

Universidade do Minho  
Escola de Engenharia  
Licenciatura em Engenharia Informática

## Arquiteturas Aplicacionais Sistemas Interativos Confiáveis

**TastyCheck**  
Portal de Restaurantes Recomendados  
Ano Letivo de 2024/2025

### Grupo 3:

**Bruno Alves** pg55924  
**Filipe Gonçalves** pg55940  
**Lucas Oliveira** pg57886  
**Luna Figueiredo** pg55979  
**Rafael Alves** pg55999

# Índice

<b>1. Introdução</b>	<b>1</b>
1.1. Motivação	1
1.2. Contextualização	1
<b>2. Modelação</b>	<b>2</b>
2.1. Requisitos	2
2.1.1. Requisitos Funcionais	2
2.1.2. Requisitos Não Funcionais	3
2.2. Diagrama de <i>Use Cases</i>	4
2.3. Modelo de Domínio	5
2.4. Diagrama de Classes - <i>PIM</i>	5
2.5. Protótipos de Interface	6
2.5.1. Página Inicial	7
2.5.2. Página de <i>Login</i>	7
2.5.3. Página de Restaurante	8
2.5.4. Avaliação de Restaurante	8
2.5.5. Perfil do Cliente	9
2.5.6. Perfil do Proprietário	9
<b>3. Implementação</b>	<b>10</b>
3.1. Arquitetura do Sistema	10
3.2. Camada de Apresentação	12
3.2.1. <i>Interfaces</i>	12
3.2.1.1. <b>Página inicial</b>	12
3.2.1.2. <b>Login</b>	13
3.2.1.3. Página de Registo	14
3.2.1.4. Perfil do Utilizador	15
3.2.1.5. Página de Restaurante	16
3.2.1.6. <i>Popup</i> de avaliação de um restaurante	18
3.3. Camada da Lógica de Negócio	19
3.3.1. Diagrama de Classes PSM - <i>Models</i>	19
3.3.2. Diagrama de Classes PSM - <i>DTO's</i>	20
3.3.3. Diagrama de Classes PSM - <i>Services</i>	23
3.3.4. Diagrama de Classes PSM - <i>DAO's</i>	23
3.3.5. Diagrama de Classes PSM - <i>Controllers</i>	24
3.4. Camada de Dados	24
<b>4. Deployment</b>	<b>27</b>
<b>5. Análise do sistema</b>	<b>28</b>
5.1. Performance e Escalabilidade	28
5.1.1. Teste de carga	28
5.1.2. Teste de stress	29
5.1.3. Teste de resistência	29
5.1.4. Teste funcional com carga	30
5.1.5. Conclusão da análise de desempenho	31

<b>6. Segurança da Aplicação</b>	<b>32</b>
<b>7. Conclusão e Trabalho Futuro</b>	<b>33</b>

# **1. Introdução**

Este relatório foi elaborado no âmbito das unidades curriculares de Arquiteturas Aplicacionais e Sistemas Interativos Confiáveis e tem como objetivo descrever o processo de conceção, implementação, análise de uma aplicação web.

O projeto foca-se no desenvolvimento de uma plataforma interativa de avaliação e recomendação de restaurantes, onde os utilizadores podem partilhar experiências gastronómicas, pesquisar estabelecimentos, visualizar avaliações e contribuir com comentários e/ou conteúdos multimédia.

## **1.1. Motivação**

Com o aumento da importância das experiências dos consumidores nas decisões diárias e na reputação digital de negócios, aplicações de avaliação tornaram-se ferramentas essenciais no dia a dia. Contudo, muitas vezes não temos toda a informação que precisamos para fazer a melhor escolha, pois podemos deparar-nos com a falta de suporte a conteúdos multimédia, mecanismos de pesquisa poucos flexíveis ou interfaces confusas, por exemplo.

Assim, decidimos concentrar-nos no campo da restauração para criar uma solução moderna, acessível e funcional que consiga satisfazer todas as necessidades reais que um utilizador possa ter, a partir de uma plataforma simples com funcionalidades bem pensadas, o *Tastycheck*.

Com isto, queremos não só facilitar o processo de descoberta e avaliação de restaurantes, como também incentivar a comunicação entre proprietários e clientes.

## **1.2. Contextualização**

O *TastyCheck* foi criado para ser acessível em qualquer dispositivo, estando assim disponível para todos. Ela disponibiliza funcionalidades para distintos tipos de utilizadores: anónimos, clientes registados e proprietários de restaurantes, como:

1. Criação e gestão de páginas de restaurantes - proprietários de restaurantes;
2. Avaliação e comentários de restaurantes - clientes registados e proprietários de restaurantes;
3. Pesquisa e filtragem de restaurantes com base em múltiplos critérios - todos os utilizadores;
4. Visualização de restaurantes num mapa - todos os utilizadores.

O sistema foi pensado para garantir escalabilidade, resiliência, segurança e uma boa experiência de utilização, proporcionando um ambiente confiável e intuitivo para todos os tipos de utilizadores.

## 2. Modelação

Nesta secção, é apresentada a modelação que orientou o desenvolvimento do sistema. Primeiramente, serão listados os requisitos funcionais e não funcionais levantados, que são seguidos pelos modelos formais elaborados - diagrama de *Use Cases*, modelo de domínio e diagrama de classes *PIM* - que definem de forma mais clara os comportamentos esperados da aplicação e das ligações entre componentes. Por fim, para haver uma representação mais visual da aplicação apresentamos os protótipos das páginas mais relevantes para a demonstração da aplicação.

### 2.1. Requisitos

O processo de levantamento de requisitos é uma parte essencial para o desenvolvimento de software, porque é a partir deles que definimos as funcionalidades do sistema e o comportamento esperado dele, em termos de desempenho, usabilidade, segurança, entre outros.

Nas seguintes secções estão listados os requisitos funcionais e não funcionais levantados pelo grupo. Estes foram definidos através de introspecção e da análise de plataformas com objetivos parecidos ao nosso, como o *Google Reviews*, o *TripAdvisor* e o *The Fork*.

#### 2.1.1. Requisitos Funcionais

##### Gestão de Restaurantes por Proprietários

- O sistema deve permitir que os proprietários criem, editem e eliminem as páginas dos seus restaurantes, através de uma interface dedicada e autenticada.
- Cada página de restaurante deve incluir pelo menos o nome.

##### Avaliações por Clientes

- Os clientes devem poder avaliar restaurantes atribuindo uma pontuação de 1 a 5 estrelas.
- Os clientes devem poder complementar a avaliação com comentários textuais.
- Deve ser possível anexar conteúdos multimédia às avaliações, nomeadamente fotografias e vídeos.

##### Pesquisa e Exploração de Restaurantes

- Os utilizadores (anónimos ou registados) devem poder pesquisar restaurantes por nome, localização, tipo de cozinha ou avaliação média.
- O sistema deve permitir a aplicação de filtros adicionais, como a distância a partir de um ponto geográfico.
- A localização dos restaurantes deve ser apresentada num mapa interativo.

##### Interação entre Utilizadores e Proprietários

- Os proprietários devem poder responder às avaliações deixadas pelos clientes.

##### Visualização de Conteúdos

- Os utilizadores devem poder visualizar as avaliações feitas por outros, incluindo textos, imagens e vídeos associados.
- As avaliações devem estar visíveis na página do restaurante respetivo e ser ordenáveis por data.

##### Funcionalidades Adicionais para Clientes

- Os clientes devem poder adicionar restaurantes à sua lista de favoritos para facilitar o acesso rápido no futuro.

- Caso encontrem um restaurante que ainda não esteja completo na plataforma, os clientes devem poder sugerir ou adicionar informação como horários, localização, tipo de cozinha, etc.

## 2.1.2. Requisitos Não Funcionais

### Performance e Escalabilidade

- O sistema deve garantir um tempo de resposta inferior a 5 segundos para todas as operações críticas.
- A plataforma deve suportar pelo menos 1000 utilizadores simultâneos sem degradação de desempenho.
- A infraestrutura deve escalar automaticamente de acordo com a carga.

### Usabilidade

- A interface deve ser intuitiva, permitindo que 90% dos utilizadores consigam executar tarefas básicas sem ajuda.
- Tarefas principais (como registar-se, fazer uma avaliação ou procurar um restaurante) devem poder ser concluídas em menos de 10 minutos.

### Segurança

- As palavras-passe devem ser armazenadas com recurso a criptografia forte.

### Legais

- O sistema deve cumprir o Regulamento Geral sobre a Proteção de Dados (RGPD), garantindo a privacidade dos dados do utilizador.
- O sistema deverá cumprir as leis relativas à propriedade intelectual, assegurando que todos os conteúdos e componentes de software utilizados se encontram devidamente licenciados e autorizados para uso.
- A aplicação deverá obedecer às normas legais aplicáveis à retenção e eliminação de dados, garantindo que as informações dos utilizadores são armazenadas apenas durante o período estritamente necessário para os fins a que se destinam.

### Aparência e Consistência

- A interface deve fornecer *feedback* claro nas ações do utilizador (ex: erros, sucesso, loading).
- Deve seguir uma linha visual consistente (cores, fontes, espaçamentos).

### Compatibilidade

- O sistema deve funcionar nos principais browsers modernos (Chrome, Firefox, Safari, Edge).
- Deve ser totalmente funcional tanto em dispositivos móveis como em desktops.

### Disponibilidade e Fiabilidade

- O sistema deve recuperar de falhas em menos de 15 minutos.
- O tempo médio entre falhas (MTBF) deve ser superior a 30 dias.

### Manutenção

- O sistema deve ter uma estrutura modular e bem documentada, facilitando a sua manutenção, testes e evolução futura.
- Deve existir documentação técnica detalhada para desenvolvimento e uso.

### Culturais e Políticos

- O conteúdo textual da interface gráfica deve estar todo escrito em Português.
- O sistema deverá estar alinhado com as normas culturais dos contextos em que é utilizado, evitando o uso de símbolos, cores ou expressões que possam ser considerados ofensivos ou inadequados em determinadas culturas.

## 2.2. Diagrama de Use Cases

O seguinte diagrama - Figura 1 - representa as interações possíveis entre os diferentes tipos de utilizadores e o sistema, conforme o que está descrito nos requisitos funcionais.

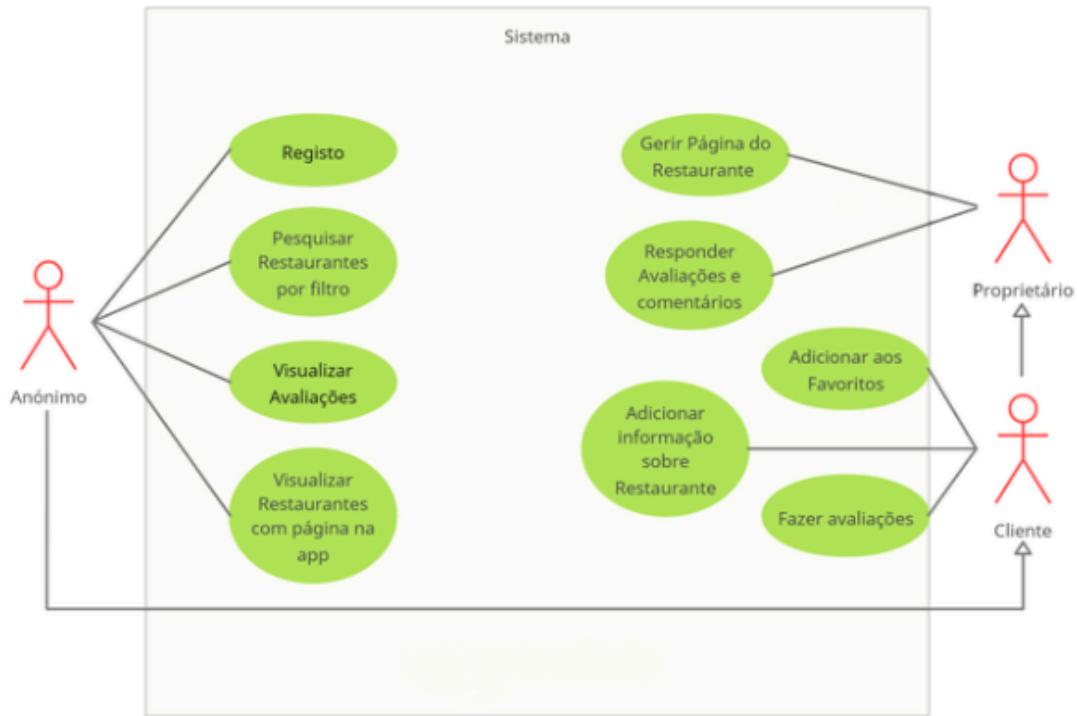


Figura 1: Diagrama de Use Cases

O sistema *TastyCheck* é composto por três tipos de utilizadores distintos: Anónimo, Cliente e Proprietário. As funcionalidades que cada um tem acesso estão de acordo com o seu perfil na aplicação.

**Utilizador Anónimo** Qualquer pessoa que accede à aplicação sem ter um sessão iniciada. Este pode:

- Efetuar registo, tornando-se um utilizador com conta ativa (Cliente);
- Pesquisar restaurantes, através dos filtros disponibilizados, como localização, tipo de cozinha ou avaliação;
- Visualização de avaliações deixadas por utilizadores autenticados.
- Visualizar páginas de restaurantes, podendo ver a informação disponibilizada.

**Clientes** (Utilizadores com conta) Depois do registo e autenticação na aplicação, o utilizador passa ter um perfil, com acesso a mais funcionalidades como:

- Fazer avaliações a restaurantes, podendo atribuir estrelas, comentários e conteúdos multimédia;
- Adicionar restaurantes aos favoritos, criando uma lista personalizada para acesso rápido;
- Adicionar informação sobre um restaurante, complementando os dados disponíveis na aplicação sobre ele, com por exemplo a morada, tipo de cozinha ou horário.

**Proprietário** É um utilizador autenticado com permissões extra para gerir a página do seu restaurante na plataforma, podendo:

- Gerir a página do restaurante, como editar as suas informações;
- Responder a avaliações e comentários deixados por clientes.

Assim, a partir deste diagrama consegue-se ter uma visão geral das principais funcionalidades e interações previstas para o sistema. Este também facilita no desenvolvimento posterior dos fluxos de interface e lógica de negócio.

## 2.3. Modelo de Domínio

Atendendo aos requisitos levantados e use cases expostos nas secções anteriores, elaboramos o seguinte modelo de domínio - Figura 2 - que representa, de forma abstrata, os principais conceitos envolvidos na nossa plataforma, bem como a relação entre eles.

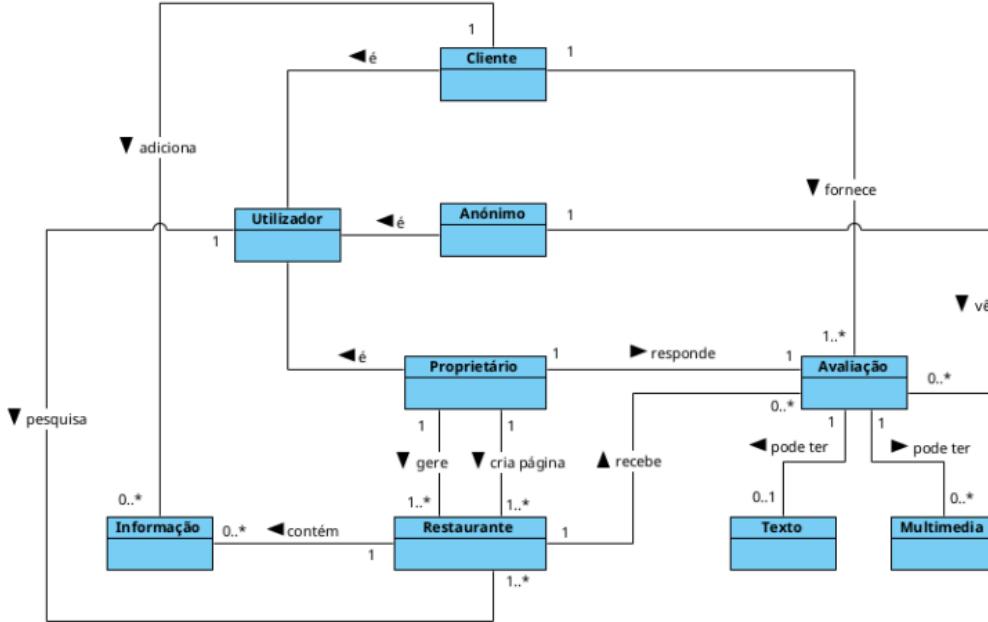


Figura 2: Modelo de Domínio

A entidade **Utilizador** representa qualquer pessoa que aceda à aplicação. Um utilizador com perfil é um Cliente ou Proprietário, tendo cada um acesso a algumas funcionalidades diferentes. Os dados associados a esta entidade incluem nome, email, password, avatar e tipo de perfil.

Para representar os restaurantes é usada a entidade **Restaurante**. Cada restaurante está associado a um proprietário, que gera a página do restaurante podendo editá-la ou responder às avaliações recebidas. Estas páginas podem conter informação detalha sobre o restaurante, como nome, morada, tipo de cozinha, horário, entre outros.

Apenas os Clientes e Proprietários podem deixar **Avaliações**, que ficam associadas a um restaurante. Cada avaliação consiste numa avaliação de 1 a 5 estrelas e/ou um comentário.

A estas avaliações apenas o Proprietário do restaurante pode responder, utilizando a entidade **Comentário**.

Uma outra funcionalidade para os clientes registados é poder guardar uma lista de restaurantes favoritos, representada pela entidade **Favorito**, que estabelece uma relação direta entre o cliente e os restaurantes que pretende manter em acesso rápido.

No Figura 2 também estão representadas algumas relações entre estas entidades como:

- um utilizador do tipo Proprietário pode ter associado várias contas de restaurantes;
- um restaurante pode ter múltiplas avaliações associadas;
- qualquer restaurante pode fazer parte da lista de favoritos de vários clientes diferentes.

## 2.4. Diagrama de Classes - PIM

No seguimento da definição do modelo de domínio, foi elaborado o Diagrama de Classes *PIM* - Figura 3 - que representa de forma mais detalhada a estrutura lógica do sistema desenvolvido, independente de tecnologia. Aqui, são identificadas as principais classes da aplicação, os seus atributos e relações entre elas.

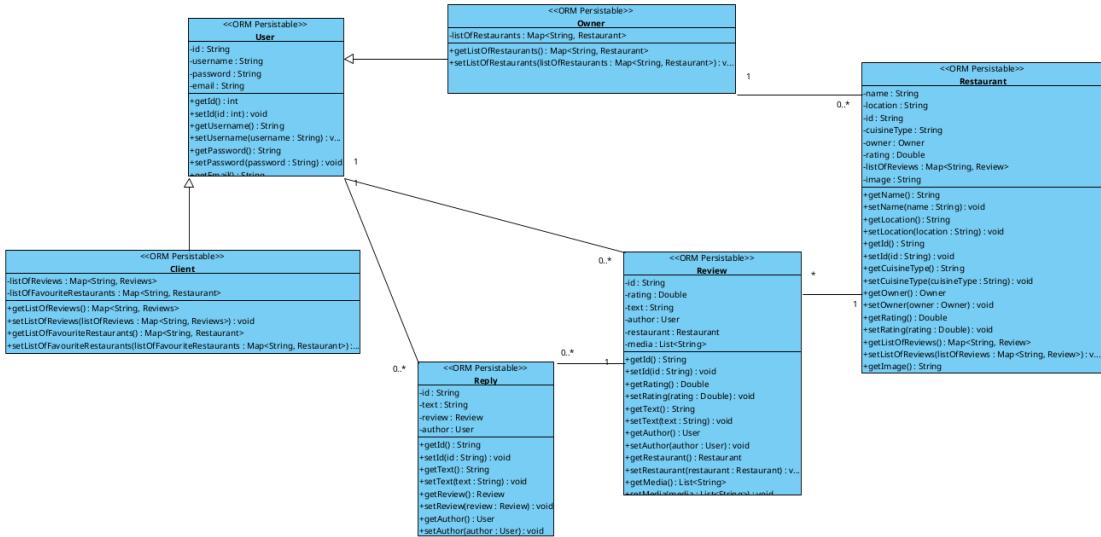


Figura 3: Diagrama de classes - PIM

As classes presentes no diagrama constituem a base estrutural permitindo a implementação das funcionalidades do sistema da seguinte forma:

- **User**: retrata qualquer utilizador na aplicação. Contém os atributos `id`, `username`, `password` e `email` e os métodos de acesso e modificação dos mesmos. Desta classe são derivadas duas outras, **Client** e **Owner**.
- **Client**: representa os utilizadores registados na aplicação que podem fazer avaliações e guardar uma lista de restaurantes favoritos. Para isso, contém duas listas, `listOfReviews` e `listOfFavouriteRestaurants`, em que ambas utilizam mapas (`Map<String, Review>` e `Map<String, Restaurant>` respetivamente) associando cada entidade ao respetivo identificador.
- **Owner**: caracteriza os utilizadores registados que também tem pelos menos uma página de um restaurante associado a eles, tendo uma lista para os guardar (`listOfRestaurants`). Esta lista está organizada num mapa `Map<String, Restaurant>`.
- **Restaurant**: descreve os estabelecimentos disponíveis na nossa plataforma. Cada um pode conter os seguintes atributos `name`, `location`, `cuisineType`, `rating` médio e `image`, estando associado diretamente a um **Owner**. Para além disso, também tem guarda uma lista das avaliações recebidas (`listOfReviews`).
- **Review**: representa as avaliações dadas pelos utilizadores registados aos restaurantes disponíveis. Inclui os atributos `rating`, `text`, `media`, uma ligação ao seu `author`, o cliente que fez a avaliação, e uma ligação ao `restaurant`, o restaurante que foi avaliado. Cada cliente pode avaliar diversos restaurantes, e cada restaurante pode ser avaliado múltiplas vezes. Esta classe está associada à classe **Reply**.
- **Reply**: corresponde às respostas dadas aos comentários feitos, sendo que estas podem ser apenas feitas pelo proprietário do restaurante avaliado. Cada resposta está apenas associada a uma única avaliação. Os atributos são `text`, `author`, o proprietário do restaurante, e `review`, a avaliação que está a ser respondida.

## 2.5. Protótipos de Interface

Como parte do processo de design da aplicação, foram desenvolvidos protótipos da interface com recurso a *Figma*. O objetivo era representar de forma visual a organização das funcionalidades pensadas para a plataforma.

## 2.5.1. Página Inicial

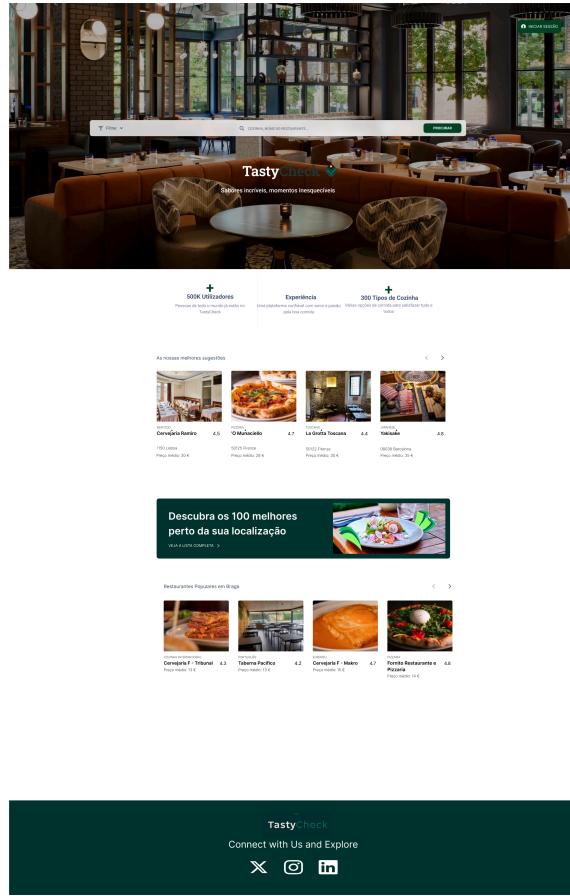


Figura 4: Protótipo da Página Inicial

## 2.5.2. Página de Login

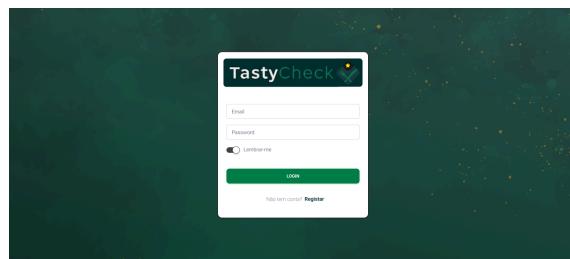


Figura 5: Protótipo da Página de Lógin

### 2.5.3. Página de Restaurante

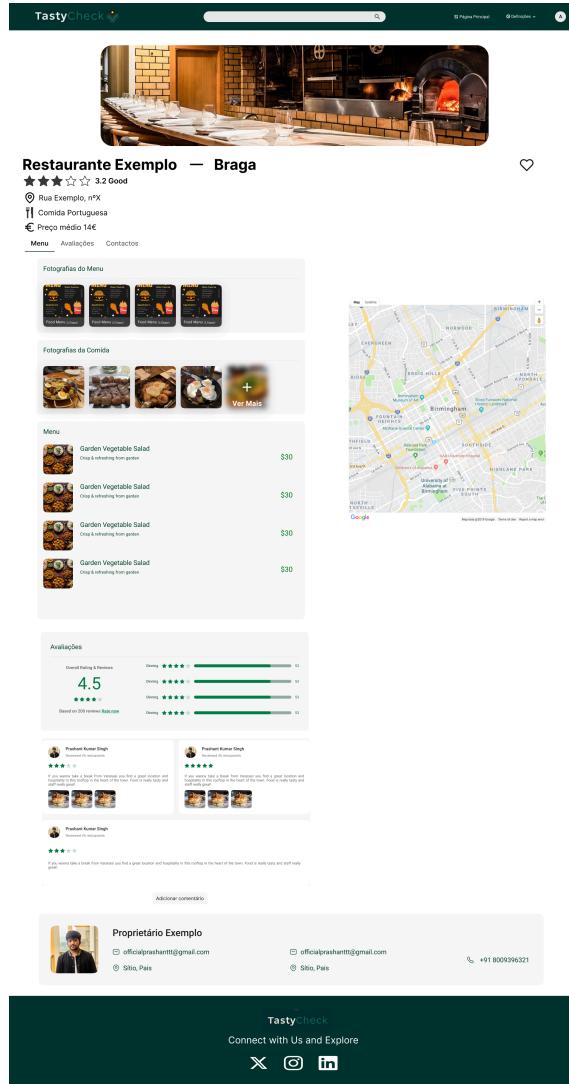


Figura 6: Protótipo da página do Restaurante

### 2.5.4. Avaliação de Restaurante

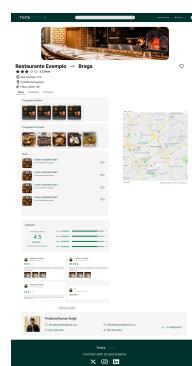
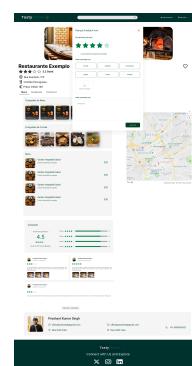
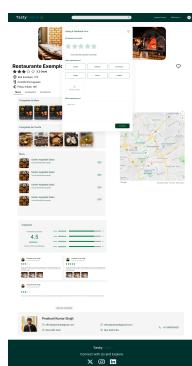


Figura 7: Protótipo da Avaliação de Figura 8: Protótipo da Avaliação de Figura 9: Protótipo da Avaliação de um Restaurante - parte 1      um Restaurante - parte 2      um Restaurante - parte 3

## 2.5.5. Perfil do Cliente

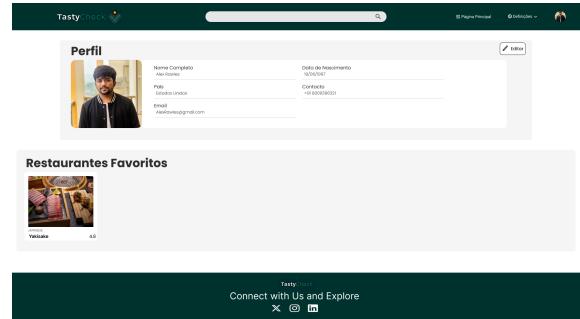


Figura 10: Protótipo do Perfil de um Cliente

## 2.5.6. Perfil do Proprietário

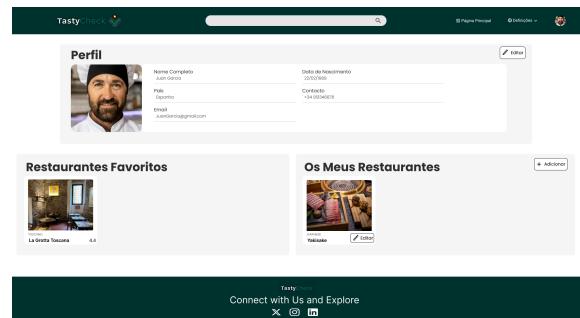


Figura 11: Protótipo do Perfil de um Cliente

## 3. Implementação

Tendo como base a modelação definida anteriormente, esta secção descrever a implementação técnica da aplicação *TastyCheck*. Seguidamente, serão descritas as decisões arquiteturais, as tecnologias usadas, e o funcionamento das principais camadas da aplicação.

### 3.1. Arquitetura do Sistema

A arquitetura da aplicação adotada usa como base de estruturação o modelo de três camadas, apresentação, lógica e dados, organizada de acordo com o padrão arquitetural *MVC* (Model-View-Controller). A nossa escolha teve como objetivo garantir uma boa separação de responsabilidades, promover a escalabilidade da aplicação e facilitar a sua manutenção.

O padrão utilizado permitiu a divisão da camada lógica em três componentes, os *Models*, que caracterizam as entidades de negócios e as regras associadas, os *Controllers*, que tratam dos pedidos externos e os *Services*, que englobam a lógica de negócio, interagindo com os controladores e modelos. Este tipo de estrutura modular tem como vantagem a habilidade de reutilização de código, o isolamento de responsabilidades e a facilidade de realização de testes unitários, sendo por isso uma abordagem seguida por muitas aplicações web modernas.

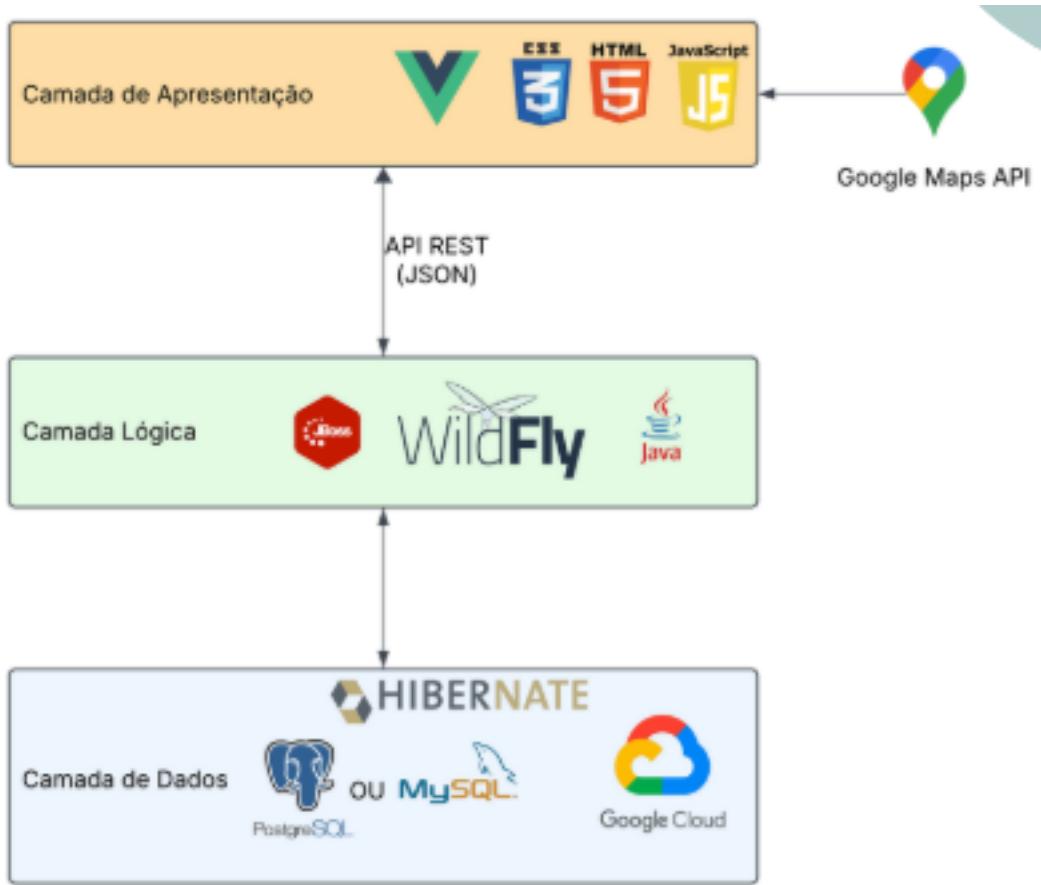


Figura 12: Diagrama de Técnologias

A camada de Apresentação é responsável pela interface gráfica com a qual os utilizadores interagem. Esta foi implementada utilizando a framework **Vue.js**, complementada por HTML5, CSS3 e JavaScript. É através desta camada que os utilizadores têm acesso às funcionalidades do sistema, como a pesquisa de restaurantes, visualização de avaliações, entre outras. Adicionalmente, foi integrada a **Google Maps API**, para responder à funcionalidade de apresentação de restaurantes num mapa interativo.

O desenvolvimento da camada da lógica de negócios foi concebido em **Java**, com recurso ao **servidor WildFly**, onde estão os serviços REST encarregues dos pedidos provenientes da interface. Estes pedidos são tratados pelos *Controllers*, que delegam a execução lógica aos *Services*. As entidades persistidas no sistema são representadas pelos *Models*. Assim, garantimos uma distribuição de responsabilidades equilibrada dentro do servidor aplicacional.

A camada de dados gere o armazenamento persistente da informação da plataforma. Esta dá uso à framework **Hibernate** para realizar o mapeamento ORM de modo a facilitar a comunicação entre a camada lógica e a base de dados relacional, **PostgreSQL**. Além disso, foi utilizado o **Google Cloud Storage** para o armazenamento de ficheiros multimédia, pois garante fiabilidade e escalabilidade no acesso a estes conteúdos estáticos.

A escolha desta arquitetura atesta a separação entre os componentes da aplicação e promove um desenvolvimento modular, o que torna o sistema mais robusto, extensível e de fácil manutenção. Paralelamente, o uso do padrão *MVC* favorece uma melhor organização da lógica interna do servidor aplicacional, porque ajuda na definição clara das responsabilidades de cada camada.

## 3.2. Camada de Apresentação

A camada de apresentação representa a interface gráfica da plataforma e é onde acontece a interação com os utilizadores. Foi desenvolvida utilizando a framework Vue.js, complementado por HTML5, CSS3 e JavaScript. Esta camada é responsável pela apresentação da informação vinda do backend e por guardar os inputs dos utilizadores de forma dinâmica e reativa.

### 3.2.1. Interfaces

Levando em consideração os protótipos desenvolvidos na fase de modelação (Secção 2.5), esta camada tem como objetivo implementar as páginas funcionais da aplicação, que permitem os utilizadores aceder às funcionalidades do sistema. Assim, as principais interfaces criadas foram:

#### 3.2.1.1. Página inicial

Esta é a primeira página visível quando se acede ao *TastyCheck*. O seu design final segue a mesma estrutura que o protótipo apresentado, mas os seus elementos visuais estão mais refinados e reativos.

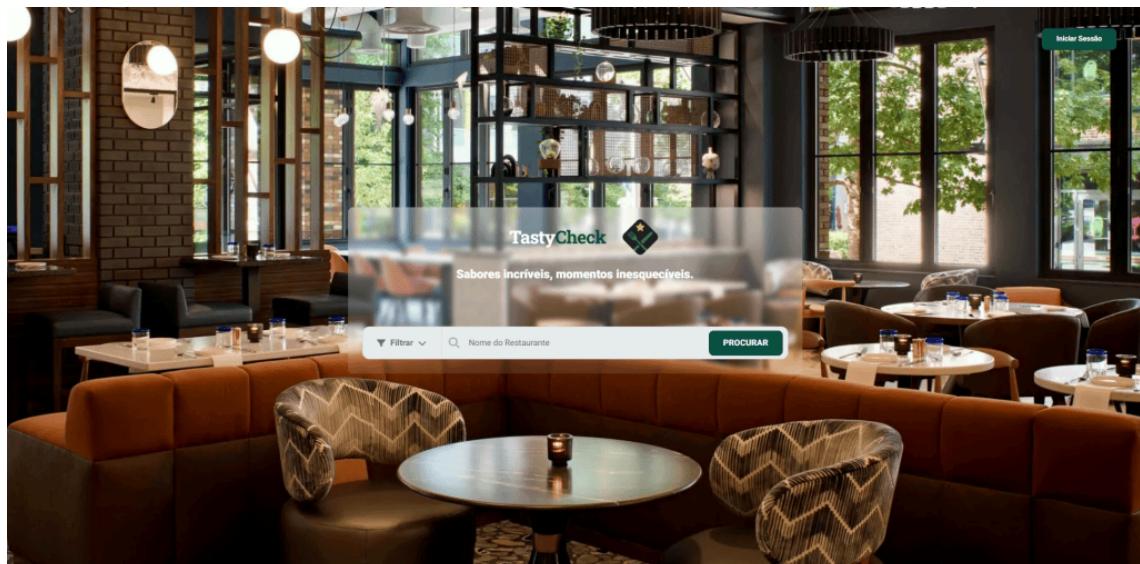


Figura 13: Página inicial - parte 1

A screenshot of the TastyCheck homepage featuring three main statistics: '500+' users from around the world, '15+' cuisines, and '1,000+' reviews. Below these are sections for 'Melhores Sugestões' (Burger Blast, Thai Spice, Coq au Vin, El Mariachi) and a call-to-action 'Descubra os 100 melhores perto da sua localização' with a 'VEJA A LISTA COMPLETA.' link. A small image of a dish is visible in the bottom right corner.

Figura 14: Página inicial - parte 2

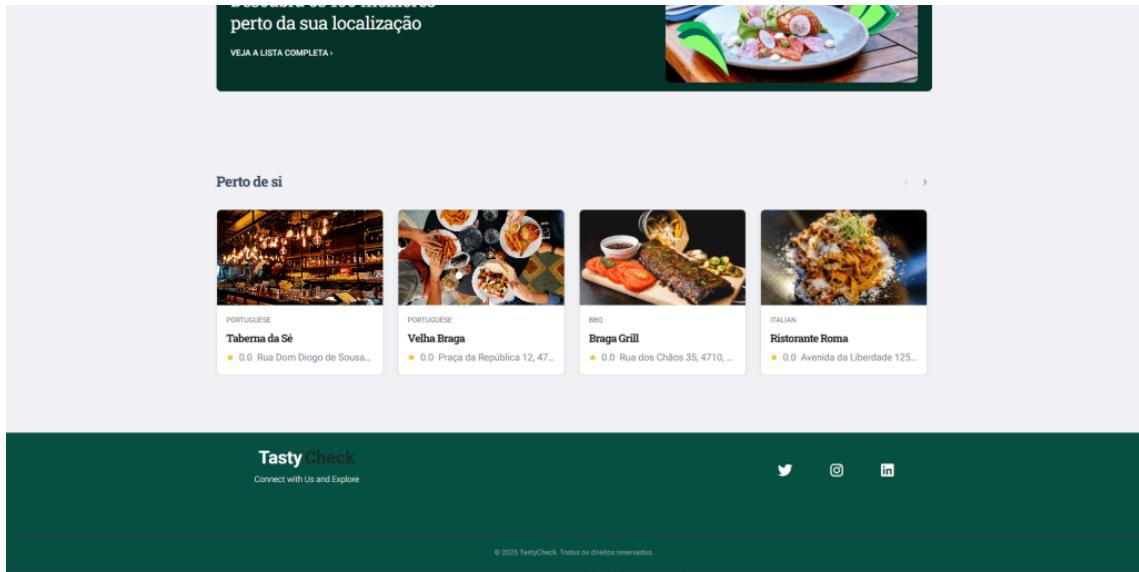


Figura 15: Página inicial - parte 3

Visualmente, destaca-se um banner central translúcido com o logótipo da aplicação, um slogan e a barra de pesquisa principal, de modo a permitir a pesquisa de restaurantes por nome e a aplicação de filtros na mesma. Os filtros incluem tipo de cozinha, avaliação ou localização.

Por baixo deste cabeçalho, são apresentadas estatísticas resumidas sobre a plataforma, tais como o número de utilizadores, tipos de cozinha disponíveis e avaliações realizadas, de forma a que os utilizadores fiquem a conhecer um pouco mais sobre a aplicação, reforçando a credibilidade da mesma.

Desta forma, nesta página podemos presenciar as seguintes funcionalidades:

- iniciar sessão: se um utilizador clicar no botão que diz iniciar sessão, que se encontra no canto superior direito, é direcionado para a página de *login*.
- pesquisa de restaurantes: permite ao utilizador pesquisar através do nome, filtrar os resultados e iniciar a navegação;
- sugestões de restaurantes: existe uma secção dinâmica que destaca alguns restaurantes disponíveis na aplicação, tendo por base a sua popularidade ou curadoria. Assim, cada cartão apresenta o nome de um restaurante, tipo de cozinha, localização e avaliação média.
- restaurantes perto do utilizador: esta secção mostra sugestões com base na localização geográfica, tendo sido integrada via a API do Google Maps.
- acesso a páginas detalhadas: se o utilizador clicar num restaurante é levado para a respetiva página. Nessa página é possível não só observar em mais detalhe as informações do restaurante selecionado, mas também tem a possibilidade de visualizar, responder ou deixar comentários, conforme o tipo de utilizador.
- rodapé informativo: esta página termina com um rodapé que inclui links para as redes sociais da plataforma, informação sobre o ano da criação da aplicação e de novo o logótipo da mesma.

### 3.2.1.2. Login

A página de *login* é o sítio no qual os utilizadores se podem autenticar na aplicação. A interface final não se distânciaria do protótipo proposto, tendo como diferenças um refinamento dos elementos gráficos e um botão “Voltar” para o utilizador poder regressar à página inicial mais facilmente.

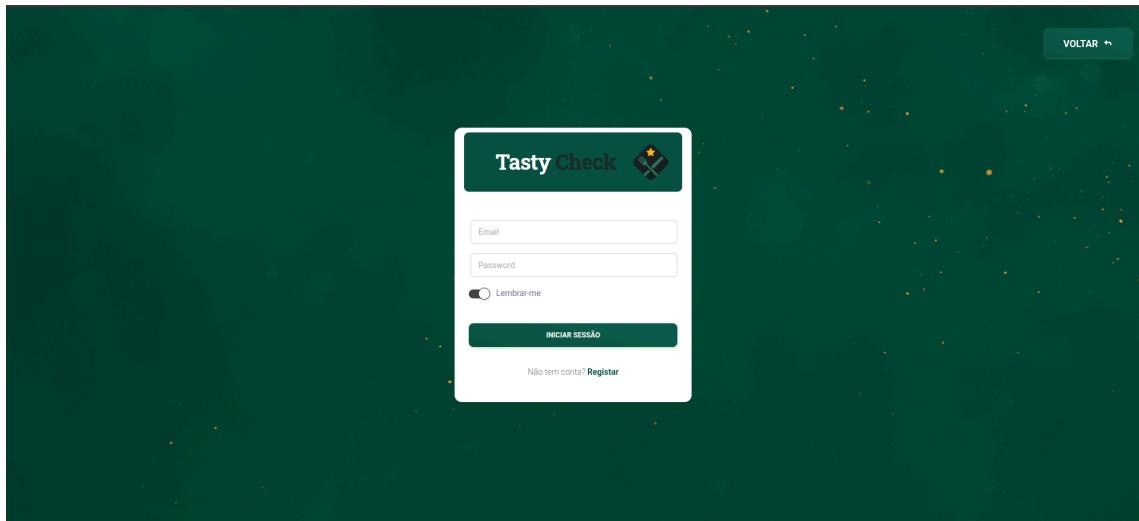


Figura 16: Página de Login

Deste modo, é possível observar um formulário central, que destaca o logo da aplicação no topo, os campos de email e password, um toogle de “Lembrar-me” e o botão de ação “Iniciar sessão”.

Em termos da autenticação, o formulário inclui uma validação do input posto nos campos de email e password, enviando-os para o backend via pedido HTTP POST para os endpoints de autenticação. Se o utilizador selecionar a opção de lembrar sessão, esta fica armazenada.

Adicionalmente, também é possível visualizar o botão de “Registar” que redireciona o utilizador para a página de criação de conta, tornando a navegação mais fluida.

### 3.2.1.3. Página de Registo

Nesta página é feita a criação de uma nova conta no sistema. Como se pode ver, a identidade visual é igual às restantes páginas, com o fundo escuro e detalhes dourados, para haver contraste com o formulário branco central.

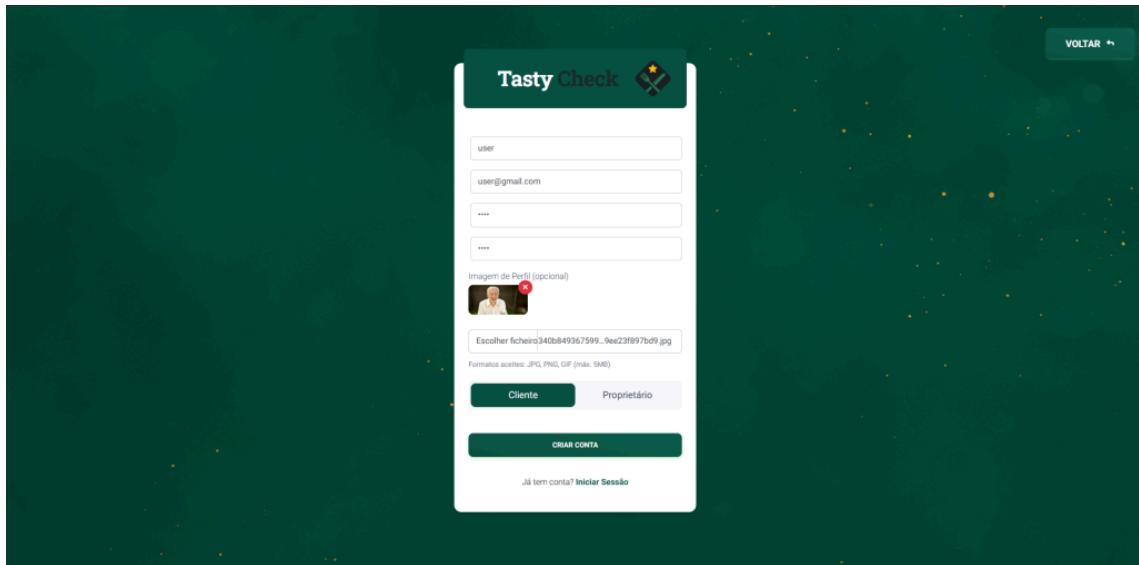


Figura 17: Página de Registo

Neste formulário são pedidos os seguintes atributos:

- nome de utilizador;
- email;
- password e confirmação da mesma;
- imagem de perfil, com pré-visualização dela se carregada;

- seleção de perfil a partir de botões visuais, pois existem dois tipos de utilizadores registados na aplicação.

A imagem de perfil suporta os formatos comuns - .jpg, .png e .gif - e pode ser carregada diretamente do dispositivo do utilizador, conforme está indicado na interface. Se não for fornecido nenhum ficheiro para este propósito, o sistema atribui uma imagem genérica.

Quando os campos estão preenchidos, o utilizador pode escolher “Criar Conta”, que como indica o botão cria a sua conta, tornando-se num utilizador registado.

Tal como na página de *login*, a interface incluiu no canto superior direito um botão “Voltar” que redireciona o utilizador para a página inicial da aplicação.

Caso o utilizador se tenha enganado e já esteja registrado mas não autenticado pode ser ir para a página de *login* ao clicar no botão “Iniciar Sessão”, que se encontra no final do elemento central.

### 3.2.1.4. Perfil do Utilizador

Após a inicialização da sessão, os utilizadores passam a ter acesso a uma área reservada, onde podem gerir a sua conta, consultar os dados pessoas e aceder à sua lista de restaurantes favoritos. Esta interface adapta-se ao tipo de utilizador - cliente ou proprietário - acrescentando, no caso de ser proprietário, um sítio para aceder às páginas dos seus restaurantes.

#### 1. Perfil do Cliente

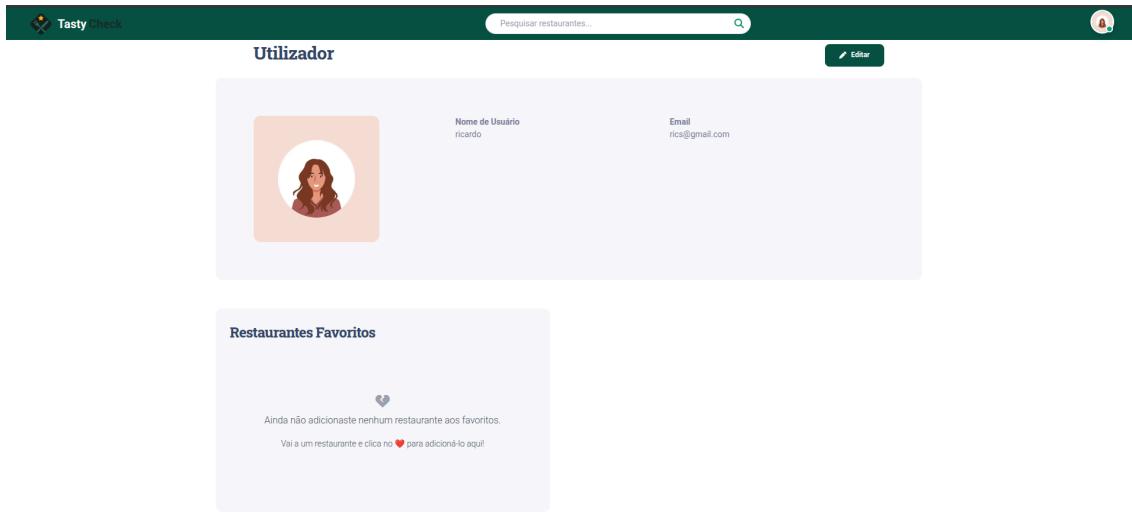


Figura 18: Página de perfil de um cliente

O cliente tem acesso a uma página de perfil simples e clara, em que pode visualizar os seus dados pessoais, como nome de utilizador, email e imagem de perfil (caso tenha sido submetida).

Depois desta secção de informações, encontra-se a lista dos “Restaurantes Favoritos”, que mostra os restaurantes que o cliente selecionou com um coração. Se esta se encontrar vazia, é apresentada uma mensagem incentivadora da procura de novos restaurantes.

Para além disso, encontramos no topo direito o botão “Editar”, que permite a atualização dos dados do perfil e alteração da imagem, caso seja o que o utilizador deseje.

#### 1. Perfil do Proprietário

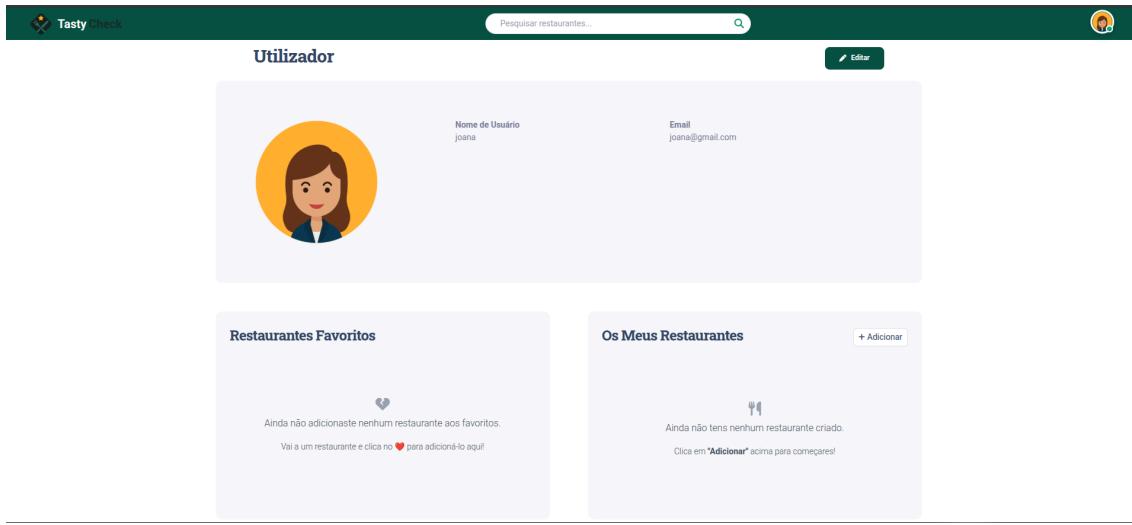


Figura 19: Página de perfil de um proprietário

No caso do utilizador ser um proprietário, a única diferença é que tem uma outra secção “Os Meus Restaurantes”, ao lado da secção dos restaurantes favoritos, que lista todas as páginas dos restaurantes criadas pelo utilizador, permitindo um acesso mais direto às mesmas.

Quando vazia, é exposta uma mensagens com essa informação juntamente com um botão “Adicionar”, que redireciona o utilizador para a página de criação de um perfil de um restaurante.

The screenshot shows the 'Criar Novo Restaurante' (Create New Restaurant) form. At the top, there is a placeholder 'Imagen do Restaurante' with a camera icon. Below it, there is a placeholder 'Imagen de Destaque' with a camera icon. The form contains several input fields: 'Nome do Restaurante \*', 'Localização \*', 'Tipo de Cozinha \*', 'Imagens do Menu (opcional)', and 'Imagens da Comida (opcional)'. Each field has a 'Choose File' button and a 'No file chosen' message. There is also a note 'Imagens da Comida (opcional)' below the second image field.

Figura 20: Página de criação de um perfil de restaurante

Caso já existam restaurantes, estes são apresentados num formato de cartão com imagem, nome, um botão de remoção e um botão de editar.

A versão final desta página facilita a personalização da experiência e o papel ativo do utilizador na aplicação, sendo uma parte crucial de gestão e navegação individual.

### 3.2.1.5. Página de Restaurante

Uma das páginas mais complexas e informativas da aplicação é a página de restaurante, onde está disposta informação detalhada sobre cada estabelecimento.

Figura 21: Página de perfil de um restaurante - parte 1

Figura 22: Página de perfil de um restaurante - parte 2

Figura 23: Página de perfil de um restaurante - parte 3

Esta página está dividida em diferentes secções com objetivos específicos sendo estas:

- **Informação geral:** No topo da página, é apresentado o nome do restaurante, classificação média em forma de estrelas, localização, tipo de cozinha, preço médio, contactos e horário de funcionamento.

mento. Estas informações podem ser atualizadas pelo proprietário ou adicionadas por um cliente. Do outro lado desta informação, encontra-se um coração, caso o utilizador clique no *icon*, o restaurante passa a aparecer na sua lista de favoritos, sendo essa informação demonstrada através da passagem da cor do coração de cinza para vermelho.

- **Galeria de Imagens:** Debaixo da secção das informações principais, são apresentados dois carrosséis, “Fotografias do Menu” e “Fotografias da Comida”, com imagens dos menus disponíveis no restaurante e dos pratos servidos, respetivamente. Estas imagens podem ser carregadas previamente pelo proprietário do restaurante ou adicionadas posteriormente por clientes.
- **Localização:** No lado direito da página é possível observar um mapa interativo, com integração da Google Maps API, que facilita ao utilizador a localização do restaurante. As coordenadas do restaurante são fornecidas pelo proprietário e ficam armazenadas num marcado dinâmico.
- **Menu (Pratos individuais):** Nesta secção, são listados os pratos disponíveis para pedir no restaurante, mostrando o nome, descrição dos ingredientes e preço. Esta funcionalidade foi pensada não só para permitir que o utilizador pudesse explorar a oferta do restaurante antes de fazer uma reserva ou visita mas também como uma maneira dos restaurantes poderem destacar especialidades deles.
- **Avaliações:** Para além de estar disponível a média de avaliação do restaurante na zona das informações principais, esta tem a sua própria secção com adição de informação mais específica e uma funcionalidade. Assim a informação adicionada é o número total de reviews e a funcionalidade é um botão de ação “Avaliar já”, onde o cliente abre um *popup* para escrever a sua avaliação do restaurante. Abaixo, são listadas as avaliações individuais deixadas pelos clientes. Estas incluem o nome do autor, data classificação, comentário. Estes comentários podem conter imagens. Se o utilizador for proprietário do restaurante, pode responder a estas avaliações clicando no botão “Comentar”.

### 3.2.1.6. *Popup* de avaliação de um restaurante

A funcionalidade de avaliação de um restaurante é uma das partes mais cruciais do propósito da aplicação, permitindo aos utilizadores partilhar as suas experiências. Assim, para realizar esta ação aparece um *popup* acessível a partir da página do restaurante que se quer avaliar, como referido anteriormente.

A versão final é visualmente distinta do protótipo, contudo apresenta melhorias a níveis de clareza visual, interação com o utilizador e integração técnica com os componentes do sistema.

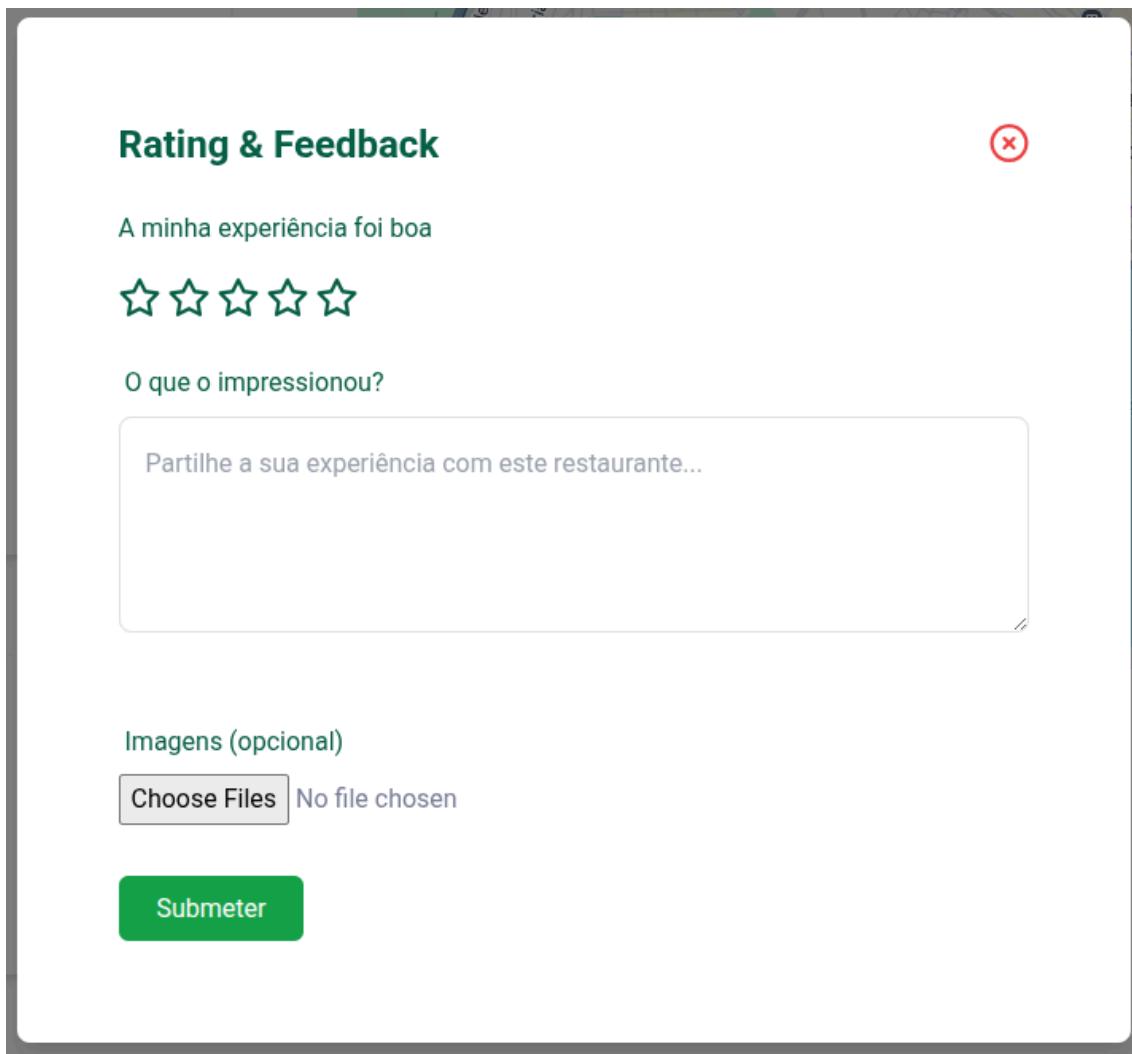


Figura 24: Popup de avaliação de um restaurante

No formulário podemos preencher os seguintes campos:

- **classificação com estrelas**, com um feedback textual;
- **campo de texto**, para o utilizador podes descrever a sua experiência;
- **campo de upload de imagem**, caso o utilizador queira adicionar elementos visuais à sua avaliação
- **botão “Submeter”**, que envia a avaliação para o backend.

### 3.3. Camada da Lógica de Negócio

A partir dos princípios do modelo MVC, foi possível o desenvolvimento de uma camada lógica de negócio segundo um arquitetura modular e orientada a objetos. Esta camada exprime a parte funcional da aplicação, onde são aplicadas as regras de negócio e processada a informação que passa entre a interface do utilizador e a base de dados.

A implementação encontra-se dividida em cinco níveis principais, que contribuíram para a separação das responsabilidades e para a facilidade de manutenção: *Models*, *DTO's*, *Services*, *DAO's* e *Controllers*.

#### 3.3.1. Diagrama de Classes PSM - *Models*

As entidades do domínio da aplicação são representadas pelos *Models*, que expressão a estrutura da base de dados e servem, com o Hibernate, como base para o mapeamento ORM. Cada classe engloba os atributos essenciais, construtores, métodos e anotações para persistência.

Na Figura 25, podemos observar que a classe User é abstrata e estendida por dois subtipos, Client e Owner. A entidade Restaurant está relacionada a um Owner e também contém uma lista de Review, que por sua vez possui um Reply. Também é possível denotar as relações com Image, por causa de menus e comida) e Dish. Assim, este diagrama serve para ser mais fácil compreender o modelo relacional e o mapeamento entre objetos e base de dados.

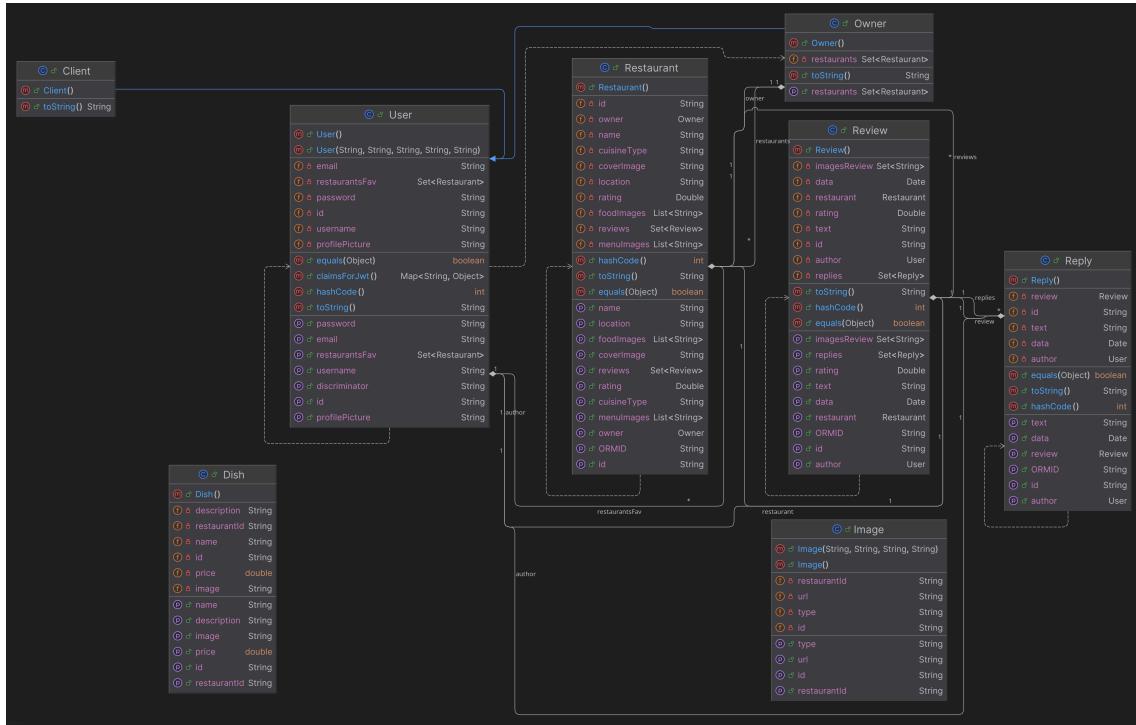


Figura 25: Diagrama de classes *PSM - Models*

### 3.3.2. Diagrama de Classes PSM -DTO's

Os *DTO's* são usados para transportar dados entre o backend e o frontend sem a exposição direta das entidades do domínio. A Figura 26, mostra que as funcionalidades estão organizadas por subpacotes (*Restaurant*, *Review*, *Replay*,...).

A partir do diagrama podemos observar as diversas classes, que englobam apenas os dados necessários para uma comunicação REST, sem fuga de informação.

As classes são constituídas por construtores, getters e setters, em que os seus atributos são os dados que as interfaces gráficas da plataforma mostram.

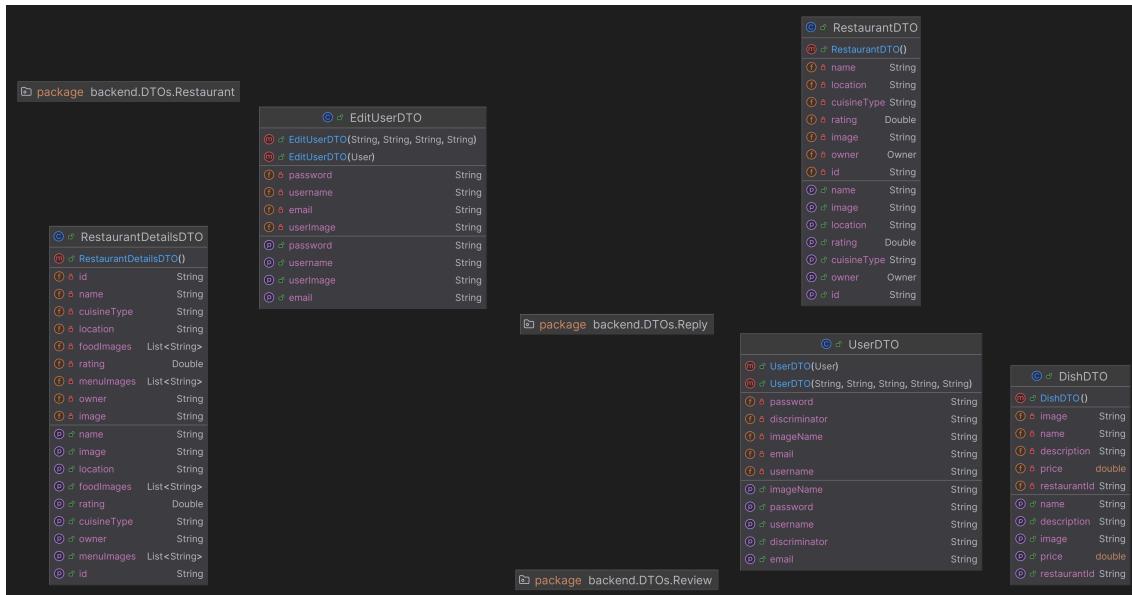


Figura 26: Diagrama de classes PSM - DTO's

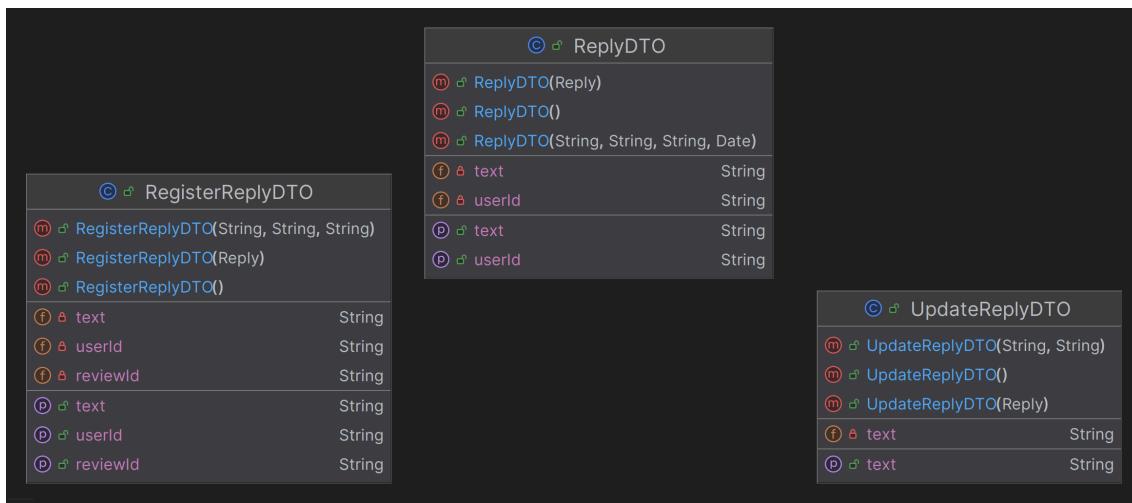


Figura 27: Diagrama de classes PSM - DTO's Reply

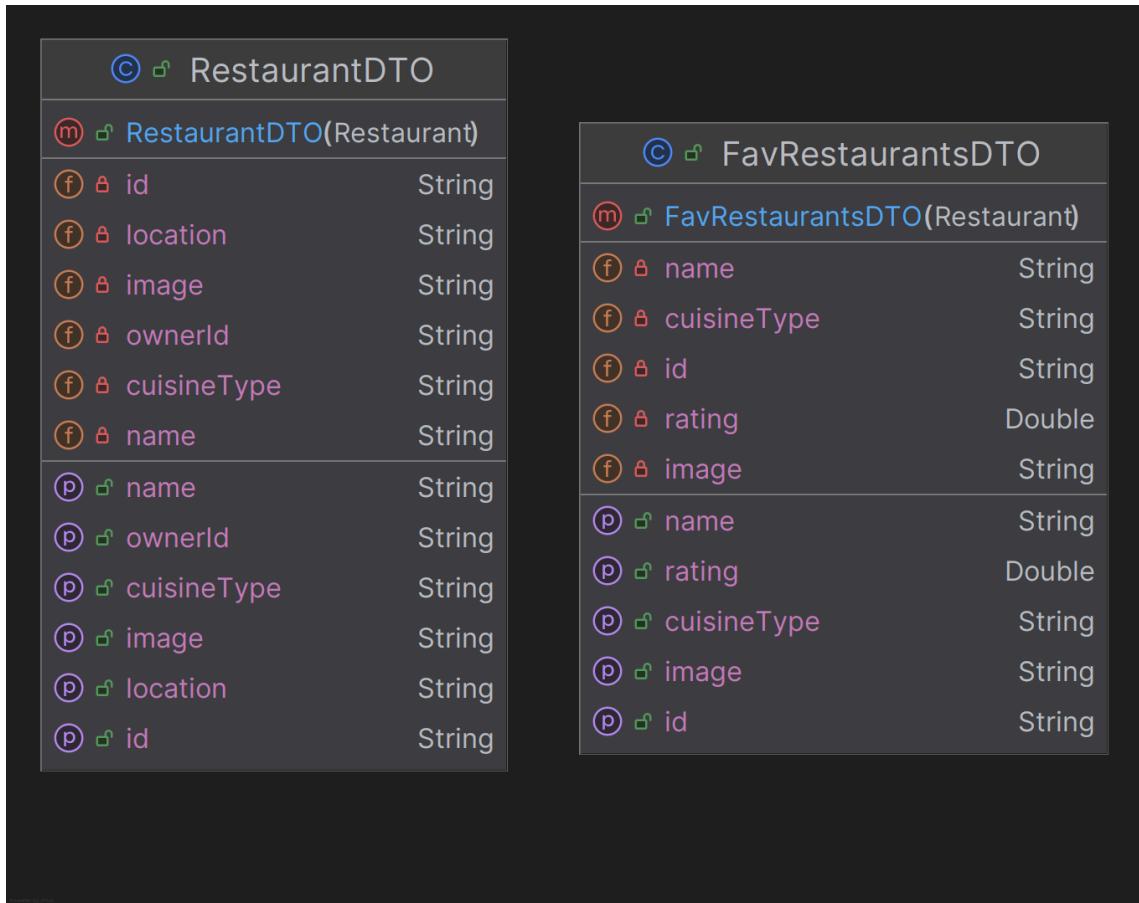


Figura 28: Diagrama de classes PSM - DTO's Restaurant

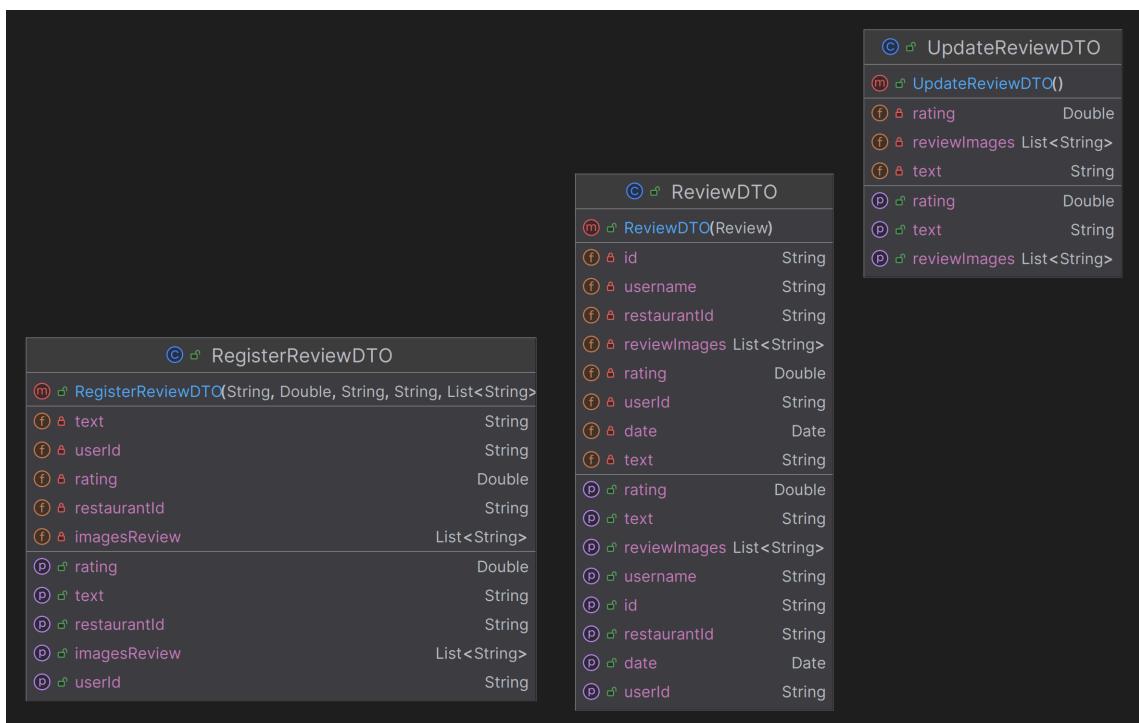


Figura 29: Diagrama de classes PSM - DTO's Review

### 3.3.3. Diagrama de Classes PSM - Services

A lógica de negócio é caracterizada nesta camada, que inclui serviços como `RestauranService`, `ReviewService`, `UserService`, `ReplayService`, entre outros.

É a partir destes serviços que são expostos os métodos que coordenam o funcionamento dos *DAO's* e *DTO's*, alguns exemplos são `createRestaurant()`, `registerReview()`, `deleteReply()`, e assim por diante. Alguns destes serviços podem ter responsabilidade específicas relacionadas com a autenticação e tokens `JWT` (`JWTService` e `AuthenticactionService`).

Com a ajuda do diagrama da Figura 30 fica mais fácil entender como os serviços colaboram e invocam métodos de várias entidades.

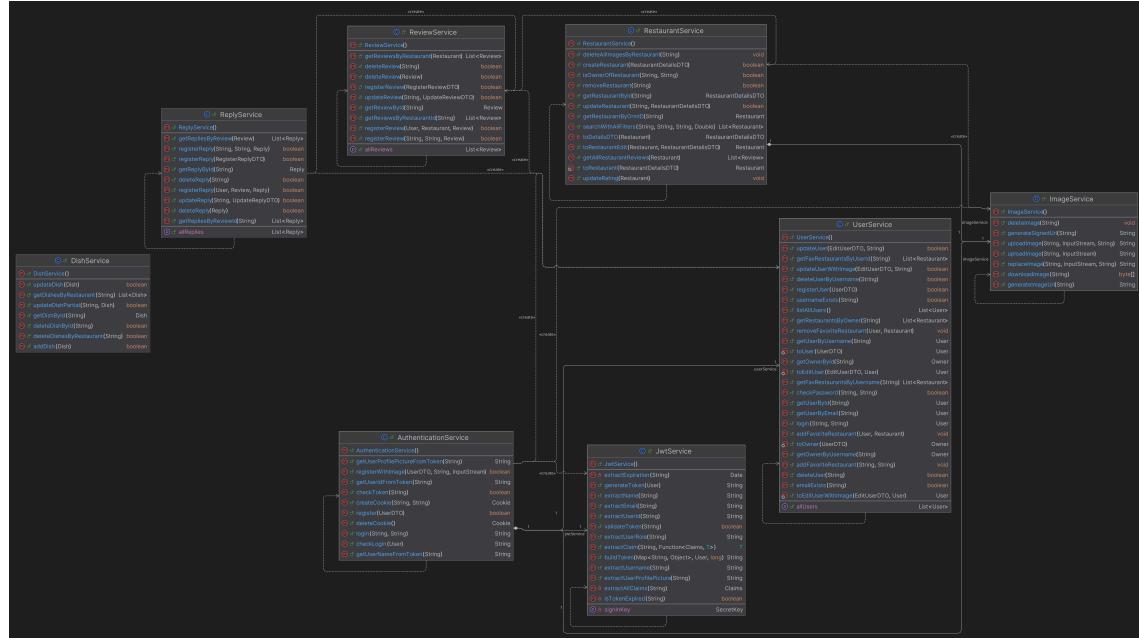


Figura 30: Diagrama de classes PSM - Services

### 3.3.4. Diagrama de Classes PSM - DAO's

Esta camada está encarregue da abstração da lógica de acesso à base de dados, através de classes como `RestaurantDAO`, `UserDAO`, `ReviewDAO`, por exemplo.

Cada uma destas classes tem métodos que permitem a execução de operações CRUD sobre as entidades, tal como `findById`, `save`, `delete`, e muitos outros. Os serviços ao usarem estas classes asseguram a separação entre a lógica e a persistência.



Figura 31: Diagrama de classes PSM - DAO's

### 3.3.5. Diagrama de Classes PSM - Controllers

A camada de *Controllers* é onde estão os endpoints HTTP do backend, sendo implementada via servlets Java. Na Figura 32 , conseguimos observar os diferentes *controllers* implementados.

São estes *controllers* que definem os métodos correspondentes às operações HTTP - *doGet*, *doPost*, *doPut* e *doDelete* - fazendo partido internamente dos serviços e *DTO's* para o processamento de pedidos e devolver respostas. Aqui também estão incluídos métodos auxiliares para o parsing de JSON ou contrução de objetos.

Assim, esta camada é o que une a interface gráfica à lógica de aplicação.

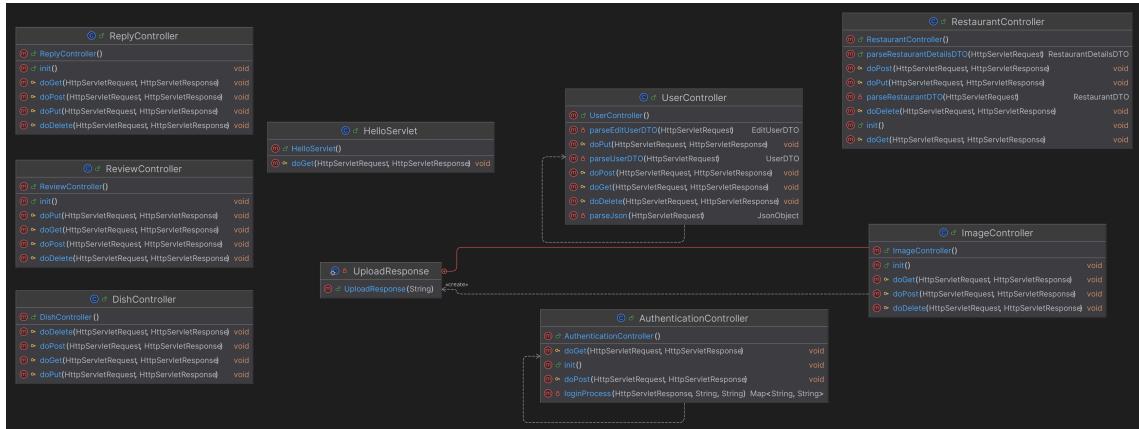


Figura 32: Diagrama de classes PSM - Controllers

## 3.4. Camada de Dados

A camada de dados é responsável pela gestão do armazenamento persistente da informação relativa com utilizadores, restaurantes, avaliações, comentários, imagens e favoritos. Para garantir isolamento, fiabilidade e portabilidade, ela está num docker container e foi implementada em PostgreSQL.

A comunicação entre o backend e a base de dados é feita via Hibernate, que permite representar as tabelas da base de dados como objetos Java. Assim, por causa desta framework, a camada de lógica de negócio pode interagir com os dados de forma abstrata, sem ter a necessidade de usar SQL direto.

A estrutura da base de dados é composta por oito tabelas, conforme se pode observar na figura Figura 33.

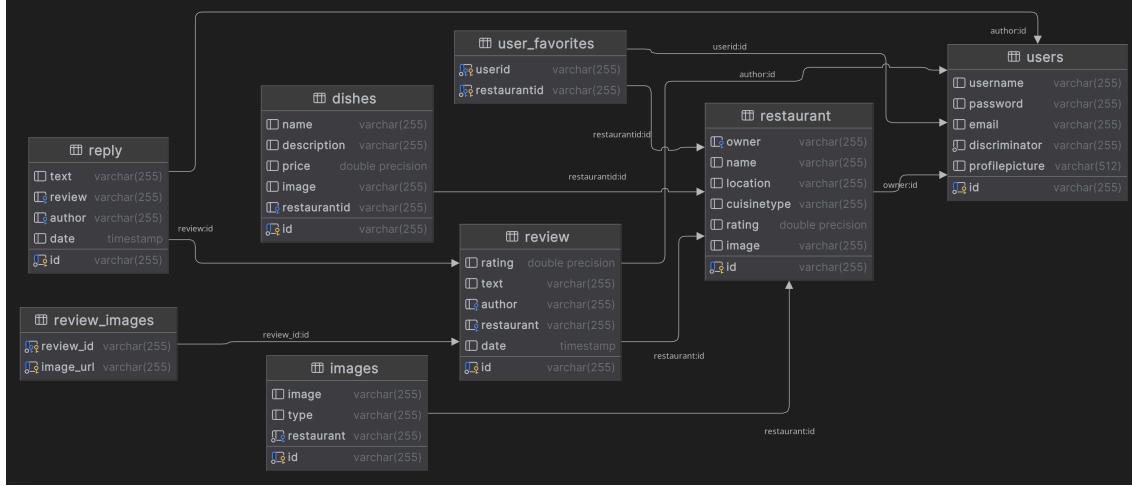


Figura 33: Esquema lógico da base de dados

Mais especificamente, a estrutura encontra-se organizada da seguinte maneira:

Campo	Descrição
id	Identificador único do utilizador
username	Nome do utilizador na aplicação
password	Palavra passe do utilizador
email	Email do utilizador
discriminator	Identifica o tipo de utilizador (se é cliente ou proprietário)
profilepicture	Fotografia de perfil do utilizador (opcional)

Tabela 1: Tabela *users* (informação dos utilizadores)

Campo	Descrição
id	Identificador único do restaurante
owner	Referência para o utilizador proprietário do restaurante (chave estrangeira para a tabela <i>users</i> )
name	Nome do restaurante
location	Localização do restaurante
cuisineType	Tipo de comida do restaurante
rating	Média das avaliações dadas pelos utilizadores
image	Imagen de perfil do restaurante

Tabela 2: Tabela *restaurant* (informação dos restaurantes disponíveis na aplicação)

Campo	Descrição
id	Identificador único da avaliação
rating	Número (double) entre 0 e 5
text	Texto da avaliação deixada
author	Referência para o utilizador que escreveu a avaliação (chave estrangeira para a tabela <i>users</i> )
restaurant	Referência para o restaurante que foi avaliado (chave estrangeira para a tabela <i>restaurant</i> )
date	Data da avaliação

Tabela 3: Tabela *review* (informação das avaliações)

Campo	Descrição
id	Identificador único da resposta deixada
text	Texto da resposta deixada
review	Referência para a avaliação que foi respondida
author	Referência para o proprietário que respondeu (chave estrangeira para a tabela <i>users</i> )
date	Data da resposta

Tabela 4: Tabela *reply* (informação das respostas)

Campo	Descrição
id	Identificador único da imagem
image	URL para a imagem
type	Categoria da imagem (comida, menu, cover, ...)
restaurant	Referência para o restaurante a que a imagem pertence (chave estrangeira para a tabela <i>restaurant</i> )

Tabela 5: Tabela *images* (informações das imagens)

Campo	Descrição
review_id	Referência para a avaliação (chave estrangeira para a tabela <i>review</i> )
image_url	Imagem deixada na avaliação

Tabela 6: Tabela *review\_images* (informações das imagens deixadas numa avaliação)

Campo	Descrição
userid	Referência para um utilizador (chave estrangeira para a tabela <i>users</i> )
restaurantid	Referência para um restaurante (chave estrangeira para a tabela <i>restaurant</i> )

Tabela 7: Tabela *userFavorites* (informações dos restaurantes favoritos dos utilizadores)

Campo	Descrição
id	Identificador único do prato
nome	Nome do prato
description	Descrição dos ingredientes do prato
price	Preço do prato
image	URL da Imagem do prato
restaurantid	Referência para o restaurante a que o prato pertence (chave estrangeira para a tabela <i>restaurant</i> )

Tabela 8: Tabela *dishes* (informações dos pratos dos restaurantes)

## 4. Deployment

O processo de *deployment* do nosso sistema, *TastyCheck*, é realizado numa máquina virtual (VM) da Google Cloud Platform, recorrendo à utilização de Docker compose para orquestrar todos os serviços necessários à aplicação. Com esta abordagem garantimos portabilidade, isolamento e consistência entre ambientes.

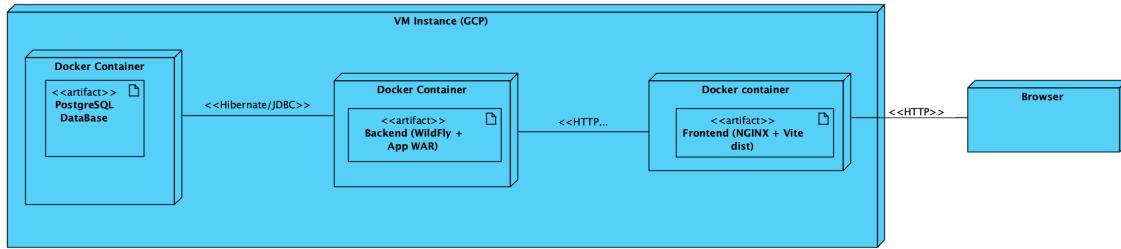


Figura 34: Diagrama de *deployment*

O Figura 34 caracteriza a organização dos conteiners utilizados no sistema. Aqui, é possível observar que a estrutura de *deployment* toda executada no mesmo host e que é constituída por quatro componentes:

- **Frontend (Vue.js)**: A interface gráfica é compilada (via `npm run build`) e os ficheiros estáticos são servidos por um servidor NGINX. O NGINX é responsável por responder aos pedidos feitos no browser e também atua como reverse proxy para o backend.
- **Backend (Java + WildFly)**: A lógica da aplicação é executada num container Docker com o WildFly, onde é feito o deploy do ficheiro .war. É neste container que residem os endpoints REST responsáveis por receber pedidos do frontend e interagir com a base de dados.
- **Base de Dados (PostgreSQL)**: O sistema da base de dados PostgreSQL corre num container isolado e recebe os dados provenientes do backend. A comunicação é feita através de JDBC/Hibernate. A base de dados é inicializada automaticamente a partir de um ficheiro .dump.
- **NGINX (Proxy + Static Server)**: Atua como ponto de entrada do sistema, servindo os ficheiros do frontend e encaminhando os pedidos REST para o backend.

Adicionalmente, conseguimos compreender como funciona o fluxo de comunicação entre os diferentes componentes do sistema. O frontend, servido por um servidor NGINX, envia pedidos HTTP para o backend, através dos endpoints REST definidos nos controladores da aplicação Java. O backend, por sua vez, recorre ao Hibernate para realizar as operações de acesso e manipulação de dados na base de dados PostgreSQL.

A utilização de Docker Compose para estruturar o deployment garante um comportamento consistente e previsível, independentemente do ambiente em que a aplicação é executada. Todas as dependências estão encapsuladas em containers isolados, o que facilita a replicação do sistema, a manutenção e uma possível escalabilidade horizontal em contextos mais exigentes.

## 5. Análise do sistema

Nesta secção. é realizada uma avaliação detalhada, quer do ponto de vista técnico (performance e escalabilidade), quer do ponto de vista da experiência do utilizador (usabilidade), do comportamento da aplicação *TastyCheck*. Para isso, foram executados testes de carga, que serviram para medir a capacidade de resposta de um sistema sob diferentes níveis de utilização, e análises de interface, com objetivo de identificar possíveis melhorias na interação com o utilizador.

### 5.1. Performance e Escalabilidade

A partir da ferramenta Apache JMeter, avaliamos a robustez e capacidade de resposta do nosso sistema. Foram realizados quatro tipos de testes de performance diferentes: teste de carga, teste de stress, teste de resistência e teste funcional com carga real. Estes permitiram medir o tempo média de resposta, a taxa de erro, o throughput e a estabilidade ao longo do tempo. O fluxo de interação do utilizador testado foi o mesmo para todos os testes, em que o utilizador faz login na aplicação, depois pesquisa um restaurante, seleciona a página do restaurante, vê as avaliações do mesmo, deixa uma avaliação, põe-no nos favoritos e por fim responde a uma avaliação deixada por outro utilizador.

#### 5.1.1. Teste de carga

O propósito da execução deste teste foi simular o comportamento do sistema sob uma utilização normal - cerca de 200 utilizadores simultâneos - a realizarem ações típicas.

A Figura 35 apresenta os resultados obtidos neste cenário.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
POST Login	2739	1317	239	2612	615.16	78.61%	1.1/sec	0.73	0.27	694.3
POST Review	2736	7434	687	53289	10733.66	77.74%	1.0/sec	0.64	55.66	625.8
POST Restaurant...	2739	801	230	2917	293.88	66.16%	1.1/sec	4.35	0.30	4135.3
POST Favorite	2649	1044	230	2938	410.42	64.93%	1.0/sec	0.51	0.29	521.3
GET Details Re...	2739	1210	234	2923	373.07	63.93%	1.1/sec	0.65	0.18	614.2
POST Reply de R...	2649	1295	235	3037	412.07	63.19%	1.0/sec	0.52	0.33	530.4
GET Reviews Res...	2739	12738	237	102855	23017.06	61.26%	1.1/sec	16.42	0.19	15965.9
TOTAL	18990	3715	230	103855	10566.53	58.01%	7.1/sec	23.36	56.93	3325.0

Figura 35: Resultados do teste de carga - resumo estatístico

A Figura 35 ilustra a evolução do tempo de resposta e throughput ao longo do tempo, que indica um aumento gradual da média e mediana sob carga progressiva.

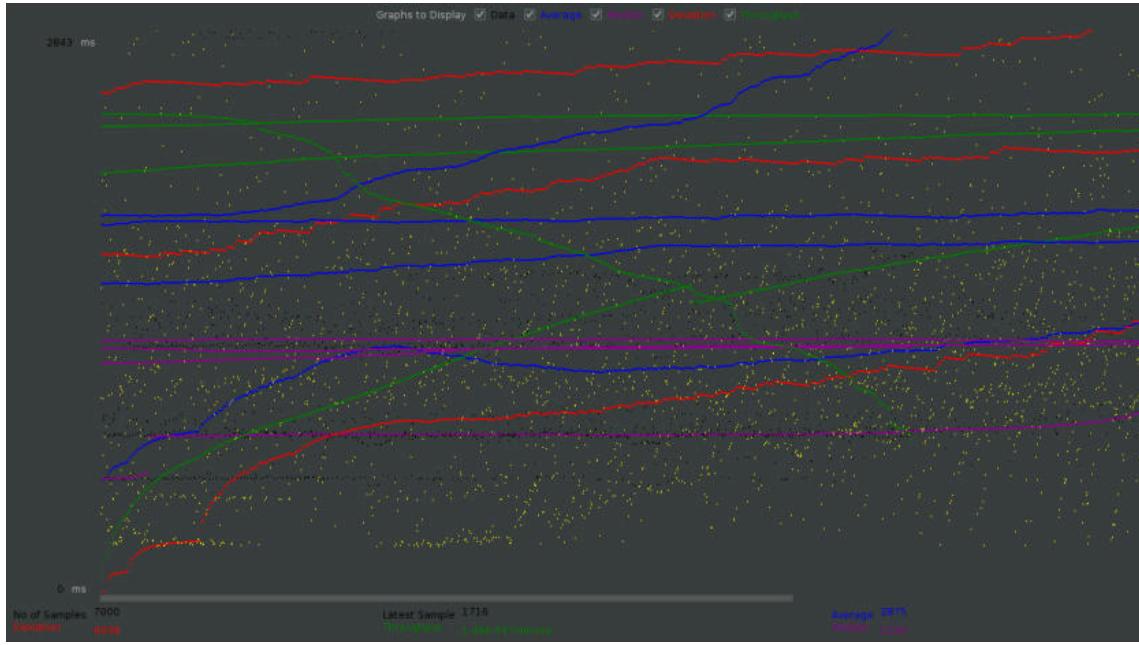


Figura 36: Gráfico de desempenho ao longo do tempo no teste

O tempo médio global foi de 5.2 segundos, sendo que o endpoint mais exigente foi o POST /review, com uma média de 12.6s e alguns picos superiores a 20s. Para além disso, conseguimos ver que a taxa de throughput do sistema se manteve aceitável com 39 req/min com erro médio de 7.72%.

### 5.1.2. Teste de stress

A partir deste teste, foi possível identificar qual o limite de carga suportado pela aplicação. Foram executadas mais de 2700 requisições por endpoint.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received kB/sec	Sent kB/sec	Avg. Bytes
POST Review...	1000	9471	850	41069	8242.03	90.90%	5.8/sec	3.35	311.21	588.5
POST Restaurant...	1000	4308	233	9806	2691.45	84.50%	13.4/sec	28.43	3.71	2179.2
POST Reply de R...	1000	4815	263	10401	2313.17	87.30%	5.7/sec	3.25	1.87	581.0
POST Login	1000	4467	248	10093	2833.92	90.70%	14.4/sec	8.50	3.64	604.3
POST Favorite	1000	5031	288	9495	2388.00	87.90%	5.8/sec	3.16	1.69	599.1
GET Reviews Res...	1000	14521	238	68760	21402.75	84.10%	7.2/sec	90.72	1.28	12917.8
GET DetailsRes...	1000	4768	239	9583	2641.40	84.30%	12.3/sec	7.15	2.09	586.7
TOTAL	7000	6789	233	68760	9628.22	87.10%	38.4/sec	96.51	310.71	2575.2

Figura 37: Resultados do teste de stress - resumo estatístico

Ao analisar a Figura 37, verificamos que as taxas de erro, especialmente nos endpoints de escritas, estavam significativamente elevadas (60% a 78%) e que a média de resposta caiu para 3.7s. Assim, podemos concluir que após um determinado ponto o sistema entra em colapso com quebras abruptas no throughput e aumento de latência.

### 5.1.3. Teste de resistência

Este teste tem como objetivo verificar o comportamento do sistema sob uma carga contínua durante um período de tempo prolongado (1 hora com 100 utilizadores constantes).

Abaixo, estão representados os principais resultados obtidos.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received kB/sec	Sent kB/sec	Avg. Bytes
POST Login	247	738	126	2261	308.63	6.48%	6.2/min	0.13	0.03	1265.8
POST Restaurant...	244	826	139	1571	229.06	6.15%	6.1/min	1.06	0.03	10656.1
GET DetailsRe...	240	1192	85	1320	207.88	7.08%	6.0/min	0.07	0.02	720.5
GET Reviews Res...	234	18685	40	38739	15429.11	11.97%	5.9/min	2.33	0.02	24437.5
POST Review	216	12645	100	21066	7204.83	12.04%	5.4/min	0.05	4.61	580.2
POST Favorite	200	780	239	1278	99.76	5.00%	5.0/min	0.03	0.02	428.8
POST Reply de R...	200	1540	245	2864	383.28	5.00%	5.0/min	0.03	0.03	412.2
TOTAL	1581	5210	40	38739	9462.01	7.72%	39.2/min	3.67	4.70	5754.3

Figura 38: Resultados do teste de resistência - resumo estatístico

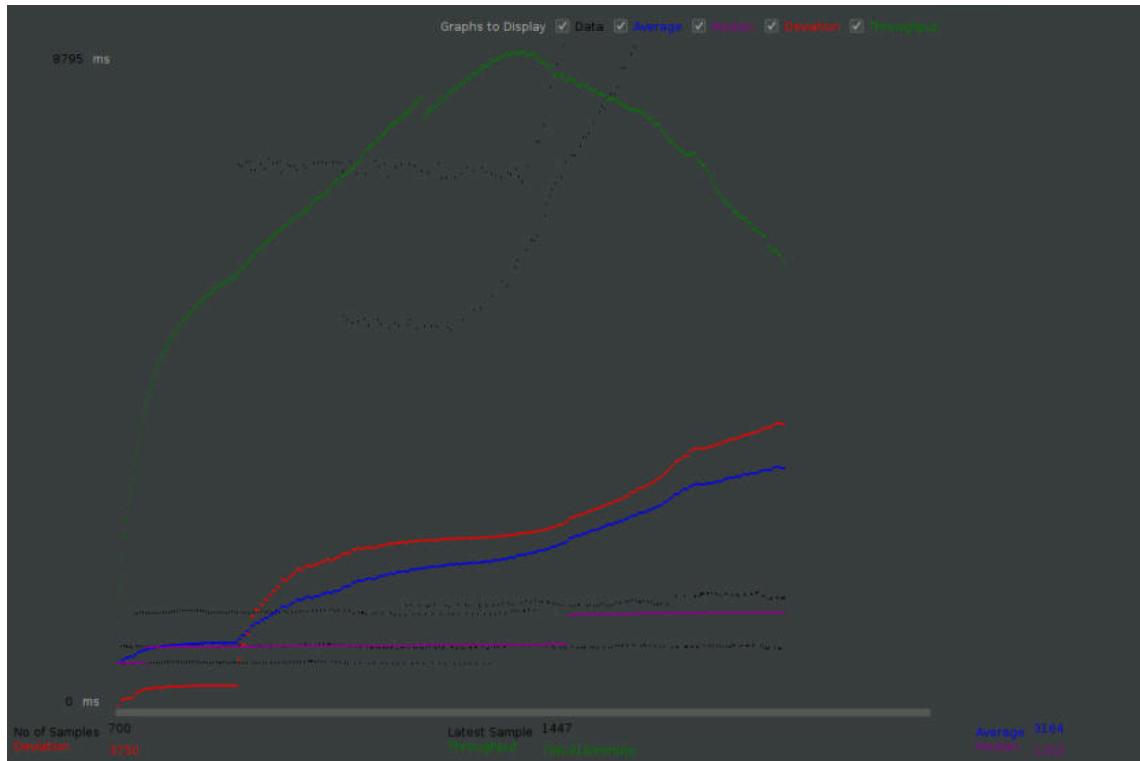


Figura 39: Desempenho do sistema durante teste de resistência

As figuras seguintes, Figura 38 e Figura 39, evidenciam que o tempo médio se manteve estável, 4.4s, e que não foram verificadas fugas de memória nem degradação do desempenho ao longo do tempo. Concluindo que o sistema se revelou estável sob utilização sustentada.

#### 5.1.4. Teste funcional com carga

Durante este teste, foram simuladas interações completas com a aplicação por parte de 50 utilizadores.

As seguintes figuras apresentam os resultados obtidos a partir da ferramenta usada.

(label)	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received kB/sec	Sent kB/sec	Avg. bytes
POST Login	400	627	547	1465	180.77	0.00%	6.0/min	0.12	0.03	1289.2
POST Restaurant...	400	815	752	1502	125.17	0.00%	6.0/min	1.09	0.03	11281.0
GET DetailsRes...	400	1241	1184	1923	47.52	0.00%	6.0/min	0.07	0.02	728.1
GET Reviews Res...	400	14731	2773	27947	B22.23	12.50%	5.9/min	1.65	0.02	17115.6
POST Review	350	11159	2966	19404	4830.29	0.00%	5.2/min	0.03	4.77	409.4
POST Favorite	350	787	755	1322	43.85	0.00%	5.2/min	0.04	0.03	421.1
POST Reply de R...	350	1558	1301	2296	239.87	0.00%	5.2/min	0.03	0.03	400.1
TOTAL	2650	4417	547	27947	6024.71	1.89%	38.9/min	3.01	4.87	4793.0

Figura 40: Resultados do teste de funcional com carga - resumo estatístico

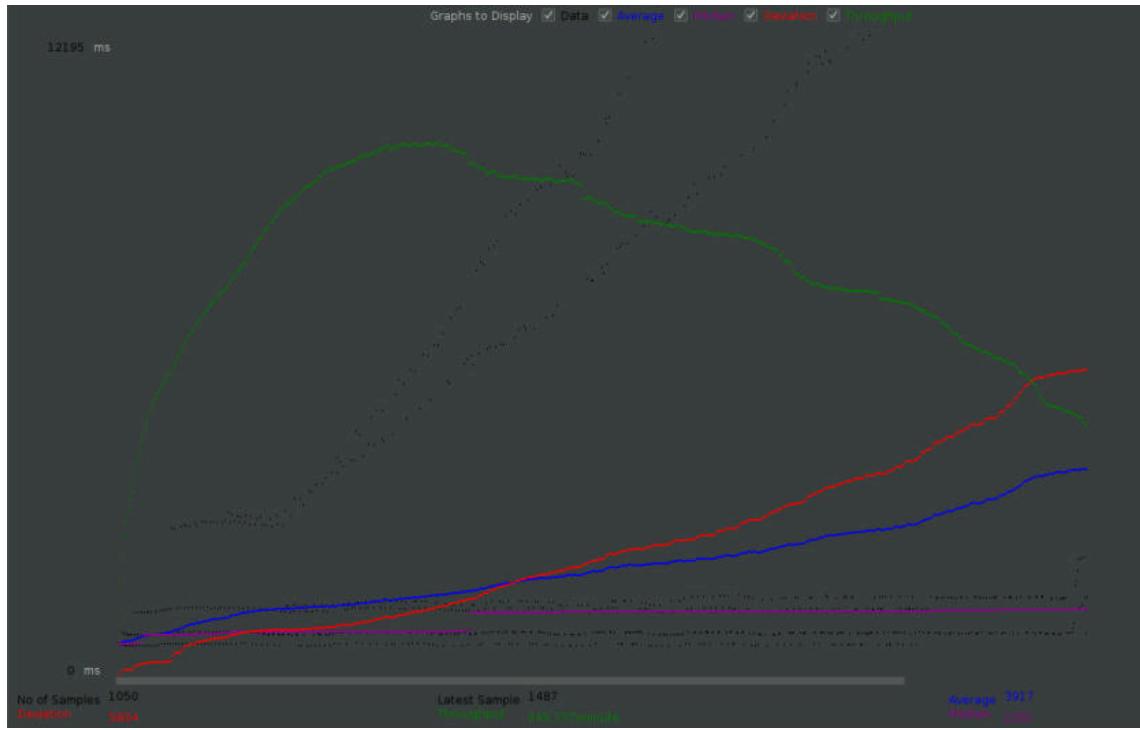


Figura 41: Desempenho durante o teste funcional com carga

Com base na Figura 40 e Figura 41, é possível compreender que todas as funcionalidades foram executadas sem falhas e que os tempos médios de resposta, 2s nas operações principais, foram aceitáveis. Portanto podemos constatar que o sistema se mostrou funcional e estável para o fluxo realista criado.

### 5.1.5. Conclusão da análise de desempenho

Por meio dos testes realizados, podemos concluir que para cenários normais e sustentados o *TastyCheck* apresenta um bom desempenho, com degradação previsível em situações extremas. A combinação da consistência do sistema ao longo do tempo com uma arquitetura com uma boa estrutura modular, garante uma fundação fiável para a utilização prática da aplicação.

## 6. Segurança da Aplicação

A segurança foi uma preocupação transversal ao desenvolvimento da aplicação TastyCheck, tendo sido aplicadas diversas medidas para proteger dados sensíveis e restringir o acesso indevido a funcionalidades críticas.

As principais práticas implementadas foram:

- **Encriptação de Passwords:** As passwords dos utilizadores são armazenadas na base de dados de forma encriptada, recorrendo a algoritmos de *hashing*, garantindo que não são guardadas em texto simples.
- **Autenticação baseada em Tokens:** Após o *login*, os utilizadores recebem um *token JWT*, que é incluído nos cabeçalhos das requisições subsequentes. Este *token* contém as permissões e a identificação do utilizador.
- **Validação de Tokens nos Endpoints:** Todos os *endpoints* protegidos verificam a validade e autenticidade do token recebido. Caso o *token* esteja ausente ou inválido, o acesso é negado automaticamente.
- **Filtro de CORS:** Foi implementado um filtro *CORS* personalizado para garantir que apenas o *frontend* autorizado (a partir do domínio/IP específico) pode comunicar com o *backend*, prevenindo acessos de origens não autorizadas.
- **Validações no Backend:** Foram também incluídas validações adicionais no *backend* para garantir que dados inválidos ou maliciosos não são persistidos ou processados incorretamente.

Estas medidas visam garantir a integridade dos dados, a confidencialidade das credenciais dos utilizadores, e o controlo rigoroso sobre o acesso à aplicação.

Embora a aplicação já integre mecanismos essenciais de segurança, a sua robustez poderia ser reforçada com a integração de ferramentas automáticas de análise de vulnerabilidades, como:

- **KICS (Keeping Infrastructure as Code Secure):** Uma ferramenta open-source da *Checkmarx* que permite detetar vulnerabilidades e más práticas em ficheiros de infraestrutura como código (por exemplo, Dockerfile, docker-compose, Terraform).
- **Semgrep:** Uma ferramenta de análise estática de código que identifica padrões inseguros e vulnerabilidades diretamente no código-fonte da aplicação.

## 7. Conclusão e Trabalho Futuro

A aplicação *Tastycheck* desenvolvida tem como objetivo ser uma plataforma interativa para partilha de experiências de utilizadores e pesquisa de restaurantes. Ao longo deste projeto, foi nos possível aplicar diversos princípios de engenharia de software, como a modelagem, prototipagem, implementação modular e deployment num ambiente controlado.

Nesta secção, serão não só expostos os principais resultados alcançados, mas também uma reflexão crítica sobre o trabalho realizado, apresentando as dificuldades sentidas e as soluções aplicadas. Para concluir, serão propostas melhorias e possíveis projetos futuros.

Com base nos testes de desempenho realizados, identificamos algumas melhorias que poderiam ser aplicadas a versões futuras da aplicação, como por exemplo a introdução de processamento assíncrono em operações mais pesada - upload de imagens, registos de avaliações, entre outros - para tentar reduzir os tempos de resposta, para aliviar a carga no backend poderíamos implementar mecanismos de cache ou até o uso de kubernetes para facilitar o escalonamento horizontal.