

# Sistema de Reserva de Restaurantes

## TechMesa



# **Projeto:** API para Sistema de Reserva e Avaliação de Restaurantes

## **Autores:**

- Camila Marques de Lima - RM: 358903
- Eduardo Bento Nakandakare - RM:359050

## **Ferramentas utilizadas**

**Event Storming:** Miro;

**Linguagem:** JAVA 17;

**Framework:** Spring Boot, Spring JPA, Hibernate, Lombok;

**Repositório:** GitHub;

**Banco de dados:** H2 DataBase;

**Publicação:** Docker, Render;

**Testes:** JUnit, Mockito;

**Dependências:** Maven;

## Sumário

Introdução.....	4
Contexto do Projeto .....	4
Objetivos do Projeto .....	4
Visão Geral do Projeto .....	5
Principais Funcionalidades.....	5
Tecnologias Utilizadas .....	6
Integração com Outros Serviços .....	7
Arquitetura do Software .....	8
Arquitetura em Camadas .....	8
Clean Architecture .....	9
Benefícios da Arquitetura .....	9
Qualidade de Software.....	9
Testes Unitários com JUnit e Mockito.....	10
Testes de Integração.....	10
Inspeção de Código .....	10
Cobertura de Testes com Coverage.....	11
Event Storming.....	12
Modelagem de Dados .....	13
Melhorias Futuras .....	15
Acesso ao Projeto.....	17
Configurando a API .....	18
Clonar o Repositório do GitHub: .....	18
Criar e Executar uma Imagem Docker:.....	18
Utilizar uma imagem Docker pública, no Registry do Docker: .....	18
Utilizar um Deploy Publicado em uma Plataforma Gratuita: .....	18
Testando a API.....	19
Considerações Finais .....	21

## Introdução

O Sistema de Reserva e Avaliação de Restaurantes – TechMesa, é uma aplicação web desenvolvida em Java, com o propósito de facilitar e otimizar o processo de reservas tanto para clientes quanto para restaurantes.

Ele busca oferecer uma plataforma simples e eficiente que permita aos usuários verificarem a disponibilidade de mesas, agendarem reservas de forma prática e receberem confirmações instantâneas. Para os restaurantes, o sistema centraliza a gestão de reservas, reduzindo erros e maximizando a organização interna.

Com o aumento do uso de dispositivos móveis e da internet, os consumidores esperam ter acesso rápido e fácil às informações sobre os restaurantes, bem como a capacidade de fazer reservas e compartilhar suas experiências online.

## Contexto do Projeto

Com o aumento da digitalização de serviços no setor de alimentação, tornou-se evidente a necessidade de ferramentas que facilitem a interação entre clientes e restaurantes. Atualmente, muitos clientes enfrentam desafios ao realizar reservas, como longas filas de espera, dificuldade para confirmar horários disponíveis ou falta de organização nos sistemas tradicionais. Além disso, a avaliação de experiências gastronômicas tem ganhado relevância, pois os feedbacks de usuários ajudam outros consumidores na tomada de decisões e auxiliam os restaurantes a melhorarem seus serviços.

Neste cenário, o projeto de um sistema de reservas e avaliação de restaurantes surge como uma solução inovadora e prática. Ele combina funcionalidades essenciais, como a reserva de mesas online, com ferramentas para avaliação de restaurantes, promovendo transparência e melhoria contínua. A proposta busca oferecer aos usuários uma experiência integrada e eficiente, ao mesmo tempo em que otimiza a gestão dos restaurantes e promove maior conexão com seus clientes.

## Objetivos do Projeto

Os principais objetivos do projeto incluem:

- Facilitar a reserva de mesas em restaurantes, oferecendo aos clientes uma maneira rápida e conveniente de encontrar e reservar um lugar para comer.

- Permitir que os clientes avaliem sua experiência em restaurantes e compartilhem feedbacks úteis para outros consumidores.
- Auxiliar os proprietários de restaurantes a gerenciar suas operações de forma mais eficiente, permitindo o acompanhamento das reservas, a análise das avaliações e o aprimoramento dos serviços.

## Visão Geral do Projeto

O Sistema de Reserva e Avaliação de Restaurantes é uma solução digital desenvolvida para atender às crescentes demandas de praticidade e eficiência no setor de restaurantes. Trata-se de uma aplicação web moderna, construída em Java e alavancada pelo framework Spring Boot, que oferece uma API robusta e flexível para gerenciar o fluxo completo de reservas.

A aplicação permite que os usuários explorem restaurantes, verifiquem a disponibilidade de mesas e realizem reservas de forma rápida e intuitiva. Além disso, o sistema incorpora funcionalidades que incentivam a interação dos clientes com os estabelecimentos, como a possibilidade de avaliar suas experiências gastronômicas e fornecer comentários detalhados, promovendo uma cultura de feedback construtivo.

Com foco na usabilidade e no desempenho, o sistema foi projetado para beneficiar tanto os clientes, ao proporcionar uma experiência integrada, quanto os restaurantes, otimizando a gestão de horários e mesas, além de criar uma conexão mais próxima com seus consumidores.

## Principais Funcionalidades

A API oferece os seguintes recursos principais:

**Cadastro de Clientes** - A API permite que clientes se registrem na plataforma, fornecendo informações como:

- Nome.
- E-mail.
- Data de cadastro.

**Cadastro de Endereços** - Oferece suporte para o registro de endereços dos clientes e restaurantes, incluindo os seguintes campos:

- Rua, número, bairro, cidade, estado, país e CEP.

**Cadastro de Restaurantes** - Permite que restaurantes sejam adicionados à plataforma, com os seguintes detalhes:

- Nome.

- Tipo de cozinha.
- Horário de funcionamento.
- Capacidade de assentos, entre outros dados relevantes.

#### **Gerenciamento de Reservas -**

- **Para os clientes:** Possibilidade de fazer reservas em restaurantes, especificando:
  - Data, horário, número de pessoas e preferências de mesa.
- **Para os restaurantes:** Visualização e confirmação de reservas pendentes.

#### **Avaliação e Comentários -**

Após a visita ao restaurante, os clientes podem:

- Avaliar sua experiência com sistema de pontuação.
- Deixar comentários detalhados sobre aspectos como comida, serviço e ambiente.

## **Tecnologias Utilizadas**

A API foi desenvolvida utilizando as seguintes tecnologias e ferramentas:

#### **Linguagem de Programação -**

- **Java:** Linguagem principal usada no desenvolvimento da aplicação, reconhecida por sua robustez e capacidade multiplataforma.

#### **Frameworks e Ferramentas -**

- **Spring Boot:** Framework utilizado para simplificar a criação de aplicações Java com o ecossistema Spring.
- **Hibernate:** Biblioteca ORM (Object-Relational Mapping) para gerenciar a persistência de dados.
- **Spring Data JPA:** Abstração que facilita o acesso e a manipulação dos dados no banco de dados.

#### **Arquitetura do Sistema -**

- **Arquitetura em Camadas:** Organização modular da aplicação em camadas (ex.: camada de apresentação, serviço, repositório, etc.), garantindo a separação de responsabilidades e facilidade de manutenção.

- **Clean Architecture:** Abordagem arquitetural que promove independência das regras de negócio em relação a detalhes técnicos, como frameworks ou bancos de dados, priorizando a testabilidade e flexibilidade do sistema.

#### **Banco de Dados -**

- **H2:** Sistema de gerenciamento de banco de dados relacional (RDBMS) leve e embutido, ideal para testes e desenvolvimento.

#### **Automação e Contêineres -**

- **Docker:** Tecnologia utilizada para criar, empacotar e implantar a aplicação em contêineres, garantindo consistência no ambiente de execução.
- **Maven:** Ferramenta para gerenciar dependências e automatizar builds.

#### **Otimização do Código -**

- **Lombok:** Framework que reduz a verbosidade do código, eliminando a necessidade de escrever métodos repetitivos como getters e setters.

#### **Qualidade do Software -**

- **JUnit:** Framework para testes unitários.
- **Mockito:** Ferramenta para mocking durante os testes, permitindo simular dependências externas.

#### **Documentação e Testes -**

- **Swagger:** Ferramenta utilizada para documentar a API e permitir testes interativos com os endpoints RESTful.

## **Integração com Outros Serviços**

A API foi projetada para oferecer alta extensibilidade, permitindo sua integração com diversos serviços externos para ampliar suas funcionalidades e proporcionar uma experiência mais rica aos usuários. Entre as possíveis integrações, destacam-se:

- **Sistemas de Pagamento:** Integração com provedores de pagamento externos para facilitar transações, como pré-pagamentos de reservas ou cobranças de taxas de cancelamento, garantindo segurança e praticidade.
- **Serviços de Geolocalização:** Uso de APIs de mapas e geolocalização para ajudar os usuários a localizar restaurantes próximos, calcular rotas e tempos estimados de chegada.

- **Sistemas de Notificação:** Integração com serviços de envio de notificações, como e-mail, SMS ou aplicativos de mensagens instantâneas, para alertar os clientes sobre confirmações de reservas, lembretes e atualizações em tempo real.

Essas integrações reforçam o propósito da API como uma solução completa e adaptável, capaz de atender às necessidades de diferentes usuários e restaurantes, além de potencializar o engajamento e a usabilidade do sistema.

## Arquitetura do Software

A arquitetura do software é uma parte fundamental do projeto, pois define a estrutura geral da aplicação e como suas diferentes partes se relacionam entre si. No caso do Sistema de Reserva e Avaliação de Restaurantes, utilizamos uma combinação de Arquitetura em Camadas e Clean Architecture para garantir uma estrutura sólida, modular e escalável.

### Arquitetura em Camadas

A Arquitetura em Camadas organiza a aplicação em módulos distintos, cada um responsável por uma parte específica do sistema, promovendo separação de responsabilidades e facilitando a manutenção. A estrutura é dividida nas seguintes camadas principais:

- **Camada de Apresentação (Presentation Layer):** Responsável por lidar com as requisições externas (ex.: HTTP) e enviar as respostas adequadas. Nesta camada, os *Controllers* atuam como ponto de entrada, intermediando a interação entre o cliente e as camadas de negócio.
- **Camada de Negócio (Business Layer):** Contém a lógica de negócio do sistema, implementada por meio de serviços que representam os casos de uso. Essa camada encapsula as operações específicas do sistema, garantindo que regras de negócio sejam centralizadas e independentes da interface do usuário.
- **Camada de Dados (Data Layer):** Gerencia o acesso e a persistência de dados no banco de dados. Os *Repositories* fornecem uma abstração para manipular as entidades de domínio, ocultando a complexidade das operações no banco.



## Clean Architecture

A arquitetura também incorpora os princípios da Clean Architecture, que promove alta independência entre as camadas do sistema e garante que as regras de negócio estejam no centro da aplicação. A implementação segue estas camadas principais:

- **Entities (Entidades):** Núcleo da aplicação, contendo as entidades de domínio, como Cliente, Endereço, Restaurante, Reserva, entre outras. Estas entidades representam os conceitos fundamentais do negócio e são completamente independentes de frameworks ou tecnologias externas.
- **Use Cases (Casos de Uso):** Representados pelos serviços da camada de negócio, encapsulam a lógica da aplicação. São responsáveis por coordenar a interação entre as entidades e os repositórios, além de implementar os fluxos de negócio.
- **Interface Adapters (Adaptadores de Interface):** Corresponde à camada de apresentação e inclui os Controllers, que gerenciam as requisições HTTP e interagem com os casos de uso para manipular as entidades.
- **Frameworks e Drivers:** Inclui detalhes técnicos, como frameworks (ex.: Spring Boot) e bibliotecas externas, além do banco de dados. Esta camada é dependente das outras, garantindo que as regras de negócio não sejam impactadas por mudanças nos frameworks.

## Benefícios da Arquitetura

A arquitetura do software baseada em Clean Architecture oferece os seguintes principais benefícios:

- **Manutenção Facilitada:** Separação clara de responsabilidades permite que alterações em uma camada não impactem as demais.
- **Testabilidade:** As regras de negócio, isoladas na camada de serviços, podem ser facilmente testadas sem dependências externas.
- **Independência de Frameworks:** A Clean Architecture garante que a lógica de negócio não dependa de frameworks, permitindo maior flexibilidade e adaptação a novas tecnologias.

## Qualidade de Software

A qualidade de software é um pilar essencial para assegurar a confiabilidade, a segurança e o desempenho superior da nossa API de Reservas e Avaliação de

Restaurantes. Para atingir e sustentar altos padrões de excelência em nosso projeto, implementamos um conjunto abrangente de práticas e ferramentas avançadas. Essas abordagens não apenas viabilizam a identificação precoce de falhas, mas também fomentam uma cultura de melhoria contínua do código, garantindo que entregamos soluções robustas e alinhadas às necessidades dos usuários.

## Testes Unitários com JUnit e Mockito

Os testes unitários são fundamentais para assegurar que cada componente da nossa API opere de forma precisa e confiável. Empregamos o framework JUnit para desenvolver e executar testes unitários de maneira automatizada, possibilitando a validação metódica do comportamento de métodos e classes individuais. Esse processo garante que cada unidade de código atenda rigorosamente aos requisitos definidos e funcione conforme o esperado.

Paralelamente, utilizamos o Mockito para criar mocks de objetos durante os testes. Essa abordagem permite isolar as unidades de código em análise, simulando com precisão o comportamento de dependências externas. Com isso, conseguimos não apenas melhorar a eficiência dos testes, mas também assegurar uma cobertura mais robusta e confiável, minimizando potenciais falhas no ambiente real.

## Testes de Integração

Os testes de integração desempenham um papel vital na validação das interações entre os diversos componentes da nossa API. Por meio deles, asseguramos que os módulos da aplicação funcionem harmoniosamente, troquem informações de maneira eficaz e gerem os resultados esperados. Para garantir a máxima precisão, esses testes são realizados em um ambiente cuidadosamente configurado para replicar as condições do ambiente de produção. Essa abordagem nos permite identificar e corrigir possíveis discrepâncias nas interações entre os componentes, promovendo uma experiência consistente e confiável para os usuários.

## Inspeção de Código

A inspeção de código é uma prática essencial para manter altos padrões de qualidade no código fonte. Por meio de revisões periódicas, identificamos e solucionamos problemas relacionados a estilo, complexidade excessiva, bugs e outros aspectos críticos que possam comprometer a eficiência e a manutenção do software. Essa atividade

colaborativa promove o compartilhamento de conhecimento, ao envolver membros da equipe de desenvolvimento na análise do código uns dos outros. Além de reforçar a qualidade do projeto, essa prática estimula a troca de ideias e o aperfeiçoamento contínuo das soluções implementadas.

## Cobertura de Testes com Coverage

A cobertura de testes é uma métrica essencial para avaliar a qualidade dos testes automatizados, indicando a porcentagem do código fonte que é exercida durante a execução dos testes. Para monitorar e analisar a cobertura em nossa API, utilizamos ferramentas especializadas, como o JaCoCo. Essas ferramentas oferecem uma visão detalhada de quais trechos de código ainda não foram testados, permitindo a identificação de lacunas que precisam ser corrigidas para garantir uma validação mais abrangente e eficaz.

O processo para gerar e visualizar os relatórios de cobertura é simples e direto. Siga os passos abaixo:

1. Execute o comando `mvn clean test` para limpar o ambiente e rodar os testes.
2. Em seguida, utilize o comando `mvn jacoco:report` para gerar os relatórios de cobertura.

Os relatórios gerados estarão localizados no diretório `target/site/jacoco/`. Eles incluem uma análise detalhada em formato HTML, apresentando a cobertura de cada classe do projeto, bem como a porcentagem de linhas de código testadas. Esses insights são fundamentais para aprimorar continuamente a qualidade e a confiabilidade do código.

techmesa

techmesa

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Cbty	Missed	Lines	Missed	Methods	Missed	Classes	
com.fao.techmesa.infrastructure.gateway	<div><div></div></div>	73%	<div><div></div></div>	34%	52	107	85	530	35	91	0	7	
com.fao.techmesa.application.usecase	<div><div></div></div>	27%	<div><div></div></div>	30%	40	54	125	196	43	49	22	26	
com.fao.techmesa.infrastructure.api	<div><div></div></div>	57%	<div><div></div></div>	0%	37	83	40	120	35	81	1	7	
com.fao.techmesa.application.usecase.exception	<div><div></div></div>	0%	<div><div></div></div>	n/a	13	13	28	28	13	13	13	13	
com.fao.techmesa.infrastructure.persistence.entity	<div><div></div></div>	0%	<div><div></div></div>	n/a	3	3	18	18	3	3	1	1	
com.fao.techmesa.application.domain.registration	<div><div></div></div>	48%	<div><div></div></div>	n/a	7	11	2	4	7	11	0	2	
com.fao.techmesa.application.domain	<div><div></div></div>	76%	<div><div></div></div>	n/a	2	7	2	7	2	7	2	7	
com.fao.techmesa.application.domain.exception	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	2	8	8	2	2	1	1	
com.fao.techmesa.application.enums	<div><div></div></div>	94%	<div><div></div></div>	n/a	6	20	6	64	6	20	0	7	
com.fao.techmesa	<div><div></div></div>	37%	<div><div></div></div>	n/a	1	2	2	3	1	2	0	1	
com.fao.techmesa.application.dto	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1	1	1	
Total	1 791 of 4 400		59%	32 of 46	30%	172	303	318	979	149	260	41	75

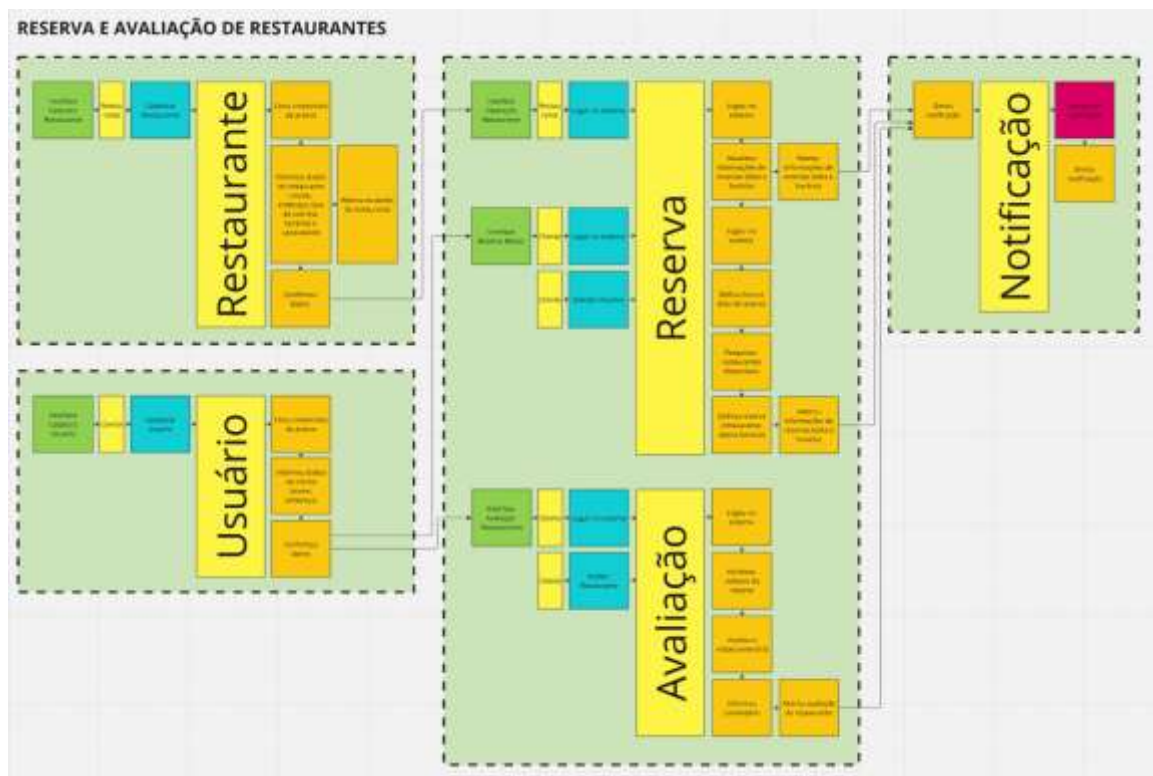
## Event Storming

O processo de *Event Storming* teve início com um brainstorming colaborativo, no qual os participantes discutiram as ideias do projeto, descrevendo os eventos e gatilhos relacionados à nossa API. A atividade foi conduzida como uma simulação de reunião com especialistas e solicitantes do negócio, promovendo um entendimento compartilhado e abrangente.

Posteriormente, organizamos elementos-chave como comandos, atores, interfaces, políticas de negócio e eventos cruciais, mapeando detalhadamente onde cada parte do processo começa e termina. Em seguida, aplicamos o objetivo principal, assim como os conceitos de agregados e contextos, segregando-os de forma estruturada para facilitar a visualização e análise.

Ao longo de cada etapa, novas ideias foram surgindo, permitindo o refinamento contínuo do processo. Dessa forma, aprimoramos tanto a compreensão quanto a eficiência do projeto. Para visualizar a fase final e o resultado obtido durante o desenvolvimento do *Event Storming*, acesse o link abaixo:

<https://miro.com/app/board/uXjVIPDJL4I=/>



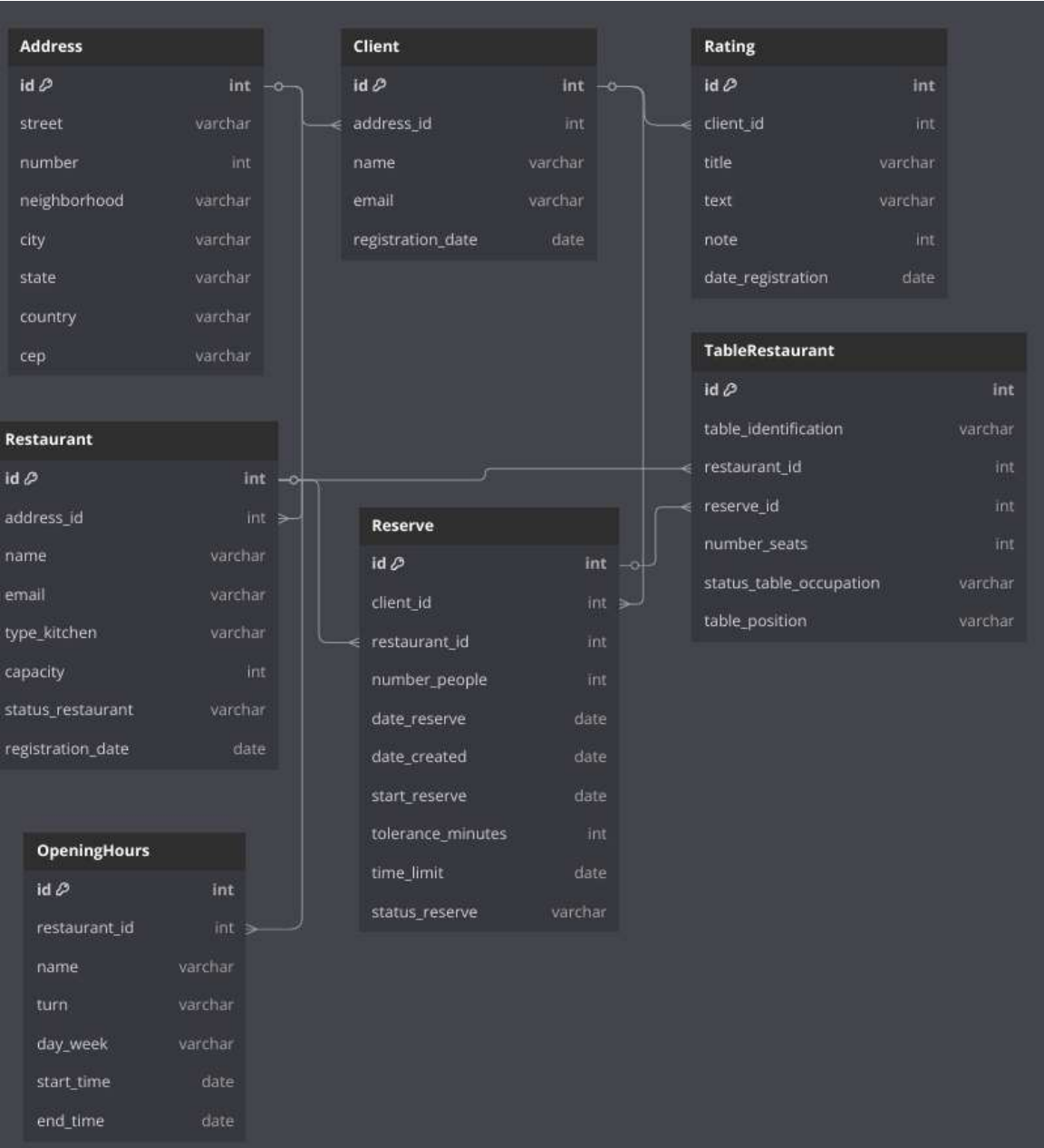
## Modelagem de Dados

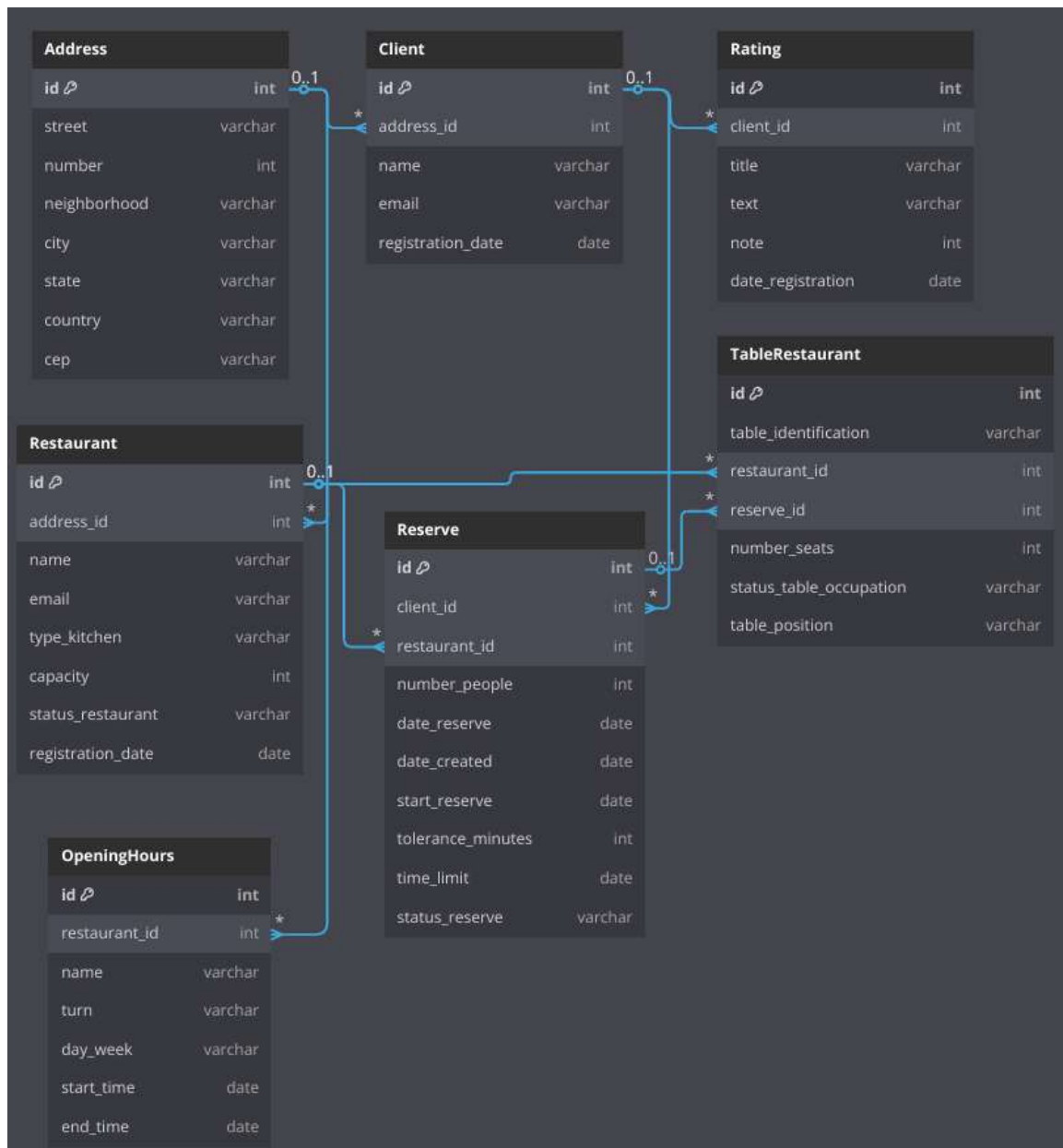
A modelagem de dados propõe a estrutura da relação entre várias entidades essenciais no contexto de um sistema para gestão de restaurantes. Estas entidades incluem **endereços, clientes, restaurantes, horários de funcionamento, avaliações, reservas e mesas de restaurante**.

1. **Endereços:** Esta entidade centraliza informações geográficas e é compartilhada entre os clientes e os restaurantes. Cada cliente e restaurante está associado a um endereço único, contendo atributos como rua, cidade, estado e código postal.
2. **Clientes:** Representa as pessoas que utilizam a plataforma para fazer reservas ou avaliar os restaurantes. A entidade de clientes mantém informações como nome, e-mail, telefone e, possivelmente, preferências pessoais.
3. **Restaurantes:** Agrupa dados relativos aos estabelecimentos, como nome, tipo de cozinha, contato, e possui uma relação direta com a entidade **endereços** para localizar cada restaurante.
4. **Horário de Funcionamento:** Relacionado aos **restaurantes**, essa entidade define os dias da semana e horários em que o restaurante está aberto. Inclui atributos como hora de abertura, hora de fechamento e especificações para feriados.
5. **Avaliações:** Esta entidade conecta **clientes** e **restaurantes** para registrar feedback. Cada avaliação contém informações como classificação (em estrelas), comentários e data da avaliação, permitindo monitorar a experiência dos clientes.
6. **Reservas:** Representa as interações entre **clientes** e **mesas de restaurante** em um restaurante específico. Inclui informações como data, hora, número de pessoas e status da reserva (confirmada, cancelada, etc.).
7. **Mesas de Restaurante:** Modela os recursos físicos disponíveis em um restaurante. Cada mesa tem atributos como capacidade de assentos e número da mesa, e está associada a um restaurante. Essa entidade se conecta às **reservas** para gerenciar disponibilidade.

A integração destas entidades garante um fluxo lógico e organizado de informações, permitindo consultar facilmente quais clientes fizeram reservas, como avaliaram os restaurantes, e verificar horários e disponibilidade. Essa abordagem também favorece a escalabilidade e consistência do sistema ao expandir o número de restaurantes ou clientes.

Abaixo segue a diagrama da modelagem de dados relacional do sistema:





## Melhorias Futuras

Após a entrega do projeto com todos os requisitos solicitados, estabelecemos um planejamento estratégico para futuras melhorias, visando aprimorar ainda mais a funcionalidade, segurança e experiência do usuário. Entre as principais melhorias propostas, destacam-se:

**Implementação de Autenticação e Autorização:** Introduzir mecanismos robustos para proteger endpoints sensíveis da API, garantindo o acesso apenas a usuários devidamente autorizados.

**Reforço na Segurança:** Adotar práticas de segurança adicionais, como proteção contra ataques de injeção SQL, XSS e CSRF, além da configuração de HTTPS para assegurar comunicações criptografadas e seguras.

**Otimização de Desempenho:** Implementar melhorias nas consultas e na indexação do banco de dados, garantindo maior eficiência e desempenho em cenários de alta carga.

**Implementação de Cache:** Aplicar técnicas de cache para armazenar dados frequentemente acessados, reduzindo a carga no banco de dados e aumentando a escalabilidade da aplicação.

**Adição de Novos Recursos:** Explorar a integração de funcionalidades adicionais, como suporte a múltiplos idiomas, integração com serviços de pagamento online para reservas pagas e recomendações personalizadas de restaurantes.

**Criação de Interface Gráfica:** Desenvolver uma interface gráfica para front-end, facilitando o uso da API em formatos como aplicações web e móveis.

**Sistema de Notificações:** Implementar notificações via SMS e e-mail, incluindo a possibilidade de cadastrar automaticamente eventos na agenda dos usuários.

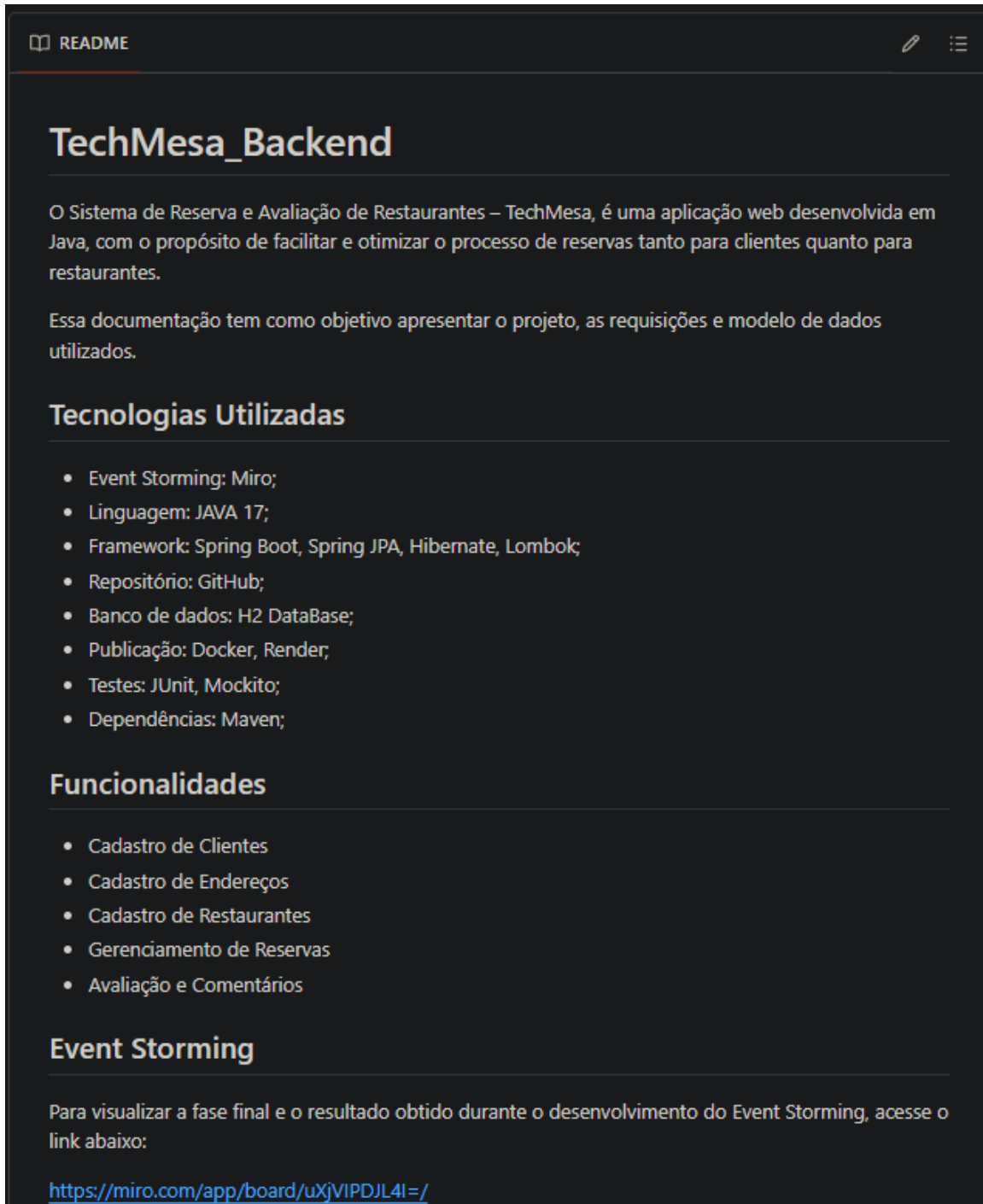
Essas iniciativas foram planejadas para atender às crescentes demandas do projeto, garantindo a evolução contínua e a entrega de valor aos usuários finais.



## Acesso ao Projeto

O projeto da API para serviço de Reserva e Avaliação com Comentários de Restaurantes, codificado seguindo o que foi definido no Objetivo do Projeto, está disponível no repositório do GitHub:

[https://github.com/CamilaLima21/TechMesa\\_Backend](https://github.com/CamilaLima21/TechMesa_Backend)



The image is a screenshot of a GitHub README file for a project named 'TechMesa\_Backend'. The interface is dark-themed. At the top, there's a header with 'README' and some icons. The main title 'TechMesa\_Backend' is prominently displayed. Below the title, there's a paragraph describing the system as a web application for restaurant reservations and reviews, developed in Java. Another paragraph states the purpose of the documentation. The 'Tecnologias Utilizadas' (Technologies Used) section lists various tools and frameworks used in the project. The 'Funcionalidades' (Features) section lists the core functionalities of the application. Finally, the 'Event Storming' section provides a link to a Miro board showing the final phase and results of the Event Storming process.

README

# TechMesa\_Backend

O Sistema de Reserva e Avaliação de Restaurantes – TechMesa, é uma aplicação web desenvolvida em Java, com o propósito de facilitar e otimizar o processo de reservas tanto para clientes quanto para restaurantes.

Essa documentação tem como objetivo apresentar o projeto, as requisições e modelo de dados utilizados.

## Tecnologias Utilizadas

- Event Storming: Miro;
- Linguagem: JAVA 17;
- Framework: Spring Boot, Spring JPA, Hibernate, Lombok;
- Repositório: GitHub;
- Banco de dados: H2 DataBase;
- Publicação: Docker, Render;
- Testes: JUnit, Mockito;
- Dependências: Maven;

## Funcionalidades

- Cadastro de Clientes
- Cadastro de Endereços
- Cadastro de Restaurantes
- Gerenciamento de Reservas
- Avaliação e Comentários

## Event Storming

Para visualizar a fase final e o resultado obtido durante o desenvolvimento do Event Storming, acesse o link abaixo:

<https://miro.com/app/board/uXjVIPDJL4I=/>

## Configurando a API

### Clonar o Repositório do GitHub:

Para configurar a API, basta clonar o projeto:

[https://github.com/CamilaLima21/TechMesa\\_Backend.git](https://github.com/CamilaLima21/TechMesa_Backend.git)

### Criar e Executar uma Imagem Docker:

- No terminal, na raiz do projeto, execute o seguinte comando para construir a imagem Docker:  
**`docker build --tag techmesa .`**
- Após a construção da imagem, execute um contêiner Docker com o seguinte comando:  
**`docker run --name techmesadocker -p 8080:8080 techmesa`**
- É possível visualizar o Swagger da aplicação neste endereço:  
<http://localhost:8080/swagger-ui/index.html#/>

### Utilizar uma imagem Docker pública, no Registry do Docker:

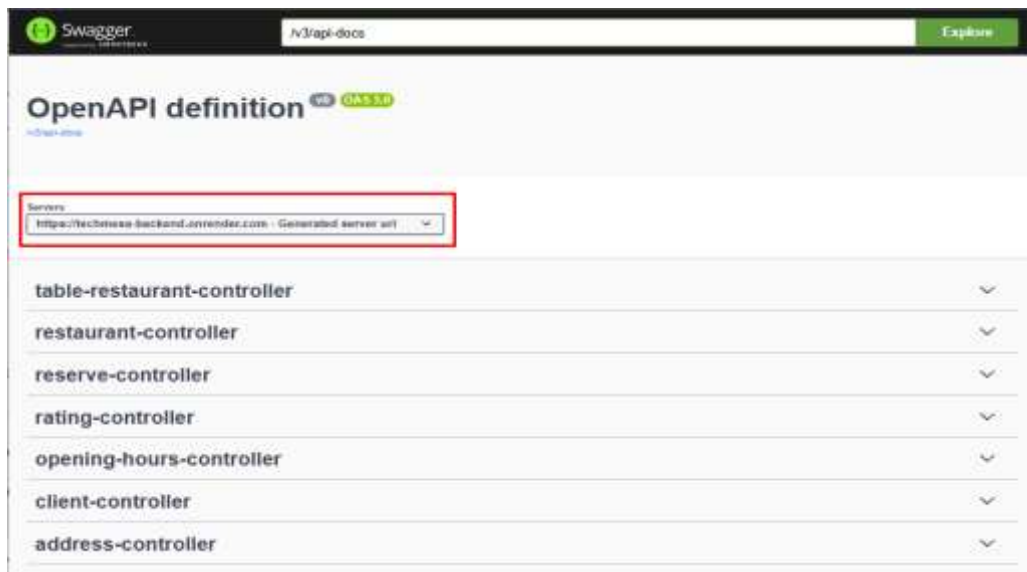
É possível acessar a imagem pública através deste endereço:

[[docker.io/camiladck23/techmesa](https://hub.docker.io/camiladck23/techmesa)]

### Utilizar um Deploy Publicado em uma Plataforma Gratuita:

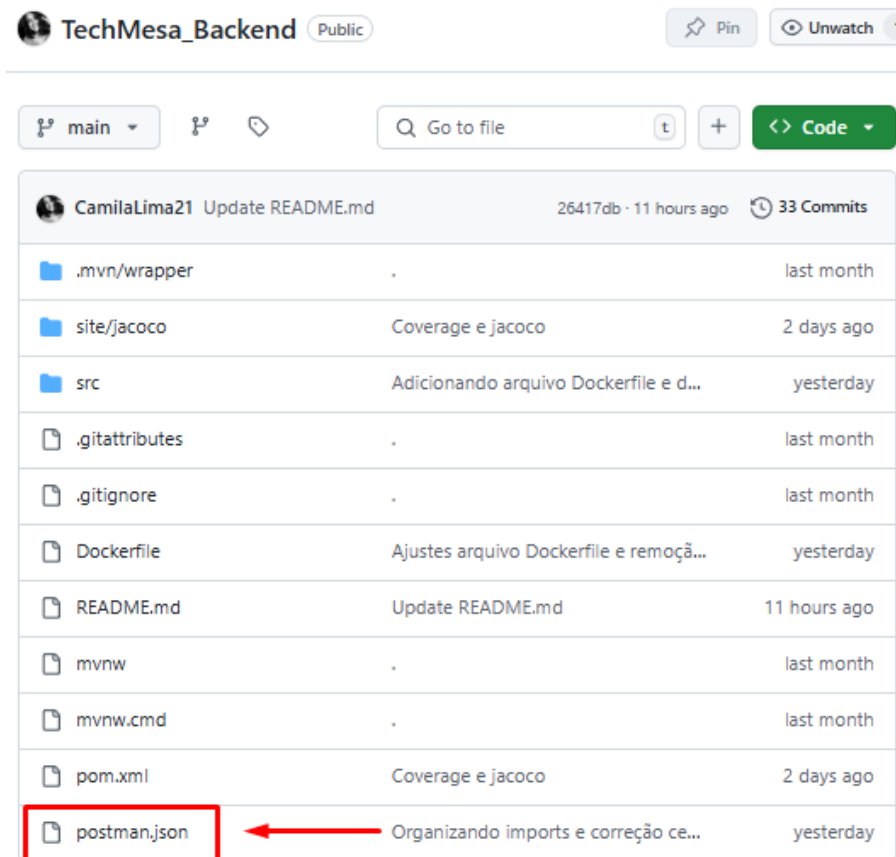
Acessar a API pública através da URL abaixo:

- <https://techmesa-backend.onrender.com/swagger-ui/index.html#/>

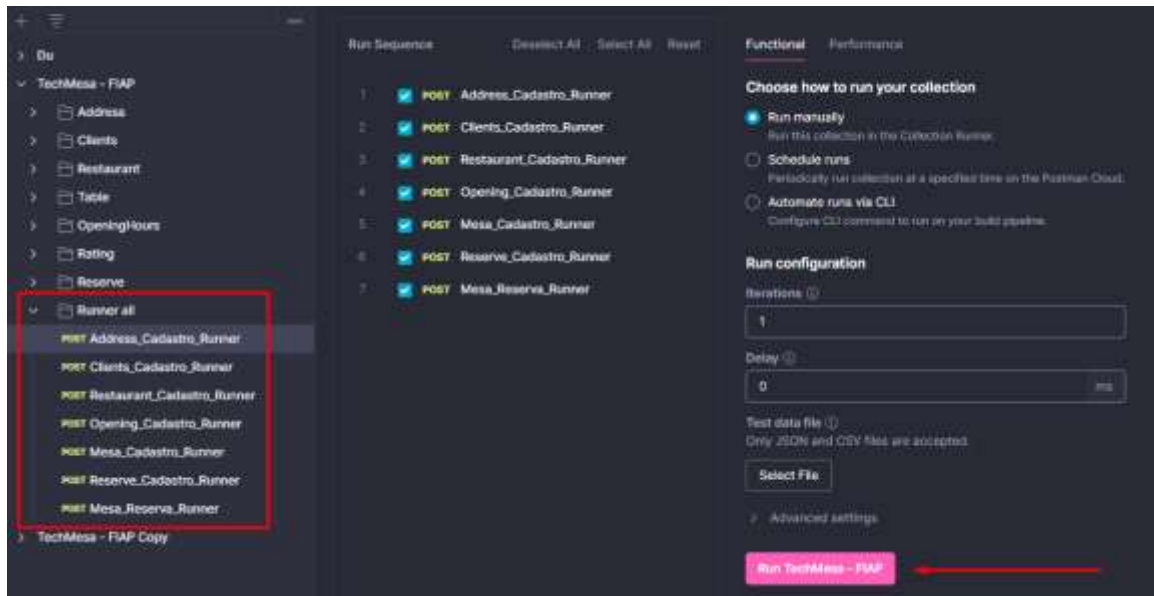


## Testando a API

Para testar nossa API, disponibilizamos os payloads do tipo Json para o envio de requisições via Postman ou outra ferramenta semelhante. O arquivo postman.json se encontra raiz do projeto:



No postman, dentro da collection, é possível executar a pasta Runner all com todas as requisições para criação das entidades do projeto já na sequência correta para evitar quebras nos fluxos:



Após executar a criação das entidades, é possível testar as demais requisições aos endpoints da aplicação para consultas, edições e deleções.

## Considerações Finais

O desenvolvimento da API de Reservas e Avaliação de Restaurantes foi uma jornada enriquecedora, marcada pela criação de uma solução robusta, eficiente e alinhada aos mais altos padrões de qualidade. Desde o início, priorizamos uma arquitetura limpa, baseada nos princípios da *Clean Architecture*, que nos permitiu desenvolver um código modular, escalável e de fácil manutenção.

Ao longo do projeto, utilizamos diversas tecnologias e ferramentas, como Spring, JPA, Hibernate e Mockito, combinadas às melhores práticas de engenharia de software, incluindo testes unitários, testes de integração e análise de cobertura de testes. Os testes desempenharam um papel central no nosso processo, com ferramentas como JUnit e Mockito assegurando a confiabilidade e qualidade da aplicação. Além disso, a integração com Docker trouxe praticidade na distribuição e execução da API em diferentes ambientes, tanto de desenvolvimento quanto de produção.

Mais do que uma aplicação prática de conceitos teóricos, essa experiência representou uma oportunidade de aprendizado contínuo e colaboração em equipe. A troca de ideias, a superação de desafios e o trabalho conjunto fortaleceram nosso conhecimento e habilidades em engenharia de software, proporcionando crescimento profissional e pessoal.

Encerramos este projeto com a consciência de que sempre há espaço para aprimoramentos, mas também com orgulho pelas conquistas alcançadas. Estamos confiantes de que a API desenvolvida oferece uma base sólida para futuras evoluções e ampliações.

Por fim, expressamos nossa gratidão à FIAP por proporcionar esta oportunidade de aprendizado, aos professores pelo suporte e orientação constantes, e aos nossos colegas pela colaboração ao longo desta jornada. Estamos entusiasmados para aplicar tudo o que aprendemos neste projeto em futuros desafios no campo do desenvolvimento de software.