**TECH CHALLENGE – FIAP**

**Projeto**: API para serviço de Reserva e Avaliação com Comentários de Restaurantes;

**Curso**: Arquitetura e Desenvolvimento em JAVA – Fase 3;

**Escola**: FIAP;

**Grupo**:

Victor Luiz Montibeller;

Leonardo Arantes Di Nizo;

Alexandre Marques;

**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;

# Introdução

O Sistema de Reserva e Avaliação de Restaurantes é uma aplicação web desenvolvida em Java, projetada para atender às necessidades dos clientes e proprietários de restaurantes em um mundo cada vez mais digitalizado. 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

Nos dias de hoje, muitos consumidores confiam em plataformas online para encontrar e avaliar restaurantes antes de fazer uma reserva. Isso inclui a leitura de comentários de outros clientes, visualização de fotos do ambiente e do cardápio, e verificação da disponibilidade de mesas em tempo real. Por outro lado, os proprietários de restaurantes buscam maneiras de atrair mais clientes e melhorar a experiência dos que já frequentam seus estabelecimentos.

## Importância da API

A API do Sistema de Reserva e Avaliação de Restaurantes visa preencher essa lacuna, proporcionando uma plataforma centralizada onde clientes e restaurantes podem interagir de forma eficiente e conveniente. Ao oferecer recursos como reserva de mesas online, avaliações e comentários, a API ajuda a melhorar a experiência do usuário e a promover a transparência e a confiança entre os consumidores e os estabelecimentos gastronômicos.

## 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 aplicação web desenvolvida em Java, utilizando o framework Spring Boot. A API foi projetada para fornecer funcionalidades abrangentes para gerenciar reservas em restaurantes, permitindo aos usuários avaliar suas experiências e deixar comentários sobre os estabelecimentos visitados.

## Funcionalidades Principais

A API oferece os seguintes recursos principais:

* Cadastro de Clientes: Permite que os clientes se cadastrem na plataforma fornecendo informações como nome, e-mail e data de cadastro.
* Cadastro de Endereços: Permite o cadastro de endereços dos clientes e dos restaurantes, incluindo informações como rua, número, bairro, cidade, estado, país e CEP.
* Cadastro de Restaurantes: Permite que os restaurantes sejam registrados na plataforma, fornecendo detalhes como nome, tipo de cozinha, horário de funcionamento, capacidade de assentos, etc.
* Gerenciamento de Reservas: Os clientes podem fazer reservas em restaurantes disponíveis, especificando a data, hora, número de pessoas e preferências de mesa. Os restaurantes podem visualizar e confirmar as reservas pendentes.
* Avaliação e Comentários: Após a visita ao restaurante, os clientes têm a oportunidade de avaliar sua experiência e deixar comentários sobre a comida, serviço, ambiente, etc.

## Tecnologias Utilizadas

A API foi desenvolvida utilizando as seguintes tecnologias e ferramentas:

* Java: Linguagem de programação principal para o desenvolvimento da aplicação.
* Spring Boot: Framework Java usado para criar aplicativos baseados em Spring com facilidade.
* Hibernate: Biblioteca ORM para mapeamento objeto-relacional e persistência de dados.
* Docker: Conjunto de produtos de plataforma como serviço (PaaS) que usam virtualização de nível de sistema operacional para entregar software em pacotes chamados contêineres.
* Lombok: Framework para Java que permite escrever código eliminando a verbosidade, o que permite ganhar tempo de desenvolvimento para o que realmente é importante.
* H2: Sistema de gerenciamento de banco de dados relacional (RDBMS) escrito em Java.
* Spring Data JPA: Implementação da camada de acesso a dados baseada em JPA.
* Spring MVC: Padrão de projeto usado para desenvolver aplicativos web em Java.
* Maven: Ferramenta desenvolvida pela Apache, ela serve para gerenciar as dependências e automatizar seus builds.
* JUnit e Mockito: Frameworks de teste unitário e mocking para garantir a qualidade do software.
* Swagger: Ferramenta para projetar, construir, documentar e consumir serviços da Web RESTful.

## Integração com Outros Serviços

A API pode ser integrada com outros serviços externos, como sistemas de pagamento, serviços de geolocalização, sistemas de notificação etc., para estender suas funcionalidades e melhorar a experiência do usuário.

# 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 dos padrões MVC e Clean Architecture para garantir uma arquitetura sólida e escalável.

## Padrão MVC (Model-View-Controller)

O padrão MVC é amplamente utilizado em aplicações web para separar as preocupações de apresentação, lógica de negócio e acesso a dados. Nossa implementação do MVC no projeto é a seguinte:

* Model (Modelo): Representa a camada de negócios da aplicação, incluindo entidades como Cliente, Endereço, Restaurante, etc. Essas entidades encapsulam os dados da aplicação e a lógica de negócio associada a eles.
* View (Visão): Responsável pela apresentação dos dados aos usuários. No contexto da API, a View é representada pelos endpoints RESTful que expõem os recursos da aplicação aos clientes.
* Controller (Controlador): Atua como intermediário entre a View e o Modelo, recebendo as requisições dos clientes, chamando a lógica de negócio apropriada e retornando as respostas correspondentes. Os controllers são responsáveis por rotear as requisições para as ações corretas e garantir uma interação suave entre a camada de apresentação e a camada de negócio.

## Clean Architecture

A arquitetura do software foi projetada com base nos princípios da Clean Architecture, que enfatiza a separação de preocupações e a independência de frameworks externos. Nossa implementação da Clean Architecture inclui as seguintes camadas:

* A camada de Entities é o núcleo da aplicação e contém as entidades de domínio que representam os conceitos fundamentais do negócio. As entidades encapsulam os dados e o comportamento relacionados a esses conceitos e são independentes de qualquer framework ou tecnologia externa. Exemplos de entidades incluem Cliente, Endereço, Restaurante, Reserva, etc.
* A camada de Controllers é responsável por lidar com as requisições HTTP, roteando-as para os serviços apropriados e retornando as respostas correspondentes aos clientes. Os controllers servem como a interface entre a aplicação e o mundo exterior e são responsáveis por receber, validar e encaminhar as requisições para os serviços adequados. Eles são implementados de forma independente dos detalhes de implementação dos serviços e das entidades de domínio.
* A camada de Services contém a lógica de negócio da aplicação, representada por serviços que implementam os casos de uso específicos do sistema. Os serviços encapsulam as operações que podem ser realizadas na aplicação e são responsáveis por coordenar a interação entre as entidades de domínio e os repositórios de dados. Eles são independentes dos detalhes de implementação dos controllers e dos repositórios de dados.
* A camada de Repository é responsável pelo acesso e persistência dos dados no banco de dados. Os repositórios fornecem uma abstração sobre o armazenamento de dados e permitem que os serviços acessem e manipulem as entidades de domínio de forma transparente. Eles encapsulam as operações de consulta e persistência e são implementados de forma independente dos detalhes de implementação dos serviços e das entidades de domínio.

## Benefícios da Arquitetura

A arquitetura do software baseada nos princípios da Clean Architecture oferece os seguintes benefícios:

* Separação de Preocupações: As responsabilidades de apresentação, lógica de negócio e acesso a dados são claramente separadas, facilitando a manutenção e a evolução da aplicação:
* Independência de Tecnologia: As camadas internas da aplicação são independentes de frameworks externos e detalhes de implementação, permitindo uma maior flexibilidade e adaptabilidade à medida que os requisitos do sistema mudam.
* Testabilidade: A arquitetura limpa facilita a escrita de testes automatizados, pois as diferentes partes do sistema podem ser testadas de forma isolada e sem depender de infraestrutura externa.

# Qualidade de Software

A qualidade de software desempenha um papel fundamental na confiabilidade, segurança e eficácia da nossa API de Reservas e Avaliação de Restaurantes. Para garantir altos padrões de qualidade em nosso projeto, adotamos diversas práticas e ferramentas que promovem a detecção precoce de defeitos e a melhoria contínua do código.

## Testes Unitários com JUnit e Mockito

Os testes unitários são essenciais para garantir que cada componente da nossa API funcione conforme o esperado. Utilizamos o framework JUnit para escrever e executar testes unitários de forma automatizada. Isso nos permite verificar o comportamento de métodos e classes individuais, garantindo que cada unidade de código atenda aos requisitos especificados.

Além disso, empregamos o Mockito para criar mocks de objetos durante os testes unitários. Essa prática nos permite isolar as unidades de código em teste e simular o comportamento de dependências externas, aumentando a eficiência e a confiabilidade dos testes.

## Testes de Integração

Os testes de integração são cruciais para validar a interação entre os diferentes componentes da nossa API. Implementamos testes de integração para garantir que os diversos módulos da aplicação se integrem corretamente e produzam o resultado esperado. Esses testes são executados em um ambiente que replica o ambiente de produção, garantindo uma validação realista das interações entre os componentes.

## Inspeção de Código

A inspeção de código é uma prática indispensável para identificar e corrigir potenciais problemas de qualidade no código fonte. Realizamos revisões de código periódicas para identificar e corrigir problemas de estilo, complexidade excessiva, bugs e outras questões de qualidade. Essa atividade colaborativa envolve membros da equipe de desenvolvimento que revisam o código uns dos outros em busca de possíveis melhorias e correções.

## Cobertura de Testes com Coverage

A cobertura de testes é uma métrica importante que indica a porcentagem do código fonte que é exercida por testes automatizados. Utilizamos ferramentas de análise de cobertura, como JaCoCo, para avaliar a cobertura de testes da nossa API. Essas ferramentas nos fornecem insights sobre quais partes do código não estão sendo testadas adequadamente, ajudando a identificar lacunas na cobertura de testes que precisam ser abordadas.

Para executar os testes com cobertura, execute os comandos abaixo:

* **mvn clean test;**
* **mvn jacoco:report;**

Dentro do diretório target/site/jacoco/, você encontrará os relatórios HTML detalhando a cobertura de cada classe em seu projeto. Esses relatórios mostram a porcentagem de linhas de código cobertas por testes.

Tabela

Descrição gerada automaticamente

# Event Storming

Iniciando com um brainstorming, discutindo as ideias do projeto entre os participantes do grupo que descrevessem os eventos e gatilhos da nossa API, simulando uma reunião com os especialistas e solicitantes do negócio.

Organizamos os comandos, atores, interfaces, políticas do negócio e eventos pivotais, identificando onde inicia e onde termina cada parte do processo.

Em seguida aplicamos o objetivo principal, agregados e contextos, separando-os para melhor visualização.

Em cada fase, novas ideias foram surgindo e assim fomos refinando o processo ao longo de sua construção. Todo esse processo de desenvolvimento do Event Storming e suas fases, podem ser visualizados através do link abaixo:

# [**https://miro.com/app/board/uXjVNqrQfdY=/**](https://miro.com/app/board/uXjVNqrQfdY=/)

Gráfico, Gráfico de mapa de árvore

Descrição gerada automaticamente

# Melhorias Futuras

Após a entrega do projeto funcionando com os requisitos solicitados, deixamos o planejamento para algumas melhorias futuras, como:

* Implementação de Autenticação e Autorização: Adicionar autenticação e autorização para proteger endpoints sensíveis da API, garantindo que apenas usuários autorizados possam acessá-los.
* Melhorias na Segurança: Implementar práticas de segurança adicionais, como proteção contra-ataques de injeção SQL, XSS e CSRF, além de configurar HTTPS para comunicação segura.
* Melhorias de Desempenho: Realizar otimizações de consulta e indexação no banco de dados para melhorar o desempenho da API, especialmente em cenários de alta carga.
* Implementação de Cache: Utilizar técnicas de cache para armazenar em cache dados frequentemente acessados e reduzir a carga no banco de dados, melhorando a escalabilidade e o desempenho geral da aplicação.
* Aprimoramento da Documentação da API: Refinar e expandir a documentação da API para torná-la mais abrangente e amigável para os desenvolvedores, incluindo exemplos de solicitações e respostas.
* Implementação de Testes de Integração Completa: Desenvolver testes de integração completos que cobrem todos os casos de uso principais da aplicação, garantindo uma cobertura de teste abrangente. problemas em tempo real e tomar medidas corretivas proativas.
* Adição de Recursos Adicionais: Considerar a adição de novos recursos, como suporte a múltiplos idiomas, integração com serviços de pagamento online para reservas pagas, ou funcionalidades de recomendação personalizada de restaurantes.
* Interface gráfica: Front-End para integração com a API, possibilitando uma melhor adaptação aos processos, como um app WEB e versão Móbile.
* Notificações: Enviar lembretes de notificações por SMS e e-mail, com a possibilidade de cadastrar o evento automaticamente na agenda dos usuários.

# 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/victormontibeller/ReservasRestaurantes>

# Configurando a API

## Clonar o Repositório do GitHub:

Para configurar a API, basta clonar o projeto: <https://github.com/victormontibeller/ReservasRestaurantes>

## Criar e Executar uma Imagem Docker:

* No terminal, construa a imagem Docker usando o seguinte comando:
  + **docker build -t image\_restaurantes .**
* Após a construção da imagem, execute um contêiner Docker com o seguinte comando:
  + **docker run -p 8080:8080 image\_restaurantes**
* Abrir a URL abaixo:
  + <http://localhost:8080>

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

Acessar a imagem através da URL abaixo:

* Docker.io/victormontibeller/image\_restaurantes:latest;

## Utilizar um Deploy Publicado em uma Plataforma Gratuita:

Acessar a API pública atráves da URL abaixo:

* <https://image-restaurantes-latest.onrender.com/>

Texto

Descrição gerada automaticamente

# Testando a API

Para testar nossa API, temos alguns exemplos de payloads do tipo Json para o envio de requisições via Postman ou outra ferramenta semelhante. Esses payloads se encontram no projeto.

Necessário seguir o fluxo para que a aplicação tenha os dados necessários para gerar corretamente as Reservas.

## Endereços:

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Clientes**:**

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

## Restaurantes**:**

## Texto Descrição gerada automaticamente

## Texto Descrição gerada automaticamente

## Texto Descrição gerada automaticamente

## Comentários**:**

## Interface gráfica do usuário, Texto Descrição gerada automaticamente

## Texto Descrição gerada automaticamente

## Texto Descrição gerada automaticamente

# Considerações Finais

O desenvolvimento da API de Reservas e Avaliação de Restaurantes representou uma jornada empolgante em direção à construção de uma solução robusta e de alta qualidade. Durante este projeto, demos destaque à arquitetura limpa, seguindo os princípios da Clean Architecture, que nos permitiram criar um código modular, escalável e de fácil manutenção.

Durante o desenvolvimento da API, exploramos diversas tecnologias e ferramentas, incluindo Spring, JPA, Hibernate e Mockito, e aplicamos práticas de engenharia de software, como testes unitários, testes de integração e análise de cobertura de testes.

Os testes desempenharam um papel crucial em nosso processo de desenvolvimento, com ênfase em testes unitários e de integração. Utilizamos ferramentas como JUnit e Mockito para garantir a qualidade e confiabilidade de nossa aplicação. Além disso, a integração com Docker facilitou a distribuição e execução da API em diferentes ambientes de desenvolvimento e produção.

Nossa jornada não foi apenas uma oportunidade de aplicar conceitos teóricos, mas também um aprendizado contínuo e colaborativo. A troca de conhecimentos e a resolução de desafios em equipe fortaleceram nossa compreensão e habilidades em engenharia de software.

À medida que concluímos este projeto, reconhecemos que sempre há espaço para melhorias e refinamentos. No entanto, estamos orgulhosos do que alcançamos e confiantes de que nossa API oferece uma base sólida para futuras iterações e extensões.

Agradecemos à FIAP por nos proporcionar esta oportunidade de aprendizado e crescimento, e aos nossos colegas e professores por seu apoio e orientação ao longo do curso. Estamos ansiosos para aplicar os conhecimentos adquiridos neste projeto em nossas jornadas futuras no desenvolvimento de software.