



**Politécnico
de Viseu**

Escola Superior
de Tecnologia
e Gestão de Viseu

Aplicação Cross-Platform com Módulo de Machine Learning para Recomendação de Vinhos

Davi Fernandes Silva Souza

Relatório de Estágio

Curso de Tecnologias de Design e Multimedia

Fevereiro de 2025



**Politécnico
de Viseu**

Escola Superior
de Tecnologia
e Gestão de Viseu

Aplicação *Cross-Platform* com Módulo de *Machine Learning* para Recomendação de Vinhos

Davi Fernandes Silva Souza

Realizado na empresa, Softinsa (IBM)

Relatório de Estágio

Curso de Licenciatura em Tecnologias de Design e Multimédia

Trabalho efetuado sob a orientação de

Paulo Rogério Perfeito Tome

Vanessa Madeira (Softinsa)

Fevereiro de 2025

Agradecimentos

Gostaria de expressar o meu sincero agradecimento à Softinsa por me ter acolhido no âmbito deste estágio curricular, proporcionando-me um ambiente de trabalho estimulante, desafiante e alinhado com os meus interesses académicos e profissionais.

Um agradecimento especial à orientadora de estágio na empresa, Vanessa Madeira, pela constante motivação, disponibilidade e confiança nas minhas capacidades. A sua orientação foi fundamental para o sucesso deste projeto, não apenas pelos conselhos técnicos e estratégicos, mas também por me conceder a liberdade de trabalhar de forma autónoma, adaptando o plano de trabalho às dificuldades reais que foram surgindo ao longo do processo.

Agradeço também ao professor Paulo Rogério Tomé, orientador pela Escola Superior de Tecnologia e Gestão de Viseu (ESTGV), pelo acompanhamento atento durante a elaboração deste relatório. As suas sugestões e observações foram cruciais para a estruturação do trabalho e para a consolidação do percurso realizado.

Por fim, agradeço à ESTGV pela oportunidade de realizar este estágio curricular, que me permitiu crescer não só a nível técnico e académico, mas também a nível pessoal. Esta experiência foi essencial para consolidar competências, enfrentar desafios reais e preparar o próximo passo na minha vida profissional.

Resumo

Este relatório descreve o desenvolvimento da aplicação multiplataforma *ChooseWINE4me*, uma solução digital que permite aos utilizadores encontrar vinhos adequados às suas preferências, localização e histórico de interações. O projeto envolveu o design de interfaces com base em protótipos Figma, a implementação de autenticação com Firebase, a criação de uma base de dados em MongoDB, e o desenvolvimento de um *backend* em Node.js com Express. Complementarmente, foi construído um micro serviço inteligente em FastAPI que recorre a técnicas de filtragem colaborativa e *clustering*, potenciando um sistema de recomendação personalizado e eficiente.

O trabalho foi desenvolvido na Unidade Curricular de Projeto do curso de Licenciatura em Tecnologias e Design de Multimédia, na empresa Softinsa (IBM), no ano letivo de 2024/2025. A entidade proporcionou um ambiente de trabalho ágil e colaborativo, onde foi possível aplicar e aprofundar conhecimentos em design, programação e integração de sistemas. Este estágio permitiu consolidar competências técnicas e metodológicas, assim como desenvolver autonomia, sentido crítico e capacidade de adaptação a contextos reais de desenvolvimento.

Palavras-chave: recomendação inteligente; aplicação multiplataforma; Flutter; Node.js; FastAPI; MongoDB, Firebase

Abstract

This report describes the development of *ChooseWINE4me*, a cross-platform application that helps users discover wines tailored to their preferences, location, and interaction history. The project involved designing user interfaces based on Figma prototypes, implementing authentication with Firebase, creating a MongoDB database, and developing a Node.js backend with Express. Additionally, an intelligent microservice was built using FastAPI and machine learning techniques such as collaborative filtering and clustering, enabling a personalised and efficient wine recommendation system.

The work was carried out during the curricular class called “Project” of the Bachelor's Degree in Multimedia and Design Technologies, at the company Softinsa (IBM), in the academic year 2024/2025. The host entity provided an agile and collaborative environment, allowing the application and enhancement of skills in design, programming, and systems integration. This internship contributed to the consolidation of technical and methodological competencies, as well as the development of autonomy, critical thinking, and adaptability in real-world development contexts.

Keywords: intelligent recommendation; cross-platform application; Flutter; Node.js; FastAPI; MongoDB; Firebase;

Índice

| | | |
|-----------|---|-----------|
| 1. | INTRODUÇÃO | 1 |
| 1.1. | CONTEXTUALIZAÇÃO E MOTIVAÇÃO | 1 |
| 1.2. | DESCRIÇÃO DA ENTIDADE DE ACOLHIMENTO | 1 |
| 1.3. | OBJETIVOS E RESULTADOS ESPERADOS | 2 |
| 1.4. | PLANO DE TRABALHO | 2 |
| 1.5. | ORGANIZAÇÃO DO RELATÓRIO | 4 |
| 2. | ESTADO DE ARTE | 5 |
| 2.1. | APLICAÇÕES DE RECOMENDAÇÃO DE VINHOS..... | 5 |
| 2.2. | SISTEMAS DE RECOMENDAÇÃO | 7 |
| 2.3. | TECNOLOGIAS UTILIZADAS EM PROJETOS SEMELHANTES | 9 |
| 2.4. | EXPERIÊNCIA DO UTILIZADOR E PADRÕES DE INTERAÇÃO | 11 |
| 3. | METODOLOGIAS, TECNOLOGIAS E FERRAMENTAS UTILIZADAS | 13 |
| 3.1. | DESCRIÇÃO DE METODOLOGIAS | 13 |
| 3.2. | FERRAMENTAS | 14 |
| 3.3. | TECNOLOGIAS | 16 |
| 4. | ATIVIDADES INICIAS DESENVOLVIDAS | 22 |
| 4.1. | MIGRAÇÃO PARA FLUTTER | 24 |
| 4.2. | PROTOTIPAGEM | 26 |
| 5. | ARQUITETURA DA PLATAFORMA..... | 27 |
| 5.1. | <i>BACKEND</i> | 31 |
| 5.2. | MODULO <i>SCRAPPER</i> COM 'API-VIVINO' | 43 |
| 5.3. | MICRO SERVIÇO DE <i>MACHINE LEARNING</i> | 47 |
| 5.4. | <i>FRONTEND</i> | 50 |
| 6. | RESULTADOS OBTIDOS | 57 |
| 6.1. | FLUXO LOGIN | 58 |
| 6.2. | PÁGINA DE INSERÇÃO DA MORADA | 60 |
| 6.3. | PÁGINA INICIAL | 60 |
| 6.4. | PÁGINA PESQUISA | 62 |
| 6.5. | PÁGINA DE DETALHES DO VINHO | 64 |
| 6.6. | PÁGINA DE RECOMENDAÇÕES | 66 |
| 6.7. | PÁGINA DE FAVORITOS | 67 |
| 7. | CONCLUSÃO | 68 |
| 7.1. | PRINCIPAIS RESULTADOS E CONTRIBUIÇÕES | 69 |

| | | |
|---|-----------------------|-----------|
| 7.2. | MELHORIAS..... | 70 |
| 7.3. | TRABALHO FUTURO | 71 |
| 7.4. | AUTORREFLEXÃO | 72 |
| REFERÊNCIAS BIBLIOGRÁFICAS | | 73 |
| 8. | ANEXO A..... | 75 |
| 9. | ANEXO B | 84 |
| 10. | ANEXO C..... | 87 |
| 11. | ANEXO D | 91 |

Índice de Figuras

| | |
|---|----|
| Figura 4-1 - Exercício com WineAPI..... | 23 |
| Figura 4-2 – Configuração da EAS DEV e GoogleCloud..... | 24 |
| Figura 4-3 – Ecras de login na build React Native..... | 25 |
| Figura 4-4 - Ecrãs do Protótipo | 27 |
| Figura 5-1 – Esquema de Arquitetura da Aplicação..... | 29 |
| Figura 5-2- Backend - Estrutura de Ficheiros | 33 |
| Figura 5-3- Backend - Modelo Vinho | 34 |
| Figura 5-4- Backend – wineController..... | 35 |
| Figura 5-5- Importação de Rotas em appRoutes.js..... | 37 |
| Figura 5-6 - atuhmiddleware, errormiddleware..... | 38 |
| Figura 5-7 - server.js e database.js | 39 |
| Figura 5-8 - Mensagem de sucesso ao iniciar o servidor | 40 |
| Figura 5-9 - Tabelas criadas com sucesso no MongoDB | 41 |
| Figura 5-10 – Post a Adresses | 42 |
| Figura 5-11 – Address inserido no MongoDB | 43 |
| Figura 5-12 – Json eficaz para pesquisa em VivinoAPi da Apify..... | 44 |
| Figura 5-13 – Json Resultante do Script de extração e formatação..... | 45 |
| Figura 5-14- Login Multiplataforma feito pela Web..... | 54 |
| Figura 5-15 - Login Multiplataforma feito pelo Mobile..... | 54 |
| Figura 6-1 - Ecrãs de Login e Registro..... | 59 |
| Figura 6-2 - Home Page e Widgets | 61 |
| Figura 6-3 - Página de Resultados..... | 63 |
| Figura 6-4 - Página de detalhes do Vinho | 64 |
| Figura 6-5 - Resultados das Recomendações | 66 |
| Figura 6-6 - Página dos Favoritos..... | 68 |

Lista de abreviaturas

| Sigla | Significado |
|-------|-------------|
|-------|-------------|

| | |
|-----|---|
| API | Interface de Programação de Aplicações (<i>Application Programming Interface</i>) |
|-----|---|

| | |
|-----|--|
| CLI | Interface de Linha de Comandos (<i>Command Line Interface</i>) |
|-----|--|

| | |
|-----|--------------------------|
| FCM | Firebase Cloud Messaging |
|-----|--------------------------|

| | |
|-----|----------------|
| JWT | JSON Web Token |
|-----|----------------|

| | |
|-----|---|
| MVC | Modelo-Vista-Controlador (<i>Model-View-Controller</i>) |
|-----|---|

| | |
|-------|--|
| NoSQL | Not Only SQL (<i>Base de dados não relacional</i>) |
|-------|--|

| | |
|-----|--|
| SDK | Kit de Desenvolvimento de Software (<i>Software Development Kit</i>) |
|-----|--|

| | |
|-----|--|
| SPA | Aplicação de Página Única (<i>Single Page Application</i>) |
|-----|--|

| | |
|----|---|
| UI | Interface do Utilizador (<i>User Interface</i>) |
|----|---|

| | |
|-----|--|
| URL | Localizador Uniforme de Recursos (<i>Uniform Resource Locator</i>) |
|-----|--|

| | |
|----|--|
| UX | Experiência do Utilizador (<i>User Experience</i>) |
|----|--|

Glossário

admin – Papel de utilizador com permissões administrativas dentro da aplicação.

API – Interface de Programação de Aplicações que permite a comunicação entre diferentes sistemas.

authenticator – Componente responsável pela verificação da identidade de um utilizador no processo de login.

backend – Parte da aplicação responsável pela lógica, base de dados e regras de negócio, executada no servidor.

body – Parte principal de um pedido ou resposta HTTP que transporta os dados a serem enviados.

clean architecture – Arquitetura de software que promove a separação de responsabilidades em camadas independentes para facilitar a manutenção e escalabilidade.

cliente – Papel de utilizador com acesso limitado às funcionalidades da aplicação.

cloud function – Função executada num ambiente de computação em nuvem, geralmente utilizada para tarefas backend sem necessidade de servidor dedicado.

cluster – Conjunto de servidores ou instâncias que trabalham em conjunto para distribuir dados e tarefas (ex. MongoDB Atlas).

controller – Componente que gere a lógica de uma funcionalidade, servindo de ponte entre a interface e os dados.

dataset – Conjunto de dados estruturados, utilizado para treino ou teste de algoritmos de machine learning.

database schema – Estrutura organizacional de uma base de dados que define tabelas, relações e tipos de dados.

endpoint – URL específica de uma API que permite a execução de operações como criação, leitura ou actualização de dados.

feature – Atributo ou variável usada como entrada para treinar um modelo de machine learning.

fetch – Método usado para obter dados de um servidor numa aplicação web ou móvel.

filter – Critério de selecção aplicado aos dados para restringir os resultados com base em determinadas condições.

frontend – Parte visual da aplicação com a qual o utilizador interage diretamente.

hardcoded – Valor definido diretamente no código-fonte, sem permitir alteração dinâmica durante a execução.

hash – Representação encriptada de um valor, usada para garantir segurança, especialmente em palavras-passe.

header – Informação adicional enviada no início de uma mensagem HTTP (como tipo de conteúdo ou token de autenticação).

JSON – Formato leve e baseado em texto para troca de dados estruturados entre aplicações.

label – Valor esperado ou variável de saída usada para treinar modelos de machine learning supervisionado.

middleware – Camada de software que atua entre o frontend e o backend, responsável por tratar pedidos e respostas.

mock – Simulação de um componente real, usada para testar funcionalidades sem depender do sistema completo.

model – Estrutura que representa os dados e lógica da aplicação ou um modelo matemático em machine learning.

payload – Conjunto de dados transportado no corpo de um pedido ou resposta HTTP.

pipeline – Conjunto sequencial de etapas de tratamento de dados e treino de modelos em machine learning.

provider – Padrão de gestão de estado em Flutter que permite partilhar e gerir dados de forma eficiente entre widgets.

query – Pedido de informação feito a uma base de dados com critérios definidos.

request – Pedido feito por um cliente (ex: navegador ou aplicação) a um servidor para obter ou enviar dados.

response – Resposta devolvida pelo servidor a um pedido, podendo conter dados, confirmações ou mensagens de erro.

router – Sistema de gestão da navegação entre ecrãs ou rotas numa aplicação.

scaffold – Estrutura base em Flutter usada para organizar visualmente um ecrã, incluindo elementos padrão como barras e menus.

session – Informação temporária associada a um utilizador durante a sua permanência numa aplicação, normalmente usada para autenticação.

state – Conjunto de dados que representa a situação atual de um componente ou aplicação, podendo ser alterado dinamicamente.

toast – Mensagem temporária que aparece no ecrã para informar o utilizador sobre o resultado de uma ação.

token – Cadeia de caracteres usada para autenticação e verificação da identidade do utilizador.

view – Componente de interface responsável por apresentar dados ao utilizador, geralmente associado ao padrão MVC.

widget – Elemento visual ou funcional de interface numa aplicação Flutter.

1. Introdução

O desenvolvimento de aplicações inteligentes tem desempenhado um papel fundamental na personalização da experiência do utilizador. No contexto dos vinhos, a grande variedade de opções pode tornar a escolha desafiadora, levando à necessidade de um sistema que auxilie nesse processo. A aplicação ChooseWINE4me foi concebida para recomendar vinhos com base nas preferências dos utilizadores, utilizando inteligência artificial para tornar as sugestões mais precisas e relevantes.

A solução foi construída com React Native e Expo no *frontend*, garantindo uma experiência fluida e multiplataforma, enquanto o backend em Node.js assegura eficiência na comunicação com a base de dados. O sistema de autenticação foi implementado com Firebase (plataforma de desenvolvimento da Google) proporcionando um login seguro e integrado. Além disso, foram aplicadas técnicas de machine learning com Scikit-learn, permitindo que a aplicação aprenda e se adapte ao comportamento dos utilizadores ao longo do tempo. Este projeto foi desenvolvido no âmbito da unidade curricular “Projeto” permitindo a aplicação prática de tecnologias modernas.

Ao longo deste relatório, são apresentadas as tecnologias utilizadas, os desafios enfrentados e as soluções adotadas para garantir o desenvolvimento de uma aplicação eficiente e inovadora.

1.1. Contextualização e Motivação

A principal motivação é simplificar o processo de escolha diário dos vinhos, eliminando a indecisão e permitindo que a app escolha pelo utilizador. Realizando cruzamento de informação dos vinhos com os utilizadores consoante os preços, regiões e avaliações.

1.2. Descrição da Entidade de Acolhimento

Nesta secção vamos falar sobre a entidade de acolhimento que tornou possível este projeto.

A Softinsa, uma subsidiária da IBM, é especialista em serviços de consultoria, gestão e desenvolvimento de aplicações. Com 26 anos de história e experiência no mercado português, conta atualmente com uma equipa de mais de 700 profissionais. A Softinsa foi considerada, em novembro de 2024, pelo segundo ano consecutivo, um dos Melhores Lugares para Trabalhar

em Portugal segundo a Great Place to Work®. Obteve ainda, no mesmo ano, a certificação como Best Workplaces em Portugal e foi reconhecida como uma das 100 Melhores Empresas para Trabalhar na Europa pela Fortune Europe (Softinsa, 2024).

1.3. Objetivos e Resultados Esperados

O principal objetivo deste projeto é desenvolver uma aplicação inovadora que facilite a escolha de vinhos, proporcionando recomendações personalizadas com base nas preferências e interações dos utilizadores. A aplicação deverá oferecer uma experiência intuitiva e acessível, garantindo sugestões precisas que tornem o processo de decisão mais simples e eficiente.

Além da funcionalidade principal de recomendação, espera-se que o projeto reforce boas práticas de desenvolvimento, incluindo uma estrutura modular, escalabilidade e um design responsivo. O sistema deverá garantir segurança no acesso dos utilizadores, eficiência na gestão de dados e um desempenho otimizado para uma navegação fluida.

Como resultado, pretende-se obter uma aplicação funcional e bem estruturada, capaz de demonstrar o potencial da personalização na experiência do utilizador. Paralelamente, o projeto visa contribuir para o desenvolvimento de competências técnicas e metodológicas, permitindo a aplicação prática de conceitos fundamentais na construção de aplicações inteligentes.

1.4. Plano de Trabalho

O plano de trabalho seguido durante o estágio foi fornecido pela orientadora da entidade de acolhimento, **Vanessa Madeira**, e pode ser consultado no **Anexo A** deste relatório. Desde o início, foi deixado claro que este plano não devia ser seguido de forma rígida, mas sim como um guia adaptável, permitindo reorganizar tarefas e ajustar o percurso sempre que necessário, com o objetivo de obter os melhores resultados possíveis dentro do tema definido para o projeto.

O trabalho teve início com uma fase de exploração e experimentação de tecnologias, essencial para tomar decisões informadas quanto à arquitetura da aplicação, ferramentas de desenvolvimento e bibliotecas de suporte. Com base nessa análise preliminar, definiu-se um plano de desenvolvimento progressivo, que evoluiu naturalmente à medida que surgiam obstáculos técnicos e oportunidades de melhoria. A orientadora de estágio na empresa

demonstrou flexibilidade e confiança ao permitir que o plano fosse adaptado consoante as necessidades identificadas em cada etapa.

De forma geral, o desenvolvimento da aplicação seguiu a seguinte sequência:

- **Exploração inicial:**
 - Análise comparativa de aplicações semelhantes e levantamento de requisitos.
 - Estudo de viabilidade das tecnologias propostas (Flutter, Firebase, Node.js, MongoDB, FastAPI).
 - Definição da arquitetura modular da aplicação.
- **Implementação do backend:**
 - Criação das rotas principais em Node.js com Express.
 - Integração com MongoDB Atlas e definição dos modelos de dados.
 - Gestão de utilizadores, favoritos, avaliações e moradas.
- **Desenvolvimento do sistema de recomendação:**
 - Criação de um micro-serviço em Python com FastAPI.
 - Implementação de lógica de recomendação baseada em filtragem colaborativa e localização.
 - Comunicação entre backend principal e micro-serviço de IA.
- **Desenvolvimento do frontend mobile (Flutter):**
 - Construção das interfaces e navegação.
 - Integração com Firebase Authentication e backend.
 - Implementação das funcionalidades principais (login, registo, avaliação, favoritos, pesquisa e recomendações).
- **Testes, melhorias e fase de redesign visual:**

-
- Validação com a equipa técnica da empresa.
 - Ajustes visuais e funcionais com base no feedback recebido.
 - Refinamento da experiência do utilizador.

Este percurso permitiu garantir que a solução final fosse tecnicamente coesa, visualmente apelativa e funcionalmente completa, mantendo sempre o foco na **utilidade prática para o utilizador final** e na **escalabilidade da aplicação**.

Ao longo do relatório, cada etapa será analisada com mais detalhe, respeitando a ordem lógica de desenvolvimento e os objetivos definidos inicialmente no plano.

1.5. Organização do Relatório

Este relatório está estruturado de forma a acompanhar o percurso real do projeto, desde os objetivos iniciais até à apresentação dos resultados. Cada capítulo procura não só documentar as decisões e etapas técnicas, mas também justificar as escolhas feitas, refletindo uma abordagem prática e adaptativa.

O **Capítulo 3** apresenta as metodologias, tecnologias e ferramentas utilizadas, contextualizando o modo como cada uma contribuiu para o desenvolvimento da aplicação. No **Capítulo 3**, é feito o levantamento do estado da arte, com uma explicação do funcionamento geral da aplicação proposta e uma comparação com soluções semelhantes já existentes no mercado. Este capítulo destaca o que foi possível aproveitar de outras abordagens e onde o projeto procurou inovar ou adaptar-se às necessidades específicas do contexto.

O **Capítulo 4** descreve as atividades desenvolvidas fase de preparação e testes iniciais, incluindo experiências com diferentes tecnologias e a Interface de Programação de Aplicações (API), e os primeiros desafios enfrentados. Já o **Capítulo 5** aprofunda os **módulos da aplicação**, explicando como foram desenvolvidos, estruturados e como cada funcionalidade foi pensada e implementada dentro da arquitetura da aplicação.

O **Capítulo 6** apresenta o resultado da aplicação, com a descrição do fluxo de navegação entre as páginas principais e das funcionalidades já concluídas. Por fim, o **Capítulo 7** encerra o relatório com uma reflexão crítica sobre o trabalho desenvolvido, abordando os principais resultados, as dificuldades encontradas e possíveis melhorias futuras.

2. Estado de Arte

Este capítulo tem como objetivo contextualizar o projeto *ChooseWINE4me* no panorama atual das aplicações digitais dedicadas à recomendação de vinhos, analisando soluções já existentes, tecnologias habitualmente utilizadas e padrões de interação reconhecidos no setor. Trata-se de uma revisão crítica e seletiva que procura identificar as principais tendências, boas práticas e limitações observadas em projetos semelhantes, com o intuito de fundamentar as opções tomadas durante o desenvolvimento da aplicação.

Através da análise de aplicações como a Vivino, Wine-Searcher, Delectable ou Hello Vino, é possível compreender de que forma estas plataformas abordam a recomendação de vinhos, quais os modelos de negócio subjacentes e que tecnologias suportam o seu funcionamento. Além disso, a revisão da literatura e de fontes técnicas permite distinguir os diferentes tipos de sistemas de recomendação utilizados em contextos digitais — da filtragem colaborativa aos modelos híbridos — bem como perceber as implicações de cada abordagem no tipo de experiência oferecida ao utilizador.

Complementarmente, são analisadas as tecnologias mais comuns em projetos deste género, destacando-se a emergência de soluções multiplataforma com Flutter, o uso de serviços *cloud* como MongoDB Atlas e Firebase, e a integração de micro-serviços para processamento inteligente. Por fim, são explorados os padrões de interface e interação que mais contribuem para a eficácia e atratividade destas aplicações, com ênfase na personalização, confiança e clareza visual.

O objetivo não é apenas identificar o que já existe, mas compreender de que forma o projeto aqui apresentado se insere nesse ecossistema, que inovações propõe, e como se posiciona face ao estado da arte atual.

2.1. Aplicações de Recomendação de Vinhos

Com o crescimento do consumo de vinho a nível global e a crescente digitalização da experiência do consumidor, surgiram diversas aplicações móveis focadas na recomendação, avaliação e descoberta de vinhos. Estas aplicações não só ajudam o utilizador a fazer escolhas

mais informadas, como também promovem a aprendizagem e o contacto com novas referências enológicas. Nesta secção, analisam-se algumas das aplicações mais relevantes no domínio da recomendação de vinhos, identificando as suas principais funcionalidades, pontos fortes e limitações.

2.1.1. Vivino

A **Vivino** é, provavelmente, a aplicação de recomendação de vinhos mais conhecida e utilizada a nível mundial. Conta com milhões de utilizadores e uma base de dados extremamente vasta, composta por avaliações, preços, descrições e fotografias de vinhos.

As principais funcionalidades da Vivino incluem:

- Digitalização do rótulo do vinho para obter informações imediatas;
- Avaliação com estrelas e comentários;
- Sistema de recomendações personalizadas com base no histórico de consumo e preferências;
- Possibilidade de comprar vinhos diretamente através da aplicação;
- Rankings globais e regionais dos vinhos mais populares.

A Vivino aposta fortemente numa **comunidade ativa** de utilizadores, o que lhe permite afinar as recomendações com base em dados colaborativos. No entanto, apesar da sua robustez, a interface pode parecer excessivamente comercial, e o foco na compra pode limitar a neutralidade das sugestões.

2.1.2. Wine-Searcher

A **Wine-Searcher** é uma aplicação mais orientada para a **comparação de preços e localização de vinhos em lojas físicas e online**. O utilizador pode pesquisar vinhos pelo nome, filtrar por tipo ou região, e aceder a uma lista de vendedores com os respetivos preços e avaliações.

Entre as suas funcionalidades destacam-se:

- Ferramenta de busca por nome ou imagem do rótulo;

-
- Informações detalhadas sobre produtores e vinhos;
 - Listagens de fornecedores e preços atualizados;
 - Classificações agregadas a partir de fontes como Wine Spectator, Decanter e outras revistas especializadas.

Embora seja uma excelente ferramenta para consumidores informados, a Wine-Searcher não se destaca tanto ao nível da **experiência personalizada**. O sistema de recomendação é limitado, sendo mais uma ferramenta de consulta do que uma aplicação interativa ou orientada à descoberta.

2.1.3. Outras Aplicações Relevantes

Existem outras aplicações no mercado que merecem destaque, como:

- **Delectable**: aposta numa comunidade de especialistas e produtores. Permite avaliações visuais, comentários e integração com redes sociais. O design é mais artístico, mas o foco está em utilizadores mais experientes.
- **CellarTracker**: muito usada por colecionadores, foca-se na gestão de adegas pessoais e histórico de consumo. Menos intuitiva para o utilizador comum, mas extremamente detalhada.
- **Hello Vino**: apresenta sugestões com base em ocasiões e pratos, usando um tom mais informal e acessível, embora menos robusta em termos de base de dados.

Estas soluções mostram que **existe uma procura clara por ferramentas digitais que ajudem o consumidor a tomar decisões informadas no mundo do vinho**, seja através de recomendações, gestão de inventário ou simples comparação de preços. No entanto, muitas destas aplicações têm lacunas ao nível da **personalização contextualizada** (como a localização geográfica) e da **inteligência artificial adaptada ao perfil do utilizador** — aspetos centrais no projeto *ChooseWINE4me*.

2.2. Sistemas de Recomendação

Os sistemas de recomendação desempenham um papel essencial em diversas plataformas digitais, permitindo personalizar conteúdos, produtos ou sugestões com base nos dados e

comportamentos dos utilizadores. No contexto de aplicações de vinho, como a *ChooseWINE4me*, a sua função é apoiar o utilizador na descoberta de novos vinhos de acordo com os seus gostos, preferências e padrões de consumo.

2.2.1. Tipos de Sistemas de Recomendação

Existem três abordagens fundamentais:

- **Filtragem Colaborativa (*Collaborative Filtering*)**

Baseia-se na análise dos comportamentos e preferências de um conjunto de utilizadores. A lógica subjacente é que, se dois utilizadores demonstraram interesse por produtos semelhantes no passado, é provável que partilhem gostos semelhantes no futuro. Esta filtragem pode ser centrada no utilizador (*user-based*) ou no produto (*item-based*). Por exemplo, se dois utilizadores avaliaram positivamente os mesmos vinhos, o sistema pode recomendar ao segundo utilizador um vinho apreciado pelo primeiro, mas ainda desconhecido por ele.

- **Filtragem Baseada em Conteúdo (*Content-Based Filtering*)**

Este método recomenda produtos com base nas características dos próprios itens e nas preferências previamente demonstradas por um utilizador específico. No caso do vinho, isto implica analisar atributos como tipo, região, teor alcoólico ou classificações médias, e sugerir vinhos com propriedades semelhantes aos que o utilizador avaliou positivamente.

- **Sistemas Híbridos**

Combinam elementos das duas abordagens anteriores, com o objetivo de potenciar as suas vantagens e minimizar limitações. Esta solução é particularmente útil para resolver o problema do “*cold start*”, ou seja, a dificuldade em gerar recomendações para novos utilizadores ou novos produtos sem histórico suficiente.

2.2.2. Aplicações em Domínios Semelhantes

Sistemas de recomendação são amplamente utilizados em plataformas digitais com uma vasta oferta de conteúdos ou produtos. Exemplos típicos incluem serviços de *streaming* de vídeo e música, como Netflix ou Spotify, plataformas de comércio eletrónico como Amazon, e aplicações de recomendação de livros como Goodreads. Nestes contextos, os sistemas são capazes de aprender com os padrões de consumo, histórico de avaliações e interações com outros utilizadores para sugerir opções personalizadas.

A aplicação *ChooseWINE4me* adota uma estratégia centrada na filtragem colaborativa, com um reforço baseado na localização geográfica do utilizador. Esta decisão visa melhorar a relevância das recomendações nos estágios iniciais de uso e explorar as tendências regionais no consumo de vinho. O sistema de recomendação foi desenvolvido com recurso a algoritmos simples, mas eficazes, recorrendo à biblioteca Scikit-learn em Python.

2.3. Tecnologias Utilizadas em Projetos Semelhantes

Ao analisar aplicações de recomendação de vinhos já existentes no mercado, é possível identificar **padrões tecnológicos comuns** e compreender as vantagens e limitações das abordagens adotadas. Esta análise permitiu fundamentar as escolhas técnicas do projeto *ChooseWINE4me*, garantindo alinhamento com boas práticas e tendências atuais do desenvolvimento de aplicações inteligentes e multiplataforma.

2.3.1. Frontend e Experiência Multiplataforma

Aplicações como a **Vivino** e a **Delectable** foram desenvolvidas com interfaces ricas, fluídas e consistentes entre plataformas. No entanto, recorrem geralmente a abordagens nativas (desenvolvimento separado para Android e iOS), o que implica maior custo de manutenção e desenvolvimento.

Em alternativa, soluções mais recentes em projetos independentes ou académicos optam por *frameworks* **multiplataforma como Flutter ou React Native**. Flutter, em particular, tem vindo a destacar-se pela sua performance próxima do nativo e pela capacidade de manter uma única base de código. Esta abordagem foi adotada no projeto *ChooseWINE4me*, permitindo reduzir o tempo de desenvolvimento e garantir consistência visual entre plataformas.

2.3.2. Backend e APIs

Em projetos semelhantes que envolvem consultas dinâmicas e bases de dados extensas (como a **Wine-Searcher**), observa-se o uso de arquiteturas RESTful com **Node.js** ou **Django**, permitindo escalabilidade e facilidade de integração entre serviços.

O projeto *ChooseWINE4me* seguiu esta tendência ao adotar **Node.js com Express**, uma solução amplamente usada pela sua leveza e adaptabilidade em sistemas baseados em micro-serviços.

Esta escolha foi também influenciada pelo facto de muitas APIs públicas de vinho utilizarem JavaScript como base para integração.

2.3.3. Bases de Dados

Aplicações com grandes volumes de avaliações e interações entre utilizadores, como a **Vivino**, utilizam normalmente bases de dados escaláveis e com estrutura flexível. Embora não sejam divulgadas publicamente todas as soluções adotadas, estima-se o uso de bases Not Only SQL (NoSQL) ou híbridas, capazes de lidar com esquemas dinâmicos e operações de leitura/escrita em tempo real.

Neste contexto, a utilização de **MongoDB** revelou-se adequada para o projeto em causa, pela sua estrutura orientada a documentos e integração natural com tecnologias JavaScript. O serviço **MongoDB Atlas** permite ainda manter uma infraestrutura fiável em *cloud*, algo comum em soluções modernas.

2.3.4. Autenticação e Serviços Externos

A maioria das aplicações modernas incorpora autenticação federada com **contas Google ou redes sociais**, facilitando o registo e reduzindo barreiras à entrada. Este padrão é evidente em aplicações como **Hello Vino**, que privilegiam o acesso rápido e experiências personalizadas.

Neste projeto, foi integrado o **Firebase Authentication**, alinhando-se com esta prática. Além disso, o uso de **Firebase Analytics** permite recolher métricas de uso, uma funcionalidade comum em aplicações com foco na experiência do utilizador e melhoria contínua.

2.3.5. Sistemas de Recomendação

Em aplicações comerciais como a Vivino, embora os algoritmos usados não sejam públicos, sabe-se que envolvem modelos avançados de *machine learning*, combinando filtragem colaborativa, preferências locais e dados de mercado.

Para uma abordagem mais leve e adaptada ao contexto académico, o projeto *ChooseWINE4me* adotou uma arquitetura modular com um micro-serviço de recomendação em **Python**, usando **FastAPI** e **Scikit-learn**. Esta solução segue o modelo adotado em várias provas de conceito e protótipos de recomendação no meio académico, permitindo experimentar estratégias híbridas com custo computacional reduzido.

2.4. Experiência do Utilizador e Padrões de Interação

A experiência do utilizador (UX) é um fator determinante no sucesso de qualquer aplicação orientada ao consumidor, particularmente em domínios como o enoturismo e a descoberta de vinhos, onde o utilizador valoriza a confiança, a estética e a facilidade de uso. A análise de aplicações existentes revelou **padrões de interação consolidados**, bem como lacunas que a aplicação *ChooseWINE4me* procurou colmatar.

2.4.1. Interface e Usabilidade

Aplicações como a **Vivino**, a **Delectable** ou a **Hello Vino** partilham elementos comuns de design que têm sido progressivamente adotados como norma:

- Utilização de **cartões verticais** para apresentar vinhos com imagem, nome, tipo e avaliação;
- Navegação por **tabs inferiores**, facilitando o acesso às secções principais;
- Ícones visuais para funcionalidades intuitivas (como favoritar, avaliar ou partilhar);
- Reforço visual de conteúdos relevantes, como rankings, medalhas ou etiquetas editoriais.

No entanto, a complexidade crescente de algumas destas plataformas tem vindo a gerar **interfaces sobrecarregadas**, com excesso de informação ou funcionalidades escondidas em menus secundários, o que pode prejudicar a navegação.

No desenvolvimento da *ChooseWINE4me*, optou-se por uma abordagem mais minimalista e funcional, com **prioridade à clareza e acessibilidade**. A interface foi concebida com base em boas práticas de usabilidade móvel, como:

- Navegação fluida com poucos toques;
- Destaque visual para as ações principais (avaliar, favoritar, explorar);
- Feedback visual imediato após interações.

A escolha da *framework* Flutter permitiu ainda garantir **transições suaves e responsividade nativa**, contribuindo para uma experiência consistente entre dispositivos.

2.4.2. Personalização e Confiança

Outro fator crítico nas aplicações de recomendação é a **confiança nas sugestões apresentadas**. A transparência dos critérios de recomendação, a apresentação clara das avaliações e a possibilidade de aceder a comentários reais de utilizadores são elementos valorizados. A Vivino, por exemplo, destaca-se por permitir ver comentários detalhados de outros consumidores, o que aumenta a credibilidade das sugestões.

Neste projeto, a personalização foi construída com base no comportamento do utilizador (favoritos, avaliações, histórico), mas também reforçada pela sua **localização geográfica**, um aspeto raramente explorado de forma explícita em apps concorrentes. A possibilidade de visualizar recomendações associadas à região onde o utilizador vive ou compra vinho reforça a **relevância local** das sugestões, promovendo uma relação de maior confiança com o sistema.

Foram ainda tomadas medidas para manter a interface transparente e compreensível, com separação clara entre recomendações baseadas em preferências e vinhos populares ou genéricos.

3. Metodologias, Tecnologias e Ferramentas Utilizadas

Este capítulo vai abordar as metodologias, isto é, os principais métodos de trabalho e organização que guiaram o desenvolvimento do projeto. Vamos explorar também as tecnologias essenciais e secundárias que tornaram possível a construção da aplicação, bem como as ferramentas utilizadas ao longo de todo o processo.

Para garantir a construção da aplicação dentro do prazo estipulado e maximizar os resultados, foi necessário seguir um conjunto de boas práticas. Isso incluiu a escolha cuidadosa das ferramentas para cada fase do desenvolvimento e o domínio adequado de como utilizá-las. Também foi essencial entender as tecnologias disponíveis que poderiam ajudar a alcançar um resultado mais robusto e rápido, como APIs ou bibliotecas open-source, bem como selecionar as linguagens de programação que oferecessem o melhor suporte para o projeto. Além disso, foi importante antecipar possíveis problemas ou incompatibilidades. Portanto, o trabalho nesta etapa do projeto incluiu pesquisa e preparação, o que envolveu entender profundamente o produto desejado e realizar pesquisas para definir a melhor estrutura a ser desenvolvida. Nos anexos deste relatório, podem ser encontrados documentos de pesquisa realizados durante esta fase do projeto, incluindo análises de tecnologias como React Native, comparações entre o uso de Expo e CLI, pesquisas sobre APIs da Vivino, entre outras.

Para alcançar os objetivos propostos, foi crucial reunir um conjunto de métodos de trabalho a serem seguidos, junto com ferramentas e tecnologias essenciais para o desenvolvimento. Abaixo, detalharei cada uma dessas tecnologias e explicarei como foram utilizadas ao longo do projeto.

3.1. Descrição de metodologias

As metodologias utilizadas para a execução deste trabalho seguiram o plano de trabalho inicial, aplicando conceitos de boas práticas que foram adquiridos durante o curso e ao longo da minha carreira profissional. Estas práticas foram implementadas com base em pesquisa contínua e desenvolvimento progressivo ao longo de cada módulo da aplicação. Em cada fase, foram adotadas técnicas específicas que serão detalhadas no Capítulo 5 – Módulos da Aplicação. No

entanto, há conceitos fundamentais que foram aplicados de forma transversal em todos os módulos, garantindo consistência e qualidade no desenvolvimento.

Entre os principais conceitos metodológicos que guiaram o projeto, destaco:

- **Segurança e proteção de dados sensíveis:** Durante o desenvolvimento, foram implementadas medidas rigorosas para proteger dados sensíveis, como endereços IP, dados do *backend*, chaves de encriptação e variáveis de ambiente. Isso assegurou que a aplicação fosse protegida contra possíveis vulnerabilidades e que a integridade dos dados fosse preservada em todas as fases do projeto.
- **Código modular e organizado:** A ênfase foi na criação de um código limpo, estruturado de forma modular e bem organizado, com arquivos específicos para cada funcionalidade, garantindo uma base sólida e de fácil manutenção. Esta abordagem também evitou redundâncias e a presença de código obsoleto, o que torna o projeto escalável e flexível para futuras modificações.
- **Documentação e comentários no código:** Todo o código foi comentado detalhadamente, para garantir que o processo de depuração fosse eficiente e para facilitar a compreensão, caso o projeto fosse transferido para outro programador no futuro. Essa documentação também facilita a realização de ajustes e melhorias no futuro.

O desenvolvimento seguiu uma metodologia ágil, com iterações rápidas e revisões contínuas dos requisitos. A cada ciclo de desenvolvimento, priorizava-se o feedback dos testes e ajustes necessários, garantindo que a aplicação fosse constantemente aprimorada. A flexibilidade e adaptabilidade do processo permitiram ajustes rápidos nos requisitos e funcionalidades com base no feedback dos utilizadores e nas iterações do desenvolvimento, o que é uma característica essencial da metodologia ágil.

Esses princípios metodológicos foram essenciais para garantir que o projeto fosse desenvolvido de forma profissional, priorizando a segurança, a organização e a eficiência do código ao longo de todas as fases do desenvolvimento.

3.2. Ferramentas

A seguir, detalho cada uma das ferramentas utilizadas e o seu papel no desenvolvimento do projeto.

GitHub

O GitHub é uma plataforma de hospedagem de código-fonte e controle de versões, amplamente utilizada para facilitar o desenvolvimento colaborativo. Ele permite que os programadores armazenem, gerenciem e acompanhem o histórico de mudanças no código (GitHub, Inc., 2025).

As Principais funcionalidades usadas no projeto:

- Repositórios e *branches* – Organização do código em diferentes fases de desenvolvimento.
- *Pull Requests* – Revisão e integração de código, garantindo qualidade antes da implementação definitiva.
- Histórico de alterações – Possibilidade de reverter mudanças caso necessário.

Foi realizada uma pesquisa aprofundada sobre controle de versões no GitHub, e suas melhores práticas estão detalhadas no **Anexo B**

VsCode

O Visual Studio Code (VSCode) é um editor de código-fonte amplamente utilizado por desenvolvedores devido à sua flexibilidade e extensibilidade (Microsoft Corporation, 2025). No projeto ChooseWINE4me, o VSCode foi escolhido como editor principal por sua capacidade de suportar várias bibliotecas e frameworks, como Node.js e React Native, essenciais para o desenvolvimento da aplicação. Tanto o *backend*, desenvolvido em JavaScript utilizando Node.js e Axios, quanto o *frontend*, implementado com React Native, foram beneficiados por extensões como Prettier e ESLint, que garantiram a consistência do código e mantiveram padrões de qualidade.

As funcionalidades de depuração do VSCode foram essenciais para identificar e corrigir erros de forma eficiente. A possibilidade de configurar ambientes de desenvolvimento personalizados contribuiu significativamente para a produtividade dos desenvolvedores. A interface intuitiva e as extensões colaborativas, como o Live Share, facilitaram a colaboração em tempo real entre os membros da equipe, tornando o desenvolvimento do ChooseWINE4me mais ágil e eficiente

MongoDB

O MongoDB é uma base de dados NoSQL amplamente utilizada para armazenar e gerenciar grandes volumes de dados de forma eficiente e escalável (MongoDB, Inc., 2025).

No projeto ChooseWINE4me, o MongoDB foi escolhido devido à sua flexibilidade na modelagem de dados e sua capacidade de lidar com grandes quantidades de informações não estruturadas ou semi-estruturadas, como as preferências dos utilizadores e as características dos vinhos.

Uma das principais vantagens do MongoDB é o seu modelo de dados orientado a documentos, que permite armazenar dados em formato JSON (ou BSON). Isso facilita a integração com aplicações baseadas em JavaScript, como o *backend* desenvolvido com Node.js. A escalabilidade horizontal do MongoDB permite adicionar novos servidores para aumentar a capacidade de armazenamento e processamento, garantindo que a aplicação possa crescer conforme necessário.

Além disso, o MongoDB oferece uma série de funcionalidades avançadas, como a indexação eficiente, a agregação de dados e a replicação, que melhoram o desempenho e a disponibilidade dos dados. Essas características foram essenciais para garantir que o ChooseWINE4me pudesse fornecer recomendações personalizadas de vinhos de maneira rápida e confiável, atendendo às expectativas dos utilizadores.

Postman

O Postman é uma ferramenta amplamente utilizada para testar e documentar APIs (Interface de Programação de Aplicações) (Postman, Inc., 2025).

Durante o desenvolvimento do projeto ChooseWINE4me, o Postman desempenhou um papel crucial na criação, teste e depuração das APIs desenvolvidas com Node.js. A facilidade de uso do Postman permitiu que criasse rapidamente requisições HTTP, como GET, POST, PUT e DELETE, verificando as respostas e garantindo que a comunicação entre o *frontend* e o *backend* funcionasse conforme esperado.

3.3. Tecnologias

A seguir, detalho cada uma das Tecnologias utilizadas e o seu papel no desenvolvimento do projeto.

3.3.1. React-Native

React Native é um *framework open-source* desenvolvido pela Meta que permite criar aplicações móveis nativas utilizando JavaScript e React. Uma das suas principais vantagens é a partilha de código entre plataformas (Android e iOS), mantendo um desempenho próximo do nativo. A arquitetura baseada em componentes reutilizáveis e a vasta comunidade tornam-no uma opção atrativa para prototipagem rápida e desenvolvimento multiplataforma (Meta Platforms, Inc., 2025).

Na fase inicial do ChooseWINE4me, foi utilizada esta tecnologia para construir uma versão funcional da interface e realizar testes de integração com APIs externas. No entanto, surgiram dificuldades relacionadas com a configuração de autenticação via Firebase, especialmente na versão web. Isso motivou a migração posterior para Flutter, que ofereceu melhor integração para o login multiplataforma.

3.3.2. Expo

Expo é uma plataforma open-source criada para acelerar o desenvolvimento de aplicações com React Native. Fornece um conjunto de ferramentas e serviços que permitem testar, compilar e distribuir aplicações de forma simples, sem a necessidade de configurar nativamente Android Studio ou Xcode. Destaca-se pelo seu ambiente pré-configurado, suporte a atualizações *over-the-air* (OTA) e integração com APIs nativas através de módulos geridos (Expo, 2025).

No início do projeto, surgiu uma dúvida entre seguir com Expo ou utilizar a abordagem React Native CLI, que oferece mais controlo, mas requer configuração nativa detalhada. Para tomar uma decisão fundamentada, foi realizada uma pesquisa comparativa entre ambas as abordagens, cujas conclusões se encontram no **Anexo C** deste relatório. Essa análise teve em conta critérios como facilidade de uso, tempo de *setup*, compatibilidade com bibliotecas e controlo sobre funcionalidades nativas.

Com base nos resultados, optou-se por utilizar Expo durante a fase de testes e prototipagem rápida, devido à sua simplicidade e velocidade. Essa escolha facilitou a validação de conceitos e funcionalidades antes de avançar para a migração definitiva para Flutter.

Node.js

Node.js é um ambiente de execução JavaScript baseado no motor V8 do Chrome. Permite desenvolver aplicações de rede rápidas e escaláveis através de um modelo de operação assíncrono e orientado a eventos. É amplamente utilizado na criação de servidores *backend* modernos, APIs RESTful e microserviços, graças à sua leveza, eficiência e vasta comunidade de pacotes disponíveis via npm (OpenJS Foundation, 2025).

No ChooseWINE4me, Node.js foi a base do servidor *backend* responsável por expor os endpoints consumidos pela aplicação Flutter. A sua estrutura assíncrona revelou-se ideal para lidar com autenticação, validação de tokens, comunicação com a base de dados e envio de respostas ao *frontend* sem bloquear o fluxo da aplicação. O *backend* foi organizado de forma modular, com separação de controladores, *middlewares* e rotas, garantindo clareza e manutenção eficiente do código. A escolha por Node.js também facilitou a futura escalabilidade e integração com serviços externos.

Express.js

Express é um *framework* minimalista para aplicações Node.js, conhecido por simplificar a criação de APIs através de rotas claras, *middlewares* e organização de controladores

Foi utilizado para estruturar a lógica de *backend* da aplicação, separando rotas públicas e privadas, validando *JSON Web Token (JWT)* recebidos do *frontend* e implementando controladores como `authController.js`, `userController.js` e `wineController.js`. O Express permitiu organizar a lógica da API de forma limpa e escalável.

Axios

Axios é uma biblioteca JavaScript baseada em *Promises* que permite realizar chamadas HTTP tanto no browser como em ambientes Node.js. É conhecida pela sua sintaxe simples, suporte a interceptores, e capacidade de lidar com *headers*, *timeouts* e respostas JSON de forma intuitiva (Axios, 2025).

No ChooseWINE4me, Axios foi utilizado durante os testes iniciais em React Native e também em scripts de teste do *backend* Node.js. Serviu para validar *endpoints* da API, testar autenticação, e simular comportamentos do *frontend* antes de o Flutter estar operacional.

Firebase

Firebase é uma plataforma da Google que fornece uma variedade de serviços *backend* para

aplicações móveis e web. Entre os seus recursos mais utilizados estão: autenticação (com suporte a diversos métodos), base de dados em tempo real, armazenamento em nuvem, notificações *push* via *Firebase Cloud Messaging (FCM)*, e ferramentas de análise como o *Firebase Analytics*. Destaca-se pela sua integração nativa com *Flutter* e *React Native*, bem como pela sua escalabilidade e facilidade de configuração (*Google, 2025*).

No *ChooseWINE4me*, o *Firebase* foi utilizado principalmente para autenticação de utilizadores, suportando login via *Google* e email/senha em ambiente web e mobile. Após o login, o token gerado era enviado ao *backend*, onde era validado usando o *Firebase Admin Software Development Kit (Firebase SDK)*, permitindo manter uma sessão segura e estruturada. A plataforma foi configurada com variáveis de ambiente e regras de segurança para garantir a proteção dos dados sensíveis.

Além da autenticação, foi integrado o *Firebase Analytics* para recolher dados sobre o comportamento dos utilizadores. Através desta funcionalidade, foi possível acompanhar o tempo médio de sessão, identificar os botões mais pressionados, e compreender o fluxo de navegação mais comum dentro da aplicação. Estes dados revelaram-se valiosos para avaliar o impacto das funcionalidades implementadas e para apoiar futuras decisões de melhoria da experiência do utilizador.

api-vivino

A *Vivino API* não oficial é um *scraper* que permite obter dados do site *Vivino* sem depender de uma *API* pública oficial. Apesar de útil, apresenta riscos como quebras frequentes e limitações legais relacionadas com termos de uso.

Durante a fase de testes, foi usada para explorar a viabilidade de importar automaticamente dados de vinhos. No entanto, devido à instabilidade e à falta de garantia de funcionamento, optou-se por criar uma base de dados própria com os dados essenciais para as recomendações.

Tensorflow

TensorFlow é uma biblioteca *open-source* desenvolvida pela *Google* para computação numérica e *machine learning*, com especial foco em *deep learning*. Suporta tanto redes neurais simples como arquiteturas complexas e pode ser usada em produção com alto desempenho (*TensorFlow Team, 2025*).

Embora não tenha sido utilizada na versão final da aplicação, foi considerada durante a fase de pesquisa como possível solução para uma recomendação baseada em redes neuronais, mas acabou por ser substituída por soluções mais leves com Scikit-learn.

Scikit-learn

Scikit-learn é uma biblioteca de *machine learning* em Python que oferece ferramentas simples e eficientes para análise preditiva, modelação estatística e algoritmos de classificação, regressão e clustering (Pedregosa et al., 2011).

No projeto, foi utilizada no microserviço de recomendação para criar um sistema baseado em filtragem colaborativa, onde o modelo NearestNeighbors sugere vinhos a partir de similaridade entre utilizadores. A integração com FastAPI permitiu disponibilizar o modelo via uma rota acessível ao *frontend*.

FastAPI

É um *framework* web moderno e leve para Python, projetado para a criação rápida de APIs com desempenho comparável ao Node.js. Utiliza *tipagem* forte baseada em Python 3.6+ e integra documentação automática com Swagger (Ramírez, 2025).

Foi a base do microserviço de recomendação inteligente do ChooseWINE4me. A escolha pelo FastAPI deveu-se à sua velocidade de resposta, facilidade de estruturação em rotas e integração direta com bibliotecas de *machine learning* como o Scikit-learn.

Dart

Dart é uma linguagem de programação desenvolvida pela Google, pensada para construir aplicações modernas e eficientes, especialmente em interfaces gráficas. É orientada a objetos, com *tipagem* opcional e forte suporte a programação assíncrona através de *async/await*. Tornou-se a linguagem principal para o desenvolvimento em Flutter, permitindo escrever código conciso, seguro e altamente performativo para aplicações multiplataforma (Dart Team, 2025).

No ChooseWINE4me, Dart foi utilizado como linguagem base para todo o desenvolvimento da aplicação em Flutter. A estrutura clara da linguagem facilitou a organização do código por módulos e a criação de serviços reutilizáveis, como o `api_service.dart` para chamadas HTTP e o `auth_service.dart` para autenticação. A sua sintaxe moderna e o suporte a *Future* e *Stream* foram fundamentais para lidar com operações assíncronas, como requisições à API, leitura de

tokens e persistência local. A utilização de Dart também contribuiu para um desempenho fluido e uma curva de aprendizagem rápida durante a migração de React Native para Flutter.

Flutter

Flutter é um *framework* de desenvolvimento de interfaces criado pela Google, que permite construir aplicações nativas com uma única base de código, suportando Android, iOS, Web e Desktop. Utiliza uma abordagem declarativa com uma linguagem própria (Dart) e um motor gráfico de alto desempenho (Skia). Destaca-se pela sua consistência visual, tempo de compilação rápido (*hot reload*) e facilidade na construção de interfaces responsivas (Google, 2025).

Após a decisão de abandonar o React Native, o ChooseWINE4me foi reconstruído com Flutter. Com esta tecnologia, foi possível criar uma aplicação responsiva e fluida para dispositivos móveis, com especial atenção ao design e reutilização de componentes. A estrutura em *widets* favoreceu a organização modular do código, facilitando a manutenção e o crescimento do projeto. A compatibilidade nativa com Firebase também contribuiu para uma gestão mais simples da autenticação e do estado da aplicação.

Dio

Dio é um cliente HTTP para Flutter e Dart, robusto e altamente personalizável. Suporta interceptores, cancelamento de pedidos, *timeout*, redirecionamento automático e muito mais. É uma das bibliotecas mais populares no ecossistema Flutter para comunicação com APIs REST (Flutter China, 2025).

Foi a biblioteca escolhida para lidar com as chamadas à API no *frontend* Flutter. Através de Dio, foi possível implementar de forma eficiente os pedidos de login, registo, consulta de vinhos, favoritos e submissão de avaliações, com tratamento de erros e inclusão automática de *headers* de autenticação (*token JWT*).

SharedPreferences

é uma biblioteca do ecossistema Flutter que permite guardar dados simples de forma persistente no dispositivo, como preferências ou *tokens*. É multiplataforma e amplamente utilizada para manter sessões de utilizador (Flutter Team, 2025).

Foi utilizada para armazenar o *token* JWT de sessão após login com sucesso, evitando que o utilizador precisasse fazer login novamente sempre que reabria a aplicação. A simplicidade da implementação tornou-a ideal para esta funcionalidade.

4. Atividades Inicias Desenvolvidas

A preparação e teste foram etapas fundamentais no desenvolvimento do projeto ChooseWINE4me, garantindo que todas as funcionalidades da aplicação fossem testadas de forma eficaz antes de sua implementação. A preparação envolveu uma análise detalhada dos requisitos da aplicação, o planeamento da estrutura de dados e a configuração de ferramentas essenciais para o desenvolvimento.

Entre as tarefas experimentais realizadas, uma das primeiras foi clonar o projeto GitHub 'api-vivino', tentando extrair vinhos e exibi-los em uma lista numa página React Native. Essa tarefa

serviu para testar a integração com APIs externas e validar se os dados poderiam ser processados e exibidos corretamente no *frontend*.

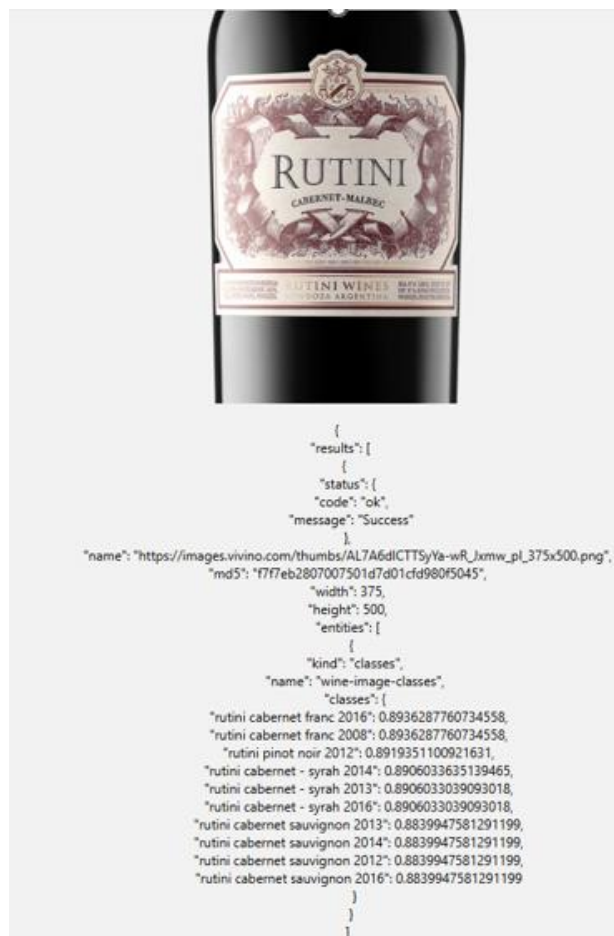


Figura 4-1 - Exercício com WineAPI

Na **Figura 4-1**, podemos observar o resultado de outro exercício realizado em React Native, que envolveu o uso de uma subscrição de período experimental da WineAPI. Esta API, voltada para o reconhecimento de imagens, foi utilizada para identificar vinhos a partir de uma foto. Na imagem, podemos ver a imagem utilizada e a resposta JSON fornecida pela API, evidenciando o sucesso da integração.

Essa fase de testes experimentais foi essencial para identificar possíveis inviabilidades no projeto e fazer ajustes no plano de trabalho antes de entrar em uma fase mais avançada do desenvolvimento. É importante destacar que o plano de trabalho não é algo fixo, mas sim uma base para iniciar o desenvolvimento, que pode e deve ser ajustada à medida que o projeto evolui.

Com esses testes realizados, foi possível confirmar a viabilidade de várias integrações e funcionalidades, ajudando a refinar o escopo do projeto. A seguir, será apresentado um

problema que surgiu um pouco mais à frente no trabalho. Após a conclusão do *backend*, foi feito um teste para realizar o login com o Google utilizando o Firebase, desenvolvido no contexto de autenticação e comunicação dos módulos da aplicação.

4.1. Migração para Flutter

Para a aplicação ChooseWINE4me que foi inicialmente prevista para react Native num *framework* Expo, deparei-me com obstáculos técnicos relevantes ao tentar implementar um sistema de autenticação Google e Firebase robusto e multiplataforma, sendo forçado a adotar diferentes estratégias ao longo do processo.

Numa primeira abordagem, procurei integrar o Firebase Authentication diretamente no *frontend* Expo, criando um projeto na consola Firebase e configurando as credenciais OAuth necessárias para login Google. Contudo, rapidamente surgiram dificuldades técnicas ao tentar articular este *frontend* com o *backend*, desenvolvido em Node.js com o Firebase Admin SDK. Especificamente, os *tokens* gerados pelo *frontend* não eram reconhecidos pelo *backend*, gerando erros recorrentes de validação como o “Firebase ID token has incorrect 'aud' (audience) claim”. Apesar de várias tentativas de ajustar configurações e permissões, não foi possível assegurar uma integração estável entre os dois lados. Para além disso, o registo direto dos utilizadores no Firebase a partir do *frontend* mostrou-se inviável, uma vez que impedia a gestão centralizada dos dados no backend (MongoDB) e no sistema de autenticação.

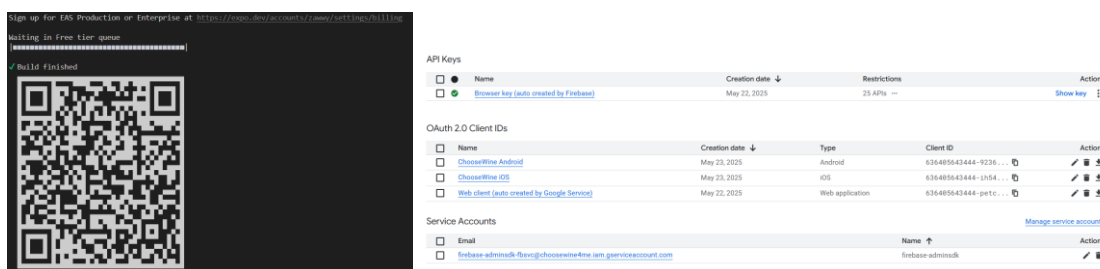


Figura 4-2 – Configuração da EAS DEV e GoogleCloud

Reconhecendo as limitações desta solução, decidi avançar para uma segunda abordagem, desta vez com React Native (Expo) e o módulo nativo de Google Sign-In. Esta escolha obrigou à migração do Expo Go para o ambiente EAS Dev Client como pode ver na **figura 4-2**, dado que o primeiro não suporta módulos nativos críticos. Utilizando o `@react-native-google-signin/google-signin`, consegui obter *tokens* Google e dados do utilizador de forma satisfatória, progredindo com sucesso no Android. No entanto, persistiram dificuldades na validação do

token no *backend*, já que o formato do *token* Google diferia daquele esperado pelo Firebase Admin SDK. Como alternativa, optei por enviar diretamente os campos de nome, email e foto do utilizador para o *backend*, contornando a verificação do *token*, solução que funcionou, mas apenas para Android como podemos ver na **figura 4-3**.

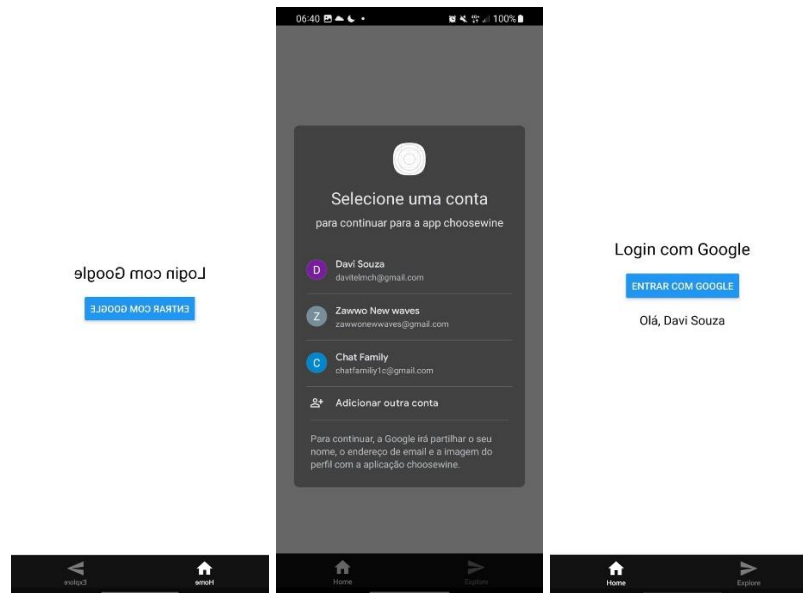


Figura 4-3 – Ecras de login na build React Native

Mesmo com estes avanços, a abordagem revelou-se insuficiente para uma verdadeira aplicação multiplataforma, pois:

- O módulo Google Sign-In não oferece suporte completo ou estável em web, iOS e Android, limitando a expansão futura da aplicação.
- O Firebase Authentication, no contexto Expo/React Native, exige dependências nativas e configurações adicionais que não são suportadas no *workflow managed*.
- A gestão manual de sessões, *tokens* e segurança resulta num código menos escalável e mais difícil de manter.

Em síntese, a experiência evidenciou que as *stacks* e bibliotecas modernas para autenticação, apesar do seu potencial, apresentam obstáculos práticos significativos quando se pretende integrar simultaneamente autenticação federada, *backend* próprio e bases de dados externas em

ambiente multiplataforma. Esta constatação reforça a importância de um planeamento criterioso da arquitetura de autenticação, considerando não só as capacidades das bibliotecas, mas também as limitações de integração entre *frontend* e *backend* num projeto real.

Principais motivos da mudança para Flutter:

- **Suporte oficial e completo do Firebase:** O Flutter, mantido pela Google, garante integração plena com todos os serviços Firebase, sem limitações nativas.
- **Login Google multiplataforma real:** Funciona em Android, iOS e Web de forma consistente.
- **Facilidade de manutenção e evolução:** Uma única base de código para todas as plataformas, sem *workarounds* complexos.
- **Performance nativa e documentação de qualidade:** O Flutter tem excelente desempenho e um ecossistema sólido.

Opto assim por migrar o projeto para Flutter nesta fase inicial, uma vez que é a opção que melhor garante a segurança, escalabilidade e sustentabilidade do ChooseWINE4me, evitando bloqueios futuros e facilitando o desenvolvimento de novas funcionalidades.

4.2. Prototipagem

O protótipo foi desenvolvido através do Figma, com o objetivo de criar um modelo de navegação detalhado da aplicação e refletir o que é esperado como resultado.

O Figma é uma ferramenta de prototipagem utilizada na área de design de UI (User Interface), lançada no mercado em 2012. É amplamente utilizada no mercado atual devido à sua capacidade de colaboração em tempo real e à versatilidade para designers e desenvolvedores.

Apesar das limitações na prototipagem, o objetivo principal é refletir o design da aplicação e demonstrar algumas das funcionalidades esperadas.

O protótipo pode ser acedido em: [Figma](#), password: softinsa123

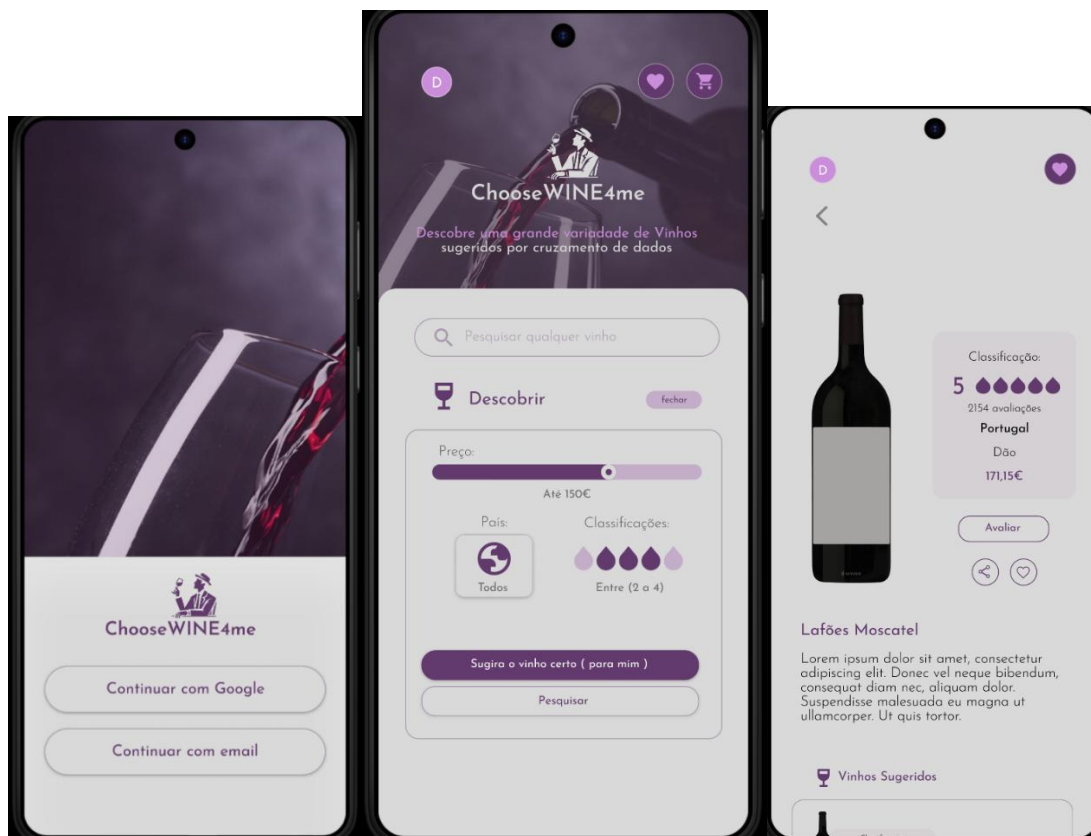


Figura 4-4 - Ecrãs do Protótipo

A prototipagem da **figura 4-4** foi essencial para o desenvolvimento da aplicação, permitindo identificar as páginas essenciais para o funcionamento, bem como os componentes a serem utilizados na aplicação.

5. Arquitetura da Plataforma

Este capítulo apresenta a arquitetura global da aplicação ChooseWINE4me, com o objetivo de descrever os seus principais módulos, a forma como se articulam entre si e os serviços externos que integram. A estrutura modular foi pensada para garantir escalabilidade, organização e independência entre componentes, permitindo que cada módulo possa evoluir de forma autónoma sem comprometer o funcionamento do sistema.

A aplicação está dividida em quatro grandes blocos funcionais:

- *Frontend* (Flutter): é a interface utilizada pelo cliente, construída com a *framework* Flutter e programada em Dart. Este módulo é responsável por apresentar os vinhos, gerir

as interações do utilizador (login, pesquisa, favoritos, etc.) e comunicar com o *backend* através de pedidos HTTP. Toda a comunicação passa por um serviço central (ApiService), que adiciona automaticamente o *token* de sessão (JWT) às requisições seguras.

- *Backend* (Node.js + Express): atua como núcleo da aplicação, onde são tratados os pedidos recebidos do *frontend*, acedida a base de dados MongoDB e feita a ligação com o sistema de recomendação. Inclui lógica de negócio, verificação de permissões, gestão de utilizadores, moradas, avaliações e favoritos.
- Base de Dados (MongoDB Atlas): armazena de forma segura e estruturada todos os dados relevantes da aplicação, incluindo os utilizadores, os vinhos recolhidos, o histórico de interações e preferências. O acesso é feito de forma segura através do Mongoose, que permite validar e organizar os dados como documentos.
- Sistema de Recomendação (FastAPI + Scikit-learn): micro serviço externo que recebe pedidos autenticados e devolve listas ordenadas de vinhos com base no perfil do utilizador. Utiliza filtragem colaborativa e proximidade por atributos, garantindo sugestões personalizadas mesmo para utilizadores com pouco histórico.

Além destes quatro módulos principais, a aplicação integra os serviços da Firebase, que desempenham um papel crucial em dois aspetos:

- Autenticação: o Firebase Authentication permite autenticar utilizadores com Google ou email/senha. Após a autenticação, o *token* de identificação é enviado ao *backend*, que o valida com o Firebase Admin SDK e gera um token JWT próprio para manter a sessão.
- Analytics: o Firebase Analytics está integrado para recolher dados sobre o uso da aplicação (ex. páginas mais visitadas, tempo médio de sessão), permitindo futuras melhorias baseadas em dados reais.

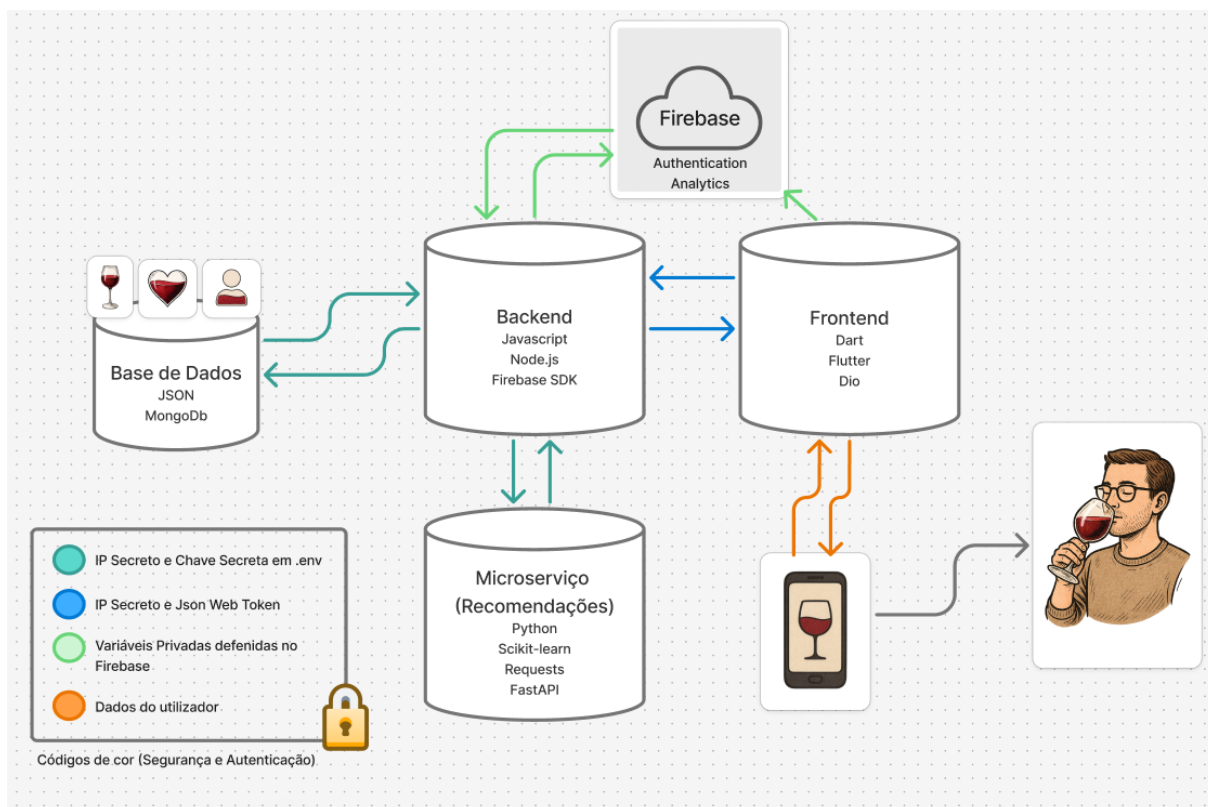


Figura 5-1 – Esquema de Arquitetura da Aplicação

A **Figura 5-1– Esquema Geral da Arquitetura** representa visualmente a relação entre todos estes módulos. Para ver o esquema com maior detalhe, pode aceder no seguinte link:

<https://www.figma.com/board/0L4tbOd4fqgSXMwdg5jklI/Untitled?node-id=0-1&t=gHM4X0J0s5EA67R0-1>

Esta arquitetura modular e bem definida foi essencial para o desenvolvimento eficiente e progressivo da aplicação, permitindo isolar problemas, testar componentes de forma independente e garantir uma base sólida para futuras melhorias.

Nos próximos capítulos, será descrita a **implementação de cada módulo**, apresentando o código, os testes realizados e os desafios enfrentados.

5.1. *Backend*

O *backend* é responsável por gerir os pedidos e processar a lógica da aplicação, garantindo a comunicação entre o *frontend* e a base de dados. Ele lida com a autenticação de utilizadores, validação de dados, armazenamento e recuperação de informações, além de implementar regras de negócio essenciais para o funcionamento do sistema.

Normalmente, o *backend* é desenvolvido utilizando linguagens como JavaScript (Node.js), Python, Java, PHP ou C#, e pode integrar-se com bases de dados SQL ou NoSQL, dependendo das necessidades da aplicação. Além disso, as APIs desempenham um papel fundamental no *backend*, permitindo que diferentes partes do sistema comuniquem entre si de forma eficiente.

Para este projeto, utilizamos a linguagem JavaScript no ambiente Node.js para desenvolver o *backend*. A estrutura da API é gerida com o Express.js, garantindo modularidade e eficiência. O armazenamento de dados é feito no MongoDB, uma base de dados NoSQL, e utilizamos a biblioteca Mongoose para facilitar a interação com os dados. Esse conjunto de tecnologias proporciona um desenvolvimento ágil, escalável e compatível com aplicações modernas.

Para implementar o *Backend*, em resumo, foram seguidos inicialmente os seguintes passos:

- Criar diretórios
- Configurar a ligação ao Servidor
- Criar Modelos
- Criar *Controllers*
- Criar Rotas
- Teste as Rotas
- Implementação do *Middleware*
- Testes ao servidor

5.1.1. Comandos de criação

Para iniciar o desenvolvimento do *backend* da aplicação ChooseWINE4me, foi necessário criar a estrutura de diretórios e ficheiros que dariam suporte à lógica do servidor, rotas, controladores e ligação à base de dados. Esta configuração inicial teve como objetivo estabelecer um ambiente organizado, modular e escalável, facilitando a manutenção e o crescimento futuro da aplicação.

Abaixo encontram-se os principais comandos utilizados durante a criação do ambiente, bem como os ficheiros e diretórios estruturantes para o *backend*:

```
• mkdir choosewine-backend
• cd backend
• npm init -y
• npm install express mongoose dotenv cors Morgan
• mkdir src / cd src
• mkdir models controllers routes middleware config
• criar ".env" na raiz do /choosewine-backend // não é um comando.
• criar "server.js" e "app.js" em /src // não é um comando.
• criar "db.js" no diretório /config //não é um comando.
• cd models
• New-Item -Path .\User.js -ItemType File
• New-Item -Path .\Wine.js -ItemType File
• New-Item -Path .\Address.js -ItemType File
• New-Item -Path .\Favorite.js -ItemType File
• New-Item -Path .\History.js -ItemType File
• New-Item -Path .\Rating.js -ItemType File
• cd ..
• cd controllers
• New-Item -Path .\userController.js -ItemType File
• New-Item -Path .\addressController.js -ItemType File
• New-Item -Path .\historyController.js -ItemType File
• New-Item -Path .\favoriteController.js -ItemType File
• New-Item -Path .\wineController.js -ItemType File
• New-Item -Path .\ratingController.js -ItemType File
• cd ..
• cd routes
• New-Item -Path .\appRoutes.js -ItemType File
• cd ..
• cd middleware
• New-Item -Path .\authMiddleware.js -ItemType File
• New-Item -Path .\errorMiddleware.js -ItemType File
```

Equation 1 - Comandos de criação

No encontram-se os principais comandas de

Após a execução dos comandos e a criação dos ficheiros correspondentes como na **Equation 1**, foi possível configurar o servidor e definir uma estrutura modular e organizada para o *backend* da aplicação. Esta estrutura contempla os principais componentes de uma arquitetura MVC (Model-View-Controller), separando responsabilidades entre modelos de dados, controladores e rotas. Foram também definidos os ficheiros de configuração e *middleware*, essenciais para funcionalidades como a autenticação de utilizadores e o tratamento centralizado de erros. A instalação das bibliotecas iniciais permitiu a gestão de rotas HTTP, ligação à base de dados,

configuração de variáveis de ambiente, comunicação segura com o *frontend* e monitorização de requisições. Com esta base técnica consolidada, o *backend* ficou preparado para suportar as funcionalidades da aplicação, assegurando desempenho, segurança, escalabilidade e facilidade de manutenção ao longo do desenvolvimento.

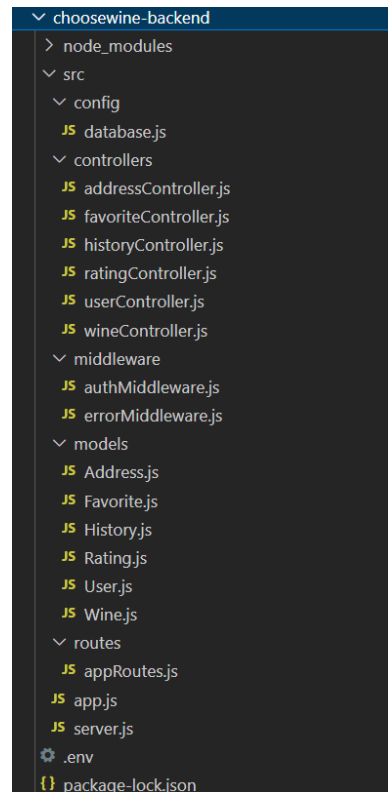


Figura 5-2- Backend - Estrutura de Ficheiros

Apos a inserção dos comandos obtive uma estrutura de ficheiros semelhante a **figura 5-2**, podendo então avançar então para a próxima etapa que será a criação dos modelos.

5.1.2. Definição dos Modelos em Javascript

Os modelos definem a estrutura dos dados na base de dados, funcionando como um esquema que determina quais informações serão armazenadas e de que forma. No caso do MongoDB, os modelos fazem o papel equivalente às tabelas nos bancos de dados relacionais, mas organizam os dados em formato de documentos JSON.

```
choosewine-backend > src > models > JS Wine.js > ...
You, 2 weeks ago | 1 author (You)
1  const mongoose = require('mongoose');
2
3  const WineSchema = new mongoose.Schema({
4    id_wine: { type: Number, required: true, unique: true },
5    name: { type: String, required: true },
6    thumb: { type: String },
7    country: { type: String },
8    region: { type: String },
9    average_rating: { type: Number, default: 0 },
10   ratings: { type: Number, default: 0 },
11   price: { type: Number, default: 0 },
12 });
13
14 module.exports = mongoose.model('Wine', WineSchema);
15
```

Figura 5-3- Backend - Modelo Vinho

A **Figura 5-3** apresenta o modelo denominado **"Wine"**, criado com recurso à biblioteca **Mongoose**, amplamente utilizada em aplicações Node.js para facilitar a comunicação com o MongoDB. Este modelo define os campos necessários para armazenar informações sobre vinhos, como `name` (nome), `thumb` (imagem), `country` (país), `region` (região), `average_rating` (pontuação média), `ratings` (número de avaliações) e `price` (preço).

Cada campo está associado a regras específicas que determinam o tipo de dados que pode conter, se o preenchimento é obrigatório e os valores por defeito, quando aplicável. Por exemplo, os campos `id_wine` e `name` são obrigatórios, e os campos numéricos possuem valores predefinidos para garantir consistência mesmo quando não preenchidos.

Este padrão é utilizado na definição de todos os modelos da aplicação, assegurando que os dados sejam armazenados de forma **consistente, organizada e validada** à entrada. A estrutura clara dos modelos também facilita o desenvolvimento das restantes camadas da aplicação, como os controladores e as rotas, promovendo um fluxo de dados fiável e estruturado.

5.1.3. Controladores

Os **controladores** são responsáveis por tratar a lógica de negócio associada às requisições feitas à aplicação. Servem como intermediários entre os **modelos** (que representam os dados na base de dados) e as **rotas** (que definem os *endpoints* da API), processando os dados recebidos do

cliente, interagindo com os modelos quando necessário e devolvendo respostas adequadas ao *frontend*.

```
choosewine-backend > src > controllers > JS wineController.js > ...
You, 2 weeks ago | 1 author (You)
1  const Wine = require('../models/Wine');
2
3  > exports.getAllWines = async (req, res) => { ...
11 };
12
13 > exports.getWineById = async (req, res) => { ...
25 };
26
27 exports.createWine = async (req, res) => {
28   try {
29     const { name, thumb, country, region, price } = req.body;
30
31     // Busca o último vinho e gera um novo id_wine de forma sequencial
32     const lastWine = await Wine.findOne({}).sort({ id_wine: -1 });
33     const newIdWine = lastWine && lastWine.id_wine ? lastWine.id_wine + 1 : 1;
34
35     const newWine = new Wine({
36       id_wine: newIdWine,
37       name,
38       thumb,
39       country,
40       region,
41       price,
42       // average_rating e ratings usarão os valores default (0)
43     });
44
45     await newWine.save();
46     return res.status(201).json(newWine);
47   } catch (error) {
48     console.error('Erro ao criar vinho:', error);
49     return res.status(500).json({ error: 'Erro ao criar vinho' });
50   }
51 };
52
53 > exports.updateWine = async (req, res) => { ...
73 };
74
75 > exports.deleteWine = async (req, res) => { ...
87 };
88
```

Figura 5-4- Backend – wineController

A **Figura 5-4** apresenta o ficheiro `wineController.js`, onde estão definidas as funções que controlam as operações sobre os dados dos vinhos. Cada função representa uma ação específica que pode ser realizada sobre o modelo `Wine`, permitindo gerir integralmente os registos armazenados:

- `getAllWines(req, res)`: Recupera e devolve todos os vinhos registados na base de dados.
- `getWineById(req, res)`: Retorna um vinho específico com base no seu `id_wine`. Caso não seja encontrado, é devolvida uma resposta com o código 404.
- `createWine(req, res)`: Cria um vinho. Atribui automaticamente um `id_wine` sequencial, com base no último registo existente. Campos como `average_rating` e `ratings` assumem valores padrão.

-
- `updateWine(req, res)`: Atualiza os dados de um vinho existente, excetuando os campos `id_wine`, `average_rating` e `ratings`, que não devem ser alterados diretamente.
 - `deleteWine(req, res)`: Elimina um vinho da base de dados com base no seu `id_wine`.

Estas funções utilizam a abordagem assíncrona com *async/await*, assegurando que a comunicação com a base de dados ocorra de forma eficiente, sem bloquear a execução da aplicação. O tratamento de erros é também implementado em cada função, garantindo que, em caso de falha, o utilizador receba uma resposta clara e adequada.

Em resumo, os controladores organizam a lógica de processamento da API e garantem que cada operação sobre os dados seja executada corretamente. A implementação do padrão CRUD torna o sistema mais estruturado e previsível, enquanto o modelo modular facilita a manutenção e a escalabilidade da aplicação, promovendo boas práticas no desenvolvimento *backend*.

Esta lógica foi replicada nos restantes controladores do sistema, resultando nos ficheiros `UserController.js`, `AddressController.js`, `HistoryController.js`, `FavoriteController.js` e `RatingController.js`, cada um com as suas operações específicas, mas todos estruturados segundo o mesmo princípio de organização e responsabilidade.

5.1.4. Rotas

As **rotas**, em contexto de uma API, são responsáveis por definir os caminhos pelos quais a aplicação recebe e processa as requisições do utilizador. Nas rotas determinam quais **endpoints** estão disponíveis e qual lógica será executada para cada pedido recebido pelo servidor.


```
choosewine-backend > src > routes > JS appRoutes.js > ...
You, 1 second ago | 1 author (You)
1  const express = require('express');
2  const router = express.Router();
3
4  const userController = require('../controllers/userController');
5  const addressController = require('../controllers/addressController');
6  const wineController = require('../controllers/wineController');
7  const favoriteController = require('../controllers/favoriteController');
8  const historyController = require('../controllers/historyController');
9
10 const { verifyToken } = require('../middleware/authMiddleware');
11
12 router.use(verifyToken);
13
14 { // Rotas para Users
15   router.get('/users', userController.getAllUsers);
16   router.get('/users/:id', userController.getUserById);
17   router.post('/users', userController.createUser);
18   router.put('/users/:id', userController.updateUser);
19   router.delete('/users/:id', userController.deleteUser);
20 }
21 // Rotas para Addresses
22 router.get('/addresses', addressController.getAllAddresses);
```

Figura 5-5- Importação de Rotas em *appRoutes.js*

Na **Figura 5-5**, podemos observar a implementação do ficheiro *Routes.js*, onde as rotas são configuradas utilizando o *Express.js*, um *framework* para *Node.js* que facilita a criação de APIs. O código segue a seguinte estrutura:

- Importação do *Express.js* e criação de um *Router* para gerir as rotas.
- Importação dos controladores, que contêm a lógica associada a cada funcionalidade.
- Definição das rotas para "Users", onde cada método HTTP é associado a uma função do controlador correspondente:

As rotas desempenham um papel essencial no funcionamento do *backend*, garantindo que as requisições sejam encaminhadas corretamente para os controladores responsáveis por processar os dados e responder ao cliente.

5.1.5. Middleware

Os *middlewares* são funções intermediárias que processam requisições antes que elas cheguem ao controlador final. Eles são fundamentais no **Express.js**, pois permitem adicionar funcionalidades como autenticação, tratamento de erros, registo de *logs* e manipulação de permissões.

```

1 const jwt = require('jsonwebtoken');
2
3 const authMiddleware = (req, res, next) => {
4   const token = req.header('Authorization');
5
6   if (!token) {
7     return res.status(401).json({ message: 'Acesso negado. Nenhum token fornecido.' });
8   }
9
10  try {
11    const decoded = jwt.verify(token, process.env.JWT_SECRET);
12    req.user = decoded;
13    next();
14  } catch (error) {
15    res.status(401).json({ message: 'Token inválido.' });
16  }
17 };
18
19 module.exports = authMiddleware;

```

```

1 const errorMiddleware = (err, req, res, next) => {
2
3   console.error(err.stack);
4   res.status(500).json({
5     message: err.message || 'Erro interno do servidor.',
6     stack: process.env.NODE_ENV === 'development' ? err.stack : null,
7   });
8
9   module.exports = errorMiddleware;
10
11

```

Figura 5-6 - *authmiddleware, errormiddleware*

Na **Figura 5-6**, vemos dois exemplos de *middleware* implementados no projeto:

authMiddleware.js (Middleware de Autenticação):

- Este *middleware* verifica se a requisição contém um **token de autenticação** válido no cabeçalho "Authorization".
- Caso o *token* esteja ausente, a resposta retornada será um erro **401 (Unauthorized)**.
- Se o *token* for inválido, a resposta será um erro **403 (Forbidden)**.
- Se o *token* for válido, a requisição segue normalmente e o utilizador autenticado é armazenado no `req.user`, permitindo que as próximas camadas acessem aos seus dados.

errorMiddleware.js (Middleware de Tratamento de Erros):

- Este *middleware* captura erros que ocorrem na aplicação e retorna uma resposta padronizada.
- Ele regista o erro no console e envia uma resposta **500 (Internal Server Error)** com uma mensagem de erro.
- Durante o desenvolvimento, pode exibir a **stack do erro**, ajudando na depuração.

Os *middlewares* são essenciais para manter a segurança e a estabilidade da aplicação, garantindo que apenas utilizadores autorizados acessem a determinadas rotas e que os erros sejam tratados de forma centralizada.

Embora o **middleware de autenticação** já tenha sido desenvolvido, ainda não aplicámos a proteção nas rotas. Esta decisão foi tomada para facilitar a fase de testes, permitindo que as requisições sejam feitas sem a necessidade de um **token de autenticação**. Além disso, ainda não implementámos um sistema de controlo de login para gerar e gerir esses *tokens* de forma segura. Assim que o módulo de autenticação estiver concluído, integraremos o *middleware* às

rotas apropriadas para garantir que apenas utilizadores autenticados possam aceder a determinadas funcionalidades da API.

5.1.6. Configuração do servidor

A configuração do servidor é um dos elementos fundamentais do *backend*, garantindo que a aplicação consiga lidar com requisições, gerir erros e comunicar-se com a base de dados. No nosso projeto, utilizamos **Express.js** para criar o servidor, aplicando *middlewares* como cors para permitir requisições externas, express.json() para interpretar JSON e morgan para registar as requisições no terminal. As rotas são organizadas num ficheiro separado (routes.js), facilitando a modularização do código, e um *middleware* de erros foi implementado para lidar com exceções de forma centralizada.

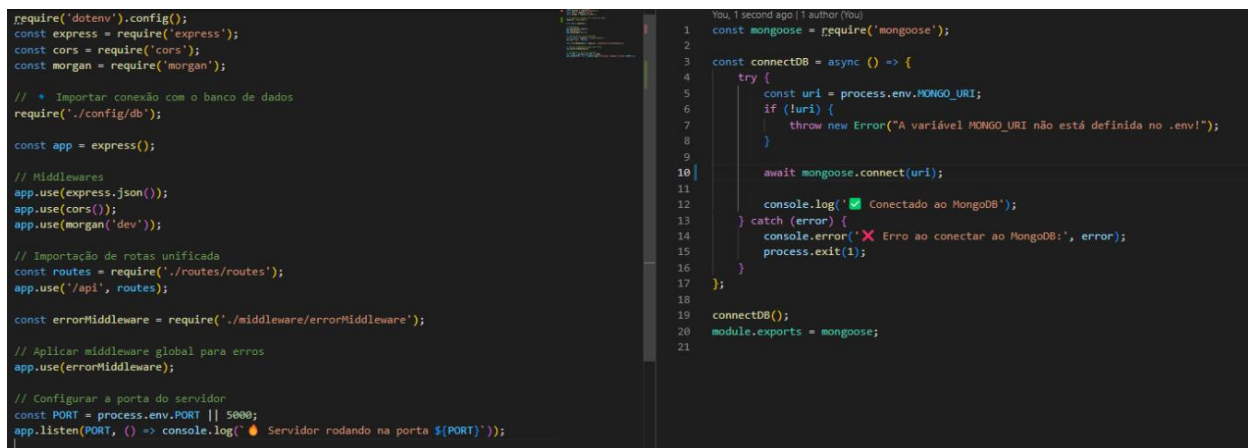


Figura 5-7 - server.js e database.js

Na **Figura 5-7**, vemos dois ficheiros fundamentais para a infraestrutura do *backend*:

1. **server.js** (Configuração do Servidor Express):
 - **Carregamento de dependências:** O ficheiro importa bibliotecas essenciais como express (para criação do servidor), cors (para permitir requisições externas) e morgan (para registo de logs).
 - **Criação da aplicação:** A instância do Express é inicializada através de `const app = express();`.
 - **Configuração de *middlewares*** (express.json(), cors())
 - **Importação das rotas:** O servidor carrega as rotas da aplicação através do ficheiro routes.js e aplica-as no *endpoint* base /api.

- **Tratamento global de erros:** O *middleware* de erros (`errorMiddleware`) é aplicado para capturar exceções e evitar que falhas inesperadas quebrem a aplicação.
- **Definição da porta e inicialização:** O servidor é iniciado na porta definida na variável de ambiente `PORT` ou, caso não esteja configurada, na porta **5000**.

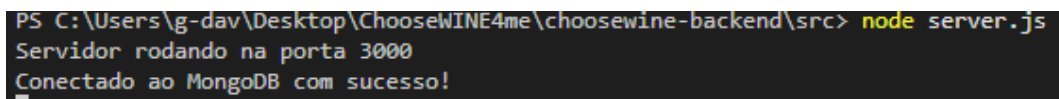
database.js (Configuração da Base de Dados com MongoDB):

- Obtém a URI do banco de dados a partir do ficheiro `.env`.
- Se a variável `MONGO_URI` não estiver definida, um erro é lançado.
- Conecta ao MongoDB de forma assíncrona utilizando `mongoose.connect(uri)`.
- Regista uma mensagem no console para indicar sucesso ("Conectado ao MongoDB") ou erro ("Erro ao conectar ao MongoDB").

A conexão com o MongoDB é gerida através do `mongoose`, que estabelece a ligação ao banco de dados de forma assíncrona. Para manter as credenciais seguras e evitar expor informações sensíveis no código, utilizamos um ficheiro `“.env”`, onde armazenamos variáveis como a `MONGO_URI` (endereço da base de dados) e a `PORT` (porta do servidor). Esse ficheiro não é incluído no repositório, garantindo mais segurança ao projeto. Com essa estrutura, o *backend* torna-se mais organizado, escalável e fácil de manter.

5.1.7. Iniciar o servidor

Para iniciar o servidor é simples, basta ter a certeza de que esta no diretório `“/src”` e executar o comando **`node server.js`**



```
PS C:\Users\g-dav\Desktop\ChooseWINE4me\choosewine-backend\src> node server.js
Servidor rodando na porta 3000
Conectado ao MongoDB com sucesso!
```

Figura 5-8 - Mensagem de sucesso ao iniciar o servidor

Na **figura 5-8** podemos observar uma mensagem de sucesso ao iniciar o servidor e a conectar ao MongoDB. Se estiver tudo configurado corretamente como na **figura 9** esta é a mensagem que devemos receber,

Alguns erros comuns são criação de pastas nos locais errados ou a linha de conexão no ficheiro `“.env”` esta mal definida.

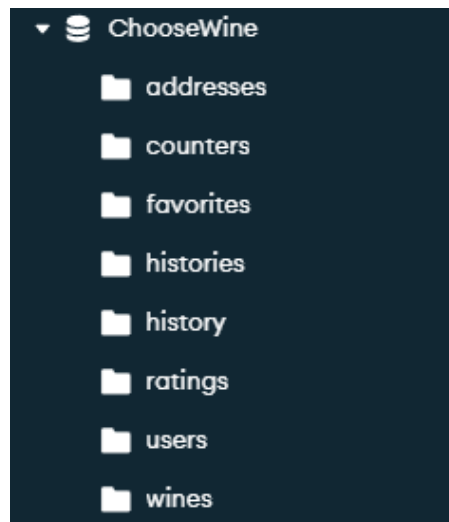


Figura 5-9 - Tabelas criadas com sucesso no MongoDB

Quando o servidor é iniciado com sucesso, como mostrado na **figura 5-9**, as tabelas (ou **coleções**, no contexto do **MongoDB**) são criadas automaticamente no banco de dados. Isto acontece porque utilizamos o **Mongoose**, que mapeia os modelos definidos no código para coleções dentro do **MongoDB**.

Uma particularidade do **Mongoose** é que ele adiciona automaticamente um **"s"** ao final do nome das coleções. Por exemplo, se definirmos um modelo chamado **"User"**, a coleção correspondente será armazenada como **"users"** na base de dados. Isso pode ser observado na imagem, onde os modelos definidos no código, como **"User"** e **"Rating"** aparecem no base de dados como **"users"** e **"ratings"**.

Testes ao *Backend*

Os **testes ao *backend*** são uma etapa essencial para garantir que as funcionalidades da API estão a funcionar corretamente. Para isso, foi utilizada a ferramenta **Postman**, que permite simular requisições HTTP e analisar as respostas do servidor de forma clara e eficiente.

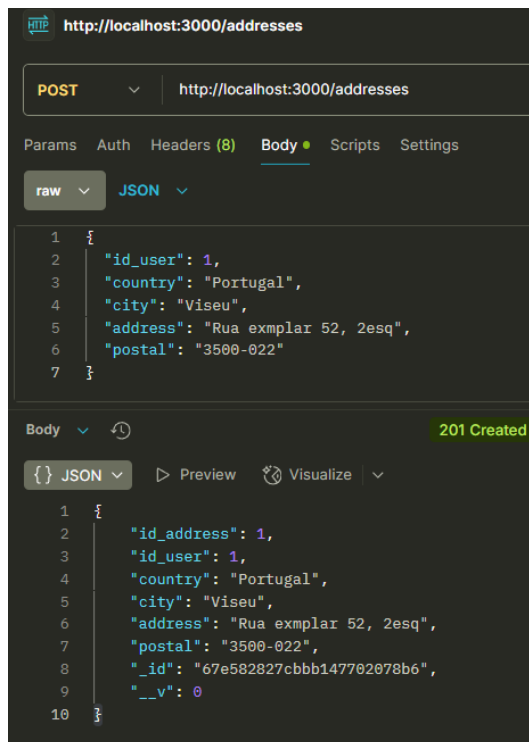


Figura 5-10 – Post a Adresses

A **Figura 5-10** apresenta um exemplo de teste realizado à rota POST /api/addresses, responsável por **registar um endereço associado a um utilizador** na base de dados. No corpo da requisição, foi enviado um objeto JSON contendo campos como country, city, address, postal e o respetivo id_user. Após o envio, a API respondeu com o código **201 Created**, confirmando que o endereço foi registado com sucesso.

Estes testes permitem **validar a estrutura dos dados recebidos pela API**, verificar se os **campos obrigatórios** estão a ser corretamente tratados e garantir que a API responde conforme o esperado antes da sua integração com o *frontend*.

5.1.8. Verificação na Base de Dados

Embora o Postman permita confirmar que a API responde corretamente, é igualmente importante verificar se os **dados estão efetivamente a ser inseridos na base de dados**, com a estrutura adequada. A **Figura 5-11** abaixo mostra a inserção bem-sucedida de um endereço na coleção “addresses” da base de dados **ChooseWine**, contendo todos os campos esperados, como id_address, id_user, country, city, address e postal.

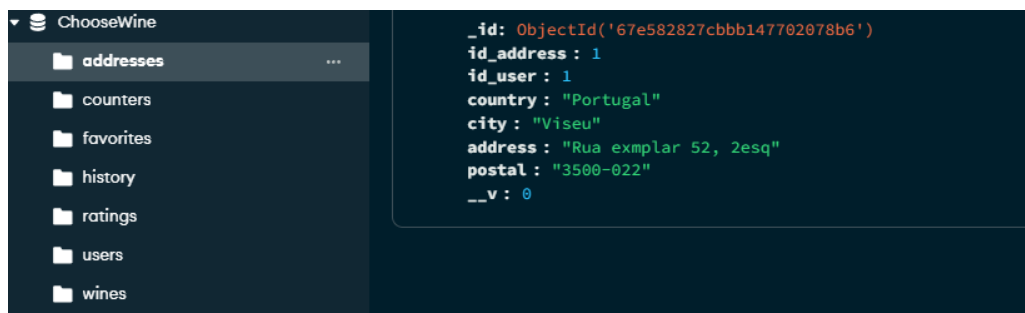


Figura 5-11 – Address inserido no MongoDB

Esta verificação foi realizada através do **MongoDB Compass**, que permite visualizar de forma gráfica os documentos armazenados. Confirmar diretamente a inserção na base de dados permite assegurar que os dados estão a ser **armazenados de forma estruturada**, sem campos ausentes, valores nulos ou tipos de dados incorretos.

Este processo de validação cruzada — entre o Postman e o MongoDB — é fundamental para garantir a **integridade da aplicação**, identificando desde cedo eventuais falhas no mapeamento entre os controladores da API e os modelos de dados.

Para consultar todos os testes realizados ao *backend*, está disponível o ficheiro “**Testes ao Backend**” nos documentos no **Anexo D**

5.2. Modulo *scrapper* com ‘API-Vivino’

Durante o desenvolvimento da aplicação ChooseWINE4me, surgiu a necessidade de recolher dados detalhados sobre vinhos disponíveis no mercado, especialmente de vinhos produzidos em Portugal. Como a Vivino não oferece uma API pública oficial, a solução passou por utilizar uma alternativa baseada na plataforma Apify, que permite criar agentes (atores) para automação de tarefas, incluindo *scraping* de sites.

5.2.1. Configuração

A primeira etapa consistiu em explorar a ferramenta “Vivino Scraper” na plataforma Apify. Esta ferramenta oferece uma interface visual (HUD) onde é possível configurar pesquisas com palavras-chave e filtros personalizados. Após alguns testes, percebeu-se que a combinação mais eficaz para obter vinhos portugueses era usar o keyword parâmetros como “vinho tinto Portugal”, “Vinho Branco”, “Moscatel” e o *searchFilter* em "Portugal" como o seguinte corpo JSON como demonstra na **figura 5-12 abaixo**, provou ser bastante eficaz:

```
{
  "process": "gs",
  "keyword": "vinho tinto Portugal",
  "searchFilter": "Portugal",
  "maximum": 50,
  "summarizedResults": false,
  "saveOnUsageCost": false,
  "delay": 2,
  "retries": 3
}
```

Figura 5-12 – Json eficaz para pesquisa em VivinoAPI da Apify

Esta configuração permitiu recolher uma lista de Localizador Uniforme de Recursos (URL) de vinhos, embora inicialmente não viessem acompanhadas do preço.

5.2.2. Implementação do script de formatação

Para contornar a ausência de preços na pesquisa geral, foi criada uma abordagem em duas fases:

- **Fase1 – Coleta de URLs relevantes:**

Utilizámos o agente da Apify com a configuração anterior para gerar uma lista de URLs de vinhos com origem em Portugal.

- **Fase 2 – Extração detalhada:**

Cada URL obtida era passada a um segundo script, que fazia *scraping* da respetiva página do vinho para recolher os detalhes que nos interessavam: nome, tipo, avaliação média, país, adega, nível de álcool, imagem e preço.

O script em Node.js foi evoluindo progressivamente para melhorar a qualidade dos dados. Abaixo, os principais pontos aperfeiçoados ao longo do processo:

- Validação para ignorar URLs repetidas ou já registadas no ficheiro final.
- Extração do ano (ignorando coleções genéricas vintages entre outras).
- Remoção do campo region, que não se demonstrou confiável ou consistente.
- Exibição de cada resposta formatada diretamente no terminal para agilizar a revisão manual.

-
- Exportação contínua e incremental para o ficheiro `vinhos-portugal-com-preco.json`.

5.2.3. Resultado

Apos aplicar o script de formatação cada vinho é armazenado num ficheiro um json com o seguinte formato da **figura 5-13**:

```
{
  "url": "https://www.vivino.com/w/9700144",
  "name": "Alecrim Tinto",
  "type": "Red",
  "rating": 3.5,
  "country": "Portugal",
  "winery": "Ségur Estates",
  "alcohollevel": 13,
  "image": "https://images.vivino.com/thumbs/H1iLppkSR3SE0ISp7xVHbQ_p1_480x640.png",
  "price": 5.68,
  "year": 2020
}
```

Figura 5-13 – Json Resultante do Script de extração e formatação

Esta estrutura garantiu uma base de dados limpa, fácil de integrar ao *backend* e suficientemente descritiva para alimentar os modelos de recomendação. Durante a implementação deste módulo, destacaram-se algumas aprendizagens relevantes:

- Nem todos os vinhos possuem os mesmos campos preenchidos (por exemplo, ano e imagem nem sempre estão disponíveis).
- Algumas páginas redirecionam para múltiplas variantes (vintages), exigindo uma lógica para extrair apenas o vinho principal.
- Foi necessário corrigir o nome do vinho, pois inicialmente estávamos a guardar o identificador da URL e não o nome real exibido no cabeçalho da página.

Este módulo, apesar de parecer técnico e independente, desempenha um papel fundamental no sucesso da aplicação: fornece dados reais e atualizados sobre vinhos, sem os quais a recomendação e a personalização não seriam possíveis.

Com isso, o sistema passou a contar com mais de **240 vinhos portugueses** detalhadamente registados, prontos para alimentar a inteligência artificial e enriquecer a experiência dos utilizadores.

5.2.4. Reconfigurações no *Backend*

À medida que o módulo de *scraping* evoluiu e os dados recolhidos se tornaram mais completos e descritivos, tornou-se evidente a necessidade de reestruturar o modelo de vinho no *backend* para acomodar novas propriedades.

Inicialmente, o modelo Wine guardava apenas informações básicas como nome, país, imagem, rating médio e preço. No entanto, com a melhoria do script de extração — que passou a recolher também o nome da adega (*winery*), o teor alcoólico (*alcoholLevel*) e o ano (*year*) — tornou-se necessário incluir esses novos campos no esquema de dados e adaptar toda a lógica do *backend* para suportar essas alterações.

O primeiro passo foi atualizar o modelo *Wine.js*, definindo os novos campos e garantindo que os dados fossem armazenados com validações adequadas. Foram adicionadas as seguintes propriedades:

- *winery*: nome da casa produtora do vinho.
- *alcoholLevel*: valor numérico representando o teor de álcool.
- *year*: quando disponível, o ano de produção do vinho.
- *type*: tipo do vinho (ex. "Red", "White").

Esta atualização permitiu que cada vinho guardado fosse representado de forma mais fiel, com dados mais ricos e úteis tanto para os utilizadores como para os algoritmos de recomendação.

Com a adição dos novos campos, também foi necessário atualizar o controlador principal *wineController.js*, garantindo que:

- A função *createWine* permitisse o armazenamento de vinhos com estas novas informações.
- A função *getAllWines* retornasse todos os campos corretamente ao *frontend*.
- A função *updateWine* pudesse ser usada para inserir essas propriedades em registos antigos, caso necessário.

Foram realizados novos testes ao controlador para garantir que nenhuma funcionalidade pré-existente fosse comprometida. O processo incluiu chamadas via Postman com objetos JSON já no novo formato e verificação no MongoDB Compass para confirmar a estrutura dos documentos armazenados.

5.3. Micro Serviço de *Machine Learning*

Vamos agora abordar o módulo de *Machine Learning* que foi desenvolvido para o projeto **ChooseWine4me**, responsável por dotar a aplicação de recomendações inteligentes de vinhos personalizadas para cada utilizador. Esse micro-serviço foi implementado em Python, utilizando a biblioteca **scikit-learn** para serializar dados e calcular pares próximos (“Nearest Neighbors”), ou, em português, “pares de maior semelhança”. Para expor suas funcionalidades de forma simples e eficiente, integrou-se a framework **FastAPI**, criando uma rota HTTP que comunica com o *backend* Node.js por meio de requisições **requests**, garantindo respostas rápidas e seguras.

Em suma, esta arquitetura em camadas permite ao *frontend* obter, em tempo real, listas de vinhos ordenadas por similaridade ou por afinidades colaborativas, de acordo com o histórico e a localização de cada utilizador, que também são obtidos em tempo real.

Nos tópicos seguintes, detalho primeiro os conceitos fundamentais de Machine Learning e proximidade de itens, depois exploro as Recomendações Colaborativas e, por fim, apresento a estrutura interna do micro-serviço e os comandos necessários para sua criação e configuração.

5.3.1. *Machine Learning*

Para modelar a proximidade entre vinhos, adotei uma abordagem de Machine Learning supervisionado por instâncias, alojado no micro-serviço em Python baseado em FastAPI. Cada vinho é representado como um vetor num espaço de características, que combina atributos numéricos (preço e avaliação agregada) e atributos categóricos (país, região, tipo e adega). As variáveis categóricas são convertidas em valores inteiros através de LabelEncoder, transformando palavras como “Portugal” ou “Red” em índices numéricos coerentes.

Obtida a matriz X (número de vinhos×número de atributos), utilizei o NearestNeighbors do Scikit-learn, configurado para métrica de cosseno. Esta métrica quantifica a similaridade angular entre dois vetores, ou seja, mede quanto dois vinhos partilham da mesma “direção” de

características, independentemente da escala absoluta de cada atributo. Durante a fase de treino, o modelo apenas indexa os vetores de todos os vinhos disponíveis. No pedido de recomendação, o sistema recebe o conjunto de vinhos já apreciados pelo utilizador (favoritos e/ou avaliados) e calcula para cada outro vinho a distância mínima até qualquer item desse conjunto de referência. O resultado é um score de proximidade que permite ordenar os vinhos de forma a apresentar primeiro os mais semelhantes ao gosto demonstrado pelo utilizador.

Este processo garante flexibilidade: não há necessidade de re-treinar um modelo complexo sempre que o catálogo se altera, bastando executar a construção da matriz e reindexar o NearestNeighbors. A extração de características e o cálculo de distâncias é rápido, escalável e mantém um perfil de dependências reduzido, ao invés de exigir *frameworks* pesadas.

5.3.2. Recomendação colaborativa

Embora a aproximação baseada apenas em atributos funcione bem para utilizadores com histórico de consumo, persiste o desafio do “*cold-start*” para novos utilizadores sem qualquer favorito ou rating. Para colmatar esta lacuna, desenvolvi um sistema híbrido de recomendação colaborativa que explora a localidade do utilizador. A aplicação armazena moradas associadas a cada utilizador; na recolha inicial de recomendações para um novo utilizador, invoco o *endpoint* de *addresses* para obter a sua cidade.

Nessa cidade, identificam-se todos os “*peers*” — utilizadores que residem na mesma localidade. Depois, fazem-se pedidos aos *endpoints* de *favorites* e *ratings* para reunir todos os vinhos consumidos por esses *peers*. A preferência coletiva local forma um conjunto inicial de sugestões: aqueles vinhos que foram verdadeiramente apreciados pela comunidade da mesma cidade. Em seguida, e para complementar essa lista, recorre-se novamente ao modelo de similaridade de atributos para ordenar todos os restantes vinhos não populares na região, garantindo diversidade e cobertura total do catálogo.

Desta forma, o ChooseWINE4me adapta-se tanto ao gosto pessoal (quando este é conhecido) como a tendências locais (quando não há histórico). O método híbrido combina a componente “*social*” — partilha de preferências entre vizinhos geográficos — com a componente “*item-based*” — similaridade técnica de atributos. O resultado é uma experiência de recomendação mais rica, que evita as limitações dos sistemas puramente colaborativos ou puramente baseados em conteúdos, enquanto mantém uma arquitetura simples, leve e de fácil manutenção.

5.3.3. Arquitetura e Comandos de Criação

Para a criação do micro-serviço de recomendação inteligente, foi utilizado **Python** com a framework **FastAPI**, que permite expor de forma simples e eficiente um *endpoint* HTTP para sugerir vinhos ao utilizador. Este serviço funciona em articulação com o *backend* Node.js, comunicando-se por requisições autenticadas via x-service-key.

A primeira etapa consistiu na criação de um ambiente virtual para isolar as dependências:

- `python -m venv .venv`
- `. .venv/bin/activate`
- `pip install --upgrade pip`

De seguida, foram instaladas as bibliotecas de dados e *machine learning*:

```
pip install pandas numpy scipy scikit-learn joblib
```

Estas bibliotecas permitiram o carregamento dos dados, tratamento de atributos e treino do modelo de recomendação baseado em **NearestNeighbors** com métrica de cosseno. Para integração com o *backend*, foram ainda instaladas as bibliotecas responsáveis pela API e comunicação HTTP:

```
pip install fastapi uvicorn python-dotenv requests
```

Com as dependências instaladas, o código foi organizado de forma modular no ficheiro `app_http.py`, começando pelo carregamento das variáveis de ambiente, onde se define a ligação ao *backend* em Node.js e a chave de autenticação necessária para as requisições. A seguir, é criada a aplicação FastAPI e definido o modelo `RecRequest` para tratamento dos pedidos do tipo POST.

A estrutura do ficheiro segue com um conjunto de funções auxiliares que acedem remotamente aos dados de utilizadores — favoritos, histórico de navegação, ratings e moradas — através de *endpoints* protegidos. Estes dados são utilizados para avaliar o grau de interação do utilizador com a aplicação e decidir qual a melhor estratégia de recomendação.

A função `load_wines_and_model()` é responsável por carregar a lista de vinhos da API, tratando os dados e codificando atributos categóricos como país, região, tipo e adega. A partir disso, é construída a matriz de características dos vinhos, que alimenta o modelo de vizinhos mais

próximos (NearestNeighbors), baseado em distância cosseno, permitindo medir a semelhança entre produtos.

No centro do código, encontra-se a função `_recommend`, onde está definida a lógica de decisão. Esta tenta primeiro recomendar com base no último favorito do utilizador, e, caso não exista, recorre ao último vinho visualizado. Se também não houver histórico, são consideradas as preferências de utilizadores da mesma cidade. Quando nenhuma destas opções está disponível, a recomendação é feita com base nos vinhos mais bem avaliados da base de dados.

A componente de exposição HTTP foi construída com recurso ao **FastAPI**, escolhido como *framework* principal pela sua leveza e desempenho. Para efeitos de desenvolvimento local, foi utilizado o **Uvicorn** como servidor ASGI, o que permite recarregamento automático a cada alteração no código. A biblioteca `python-dotenv` foi usada para carregar as variáveis de ambiente definidas no ficheiro `.env`, enquanto a biblioteca `requests` permitiu consumir os dados da API Node.js, assegurando a autenticação através do cabeçalho `x-service-key`.

No que respeita ao processamento de dados e *machine learning*, recorreu-se ao uso de bibliotecas como **Pandas**, **NumPy** e **Scikit-learn**. O Pandas foi essencial para a manipulação de dados tabulares, enquanto o NumPy garantiu operações vetoriais rápidas e eficientes. O Scikit-learn foi utilizado para aplicar a transformação dos atributos com `LabelEncoder` e para treinar o modelo de recomendação com o `NearestNeighbors`. Foram ainda incluídas as bibliotecas `scipy` e `joblib`, que complementam o processo de cálculo de distâncias e permitem, se necessário, guardar o modelo para reutilização posterior.

Por fim, são disponibilizados dois *endpoints* HTTP, um do tipo GET e outro POST, ambos no caminho `/recommend`, que recebem o identificador do utilizador e devolvem uma lista ordenada de sugestões. Esta arquitetura leve, mas eficaz, garante que a aplicação consiga gerar recomendações inteligentes com base em múltiplos contextos, mantendo a escalabilidade e simplicidade do sistema.

5.4. Frontend

Nesta secção, é apresentado o processo de desenvolvimento do *frontend* da aplicação ChooseWINE4me. Após a fase de planeamento e estruturação dos módulos *backend* e de inteligência artificial, foi necessário construir uma interface visual clara, intuitiva e funcional que permitisse aos utilizadores interagir com a aplicação de forma fluida e eficaz.

A construção do *frontend* passou por várias fases e exigiu decisões técnicas relevantes. Inicialmente, foi necessário definir a arquitetura geral da aplicação, escolhendo Flutter como framework principal. Esta escolha garantiu uma solução multiplataforma robusta, com excelente integração com o Firebase e outras bibliotecas modernas. O Flutter revelou-se particularmente adequado pelas suas capacidades de desempenho, simplicidade de estruturação de interfaces e manutenção de um único código para Android e iOS.

Durante o desenvolvimento, surgiram alguns desafios técnicos, nomeadamente na gestão da autenticação federada com o Firebase e na integração com o *backend* em Node.js. Para resolver esses obstáculos, foi estabelecida uma separação clara entre o *frontend* e o *backend*, com um serviço central de comunicações — o ApiService — responsável por gerir todos os pedidos HTTP. Este serviço permite incluir automaticamente o token JWT nas requisições autenticadas, mantendo a segurança e a consistência de sessão ao longo da aplicação. Esta abordagem evitou a repetição de lógica de autenticação em várias partes do código, facilitando a manutenção e o controlo da segurança.

A organização do *frontend* foi pensada com o objetivo de promover a escalabilidade e a reutilização de componentes. Foram criadas páginas modulares e *widgets* reutilizáveis, agrupando os elementos por funcionalidades e evitando a criação de páginas redundantes. Por exemplo, vários ecrãs partilham componentes visuais como cartões de vinho, listas de resultados e caixas de pesquisa, o que reduziu significativamente o volume de código e aumentou o desempenho da aplicação.

Além disso, foi dada especial atenção à separação dos fluxos principais: *login*, *onboarding*, pesquisa, perfil e recomendações. Cada fluxo foi tratado como um conjunto lógico de páginas e *widgets* interligados, o que permitiu estruturar a navegação de forma clara e previsível para o utilizador.

Do ponto de vista visual e funcional, o *frontend* atingiu um nível profissional. A aplicação apresenta um desempenho responsivo, adaptando-se bem a diferentes tamanhos de ecrã e mantendo tempos de carregamento reduzidos. A integração com o Google Analytics permitiu monitorizar a interação dos utilizadores com a aplicação, fornecendo dados valiosos para futuras melhorias. A autenticação com Firebase, por sua vez, assegura uma entrada segura e integrada com os restantes serviços.

5.4.1. Configurações iniciais

Para iniciar o desenvolvimento da aplicação ChooseWINE4me com Flutter, comecei por configurar o ambiente de desenvolvimento sem a necessidade do Android Studio, utilizando apenas as ferramentas essenciais, como o SDK do Flutter e o Android SDK diretamente.

A primeira etapa foi garantir que o **Flutter SDK** estava instalado corretamente. Para isso, segui os passos abaixo:

1. **Instalar o Flutter SDK:**

Baixei o Flutter SDK diretamente do site oficial e extraí o conteúdo para um diretório no meu sistema. Em seguida, adicionei o caminho do Flutter ao PATH do sistema, para que o comando flutter fosse reconhecido globalmente.

2. **Instalar o Android SDK:**

Utilizei o **Android SDK Tools** diretamente, sem instalar o Android Studio. Para isso, fiz o download dos **Command Line Tools** do Android, configurando-os no VS Code com os caminhos corretos para o SDK.

3. **Configuração do Java:**

Como o Flutter depende do Java, também instalei o **JDK 11** (o qual é suportado pelo Flutter) e configurei a variável de ambiente JAVA_HOME para que o sistema reconhecesse corretamente o Java.

4. **Verificação do Ambiente:**

Após configurar o Flutter e o Android SDK, utilizei o comando “flutter doctor” para verificar se o ambiente estava corretamente configurado. Esse comando foi útil para identificar possíveis problemas e garantir que todas as dependências necessárias estavam instaladas corretamente.

5. **Criar o Projeto Flutter:**

Para iniciar o desenvolvimento, executei o comando abaixo para criar o projeto com Flutter. Isso gerou uma estrutura básica de arquivos e pastas que eu poderia começar a personalizar conforme as necessidades da aplicação.

```
flutter create choosewine_frontend
```

```
cd choosewine_frontend
```

6. Inicializar e Rodar o Projeto:

Após criar o projeto, comecei o desenvolvimento inicial com o comando para rodar o app no emulador ou dispositivo físico.

7. Instalação de Dependências:

Para integrar o Firebase e outras funcionalidades necessárias, instalei as bibliotecas essenciais, como o `firebase_core`, `firebase_auth` e outras dependências que estariam no arquivo `pubspec.yaml`. A instalação foi feita com:

8. Uso do VS Code:

Para o desenvolvimento do código, utilizei o **Visual Studio Code (VS Code)** como editor. VS Code é uma IDE leve e altamente configurável, ideal para o desenvolvimento com Flutter. Utilizei extensões como o **Flutter** e o **Dart** para facilitar a escrita do código e garantir o suporte adequado para o Flutter.

5.4.2. Implementação do Login Multiplataforma

Após a criação de uma nova aplicação Flutter, a estratégia inicial foi focar no desenvolvimento da página de login, para que fosse possível testar os logins assíncronos desde o início. Essa abordagem ajudou a evitar manutenção pesada no futuro e garantiu que o processo de autenticação já fosse configurado corretamente, permitindo que um *token* de segurança fosse gerado. Esse processo de segurança será explicado detalhadamente no próximo capítulo, sobre Segurança e Gestão de Sessão.

O que é importante compreender neste momento é que o login seria a primeira etapa necessária para, posteriormente, permitir o acesso aos outros *endpoints* do *backend*. O login via Google foi implementado através do Firebase Authentication no *frontend*, onde o utilizador se autentica e recebe uma chave que é enviada para o *backend*. Esse, por sua vez, trabalha com o Firebase SDK para validar essa autenticação. Além disso, foi criado um *controller* personalizado que adiciona o utilizador simultaneamente à base de dados MongoDB, enquanto a autenticação no Firebase também é realizada.

Para configurar o Firebase, foi necessário criar uma conta e um projeto, a fim de obter as variáveis de ambiente que são aplicadas tanto no *backend* quanto no *frontend*, garantindo que as ferramentas da Firebase funcionem corretamente. Após a configuração, foram utilizados

controllers genéricos fornecidos pelo Firebase para as funções de login e logout, e essas funções podem ser encontradas no arquivo “auth_service” dentro da pasta “/services” no *frontend*.

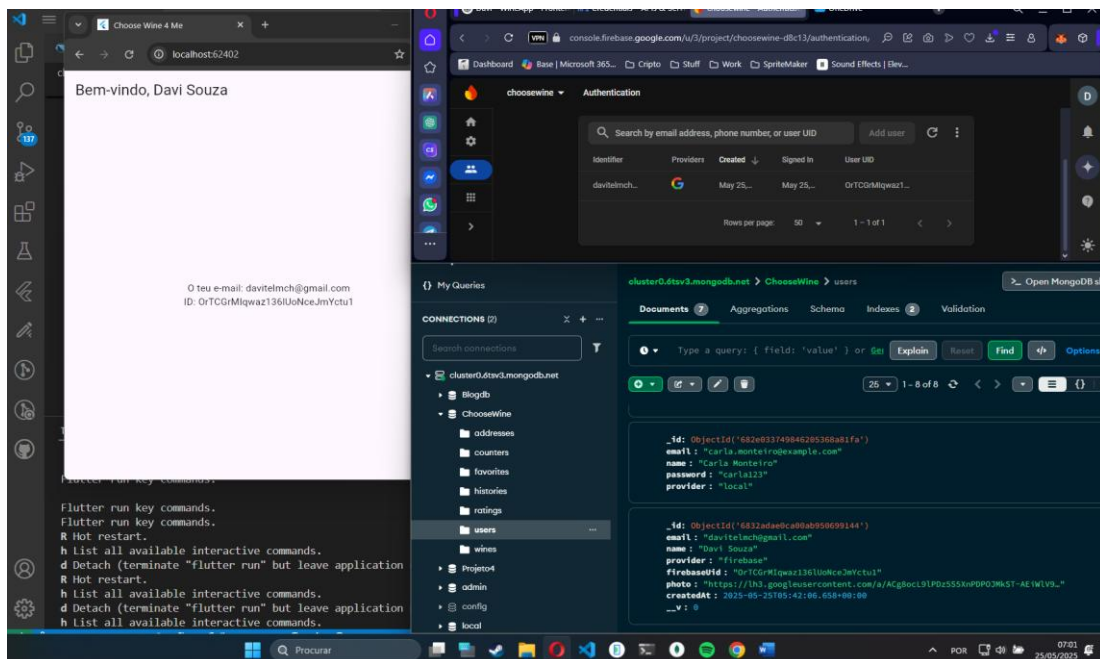


Figura 5-14- Login Multiplataforma feito pela Web

Na **Figura 5-14**, podemos ver o login realizado através da versão web da aplicação. Já na **figura -15 na próxima página** vemos o login feito no mobile. Em ambos os cenários, o utilizador é simultaneamente adicionado ao MongoDB e ao Firebase, funcionando de maneira semelhante, conforme detalhado posteriormente.

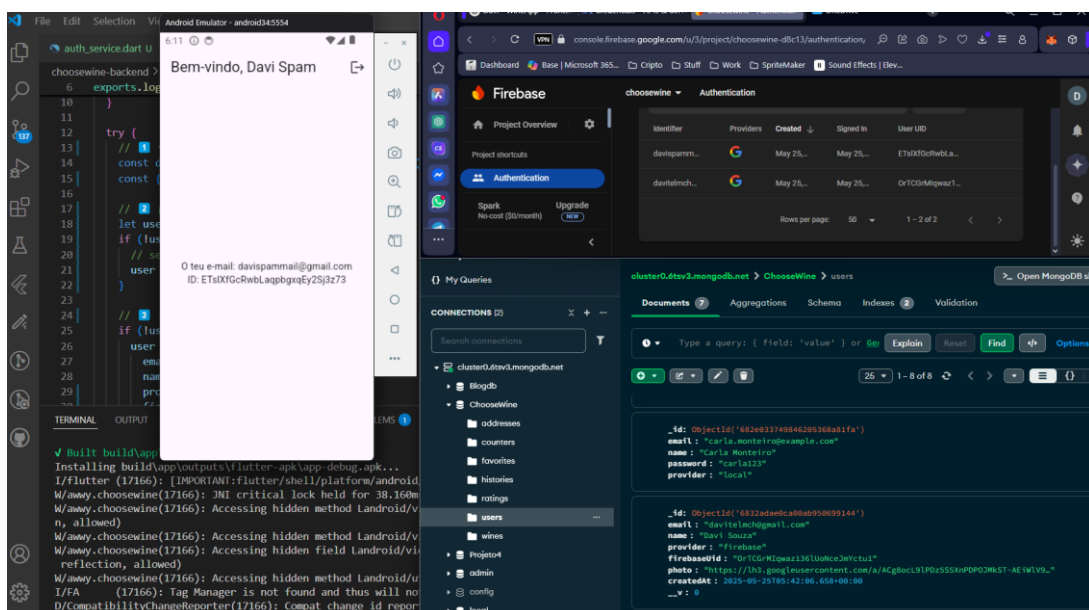


Figura 5-15 - Login Multiplataforma feito pelo Mobile

Além disso, foi implementado um sistema de **login e registo genérico** usando **email** como chave única, juntamente com **nome** e **senha**, sendo esta última criptografada antes de ser enviada para o servidor. Em contraste com os utilizadores que se autenticam através do **Firebase**, onde a senha não é necessária, neste caso foi necessário encriptar a senha e validá-la no *backend* antes que a chave de segurança fosse gerada.

Por fim, foram feitos pequenos ajustes na tabela de **utilizadores**, adicionando um campo para identificar se o registo foi feito através do **Firebase** ou do **email**, garantindo uma melhor gestão e entendimento do processo de autenticação.

5.4.3. Estratégias de Segurança e Gestão de Sessão

A comunicação segura entre o *frontend* e o *backend* desta aplicação foi estruturada de modo a garantir a integridade e a persistência da sessão do utilizador, assentando numa arquitetura avançada onde o serviço ApiService desempenha um papel central. Este serviço é responsável por intermediar todas as requisições HTTP do *frontend*, adicionando automaticamente o *token* JWT de autenticação emitido pelo *backend* a cada pedido protegido, e garantindo que a lógica de segurança e sessão está unificada num ponto único do código.

Após a autenticação federada do utilizador (via Firebase Authentication), o *backend* valida o *token* recebido e emite um JWT próprio, associado inequivocamente ao identificador interno do utilizador na base de dados. Ao contrário de abordagens que armazenam *tokens* de modo volátil ou dependem apenas do ciclo de vida do *frontend*, este JWT é guardado de forma persistente no cliente através do sistema SharedPreferences. Esta tecnologia permite que a aplicação retenha os dados essenciais da sessão — nomeadamente o token JWT e o identificador interno do utilizador — funcionando como uma versão multiplataforma e moderna dos cookies, compatível tanto com browsers como com aplicações móveis Android e iOS.

Esta estratégia assegura que a sessão permanece ativa mesmo após fechar o browser ou a aplicação, navegar entre separadores, ou reiniciar o dispositivo. Sempre que é necessário efetuar um pedido autenticado ao *backend*, o JWT guardado localmente é recuperado e incluído automaticamente pelo ApiService no *header* de autorização, garantindo que apenas utilizadores devidamente autenticados conseguem aceder a operações sensíveis ou dados protegidos. O controlo da sessão passa assim a ser totalmente centralizado no *backend*, que pode definir tempos de expiração, revogar *tokens*, monitorizar acessos ou implementar políticas de

segurança adaptadas à evolução da aplicação, sem depender do estado do *frontend* ou da autenticação federada.

Ao dissociar o processo de autenticação do mecanismo de autorização e ao optar por utilizar sempre o identificador interno do utilizador (em detrimento de identificadores externos, como o UID do Firebase), a arquitetura adotada impede ataques comuns de reutilização de *tokens*, manipulação de sessões ou tentativa de acesso a dados alheios. Todas as operações críticas estão protegidas por um ciclo de vida do *token* rigorosamente gerido pelo *backend*, reforçando a integridade e a segurança global do sistema.

O tempo investido nesta configuração revelou-se essencial para preparar a aplicação para cenários de utilização exigentes e escaláveis. A solução escolhida suporta facilmente múltiplos dispositivos e plataformas, facilita a implementação de funcionalidades futuras de segurança ou auditoria, e protege eficazmente os dados sensíveis dos utilizadores e os recursos críticos do sistema. Desta forma, foi estabelecida uma fundação tecnológica robusta, que conjuga a fluidez da experiência de utilização com elevados padrões de segurança e controlo de sessão.

5.4.4. Produção de páginas *Frontend*

Após ter preparado e testado toda a estrutura de login e a comunicação com o *backend*, passei ao desenvolvimento das páginas do *frontend*. A primeira etapa foi seguir ao máximo o fluxo previamente definido na prototipagem em Figma. A prototipagem anterior foi crucial para identificar rapidamente o que poderia ser transformado em componentes reutilizáveis e as páginas essenciais para o funcionamento da aplicação.

Com a estrutura inicial definida, fui avançando no desenvolvimento das páginas, e conforme necessário, fui adicionando as chamadas aos *endpoints* corretos no arquivo *api_service*, que, como explicado anteriormente, contém todas as chamadas ao *backend*. Para cada página, primeiramente estruturei a estrutura básica, adicionei os campos necessários e fiz as chamadas aos *endpoints* do *backend*. Após os testes de verificação, o design foi refinado.

A mesma abordagem foi aplicada aos componentes: criá-los, testá-los, aplicá-los na página correta e depois ajustar o design conforme necessário. Seguindo essa lógica e o modelo do Figma, consegui montar um *frontend* responsivo com o código bem estruturado e organizado, sem resíduos de código obsoleto ou repetições redundantes.

Além disso, foi realizada uma validação rigorosa nos campos, garantindo que todos os dados enviados para a base de dados estivessem em conformidade com os parâmetros aceitos. Isso garantiu a integridade dos dados e evitou problemas de incompatibilidade.

Foquei em garantir que o design fosse responsivo para uma variedade de dispositivos, incluindo smartphones e tablets, utilizando Flexbox e MediaQuery para adaptar o layout ao tamanho da tela. O design foi otimizado para garantir uma experiência fluida em diferentes dispositivos. Após concluir o design para mobile, comecei a criar algumas variantes para web, mas, para não comprometer os prazos, decidi adiar a conclusão do design para a web para uma melhoria futura, visto que não era um dos requisitos obrigatórios do projeto de estágio.

Para entender melhor o que funcionava bem na aplicação e onde poderia haver melhorias, foi integrado o Google Analytics. Essa ferramenta me ajudou a monitorizar a interação dos utilizadores com a aplicação, fornecendo dados valiosos que poderão ser usados para aprimorar o fluxo de navegação e funcionalidades no futuro. Esse processo me permitiu alcançar uma aplicação de que me orgulho de ter desenvolvido e tido a oportunidade de trabalhar. No próximo capítulo, vou mostrar os resultados e detalhar as funcionalidades de cada página.

6. Resultados Obtidos

Este capítulo apresenta os principais resultados do projeto, consolidando todo o trabalho desenvolvido nas fases anteriores. Após a conclusão da implementação dos diferentes módulos — *backend*, *frontend*, sistema de recomendação e *scraping* de dados para enriquecer a base de dados — foi possível integrar todos os componentes e obter uma aplicação funcional e coesa.

A implementação revelou-se um sucesso, cumprindo os objetivos propostos e demonstrando que a arquitetura definida foi eficaz. O sistema ChooseWINE4me oferece aos utilizadores uma experiência fluida, com funcionalidades essenciais como autenticação segura, inserção de dados pessoais, consulta e avaliação de vinhos, favoritos, e acima de tudo, recomendações personalizadas com base em inteligência artificial. A aplicação permite login multiplataforma com Google ou registo tradicional, suporta sessões persistentes, e garante a proteção dos dados com uma gestão segura de *tokens*.

Após a entrega funcional da aplicação, foi feita uma apresentação do projeto à equipa técnica da empresa. A receção foi bastante positiva: demonstraram-se satisfeitos com os resultados

obtidos e reforçaram que os objetivos propostos foram atingidos com excelência. No entanto, sugeriram alguns ajustes, sobretudo ao nível visual e da experiência do utilizador. Com base nesse feedback, procedi a um redesign parcial da interface, o que explica por que motivo os ecrãs apresentados nos próximos tópicos podem diferir ligeiramente do protótipo inicial desenvolvido no Figma.

Além disso, os serviços da Firebase — Authentication e Analytics — integraram-se de forma eficiente, garantindo segurança e recolha de dados relevantes para avaliar o comportamento dos utilizadores e apoiar decisões futuras de melhoria.

A seguir, serão apresentados os fluxos principais da aplicação, com uma explicação detalhada das páginas desenvolvidas e das funcionalidades disponíveis em cada uma delas. Cada subsecção foca-se numa parte do percurso do utilizador dentro da aplicação — desde o login até à visualização das recomendações —, destacando as decisões técnicas, os elementos visuais e as funcionalidades implementadas.

6.1. Fluxo Login

O fluxo de login é o ponto de entrada da aplicação *ChooseWINE4me* e foi concebido para oferecer uma experiência acessível, segura e intuitiva para diferentes perfis de utilizadores. A aplicação disponibiliza dois métodos principais de autenticação como vemos na **figura 6-1**.

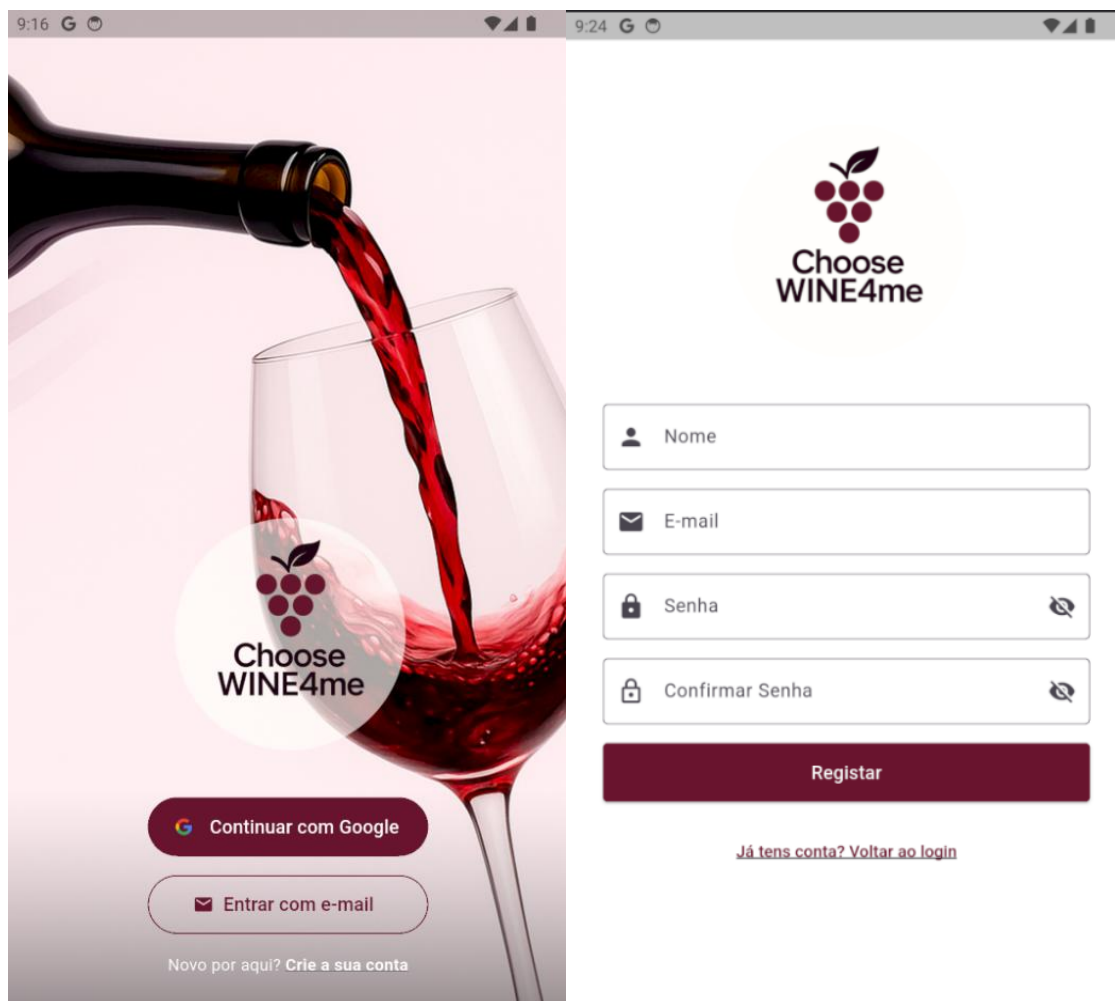


Figura 6-1 - Ecrãs de Login e Registro

Login com Google: utilizando o serviço Firebase Authentication, o utilizador pode autenticar-se rapidamente com a sua conta Google. Este método é multiplataforma (Android, iOS e Web) e oferece uma integração fluida, eliminando a necessidade de memorizar senhas e simplificando o acesso.

Login com email e password: para quem prefere o método tradicional, está disponível um formulário de autenticação com email e palavra-passe. Caso o utilizador ainda não tenha conta, pode facilmente aceder ao **formulário de registo**, onde são solicitados Nome, Email e Password.

Todos os dados são validados antes do envio, e o sistema garante a encriptação segura da palavra-passe antes do seu armazenamento.

Após o primeiro desenvolvimento funcional destes ecrãs, foi feito um **redesign com base no feedback da equipa técnica da empresa**, resultando numa interface mais apelativa e moderna.

O novo layout apresenta cores mais equilibradas, botões com melhor destaque visual e uma hierarquia de informação mais clara, o que melhora significativamente a experiência do utilizador. A organização visual dos campos e a fluidez das animações tornaram o processo de login mais agradável e profissional.

Estas melhorias no design refletem uma preocupação em alinhar a estética da aplicação com padrões atuais de usabilidade e design de interfaces, promovendo uma primeira impressão positiva junto dos utilizadores.

6.2. Página de inserção da Morada

A página de inserção da morada é apresentada **exclusivamente a novos utilizadores**, imediatamente após a conclusão do processo de registo. Esta etapa foi concebida para recolher informação geográfica básica, essencial para o funcionamento do sistema de **recomendações colaborativas baseadas na localidade**.

Ao fornecer a sua morada, o utilizador permite que o sistema identifique outros utilizadores com perfis semelhantes na mesma região. Isso possibilita que, mesmo sem um histórico de avaliações ou favoritos, o utilizador possa receber sugestões de vinhos populares entre pessoas com hábitos geográficos semelhantes — mitigando o problema do *cold-start* típico em sistemas de recomendação.

São solicitados os seguintes campos obrigatórios: **País, Cidade, Morada, Código Postal**.

Todos os dados são validados no *frontend* antes de serem enviados para o *backend*, garantindo que o formulário seja preenchido corretamente. A base de dados MongoDB armazena esta informação de forma segura, associando-a ao utilizador autenticado.

Os campos foram organizados de forma mais clara e intuitiva, e o botão de submissão está destacado com uma cor de contraste para facilitar a navegação. Após a submissão bem-sucedida da morada, o utilizador é automaticamente redirecionado para a **página inicial** da aplicação, onde pode explorar os vinhos e aceder às funcionalidades disponíveis.

6.3. Página Inicial

A página inicial da aplicação ChooseWINE4me funciona como o **hub central** a partir do qual o utilizador pode navegar para as principais funcionalidades. Esta interface foi pensada para ser

funcional e ao mesmo tempo visualmente apelativa, oferecendo **acesso rápido e intuitivo aos conteúdos mais relevantes** da aplicação (**Figura 6-2**).

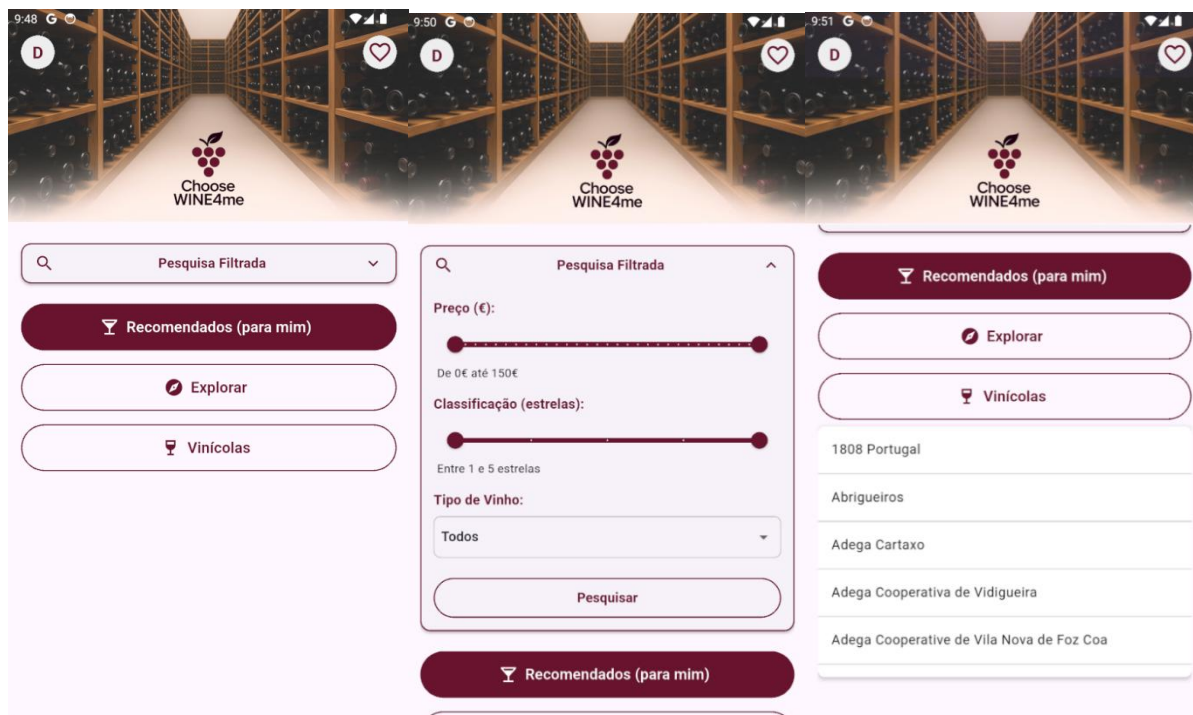


Figura 6-2 - Home Page e Widgets

No topo da página, é apresentado um conjunto de ícones de navegação:

Um ícone circular com a inicial do nome do utilizador, que funciona como botão para aceder ao perfil. Esta representação personalizada reforça a identidade do utilizador e facilita o reconhecimento visual. Na página de perfil é possível visualizar e editar a morada e também fazer *logout*.

Existe também ícone de coração, que permite aceder diretamente à secção de favoritos, onde o utilizador pode consultar os vinhos que guardou.

Logo abaixo, está disponível uma barra de pesquisa com filtros avançados, que permite ao utilizador procurar vinhos de forma mais direcionada. Os principais campos de filtragem incluem: Tipo de vinho (ex.: Tinto, Branco, Rosé), Região (ex.: Alentejo, Douro, Dão), Faixa de preço, Classificação média.

Estes filtros podem ser usados individualmente ou combinados, tornando o processo de descoberta mais eficiente. A baixo da pesquisa, o utilizador encontra os fluxos principais de descoberta:

Explorar todos os vinhos: apresenta a lista completa de vinhos disponíveis na base de dados. Os vinhos são apresentados com imagem, nome, preço e classificação, permitindo ao utilizador navegar livremente e descobrir novas opções.

Recomendados para mim: esta secção utiliza o sistema de inteligência artificial para apresentar vinhos personalizados, com base nas preferências, avaliações e localização do utilizador. Se o utilizador ainda não tiver interagido com a aplicação, são utilizadas as preferências da comunidade da mesma cidade.

Vinícolas populares: apresenta uma lista de vinícolas (produtor), permitindo ao utilizador conhecer as marcas mais relevantes ou descobrir vinhos semelhantes dentro da mesma casa produtora.

Este layout proporciona uma navegação fluida e clara, equilibrando personalização e descoberta livre. A estrutura modular da página inicial foi desenhada para permitir escalabilidade futura, como a introdução de promoções, rankings ou sugestões sazonais.

6.4. Página Pesquisa

A **página de pesquisa** apresenta os **resultados obtidos a partir da pesquisa filtrada** realizada na página inicial. Esta secção tem como objetivo ajudar o utilizador a encontrar vinhos que correspondam exatamente às suas preferências, facilitando a navegação e a descoberta de novas opções de forma eficiente.

Os vinhos são apresentados numa lista visual limpa e funcional, onde cada cartão contém: Imagem do vinho, Nome, Tipo (Tinto, Branco, Rosé...), Região, Preço médio, Classificação média (rating). Além dos filtros por tipo, região, preço e avaliação, o utilizador pode também pesquisar diretamente pelo nome do vinho, tornando a ferramenta útil tanto para descobertas como para encontrar rapidamente um vinho específico.

Dado que o foco principal da aplicação é a descoberta e avaliação de vinhos, a interface destas páginas dá especial destaque aos rankings e avaliações de cada vinho. Isso incentiva os utilizadores a explorar novas sugestões com base na reputação entre outros utilizadores, contribuindo para uma experiência de navegação mais informada e personalizada.

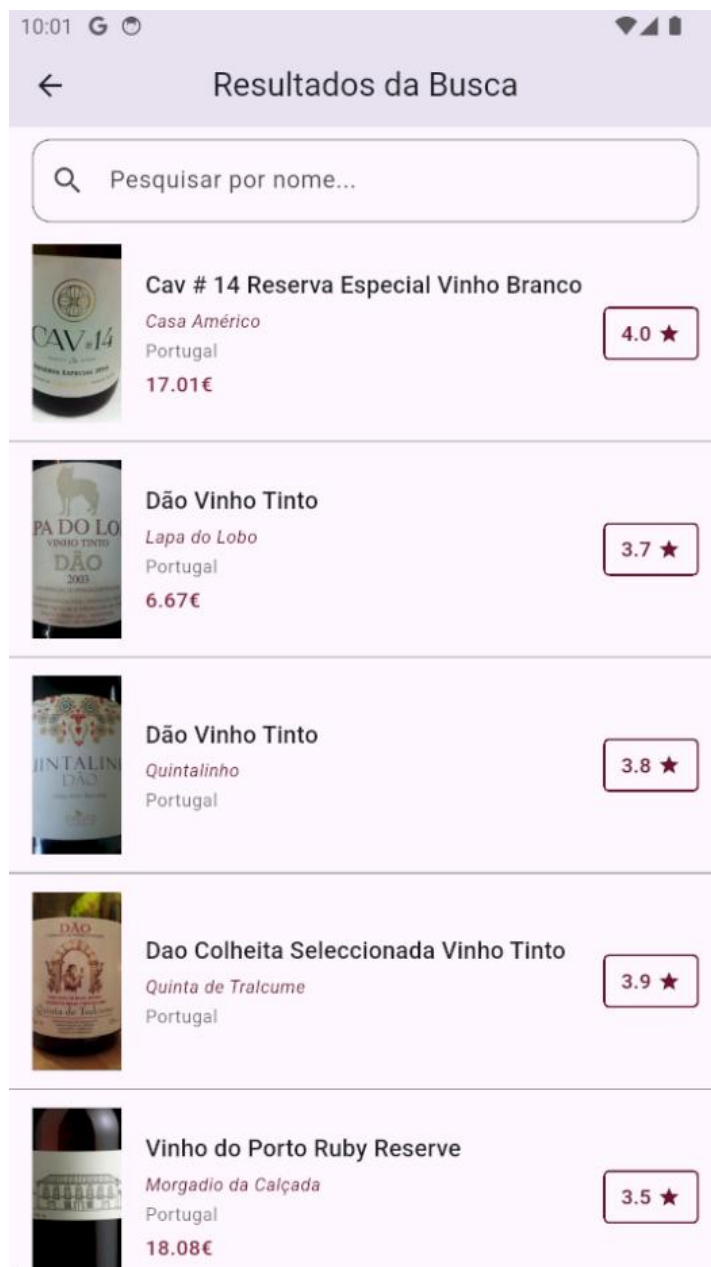


Figura 6-3 - Página de Resultados

O layout da página de resultados que podemos ver na **figura 6-3** e a página "Explorar", acessível a partir do botão principal na *home*, seguem um layout idêntico, mas apresenta todos os vinhos disponíveis na base de dados, sem filtros aplicados. Este modo de navegação foi desenvolvido com *scroll* infinito, permitindo que o utilizador continue a carregar novos resultados automaticamente à medida que desliza pela página, sem necessidade de clicar em “Próxima Página” ou realizar carregamentos manuais.

6.5. Página de detalhes do Vinho

A **página do vinho** apresenta uma **visão detalhada de um vinho específico** selecionado pelo utilizador. Esta página foi pensada para fornecer toda a informação relevante de forma clara, organizada e visualmente atrativa.

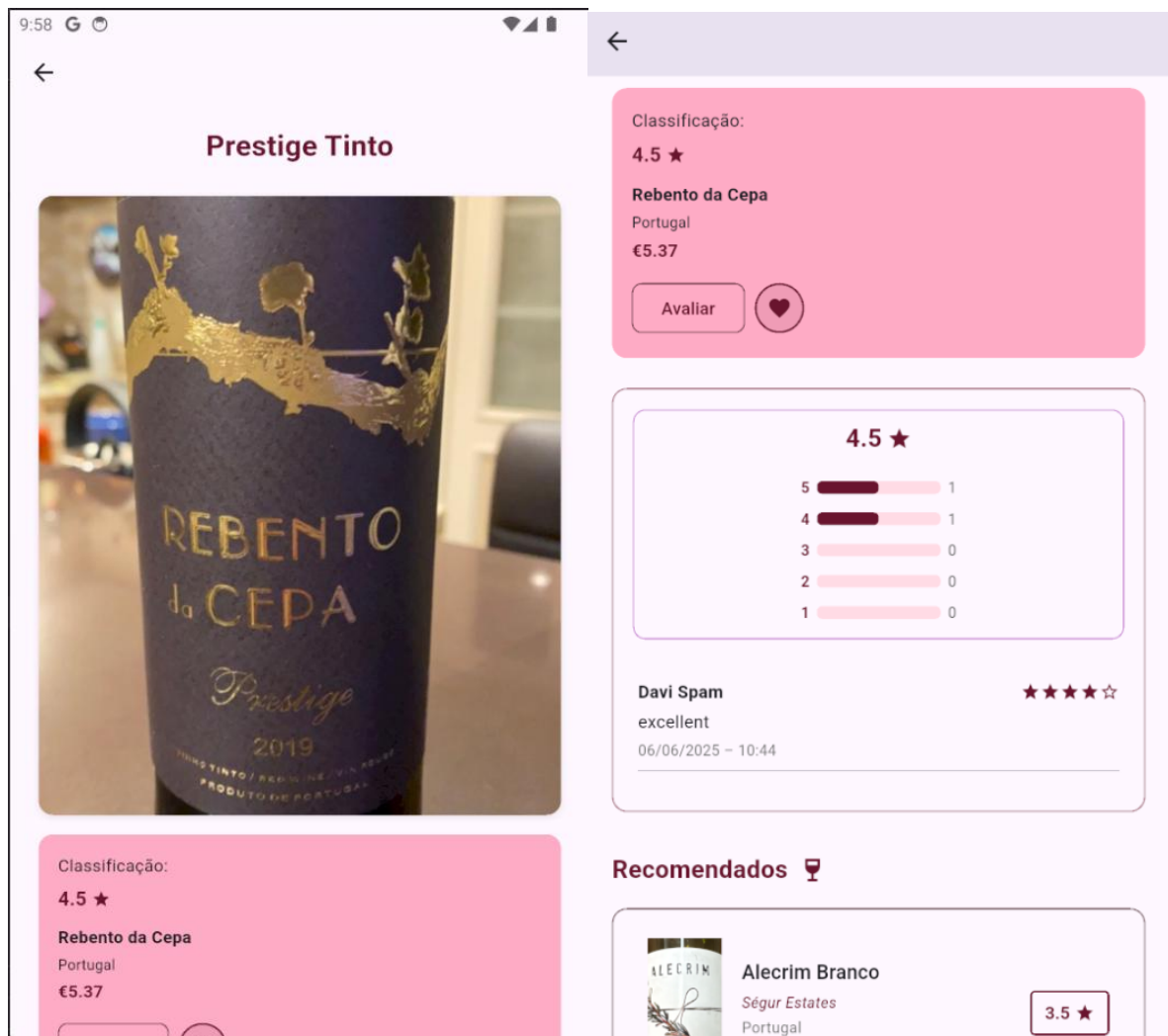


Figura 6-4 - Página de detalhes do Vinho

No topo da página da **figura 6-4**, são apresentados os **detalhes principais do vinho**, nomeadamente:

- Nome do vinho
- Imagem do rótulo
- Classificação média

-
- Nome da vinícola (**winery**)
 - País de origem
 - Teor alcoólico quando disponível a informação.
 - Preço médio

Abaixo dessas informações, o utilizador encontra dois botões principais:

- **Favorito:** permite guardar o vinho na sua lista pessoal de favoritos.
- **Avaliar:** ao pressionar este botão, é exibido um *overlay* interativo que permite ao utilizador submeter uma avaliação.

Neste *overlay* de avaliação, o utilizador pode:

- Atribuir uma pontuação de **1 a 5 estrelas**, bastando clicar nas estrelas correspondentes.
- **Adicionar (ou não) um comentário**, de forma opcional.
- Submeter a avaliação, que será imediatamente guardada na base de dados.

As avaliações são **editáveis e removíveis** no futuro, permitindo que o utilizador possa atualizar a sua opinião ou corrigir comentários anteriores, o que torna o sistema mais flexível e transparente.

Abaixo do bloco de informações principais do vinho, existe uma *tab* dedicada às avaliações. Esta secção apresenta:

- Um gráfico resumido com o **número de avaliações por cada nível de estrela** (1 a 5).
- A lista de comentários de utilizadores, organizada por ordem **mais recente primeiro**.
- A navegação por **página**, com **5 comentários por página**, para garantir que a leitura seja fluida e sem sobrecarga visual.

Por fim, na parte inferior da página, é exibido o **bloco de “Recomendados para ti”**, que apresenta vinhos sugeridos com base nas preferências e interações do utilizador. Esta secção é alimentada pelo sistema de **recomendações inteligentes por inteligência artificial**, que será detalhado na próxima secção: **6.6 Página de Recomendações**.

6.6. Página de Recomendações

A **página de recomendações** é uma das mais importantes da aplicação *ChooseWINE4me*, pois reflete diretamente o **módulo de inteligência artificial** desenvolvido para oferecer uma experiência personalizada e relevante para cada utilizador.

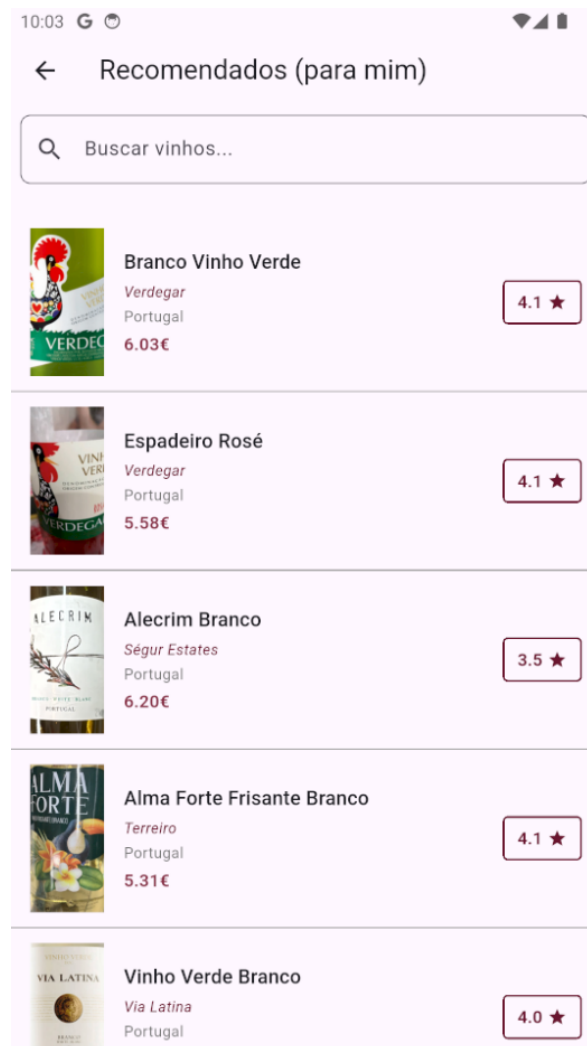


Figura 6-5 - Resultados das Recomendações

O layout da página, como vemos na **figura 6-5**, segue o mesmo padrão das outras páginas de listagem, com **cartões de vinhos dispostos em scroll infinito**, que vão sendo carregados automaticamente à medida que o utilizador desliza o ecrã. Cada vinho apresentado inclui: Imagem, Nome, Classificação média, Tipo, região e preço

No entanto, a **grande diferença desta página** em relação às restantes é que **os vinhos aqui apresentados são diferentes para cada utilizador**, pois são gerados de forma dinâmica com base no seu comportamento dentro da aplicação.

O sistema analisa:

- **Vinhos favoritos,**
- **Vinhos avaliados,**
- **Interações anteriores,** como visualizações recentes

Através deste histórico, o algoritmo compara as preferências do utilizador com os outros vinhos disponíveis na base de dados, e ordena os resultados por **grau de similaridade**, apresentando primeiro os mais compatíveis com o seu gosto pessoal.

Nos casos em que o utilizador ainda não interagiu com a aplicação (ex: utilizadores novos), a aplicação recorre a uma estratégia de recomendação colaborativa baseada na morada, onde são sugeridos os vinhos mais populares entre outros utilizadores da mesma cidade ou região.

O design desta página mantém a consistência visual com as outras secções, incluindo a exibição das avaliações e ratings, mas destaca-se pela ordenação inteligente dos resultados, que proporciona uma experiência de descoberta mais útil e direcionada.

Desta forma, a página de recomendações representa o culminar da proposta da aplicação: ajudar cada utilizador a descobrir vinhos novos, relevantes e com elevada probabilidade de satisfação, de forma prática, rápida e personalizada.

6.7. Página de Favoritos

A **página de favoritos** permite ao utilizador visualizar rapidamente todos os vinhos que adicionou à sua lista pessoal de favoritos ao longo da navegação pela aplicação *ChooseWINE4me*.

A apresentação segue o mesmo **layout limpo e funcional** das outras páginas de listagem, com os cartões de vinho a exibirem: Imagem, Nome, Tipo e região, Classificação média e Preço

Cada cartão contém ainda o **ícone de coração preenchido**, indicando que o vinho faz parte da lista de favoritos. Este ícone funciona como um **botão de alternância**: ao clicar nele, o utilizador pode **remover o vinho da lista de favoritos em tempo real**.

Para garantir que a interface se mantém atualizada, a aplicação inclui a funcionalidade de **“pull to refresh”** — ou seja, o utilizador pode **arrastar do topo para baixo para atualizar a lista**,

especialmente útil após remover favoritos. Este gesto é simples e intuitivo, garantindo uma experiência dinâmica sem necessidade de navegar para trás ou reiniciar a aplicação.

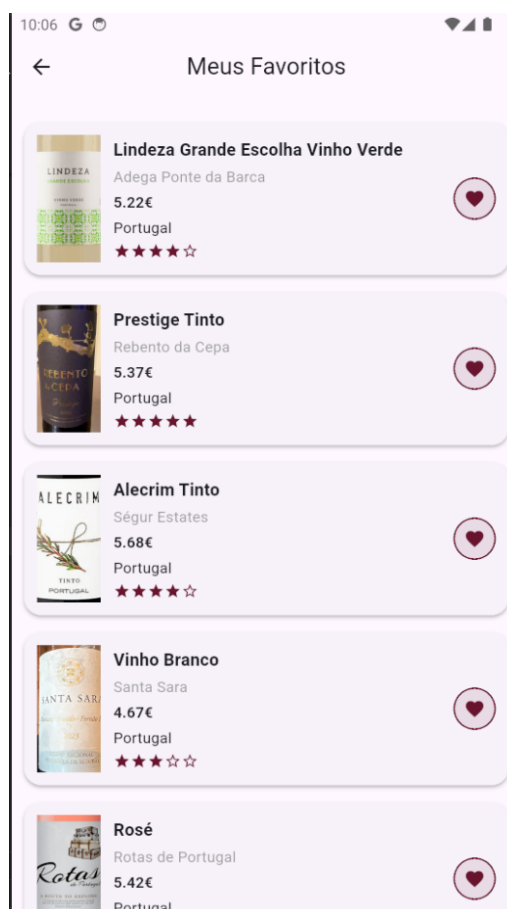


Figura 6-6 - Página dos Favoritos

A página de favoritos da **figura 6-6** foi concebida para ser uma **área de acesso rápido aos vinhos preferidos**, funcionando como um atalho para consulta ou reavaliação posterior, e mantendo a consistência visual e de navegação com o resto da aplicação.

7. Conclusão

Este capítulo final do relatório tem como objetivo refletir sobre os resultados alcançados ao longo do desenvolvimento da aplicação ChooseWINE4me, comparando o plano inicialmente

proposto com os resultados efetivamente obtidos. Pretende-se demonstrar como o projeto evoluiu, atingindo os objetivos traçados de forma sólida e satisfatória, e culminando numa aplicação funcional, personalizada e tecnicamente robusta — um resultado de que me orgulho profundamente e que foi muito bem recebido pela equipa técnica da empresa onde o projeto foi desenvolvido.

Ao longo das secções seguintes, será feita uma análise dos principais contributos alcançados, destacando os pontos mais relevantes e as decisões que mais impactaram positivamente a construção do produto. Serão também apresentadas algumas melhorias possíveis, com base na experiência adquirida e no feedback recebido durante o processo de validação.

Adicionalmente, será proposta uma linha de trabalho futuro, identificando oportunidades de expansão, refinação de funcionalidades e escalabilidade do sistema.

Por fim, é apresentada uma autorreflexão crítica sobre o meu desempenho pessoal, as estratégias que adotei face aos desafios encontrados, e as aprendizagens mais significativas resultantes desta experiência. Esta análise pessoal encerra o ciclo do estágio e reforça o crescimento técnico e profissional proporcionado por este projeto.

7.1. Principais Resultados e Contribuições

A aplicação *ChooseWINE4me* foi concebida com o objetivo de ajudar os utilizadores a **descobrirem e escolherem vinhos de forma personalizada**, através de um sistema inteligente que combina a experiência do utilizador com dados reais e preferências locais.

Durante o estágio, foi possível:

- **Implementar todos os módulos previstos** no plano de trabalho, incluindo backend, frontend, sistema de recomendação e integração com serviços externos como Firebase e MongoDB.
- Garantir uma **autenticação segura e multiplataforma**, com suporte a login via Google ou registo tradicional.
- Desenvolver um **sistema de recomendação personalizado**, utilizando machine learning e estratégias colaborativas, adaptando-se tanto ao comportamento do utilizador como à sua localização geográfica.

-
- Criar uma **interface fluida, responsiva e intuitiva**, com atenção ao design e à experiência do utilizador, fruto de um processo de *redesign* baseado em feedback direto da equipa técnica.
 - Organizar o projeto com base numa **estrutura modular, escalável e segura**, permitindo manutenção futura e possíveis evoluções do sistema.

Como resultado, foi construída uma aplicação **completa, funcional e bem estruturada**, que não só cumpriu os requisitos técnicos como também superou as expectativas iniciais, nomeadamente pela **integração bem-sucedida entre todos os componentes** e pela **capacidade de gerar recomendações relevantes e dinâmicas**.

Este projeto representou um **desafio técnico ambicioso**, mas permitiu-me aplicar e consolidar conhecimentos adquiridos ao longo da formação em Tecnologias e Design Multimédia. Além disso, a aplicação final foi **bem acolhida pela equipa técnica da empresa**, que reconheceu o esforço, a organização e a qualidade geral do trabalho entregue.

A nível pessoal e profissional, considero que os resultados obtidos **refletem um compromisso com a qualidade, a aprendizagem contínua e a superação de obstáculos técnicos**, características que pretendo continuar a desenvolver na minha carreira.

7.2. Melhorias

Num projeto desta escala, especialmente **desenvolvido por uma única pessoa num prazo limitado de poucos meses**, é natural — e até saudável — reconhecer que existem sempre aspetos que poderiam ser melhorados. Estar ciente disso demonstra maturidade profissional e abertura à evolução contínua.

Embora o resultado tenha superado as expectativas iniciais, considero que **há áreas da aplicação que poderiam ser refinadas ou expandidas** para oferecer uma experiência ainda mais completa ao utilizador. Destaco, em particular:

- A **apresentação visual da página de detalhe do vinho**, que poderia ser mais envolvente e informativa, com animações suaves ou uma secção de destaques para vinhos premiados.
- A **página de perfil do utilizador**, que neste momento é funcional, mas poderia beneficiar de uma reorganização visual e mais opções de personalização.

-
- A adição de uma **página de histórico de vinhos visualizados**, que permitiria ao utilizador rever facilmente os vinhos que explorou recentemente.
 - A criação de uma funcionalidade para **adicionar os seus próprios vinhos à aplicação**, ideal para enófilos ou produtores locais que queiram partilhar as suas descobertas com a comunidade.
 - E, por fim, a possibilidade de **partilhar vinhos com amigos dentro da aplicação**, permitindo criar uma rede de seguidores e receber sugestões entre utilizadores com gostos semelhantes.

Apesar destas oportunidades de melhoria, estou **genuinamente satisfeito com os resultados alcançados**. A base desenvolvida é sólida e bem estruturada, e a arquitetura modular adotada garante que futuras melhorias ou novas funcionalidades possam ser implementadas **de forma eficiente e sustentável**.

7.3. Trabalho Futuro

Caso o projeto *ChooseWINE4me* venha a ser aplicado em contexto real de mercado, existem várias **possibilidades de expansão e monetização** que podem ser exploradas, para além das melhorias técnicas já referidas na secção anterior.

Um dos caminhos mais promissores seria **transformar a aplicação numa plataforma de venda direta de vinhos**, aproveitando os dados recolhidos sobre os vinhos mais populares e bem avaliados. A partir dessa informação, poderia ser feito o **aprovisionamento de stock dos vinhos mais recomendados** e, posteriormente, disponibilizar esses produtos para venda diretamente na aplicação. Esta abordagem permitiria **gerar receita de forma direta**, mantendo a experiência centrada nas preferências do utilizador.

Como alternativa — ou em complemento — poderia ser implementado um **sistema de registo de fornecedores**, permitindo que produtores e distribuidores de vinho disponibilizassem os seus catálogos na plataforma. Cada fornecedor indicaria os seus **preços, disponibilidade e localização**, sendo que a plataforma reteria uma **percentagem sobre cada venda concretizada** (valor a definir posteriormente). Este modelo de negócio, baseado numa **comissão por intermediação**, é viável e já validado em outras áreas de e-commerce.

Ambas as abordagens, aliadas às melhorias funcionais e visuais sugeridas anteriormente, permitiriam uma **transição natural da aplicação enquanto recomendador inteligente para**

um verdadeiro mercado digital de vinhos, com recomendações personalizadas, compras integradas e comunidade ativa.

É importante reforçar que estas ideias devem ser **validadas com especialistas da área do vinho, do marketing e da gestão empresarial**, de modo a garantir a viabilidade legal, logística e financeira do projeto. O que aqui se apresenta é uma **proposta de idealização**, com o objetivo de demonstrar o **potencial de evolução do projeto com o devido investimento e atenção estratégica**.

7.4. Autorreflexão

O desenvolvimento deste projeto foi uma oportunidade única de aplicar, de forma integrada, os conhecimentos adquiridos ao longo da minha formação. Trabalhar num projeto real, com objetivos concretos e um público-alvo definido, exigiu organização, autonomia e capacidade de adaptação a imprevistos — competências que considero ter consolidado ao longo deste processo.

Enfrentei diversos desafios técnicos e de gestão de tempo, mas consegui superá-los com persistência e uma abordagem prática orientada à solução. Sinto que evoluí tanto a nível técnico como pessoal, e termino este projeto com um sentimento de satisfação pelo percurso realizado e pelo produto final entregue.

Referências Bibliográficas

Axios. (2025). *Axios – Promise based HTTP client for the browser and node.js* [Biblioteca]. <https://axios-http.com/>

Dart Team. (2025). *Dart programming language* [Linguagem de programação]. <https://dart.dev/>

Expo. (2025). *Expo* [Plataforma]. <https://expo.dev/>

FastAPI. (2025). *FastAPI* [Framework]. <https://fastapi.tiangolo.com/>

Firebase. (2025). *Firebase* [Plataforma]. Google. <https://firebase.google.com/>

Flutter. (2025). *Flutter* [Framework de UI]. Google. <https://flutter.dev/>

Flutter China. (2025). *Dio: Powerful HTTP client for Dart* [Biblioteca]. <https://pub.dev/packages/dio>

GitHub, Inc. (2025). *GitHub* [Plataforma]. <https://github.com/>

GitLab, Inc. (2025). *GitLab* [Plataforma]. <https://about.gitlab.com/>

Microsoft Corporation. (2025). *Visual Studio Code* [Software]. <https://code.visualstudio.com/>

MongoDB, Inc. (2025). *MongoDB* [Software]. <https://www.mongodb.com/>

Meta Platforms, Inc. (2025). *React Native* [Framework]. <https://reactnative.dev/>

OpenJS Foundation. (2025). *Node.js* [Ambiente de execução]. <https://nodejs.org/>

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research*, 12, 2825–2830. <https://scikit-learn.org/>

Postman, Inc. (2025). *Postman* [Software]. <https://www.postman.com/>

Softinsa. (2024). *Softinsa distinguida como um dos melhores lugares para trabalhar em Portugal e na Europa*. Recuperado de <https://www.softinsa.pt/>

SharedPreferences. (2025). *SharedPreferences* [Biblioteca]. Flutter Team.
https://pub.dev/packages/shared_preferences

StrongLoop, IBM & Contributors. (2025). *Express.js* [Framework]. <https://expressjs.com/>

TensorFlow Team. (2025). *TensorFlow* [Biblioteca]. <https://www.tensorflow.org/>


8. Anexo A

Plano de Estágio Curricular

Fornecido pela orientadora **Vanessa Madeira** da empresa acolhedora **Softinsa**.

Plano Atualizado do Estágio Curricular Davi (19 de Fevereiro – 2 de Junho, 420 horas)

Fase 1: Planeamento e Configuração (2 semanas - 48 horas)

 **Objetivo:** Configurar o ambiente de desenvolvimento e explorar tecnologias.

Semana 1 e 2:


- Apresentação do projeto modular (explicar que o sistema pode ser expandido para outros tipos de conteúdo).
- Configuração do ambiente:
 - Backend: **Node.js, Express e MongoDB + Mongoose.**
 - Frontend: React Native.
 - IA: **Python (Scikit-learn)**
 - Firebase: Firebase Authentication e Crashlytics
- Definição detalhada dos requisitos e funcionalidades da aplicação.
- Criação de um repositório Git para controlo de versões.
- Estudo da documentação da API Vivino ou similares (ou base de dados de vinhos).
 - <https://www.postman.com/spoonacular-api/spoonacular-api/collection/rqqne3j/spoonacular-api>
 - <https://www.winespectator.com/dailypicks>
 - Possibilidade de vivino + dados locais
- Criação de um layout base da interface de utilizador (wireframes).
- Criação do esquema da base de dados **MongoDB.**
- Configuração do Firebase Authentication e Crashlytics no projeto.

Materiais e Tutoriais:

 [Configuração de React Native](#)

- Links alternativos: [Youtube](#)

 [Setup do Node.js e Express](#)

 Guia de integração da TMDb com React Native: "[Using TMDb API for Movie Apps](#)". (exemplo)

 [Documentação da API TMDb](#)

 [Documentação da Scikit-learn](#)


 [Configuração de Firebase.](#)

 [Documentação do MongoDB](#)

- [Com React-Native](#)

- [MongoDB & Mongoose](#)

Fase 2: Desenvolvimento do Backend com Node.js e MongoDB (8 semanas - 192 horas)

 **Objetivo:** Configuração do ambiente Node.js e instalação das dependências necessárias.

- Instalar **Node.js**, **npm/yarn** e configurar MongoDB Atlas.
- Instalar **dependências essenciais**:
 - Express (Framework da API).
 - Mongoose (ORM para MongoDB).
 - Firebase Admin (para autenticação).
 - dotenv (para variáveis de ambiente).
- Criar **variáveis de ambiente** para credenciais do Firebase e MongoDB.

Tarefas Detalhadas:

Semana 3 - 11:

1. **Configuração do Ambiente Node.js e MongoDB (Semana 3)**

- Instalar **Node.js**, **npm/yarn** e configurar MongoDB Atlas.
- Instalar **dependências essenciais**:
 - Express (Framework da API).
 - Mongoose (ORM para MongoDB).
 - Firebase Admin (para autenticação).
 - dotenv (para variáveis de ambiente).
- Criar **variáveis de ambiente** para credenciais do Firebase e MongoDB

2. **Modelação da Base de Dados (Semanas 3-4)**

- Criar coleções no **MongoDB** para armazenar:
 - **Vinhos** (nome, região, tipo de uva, teor alcoólico, harmonizações, avaliações, etc.).

- **Utilizadores** (ID do Firebase, preferências, histórico, vinhos favoritos).
- **Avaliações** (ID do utilizador, ID do vinho, nota de 1 a 5).
- Criar relações entre documentos com **Mongoose schemas**.
- Implementar a lógica para processar e formatar os dados.

3. Criar API RESTful (Semanas 4-6)

- Criar endpoint modular rotas para:
 - **/vinhos** → Listar vinhos por tipo, região, popularidade.
 - **/vinhos/:id** → Detalhes de um vinho específico.
 - **/utilizadores/:id** → Gerir perfis de utilizadores.
 - Criar e atualizar perfis de utilizador.
 - Obter dados do perfil de um utilizador.
 - Gerir preferências do utilizador.
 - Gerir o histórico de visualização do utilizador.
 - **/avaliacoes** → Registrar avaliações dos utilizadores.
 - **/recomendacoes (6ª-7ª Semana):**
 - Integrar a lógica de recomendação do **Scikit-learn** (a ser desenvolvida na Fase 3) no backend.
 - Criar rotas para:
 - Gerar recomendações personalizadas para um utilizador.
 - Retornar a lista de vinhos/castas recomendados

1. Testes e Refinamento do Backend (7ª-8ª Semana):

- Escrever testes unitários e de integração para as rotas da API.
- Testar a comunicação entre o backend e a base de dados MongoDB.
- Refinar o código e otimizar o desempenho do backend.
- Documentar as rotas da API

Fase 3: Implementação do Sistema de Recomendação com IA (6 semanas - 144 horas)

 **Objetivo:** Criar o sistema de recomendação de vinhos usando **Scikit-learn**

Tarefas Detalhadas:

 **Semana 12- 18:**

1. Preparação dos Dados (Semanas 12-13)

- Criar **scripts em Python** para extrair avaliações do MongoDB.
- Converter dados categóricos (ex.: tipos de uva, regiões) em **valores numéricos**.
- Criar matriz de utilizador-vinho para recomendações.

2. Desenvolvimento do Modelo de IA (Semanas 13-15)

- Implementar **Filtragem Baseada em Conteúdo** (ex.: recomendar vinhos com características semelhantes aos favoritos).
- Implementar **Filtragem Colaborativa** (ex.: recomendar vinhos que utilizadores semelhantes avaliaram bem).
- Ajustar **parâmetros do modelo para otimizar a precisão**.

3. Integração do Modelo com o Backend (Semanas 15-16)

- Criar uma **API Flask** em Python para expor as recomendações.
- Implementar **rota /recomendacoes no backend Node.js**, que faz um pedido à API Flask.
- Testar integração com **Postman**.

4. Testes e Otimização (Semanas 17-18)

- Testar recomendações com utilizadores reais.
- Melhorar a experiência ajustando os resultados com base no feedback.

Fase 4: Desenvolvimento da Interface com React Native e Firebase (4 semanas - 96 horas)

 **Objetivo:**

- Criar uma interface de utilizador intuitiva e responsiva em React Native.
- Implementar a autenticação de utilizadores com Firebase Authentication no frontend.
- Integrar o Firebase Crashlytics para monitorização de erros.
- Consumir as rotas da API do backend Node.js para exibir dados e recomendações.

Tarefas Detalhadas:

Semana 19 - 23:

1. Configuração do Ambiente React Native e Firebase (1ª Semana):

- Configurar o ambiente de desenvolvimento React Native (Expo ou React Native CLI).
- Instalar as dependências necessárias (Firebase SDK, Axios, etc.).
- Configurar o Firebase no projeto React Native.
- Configurar o Crashlytics.

2. Desenvolvimento dos Ecrãs de Autenticação (1ª-2ª Semana):

- Criar os ecrãs de registo, login e recuperação de password.
- Implementar a lógica de autenticação com Firebase Authentication.
- Gerir o estado de autenticação do utilizador na aplicação.

3. Desenvolvimento dos Ecrãs Principais (2ª-3ª Semana):

- Criar os ecrãs de:
 - Ecrã inicial com recomendações personalizadas.
 - Ecrã de pesquisa de vinhos.
 - Ecrã de detalhes de um vinho.
 - Ecrã de Avaliações e Preferências.
 - Consumir rotas do backend para mostrar **dados e recomendações**.
 - Implementar funcionalidades como **favoritos e avaliações**
- Integrar o Crashlytics para reportar erros.
- Gerir o estado da aplicação (loading, erros, etc.).

Fase 5: Testes, Refinamento e Documentação (2 semanas - 48 horas)

Objetivo:

- Testes abrangentes da aplicação em diferentes dispositivos e emuladores.
- Refinamento da interface de utilizador e da experiência do utilizador (UX).
- Otimização do desempenho da aplicação, do backend e do sistema de recomendação.
- Criação de documentação técnica do projeto, incluindo integração com Firebase.
- Planeamento e escrita do relatório final do projeto.

Tarefas Detalhadas:

Semana 24 - 26:

1. Testes da Aplicação (1ª Semana):
 - Testar a aplicação em diferentes dispositivos e emuladores (iOS e Android).
 - Testar a funcionalidade de autenticação com Firebase Authentication.
 - Testar a integração com a API do backend Node.js.
 - Testar a lógica de recomendação.
 - Utilizar o Crashlytics para monitorizar erros e relatórios de falhas.
 - Realizar testes de usabilidade com utilizadores reais.
2. Refinamento da Interface e UX (1ª-2ª Semana):
 - Refinar a interface de utilizador com base nos resultados dos testes.
 - Melhorar a experiência do utilizador (UX) com base no feedback dos utilizadores.
 - Otimizar o desempenho da aplicação (tempo de carregamento, consumo de bateria, etc.).
3. Documentação Técnica (2ª Semana):
 - Criar documentação técnica do projeto, incluindo:
 - Arquitetura da aplicação.
 - Esquema da base de dados.

- Descrição das rotas da API.
- Descrição dos algoritmos de recomendação.
- Instruções de configuração e instalação.

Planeamento e Escrita do Relatório Final (2ª Semana):

- Planear a estrutura do relatório final.
- **Documentar o Trabalho Realizado:**

1. Descrição do Projeto:

- Fornecer uma visão geral clara do projeto, seus objetivos e funcionalidades.
- **Metodologia:**
 - Detalhar as abordagens e técnicas utilizadas no desenvolvimento da aplicação, incluindo as tecnologias (React Native, Node.js, PostgreSQL, Scikit-learn, Firebase) e as metodologias de desenvolvimento (e.g., desenvolvimento ágil).
- **Implementação:**
 - Descrever as etapas de desenvolvimento, desde a configuração do ambiente até a implementação das funcionalidades, incluindo os desafios e soluções encontradas.
- **Arquitetura:**
 - Descrever a arquitetura do projeto, tanto do backend como do frontend.

2. Apresentar os Resultados Alcançados:

- **Funcionalidades Implementadas:**
 - Listar e descrever as funcionalidades implementadas na aplicação, demonstrando o progresso alcançado.
- **Resultados dos Testes:**
 - Apresentar os resultados dos testes realizados, demonstrando a qualidade e o desempenho da aplicação.
- **Avaliação do Sistema de Recomendação:**

- Apresentar os resultados da avaliação do sistema de recomendação, demonstrando a precisão e a eficácia do modelo.
- **Relatório de erros:**
 - Apresentar um relatório dos erros encontrados durante o teste, e as soluções implementadas.

3. Partilhar as Lições Aprendidas:

- **Desafios e Soluções:**
 - Descrever os desafios encontrados durante o estágio e as soluções implementadas para superá-los.
- **Lições Técnicas:**
 - Partilhar as lições técnicas aprendidas durante o desenvolvimento da aplicação, incluindo boas práticas de programação e novas tecnologias aprendidas.
- **Lições Profissionais:**
 - Refletir sobre o desenvolvimento profissional durante o estágio, incluindo habilidades de comunicação, trabalho em equipa e gestão de tempo.
- **Melhorias futuras:**
 - Apresentar melhorias que podem ser feitas no projeto, no futuro.

4. Cumprir Requisitos Académicos ou Profissionais:

- **Requisitos da Instituição:**
 - Cumprir os requisitos do programa de estágio ou da instituição de ensino, demonstrando o cumprimento dos objetivos do estágio.
- **Documentação para Futuro:**
 - Servir como documentação para futuros desenvolvedores que venham a trabalhar no projeto.

9. Anexo B

GitHub – Controle de versões e boas praticas

GitHub

Controle de versões

Realizado por Davi Fernandes Souza

Index Terms—Git, Github, Controle de versões

I. INTRODUÇÃO

O controle de versões é uma prática essencial no desenvolvimento de software, permitindo que os desenvolvedores rastreiem mudanças nos arquivos ao longo do tempo. O GitHub é uma plataforma popular que facilita o controle de versões e a colaboração entre desenvolvedores.

II. O QUE É GITHUB?

O GitHub é uma plataforma de hospedagem de código-fonte e controle de versão distribuído. Ele permite que os desenvolvedores armazenem, gerenciem e colaborem em projetos de software. O GitHub oferece recursos como repositórios, branches, pull requests e issues, facilitando o trabalho em equipe e a organização do código [1].

III. COMO MANTER CONTROLE DE VERSÕES

Para manter o controle de versões no GitHub devemos:

- 1) **Clonar Repositório:** Para iniciar, clone o repositório para sua máquina local:

```
git clone https://github.com/user/repositorio.git
cd seu-repositorio
```

- 2) **Criar um Branch:** Crie um branch para desenvolver novas funcionalidades ou corrigir bugs. Isso ajuda a manter o branch principal (main) estável e funcional:

```
git checkout -b nova-funcionalidade
```

- 3) **Adicionar Mudanças:** Faça alterações no código, adicione os arquivos modificados ao controle de versão e crie um commit com uma mensagem clara:

```
git add .
git commit -m "Descrição clara das mudanças"
```

- 4) **Enviar o Branch para o GitHub:** Envie o seu branch para o repositório remoto no GitHub:

```
git push origin nova-funcionalidade
```

- 5) **Abrir um Pull Request:** No GitHub, navegue até o repositório e abra um Pull Request (PR) para o branch main. Isso permite que outros membros da equipe revisem e aprovem suas mudanças antes de integrá-las ao branch principal.

- 6) **Revisar e Aprovar Mudanças:** Outros membros da equipe devem revisar o Pull Request. Eles podem deixar comentários e solicitar alterações. Uma vez que o PR esteja aprovado, ele pode ser merged ao branch principal:

```
git checkout main
git merge nova-funcionalidade
```

- 7) **Resolver Conflitos:** Se houver conflitos durante o merge, resolva-os manualmente. Atualize o branch principal após resolver os conflitos:

```
git add .
git commit -m "Resolver conflitos de merge"
git push origin main
```

- 8) **Eliminar Branches:** Após o merge, elimine o branch desnecessário para manter o repositório organizado:

```
git branch -d nova-funcionalidade
git push origin --delete nova-funcionalidade
```

Seguindo essas etapas é possível manter um controle de versões e consistentemente no GitHub, garantindo um desenvolvimento organizado e colaborativo. Criar branches para dividir fases do trabalho e utilizar pull requests para revisar e aprovar mudanças antes de dar merge ao branch principal.

IV. BOAS PRÁTICAS EM CONTROLE DE VERSÕES

Seguir boas práticas de controle de versões é crucial para um desenvolvimento organizado e eficiente:

- ****Commit Regularmente**:** Faça commits frequentes com mensagens claras para documentar as mudanças feitas no código.

- ****Use Branches**:** Utilize branches para desenvolver novas funcionalidades ou corrigir bugs. O branch principal (main) deve sempre conter uma versão estável e funcional do código.

- ****Quando Criar Branches**:** Criar um branch para cada nova funcionalidade, correção de bugs ou tarefas importantes. Isso facilita a organização e a colaboração.

- ****Revisão de Código**:** Utilize pull requests para revisar e aprovar mudanças antes de integrá-las ao branch principal. Isso ajuda a garantir a qualidade e a integridade do código.

- ****Realizar Merges**:** Faça merges apenas quando a funcionalidade ou correção estiver completa, testada e revisada. Isso evita problemas e conflitos no branch principal.

- ****Documentação**:** Mantenha a documentação atualizada e clara, facilitando a compreensão e a manutenção do projeto.

- ****Colaboração**:** Comunique-se com outros membros da equipe e utilize as funcionalidades de colaboração do GitHub, como issues e pull requests.

V. CONCLUSÃO

O GitHub é uma ferramenta poderosa para o controle de versões e a colaboração em projetos de software. Seguindo as boas práticas e utilizando os comandos básicos do Git, é possível gerenciar os projetos de forma eficiente. A criação de branches para dividir fases do trabalho e o uso adequado de merges são fundamentais para manter o projeto organizado e estável.

VI. REFERÊNCIAS

REFERÊNCIAS

- [1] GitHub Documentation. [Online]. Available: <https://docs.github.com/>

10. Anexo C

React Native, Expo vs Native CLI

React Native (Expo vs CLI)

Realizado por Davi Fernandes Souza

Resumo—React Native: é um framework popular para construir aplicações móveis usando JavaScript e React. Ele permite que os desenvolvedores criem apps nativas para Android, iOS e outras plataformas com uma única base de código.

Expo: é um conjunto de ferramentas e um framework construído sobre o React Native que simplifica o desenvolvimento, teste e implantação de aplicações.

React Native CLI: Ferramenta de linha de comando que oferece uma abordagem manual e personalizada para o desenvolvimento de aplicações React Native, proporcionando controle total sobre o projeto.

Index Terms—React Native, Expo, Expo CLI, React Native CLI.

I. INTRODUÇÃO

ESTE relatório de pesquisa foi realizado pelo aluno da Escola Superior de Tecnologia e Gestão de Viseu, no âmbito do estágio curricular na empresa Sofinsa (IBM). A presente pesquisa faz parte dos estudos destinados ao desenvolvimento de um projeto, focando-se na utilização do React Native para o desenvolvimento de aplicações móveis. O objetivo principal desta pesquisa é comparar o uso do Expo e do React Native CLI para determinar qual abordagem é mais adequada para diferentes necessidades de desenvolvimento.

II. REACT NATIVE

React Native é um framework popular para construir aplicações móveis usando JavaScript e React. Ele permite que os desenvolvedores criem apps nativos para Android, iOS e outras plataformas com uma única base de código. O React Native oferece um conjunto de componentes nativos independentes da plataforma, como `View`, `Text` e `Image`, que mapeiam diretamente para os blocos de construção de UI nativos da plataforma [1].

O principal benefício do React Native é a capacidade de reutilizar a maior parte do código entre diferentes plataformas, reduzindo significativamente o tempo e o esforço de desenvolvimento. Além disso, o React Native permite uma atualização em tempo real, o que significa que os desenvolvedores podem ver as mudanças instantaneamente sem precisar recompilar a aplicação. Ele também possui uma grande comunidade de desenvolvedores e uma ampla gama de bibliotecas e plugins disponíveis, facilitando a adição de novas funcionalidades [2].

III. EXPO

Expo é um conjunto de ferramentas e um framework construído sobre o React Native que simplifica o desenvolvimento, teste e implantação de aplicações. Ele oferece um ambiente

pré-configurado, APIs e componentes prontos para uso, facilitando a criação de projetos sem a necessidade de configuração manual de ambientes de desenvolvimento nativos. Expo é ideal para prototipagem rápida e desenvolvimento ágil, permitindo aos desenvolvedores iniciar projetos rapidamente e continuar o desenvolvimento com alterações nativas [3].

IV. REACT NATIVE CLI

React Native CLI (Command Line Interface) é uma ferramenta de linha de comando que oferece uma abordagem mais manual e personalizada para o desenvolvimento de aplicações React Native. Ela permite controle total sobre o projeto, incluindo a capacidade de adicionar e modificar código nativo (Java/Kotlin para Android, Objective-C/Swift para iOS), sendo ideal para projetos que exigem customizações avançadas e integração com bibliotecas nativas específicas. A CLI é recomendada para desenvolvedores que precisam de maior flexibilidade e controle sobre o projeto [4].

V. COMPARAÇÃO

Nesta seção, comparamos as abordagens de desenvolvimento de aplicações móveis usando **Expo** e **React Native CLI**, destacando suas principais diferenças e características.

| Características | Expo |
|-------------------------|------------------------------|
| Configuração | Fácil e rápida |
| Controle | Limitado |
| Ferramentas Nativas | Menos personalizável |
| Componentes e APIs | Incluídos e prontos para uso |
| Adição de Código Nativo | Limitada |

Tabela I
COMPARAÇÃO EXPO

A. Expo

Expo é ideal para prototipagem rápida e desenvolvimento ágil, permitindo aos desenvolvedores iniciar projetos rapidamente e continuar o desenvolvimento com alterações nativas. Ele oferece um ambiente pré-configurado, APIs e componentes prontos para uso, facilitando a criação de projetos sem a necessidade de configuração manual de ambientes de desenvolvimento nativos.

B. React Native CLI

React Native CLI é recomendado para desenvolvedores que precisam de maior flexibilidade e controle sobre o projeto. Ele oferece uma abordagem mais manual e personalizada para

| Características | React Native CLI |
|-------------------------|---------------------------------------|
| Configuração | Manual e personalizada |
| Controle | Total |
| Ferramentas Nativas | Totalmente personalizável |
| Componentes e APIs | Depende das escolhas do desenvolvedor |
| Adição de Código Nativo | Completa flexibilidade |

Tabela II
COMPARAÇÃO REACT NATIVE CLI

o desenvolvimento de aplicações, incluindo a capacidade de adicionar e modificar código nativo (Java/Kotlin para Android, Objective-C/Swift para iOS). Esta abordagem é ideal para projetos que exigem customizações avançadas e integração com bibliotecas nativas específicas.

C. Resumo

Em resumo, Expo é ideal para quem deseja um ambiente de desenvolvimento simplificado e rápido, enquanto a React Native CLI é mais adequada para desenvolvedores que precisam de um controle total e personalização avançada.

VI. UTILIZAÇÃO

Nesta seção, vamos abordar como utilizar React Native, Expo e React Native CLI, com exemplos de comandos de instalação e uso no terminal para Node.js.

A. Instalação do Node.js

Antes de começar, é necessário ter o Node.js instalado em no sistema, pode instalar o Node.js a partir do site oficial: <https://nodejs.org/>

B. Instalação do React Native CLI

Para instalar o React Native CLI, execute o seguinte comando no terminal:

```
"" npm install -g react-native-cli ""
```

Após a instalação pode criar um novo projeto React Native com o comando:

```
"" react-native init MeuProjeto cd MeuProjeto ""
```

Para iniciar o projeto, utilize:

```
"" react-native run-android ou react-native run-ios ""
```

C. Instalação do Expo CLI

Para instalar o Expo CLI, execute no terminal:

```
"" npm install -g expo-cli ""
```

Após a instalação, criar um novo projeto Expo com o seguinte comando:

```
"" expo init MeuProjeto cd MeuProjeto ""
```

Iniciar o projeto, utilize:

```
"" expo start ""
```

D. Comandos Básicos

Aqui estão alguns comandos básicos para trabalhar com React Native, Expo e React Native CLI:

- ****Instalar Dependências****: Para instalar as dependências do projeto, execute: `"" npm install ""`

- ****Executar o Projeto****: - Expo: `"" expo start ""` - React Native CLI: `"" react-native run-android react-native run-ios ""`

- ****Adicionar Bibliotecas****: Para adicionar uma biblioteca ao projeto, utilize: `"" npm install nome-da-biblioteca ""`

- ****Atualizar Expo****: Para atualizar o Expo CLI, execute: `"" npm install -g expo-cli ""`

Esta seção fornece uma visão geral de como iniciar e gerenciar projetos usando React Native, Expo e React Native CLI, com exemplos práticos de comandos de terminal em Node.js.

VII. BOAS PRÁTICAS EM REACT NATIVE

Nesta seção, abordamos algumas das melhores práticas recomendadas para o desenvolvimento de aplicações móveis usando React Native. Seguir estas práticas pode ajudar a garantir que o código seja eficiente, sustentável e de fácil manutenção.

A. Estrutura do Projeto

Manter uma estrutura de projeto organizada é essencial para facilitar a navegação e a manutenção do código. Aqui estão algumas dicas:

Agrupe Arquivos por Função: Organize seus arquivos por funcionalidades ou componentes ao invés de tipos de arquivos. Isso ajuda a manter a lógica relacionada em um único lugar.

Use Componentes Funcionais: Sempre que possível, use componentes funcionais em vez de componentes de classe. Componentes funcionais são mais simples e facilitam o uso de hooks, como `useState` e `useEffect`.

Utilize Pastas para Componentes: Crie pastas separadas para componentes reutilizáveis, páginas, estilos e serviços.

B. Estilo e Layout

Manter um estilo consistente e utilizar boas práticas de layout é fundamental para garantir a legibilidade e a manutenção do código.

Use Estilos Consistentes: Utilize uma abordagem consistente para estilos, como o uso de `StyleSheet.create()` para definir estilos e manter a consistência em toda a aplicação.

Flexbox para Layout: Aproveite o Flexbox para criar layouts responsivos e adaptáveis. Isso facilita o ajuste do layout em diferentes tamanhos de tela.

Evite Inline Styles: Sempre que possível, evite definir estilos inline. Utilize arquivos de estilos separados para manter o código limpo e organizado.

C. Acessibilidade

Garantir que sua aplicação seja acessível para todos os usuários é uma prática importante.

Labels e Roles: Adicione labels e roles apropriados aos elementos interativos para garantir que leitores de tela possam identificar e descrever esses elementos.

Contraste de Cores: Certifique-se de que o contraste de cores seja suficiente para usuários com deficiências visuais.

Apoio a Navegação por Teclado: Garanta que todos os elementos interativos possam ser acessados e manipulados através da navegação por teclado.

VIII. CONCLUSÃO

Esta pesquisa explorou as abordagens de desenvolvimento de aplicações móveis usando **React Native**, com foco na comparação entre **Expo** e **React Native CLI**. Ambos têm suas próprias vantagens e desvantagens, dependendo das necessidades específicas do projeto.

Expo é uma excelente escolha para desenvolvedores que desejam um ambiente de desenvolvimento simplificado e rápido, com uma ampla gama de componentes e APIs prontos para uso. É ideal para prototipagem rápida e projetos que não requerem muitas customizações nativas.

Por outro lado, React Native CLI é mais adequado para desenvolvedores que precisam de controle total e flexibilidade sobre o projeto. Ele permite a adição e modificação de código nativo, tornando-se a melhor opção para projetos complexos que exigem integrações específicas e customizações avançadas.

Ao escolher entre Expo e React Native CLI, os desenvolvedores devem considerar fatores como a complexidade do projeto, a necessidade de customizações nativas e o nível de controle desejado. Ambas as abordagens oferecem poderosas ferramentas para o desenvolvimento de aplicações móveis, e a escolha entre elas dependerá das necessidades individuais de cada projeto.

REFERÊNCIAS

- [1] React Native. Facebook. [Online]. Available: <https://reactnative.dev/>
- [2] React Native Community. [Online]. Available: <https://github.com/react-native-community>
- [3] Expo Documentation. [Online]. Available: <https://docs.expo.dev/>
- [4] React Native CLI Documentation. [Online]. Available: <https://reactnative.dev/docs/environment-setup>

11. Anexo D

Testes iniciais as rotas do *backend*

Testes de Validação

ChooseeWine4me-Backend

Desenvolvido por:

Davi Fernandes Silva Souza

Em contexto de Estágio Curricular

Entidades Acolhedoras:

Softinsa (IBM)

Escola Superior de Tecnologias e Gestão de Viseu

Março, 2025

Índice de Figuras

| | |
|---|----|
| Figura 1- Post User - criado com sucesso | 4 |
| Figura 2- MongoDB - Coleção atualizada | 4 |
| Figura 3 - Post User - Email já registrado. Bad request..... | 5 |
| Figura 4- Put User - Editar Nome. | 5 |
| Figura 5- MongoDB - Nome alterado de Davi para Davi Souza. | 6 |
| Figura 6- MongoDB - user exemplo adicionado para teste da rota delete..... | 6 |
| Figura 7- Delete User – Deletar user por id_user. | 6 |
| Figura 8- MongoDB - Confirmação que foi deletado da base de dados..... | 7 |
| Figura 9- Get Users - Lista de users existente, neste caso so existe um..... | 7 |
| Figura 10- Post Addresses - Address adicinado com sucesso. | 8 |
| Figura 11- MongoDB - Confirmação dos dados inseridos na base de dados. | 8 |
| Figura 12- MongoDB - Morada duplicada para teste da rota Dlete..... | 9 |
| Figura 13- Delete Addresses - Deletar address por id_address. | 9 |
| Figura 14- MongoDB - Morada duplicada eliminada com sucesso. | 9 |
| Figura 15- Put Addresses - Alterar a morada. | 10 |
| Figura 16- MongoDB - Morada alterada com sucesso | 10 |
| Figura 17- Get Adresses - Lista de moradas existentes, só existe uma neste caso | 11 |
| Figura 18- Post Wines - Vinho criado com sucesso..... | 11 |

Introdução

Este documento tem como objetivo apresentar os testes de validação realizados sobre o backend da aplicação *ChooseWINE4me*, desenvolvida no âmbito do estágio curricular na Softinsa, em parceria com a Escola Superior de Tecnologia e Gestão de Viseu.

O backend foi desenvolvido com **Node.js e Express**, e comunica com uma base de dados **MongoDB Atlas**. Está organizado segundo um modelo modular, com rotas REST responsáveis por gerir as entidades principais da aplicação: utilizadores, moradas, vinhos e favoritos. Os testes aqui descritos visam verificar o correto funcionamento das rotas implementadas, a integridade dos dados inseridos na base de dados e o tratamento adequado de erros e exceções.

Para cada entidade, foram realizados testes aos métodos HTTP mais relevantes (*POST, GET, PUT, DELETE*), com verificação dos efeitos no MongoDB e análise das respostas da API. As figuras incluídas neste documento apresentam os pedidos realizados e os respetivos resultados, acompanhados de registos visuais do estado da base de dados após cada operação.

Estes testes foram fundamentais para assegurar que o backend cumpre os requisitos definidos, garantindo fiabilidade e consistência na gestão dos dados da aplicação.

Testes

```
\ChooseWINE4me\choosewine-backend> node src/server.js
Servidor rodando na porta 3000
Conectado ao MongoDB com sucesso!
```

User

POST

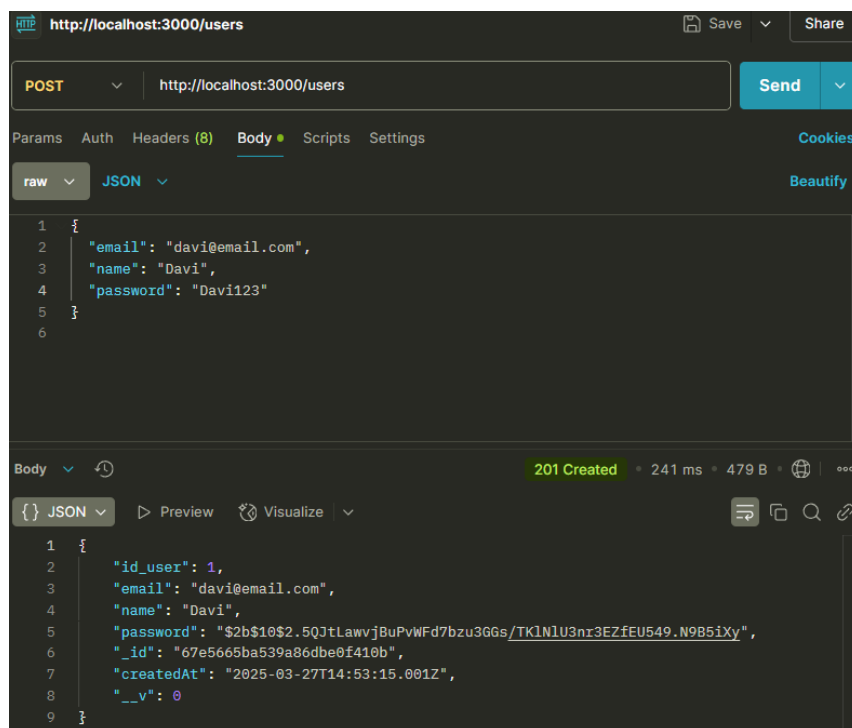


Figura 1- Post User - criado com sucesso

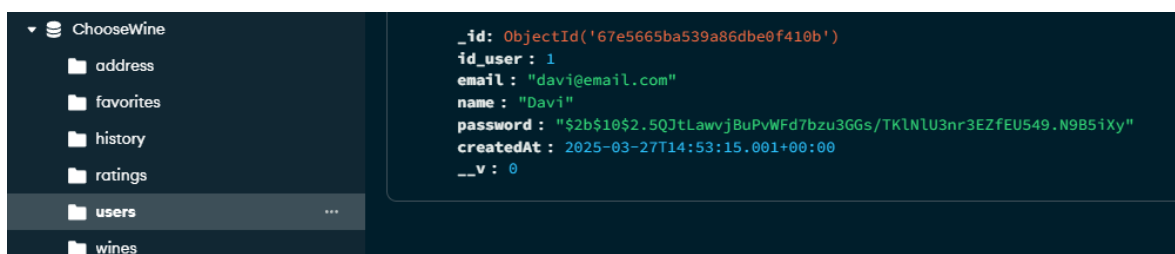


Figura 2- MongoDB - Coleção atualizada

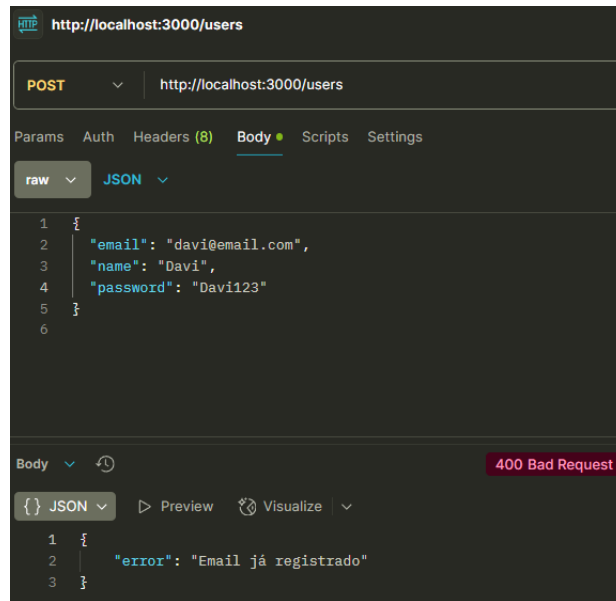


Figura 3 - Post User - Email já resgistrado. Bad request.

PUT

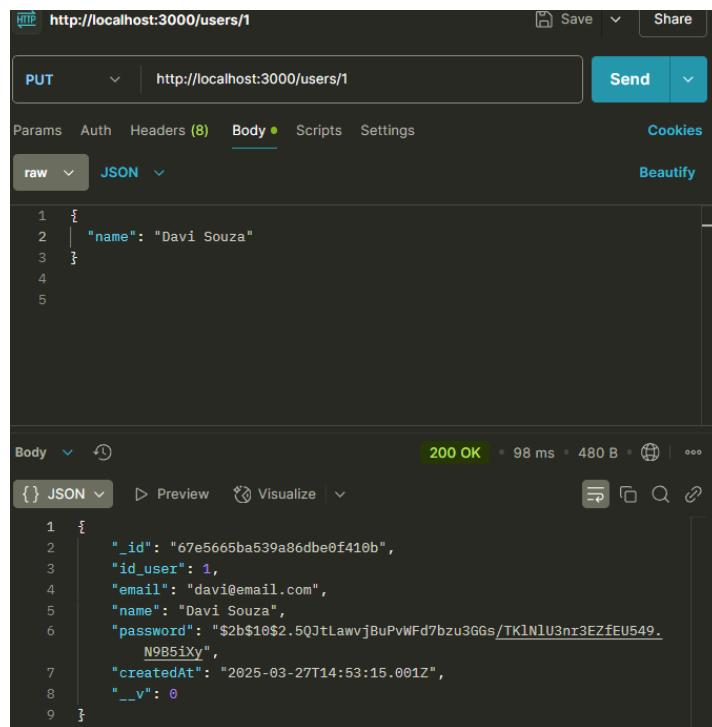


Figura 4- Put User - Editar Nome.

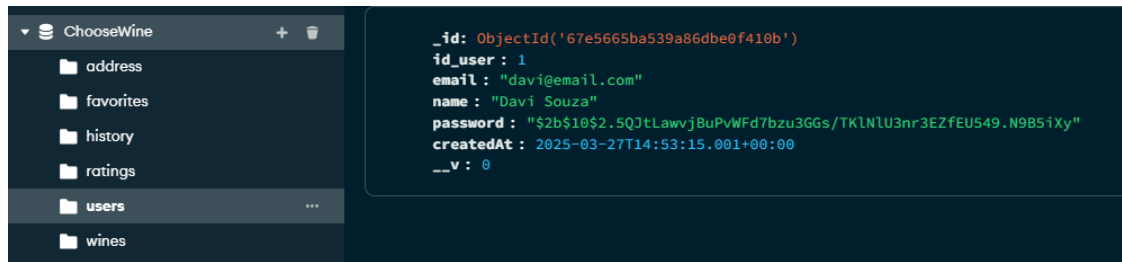


Figura 5- MongoDB - Nome alterado de Davi para Davi Souza.

DELETE



Figura 6- MongoDB - user exemplo adicionado para teste da rota delete.

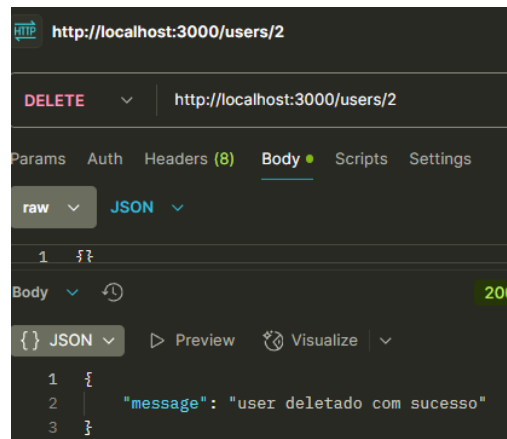


Figura 7- Delete User – Deletar user por id_user.



Figura 8- MongoDB - Confirmação que foi deletado da base de dados.

GET

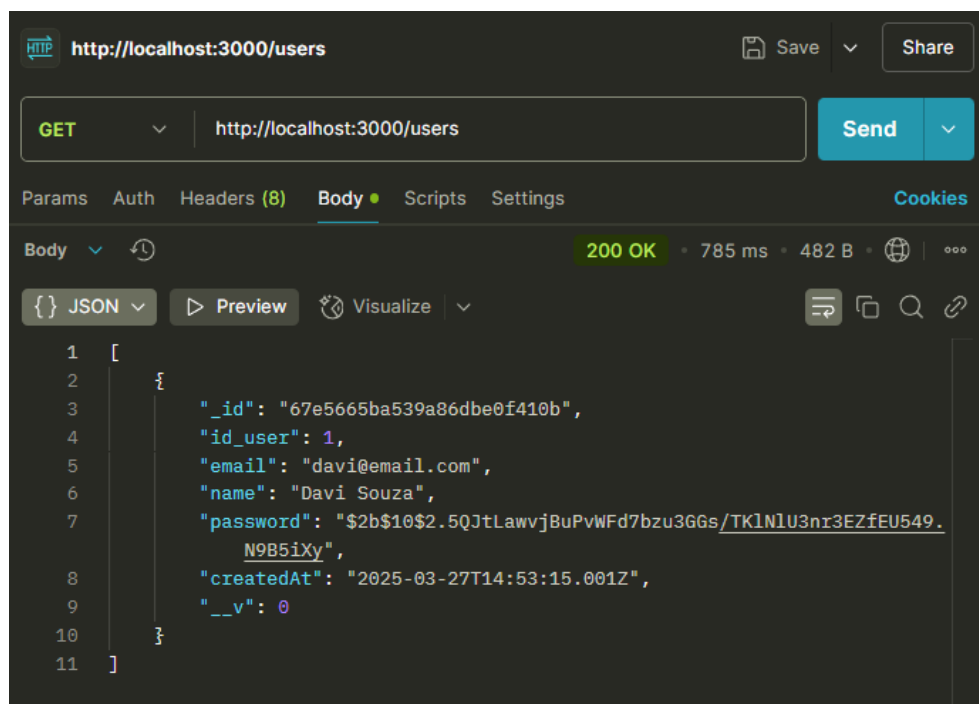


Figura 9- Get Users - Lista de users existente, neste caso so existe um.

Addresses

POST

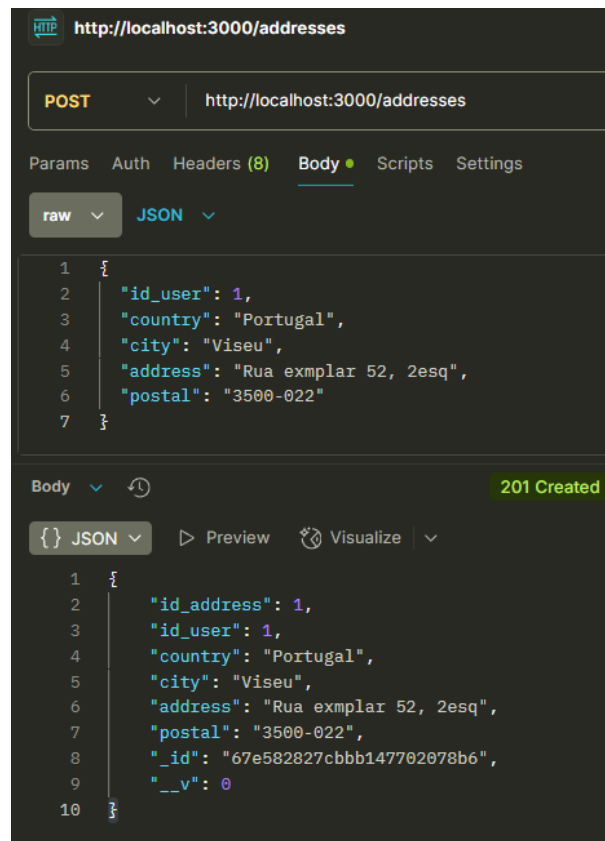


Figura 10- Post Addresses - Address adicionado com sucesso.

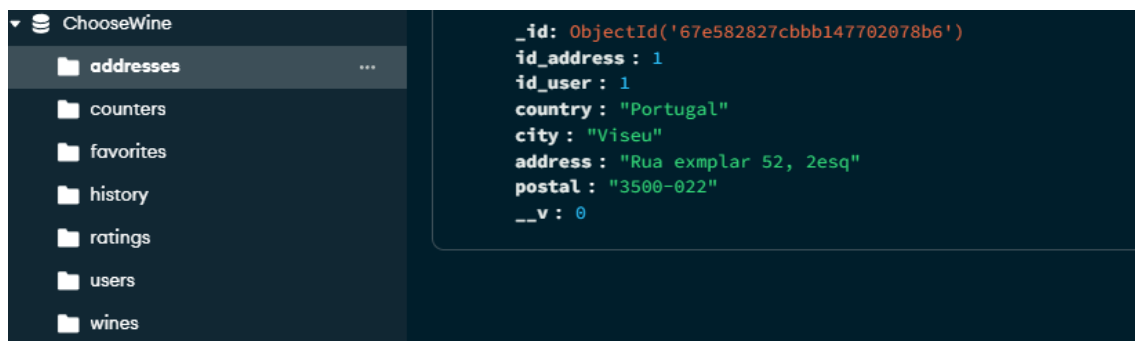


Figura 11- MongoDB - Confirmação dos dados inseridos na base de dados.

DELETE

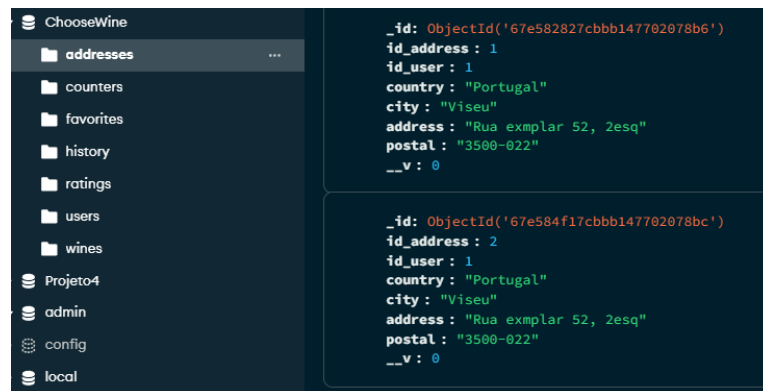


Figura 12- MongoDB - Morada duplicada para teste da rota Delete.

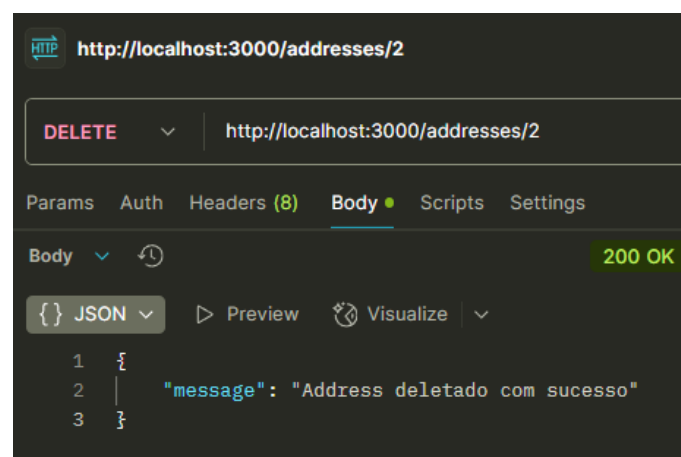


Figura 13- Delete Addresses - Deletar address por id_address.

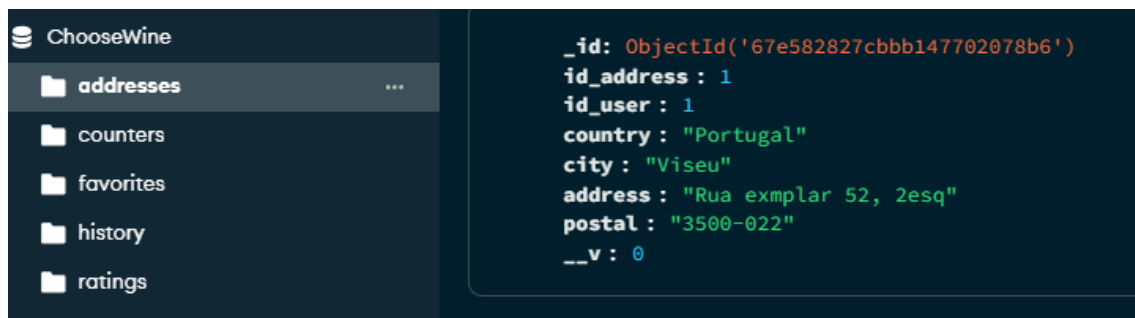


Figura 14- MongoDB - Morada duplicada eliminada com sucesso.

PUT

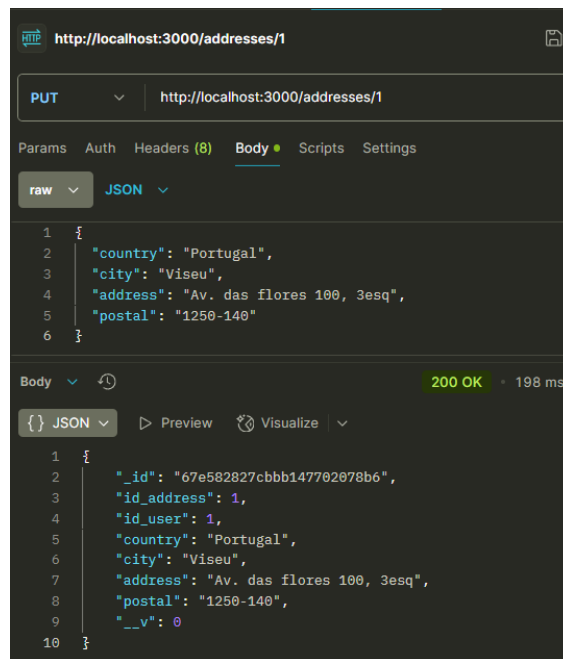


Figura 15- Put Addresses - Alterar a morada.



Figura 16- MongoDB - Morada alterada com sucesso

GET

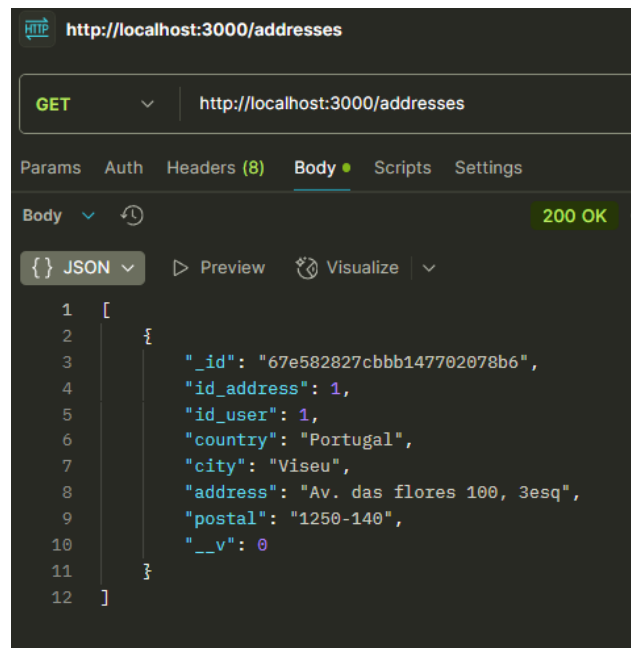


Figura 17- Get Adresses - Lista de moradas existentes, só existe uma neste caso

Wines

POST

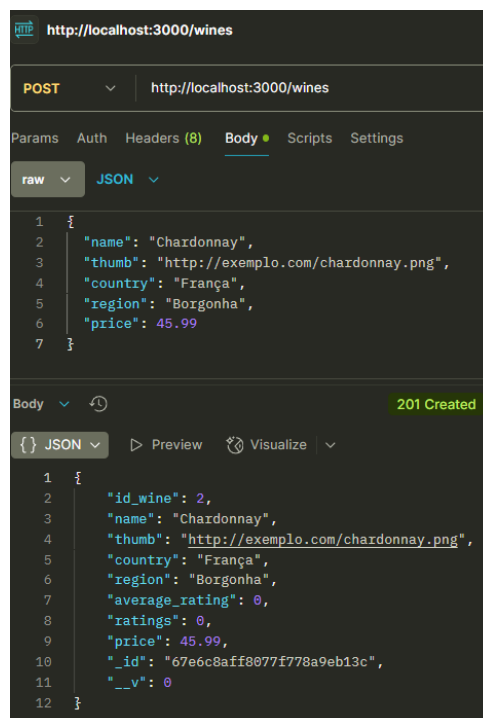
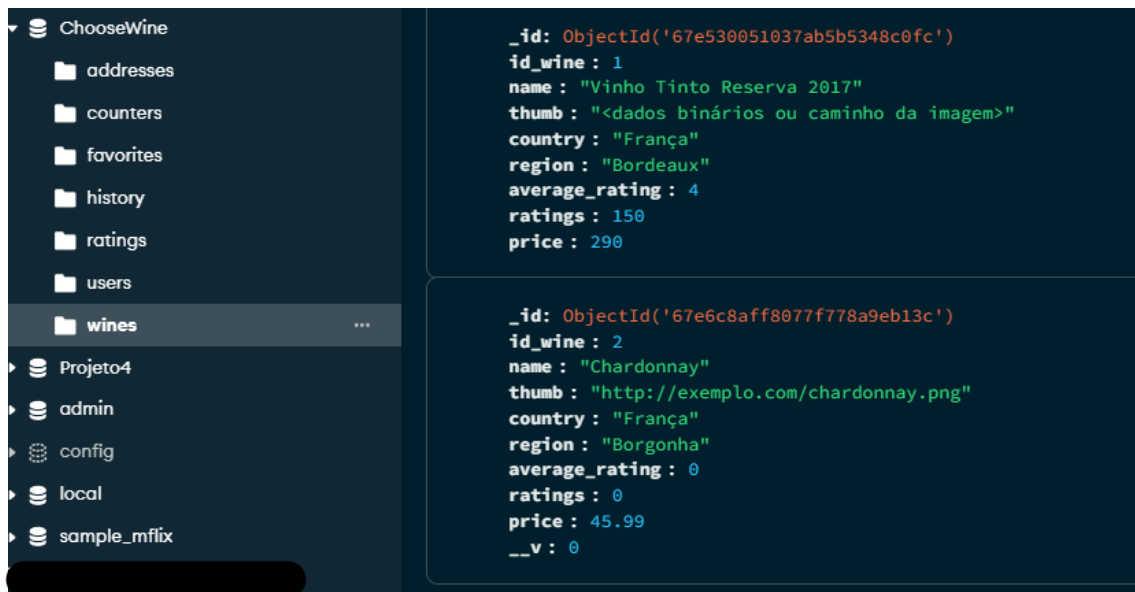
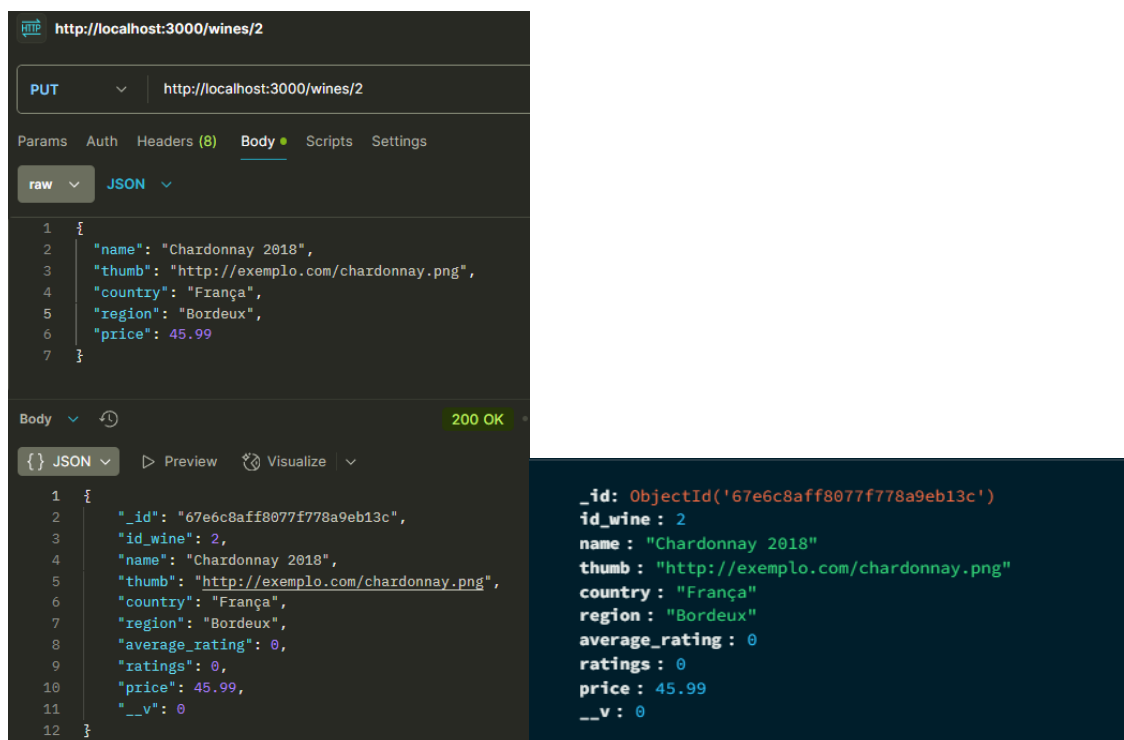


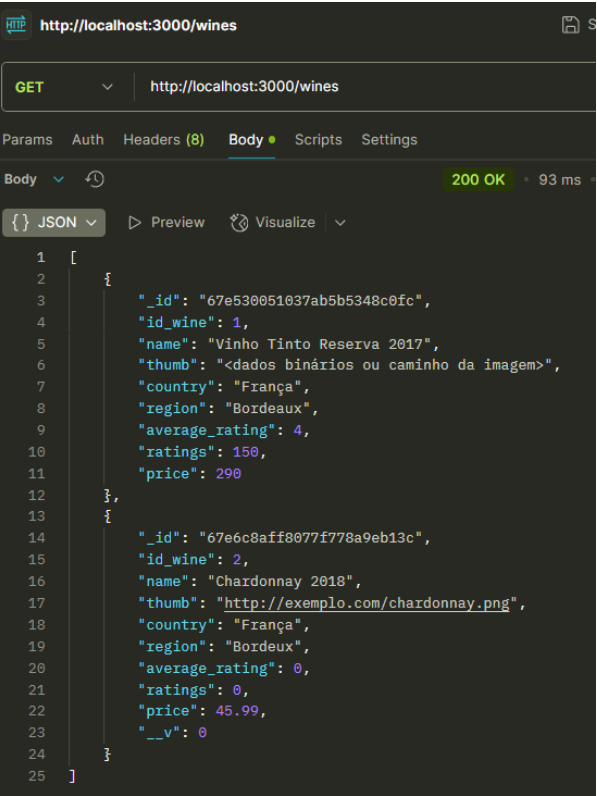
Figura 18- Post Wines - Vinho criado com sucesso.



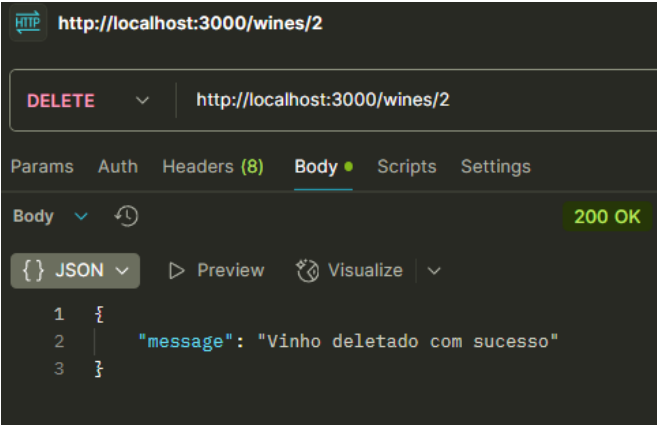
PUT



GET

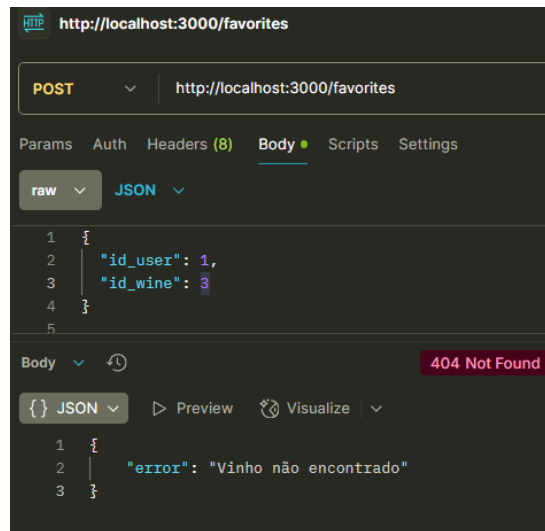


DELETE



Favoritos

POST



HTTP <http://localhost:3000/favorites>

POST <http://localhost:3000/favorites>

Params Auth Headers (8) **Body** Scripts Settings

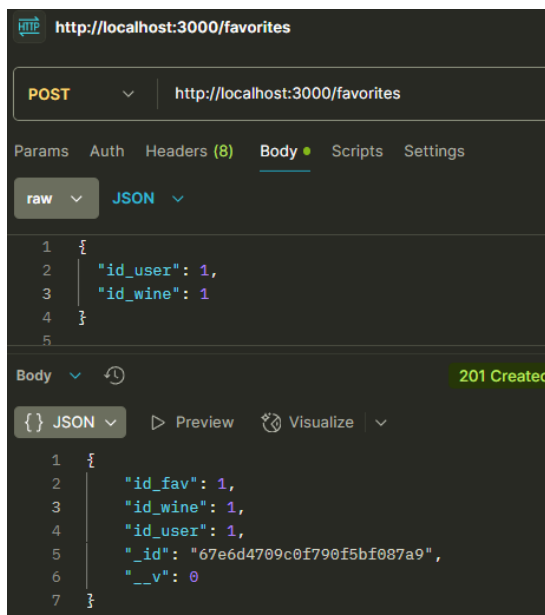
raw JSON

```
1 {
2   "id_user": 1,
3   "id_wine": 3
4 }
5
```

Body [View](#) [Refresh](#) **404 Not Found**

{ } JSON [Preview](#) [Visualize](#)

```
1 {
2   "error": "Vinho não encontrado"
3 }
```



HTTP <http://localhost:3000/favorites>

POST <http://localhost:3000/favorites>

Params Auth Headers (8) **Body** Scripts Settings

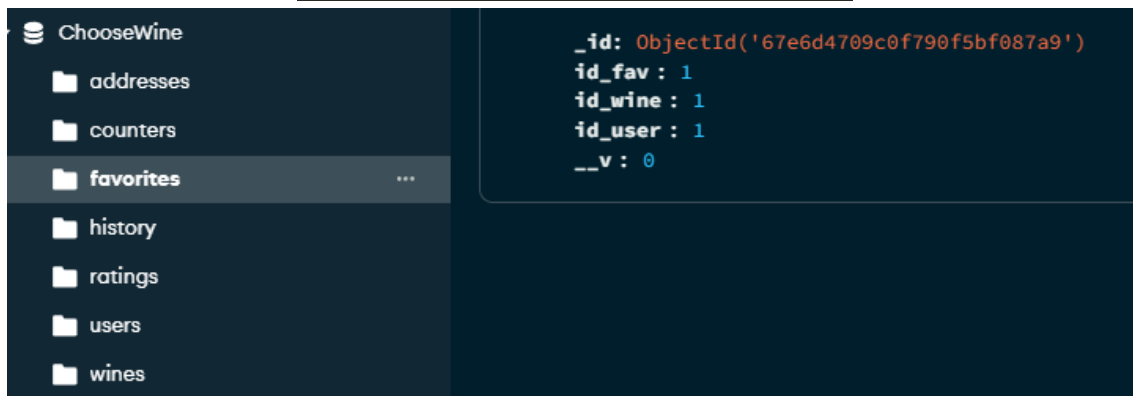
raw JSON

```
1 {
2   "id_user": 1,
3   "id_wine": 1
4 }
5
```

Body [View](#) [Refresh](#) **201 Created**

{ } JSON [Preview](#) [Visualize](#)

```
1 {
2   "id_fav": 1,
3   "id_wine": 1,
4   "id_user": 1,
5   "_id": "67e6d4709c0f790f5bf087a9",
6   "__v": 0
7 }
```



ChooseWine

- addresses
- counters
- favorites**
- history
- ratings
- users
- wines

```
_id: ObjectId('67e6d4709c0f790f5bf087a9')
id_fav : 1
id_wine : 1
id_user : 1
__v : 0
```

PUT

HTTP <http://localhost:3000/favorites/1>

PUT <http://localhost:3000/favorites/1>

Params Auth Headers (8) **Body** Scripts Settings

raw JSON

```
1 {
2   "id_user": 1,
3   "id_wine": 2
4 }
5
```

Body 200 OK - 97 ms

{ } JSON Preview Visualize

```
1 {
2   "_id": "67e6d4709c0f790f5bf087a9",
3   "id_fav": 1,
4   "id_wine": 2,
5   "id_user": 1,
6   "__v": 0
7 }
```

```
_id: ObjectId('67e6d4709c0f790f5bf087a9')
id_fav : 1
id_wine : 2
id_user : 1
__v : 0
```

GET

HTTP <http://localhost:3000/favorites>

GET <http://localhost:3000/favorites>

Params Auth Headers (8) **Body** Scripts Settings

Body 200 OK

{ } JSON Preview Visualize

```
1 [
2   {
3     "_id": "67e6d4709c0f790f5bf087a9",
4     "id_fav": 1,
5     "id_wine": 2,
6     "id_user": 1,
7     "__v": 0
8   }
9 ]
```

DELETE

