos (durante o estágio), bem como uma explicação e análise comparativa com outras existentes no mercado, para tal iremos dividir a análise em *frameworks front-end* e *back-end*.

Nos *frameworks* e restantes ferramentas apresentadas nas seguintes seções deste capítulo optou-se para deixar em último lugar de cada listagem a *framework*/ferramenta que foi usada para o desenvolvimento da *web app*, tendo sido feita para essa uma análise SWOT, i.e., a análise das *Strengths* (Forças), *Weaknesses* (Fraquezas), *Opportunities* (Oportunidades) e *Threats* (Ameaças) dessa *framework*/ferramenta.

2.3.1 FRONT-END

O *front-end*, também conhecido como interfaces do utilizador é responsável pela parte visual (interface gráfica) e interativa de uma aplicação ou *website*, desempenha um papel crucial na usabilidade, na acessibilidade e na satisfação do utilizador. Neste seção, exploraremos os principais elementos do *front-end*, discutiremos as tecnologias envolvidas e analisaremos como essa área influencia a interação entre utilizadores e sistemas.

O desenvolvimento *front-end* envolve o uso de várias linguagens de programação, ferramentas e *frameworks* para criar a interface de utilizador. As principais linguagens incluem HTML, CSS e JavaScript. A linguagem HTML é a base do *front-end*, ela define a estrutura e o conteúdo dos elementos da página, como texto, imagens, formulários e *links*. A linguagem de estilização CSS é utilizada para controlar a aparência visual dos elementos HTML. Ela permite definir cores, fontes, *layouts* e animações, tornando a interface do utilizador atraente e coesa. A linguagem de programação JavaScript é usada para adicionar interatividade e dinamismo às páginas web. Ela permite criar elementos interativos, validar formulários, realizar animações e manipular o conteúdo da página em tempo real. Também pode interagir com o HTML e o CSS, alterando-os dinamicamente com base nas ações do utilizador.

Existem vários *frameworks* populares que simplificam o desenvolvimento *front-end*, como React, Angular e Vue.js que serão detalhados de seguida. Eles fornecem uma estrutura e um conjunto de bibliotecas para facilitar a criação de interfaces mais complexas e responsivas. Com o aumento do uso de dispositivos móveis, é crucial que as interfaces *front-end* sejam responsivos, ou seja, se adaptem a diferentes tamanhos de ecrã. A abordagem "*mobile-first*"

prioriza o *design* e o desenvolvimento para dispositivos móveis, garantindo uma experiência consistente em todas as plataformas.

O *front-end* também deve ser acessível a todo o tipo de utilizador, incluindo pessoas com deficiências visuais, auditivas ou motoras. Isso envolve a utilização de práticas e padrões de acessibilidade, como a adição de descrições alternativas para imagens e o uso de técnicas de navegação por teclado. A interface deve ser intuitiva, com elementos de navegação claros e facilmente compreensíveis pelo utilizador. A disposição dos elementos, a legibilidade do texto e o *feedback* visual adequado são aspetos fundamentais para garantir uma boa experiência do utilizador.

A otimização do *front-end* é essencial para garantir um carregamento rápido e eficiente da página. O tamanho dos arquivos, a minimização de solicitações ao servidor e o uso adequado de técnicas de cache são alguns dos aspetos considerados para melhorar o desempenho. Uma interface bem projetada e responsiva melhora a experiência do utilizador, permitindo que ele aceda e interaja com o sistema de maneira confortável em diferentes dispositivos.

O desenvolvedor *front-end* desempenha um papel fundamental na criação de interfaces atraentes, intuitivas e responsivas para aplicações e *websites*, este deverá compreender os fundamentos, dominar as tecnologias envolvidas e adotar práticas de usabilidade e acessibilidade são aspetos essenciais para garantir uma experiência positiva do utilizador. O contínuo avanço das tecnologias *front-end* abre caminho para interfaces cada vez mais ricas e interativas, possibilitando uma interação efetiva e envolvente entre os usuários e os sistemas.

A Fig. 2 ilustra a divisão do *front-end* e do *back-end* (descrita na próxima seção), expondo os elementos de cada camada.

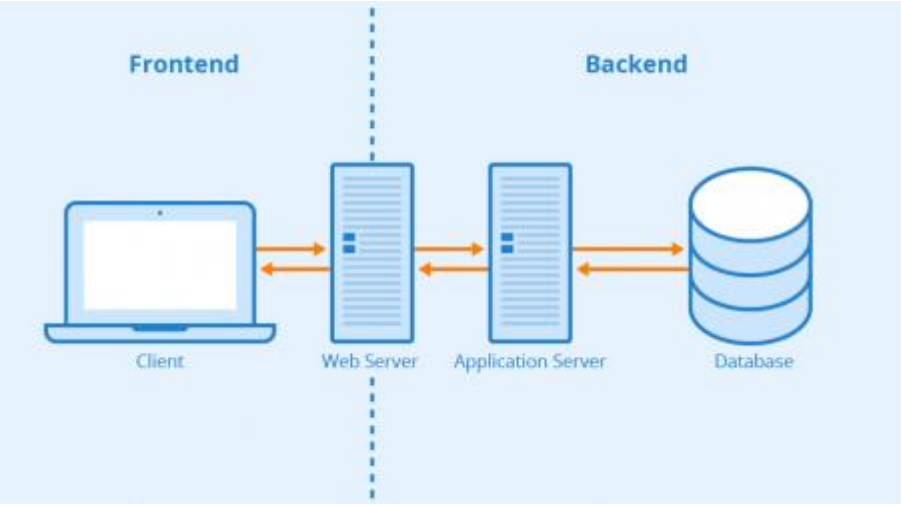

Figura 2 Front-end and back-end, adaptado de Seobility (2023).

Existem vários *frameworks* JavaScript para a construção de aplicações web, cada um desses *frameworks* possui suas próprias vantagens e desvantagens, e a escolha do *framework* depende do tipo de projeto que está sendo desenvolvido e das preferências pessoais do desenvolvedor. De seguida serão listados os principais *frameworks* JavaScript utilizados e suas principais diferenças. O **React** (React, 2023) é um popular *framework* JavaScript para a construção de interfaces de utilizador, criado e mantido pelo Facebook. Ele se concentra na construção de componentes reutilizáveis, também permitindo que os desenvolvedores criem interfaces de utilizador complexas e dinâmicas com facilidade. A principal diferença entre o React e o Angular (Angular, 2023), ver em baixo, é que o React não é um *framework* completa, ele se concentra apenas na construção de interfaces de utilizador. Para a construção de uma aplicação completa, o React precisa ser combinado com outras ferramentas e bibliotecas. Outra diferença notável é que o Angular utiliza o TypeScript como linguagem padrão, enquanto o React é mais flexível e pode ser usado com JavaScript puro.

O **Vue.js** (Vue.js, 2023) é um *framework* também JavaScript para a construção de interfaces de utilizador, que tem ganho popularidade nos últimos anos. Ele é semelhante ao Angular e ao React em muitos aspetos, mas é mais leve e fácil de aprender. O Vue.js possui um sistema de *templates* simples e flexível, o que o torna uma boa escolha para pequenos projetos e protótipos. Como o Angular, o Vue.js utiliza uma abordagem baseada em componentes, mas é mais simples. O Vue.js também é altamente customizável e permite que os desenvolvedores escolham quais recursos e bibliotecas integrar, ao contrário do Angular, que oferece um conjunto completo de recursos.

O já acima referido **Angular** (Angular, 2023) é o *framework* proposto inicialmente neste estágio pela Atos para desenvolver a *web app*, usa JavaScript *open-source* criado e mantido pela Google, que é utilizado para construir aplicações web escaláveis e dinâmicas. Ele é uma evolução do antigo AngularJS, apresenta uma abordagem baseada em componentes para a construção de interfaces de utilizador. Com o Angular, é possível criar interfaces de utilizador complexas e interativas, que se adaptam a diferentes dispositivos e tamanhos de ecrã. Ele utiliza o conceito de *Single Page Application* (SPA), que permite carregar uma única página HTML e atualizar a interface do utilizador dinamicamente conforme o utilizador interage com a aplicação.

O Angular também possui um conjunto de ferramentas que tornam o desenvolvimento mais produtivo, incluindo um poderoso sistema de injeção de dependências, ferramentas para testes automatizados, suporte a internacionalização, entre outras. Além disso, o Angular é altamente

compatível com outras ferramentas e tecnologias, permitindo sua integração com bibliotecas de terceiros, *Application Programming Interface* (API) *Representational State Transfer* (REST), bases de dados, entre outras. O que torna o Angular uma ferramenta poderosa para a construção de aplicações web modernas, permitindo que desenvolvedores criem interfaces de utilizadores complexas e dinâmicas de forma produtiva e escalável (Angular, 2023). A Fig. 3 apresenta a análise SWOT do Angular.

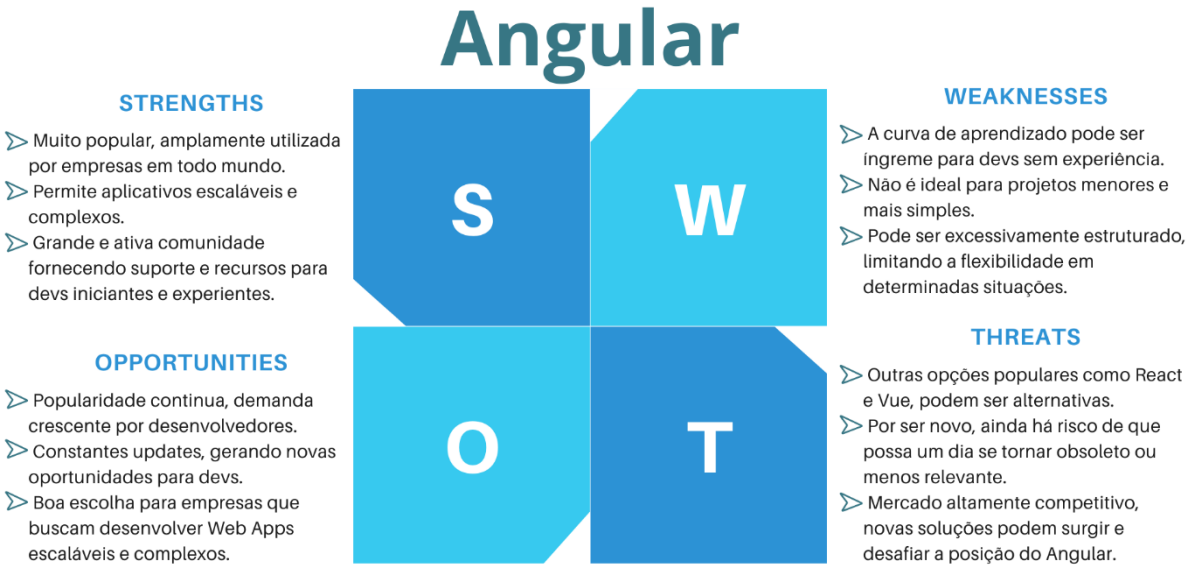

Figura 3 SWOT do Angular.

2.3.2 BACK-END

O *back-end* é responsável pela lógica de negócios, pelo processamento de dados e pela comunicação entre o *front-end* e a base de dados. Serão explorados os principais elementos do *back-end*, as arquiteturas empregadas e como influencia a funcionalidade e o desempenho de uma aplicação.

As linguagens de programação mais comuns no desenvolvimento *back-end* incluem Java, C#, Python, Ruby e Node.js. Essas linguagens permitem a criação de funcionalidades complexas, manipulação de dados e interação com outros componentes do sistema. Os servidores web, como Apache, Nginx e Microsoft IIS, são componentes essenciais no *back-end*. Eles são responsáveis por receber as requisições do *front-end*, processá-las e enviar as respostas adequadas. Os servidores também podem gerir sessões, autenticação e segurança.

As APIs são utilizadas no *back-end* para fornecer um conjunto de funcionalidades que podem ser acedidas pelo *front-end* ou por outros sistemas externos, elas permitem a integração entre diferentes partes do sistema e a troca de informações de forma estruturada. Em geral temos diversos tipos de arquitetura, como por exemplo, a monolítica que é caracterizada pela

construção de um único sistema onde todas as funcionalidades estão agrupadas em um único código-fonte. Embora seja simples, essa arquitetura pode ser difícil de escalar e manter, especialmente em sistemas complexos.

A arquitetura em camadas, que divide o sistema em diferentes camadas lógicas, como a camada de apresentação, a camada de lógica de negócios e a camada de acesso a dados. Essa abordagem facilita a manutenção e a escalabilidade do sistema, permitindo a separação de preocupações e a reutilização de componentes.

Por último, a arquitetura em microserviços é caracterizada pela divisão do sistema em pequenos serviços independentes, cada um responsável por uma funcionalidade específica. Esses serviços podem ser desenvolvidos, implantados e escalados de forma independente, proporcionando maior flexibilidade e facilidade de manutenção. A Fig. 4 ilustra a diferença entre as arquiteturas monolítica e de microserviços. Observando a figura, fica fácil entender porque a arquitetura em microserviços é mais fácil para manter e escalar, também para diferentes desenvolvedores trabalharem em conjunto.

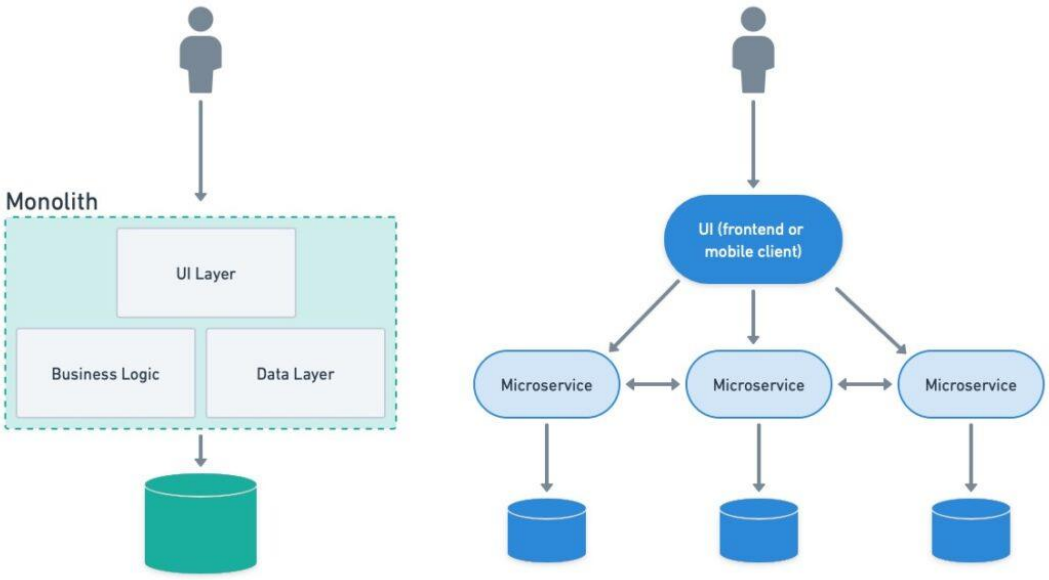

Figura 4 Arquitetura monolítica vs. microserviços, adaptado de Semaphore (2023).

Reforçando, o *back-end* é responsável por armazenar, recuperar e manipular dados na base dados, isso inclui a criação de consultas, a garantia da integridade dos dados e a implementação de mecanismos de segurança. Também lida com a autenticação e autorização de utilizadores, garantindo que apenas utilizadores autorizados tenham acesso a determinadas funcionalidades e recursos do sistema. O *back-end* recebe as requisições do *front-end*, processa-as de acordo com as regras de negócio, realiza operações na base de dados, e retorna as respostas adequadas.

Pelo exposto, o *back-end* desempenha um papel fundamental no desenvolvimento de sistemas, sendo responsável pela lógica de negócios, processamento de dados e comunicação com outros componentes do sistema. Compreender os fundamentos do *back-end*, escolher a arquitetura adequada e dominar as tecnologias envolvidas são aspetos essenciais para construir sistemas eficientes, escaláveis e seguros.

O contínuo avanço das tecnologias *back-end*, proporciona oportunidades para o desenvolvimento de sistemas cada vez mais robustos e funcionais, de seguida serão listados alguns *frameworks* para *back-end* bastante utilizados e suas principais diferenças, tal como feito para os *frameworks front-end*, a listagem termina no *framework* sugerido pela Atos para iniciar os trabalhos do estágio.

O **Micronaut** (Micronaut, 2023) é um *framework* Java recente, modular que é rápido, leve e fácil de usar. Ele tem como objetivo facilitar a criação de aplicações Java para a nuvem, microserviços e aplicações de *Internet of Things* (IoT). O Micronaut, como o Spring Boot (Spring, 2023), ver em baixo, usa anotações para simplificar o desenvolvimento e fornece uma ampla gama de recursos, incluindo injeção de dependência, suporte a bases de dados, integração com sistemas de mensagens, entre outros. O Micronaut utiliza uma abordagem mais modular de configuração, o que significa que os desenvolvedores precisam configurar apenas os módulos necessários para a aplicação em vez de uma configuração mais ampla. O que torna uma vantagem, mas o Spring Boot é conhecido por ter uma documentação completa e abrangente, com uma grande quantidade de recursos disponíveis para os desenvolvedores. Embora o Micronaut também tenha uma documentação abrangente, ele ainda não tem tantos recursos e tutoriais quanto o Spring Boot.

Quarkus (Quarkus, 2023) é uma *framework* Java nativo para a nuvem que permite a criação de aplicações Java ultra rápidas e leves. Ele é otimizado para uso em *conteiners* e é executado com baixo consumo de memória e CPU (*Central Process Unit*). Quarkus é conhecido por seu forte suporte a tecnologias nativas da nuvem, enquanto o Micronaut tem um foco maior na modularidade e no suporte a várias linguagens de programação *Java Virtual Machine* (JVM). O Quarkus usa uma abordagem de codificação orientada a anotações semelhante ao Spring Boot e oferece um amplo suporte a tecnologias populares, como Hibernate, Kafka, Kubernetes e outras. Quarkus também tem uma grande ênfase na produtividade do desenvolvedor, fornecendo recursos como *Hot Reload* e *Live Coding*. Como esta *framework* é otimizada para uso em *conteiners*, é mais veloz e eficaz em termos de desempenho. Em contraste, o Spring Boot é um pouco mais lento em termos de inicialização e consumo de recursos, o que não chega a ser um problema já que o Spring Boot tem um

ecossistema de plugins e bibliotecas muito grande e estabelecido, enquanto o ecossistema do Quarkus ainda está em desenvolvimento, embora esteja crescendo rapidamente.

O já acima referido **Spring Boot** (Spring, 2023) é uma *framework* proposta inicialmente neste estágio pela Atos para desenvolver a *web app*, permite o desenvolvimento de aplicações Java que tem como objetivo simplificar e agilizar o processo de criação de aplicações, fornecendo um conjunto de ferramentas e convenções pré-definidas para facilitar o desenvolvimento. O Spring Boot é baseado no *framework* Spring e segue os mesmos princípios de modularidade, escalabilidade, testabilidade e segurança. Com o Spring Boot, os desenvolvedores podem concentrar-se na lógica de negócios das suas aplicações, sem se preocupar com a configuração e integração de tecnologias e serviços. Ele possui uma série de recursos que automatizam tarefas comuns de desenvolvimento, como configuração de bases de dados, gerenciamento de dependências, segurança, *loging* e muito mais.

O Spring Boot é uma escolha popular para a construção de aplicações web e APIs REST, bem como para a criação de microsserviços, pois fornece uma maneira fácil e rápida de criar, publicar e escalar aplicações. Ele também é compatível com uma ampla variedade de ferramentas e tecnologias, o que o torna uma opção flexível para desenvolvedores Java. A Fig. 5 apresenta a análise SWOT do Spring Boot.

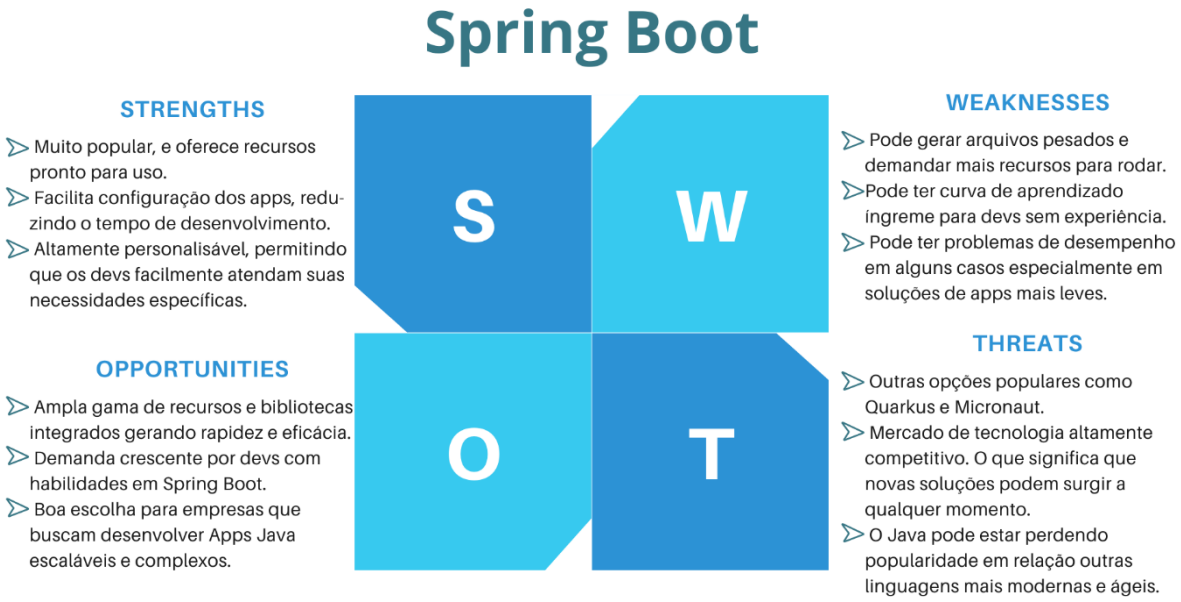

Figura 5 Spring Boot SWOT.

2.4 BASE DE DADOS

Esta seção descreve sucintamente o que são Bases de Dados (BD), destacando seus fundamentos, diferentes tipos e ferramentas para o seu uso. Bases de dados é um conceito

fundamental em ciência da computação, refere-se a uma coleção organizada de dados relacionados, armazenados e acedidos eletronicamente. Uma BD permite que informações estruturadas sejam armazenadas de forma eficiente, manipuladas, consultadas e atualizadas por meio de operações definidas. As operações básicas são comumente referidas como CRUD, que significa *Create* (Criar), *Read* (Ler), *Update* (Atualizar) e *Delete* (Excluir).

Existem vários tipos de BD, cada uma projetada para atender a diferentes necessidades e requisitos específicos. Alguns dos principais tipos de BD incluem:

(a) Bases de dados relacionais (RDBMS - *Relational Database Management System*) que são baseados no modelo relacional, que organiza os dados em tabelas com linhas e colunas. Essas tabelas possuem esquemas definidos, onde as relações entre os dados são estabelecidas por meio de chaves primárias e chaves estrangeiras. Exemplos populares de sistemas de gestão de BD relacionais incluem o MySQL, PostgreSQL, Oracle e SQL Server. Estas BD são amplamente utilizados em várias aplicações, desde sistemas empresariais até sites dinâmicos (Arshad *et al.*, 2023).

(b) Bases de dados NoSQL (*Not Only* SQL), são projetados para armazenar e recuperar grandes volumes de dados não estruturados ou semi-estruturados. Eles oferecem esquemas flexíveis, escalabilidade horizontal e alta disponibilidade. Existem diferentes tipos de DB NoSQL, como BD de documentos (exemplo: MongoDB), BD de chave-valor (exemplo: Redis), BD de coluna ampla (exemplo: Cassandra) e bancos de dados de gráficos (exemplo: Neo4j). Estas BD são amplamente utilizadas em aplicações web, análise de dados, IoT e outros casos em que a flexibilidade e a escalabilidade são prioritárias.

Outro tipo de bases de dados são as orientadas a objetos, projetadas para armazenar e manipular objetos complexos, podem incluir dados estruturados e comportamentos associados. Eles permitem que objetos sejam armazenados diretamente na BD, preservando suas características e relações. Exemplos de BD orientadas a objetos incluem o Db4o e o ObjectDB. Estas BD são frequentemente usados em aplicações que lidam com sistemas de CAD (*Computer-Aided Design*), sistemas de simulação e jogos (Reddy *et al.*, 2022).

Esses são apenas alguns exemplos dos tipos de BD existentes. No entanto, vale ressaltar que a escolha do tipo de BD depende dos requisitos específicos da aplicação, como escalabilidade, desempenho, integridade dos dados, complexidade da estrutura dos dados, entre outros fatores. É comum utilizar uma combinação de diferentes tipos de BD num sistema, dependendo das necessidades de armazenamento e acesso aos dados.

No caso deste estágio foi solicitado pelo empresa o uso de DB relacionais para o desenvolvimento da *web app*. Existe uma infinidade de sistemas gerenciadores de BD

relacionais disponíveis, de seguida serão apresentados três deles, não significando que os escolhidos para a comparação são os melhores ou mais utilizados, mas com certeza estão entre os mais populares do mercado.

O **Oracle Database** (Oracle, 2023) é um RDBMS desenvolvido e comercializado pela Oracle Corporation, é conhecida por ser altamente escalável, segura e confiável, sendo usado principalmente em empresas e organizações com grandes volumes de dados. Algumas das principais diferenças em relação ao MySQL (Kofler, 2005), ver em baixo, incluem a capacidade de processar grandes volumes de dados e transações em tempo real, recursos avançados de segurança, e suporte a recursos de inteligência artificial e *machine learning*. Outro fator a ser levado em consideração é que o Oracle *database* tem um custo total de propriedade geralmente mais alto devido à sua licença e recursos avançados, enquanto o MySQL é geralmente mais acessível e adequado para pequenas e médias empresas.

O **PostgreSQL** (PostgreSQL, 2023) é um RDBMS *open source*, desenvolvido pela comunidade PostgreSQL Global Development Group. O PostgreSQL é conhecido por sua confiabilidade, escalabilidade e recursos avançados, como suporte a ACID[3] (*Atomicity, Consistency, Isolation, and Durability*), controle de concorrência multiversão e extensibilidade. O PostgreSQL é distribuído sob a licença BSD (licença de distribuição de Software Berkeley; tipo de licença de código aberto que permite a livre utilização, modificação e distribuição de software), que é mais permissiva, dando assim a liberdade de alterar seus códigos e depois ainda utilizar para fins comerciais, gratuitamente. O PostgreSQL oferece uma variedade maior de tipos de dados, como tipos geométricos e de rede.

Algumas das principais diferenças do PostgreSQL em relação ao MySQL inclui a capacidade de suportar dados espaciais e geoespaciais, tipos de dados personalizados, e a

[3] ACID (Navathe & Elmasri, 2005) é um acrônimo que se refere a um conjunto de propriedades de transações em banco de dados. Atomicidade: significa que uma transação é tratada como uma unidade atômica, sendo assim, todas as operações dentro dela são executadas ou nenhuma delas é executada. Se uma das operações falhar, todas as operações são desfeitas para que o banco de dados não fique em um estado inconsistente. Consistência: significa que uma transação deve levar o banco de dados de um estado consistente para outro estado consistente. Isso garante que todas as restrições e regras de integridade referencial do banco de dados sejam respeitadas. Isolamento: significa que cada transação deve ser isolada de outras transações em andamento no banco de dados. Isso garante que cada transação seja executada independentemente das outras, sem interferir nas operações de outras transações. Durabilidade: significa que as mudanças realizadas por uma transação devem ser permanentes e resistir a falhas do sistema, como falhas de energia ou falhas de hardware.

capacidade de criar e usar funções definidas pelo utilizador. O MySQL é amplamente utilizado em aplicações com muitos pedidos e tem melhor desempenho em operações de leitura e gravação em massa, enquanto o PostgreSQL é mais adequado para aplicações que requerem transações complexas e alta integridade de dados.

O sistema de gerenciamento de BD proposto pela empresa para ser usada neste estágio é o **MySQL** (Kofler, 2005), é RDBMS *open source* que é amplamente utilizado em aplicações web e empresariais, que implementa o modelo relacional de BD, permitindo que os dados sejam armazenados em tabelas com relações entre elas. É uma das soluções de BD mais populares do mundo devido ao seu alto desempenho, confiabilidade, escalabilidade e facilidade de uso. Ela suporta várias linguagens de programação, incluindo Java, Python, PHP e C++, sendo compatível com muitos sistemas operativos, como Windows, Linux e MacOS. MySQL é compatível com propriedades ACID que garantem a validade dos dados, independentemente de erros, falhas ou outros possíveis contratempos (MySQL, n.d.; MySQL, 2023). A Fig. 6 apresenta a análise SWOT do MySQL.

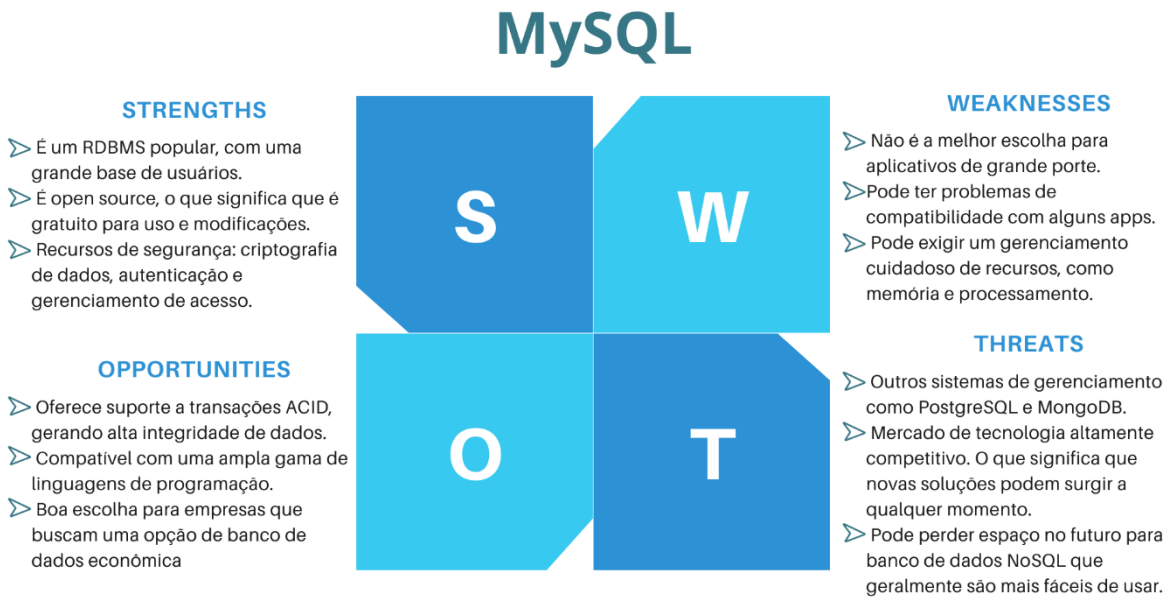

Figura 6 MySQL SWOT.

Em resumo, o MySQL é projetado para oferecer desempenho otimizado em ambientes de alta concorrência de transações, usando técnicas como *caching*, indexação e otimização de consultas para maximizar a eficiência da DB. Além disso oferece várias opções de replicação e balanceamento de carga para garantir alta disponibilidade e escalabilidade horizontal (MySQL, 2023). Assim, o MySQL é amplamente utilizado em muitas aplicações web e sites, incluindo WordPress, Facebook, Twitter e YouTube. É conhecido por ser fácil de instalar,

configurar e usar, com muitas ferramentas e recursos disponíveis para os utilizadores, como o MySQL *Workbench*, que permite aos desenvolvedores criar, gerir e visualizar esquemas de BD, bem como realizar consultas SQL complexas.

2.5 SISTEMAS DE AUTENTICAÇÃO

Esta seção expõe uma explicação sobre sistemas de autenticação, abordando seus conceitos fundamentais, os mecanismos comumente utilizados e as medidas de segurança associadas. Os sistemas de autenticação desempenham um papel crucial na proteção de informações sensíveis e no controle de acesso a recursos. São utilizados para verificar a identidade dos utilizadores e conceder acesso a recursos específicos.

A identificação é o processo de fornecer uma identidade exclusiva a um utilizador, geralmente na forma de um nome de utilizador ou identificador único, i.e., a autenticação é o processo de verificar a identidade do utilizador por meio de credenciais, como senhas, *tokens* ou certificados digitais. Os sistemas de autenticação podem utilizar diferentes fatores para verificar a identidade de um utilizador.

Os três fatores comumente empregados são: (i) algo que o utilizador sabe (senha), (ii) algo que o utilizador possui (*token* ou dispositivo físico) e (iii) algo que o utilizador é (biometria).

A autenticação baseada em senhas é amplamente utilizada e envolve o uso de uma combinação de *login* de utilizador e senha para verificar a identidade. É importante que as senhas sejam complexas, armazenadas de forma segura e protegidas contra-ataques, como força bruta. O MFA (*multifactor authentication*), autenticação multifator, envolve o uso de dois ou mais fatores para verificar a identidade do utilizador, permitindo adicionar uma camada extra de segurança, exigindo que o utilizador forneça uma segunda forma de autenticação, como um código enviado por SMS (*Short Message Service*), um *token* gerado por uma aplicação de autenticação ou uma impressão digital.

A autenticação de certificados digitais envolve o uso de chaves criptográficas para verificar a identidade do utilizador. Os certificados digitais são emitidos por autoridades de certificação confiáveis e garantem a autenticidade e a integridade das informações transmitidas. Na autenticação por *token*, o utilizador recebe um *token* exclusivo, que é um código único e temporário. Esse *token* pode ser um valor alfanumérico longo ou um arquivo criptografado. Quando o utilizador tenta aceder um sistema ou recurso protegido, ele apresenta o *token* em vez de fornecer credenciais tradicionais. O servidor ou serviço recebe o *token* e verifica sua validade, se o *token* for válido e correspondente ao utilizador, o acesso é concedido.

A autenticação por *token* oferece algumas vantagens em relação às credenciais tradicionais, como senhas, melhora a segurança, pois os *tokens* são geralmente mais difíceis de serem adivinhados ou comprometidos do que senhas (fracas). Além disso, como os *tokens* são temporários, eles podem expirar após um determinado período, aumentando a segurança ainda mais. Os *tokens* também podem ser usados para fornecer acesso a recursos específicos em vez de conceder acesso completo a um sistema. Isso é especialmente útil em sistemas de autorização granular, onde diferentes utilizadores têm permissões diferentes para diferentes partes do sistema.

Por último, a autenticação por *token* é amplamente utilizada em aplicações web e APIs para proteger o acesso a recursos e serviços. Exemplos de implementações de autenticação por *token* incluem JWT (JSON Web Tokens), OAuth e OpenID Connect (Pluralsight, 2023). A Fig. 7 ilustra um exemplo de autenticação por *token*.

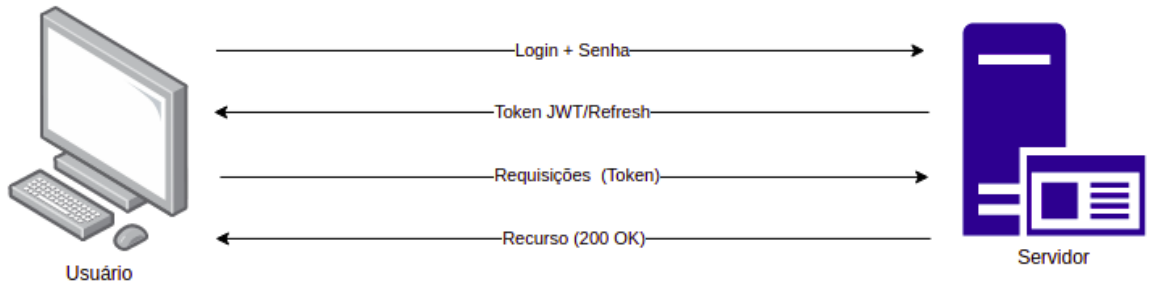

Figura 7 Autenticação por token, adaptado de Bessa (2021).

Algumas medidas de segurança são indispensáveis em sistemas de autenticação, como a criptografia, que é uma medida de segurança essencial para proteger as informações de autenticação durante a transmissão e armazenamento.

Protocolos criptográficos, como o SSL (*Secure Sockets Layer*) ou TLS (*Transport Layer Security*), garantem a confidencialidade e a integridade dos dados. Também é preciso levar em consideração a gestão de senhas/*passwords*, que feito de maneira adequada, inclui práticas como o armazenamento seguro das senhas, o uso de algoritmos de *hash* seguros e a implementação de políticas de senha fortes. A implementação de sistemas de monitoramento e deteção de atividades suspeitas, que permite identificar tentativas de autenticação não autorizadas ou comportamentos anômalos, possibilitando ação rápida para proteger o sistema é de extrema importância.

O contínuo avanço das tecnologias e das ameaças de segurança exige a constante atualização e aprimoramento dos sistemas de autenticação, visando oferecer uma experiência

segura e confiável aos utilizadores, a isso acresce a crescente adoção de serviços *online* e/ou aplicações como a que se apresentam de seguida.

O **Okta** (Okta, 2023), que é um servidor de autenticação em nuvem que oferece recursos semelhantes ao Keycloak (Keycloak, 2023), detalhado em baixo, como autenticação multifatorial, gerenciamento de utilizadores e provisionamento de contas. Uma das principais diferenças entre o Okta e o Keycloak é que o Okta é oferecido como um serviço hospedado, enquanto o Keycloak é uma solução *open source* que pode ser executada em ambientes locais ou em nuvem. O Okta também é mais voltado para empresas de grande porte, com preços mais altos e recursos mais avançados de gerenciamento de identidade.

O **Gluu** (Gluu, 2023) é mais um servidor de autenticação *open source* que oferece recursos semelhantes ao Keycloak, como autenticação MFA, gerenciamento de utilizadores e integração com provedores de identidade externos. Uma das principais diferenças entre o Gluu e o Keycloak é que o Gluu é mais focado em soluções de identidade federada, onde um utilizador faz *login* uma única vez e pode aceder a várias aplicações ou serviços sem precisar fazer *login* novamente, e tem recursos avançados de gerenciamento de confiança e de políticas de acesso. O Gluu também tem um modelo de negócios baseado em assinaturas, o que pode ser uma vantagem para empresas que preferem um modelo de suporte pago em vez de confiar apenas na comunidade *open source*.

Como referido, existem vários outros servidores de autenticação disponíveis no mercado que são semelhantes em termos de recursos e funcionalidades. O servidor de autenticação que a Atos propôs para o desenvolvimento da *web app* foi o **Keycloak** (Keycloak, 2023), desenvolvido pela Red Hat, o Keycloak é um servidor de autenticação e autorização baseado em *open-source*, como OAuth 2.0 e OpenID Connect, que foi criado para atender a essas necessidades. Este sistema permite que as empresas gerenciem utilizadores e grupos, autenticação e autorização, integração de provedores de identidade externos e gerenciamento de sessões. Ele também oferece recursos de segurança avançados, como autenticação MFA e proteção contra ataques de força bruta.

Uma das principais vantagens do Keycloak é sua capacidade de integrar-se com provedores de identidade externos, como o Facebook, o Google ou o GitHub. Isso permite que os utilizadores sejam autenticados usando as suas credenciais de outras contas, evitando a necessidade de criar e lembrar-se de senhas separadas para cada aplicação ou serviço. Além de altamente configurável e escalável, tornando-o uma solução popular para empresas que precisam de gerenciamento de identidade robusto e seguro para suas aplicações e serviços. Ele

pode ser executado em um único servidor ou distribuído em um cluster para fornecer alta disponibilidade e escalabilidade horizontal. A Fig. 8 apresenta a análise SWOT do Keycloak.

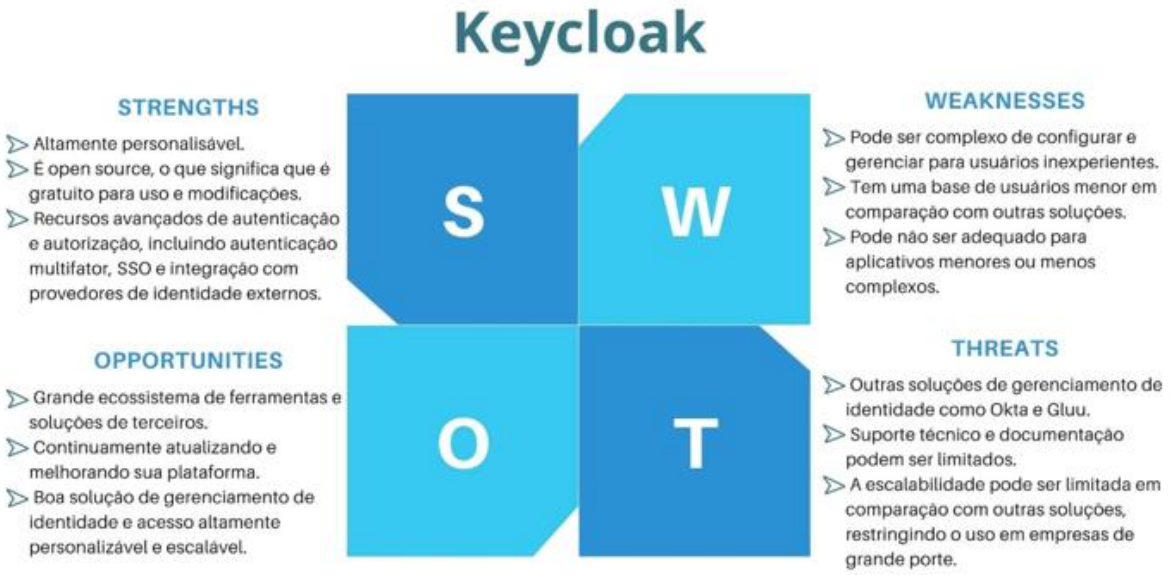

Figura 8 Keycloack SWOT.

2.6 MESSAGE BROKER

O *Message Broker* (MB) (Duarte *et al.*, 2022) é uma peça fundamental na arquitetura de sistemas assíncronos, atuando como intermediário entre produtores e consumidores de mensagens. Nesta seção serão abordados os seus conceitos fundamentais, o seu funcionamento e as suas aplicações no contexto de sistemas distribuídos. O MB é um componente de software que desempenha um papel essencial na comunicação assíncrona entre diferentes partes de um sistema distribuído.

No contexto do *Message Broker*, as mensagens são unidades de informação que são enviadas e recebidas pelos componentes do sistema distribuído. Elas podem conter dados estruturados, eventos ou comandos, que são encapsuladas em um formato apropriado para a transmissão. O MB atua como intermediário entre os produtores e consumidores de mensagens. Os produtores são responsáveis por gerar e enviar as mensagens, enquanto os consumidores as recebem e processam. Garantindo a entrega confiável das mensagens a gestão e a comunicação assíncrona entre os componentes.

Geralmente o MB oferece duas abordagens para a troca de mensagens: (i) filas e (ii) tópicos. Nas filas, as mensagens são entregues a um único consumidor, garantindo a ordem de processamento. Nos tópicos, as mensagens são distribuídas para vários consumidores

interessados no mesmo tipo de informação, permitindo a comunicação de estilo pub/sub (publicação/assinatura).

O *Message Broker* é responsável por garantir a entrega confiável das mensagens, ele gere mecanismos de persistência, como o armazenamento em disco, para garantir que as mensagens não sejam perdidas em caso de falhas ou interrupções na rede. Além disso, podem ser implementados mecanismos de confirmação de recebimento (*acknowledgment*) para garantir que as mensagens sejam processadas com sucesso.

O MB permite assim o roteamento eficiente das mensagens para os consumidores corretos, isto pode ser baseado em critérios como o conteúdo da mensagem, metadados ou regras de filtragem definidas pelo sistema. O roteamento adequado garante que as mensagens sejam entregues apenas aos consumidores interessados.

O MB facilita a integração de sistemas heterogêneos, permitindo a troca de mensagens entre diferentes plataformas e tecnologias. Ele atua como uma camada intermediária que abstrai a complexidade da comunicação entre os sistemas, garantindo a interoperabilidade e a escalabilidade. O *Message Broker* é amplamente utilizado em sistemas assíncronos, nos quais o processamento dos componentes não precisa ocorrer em tempo real. Ele permite o desacoplamento entre produtores e consumidores, possibilitando a escalabilidade horizontal e o balanceamento de carga. O MB é um componente-chave na arquitetura orientada a eventos (*Event-Driven Architecture*). Ele viabiliza a comunicação entre os diversos eventos gerados pelo sistema e os componentes interessados em reagir a esses eventos, isto permite a construção de sistemas altamente responsivos e flexíveis. Por último, o *Message Broker* desempenha um papel essencial na troca eficiente de mensagens em sistemas distribuídos.

Compreender os fundamentos do MB, seus mecanismos de funcionamento e suas aplicações práticas é crucial para projetar sistemas robustos, escaláveis e assíncronos. O contínuo avanço das tecnologias de comunicação e os requisitos cada vez mais complexos das aplicações impulsionam a evolução do MB, oferecendo soluções cada vez mais poderosas e eficientes. A Fig. 9 apresenta um exemplo de comunicação em tópicos, modelo pub/sub.

Figura 9 Message Broker, modelo pub/sub, adaptado de Kouomeu (2022).

De seguida serão apresentados três exemplos de servidor de mensagens. A escolha entre esses softwares dependerá das necessidades específicas de cada projeto ou aplicação. O **RabbitMQ** (RabbitMQ, 2023) é um software de mensagens *open source* que implementa o protocolo AMQP (*Advanced Message Queuing Protocol*). Ele oferece recursos semelhantes ao ActiveMQ (ActiveMQ, 2023a), detalhado em baixo, como suporte a filas, tópicos e canais de publicação/assinatura, bem como suporte a várias linguagens de programação, como Java, C#, Python e Ruby (LuisDEV, 2022). Uma das principais diferenças entre o RabbitMQ e o ActiveMQ é que o RabbitMQ é projetado com recursos avançados de *clustering* e replicação de mensagens. Além disso, o RabbitMQ oferece suporte a vários protocolos de rede, como TCP (*Transmission Control Protocol*), HTTP (*Hypertext Transfer Protocol*), STOMP (*Streaming Text Oriented Messaging Protocol*) e MQTT (*Message Queuing Telemetry Transport*).

O **Apache Kafka** (Kafka, 2023) é uma plataforma de *streaming* de dados distribuída que é amplamente utilizada em ambientes de *Big Data*. Ele oferece recursos semelhantes ao ActiveMQ, como suporte a filas e canais de publicação/assinatura, mas é projetado especificamente para lidar com grandes volumes de dados em tempo real. O Kafka é escalável, tolerante a falhas e altamente disponível, com recursos avançados de particionamento e replicação de dados. Além disso, ele oferece recursos avançados de *streaming* de dados, como processamento de fluxo de eventos e agregação de dados em tempo real.

O **ActiveMQ** (ActiveMQ, 2023a, b) foi o MB proposto pela empresa para o desenvolvimento da *web app*, é um software servidor de mensagens *open source* que implementa o protocolo JMS (*Java Message Service*), que é um padrão para comunicação assíncrona entre aplicações em uma rede. O ActiveMQ permite que as aplicações enviem e recebam mensagens em filas, tópicos e canais de publicação/assinatura, que são geridos por

um servidor de mensagens central. O ActiveMQ oferece uma série de recursos avançados, como suporte a *failover* e *clustering* para alta disponibilidade, integração com várias linguagens de programação, incluindo Java, C++, .NET, Ruby, Python e PHP (ActiveMQ, 2023b), suporta protocolos de rede, como TCP, SSL e HTTP.

Este MB é amplamente utilizado em ambientes de integração de aplicações e em sistemas distribuídos, onde a comunicação assíncrona é essencial para garantir o desempenho e a escalabilidade do sistema. O ActiveMQ é um dos servidores de mensagens mais populares e amplamente utilizados no mundo, devido à sua flexibilidade, escalabilidade e facilidade de uso, ele é mantido pela Apache Software Foundation e está disponível gratuitamente sob a licença Apache 2.0. A Fig. 10 apresenta a análise SWOT do ActiveMQ.

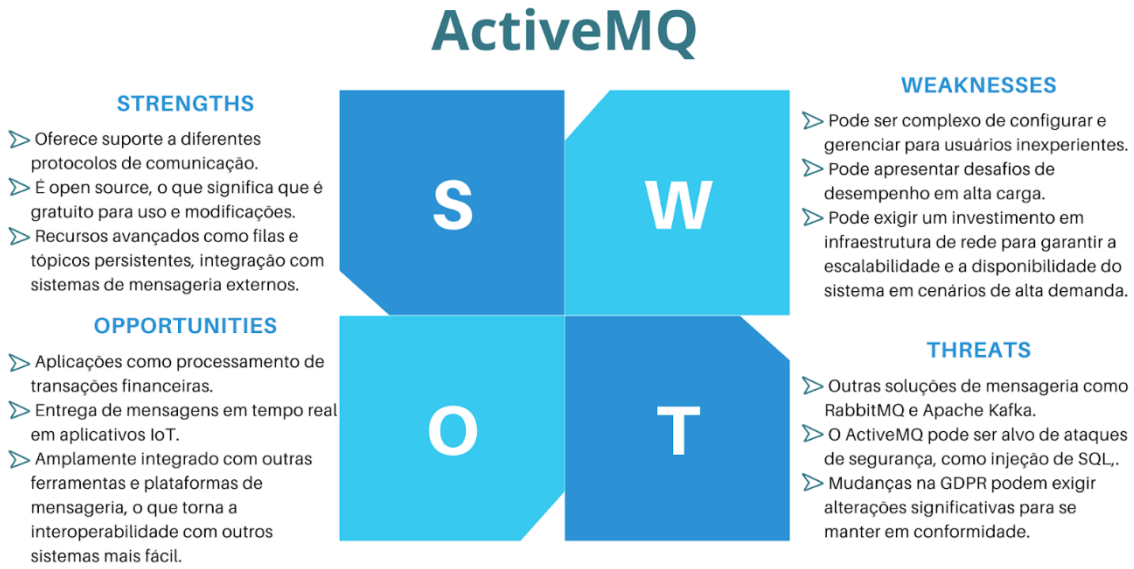

Figura 10 ActiveMQ SWOT.

2.7 SCRUM

Scrum é uma *framework* ágil utilizado para gerir projetos complexos, especialmente no desenvolvimento de software. Foi projetado para permitir que as equipas sejam mais ágeis, flexíveis e responsivas às mudanças. Enfatizando a colaboração, a transparência e a entrega incremental de valor (Takeuchi & Nonaka, 1986; Patrucco *et al.*, 2022).

No Scrum, o trabalho é dividido em iterações chamadas de *sprints*, que geralmente têm uma duração fixa de uma a quatro semanas (Hron & Obwegeser, 2022; Aripradono, & Hisham, 2022). Cada *sprint* começa com uma reunião de planeamento, onde a equipa seleciona um conjunto de itens de trabalho a serem realizados durante o *sprint*, com base nas prioridades e capacidades da equipa. Durante o *sprint*, a equipa realiza reuniões diárias curtas chamadas de *daily scrums* ou *stand-ups*. Essas reuniões são realizadas no mesmo local e

horário todos os dias e têm o objetivo de sincronizar o trabalho da equipa, identificar impedimentos e promover a colaboração.

O Scrum também utiliza “artefatos” para facilitar a transparência e o acompanhamento do progresso do projeto, os principais incluem: (a) *Backlog* do produto que é uma lista priorizada de todos os requisitos, funcionalidades e melhorias que devem ser feitas no produto. É de responsabilidade do *Product Owner* gerenciar e atualizar o *backlog* do produto. O *backlog* do *sprint* que é uma seleção de itens do *backlog* do produto que a equipa se compromete a concluir durante um *sprint* específico. O *backlog* do *sprint* é definido durante a reunião de planeamento do *sprint*.

(b) Quadro Scrum (*scrum board*) que é uma representação visual do trabalho em andamento. Geralmente é dividido em colunas que representam os estados dos itens, como "A fazer", "Em andamento" e "Concluído". Isso ajuda a equipa a acompanhar o progresso e identificar quais itens estão bloqueados ou prontos para serem trabalhados. No final de cada *sprint*, ocorre a revisão do *sprint*, onde a equipa demonstra o trabalho concluído e recebe *feedback* dos *stakeholders*. Em seguida, há uma retrospetiva do *sprint*, onde a equipa reflete sobre o processo, identifica melhorias e define ações para implementar no próximo *sprint*.

A Fig. 11 ilustra todos os “artefatos”, papéis-chave e o funcionamento da metodologia.

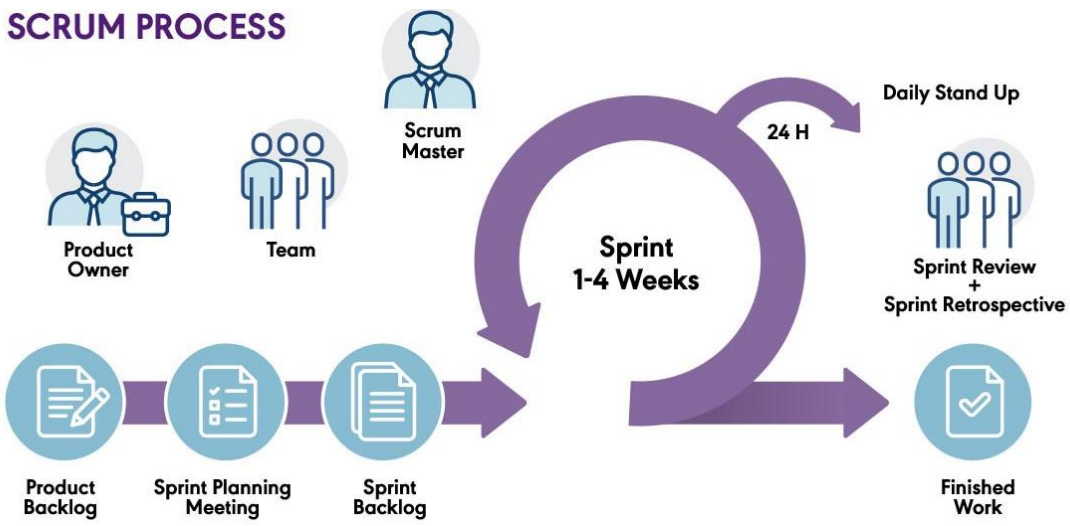

Figura 11 Etapas e artefatos do SCRUM, adaptado de Pm-partners (2021).

Para garantir o funcionamento, o Scrum também possui papéis-chave, como: (i) *Product Owner* que é o responsável por definir as necessidades do cliente, estabelecer prioridades e tomar decisões sobre o produto. O *Product Owner* representa os interesses dos *stakeholders* e trabalha em colaboração com a equipa Scrum. (ii) *Scrum Master* que é o responsável por

garantir que o Scrum seja compreendido e aplicado corretamente. O *Scrum Master* ajuda a equipa a remover impedimentos, facilita as reuniões e promove um ambiente de trabalho colaborativo. (iii) Equipa de Desenvolvimento que é o grupo de profissionais responsáveis por realizar o trabalho para entregar os itens do *backlog* do produto. A equipa é autogerenciada e multidisciplinar, trabalhando em conjunto para entregar o produto.

2.8 SUMÁRIO

O desenvolvimento da *web app* utilizará as melhores platicas de CI/CD, como *framework front-end* o Angular, *back-end* Spring Bott, como base de dados MySQL, como sistema de autenticação Keycloak, como *message broker* ActiveMQ. Como metodologia de gestão e desenvolvimento usará o Scrum.

Este sumário de tecnologias, *frameworks* e metodologias a aplicar não significa que outras tecnologia/*frameworks* não tenham sido usadas ao longo do estágio, sendo que, sempre que é introduzida uma noma tecnologia/*framework* a mesma é explicada sumariamente.

3 ESTADO DA ARTE DE APLICAÇÕES PARA A VISUALIZAÇÃO DE PARÂMETROS AMBIENTAIS

3.1 INTRODUÇÃO

Este capítulo está divido em três seções, a primeira reflete o estado da arte conciso de aplicações existentes no âmbito da visualização da previsão do tempo e visualização de parâmetros ambientais, a segunda seção foca *mockups* e a sua importância para o desenvolvimento de uma aplicação, neste caso *web app*. Na última seção é feita uma pequena discussão em forma de sumário de quais são as características principais que a *web app* deverá ter.

3.2 ESTADO DA ARTE

A previsão do tempo e condições ambientais desempenham um papel fundamental na vida cotidiana das pessoas, fornecendo informações valiosas, como por exemplo sobre as condições climáticas futuras. Com o avanço da tecnologia, aplicações que consomem dados meteorológicos de APIs tornaram-se populares, permitindo que os utilizadores acedam a informações atualizadas sobre o clima de forma conveniente.

O presente estado da arte esta dividido em: (i) APIs para a previsão do tempo, (ii) aplicações (web) para previsão do tempo, e (iii) desenvolvimentos/aplicações científicas para a previsão do tempo/parâmetros ambientais.

As aplicações de previsão do tempo dependem do acesso a dados meteorológicos precisos e atualizados. Para isso, muitas aplicações consomem informações de APIs de serviços meteorológicos reconhecidos, como o OpenWeatherMap (OpenWeather, 2023) ou Weatherbit (Weatherbit, 2023). Essas APIs fornecem dados como temperatura, humidade, velocidade do vento, precipitação, índice UV, entre outros, para uma ampla variedade de localidades em todo

o mundo. Estas APIs oferecem diferentes planos de preços que variam de acordo com o nível de acesso, recursos disponíveis e limites de uso. Cada uma delas possui opções gratuitas e pagas, com diferentes restrições e funcionalidades.

O site "O Tempo" (OTempo, 2022) identifica as 5 melhores aplicações para ver a previsão do tempo. Nessas estão incluídas o AccuWeather (AccuWeather, 2023), The Weather Channel (Weather, 2023) e Weather Underground (Wunderground, 2023). É importante ressaltar que a popularidade das aplicações de previsão do tempo pode variar de acordo com a região e as preferências individuais dos utilizadores. Essas três aplicações mencionados têm uma ampla base de utilizadores e são bem conhecidas em todo o mundo. Em especial o AccuWeather com mais de 100 milhões de downloads efetuados apenas para o sistema Android.

Na figura 12, vemos o exemplo do resultado de uma busca no AccuWeather, o acesso a esta *app* é 100% gratuito, mas existe a possibilidade de assinatura anual para evitar as publicidades. A busca foi feita com a cidade "Faro" no campo de pesquisa localizado no menu de navegação da *web app*, mostrando a previsão atual, com alguns dados extras, mas uma interface bem simples. A aplicação está disponível para Android e iOS, além de sua versão *web app*.

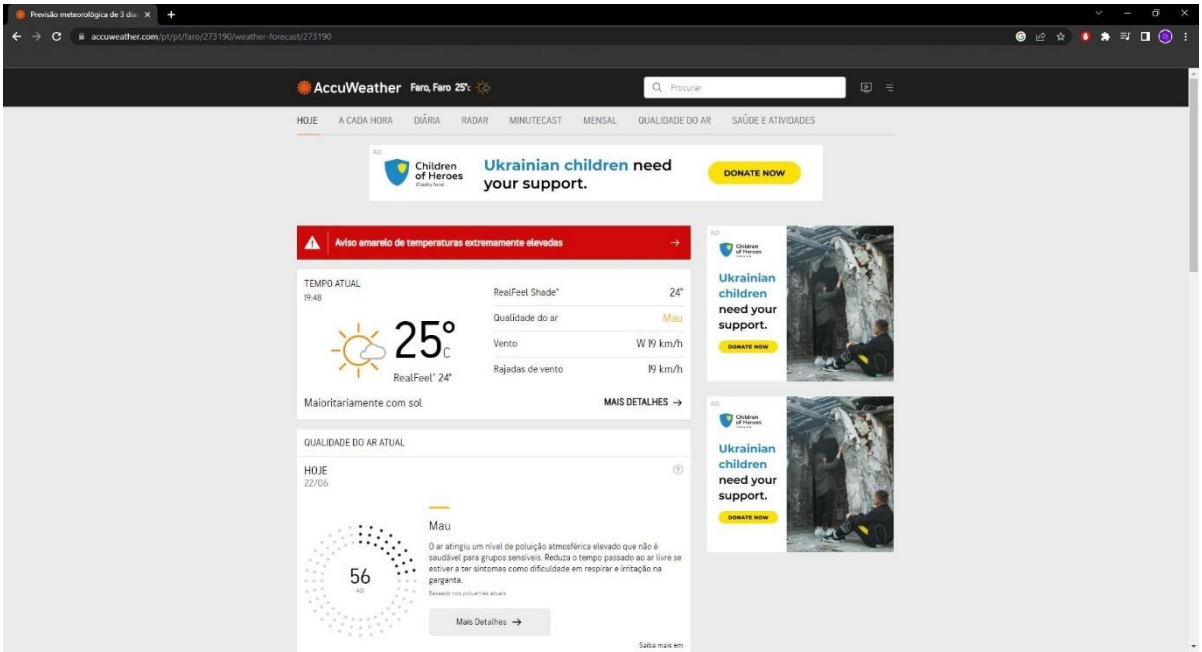

Figura 12 AccuWeather, previsão atual e futura, adaptado de AccuWeather (2023).

O Weather Underground (Wunderground, 2023) também apresenta muitos dados extras além das temperaturas atual, máxima e mínima. E tenta condensar o máximo de informação possível no ecrã de forma que não fique desorganizado, e acaba sendo informação excessiva para quem busca apenas dados simples como a temperatura. Outro ponto que deve ser levando

em consideração nessa *web app*, se rejeitado os *cookies*, a unidade padrão da temperatura é o Fahrenheit e não encontrei onde trocar para graus Celsius.

A figura 13, ilustra o exemplo do resultado de uma busca no Weather Underground, a busca foi feita com a cidade "Faro" no campo de pesquisa localizado no menu de navegação da *web app*, mostrando a previsão atual em Fahrenheit, com bastante dados na tela, até mesmo um "radar de nuvens". Apesar da interface apresentar muitos dados, segue na linha de uma interface simples.

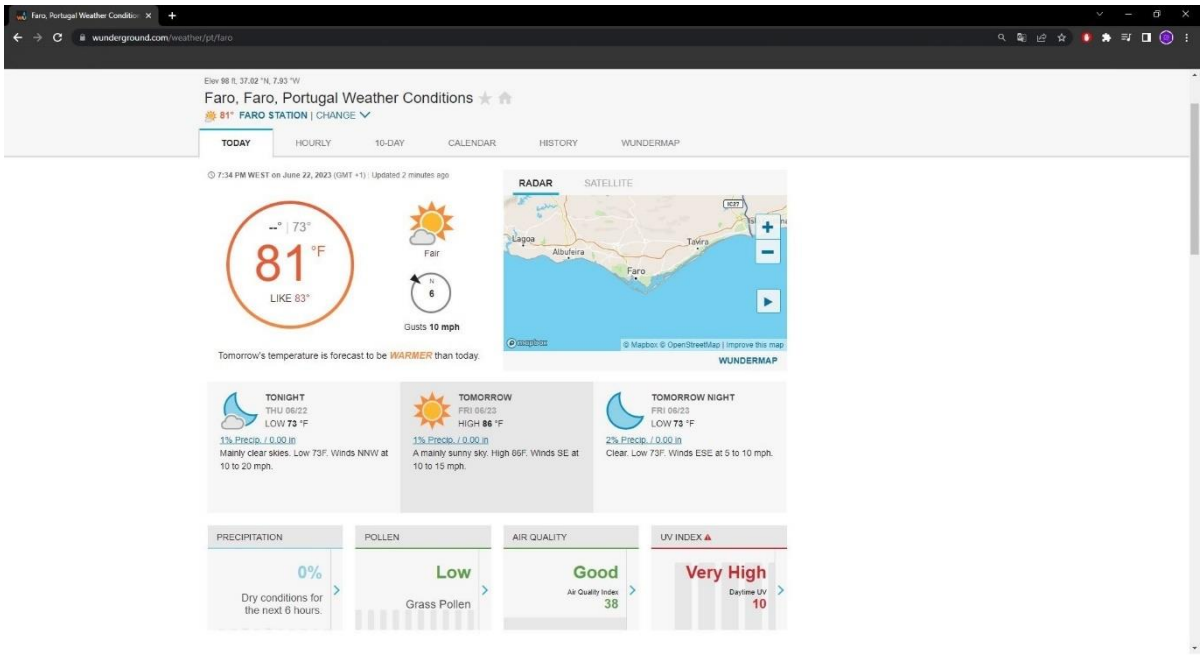

Figura 13 Weather Underground, previsão atual, adaptado de Wunderground (2023).

Por último temos o The Weather Channel (Weather, 2023), que é o meio termo entre as *web apps* apresentadas, com uma interface limpa, conseguiu organizar de maneira eficiente todos os dados extras na tela sem que ficasse de uma forma massiva ou acabasse desviando a atenção do utilizador.

A figura 14, ilustra o exemplo do resultado de uma busca no The Weather Channel, a busca foi feita com a cidade "Faro" no campo de pesquisa localizado no menu de navegação da *web app*. Também segue na linha de uma interface simples, porém muito bem organizada para disposição de dados extras sem interferir nos dados principais, que são as temperaturas.

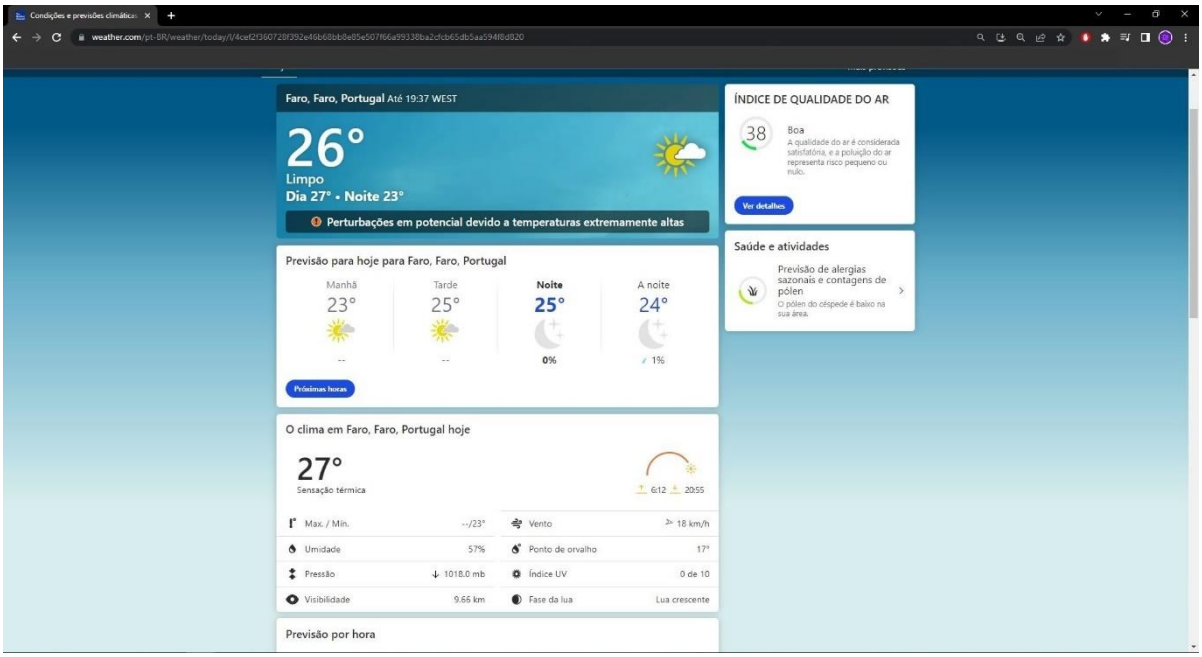

Figura 14 The Weather Channel Previsão atual, adaptado de Weather (2023).

Outros exemplos são os de Beeharry *et al.* (2019) que apresenta a implementação de um sistema de ponta a ponta, desde a aquisição de dados em tempo real até a previsão em tempo real. O sistema de armazenamento é implementado na plataforma de nuvem usando o serviço de banco de dados *Cloudant* NoSQL. O algoritmo de previsão é implementado em JAVA usando o serviço *Devops Insights* em conjunto com uma interface web e uma aplicação móvel Android. O sistema demonstra previsão rápida em tempo real quando implantado na plataforma de nuvem IBM, bem como recursos de armazenamento em tempo real.

Brendel *et al.* (2019) apresentam a aplicação *Stream Hydrology And Rainfall Knowledge System* (SHARKS), desenvolvido para expandir as plataformas existentes, fornecendo um conjunto de recursos de análise de dados exploratórios, incluindo a capacidade de calcular a profundidade total de precipitação registrada para qualquer período, interpolar o intervalo de recorrência anual médio para eventos de precipitação, realizar separações hidrográficas e calcular o volume de escoamento para qualquer período. Um estudo de caso apresentado pelos autores fornece uma descrição conceitual dos tipos de análise rápida que podem ser realizados usando a aplicação SHARKS. Kox *et al.* (2020) apresentam o projeto KARE-*citizen science* que estabeleceu uma rede meteorológica para leigos para construir conhecimento relevante para decisões sobre tempo e clima. Durante o verão de 2020, alunos adquiriram dados meteorológicos com equipamentos autônomos de baixo custo montados em estações meteorológicas e impactos climáticos relatados. O projeto incluiu um pré e pós-teste de alfabetização meteorológica e consciência das mudanças climáticas e sua expectativa do clima

local antes verão e perceção em retrospetiva. Rudenko *et al.* (2022) propõem uma revisão da literatura dos sistemas existentes que visualizam fenômenos meteorológicos em quatro (4D) e três dimensões (3D). Além disso, avaliam ocorrências meteorológicas para entender melhor a dinâmica associada a um fenômeno meteorológico e visualizar diferentes dados meteorológicos. Este artigo encontrou onze soluções existentes para visualização meteorológica 4D e fenômenos meteorológicos.

A visualização meteorológica 4D é uma forma de analisar, investigar e visualizar dados meteorológicos atmosféricos em quatro dimensões (espaço 3D + tempo). Ajuda a entender melhor a dinâmica associada aos fenômenos meteorológicos e visualizar diferentes dados climáticos.

Triska *et al.* (2023) apresentam uma aplicação baseado na web, WAVES, para visualização e análise de dados obtidos do sequenciamento de amostras ambientais. O painel fornece visualização em várias camadas de dados geográficos e genômicos. Ele permite exibir frequências de variantes de patógenos detetadas, bem como frequências de mutações individuais. A interface WAVES é facilmente personalizada por meio do arquivo de configuração editável e pode ser usado para diferentes tipos de patógenos e amostras ambientais.

3.3 MOCKUPS

Mockup é uma representação visual estática ou interativa de um produto, aplicação, website ou interface de utilizador (Clique, 2022; Axure, 2023). É uma forma de protótipo de baixa fidelidade que permite aos designers, desenvolvedores e *stakeholders* visualizarem como um projeto será apresentado antes da implementação final (Figma, 2023).

Assim, o *mockup* é a melhor forma de visualizar o design de maneira clara e realista, sem que ele entre em produção. É uma representação de um determinado projeto, que pode ser feita em escala ou em tamanho real. É aplicado na apresentação de ideias de maneira mais elaborada, possuindo o *design* bastante próximo ao projeto final. Dessa forma, a visualização do projeto final ocorre de forma bem mais facilitada, contribuindo para a aprovação da ideia ou para fazer as devidas alterações.

No *design* gráfico, o *mockup* é utilizado principalmente para demonstrar ao cliente como o projeto ficará depois de concluído. Por exemplo, se um designer está desenvolvendo uma *web app* para um cliente que deseja ver como o *web app* ficará após concluída, não é necessário realizar a codificação da mesma (Axure, 2023). Basta utilizar um *mockup* para demonstrá-lo

visualmente pelo computador e, dessa forma, facilitar a aprovação da ideia ou apontar as modificações necessárias.

Existem *mockups* para os mais variados tipos de projetos gráficos com ótimas informações e imagens de alta qualidade. *Mockups* de revistas, panfletos, *flyers*, cartões, convites, embalagens, crachás e etc. A forma mais comum utilizada pelos designers para fazer *mockups* provavelmente através do Adobe Photoshop (Adobe, 2023), no entanto existem uma enorme variedade de plataformas (incluindo grátis) para o desenvolvimento de *mockups* (Figma, 2023; Axure, 2023; Canva, 2023).

3.4 SUMÁRIO/DISCUSSÃO

Uma das principais características de uma aplicação da visualização do tempo e parâmetros ambientais é sua interface de utilizador clara e intuitiva, que permite aos utilizadores obter facilmente as informações desejadas. Para isso, muitas aplicações utilizam *layouts* simples e organizados, com gráficos, ícones e animações para representar visualmente as condições climáticas. Além disso, recursos de interação, como toques e gestos, são frequentemente incorporados para facilitar a navegação e a personalização.

Para oferecer uma experiência personalizada aos utilizadores, muitas aplicações de previsão do tempo incluem funcionalidades de *login/logout*. Essa opção permite aos utilizadores criarem contas individuais, onde podem guardar as suas preferências, como localidades favoritas, unidades de medida preferidas e configurações de notificações. A funcionalidade de *logout* também é essencial para que os usuários possam encerrar suas sessões e proteger suas informações pessoais.

Uma funcionalidade comum nas aplicações de previsão do tempo é a capacidade de adicionar localidades à lista de favoritos. Isso permite aos utilizadores terem um acesso rápido às localidades mais pesquisadas, sem a necessidade de pesquisar novamente a previsão do tempo toda vez. A lista de favoritos pode ser personalizada pelo utilizador, que pode adicionar, remover e reordenar as localidades conforme sua preferência.

Recentemente, observamos algumas tendências e inovações notáveis no campo das aplicações de previsão do tempo. O uso de técnicas de aprendizado de máquina e inteligência artificial para aprimorar a precisão das previsões tem ganhado destaque, e consequentemente muitas vezes a passagem para a visualização da informação em 4D (Rudenko *et al.*, 2022). Além disso, a integração com dispositivos IoT, como relógios inteligentes e assistentes

virtuais, permite que os utilizadores acederem a informações meteorológicas de forma mais conveniente e rápida.

Um dos principais pontos fortes das aplicações está em oferecer informações climáticas que se atualizam em tempo real, que informa até mesmo a sensação térmica do momento. O utilizador pode aceder os dados meteorológicos e ambientais previstos para os próximos 10 dias. Níveis de pólen, pó e outros existente no ambiente estão presentes no programa.

Sendo fundamental para este tipo de app a interface de utilizador (clara e intuitiva) é fundamental preparar um *mockup* da interface antes do início do desenvolvimento da *web app.* Na seção seguinte, depois de efetuada uma pequena introdução, vai ser apresentado sucintamente o *mockup* proposto e depois todos os detalhes de implementação da *web app*.

4 DESENVOLVIMENTO DA WEB APP

4.1 INTRODUÇÃO

Este capítulo reflete os passos de desenvolvimento da *web app*, relacionando os mesmos com os conceitos anteriormente apresentados. Como referido no capítulo 1, a *web app* foi desenvolvida com mais dois estagiários da Atos durante o processo de formação da empresa.

Como objetivo, a equipa tinha de desenvolver uma *web app* capaz de consumir uma API de dados meteorológicos, previamente definida pelo cliente, apresentar ao utilizador dados sobre o dia atual: temperatura atual, a máxima e a mínima do dia e probabilidade de chuva. A *web app* deverá também apresentar informação sobre os próximos seis dias: temperatura máxima e mínima, e probabilidade de chuva. Totalizando uma semana de dados analisados, de acordo com a localidade escolhida pelo utilizador, sendo obrigatório pelo menos, todos os municípios da Espanha (8.131 municípios ao total). Além de outras funcionalidades requeridas pelo cliente, como: *login/logout*, e a opção de adicionar uma localidade à lista de favoritos. Estes eram os itens mínimos necessários para a entrega do projeto, ficando ao critério da equipa o *design* da aplicação.

4.2 APPLICATION PROGRAMMING INTERFACE

A escolha de uma API é uma etapa extremamente importante para o desenvolvimento de uma *web app*, principalmente quando se utiliza uma API desenvolvida por terceiros. É ela que é responsável por listas todos os dados que vão ser utilizados na *web app*, então é de suma importância estudar bem as APIs disponíveis para cada finalidade e ter em mente suas limitações.

Neste projeto a API foi escolhida pelo cliente, era um fator obrigatório do desenvolvimento utilizar a API *el-tiempo* [https://www.el-tiempo.net/api] onde pode ser conferida sua documentação. Esta API retorna diversos tipos de dados e utiliza como formato de resposta o JSON que é um dos formatos mais populares atualmente devido ser legível por humanos e fácil de analisar o formato de troca de dados. O JSON é independente de linguagem de

programação pode ser usado com qualquer linguagem além de JavaScript. Os *endpoints* utilizados foram:

[A] https://www.el-tiempo.net/api/json/v2/provincias, que retorna uma lista de todas as províncias espanholas neste estilo:


```
// 20230618170835
// https://www.el-tiempo.net/api/json/v2/provincias
"origen": {
    "productor": "Agencia Estatal de Meteorología - AEMET. Gobierno de
España",
    "web": "https://www.aemet.es",
    "language": "es",
    "copyright": "© AEMET. Autorizado el uso de la información y su
reproducción citando a AEMET como autora de la misma.",
    "nota_legal": "https://www.aemet.es/es/nota_legal",
    "descripcion": "elTiempo.net muestra la información meteorológica
creada por © AEMET"
  },
  "title": "Lista de provincias",
  "provincias": [
    {
      "CODPROV": "01",
      "NOMBRE_PROVINCIA": "Araba/Álava",
      "CODAUTON": "16",
      "COMUNIDAD_CIUDAD_AUTONOMA": "País Vasco/Euskadi",
      "CAPITAL_PROVINCIA": "Vitoria-Gasteiz"
    },
    {
      "CODPROV": "02",
      "NOMBRE_PROVINCIA": "Albacete",
      "CODAUTON": "08",
      "COMUNIDAD_CIUDAD_AUTONOMA": "Castilla-La Mancha",
      "CAPITAL_PROVINCIA": "Albacete"
    },
…
    {
      "CODPROV": "52",
      "NOMBRE_PROVINCIA": "Melilla",
      "CODAUTON": "19",
      "COMUNIDAD_CIUDAD_AUTONOMA": "Ciudad Autónoma de Melilla",
      "CAPITAL_PROVINCIA": "Melilla"
    }
  ] ,
  "metadescripcion": "Provincias de España | Seleccionar una provincia
para ver la previsión meteorológica.",
  "keywords": "Provincias de España, Previsión meteorológica por
provincias, El tiempo en España, El tiempo por provincias",
  "breadcrumb": [
    {
      "name": "Provincias",
      "url": null,
      "title": "Provincias y Ciudades Autónomas"
    }
  ]
}
```

[B] https://www.el-tiempo.net/api/json/v2/provincias/[CODPROV]/municipios, retorna uma lista de todos os municípios da províncias escolhida, onde para determinar a província é necessário o CODPROV que pode ser obtido consumindo o (A). Como por exemplo, todos os municípios da província de Araba/Álava que tem CODPROV=1, gerando uma resposta neste estilo:


```
// 20230618174559
// https://www.el-tiempo.net/api/json/v2/provincias/01/municipios
{
  "origen": {
    "productor": "Agencia Estatal de Meteorología - AEMET. Gobierno de
España",
    "web": "https://www.aemet.es",
    "language": "es",
    "copyright": "© AEMET. Autorizado el uso de la información y su
reproducción citando a AEMET como autora de la misma.",
    "nota_legal": "https://www.aemet.es/es/nota_legal",
    "descripcion": "elTiempo.net muestra la información meteorológica
creada por © AEMET"
  },
  "title": "elTiempo.net | Municipios de Araba/Álava",
  "provincia": "Araba/Álava",
  "codprov": "01",
  "metadescripcion": "Lista de municipios de la provincia de Araba/Álava |
Seleccionar un muncipio",
  "keywords": "Lista de municipios, Previsión meteorológica para los
municipios de la provincia de Araba/Álava , El tiempo",
  "municipios": [
    {
      "CODIGOINE": "01001000000",
      "ID_REL": "1010014",
      "COD_GEO": "01010",
      "CODPROV": "01",
      "NOMBRE_PROVINCIA": "Araba/Álava",
      "NOMBRE": "Alegría-Dulantzi",
      "POBLACION_MUNI": 2925,
      "SUPERFICIE": 1994.5872,
      "PERIMETRO": 35069,
      "CODIGOINE_CAPITAL": "01001000101",
      "NOMBRE_CAPITAL": "Alegría-Dulantzi",
      "POBLACION_CAPITAL": "2815",
      "HOJA_MTN25": "0113-3",
      "LONGITUD_ETRS89_REGCAN95": -2.51243731,
      "LATITUD_ETRS89_REGCAN95": 42.83981158,
      "ORIGEN_COORD": "Mapa",
      "ALTITUD": 568,
      "ORIGEN_ALTITUD": "MDT5",
      "DISCREPANTE_INE": 0
    },
…
    {
      "CODIGOINE": "01902000000",
      "ID_REL": "1019020",
```

```
      "COD_GEO": "01305",
      "CODPROV": "01",
      "NOMBRE_PROVINCIA": "Araba/Álava",
      "NOMBRE": "Lantarón",
      "POBLACION_MUNI": 931,
      "SUPERFICIE": 6748.9609,
      "PERIMETRO": 63204,
      "CODIGOINE_CAPITAL": "01902000401",
      "NOMBRE_CAPITAL": "Comunión/Komunioi",
      "POBLACION_CAPITAL": "89",
      "HOJA_MTN25": "0137-4",
      "LONGITUD_ETRS89_REGCAN95": -2.96649719,
      "LATITUD_ETRS89_REGCAN95": 42.71657557,
      "ORIGEN_COORD": "Mapa",
      "ALTITUD": 482,
      "ORIGEN_ALTITUD": "MDT5",
      "DISCREPANTE_INE": 0
    }
  ],
  "breadcrumb": [
    {
      "name": "Provincias",
      "url": "/provincias",
      "title": "El tiempo | Lista de provincias"
    },
    {
      "name": "Araba/Álava",
      "url": "/provincias/01",
      "title": "El tiempo en la provincia de Araba/Álava"
    },
    {
      "name": "Municipios",
      "url": null,
      "title": "El tiempo | Lista de municipios de Araba/Álava"
    }
  ]
}
```


[C] https://www.el-tiempo.net/api/json/v2/provincias/[CODPROV]/municipios/[ID], terceiro e último *endpoint* utilizado na *web app*, retorna informações geográficas e meteorológicas de um município espanhol. Onde o ID são os cinco primeiros dígitos do CODIGOINE, que pode ser obtido consumindo o (B). Como por exemplo, o município Alegría-Dulantzi da província de Araba/Álava que tem CODPROV=01 e ID=01001 gerando uma resposta neste estilo:


```
// 20230618181101
// https://www.el-tiempo.net/api/json/v2/provincias/01/municipios/01001

{
  "origin": {
    "productor": "Agencia Estatal de Meteorología - AEMET. Gobierno de
España",
    "web": "https://www.aemet.es",
```

```
    "language": "es",
    "copyright": "© AEMET. Autorizado el uso de la información y su
reproducción citando a AEMET como autora de la misma.",
    "nota_legal": "https://www.aemet.es/es/nota_legal",
    "descripcion": "elTiempo.net muestra la información meteorológica
creada por © AEMET"
  },
  "title": "elTiempo.net | El tiempo en Alegría-Dulantzi (Araba/Álava)",
  "metadescripcion": "El tiempo en Alegría-Dulantzi (Araba/Álava)",
  "keywords": "Alegría-Dulantzi, Provincia de Álava, previsión
meteorológica, previsión del tiempo, España, viento, humedad,
precipitaciones, lluvia, estado del cielo",
  "municipio": {
    "CODIGOINE": "01001000000",
    "ID_REL": "1010014",
    "COD_GEO": "01010",
    "CODPROV": "01",
    "NOMBRE_PROVINCIA": "Araba/Álava",
    "NOMBRE": "Alegría-Dulantzi",
    "POBLACION_MUNI": 2925,
    "SUPERFICIE": 1994.5872,
    "PERIMETRO": 35069,
    "CODIGOINE_CAPITAL": "01001000101",
    "NOMBRE_CAPITAL": "Alegría-Dulantzi",
    "POBLACION_CAPITAL": "2815",
    "HOJA_MTN25": "0113-3",
    "LONGITUD_ETRS89_REGCAN95": -2.51243731,
    "LATITUD_ETRS89_REGCAN95": 42.83981158,
    "ORIGEN_COORD": "Mapa",
    "ALTITUD": 568,
    "ORIGEN_ALTITUD": "MDT5",
    "DISCREPANTE_INE": 0
  },
  "fecha": "2023-06-18",
  "stateSky": {
    "description": "Poco nuboso",
    "id": "12"
  },
  "temperatura_actual": "25",
  "temperaturas": {
    "max": "27",
    "min": "16"
  },
  "humedad": "58",
  "viento": "13",
  "precipitacion": "0",
  "lluvia": "45",
  "imagen": null,
  "pronostico": {
    "hoy": {
      "@attributes": {
        "fecha": "2023-06-18",
        "orto": "06:30",
        "ocaso": "21:51"
      },
      "estado_cielo": [
        "17",
```

```
        "17",
        "17",
        "17",
        "17",
        "12",
        "17",
        "17",
        "64",
        "23",
        "64",
        "64",
        "44",
        "17",
        "17n",
        "82n"
      ],
      "precipitacion": [
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0.2",
        "2",
        "0.5",
        "0.2",
        "1",
        "0",
        "0",
        "0"
      ],
      "prob_precipitacion": [
        "45",
        "100",
        "65"
      ],
      "prob_tormenta": [
        "45",
        "95",
        "65"
      ],
      ...
    }, …
```


Foram apresentados somente alguns dados meteorológicos do dia atual e cortadas os dados meteorológicos dos dias seguintes, pois a resposta da API tem 1.115 linhas, e esse trecho da resposta é somente para ilustrar como obtemos os dados. Para completa visualização da resposta, consultar o Apêndice A.

4.3 DESENVOLVIMENTO DO *MOCKUP* PARA A *WEB APP*

Após estudar a documentação da API, entender os dados recebidos, o próximo passo foi a elaboração dos *mockups*. Num projeto real, existe uma equipa especialista formada por *designers*, para fazer o protótipo e chegar o mais perto possível do que o cliente deseja, trazendo para realidade dos desenvolvedores. Mas como estamos no processo de formação, o protótipo seria desenvolvido e apresentado ao cliente por nós mesmos.

Por isso não foi utilizado nenhum software específico para o desenvolvimento do *mockup*, feito com HTML, CSS, JavaScript e auxílio do Bootstrap, que é um framework CSS utilizado em aplicações *front-end*, ou seja, na camada de interface com o utilizador para o desenvolvimento de aplicações adaptáveis à tela de qualquer dispositivo.

Começando pela página inicial da *web app*, a figura 15, apresenta o *mockup* da *homepage*.

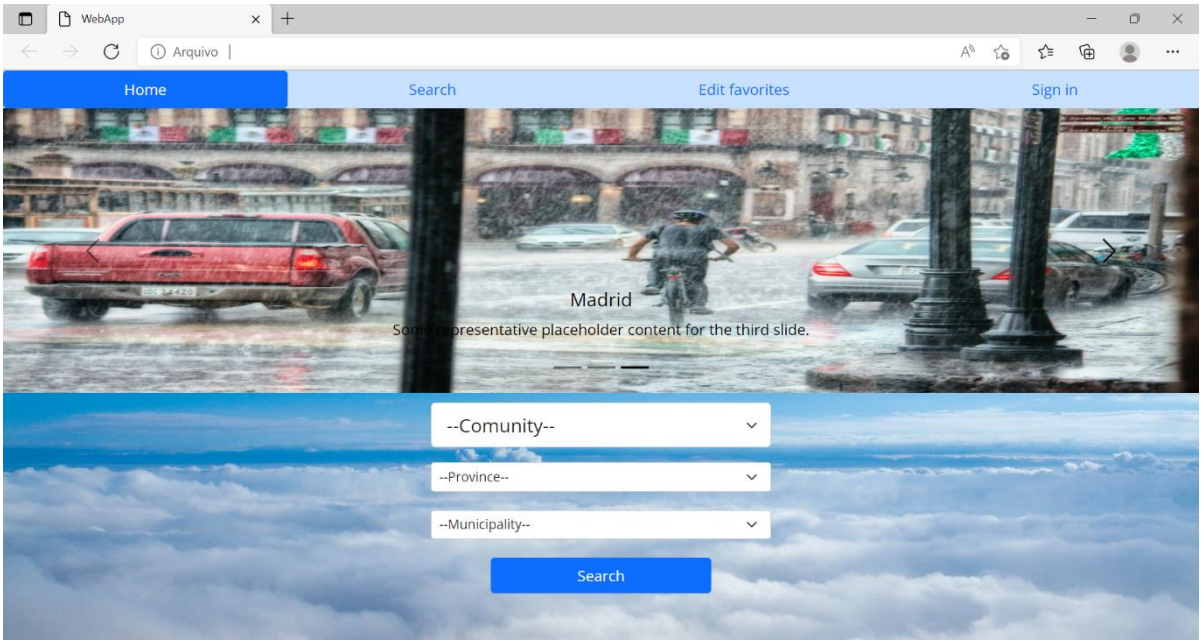

Figura 15 Mockup homepage.

Como é um *mockup*, as caixas de combinação, chamadas de *combobox*, ainda não eram funcionais, nem o botão *search*, assim como as imagens no carrossel eram meramente ilustrativas e os textos inseridos na imagem, chamados de *placeholders*. A ideia era apresentar na *homepage*, em carrossel, os municípios favoritos de cada utilizador, com uma imagem que representasse o tempo naquele instante naquela localidade favorita. Para identificar o município, cada imagem tem o nome do município como título e subtítulo a descrição do tempo atual. O carrossel não foi um elemento pedido pelo cliente, e a ideia foi criada pela equipa de desenvolvedores.

Também na *homepage*, o utilizador era capaz de filtrar o município desejado através de cada *combobox* representadas por comunidade, província e por fim, município. Assim após

escolhido o município desejado, clicar no botão *search* para que fossem exibidos os dados meteorológicos daquele município.

Por último, a *navbar*, que é comum para toda a *web app*, onde o utilizador pode navegar por todas as páginas da aplicação, note que, na figura 12 tem quatro opções na *navbar*, *homepage*, *search*, *edit favorites* e *sign in*, cada uma dessas páginas vão ser exemplificadas separadamente. A figura 16, apresenta o *mockup* da página *search*, que traz o resultado da pesquisa do município selecionado.

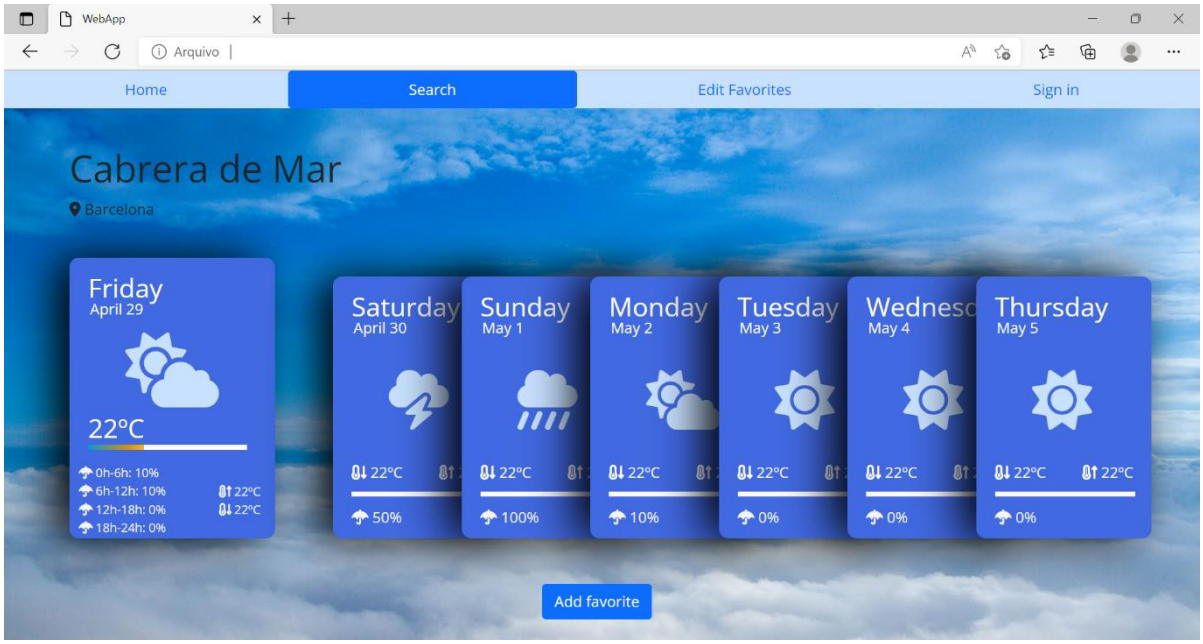

Figura 16 Mockup da página de resultado de pesquisa.

Após o utilizador efetuar o *login* na *web app*, ele tem a opção de adicionar municípios à sua lista de favoritos, de forma bem simples. Para realizar esta ação, basta clicar no botão "*Add favorite*" na página de resultados, *vide* figura 16. Ao aceder a página dos favoritos, o utilizador tem acesso a todos os municípios favoritos de uma forma rápida para consulta dos principais dados meteorológicos, e onde também é possível editar a lista, removendo as localidades que não deseja mais.

A figura 17, apresenta o *mockup* da página *Edit favorites*, que traz a lista de todos os municípios previamente marcados como favorito de um utilizador específico.

Por último as interfaces de *sign in* e *sign up*, que serão feitas com o Keycloak, onde preferimos manter a linha minimalista e aplicar a paleta de cores da *web app*. A figura 18, apresenta o *mockup* da página *sign in*, onde o utilizador que já é registado na *web app* pode fazer o *login*.

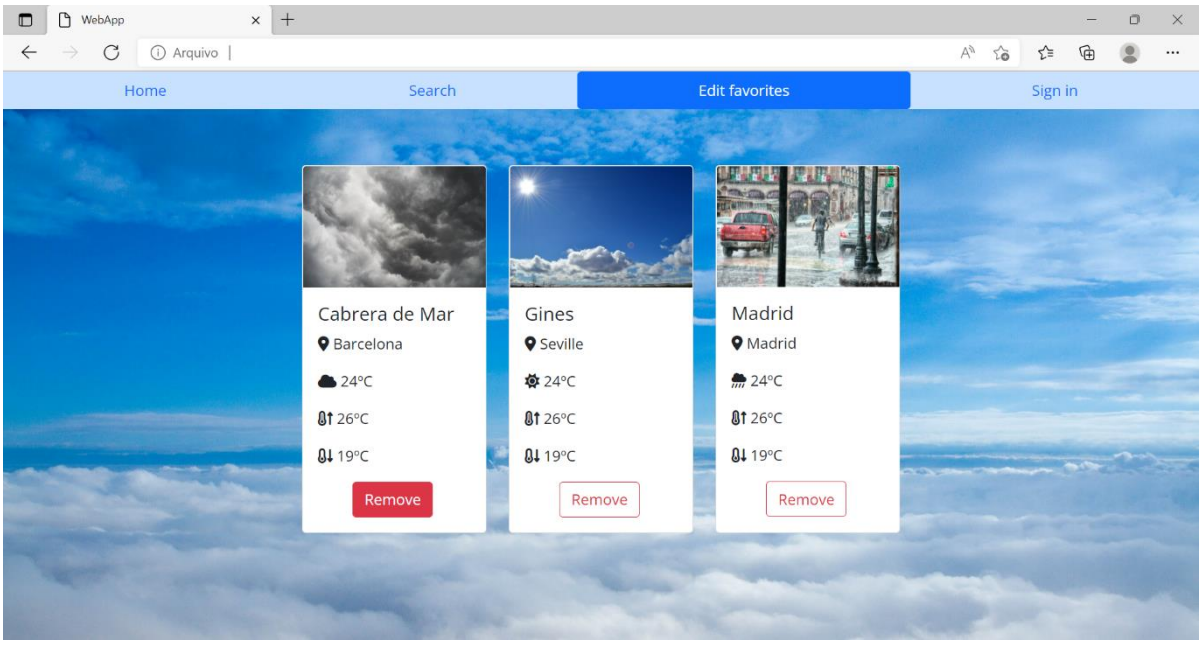

Figura 17 Mockup da página lista de favoritos.

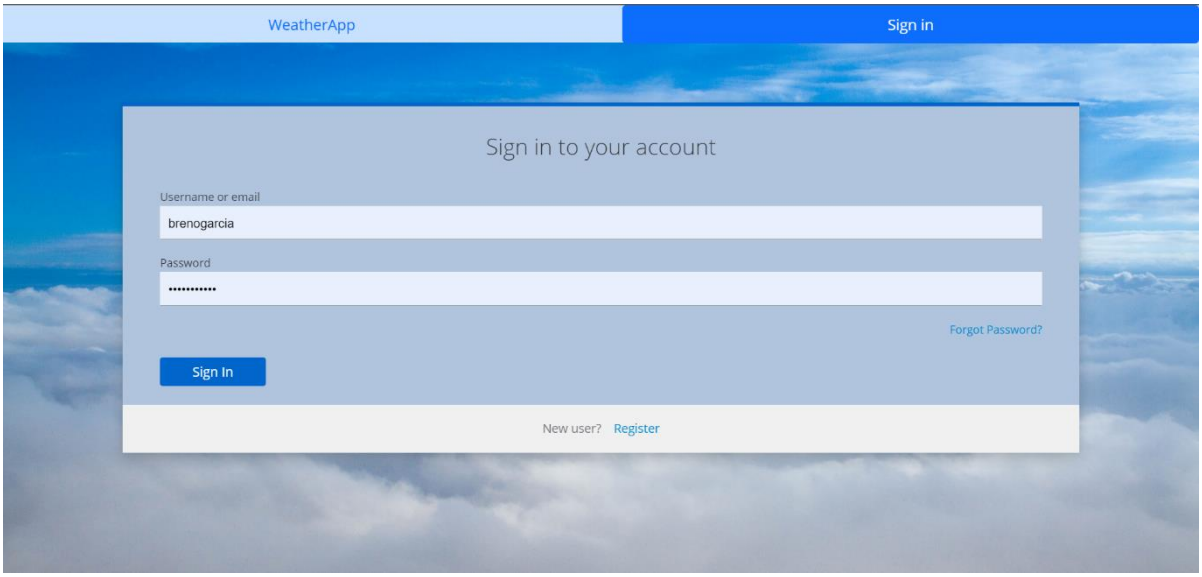

Figura 18 Mockup da página sign in.

Para novos utilizadores que desejam se cadastrar na *web app*, devem preencher um formulário com dados básicos para criação de uma conta na aplicação. A figura 19, apresenta o *mockup* da página *sign up*, onde é possível ver o formulário de inscrição da *web app*.

Figura 19 Mockup da página sign up.

Uma vez que tínhamos todos os *mockups* prontos, apresentamos ao PO em nossa próxima *daily meeting* e tivemos as interfaces aprovadas. Com as interfaces aprovadas, o próximo passo é colocar em prática o que foi mostrado, e criar as telas com o Angular. Como todas nossas telas apresentam dados consumidos da API, o *back-end* tem de ser feito em paralelo com o *front-end*

4.4 FRONT-END

Todo o *front-end* foi desenvolvido com Angular, dividimos o desenvolvimento da *web app* em 8 módulos e o primeiro foi o modulo de todo o *front-end*, que chamamos de UserInterface. Como mencionado na seção 2.3.1.1. Angular, a aplicação é dividida em componentes, para nosso projeto UserInterface, além do componente *app* padrão do Angular, criamos mais cinco componentes. Que são:

1. ***navbar***: Componente comum para toda a *web app* é a barra de navegação no topo da tela;
2. ***dropdown***: Componente só é usado na *homepage* e representa todos os três *combobox;*
3. ***search***: Componente é utilizado para mostrar o resultado dos dados meteorológicos do município em questão;
4. ***favorites***: Componente só é usado na página de favoritos e representa todos os municípios na lista de favoritos de um utilizador específico.

A seção seguinte apresenta o detalhe do projeto UserInterface, feito página a página.

4.4.1 HOMEPAGE

Seguindo a metodologia Scrum o primeiro *sprint* foi focado no desenvolvimento do UserInterface que começou pela *homepage* juntamente com a parte de *back-end* que foi o módulo *search*, apresentado na seção 4.5.3. O *sprint backlog* que normalmente é criado pelo PO neste projeto foi criado por nós mesmos (equipa de desenvolvedores), fizemos a divisão das tarefas para o primeiro sprint.

Para o plano de fundo da *homepage*, que é comum para toda a *web app*, foi escolhido uma imagem sem direitos de autor que pudesse ser usada em outros projetos, e que remetesse algo relacionado ao clima ou a natureza (essa imagem pode ser encontrada em: https://www.pexels.com/pt-br/foto/vista-aerea-de-nuvens-258149/).

O próximo elemento a ganhar vida foi a *navbar*. Usamos como *template* a *navbar* do Bootstrap disponível em: https://ng-bootstrap.github.io/#/components/nav/overview.

Como o carrossel era um elemento extra apresentado por nós no *mockup* e só estaria disponível para utilizadores que estivessem "logados" na *web app*, optamos por deixar por ser uma das últimas coisas a serem implementadas. E então passamos para a implementação do *combobox*.

O primeiro *combobox* traz a lista de todas as comunidades da Espanha obtidos consumindo o *endpoint* (A) apresentado na subseção 4.1.1, mas para isso devemos fazer uma chamada ao nosso servidor, e o servidor que deve consumir a API, fazer o *mapping* do JSON para uma entidade e retornar a entidade parametrizada para o cliente. Então fazemos uma chamada ao nosso servidor com o *path* '/communities' quando a *homepage* é carregada, os detalhes estão na subseção 4.1.4 *Back-end*:

```
this.http.get<community[]>("http://localhost:8100/communities")
    .subscribe((data: community[]) => {
      this.communityData = data;
    });
```

Onde comunityData é uma lista de *community* e a interface *community* é descrita por:

```
interface community {
  COMUNIDAD_CIUDAD_AUTONOMA:String;
  NOMBRE_PROVINCIA:String;
  CODPROV:String;
  CODAUTON:String;
}
```

O arranjo de *combox* foi construído de uma maneira que ao carregar a *homepage* apenas o primeiro *combox* para selecionar a comunidade estará habilitado, os outros dois desabilitados,

e somente quando selecionado uma comunidade, a lista de opções do segundo *combox* para selecionar a província será habilitada, assim como, somente após selecionar uma província o último *combox* para selecionar o município será habilitado, essa combinação é chamada de arranjo aninhado. A seguir vemos a construção do arranjo, começando pelo primeiro *combox*:

```
<select class="form-select form-select-sm" #communitySelect
(change)="communityChange(communitySelect.value)">
      <option value="" selected hidden>Community</option>
      <option *ngFor="let data of communityData" [value]="data.CODAUTON"
[selected]="data.CODAUTON == selectedCommunity">
{{data.COMUNIDAD_CIUDAD_AUTONOMA}}</option>
</select>
```

A primeira opção do *combobox* e já previamente selecionada, dentro da primeira *tag* HTML *option* é apenas um *placeholder* nota-se pelas propriedades atribuídas a *tag*, como: value="" selected *hidden*. A segunda *tag option* traz um *ngFor que é uma função padrão do Angular, um *loop* de iteração *for* bem conhecido em linguagem de programação, onde mostra como opção todas as comunidades dentro da lista de comunidades comunityData.

A função communityChange é chamada quando tem uma troca de valores no *combobox*, essa função que recebe como parâmetro o CODAUTON que representa um código único por comunidade, e após verificar se o código é valido, desabilitamos o *combobox* de seleção do município (para garantir que não está previamente carregado com municípios de outra comunidade e província), habilitamos o *combobox* de seleção da província (caso ainda não estivesse habilitado), após isso, filtramos a resposta obtida do servidor com todas as províncias pertencentes a comunidade com aquele CODAUTON previamente selecionado:

```
  communityChange(CODAUTON:String) {
    if(CODAUTON!=""){
      this.auton=CODAUTON;
      this.municipalityValid=false;
      this.municipalityData = [];
      this.provinceValid=true;
      this.filtered = this.provinceData.filter(provinceData =>
provinceData.CODAUTON == CODAUTON);
    }
  }
```

A lista de províncias também é obtida chamando nosso servidor, assim que a *homepage* é carregada pelo *path* '/provinces' e filtrada quando selecionado alguma comunidade, para mostrar como opções somente as províncias daquele município específico:

```
    this.http.get<community[]>("http://localhost:8100/provinces")
```

```
    .subscribe((data: community[]) => {
      this.provinceData = data;
  });
```

Também como resposta temos uma lista de *community* atribuídos dessa vez a uma variável que chamamos de provinceData. E exatamente com a mesma estrutura temos o código do segundo *combobox*.

```
<select class="form-select form-select-sm" #provinceSelect [disabled]=
"provinceValid==false" (change)="provinceChange(provinceSelect.value)">
      <option value="" selected hidden>Province</option>
      <option *ngFor="let data of filtered" [value]="data.CODPROV"
[selected]="data.CODPROV == selectedProvince">
{{data.NOMBRE_PROVINCIA}}</option>
</select>
```

Da mesma maneira, a função provinceChange é chamada quando há uma troca de valores no *combobox*, neste caso recebendo CODPROV como parâmetro, representando um código único por província, e após verificar se o código é valido, habilitamos o *combobox* de seleção do município, após isso, fazemos a chamada ao servidor ao *path* '/municipalities' passando o CODPROV previamente selecionado:

```
  provinceChange(CODPROV:String) {
    if(CODPROV!=""){
      this.municipalityValid=true;
      this.http.get<municipality[]>("http://localhost:8100/municipalities?cod
prov="+CODPROV)
      .subscribe((data: municipality[]) => {
        this.municipalityData = data;
      });
    }
  }
```

Onde municipalityData é uma lista de municipality. E municipality é a seguinte interface:

```
interface municipality {
  NOMBRE:String;
  NOMBRE_PROVINCIA:String;
  CODIGOINE:String;
  CODPROV:String;
}
```

Seguindo a mesma estrutura temos o código do terceiro *combobox*:

```
<select class="form-select form-select-sm" #municipalitySelect
[disabled]="municipalityValid==false"
(change)="municipalityChange(municipalitySelect.value)">
      <option value="" selected hidden>Municipality</option>
      <option *ngFor="let data of municipalityData" [value]="data.CODIGOINE">
      {{data.NOMBRE}}</option>
</select>
```

Da mesma maneira, a função municipalityChange é chamada quando há uma troca de valores no *combobox*, neste caso recebendo codigoINE como parâmetro, representando um código único por município, e após verificar se o código é valido, habilitamos o botão de busca dos dados meteorológicos daquele município previamente selecionado:

```
municipalityChange(codigoINE:String) {
    if(codigoINE!=""){
      this.codigoINE=codigoINE;
      this.couldSearch=true;
      };
  }
```

Uma vez habilitado o botão, assim que recebesse o evento click, executaria o comando:

```
this.router.navigate(['search_result',this.auton,this.codigoINE.slice(0, -
6)]);
```

Que navega para página de resultados, chamada de *search_result*, ilustrada na figura 16. Com a estrutura da *homepage* pronta faltando apenas o carrossel, que só é visível para utilizadores "logados", partimos para implementação da página *search_result*, pois já se aproximava o final do primeiro *sprint* e durante nossas *daily meetings* concordamos em adiar a implementação do carrossel da *homepage* por hora, visto que tomaria muito tempo.

4.4.2 SEARCH RESULT

O segundo *sprint* foi focado no desenvolvimento da página que mostra o resultado da pesquisa dos dados meteorológicos do município em questão incluindo também o *back-end* módulo WeatherData apresentado na seção 4.5.4. A equipa criou o *sprint backlog* e fizemos a divisão das tarefas. Essa página foi bastante trabalhosa porque no *mockup* apresentamos vários elementos extras que não tinham sido pedidos inicialmente pelo cliente, como ícones no *card* que representam o clima atual/previsto, uma barra em cada *card* que era preenchida percentualmente de acordo com a média de temperatura para o dia, quando o ocorria o evento *mouse over* (ponteiro do rato em cima do *card*).

O primeiro elemento a ser feito foi o carrossel de cartas, onde o primeiro *card* ficava separado dos demais e representava o dia atual, é o *card* que traz mais informação porque a previsão de chuva era dividida em quatro períodos de 6h cada. A estrutura do primeiro *card*:

```
<div class="cardOne"
(mouseover)="toggle(11,weatherData.tempAtualMax,weatherData.tempAtualMin)"
(mouseout)="toggle(11,weatherData.tempAtualMax,weatherData.tempAtualMin)">
      <div id="cardDate">
            <h3 class="title">{{GetWeekdayFromDate(weatherData.fecha)}}</h3>
            <h5 class="diaOne">{{GetMonthFromDate(weatherData.fecha)}}</h5>
      </div>
      <div id="cardIconOne">
            <img [src]="displayImg(weatherData.stateskyid)">
      </div>

      <div id="weatherDatasOne">
            <div id="maxMinOne">
            <p>
                  <i class="fa-solid fa-temperature-arrow-up"></i>
{{weatherData.tempAtualMax}}ºC
                  <br><i class="fa-solid fa-temperature-arrow-down"></i>
{{weatherData.tempAtualMin}}ºC
            </p>
            </div>
            <div id="probRainOne">
                  <div *ngFor="let data of weatherData.probPrecipitation;
let i = index">
                        <i class="fa-solid fa-umbrella"></i>
{{timearray[i]}} {{data}}%
                  </div>
            </div>
            <h3 class="tempOne">{{weatherData.temperatura_actual}}ºC</h3>
            <div class="barOne">
            <div class="emptybar"></div>
            <div id="11" class="filledbar" [ngStyle]="barStyle()"></div>
            </div>
      </div>
</div>
```

Os *card*s que representavam o resto da semana:

```
<div class="restOfWeek" >
      <div class="card" *ngFor="let dia of weatherData.proximos_dias; let i =
index" (mouseover)="toggle(i,dia.temperaturaMax,dia.temperaturaMin)"
(mouseout)="toggle(i,dia.temperaturaMax,dia.temperaturaMin)">
            <div id="cardDate">
                  <h3 class="title">{{GetWeekdayFromDate(dia.date)}}</h3>
                  <h5 class="dia">{{GetMonthFromDate(dia.date)}}</h5>
```

```
            </div>
            <div id="cardIcon">
              <img class="filter-white" [src]="displayImg(dia.estado_cielo)">
            </div>
            <div id="weatherDatas">
                  <div class="temp">
                        <i class="fa-solid fa-temperature-arrow-down"></i>
{{dia.temperaturaMin}}ºC <i id="iconMaxtemp" class="fa-solid fa-temperature-
arrow-up"></i> {{dia.temperaturaMax}}ºC
                        <br>
                        <br>

                  <i class="fa-solid fa-umbrella"></i>
{{dia.prob_precipitacion}}%
            </div>
          <div class="bar">
            <div class="emptybar"></div>
            <div id="{{i}}" class="filledbar" [ngStyle]="barStyle()"></div>
          </div>
        </div>
      </div>
</div>
```

Todos os ícones usados são grátis e podem ser usados ilimitadamente para qualquer projeto (Awesome, 2023). WeatherData era do tipo:

```
interface weather {
  fecha:String;
  temperatura_actual:String;
  municipio:municipio;
  stateskyid:String;
  tempAtualMax:String;
  tempAtualMin:String;
  probPrecipitation:String[];
  proximos_dias: ProximosDia[];
}
```

Onde ProximosDias:

```
interface ProximosDia {
  date: any;
  estado_cielo: any;
  prob_precipitacion: any;
  temperaturaMax: any;
  temperaturaMin: any;
}
```

Para finalizar a página falta o botão *Add favorite* e o nome do município e da província, que de acordo com o *Mockup* aparece na parte superior do carrossel de *cards*, com uma estrutura bem simples, temos:

```
<div class="data_card">
      <h1>{{weatherData.municipio.NOMBRE}}</h1>
      <p><i class="fa-solid fa-location-dot"></i>
{{weatherData.municipio.NOMBRE_PROVINCIA}}</p>
    </div>
```

Neste ponto ainda não tinha sido implementado a função *login/logout* na *web app*, então o botão *Add favorite* não fazia nada, mesmo assim decidimos colocar no ecrã, antes da primeira apresentação com o cliente, para acompanhar o progresso da *web app*. Após a reunião o cliente requisitou algumas mudanças pontuais, como o nome da província e município precisam estar mais destacados e com mais contraste, os ícones que apresentamos não eram o suficiente para descrever todos os estados que ele gostaria e que a API nos fornecia, e para o dia atual a probabilidade de precipitação dividida em quatro períodos de 6h devia ser oculta quando a hora atual da pesquisa já estivesse passado do *range* dos períodos. Além de questionar o porquê ter um botão *Add favorite* na tela que não faz nada, e cobrar o carrossel que apresentamos no *Mockup* e não tínhamos entregado ainda.

Para avançar com as próximas telas do UserInterface, e poder testar com fidelidade, a equipa sentiu a necessidade de avançar para a implementação do sistema de autenticação. Antes de iniciar o próximo *sprint* tivemos uma reunião com o cliente para apresentar o que tínhamos do UserInterface até o momento. Durante nossas *daily meetings* percebemos que a implementação do carrossel na *homepage* seria muito mais complexa do que imaginávamos e poderia comprometer o prazo estimado para entrega do projeto, por isso durante nossa reunião com o cliente apresentamos os pontos negativos da implementação do carrossel e como não foi um item requerido pelo cliente no projeto inicial, ele aceitou a implementação sem o carrossel na *homepage*.

4.4.3 LOGIN/SIGNIN

O terceiro *sprint* começou com a sensação de prazo apertado após a primeira apresentação parcial ao cliente, pois ele esperava mais na primeira apresentação. Nesse *sprint* então decidimos incluir além da implementação do Keycloak, a página *favorites* e suas funcionalidades. O primeiro passo para implementar o Keycloak é descarregar do repositório oficial do Keycloak no github: https://github.com/keycloak/keycloak, numa diretoria de sua

preferência. Quando terminado o *download* a pasta deve apresentar a estrutura apresentada na figura 20, que apresenta o projeto Keycloak e suas subpastas.

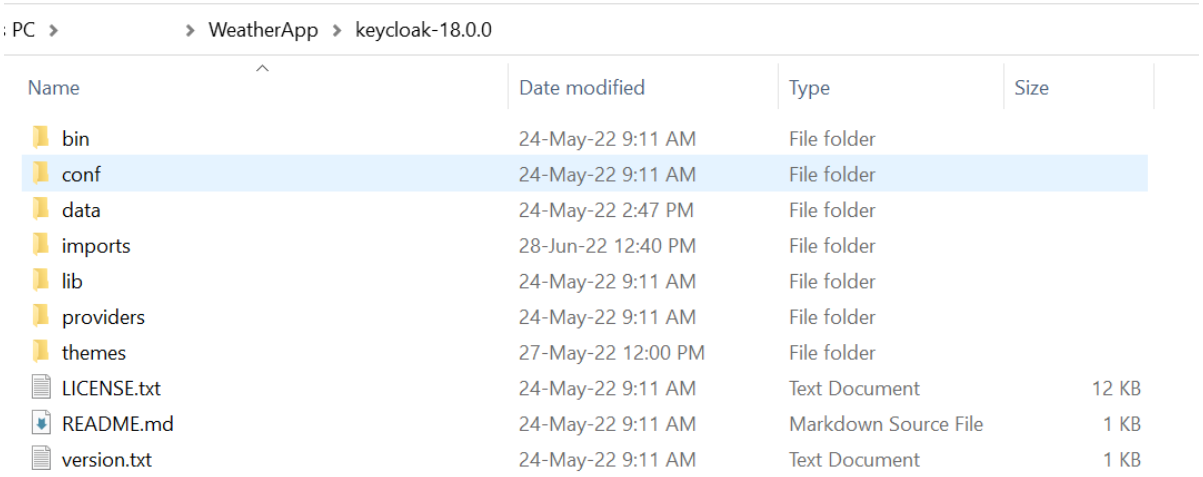

Figura 20 Projeto Keycloak, adaptado do projeto da web app.

Para executar o microserviço basta abrir o *command prompt* do Windows na pasta raiz (apresentada na figura 20) e executar o comando:

```
$ bin\kc.bat
```

Assim o Keycloak já vai estar disponível para ser acedido e iniciar sua configuração em localhost:8080 que é sua porta padrão. O primeiro acesso exige a criação de um utilizador para gerir a console da ferramenta. A figura 21 mostra a interface do Keycloak em seu primeiro acesso a: http://localhost:8080/auth

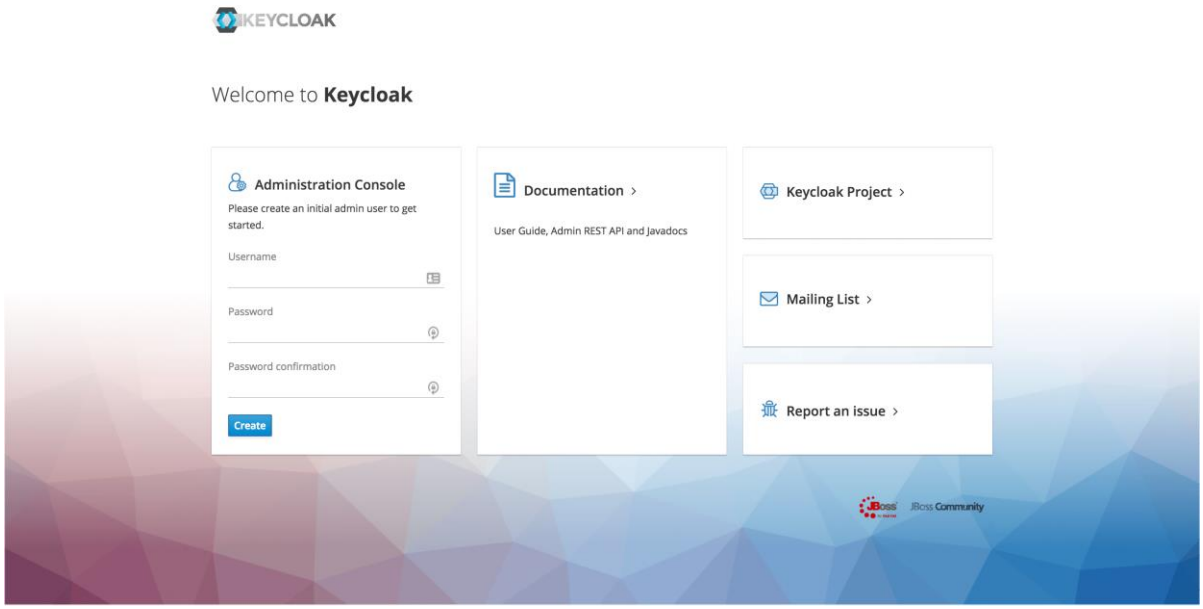

Figura 21 Keycloak primeiro acesso.

Após a criação do utilizador administrador do sistema, e feito o *login*, temos acesso ao console do administrador. A figura 22 apresenta o console do administrador após o primeiro *login*.

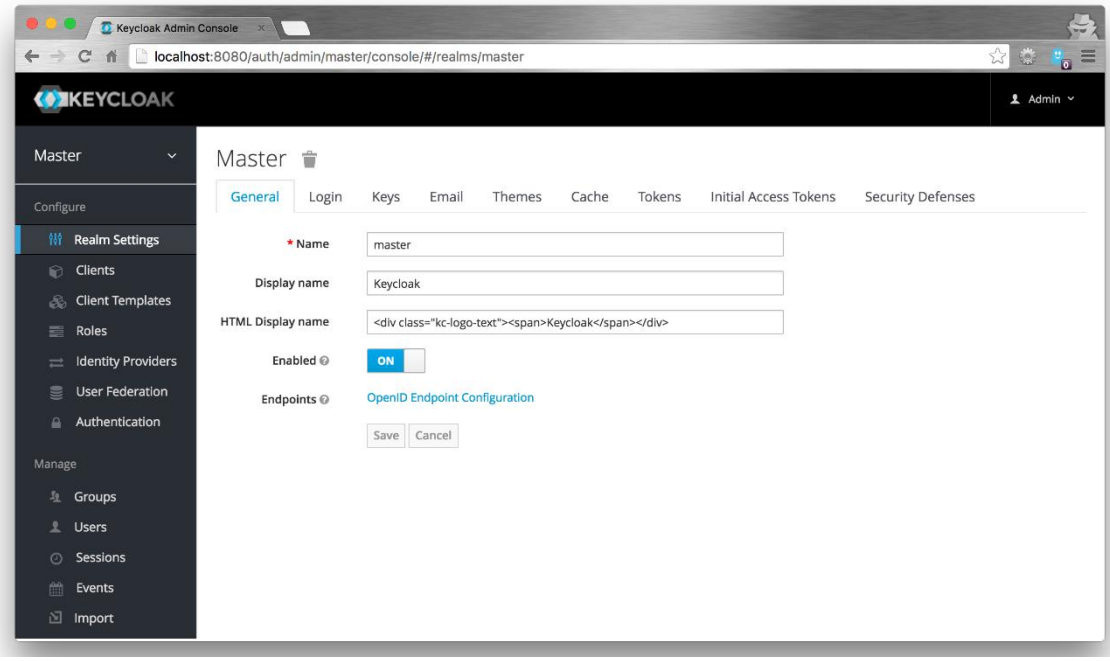

Figura 22 Keycloak primeiro login, adaptado de Köbler (2023).

O segundo passo é criar um *client* novo. A figura 23 apresenta a aba "*Clients*" do Keycloak. Para criação de um novo *cliente* basta clicar no botão "*Create*" ao topo direito da tabela, indicando o nome do novo cliente e a URL. Neste caso o nome do *cliente* escolhido foi WeatherApp e a URL foi onde estamos rodando o UserInterface: http://localhost:4200.

A criação do novo *cliente* já vem pré configurada e já pode ser utilizada, se o desenvolvedor do sistema precisar, pode ainda fazer novas configurações. A figura 24 apresenta a página de configuração do *client* após a criação de um novo *client*. Com as configurações básicas do Keycloak para que o torna-se funcional, agora vamos mudar a sua interface de *login*. A figura 25 apresenta a interface padrão do Keycloak sem nenhuma modificação.

Como dito aquando da apresentação do Keycloak, ele é muito fácil de customizar, v.d. também figura 25, que pode também ser modificado e criar temas a partir deste já pré-definido ou usar um tema base, onde não existe CSS, apenas as *tags* HTML e começar toda a estilização do zero.

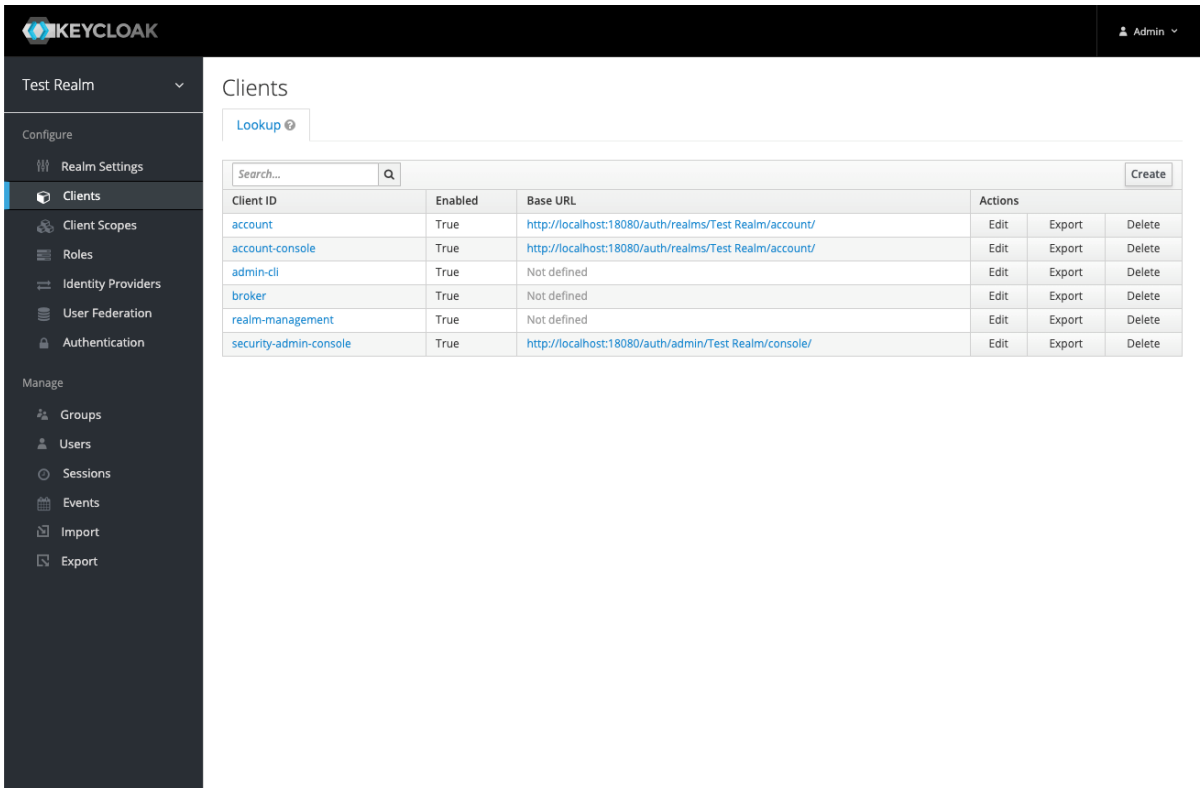

Figura 23 Keycloak clientes.

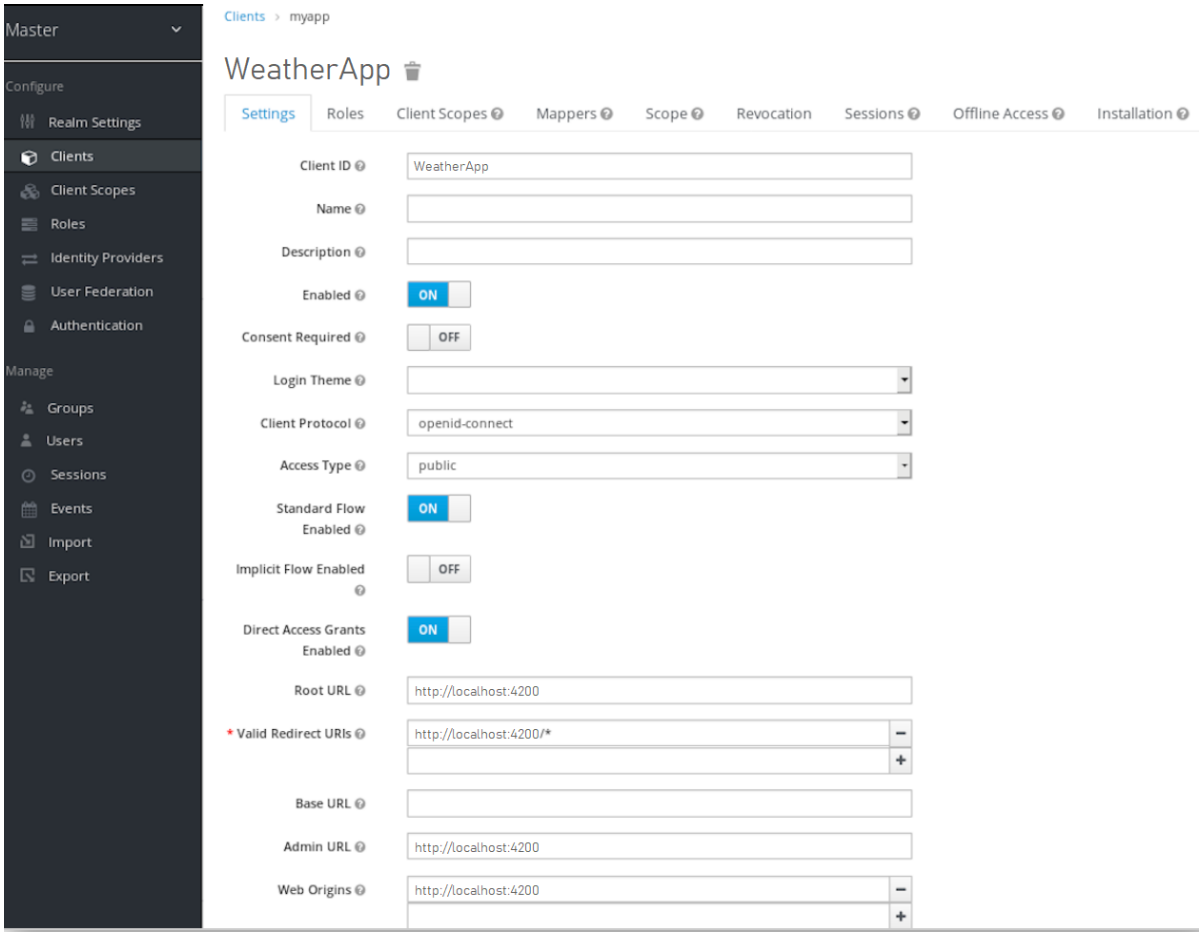

Figura 24 Keycloak client settings.

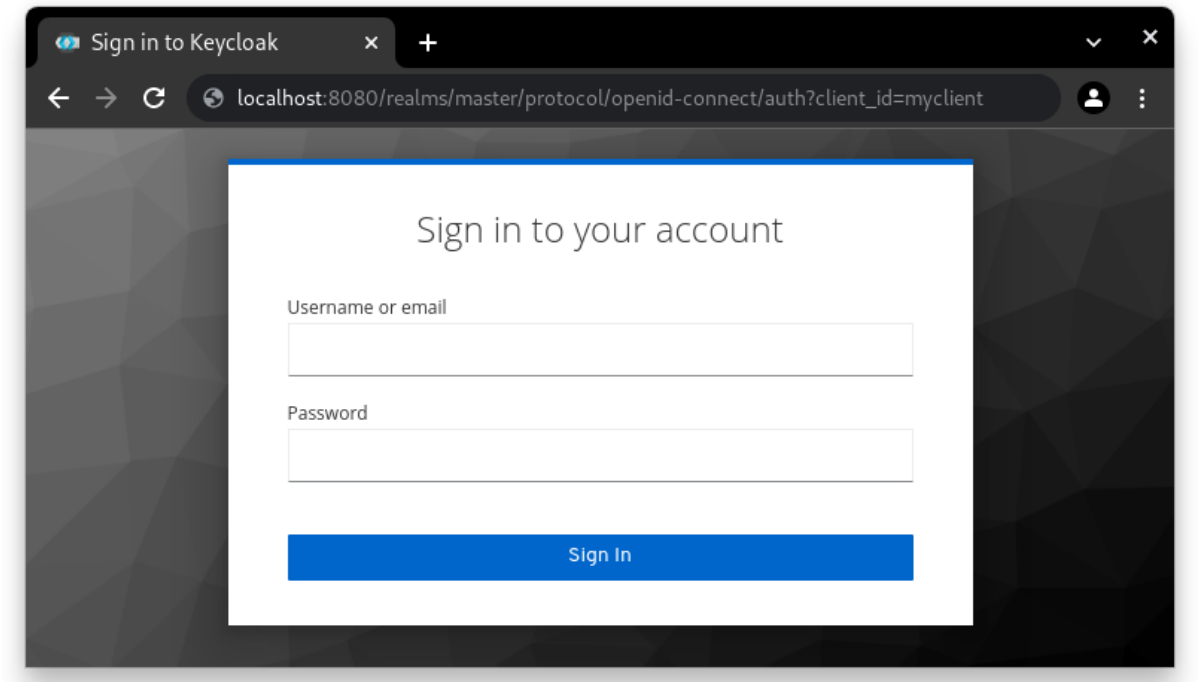

Figura 25 Interface padrão Keycloak, adaptado de Keycloak (2023).

A equipa começou a partir do tema padrão do Keycloak e criamos o tema WeatherAppTheme que pode ser conferido nos Apêndice A, para isso além de modificar os arquivos fontes para modificar o estilo da página, adicionamos a opção "*Forgot Password*" e "*Register*" a partir da configuração do Keycloak que é acessível em Realm Settings>Login. A figura 26 apresenta a configuração da página de *login* da *web app*.

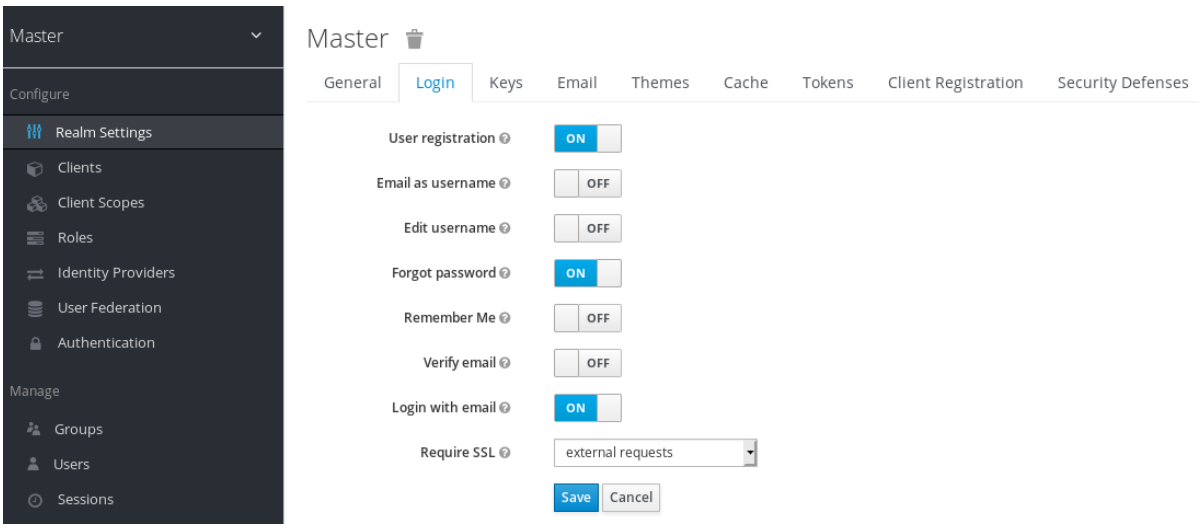

Figura 26 Realm settings, adaptado de Keycloak (2023).

Após a configuração e modificação do estilo criando o novo tema WeatherAppTheme, a página de *login* e de registro de novas contas ficou exatamente como apresentadas no *Mockup* (figura 18 e 19).

Com o Keycloak configurado e estilizado, é hora de fazer a integração com o Angular, para isso precisamos instalar o pacote angular-oauth2-oidc e para validar o *token* o pacote angular-oauth2-oidc-jwks (https://www.npmjs .com/package/angular-oauth2-oidc) com o gerenciador de pacotes npm, através do comando **npm i angular-oauth2-oidc –save** e **npm i angular-oauth2-oidc-jwks** após a instalação dos pacotes, adicionamos no *array* de *imports* do ficheiro app.module.ts as duas livrarias que vão ser usadas:

```
imports: [
      …
    HttpClientModule,
    OAuthModule.forRoot(),
      …
  ],
```

Depois é preciso um ficheiro de configuração, para isso criamos uma pasta keycloak_config dentro do diretório *app* onde criamos o ficheiro de configuração chamado de keycloak_weatherapp_config.ts que tem o seguinte conteúdo:

```
export const authCodeFlowConfig: AuthConfig = {
    // Url of the Identity Provider
    issuer: environment.keycloak.issuer,

    // URL of the SPA to redirect the user to after login
    redirectUri: environment.keycloak.redirectUri,

    // The SPA's id. The SPA is registerd with this id at the auth-server
    clientId: environment.keycloak.clientId,

    responseType: 'code',

    // set the scope for the permissions the client should request
    // The first four are defined by OIDC.
    // Important: Request offline_access to get a refresh token
    // The api scope is a usecase specific one
    scope: environment.keycloak.scope,

    requireHttps: false,
    showDebugInformation: false,
  };
```

Seguindo o *template* da documentação do pacote angular-oauth2-oidc, onde a constante *environment* é:

```
export const environment = {
  production: false,
  keycloak: {
```

```
    issuer: "http://localhost:8080/auth/realms/WeatherApp",
    redirectUri: "http://localhost:4200/",
    clientId: "angular_login",
    scope: "openid profile email offline_access"
  },
};
```

Agora a partir do componente navbar fazemos a injeção de dependência pelo construtor:

```
constructor(private oauthService: OAuthService) {}
```

Próximo passo é criar um método para configurar o oauthService, usando as funções padrões das livrarias e o arquivo de configuração do Keycloak apresentado acima:

```
configureSingleSignOn(){
    this.oauthService.configure(authCodeFlowConfig);
    this.oauthService.tokenValidationHandler = new JwksValidationHandler();
    this.oauthService.loadDiscoveryDocumentAndTryLogin();
  }
```

Que é chamado pelo método padrão do Angular ngOnInit(). Agora temos tudo o que é necessário para criar uma função de *login/logout* simples como:

```
  login(){
    this.oauthService.initCodeFlow();
  }

  logout(){
    this.oauthService.logOut();
  }
```

Onde a função *login* é chamada quando pressionado o botão *login* na navbar e o mesmo para a função *logout*. Criando assim o redirecionamento para a interface do Keycloak configurada por nós e após o *login*, o utilizador é redirecionado para *homepage* novamente.

4.4.4 FAVORITOS

Essa página *Favorites* foi desenvolvida também no terceiro *sprint* juntamente com sua parte *back-end*, o módulo *Favorites* apresentado na seção 4.5.5.

Uma página que também foi trabalhosa pois assim como no carrossel da *homepage*, aqui decidimos usar imagens em cada localidade salva como favorita, que representasse o clima atual. Tornando um trabalho extra (não solicitado).

Para representar as localidades salvas como favoritos, nos utilizamos uma grade de *cards* (Bootstrap, 2023) disponível em https://getbootstrap.com/docs/4.0/components/card/#using-

grid-markup, onde cada *card* tem uma imagem representando o clima atual da localidade, seguidos do nome do município e província, e pelas temperaturas atual, máxima e mínima. Cada *card* também apresenta um botão para remover a localidade dos favoritos.

Percebi-me que seria muito complicado carregar uma imagem representando o clima atual para cada município marcado como favorito, e para não tirar a imagem do *card* outra ideia seria uma imagem para localidade, mas cairíamos no problema de pesquisa e armazenamento das imagens, pois na Espanha existem 8.131 municípios, tornando inviável uma imagem para cada município, 50 províncias, já tornando viável, mas extremamente trabalhoso, e 17 comunidades, que seriam perfeitamente viável.

Sugerimos a mudança da imagem dos *cards* ao invés de representar o clima com imagens genéricas, usar imagens reais representando cada comunidade espanhola, e foi aceito pelo cliente. Assim temos a estrutura da grade de *cards*:

```
<div *ngIf="showFavorite && isLoaded==true" class="favorites">
      <div class="row row-cols-1 row-cols-md-4 g-4 favorite-grid">
            <div class="col" *ngFor="let data of filteredData;">
                  <div class="card">
                        <div>
                       <img [src]="displayImg(data.auton)" class="card-img-
top" style="height: 17vh;">
                        </div>

                        <div class="card-body" style="display: flex;justify-
content: space-between;align-items: center;">
                              <div class="cardtext">
                                    <div class="card-title">
{{data.municipality}}</div>
                                    <div class="card-title"><i class="fa-
solid fa-location-dot"></i> {{data.province}}</div>
                              </div>
                              <div style="display: flex;align-content:
space-between;flex-direction: column;">
                                    <i class="fa-solid fa-
temperature"></i>{{weatherData.temperatura_actual}}ºC
                                    <br><i class="fa-solid fa-temperature-
arrow-up"></i> {{weatherData.tempAtualMax}}ºC
                                    <br><i class="fa-solid fa-temperature-
arrow-down"></i> {{weatherData.tempAtualMin}}ºC

                              </div>
                        </div>
                  </div>
            </div>
      </div>
```

```
</div>
```

Onde a função displayImg() retorna o *path* para cada imagem, de acordo com o código auton, armazenadas no projeto. As imagens foram retiradas do google e são imagens de uso livre, sem direitos autorais:

```
displayImg(auton:any): String {
    return "/assets/imgs/communities/"+auton+".jpg";
  }
```

A variável filteredData é uma lista de *favorite*, como mostrado abaixo:

```
interface favorite {
      municipality:String;
      province:String;
      codigoine:String;
      auton:String;
}
```

4.5 BACK-END

Todo o *back-end* foi desenvolvido com Spring boot (Spring, 2023), como referido anteriormente, dividimos o desenvolvimento da *web app* em 8 módulos, o UserInterface e Keycloak já apresentados anteriormente, restando 6 módulos que serão apresentados nessa seção.

4.5.1 APPLICATION GATEWAY

O quarto *sprint* iniciou com o planejamento de implementar o *Application Gateway* e o *Registry*, criamos o *sprint backlog* e fizemos a separação das tarefas.

O *Application Gateway* (terceiro módulo) é a porta de ligação da rede interna com a rede externa, suas principais vantagens são adicionar uma camada de segurança, ter a praticidade de alterar as portas dos serviços sem alterar o *front-end*, além de fornecer suporte a algoritmos de balanceamento de carga (https://spring.io/projects/spring-cloud-gateway).

Seguindo a documentação do Spring Boot, criamos nossa classe *gateway* e o microserviço está configurado para executar na porta 8100:

```
@SpringBootApplication
public class ApplicationGatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApplicationGatewayApplication.class, args);
    }
```

```
    @Bean
    RouteLocator gateway(RouteLocatorBuilder rlb) {
        return rlb
                .routes()
                .route(routeSpec -> routeSpec
                        .path("/communities")
                        .uri("lb://search/"))
                .route(routeSpec -> routeSpec
                        .path("/provinces")
                        .uri("lb://search/"))
                .route(routeSpec -> routeSpec
                        .path("/municipalities")
                        .uri("lb://search/"))
                .route(routeSpec -> routeSpec
                        .path("/weather")
                        .uri("lb://weatherdata/"))
                .route(routeSpec -> routeSpec
                        .path("/favorites")
                        .uri("lb://favorites/"))
                .build();
    }

}
```

No .path() temos o *path* que o cliente irá requisitar, e no .uri() temos o *path* que nossa *gateway* fará o redirecionamento. Neste caso estamos redirecionando para o nosso balanceador de cargas com o nome de cada serviço (estes nomes são definidos no ficheiro de configuração de cada microserviço).

No ficheiro de configuração application.yml do *gateway* definimos as origens e métodos permitidos para acessar o *gateway:*

```
spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/**]':
            allowedOrigins: "http://localhost:4200"
            allowedHeaders: "*"
            allowedMethods:
            - GET
            - POST
            - DELETE
```

Onde porta 4200 é onde está a executar o UserInterface. Além de definir o URL do nosso balanceador de cargas que está a rodar na porta 8761:

```
eureka.client.serviceUrl.defaultZone=http://eureka-server:8761/eureka
```

4.5.2 REGISTRY

O quarto módulo foi o *Load Balancer* ou em português, Balanceador de Cargas, aqui chamado de *registry*, conseguimos manter a meta e implementar no quarto *sprint*. Seu principal objetivo é distribuir a carga de tráfego de entrada de maneira uniforme entre vários servidores ou instâncias de um serviço. Isso evita que um único servidor fique sobrecarregado e melhora o desempenho geral do sistema. Indispensável para projetos altamente escaláveis como uma *web app* de previsão do tempo, que poderia ser usada por toda a população espanhola por exemplo.

Para essa *web app* foi usado o Eureka pois já tem anotações do Spring boot para facilitar sua implementação. Para configurar o microserviço registry como um microserviço de balanceador de cargas, só é preciso a anotação @EnableEurekaSerber no todo da classe: https://spring.io/guides/gs/service-registration-and-discovery/:

```
@EnableEurekaServer
@SpringBootApplication
public class RegistryApplication {

    public static void main(String[] args) {
        SpringApplication.run(RegistryApplication.class, args);
    }

}
```

Procedeu-se com a definição as seguintes propriedades no ficheiro application.properties:

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
spring.application.name=registry
```

A primeira propriedade definida é a porta que irá rodar o Eureka, as duas seguintes propriedades só servem para o cliente, como estamos fazendo um servidor, podemos definir como *false* e por último é definido o nome do microserviço.

4.5.3 SEARCH

O quinto módulo modulo foi o microserviço chamado de *search*, é neste modulo onde fazemos alguma das chamadas necessárias para a API *el-tiempo*, referida anteriormente.

Utilizamos uma arquitetura em camadas para criação deste microserviço, mais especificamente três camadas:

(i) *Controller*: Responsável por receber as requisições do utilizador e por direcionar os dados para a camada de serviço apropriada;

(ii) *Service*: Contém a lógica de negócios da aplicação, processa as requisições recebidas do *Controller*, realiza operações de negócio, faz chamadas apropriadas para a camada de acesso a dados (quando existente) e retorna os resultados para o *Controller*.

(iii) *Model*: Foram definidas as classes para criação dos nossos *Data Transfer Object* (DTO[4]).

Assim, temos três *controllers* simples, um para município, um para província e outro para comunidade. Cada um dos *controllers* tem a anotação do spring boot @RestController, recebe uma requisição HTTP GET e faz o redirecionamento para a camada de serviço:

```
@RestController
public class CommunityController {

    @Autowired
    private CommunityService communityService;

    //get all communities
    @GetMapping(path = "/communities")
    public List<Community> getCommunities() {
        return communityService.getAllCommunities();
    }

}
```

Neste caso acima, executa o método getAllCommunities() da camada de serviço apresentada a seguir:

```
public class CommunityService {
```

[4] Os *Data Transfer Object*, usualmente designados por DTOs geralmente são objetos simples, contendo apenas campos públicos ou propriedades para armazenar os dados relevantes. Eles podem ser usados para transferir dados em um formato mais adequado para o consumo e exibição, além de reduzir a sobrecarga de transferir informações desnecessárias.

```
    //lista todas as communidades
    public List<Community> getAllCommunities() {
        String url = "https://www.el-tiempo.net/api/json/v2/provincias";
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<CommunityList> responseEntity =
restTemplate.getForEntity(url, CommunityList.class);
        List<Community> communityList;

        //Verifica se o Body está vazio
        CommunityList responseBody = responseEntity.getBody();
        if (responseBody != null) {
            communityList = responseBody.communities;
        }else {
            return new ArrayList<>();
        }

        //Remove duplicados
        List<Community> filteredCommunityList =
communityList.stream().collect(
            collectingAndThen(toCollection(() ->
            new TreeSet<>(comparing(Community::getCodauton))),
ArrayList::new));

        //Compara nome da comunidade para colocar em ordem alfabetica
        Comparator<Community> nameComparator = (Community obj1, Community
obj2) ->
            Normalizer.normalize(obj1.communityName, Normalizer.Form.NFD)
            .compareTo(Normalizer.normalize(obj2.communityName,
Normalizer.Form.NFD));

        //lista ordenada
        filteredCommunityList.sort(nameComparator);

        return filteredCommunityList;
    }

}
```

Onde a classe Community tem respectivos atributos:

```
    @JsonProperty("COMUNIDAD_CIUDAD_AUTONOMA")
    public String communityName;
    @JsonProperty("CODAUTON")
    public String codauton;
    @JsonProperty("NOMBRE_PROVINCIA")
    public String provinceName;
    @JsonProperty("CODPROV")
    public String codprov;
```

A anotação @JsonProperty é usada para controlar a serialização e desserialização de propriedades de objetos Java para JSON e vice-versa. A CommunityList é apenas uma lista de Community declarada desta forma List<Community>. No exemplo visto, um GET *request* em localhost:8100/communities retorna a lista de todas as comunidades da Espanha em ordem alfabética.

Seguindo essa mesma estrutura tempos o *path* /provinces listando todas as províncias e o *path* /municipalities recebendo como parâmetro no corpo da requisição o CODPROV listando todas os municípios dentro da província com o CODPROV específico.

4.5.4 WEATHER DATA

O sexto módulo é o WeatherData, seguindo rigorosamente a mesma arquitetura de camadas do módulo *search*, este módulo tem como objetivo listar todos os dados meteorológicos de um município previamente selecionado pelo utilizador. Para isso temos um GET *request* que recebe como parâmetro o codigoine:

```
@GetMapping(path = "/weather")
    public WeatherList getWeatherdata(@RequestParam String codigoine) {
        return weatherService.getWeatherData(codigoine);
    }
```

Assim, executa o método getWeatherData() dentro da camada de serviço também passando como parâmetro o codigoine. Como explicado no capítulo 4 a partir do codigoine obtém-se o codprov e o ID para de realizar o pedido à API:

```
public WeatherList getWeatherData(String codigoINE) {
        String codprov = codigoINE.substring(0, 2);
        String id = codigoINE.substring(0, 5);
        String url = "https://www.el-
tiempo.net/api/json/v2/provincias/"+codprov+"/municipios/"+id;
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<WeatherList> responseEntity =
restTemplate.getForEntity(url, WeatherList.class);

        return responseEntity.getBody();
    }
```

Onde a classe WeatherList tem os seguintes atributos necessários para representar um objeto que contém todos os dados necessários da previsão do tempo para aquele município em questão:

```
@JsonProperty("fecha")
private String diaAtual;
```

```
@JsonProperty("temperatura_actual")
private String temperaturaActual;

@JsonProperty("municipio")
private Municipality municipio;

@JsonProperty("proximos_dias")
private List<NextDays> proximosDias;

private String stateskyid;

@JsonProperty("stateSky")
private void stateSkyUnpack(Map<String,Object> stateSky) {
      this.stateskyid = (String)stateSky.get("id");
}

private String tempAtualMax;
private String tempAtualMin;

@JsonProperty("temperaturas")
private void temperaturasUnpack(Map<String,Object> temperaturas) {
        this.tempAtualMax = (String)temperaturas.get("max");
        this.tempAtualMin = (String)temperaturas.get("min");
}

private List<String> probPrecipitation;

@SuppressWarnings("unchecked")
@JsonProperty("pronostico")
private void precipitationUnpack(Map<String,Object> pronostico) {
        Map<String, Object> hoy = (Map<String,Object>)pronostico.get("hoy");
        this.probPrecipitation = (List<String>)hoy.get("prob_precipitacion");
}
```

Como pode ser observado, alguns dados são obtidos diretamente a partir da desserialização do JSON, outros precisam ser calculados a partir dos dados lidos.

4.5.5 FAVORITOS

O sétimo modulo é o *favorites*, também segue a arquitetura de camadas como nos módulos anteriores, mas com uma camada a mais, o *repository*. Porque este é o único módulo que tem acesso a BD para adicionar, remover e listar todos os municípios adicionados à lista de favoritos de um utilizador.

Assim, na camada *controller* temos os seguintes *paths* que respectivamente tem a seguinte função: (i) Apagar um município da lista de favoritos (ii) Adicionar um município na lista de

favoritos (iii) Listar todos os favoritos de um utilizador específico (iv) Verificar se um município especifico já está na lista de favoritos do utilizador:

```
    @DeleteMapping(path="/favorites")
    public @ResponseBody Map<String, Object> removeFavorite (@RequestParam
String sid, @RequestParam String codigoine) {
        return favoriteservice.removeFavorite(sid, codigoine);
    }

    @PostMapping(path = "/favorites")
    public @ResponseBody Map<String, Object> addFavorite(@RequestBody
FavoriteDTO favorite) {
        return favoriteservice.addFavorite(favorite);
    }


@GetMapping(path = "/favorites")
    public @ResponseBody List<FavoriteQuery> getFavorites(@RequestParam
String sid) {
        return favoriteservice.getFavorites(sid);
    }

    @GetMapping(path = "/isFavorite")
    public @ResponseBody Map<String, Object> isFavorite(@RequestParam String
sid, @RequestParam String codigoine) {
        return favoriteservice.isFavorite(sid, codigoine);
    }
```

Seguindo o padrão de camadas, executando os métodos na camada de serviço, como por exemplo o método addFavorite() que adiciona um novo favorito à lista de favoritos do utilizador. Para implementar esse método um dos requisitos era que a execução dos comandos na BD fosse assíncrona para que não sobrecarregasse nossa BD:

```
public Map<String, Object> addFavorite(FavoriteDTO favorite) {
        Map<String, Object> map = new HashMap<>();
        if(favoriteRepository.findBySidAndCodigoine(favorite.getSid(),favorit
e.getCodigoine()).isEmpty()) {
            Favorite persistent = new Favorite();

            persistent.setSid(favorite.getSid());
            persistent.setCodigoine(favorite.getCodigoine());
            persistent.setAuton(favorite.getAuton());
            persistent.setMunicipality(favorite.getMunicipality());
            persistent.setProvince(favorite.getProvince());

            try {
                ObjectMapper mapper = new ObjectMapper();
```

```
                String userAsJson = mapper.writeValueAsString(persistent);

                jmsTemplate.convertAndSend(queue, userAsJson);
                map.put(status, "added to queue");
            } catch (Exception e) {
                logger.error("Error", e);
                map.put(status, "error");
            }
        }else {
            map.put(status, "already exits");
        }
        return map;
    }
```

Que recebe um DTO do *front-end* contendo os dados do município em questão para ser adicionado à lista de favoritos. Executa a verificação para ver se o município já existe na lista, caso não exista cria um novo objeto do tipo *Favorite* que é a entidade que mapeia as colunas da tabela em um objeto Java. Após atribuir todos os dados do DTO ao novo objeto que será salvo na BD, é feita uma conversão dos dados para JSON e o objeto é enviado para fila do ActiveMQ utilizando uma função padrão do repositório JmsTemplate converstAndSend() que recebe como argumentos nossa fila criada no ActiveMQ (explicado na próxima seção) e o JSON. Tendo como entidade a classe *Favorite*:

```
@Entity
public class Favorite implements FavoriteQuery {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @Column
    private String sid;

    @Column
    private String codigoine;

    @Column
    private String auton;

    @Column
    private String municipality;

    @Column
    private String province;
```

Além das propriedades que os outros módulos também apresentam, como a configuração do Eureka e escolha de porta, como esse módulo é o único que acesso a BD e o ActiveMQ, é preciso também definir o acesso a BD com o link de acesso e as credenciais, a mesma configuração para acessar o ActiveMQ, tendo o ficheiro application.properties:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://mysqldb:3306/db
spring.datasource.username=root
spring.datasource.password=ThePassword
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.application.name=favorites
server.port=8082

spring.activemq.broker-url=tcp://activemq:61616
spring.activemq.user=admin
spring.activemq.password=admin

eureka.client.serviceUrl.defaultZone=http://eureka-server:8761/eureka
```

4.5.6 ACTIVEMQ

O quinto e último *sprint* ficou para implementação do último módulo que é ActiveMQ e testes da *web app*. No site oficial da apache pode ser encontrado uma documentação bem detalhada sobre os primeiros passos no ActiveMQ: https://activemq.apache.org/version-5-getting-started.html.

O processo de configuração muito simples, uma vez o projeto descarregado em um diretório de sua preferência, abrindo o *prompt* de comando do Windows na pasta raiz do ActiveMQ basta executar o comando **bin\activemq** para que o microserviço se inicie. Assim como o Keycloak o ActiveMQ tem um console de administrador que pode ser acessado por padrão em localhost:8161/admin com o *login* e senha iguais a **admin**. Com isso o *broker* já está disponível.

A figura 27 apresenta o console administrador do ActiveMQ na página Queues, onde é possível gerenciar as filas criadas ou criar novas. Para criar uma nova fila basta escolher um nome e clicar no botão *Create*.

4.6 APRESENTAÇÃO DA *WEB APP* AO CLIENTE

Ao fim do quinto *sprint* e integração de todos os módulos concluímos o desenvolvimento da *web app* e tivemos que apresentar o resultado final ao nosso cliente. Com exceção do

carrossel na *homepage* todas as outras páginas ficaram exatamente iguais às telas apresentadas no *mockup*.

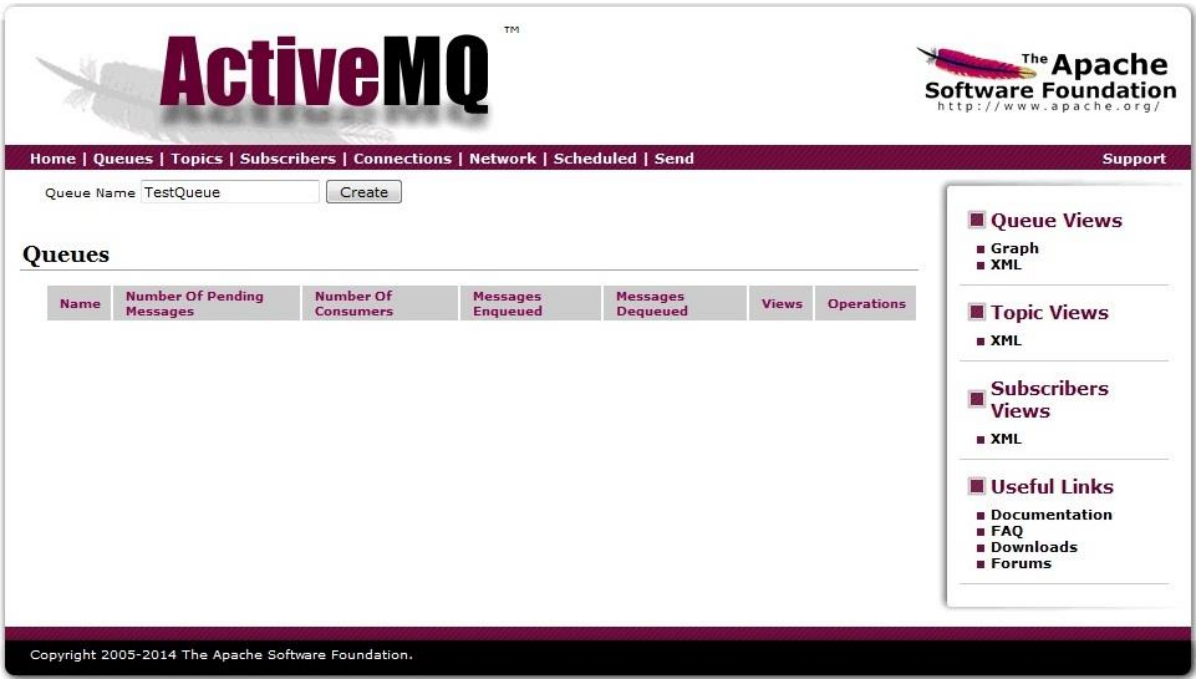

Figura 27 ActiveMQ página queue.

Durante a apresentação, mesmo sendo fiel ao *mockup* o cliente ainda pede algumas modificações pois não ficou satisfeito com o resultado ou acabou sugerindo novas implementações que otimizariam o projeto, como:

(i) Eliminar o botão *Search* na *homepage* e assim que o usuário selecionasse um município no último *combobox* já seria redirecionado para a página *Search result*;

(ii) Na página *Search result*, remover do *card* as probabilidades de chuva dos períodos que já passaram no dia (ex.: aceder a web às 13h, remover os dois primeiros intervalos, 0h~6h e 6h~12h) porque já não seria mais útil. Aumentar a quantidade de ícones que usávamos para representar mais estados do clima. Mudar a maneira que estávamos exibindo o nome do município e da província pois estava com pouco contraste com o *background*. E adicionar um botão na página *Search result* que pudesse redirecionar para a *homepage* sem perder a seleção da comunidade e da província para que o utilizador pudesse de maneira rápida apenas selecionar um novo município para ver a previsão do tempo;

(iii) Na página *favorites* ao invés de carregar todos os favoritos de uma vez que poderia trazer problemas de otimização à *web app* o cliente sugeriu que organizássemos por províncias a lista de favoritos e em vez de trazer todas as informações no *card*, adicionar um botão para redirecionar para a página *Search result* com as informações

meteorológicas do município em questão. Além de não ter gostado das imagens que selecionamos para representar o clima do município nos *cards* então sugerimos uma imagem que representasse a comunidade em que pertence o município em questão.

4.7 WEB APP 2.0

De forma a solucionar o que o cliente solicitou, *Updates Homepage*, tivemos mais um *sprint* extra com uma duração flexível, o fim seria quando terminássemos todas as melhorias requisitadas pelo cliente. Começamos as modificações na *homepage* que foi uma das mais rápidas onde tivemos que adicionar no constructor() a seguinte linha:

```
this.selectedMunicipality=this.route.snapshot.paramMap.get('ine');
```

Para que pudesse receber o CODIGOINE da página *Search result* e saber qual município foi previamente selecionado. E também modificar o método municipalityChange():

```
  municipalityChange(codigoINE:String) {
    if(codigoINE!=""){
      this.codigoINE=codigoINE;
      this.router.navigate(['search_result',this.auton,this.codigoINE.slice(0
, -6)]);
      };
  }
```

Agora ao invés de habilitar o botão de *Search* ao selecionar um município o utilizador já é redirecionado para a página *Search results*. A figura 28 apresenta o resultado final da *homepage.*

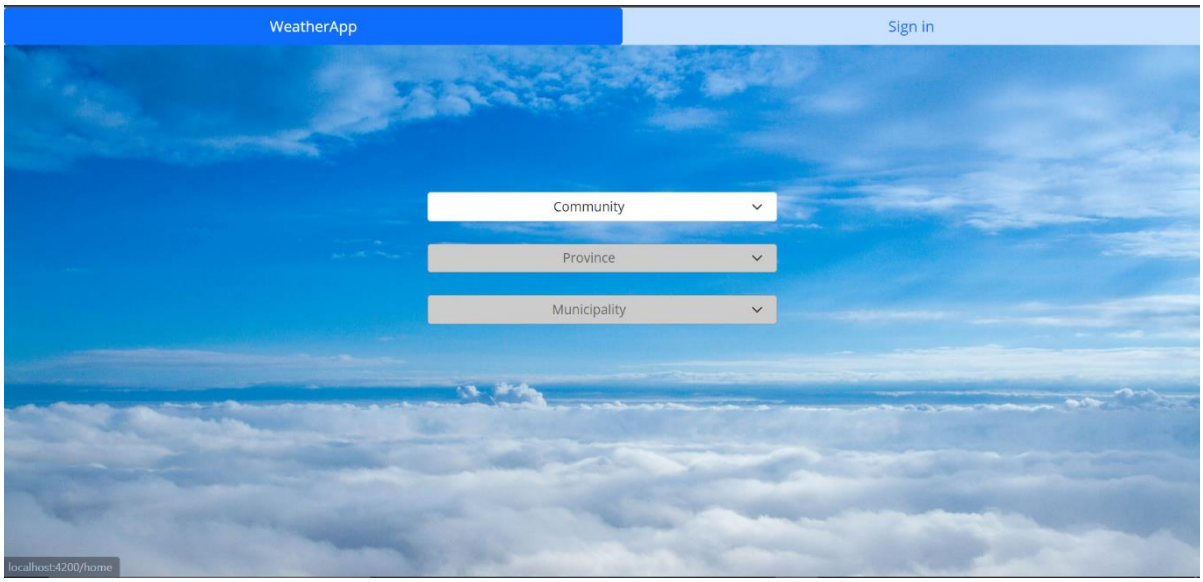

Figura 28 Homepage resultado final.

4.7.1 UPDATES SEARCH RESULT

Os *updates* foram um pouco mais trabalhosos nessa página, começando pelo mais simples que era modificar a maneira que exibíamos o nome do município e da província selecionados. Essa modificação de estilo é puro CSS então alteramos a classe da div que mostrava os nomes:

```
.data_card{
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0, 0.4); /* Black w/opacity/see-through */
  color: white;
  font-weight: bold;
  border: 3px solid #f1f1f1;
  position: relative;
  top: 80px;
  left: 50%;
  transform: translate(-50%, -50%);
  z-index: 2;
  width: 50%;
  height: 130px;
  padding: 20px;
  text-align: center;
}
```

Para os ícones, conseguimos os ícones antigos da *el-tiempo* que geria a API que utilizamos para conseguir os dados meteorológicos, ao todo são 51 ícones representando 21 estados diferentes. O maior trabalho foi renomear todos os ícones para o código do estado que a API retorna, para exibir o ícone de uma forma mais simples:

```
        displayImg(id:any): String {
    return "/assets/icons/png/"+id+".png";
  }
```

Onde o id é o "estado_cielo" retornado pela API. Para não exibir as probabilidades de chuva dos intervalos que já passaram, isto é, não incluem a hora atual da consulta, voltamos a estudar a API e percebemos que o *array* "prob_precipitacion" devolvido pela API já elimina os intervalos que já passaram. Então adicionamos esse trecho de código dentro da função padrão do Angular ngOnInit():

```
let time = this.now.getHours()
let count = 0;
this.timenumarray.forEach(function(value) {
      if(time<value){
            count++;
      }
});
this.timearray = this.timearray.slice(-count);
```

```
this.weatherData.probPrecipitation =
this.weatherData.probPrecipitation.slice(-count);
```

Onde,

```
timearray:String[]=['00h-06h:', '06h-12h:', '12h-18h:', '18h-24h:'];
timenumarray:number[]=[6, 12, 18, 24];
now:Date = new Date();
```

Neste trecho de código basicamente ao carregar a página obtém a hora atual, e faz uma varredura dentro do *array* timenumarray e verifica se a hora atual é menor do que os valores do *array* adicionando +1 a um contador para as vezes que isto acontecer. Assim eliminamos do timearray os intervalos que já passaram, se houver. O último update desta página é o botão para retornar a *homepage:*

```
<input id="backHomeButton" (click)="backHome()" type="image"
src="/assets/imgs/backArrow.png">
```

Que executa a função backHome() em um *click event* passando como parâmetro o CODIGOINE:

```
backHome(){
    this.router.navigate(['home',{ine: this.codigoINE}]);
  }
```

A figura 29 apresenta o resultado final da *search result.*

Figura 29 Search result resultado final.

4.7.2 UPDATES FAVORITES

Começamos modificando os *cards* dos favoritos removendo os dados meteorológicos, adicionando o botão para redirecionar para a página *search result* e ver mais informações:

```
<div *ngIf="showFavorite && isLoaded==true" class="favorites">
  <div class="row row-cols-1 row-cols-md-4 g-4 favorite-grid">
    <div class="col" *ngFor="let data of filteredData;">
      <div class="card">
        <div>
          <img [src]="displayImg(data.auton)" class="card-img-top"
style="height: 17vh;" alt="la sagrada familia">
        </div>
        <div class="card-body" style="display: flex;justify-content: space-
between;align-items: center;">
          <div class="cardtext">
            <div class="card-title">{{data.municipality}}</div>
            <div class="card-title"><i class="fa-solid fa-location-dot"></i>
{{data.province}}</div>
          </div>
          <div style="display: flex;align-content: space-between;flex-
direction: column;">
            <div
(click)="redirect_search_result(data.codigoine,data.auton)"><i class="fa-
solid fa-magnifying-glass btn btn-outline-primary"></i></div>
            <div (click)="delete(data.codigoine,data)" style="padding-top:
10px;"><i class="fa-solid fa-trash-can btn btn-outline-danger"></i></div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Depois implementamos um *combobox* para filtrar os municípios favoritos por localidade e ainda adicionamos uma mensagem que fica visível somente quando não há favoritos, para indicar que a lista está vazia, caso alguém acesse a página.

```
<div *ngIf="!showFavorite" class="favorites-empty">
  {{messageFavorite}}
</div>

<div *ngIf="showFavorite && isLoaded==true" class="dropdown">
  <span style="font-size: 3vh;">{{selectedFavorite}}</span>
  <ul class="dropdown-content">
    <li class="li-content" (click)="filterFavorites('All')">All</li>
    <li class="li-content" *ngFor="let data of filterdrop" [value]="data"
#liSelect (click)="filterFavorites(liSelect.innerText)">{{data}}</li>
```

```
  </ul>
</div>
```

A figura 30 apresenta o resultado final da página *favorites*.

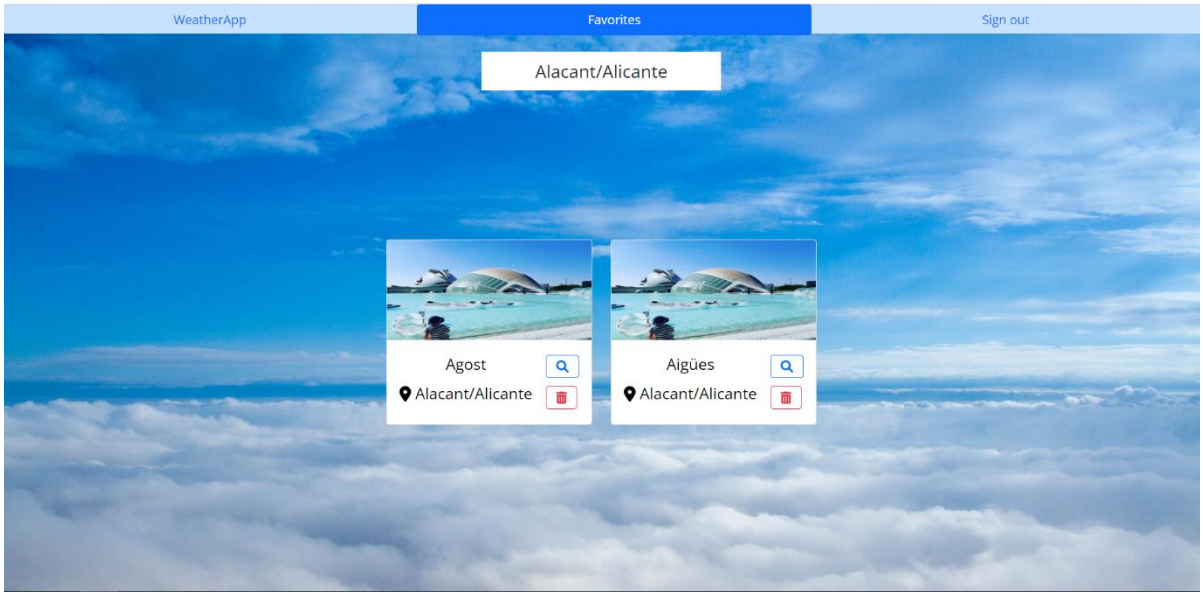

Figura 30 Favorites resultado final.

4.8 INTEGRAÇÃO DE PARÂMETROS AMBIENTAIS

Para integração de mais parâmetros ambientais na *web app* poderíamos explorar duas situações: (a) utilizar mais funcionalidades da API já utilizada, juntando por exemplo também informação sobre o vento (direção e velocidade), humidade e sensação térmica. (b) Explorar outras APIs (gratuitas) que respondam ao tipo de informação que pretendemos mostrar na web app.

Neste projeto vamos focar somente na situação (a), usar a API *el-tiempo*, e usar os outros parâmetros/dados que ela nos fornece. Para isso teremos que fazer alterações no *front-end* e no *back-end*.

Assim, voltamos a analisar a documentação da API, para poder saber como os dados (informação do vento, humidade, etc.) são recebidos, em qual unidade estão, escala de medição ou outras propriedades relevantes que influenciam na medição e obtenção do valor. Depois incluir na classe *WeatherList*, apresentada na seção 4.5.4, os novos dados que queremos mapear do JSON obtido do *endpoint* C. Como exemplo, a seguir, adicionaremos esses dados somente para o "dia atual" no resultado da pesquisa (para isso devemos adicionar na classe WeatherList):

```
@JsonProperty("humedad")
private String humidade;

@JsonProperty("viento")
private String vento;

private List<String> senTermica;

@SuppressWarnings("unchecked")
@JsonProperty("pronostico")
private void precipitationUnpack(Map<String,Object> pronostico) {
        Map<String, Object> hoy = (Map<String,Object>)pronostico.get("hoy");
        this.probPrecipitation = (List<String>)hoy.get("prob_precipitacion");
         this.sensTermica = (List<String>)hoy.get("sens_termica");
}
```

Observe que a sensação térmica é obtida da mesma maneira que probabilidade de chuva, então foi incluída no método precipitationUnpack(). Desta maneira já estamos enviando os novos parâmetros para o *front-end*. Provavelmente o *design* da tela *Search Result* precisará ser redimensionado para a adaptação dos novos parâmetros no *card*.

E agora claro, os novos parâmetros precisam ser incluídos na interface:

```
interface weather {
  fecha:String;
  temperatura_actual:String;
  municipio:municipio;
  stateskyid:String;
  tempAtualMax:String;
  tempAtualMin:String;
  probPrecipitation:String[];
  proximos_dias: ProximosDia[];
  humedad:String;
  viento:String;
  senTermica:String[];
}
```

Apenas para exemplificar, não vamos alterar o design da página *Search Result*, vamos apenas incluir os novos dados no *card* que representa o dia atual da pesquisa:

```
<div class="cardOne"
(mouseover)="toggle(11,weatherData.tempAtualMax,weatherData.tempAtualMin)"
(mouseout)="toggle(11,weatherData.tempAtualMax,weatherData.tempAtualMin)">
      <div id="cardDate">
            <h3 class="title">{{GetWeekdayFromDate(weatherData.fecha)}}</h3>
            <h5 class="diaOne">{{GetMonthFromDate(weatherData.fecha)}}</h5>
      </div>
```

```
      <div id="cardIconOne">
            <img [src]="displayImg(weatherData.stateskyid)">
      </div>

      <div id="weatherDatasOne">
            <div id="maxMinOne">
            <p>
                  <i class="fa-solid fa-temperature-arrow-up"></i>
{{weatherData.tempAtualMax}}ºC
                  <br><i class="fa-solid fa-temperature-arrow-down"></i>
{{weatherData.tempAtualMin}}ºC
            </p>
            </div>
            <div id="probRainOne">
                  <div *ngFor="let data of weatherData.probPrecipitation;
let i = index">
                        <i class="fa-solid fa-umbrella"></i>
{{timearray[i]}} {{data}}%
                  </div>
            </div>
            <i class="fa-solid fa-droplet-
percent"></i>{{weatherData.humedad}}%
            <i class="fa-regular fa-wind"></i>{{weatherData.viento}}Km/h
            <i class="fa-regular fa-wind"></i>{{weatherData.senTermica[h]}}ºC
            <h3 class="tempOne">{{weatherData.temperatura_actual}}ºC</h3>
            <div class="barOne">
            <div class="emptybar"></div>
            <div id="11" class="filledbar" [ngStyle]="barStyle()"></div>
            </div>
      </div>
</div>
```

Com o índice do *array* de sensação térmica igual a:

```
h:Date = new Date().getHours();
```

Com isso fizemos todas as alterações técnicas necessárias para apresentar ao utilizador novos parâmetros ambientais.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 CONCLUSÕES

Desenvolver uma *web app* pode parecer uma tarefa simples, mas envolve muitos detalhes que à primeira vista não são levados em consideração. Neste relatório foi apresentado a técnica de CI/CD que é vital para desenvolvimento de aplicações modernas e softwares que podem ser utilizados para o desenvolvimento de uma aplicação do zero. Descrevendo a função de cada um e fazendo comparações com outros softwares similares, conclui-se que não existe um melhor e um pior, mas sim o ideal para cada projeto.

Alguns fatores que interferem na escolha do software são, a complexidade do projeto, quantas pessoas estão envolvidas no desenvolvimento, funcionalidades e tamanho da aplicação. Neste projeto específico de desenvolvimento da *web app* para visualização de parâmetros ambientais foram escolhidos os softwares destacados no capítulo 2 por serem ferramentas muito utilizadas em diversos projetos na Atos e em outras empresas de TI na atualidade.

Ao realizar o estado da arte, percebe-se que em termos de *design* a *web app* desenvolvida não é muito diferente das mais utilizadas na atualidade (i.e., já existem no mercado). A maior diferença está na quantidade de dados e funcionalidades apresentadas. Cada *web app* tem sua própria API ou próprio fornecedor desses dados, algumas conseguem muitos mais dados, como qualidade do ar, nível de radiação, visibilidade, pólen, etc.
Concluindo, que que a *web app* apresentada neste relatório para ser um real concorrente dessas *web apps* precisaríamos de outro serviço API mais completo e provavelmente pago.

É importante salientar que como esse projeto foi um projeto de formação, as etapas seguidas nele e descritas durante o processo de desenvolvimento nem sempre representam fielmente o dia a dia de trabalho de um desenvolvedor da Atos. O nosso cliente por exemplo falou diretamente com a equipa de desenvolvedores e apresentou sua proposta, uma *web app* para

visualização de parâmetros ambientais, deixou claro os parâmetros mínimos necessários para o desenvolvimento do projeto, além de requerer outras mínimas funcionalidades.

Um erro cometido pela equipa de desenvolvimento foi incluir coisas além do que foi pedido, deixando o trabalho mais complexo e com o mesmo prazo de entrega, como por exemplo o carrossel na *homepage*, as barras de temperatura dentro de cada *card*, os ícones para representar o estado atual do clima em cada *card*, as imagens usadas em cada município salvo como favorito na página *favorites*. Desses itens listados apenas o carrossel foi eliminado, todos os outros o cliente quando viu numa das reuniões de *sprint*, optou por incluir no projeto e mesmo assim não mudamos o prazo final de entrega da *web app*. Ficando como ensinamento fazer somente o que foi pedido e da maneira mais simples possível, pois sempre teremos o mesmo prazo para execução das tarefas.

Durante esse projeto deparei pela primeira vez com algumas das tecnologias utilizadas, e serviu ainda para colocar em prática em uma *web app* real outros conceitos vistos durante o curso, como o *message broker* por exemplo. Por último, é de referir que esta *web app* permitiu aprender com os processos e divisões de trabalho bem definidos das equipas de desenvolvedores e como se encaixam e conversam com outras equipas utilizando a metodologia ágil Scrum.

Após o término do processo de formação fui chamado para uma pequena entrevista com a equipa da Atos responsável pelo sistema de gerenciamento dos jogos olímpicos e tive o prazer de me juntar a eles, onde sigo na mesma equipa até o presente momento. Sem dúvidas o processo de formação me deu uma base de confiança para que pudesse dar esse primeiro passo dentro de uma equipa e projeto real. Seguimos a metodologia Scrum com sprints de aproximadamente 10 dias.

5.2 TRABALHO FUTURO

A *web app* está funcional e pronta para ser lançada ao público, mas existem mais alguns pontos além dos identificados na seção 4.6 que podem ser implementados como melhoria, antes ou depois da *web app* já estar disponível para acesso público.

A primeira e maior melhoria, seria também na página dos favoritos, após alguns testes, percebemos que se um utilizador tem muitos municípios na sua lista de favoritos não é tão pratico ficar filtrando os resultados pelo *combobox* na página, então como uma possível melhoria, seria implementar uma barra de pesquisa, onde o utilizador pudesse pesquisar diretamente pelo nome do município que gostasse de ver os dados.

Outra possível melhoria seria acrescentar outras opções de login à *web app*, como poder usar sua conta google ou github, essas funcionalidades podem ser implementadas com o Keycloak. Esses são os dois pontos mais claros identificados durante o período de testes da *web app*, possivelmente após o lançamento e com uso em massa haverá outros pontos passiveis de melhorias que poderão ser levados em consideração e corrigidos em uma nova versão da *web app*.

Como trabalho futuro para disponibilizar a *web app* para acesso ao público, será realização do *deploy* seguindo as melhores práticas de desenvolvimento, e as técnicas modernas utilizadas atualmente no mercado, i.e., realizar o *deploy* com Docker e Kubernetes . Esse trabalho normalmente é feito por outro time de profissionais DevOps que são responsáveis por automatizar o desenvolvimento, a implantação e a operação de aplicações, o que inclui a utilização do Docker e do Kubernetes para criar, empacotar e orquestrar containers. Mas ao fim do desenvolvimento da *web app* estudamos as etapas necessárias para que isso fosse possível.

Primeiro devemos “empacotar” nossa *web app* em containers com Docker, separando cada módulo em um container. Para isso certificamo-nos de ter uma estrutura de projeto organizada, por exemplo, um projeto Maven deve ter o arquivo "pom.xml" na raiz, onde as dependências e configurações do projeto são definidas além de instalar o Docker na máquina que vai realizar o *deploy* da *web app*. Em seguida criar um arquivo chamado "Dockerfile" no diretório raiz do projeto. Este arquivo definirá como o container será construído. O terceiro passo é escolher uma imagem base no Docker Hub (https://hub.docker.com/), para os módulos do *back-end*, uma imagem que contenha o mesmo JDK (Java Development Kit) utilizado para o desenvolvimento, uma boa opção é utilizar a imagem oficial do Java, várias outras tecnologias também possuem sua imagem oficial no Docker Hub, como por exemplo ActiveMQ, Keycloak, MySQL, que foram utilizados também no desenvolvimento da nossa *web app*. Após escolher a imagem devemos configurar o Dockerfile, assim que o arquivo está pronto, abrir um CMD no diretório do projeto e executar o seguinte comando para criar a imagem:”*docker build -t <nome_da_imagem>*”.

Com a imagem criada, basta apenas executar o comando: “*docker run -p <porta_host>:<porta_container> <nome_da_imagem>*”. Agora a *web app* está “dockerizada” e pronta para ser executada em qualquer ambiente que suporte o Docker, garantindo a portabilidade e a consistência em diferentes cenários. O próximo passo seria instalar também o Kubernetes e configura-lo para poder fazer o *deploy* de nossos containers.

REFERÊNCIAS

AccuWeather (2023). AccuWeather https://www.accuweather.com/, acedido 2023/06/23

ActiveMQ (2023a). Flexible & Powerful Open Source Multi-Protocol Messaging, https://activemq.apache.org/, acedido 2023/03/07

ActiveMQ (2023b). Cross Language Clients, https://activemq.apache.org/cross-language-clients, acedido 2023/03/07

Adobe (2023). Photoshop. https://www.adobe.com/pt/products/photoshop.html, acedido 2023/06/23

Angular (2023). The web development framework for building the future. https://angular.io/, acedido 2023/03/07

Aripradono, H. W., & Hisham, M. R. (2022). Web Marketplace Design and Development for Household Needs Using the Wdlc Model With the Scrum Method. In CoMBInES-Conference on Management, Business, Innovation, Education and Social Sciences (Vol. 2, No. 1, pp. 507-515).

Arshad, M., Brohi, M. N., Soomro, T. R., Ghazal, T. M., Alzoubi, H. M., & Alshurideh, M. (2023). NoSQL: Future of BigData Analytics Characteristics and Comparison with RDBMS. In The Effect of Information Technology on Business and Marketing Intelligence Systems (pp. 1927-1951). Cham: Springer International Publishing.

Awesome (2023). Take the hassle out of icons in your website. https://fontawesome.com/, acedido 2023/07/12

Axure (2023). Design Interactive UI Mockups Without Code. https://www.axure.com/a/UI-mockup-tool, acedido 2023/06/23

Beeharry, Y., Fowdur, T. P., & Sunglee, J. A. (2019). A cloud-based real-time weather forecasting application. In 2019 14th international conference on advanced technologies, systems and services in telecommunications (TELSIKS) (pp. 294-297). IEEE.

Bello, Y., Figetakis, E., Refaey, A., & Spachos, P. (2022). Continuous Integration and Continuous Delivery Framework for SDS. In 2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (pp. 406-410). IEEE.

Bessa, André (2021). Tipos de Autenticação: Senha, Token, JWT, Dois Fatores e Mais. https://www.alura.com.br/artigos/tipos-de-autenticacao, acedido 2023/06/21

Bobrovskis, S., & Jurenoks, A. (2018). A Survey of Continuous Integration, Continuous Delivery and Continuos Deployment. In BIR workshops (pp. 314-322).

Bootstrap (2023). Build fast, responsive sites with Bootstrap. https://getbootstrap.com/, acedido 2023/06/23

Brendel, C. E., Dymond, R. L., & Aguilar, M. F. (2019). An interactive web app for retrieval, visualization, and analysis of hydrologic and meteorological time series data. Environmental Modelling & Software, 117, 14-28.

Canva (2023). Free Mockup Generator, https://www.canva.com/create/mockup-generator/, acedido 2023/06/23

Clique (2022). What is a Mockup? (+How to Create a Mockup in 2022). https://cliquestudios.com/mockups/, acedido 2023/06/23

Codebit (2022). 7 Principais Linguagens de Programação para o Desenvolvimento Web. https://codebit.com.br/blog/developer/7-principais-linguagens-programacao-desenvolvimento-web, acedido 2023/03/07

Dantas, J. C. (2021). Contribuições da Implantação do Scrum como Metodologia Ágil para a Otimização da Gestão de Projetos nas Organizações. RECIMA21-Revista Científica Multidisciplinar-ISSN 2675-6218, 2(7), e27541-e27541.

Duarte, Miguel, et al. "Evaluation of iot self-healing mechanisms using fault-injection in message brokers." 2022 IEEE/ACM 4th International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT). IEEE, 2022.

Figma (2023). Create realistic experiences. https://www.figma.com/prototyping/, acedido 2023/06/23

GKE (n.d.). Práticas recomendadas para entrega e integração contínuas no Google Kubernetes Engine. https://cloud.google.com/kubernetes-engine/docs/concepts/best-practices-continuous-integration-delivery-kubernetes, acedido 2023/03/07

Gluu (2023). Build Secure Identity Journeys Faster and Easier, https://gluu.org/, acedido 2023/03/07

Hidayati, A., Budiardjo, E. K., & Purwandari, B. (2022). Scrum Team Competencies in Information Technology Professionals in the Global Software Development Environment. International Journal of Human Capital and Information Technology Professionals (IJHCITP), 13(1), 1-21.

Hron, M., & Obwegeser, N. (2022). Why and how is Scrum being adapted in practice: A systematic review. Journal of Systems and Software, 183, 111110.

Jenkins (2023). Jenkins. https://www.jenkins.io, acedido 2023/06/22

JETBRAINS (2022).CI/CD Best Practices. https://www.jetbrains.com/teamcity/ci-cd-guide/ci-cd-best-practices/, acedido 2023/03/07

Kafka (2023). Apache Kafka, https://kafka.apache.org/, acedido 2023/03/07

Keycloak (2023). Open Source Identity and Access Management, https://www.keycloak.org/, acedido 2023/03/07

Köbler, Niko (2023) HALLO, ICH BIN NIKO!, https://n-k.de/, acedido 2023/07/12

Kofler, M. (2005). The Definitive Guide to MySQL5. Berkeley, CA: Apress.

Kouomeu, Kevin (2022). Spring Boot and Apache ActiveMQ - JMS Messaging. https://1kevinson.com/springboot-artemis-broker/, acedido 2023/03/07

Kox, T., Göber, M., Wentzel, B., Freundl, E., & Rust, H. W. (2020, September). Fostering weather and climate literacy among pupils by engagement in a weather citizen science project. In Proceedings of Austrian Citizen Science Conference, 6pp.

Liao, Q. (2020, October). Modelling CI/CD Pipeline Through Agent-Based Simulation. In 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 155-156). IEEE.

LuisDEV (2022). Introdução ao RabbitMQ, https://www.luisdev.com.br/2022/07/06/introducao-ao-rabbitmq/, acedido 2023/03/07

Micco, J. (2021) "Keynote Talk - Industry Best Practices for CI/CD," 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), Porto de Galinhas, Brazil, 2021, pp. 25-25, doi: 10.1109/ICST49551.2021.00012.

Micronaut (2023). A modern, jvm-based, full-stack framework for building modular, easily testable microservice and serverless applications, https://micronaut.io/, acedido 2023/03/07

Morandini, M., Coleti, T. A., Oliveira Jr, E., & Corrêa, P. L. P. (2021). Considerations about the efficiency and sufficiency of the utilization of the Scrum methodology: A survey for analyzing results for development teams. Computer Science Review, 39, 100314.

MySQL (2023). MySQL Documentation. https://dev.mysql.com/doc/, acedido 2023/03/07

MySQL (n.d.). O que é o MySQL, https://cloud.google.com/mysql, acedido 2023/03/07

Navathe, S. & Elmasri, R. (2005). Sistemas de Banco de Dados. Pearson Universitários

Nogueira, A. F., & Zenha-Rela, M. (2021). Monitoring a ci/cd workflow using process mining. SN Computer Science, 2, 1-16.

Noleto, Cairo (2020). Aplicações Web: Entenda o que são e como funcionam!. https://blog.betrybe.com/desenvolvimento-web/aplicacoes-web/, acedido 2023/06/18

Okta (2023). Everything starts with Identity, https://www.okta.com/, acedido 2023/03/07

OpenWeather (2023). OpenWeather. Weather forecasts, nowcasts and history in a fast and elegant way https://openweathermap.org/, acedido 2023/06/23

Oracle (2023). Database, https://www.oracle.com/database/, acedido 2023/03/07

OTempo (2022). Top 5 melhores aplicativos para ver a previsão do tempo. https://www.otempo.com.br/interessa/top-5-melhores-aplicativos-para-ver-a-previsao-do-tempo-1.2683702, acedido 2023/06/23

Pan, Z., Shen, W., Wang, X., Yang, Y., Chang, R., Liu, Y., ... & Ren, K. (2023). Ambush From All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines. IEEE Transactions on Dependable and Secure Computing.

Patrucco, A. S., Canterino, F., & Minelgaite, I. (2022). How do scrum methodologies influence the team's cultural values? A multiple case study on agile teams in Nonsoftware industries. IEEE Transactions on Engineering Management, 69(6), 3503-3513.

Pluralsight (2023). Introduction to OAuth2, OpenID Connect and JSON Web Tokens (JWT). https://www.pluralsight.com/courses/oauth2-json-web-tokens-openid-connect-introduction, acedido 2023/03/07

Pm-partners (2021) The Agile Journey: A Scrum overview. https://www.pm-partners.com.au/the-agile-journey-a-scrum-overview/, acedido 2023/06/23

PostgreSQL (2023). PostgreSQL: The World's Most Advanced Open Source Relational Database, https://www.postgresql.org/, acedido 2023/03/07

Quarkus (2023). A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards. https://quarkus.io/, acedido 2023/03/07

RabbitMQ (2023). RabbitMQ is the most widely deployed open-source message broker, https://www.rabbitmq.com/, acedido 2023/03/07

Rangnau, T., Buijtenen, R. V., Fransen, F., & Turkmen, F. (2020, October). Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines. In 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC) (pp. 145-154). IEEE.

React (2023). React: The library for web and native user interfaceshttps://react.dev/, acedido 2023/03/07

Reddy, H. B. S., Reddy, R. R. S., Jonnalagadda, R., Singh, P., & Gogineni, A. (2022). Analysis of the Unexplored Security Issues Common to All Types of NoSQL Databases. Asian Journal of Research in Computer Science, 14(1), 1-12.

Rudenko, R., Pires, I. M., Liberato, M., Barroso, J., & Reis, A. (2022). A brief review on 4D weather visualization. Sustainability, 14(9), 5248.

Sabharwal, N., Pandey, P., Sabharwal, N., & Pandey, P. (2020). Automation and Orchestration of Container Monitoring. Monitoring Microservices and Containerized Applications: Deployment, Configuration, and Best Practices for Prometheus and Alert Manager, 271-302.

Semaphore (2023). What Is Microservice Architecture?. https://semaphoreci.com/blog/microservice-architecture, acedido 2023/03/07

Seobility (2023). All-In-One SEO Software & Tools. https://www.seobility.net/en/, acedido 2023/03/07

Singh, A., Singh, V., Aggarwal, A., & Aggarwal, S. (2022). Improving Business deliveries using Continuous Integration and Continuous Delivery using Jenkins and an Advanced Version control system for Microservices-based system. In 2022 5th International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT) (pp. 1-4). IEEE.

Spring (2023). Spring. https://spring.io/, acedido 2023/03/07

STH (2023). What Is The Difference Between Website And Web Application. https://www.softwaretestinghelp.com/website-vs-web-application/, acedido 2023/06/18

StrongNet (n.d.). Web Applications. https://www.strongnet.pt/web/web_applications, acedido 2023/03/07

Takeuchi, H. & Nonaka, I. (1986). The New New Product Development Game. https://hbr.org/1986/01/the-new-new-product-development-game, acedido 2023/06/23

TIBCO (2023). O que é a integração contínua?. https://www.tibco.com/pt-br/reference-center/what-is-continuous-integration, acedido 2023/06/18

Travis (2023). Travis CI. https://www.travis-ci.com/, acedido 2023/06/22

Triska, P., Amman, F., Endler, L., & Bergthaler, A. (2023). WAVES (Web-based tool for Analysis and Visualization of Environmental Samples)—a web application for visualization of wastewater pathogen sequencing results. Bioinformatics, 39(4), btad160.

Vue.js (2023). The Progressive JavaScript Framewor, khttps://vuejs.org/, acedido 2023/03/07

Weather (2023)The Weather Channel. https://weather.com/, acedido 2023/06/23

Weatherbit (2023).Weather API for Scientists. https://www.weatherbit.io/, acedido 2023/06/23

Wunderground (2023). Weather Underground. https://www.wunderground.com/, acedido 2023/06/23

Ziolkowski D. (2022). 6 CI/CD best practices you need to know. https://www.architect.io/blog/2022-07-13/six-cicd-best-practices/, acedido 2023/03/07

APÊNDICE A: DADOS METEOROLÓGICOS DO DIA ATUAL E SEGUINTES

O presente Apêndice apresenta o terceiro *endpoint* apresentado na seção 4.2 API. Retorna informações geográficas e meteorológicas de um município espanhol do dia atual e dos próximos dias, totalizando uma semana de previsão.


```
// 20230706160417
// https://www.el-tiempo.net/api/json/v2/provincias/01/municipios/01001

{
  "origin": {
    "productor": "Agencia Estatal de Meteorología - AEMET. Gobierno de España",
    "web": "https://www.aemet.es",
    "language": "es",
    "copyright": "© AEMET. Autorizado el uso de la información y su reproducción citando a AEMET como autora de la misma.",
    "nota_legal": "https://www.aemet.es/es/nota_legal",
    "descripcion": "elTiempo.net muestra la información meteorológica creada por © AEMET"
  },
  "title": "elTiempo.net | El tiempo en Alegría-Dulantzi (Araba/Álava)",
  "metadescripcion": "El tiempo en Alegría-Dulantzi (Araba/Álava)",
  "keywords": "Alegría-Dulantzi, Provincia de Álava, previsión meteorológica, previsión del tiempo, España, viento, humedad, precipitaciones, lluvia, estado del cielo",
  "municipio": {
    "CODIGOINE": "01001000000",
    "ID_REL": "1010014",
    "COD_GEO": "01010",
    "CODPROV": "01",
    "NOMBRE_PROVINCIA": "Araba/Álava",
    "NOMBRE": "Alegría-Dulantzi",
    "POBLACION_MUNI": 2925,
```

```
    "SUPERFICIE": 1994.5872,
    "PERIMETRO": 35069,
    "CODIGOINE_CAPITAL": "01001000101",
    "NOMBRE_CAPITAL": "Alegría-Dulantzi",
    "POBLACION_CAPITAL": "2815",
    "HOJA_MTN25": "0113-3",
    "LONGITUD_ETRS89_REGCAN95": -2.51243731,
    "LATITUD_ETRS89_REGCAN95": 42.83981158,
    "ORIGEN_COORD": "Mapa",
    "ALTITUD": 568,
    "ORIGEN_ALTITUD": "MDT5",
    "DISCREPANTE_INE": 0
  },
  "fecha": "2023-07-06",
  "stateSky": {
    "description": "Muy nuboso con tormenta y lluvia escasa",
    "id": "63"
  },
  "temperatura_actual": "25",
  "temperaturas": {
    "max": "26",
    "min": "15"
  },
  "humedad": "61",
  "viento": "3",
  "precipitacion": "0.1",
  "lluvia": "40",
  "imagen": null,
  "pronostico": {
    "hoy": {
      "@attributes": {
        "fecha": "2023-07-06",
        "orto": "06:37",
        "ocaso": "21:51"
      },
      "estado_cielo": [
        "15",
        "16",
        "16",
        "16",
        "16",
```

```
      "16",
      "12",
      "12",
      "16",
      "63",
      "16",
      "12",
      "12",
      "14",
      "11n",
      "11n"
    ],
    "precipitacion": [
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0.1",
      "0",
      "0",
      "0",
      "Ip",
      "0",
      "0"
    ],
    "prob_precipitacion": [
      "40",
      "90",
      "15"
    ],
    "prob_tormenta": [
      "40",
      "90",
      "15"
    ],
    "nieve": [
```

```
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0"
    ],
    "prob_nieve": [
      "0",
      "0",
      "0"
    ],
    "temperatura": [
      "17",
      "19",
      "20",
      "21",
      "22",
      "23",
      "25",
      "26",
      "25",
      "24",
      "24",
      "23",
      "22",
      "20",
      "19"
    ],
    "sens_termica": [
      "17",
```

```
        "19",
        "20",
        "21",
        "22",
        "23",
        "25",
        "26",
        "25",
        "24",
        "24",
        "23",
        "22",
        "20",
        "19"
      ],
      "humedad_relativa": [
        "82",
        "75",
        "69",
        "67",
        "64",
        "69",
        "67",
        "61",
        "61",
        "70",
        "67",
        "71",
        "72",
        "80",
        "84"
      ],
      "viento": [
        {
          "@attributes": {
            "periodo": "09"
          },
          "direccion": "NE",
          "velocidad": "3"
        },
        {
```

```
    "@attributes": {
      "periodo": "10"
    },
    "direccion": "N",
    "velocidad": "3"
  },
  {
    "@attributes": {
      "periodo": "11"
    },
    "direccion": "NO",
    "velocidad": "4"
  },
  {
    "@attributes": {
      "periodo": "12"
    },
    "direccion": "O",
    "velocidad": "5"
  },
  {
    "@attributes": {
      "periodo": "13"
    },
    "direccion": "NO",
    "velocidad": "3"
  },
  {
    "@attributes": {
      "periodo": "14"
    },
    "direccion": "SE",
    "velocidad": "7"
  },
  {
    "@attributes": {
      "periodo": "15"
    },
    "direccion": "SE",
    "velocidad": "9"
  },
```

```
{
  "@attributes": {
    "periodo": "16"
  },
  "direccion": "SE",
  "velocidad": "6"
},
{
  "@attributes": {
    "periodo": "17"
  },
  "direccion": "E",
  "velocidad": "3"
},
{
  "@attributes": {
    "periodo": "18"
  },
  "direccion": "N",
  "velocidad": "19"
},
{
  "@attributes": {
    "periodo": "19"
  },
  "direccion": "NE",
  "velocidad": "19"
},
{
  "@attributes": {
    "periodo": "20"
  },
  "direccion": "E",
  "velocidad": "16"
},
{
  "@attributes": {
    "periodo": "21"
  },
  "direccion": "E",
  "velocidad": "19"
```

```
      },
      {
        "@attributes": {
          "periodo": "22"
        },
        "direccion": "NE",
        "velocidad": "4"
      },
      {
        "@attributes": {
          "periodo": "23"
        },
        "direccion": "S",
        "velocidad": "4"
      }
    ],
    "racha_max": [
      "9",
      "8",
      "10",
      "8",
      "13",
      "19",
      "18",
      "14",
      "29",
      "32",
      "26",
      "33",
      "9",
      "6",
      "3"
    ]
  },
  "manana": {
    "@attributes": {
      "fecha": "2023-07-07",
      "orto": "06:38",
      "ocaso": "21:51"
    },
    "estado_cielo": [
```

```
      "12n",
      "17n",
      "12n",
      "12n",
      "12n",
      "12n",
      "12n",
      "12",
      "13",
      "13",
      "13",
      "13",
      "13",
      "13",
      "12",
      "64",
      "23",
      "15",
      "12",
      "17",
      "11",
      "11",
      "12n",
      "11n"
    ],
    "precipitacion": [
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
```

```
      "0.8",
      "6",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0"
    ],
    "prob_precipitacion": [
      "0",
      "30",
      "80",
      "0"
    ],
    "prob_tormenta": [
      "0",
      "30",
      "80",
      "0"
    ],
    "nieve": [
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
      "0",
```

```
      "0",
      "0",
      "0",
      "0",
      "0",
      "0"
    ],
    "prob_nieve": [
      "0",
      "0",
      "0",
      "0"
    ],
    "temperatura": [
      "19",
      "18",
      "18",
      "18",
      "19",
      "19",
      "18",
      "18",
      "19",
      "20",
      "22",
      "23",
      "25",
      "26",
      "27",
      "29",
      "22",
      "25",
      "26",
      "27",
      "26",
      "24",
      "23",
      "22"
    ],
    "sens_termica": [
      "19",
```

```
      "18",
      "18",
      "18",
      "19",
      "19",
      "18",
      "18",
      "19",
      "20",
      "22",
      "23",
      "25",
      "26",
      "28",
      "30",
      "22",
      "25",
      "26",
      "28",
      "26",
      "24",
      "23",
      "22"
    ],
    "humedad_relativa": [
      "80",
      "80",
      "77",
      "76",
      "75",
      "72",
      "72",
      "80",
      "82",
      "78",
      "73",
      "68",
      "62",
      "60",
      "56",
      "50",
```

```
      "90",
      "57",
      "50",
      "52",
      "61",
      "66",
      "65",
      "65"
    ],
    "viento": [
      {
        "@attributes": {
          "periodo": "00"
        },
        "direccion": "SO",
        "velocidad": "2"
      },
      {
        "@attributes": {
          "periodo": "01"
        },
        "direccion": "NE",
        "velocidad": "4"
      },
      {
        "@attributes": {
          "periodo": "02"
        },
        "direccion": "NE",
        "velocidad": "9"
      },
      {
        "@attributes": {
          "periodo": "03"
        },
        "direccion": "E",
        "velocidad": "12"
      },
      {
        "@attributes": {
          "periodo": "04"
```

```
    },
    "direccion": "SE",
    "velocidad": "17"
  },
  {
    "@attributes": {
      "periodo": "05"
    },
    "direccion": "SE",
    "velocidad": "30"
  },
  {
    "@attributes": {
      "periodo": "06"
    },
    "direccion": "SE",
    "velocidad": "35"
  },
  {
    "@attributes": {
      "periodo": "07"
    },
    "direccion": "SE",
    "velocidad": "33"
  },
  {
    "@attributes": {
      "periodo": "08"
    },
    "direccion": "SE",
    "velocidad": "31"
  },
  {
    "@attributes": {
      "periodo": "09"
    },
    "direccion": "SE",
    "velocidad": "37"
  },
  {
    "@attributes": {
```

```
      "periodo": "10"
    },
    "direccion": "SE",
    "velocidad": "40"
  },
  {
    "@attributes": {
      "periodo": "11"
    },
    "direccion": "SE",
    "velocidad": "35"
  },
  {
    "@attributes": {
      "periodo": "12"
    },
    "direccion": "SE",
    "velocidad": "27"
  },
  {
    "@attributes": {
      "periodo": "13"
    },
    "direccion": "SE",
    "velocidad": "25"
  },
  {
    "@attributes": {
      "periodo": "14"
    },
    "direccion": "SE",
    "velocidad": "27"
  },
  {
    "@attributes": {
      "periodo": "15"
    },
    "direccion": "SE",
    "velocidad": "26"
  },
  {
```

```
    "@attributes": {
      "periodo": "16"
    },
    "direccion": "S",
    "velocidad": "34"
  },
  {
    "@attributes": {
      "periodo": "17"
    },
    "direccion": "E",
    "velocidad": "23"
  },
  {
    "@attributes": {
      "periodo": "18"
    },
    "direccion": "E",
    "velocidad": "25"
  },
  {
    "@attributes": {
      "periodo": "19"
    },
    "direccion": "NE",
    "velocidad": "20"
  },
  {
    "@attributes": {
      "periodo": "20"
    },
    "direccion": "N",
    "velocidad": "9"
  },
  {
    "@attributes": {
      "periodo": "21"
    },
    "direccion": "SE",
    "velocidad": "7"
  },
```

```
    {
      "@attributes": {
        "periodo": "22"
      },
      "direccion": "SE",
      "velocidad": "17"
    },
    {
      "@attributes": {
        "periodo": "23"
      },
      "direccion": "SE",
      "velocidad": "30"
    }
  ],
  "racha_max": [
    "5",
    "13",
    "19",
    "26",
    "45",
    "52",
    "52",
    "48",
    "57",
    "62",
    "58",
    "47",
    "42",
    "44",
    "44",
    "50",
    "34",
    "42",
    "30",
    "19",
    "11",
    "26",
    "46",
    "56"
  ]
```

```
    }
  },
  "proximos_dias": [
    {
      "@attributes": {
        "fecha": "2023-07-07"
      },
      "prob_precipitacion": [
        "85",
        "30",
        "80",
        "0",
        "30",
        "80",
        "0"
      ],
      "cota_nieve_prov": [
        {
          "@attributes": {
            "periodo": "00-24"
          }
        },
        {
          "@attributes": {
            "periodo": "00-12"
          }
        },
        {
          "@attributes": {
            "periodo": "12-24"
          }
        },
        {
          "@attributes": {
            "periodo": "00-06"
          }
        },
        {
          "@attributes": {
            "periodo": "06-12"
          }
```

```
    },
    {
      "@attributes": {
        "periodo": "12-18"
      }
    },
    {
      "@attributes": {
        "periodo": "18-24"
      }
    }
  ],
  "estado_cielo": [
    "23",
    "13",
    "51",
    "12n",
    "13",
    "61",
    "17"
  ],
  "viento": [
    {
      "@attributes": {
        "periodo": "00-24"
      },
      "direccion": "SE",
      "velocidad": "25"
    },
    {
      "@attributes": {
        "periodo": "00-12"
      },
      "direccion": "SE",
      "velocidad": "30"
    },
    {
      "@attributes": {
        "periodo": "12-24"
      },
      "direccion": "SE",
```

```
      "velocidad": "25"
    },
    {
      "@attributes": {
        "periodo": "00-06"
      },
      "direccion": "SE",
      "velocidad": "30"
    },
    {
      "@attributes": {
        "periodo": "06-12"
      },
      "direccion": "SE",
      "velocidad": "25"
    },
    {
      "@attributes": {
        "periodo": "12-18"
      },
      "direccion": "N",
      "velocidad": "10"
    },
    {
      "@attributes": {
        "periodo": "18-24"
      },
      "direccion": "S",
      "velocidad": "35"
    }
  ],
  "racha_max": [
    "50",
    "50",
    "45",
    "50",
    "45",
    {
      "@attributes": {
        "periodo": "12-18"
      }
```

```
      },
      "55"
    ],
    "temperatura": {
      "maxima": "30",
      "minima": "18",
      "dato": [
        "19",
        "27",
        "26",
        "21"
      ]
    },
    "sens_termica": {
      "maxima": "30",
      "minima": "18",
      "dato": [
        "19",
        "28",
        "26",
        "21"
      ]
    },
    "humedad_relativa": {
      "maxima": "90",
      "minima": "50",
      "dato": [
        "80",
        "55",
        "60",
        "65"
      ]
    },
    "uv_max": "8"
  },
  {
    "@attributes": {
      "fecha": "2023-07-08"
    },
    "prob_precipitacion": [
      "10",
```

```
      "0",
      "10"
    ],
    "cota_nieve_prov": [
      {
        "@attributes": {
          "periodo": "00-24"
        }
      },
      {
        "@attributes": {
          "periodo": "00-12"
        }
      },
      {
        "@attributes": {
          "periodo": "12-24"
        }
      }
    ],
    "estado_cielo": [
      "13",
      "12",
      "13"
    ],
    "viento": [
      {
        "@attributes": {
          "periodo": "00-24"
        },
        "direccion": "C",
        "velocidad": "0"
      },
      {
        "@attributes": {
          "periodo": "00-12"
        },
        "direccion": "S",
        "velocidad": "35"
      },
      {
```

```
        "@attributes": {
          "periodo": "12-24"
        },
        "direccion": "N",
        "velocidad": "10"
      }
    ],
    "racha_max": [
      {
        "@attributes": {
          "periodo": "00-24"
        }
      },
      "55",
      {
        "@attributes": {
          "periodo": "12-24"
        }
      }
    ],
    "temperatura": {
      "maxima": "32",
      "minima": "16"
    },
    "sens_termica": {
      "maxima": "32",
      "minima": "16"
    },
    "humedad_relativa": {
      "maxima": "90",
      "minima": "35"
    },
    "uv_max": "9"
  },
  {
    "@attributes": {
      "fecha": "2023-07-09"
    },
    "prob_precipitacion": [
      "45",
      "10",
```

```
      "30"
    ],
    "cota_nieve_prov": [
      {
        "@attributes": {
          "periodo": "00-24"
        }
      },
      {
        "@attributes": {
          "periodo": "00-12"
        }
      },
      {
        "@attributes": {
          "periodo": "12-24"
        }
      }
    ],
    "estado_cielo": [
      "14",
      "14",
      "14"
    ],
    "viento": [
      {
        "@attributes": {
          "periodo": "00-24"
        },
        "direccion": "SO",
        "velocidad": "10"
      },
      {
        "@attributes": {
          "periodo": "00-12"
        },
        "direccion": "NE",
        "velocidad": "10"
      },
      {
        "@attributes": {
```

```
          "periodo": "12-24"
        },
        "direccion": "N",
        "velocidad": "15"
      }
    ],
    "racha_max": [
      {
        "@attributes": {
          "periodo": "00-24"
        }
      },
      {
        "@attributes": {
          "periodo": "00-12"
        }
      },
      {
        "@attributes": {
          "periodo": "12-24"
        }
      }
    ],
    "temperatura": {
      "maxima": "33",
      "minima": "17"
    },
    "sens_termica": {
      "maxima": "33",
      "minima": "17"
    },
    "humedad_relativa": {
      "maxima": "95",
      "minima": "45"
    },
    "uv_max": "9"
  },
  {
    "@attributes": {
      "fecha": "2023-07-10"
    },
```

```
    "prob_precipitacion": "40",
    "cota_nieve_prov": {

    },
    "estado_cielo": "13",
    "viento": {
      "direccion": "N",
      "velocidad": "5"
    },
    "racha_max": {

    },
    "temperatura": {
      "maxima": "30",
      "minima": "17"
    },
    "sens_termica": {
      "maxima": "30",
      "minima": "17"
    },
    "humedad_relativa": {
      "maxima": "100",
      "minima": "40"
    },
    "uv_max": "9"
  },
  {
    "@attributes": {
      "fecha": "2023-07-11"
    },
    "prob_precipitacion": "30",
    "cota_nieve_prov": {

    },
    "estado_cielo": "13",
    "viento": {
      "direccion": "N",
      "velocidad": "10"
    },
    "racha_max": {
```

```
      },
      "temperatura": {
        "maxima": "26",
        "minima": "16"
      },
      "sens_termica": {
        "maxima": "26",
        "minima": "16"
      },
      "humedad_relativa": {
        "maxima": "95",
        "minima": "65"
      }
    },
    {
      "@attributes": {
        "fecha": "2023-07-12"
      },
      "prob_precipitacion": "30",
      "cota_nieve_prov": {

      },
      "estado_cielo": "15",
      "viento": {
        "direccion": "N",
        "velocidad": "10"
      },
      "racha_max": {

      },
      "temperatura": {
        "maxima": "24",
        "minima": "15"
      },
      "sens_termica": {
        "maxima": "24",
        "minima": "15"
      },
      "humedad_relativa": {
        "maxima": "100",
        "minima": "65"
```

```
      }
    }
  ],
  "breadcrumb": [
    {
      "name": "Provincias",
      "url": "/provincias",
      "title": "El tiempo | Lista de provincias"
    },
    {
      "name": "Araba/Álava",
      "url": "/provincias/01",
      "title": "El tiempo en la provincia de Araba/Álava"
    },
    {
      "name": "Municipios",
      "url": "/provincias/01/municipios",
      "title": "El tiempo | Lista de municipios de Araba/Álava"
    },
    {
      "name": "Alegría-Dulantzi",
      "url": "/provincias/01/municipios/01010",
      "title": "El tiempo en Alegría-Dulantzi"
    }
  ]
}
```