

**FIAP – FACULDADE DE INFORMÁTICA E
ADMINISTRAÇÃO PAULISTA**

TECH CHALLENGE – FASE 3

Integrantes do Grupo 59

Marcelo Akio Nishimori
mnishimori@yahoo.com.br
RM-350802

2024

O projeto foi desenvolvido com a utilização das ferramentas ClickUp e GitHub. Seguem os links para acesso:

- GitHub do serviço de reserva de mesa em restaurante:

[fiapAluraTechChallengeFase03](#)

<https://github.com/EvolutionTeamFiapAluraPostech/fiapAluraTechChallengeFase03>

- A ferramenta ClickUp foi utilizada para documentar o projeto, substituindo o Notion, pois minha licença free expirou.

Observação: a documentação foi produzida sobre um template do Notion/ClickUp, mas o conteúdo é atualizado para cada fase do tech challenge.

Evolution Team

Private (<https://app.clickup.com/9013160904/docs/8ckkuy8-413/8ckkuy8-473>)

Private (<https://app.clickup.com/9013160904/docs/8ckkuy8-413/8ckkuy8-493>)

Private (<https://app.clickup.com/9013160904/docs/8ckkuy8-413/8ckkuy8-513>)

Private (<https://app.clickup.com/9013160904/docs/8ckkuy8-413/8ckkuy8-533>)

Private (<https://app.clickup.com/9013160904/docs/8ckkuy8-413/8ckkuy8-553>)

Storytelling

Objetivo

Esta página da documentação tem como objetivo informar dados do time e da solução, com um conceito voltado ao produto, ou seja, explicar as funcionalidades do aplicativo, sem entrar nos detalhes técnicos mais profundos.

A Evolution

A Evolution Team é uma empresa responsável por fornecer uma solução para a reserva de mesas em restaurantes, bem como sua avaliação.

Solução proposta

O projeto **Fiap Restaurant** foi criado para ajudar as pessoas a encontrarem um restaurante de seu interesse, conforme o tipo de cozinha, localização ou nome e realizar uma reserva.

Fluxo principal de utilização do aplicativo

1. O usuário se cadastra na aplicação. É necessário informar o nome completo, e-mail, cpf e senha. Será permitido um cadastro por CPF.
2. Pesquisa e seleciona o restaurante desejado. A pesquisa pode ser feita pelo nome do restaurante, tipo de cozinha e/ou localização.
3. Realiza a reserva de uma mesa disponível no restaurante desejado, sendo que cada mesa comporta 4 pessoas e deve-se respeitar a capacidade de pessoas do restaurante. Após a reserva ser confirmada, o usuário receberá um e-mail de confirmação da reserva, com os respectivos dados cadastrados.
4. O usuário pode realizar a avaliação do restaurante, caso lhe interesse.
5. A aplicação permitirá o cadastro de novos restaurantes e manutenção dos restaurantes já cadastrados.
6. A aplicação permitirá o cadastro e o gerenciamento das reservas do restaurante.

Design da solução

O projeto está dividido em uma aplicação backend Java Spring Boot, com imagem docker construída e com um banco de dados Postgresql. A aplicação backend e o banco de dados estão na cloud free Render (<https://render.com/>).

O backend foi implementado seguindo as recomendações da Clean Architecture, com Clean Code, SOLID e testes automatizados de unidade e integração, seguindo os princípios do FIRST e Clean Tests.

Principais endpoints:

- <https://fiaprestaurant-tc03.onrender.com/authenticate>
- <https://fiaprestaurant-tc03.onrender.com/users>
- <https://fiaprestaurant-tc03.onrender.com/restaurants>

1 - Serviço para o gerenciamento de restaurantes/mesas.

Autenticação do usuário na aplicação

A autenticação do usuário na aplicação será realizada por credencial de usuário e senha, onde o usuário é equivalente a seu e-mail. A senha deve possuir no mínimo 8 e no máximo 20 caracteres, sendo no mínimo um número, uma letra minúscula, uma letra maiúscula e um caractere especial (@#\$%^&+=).

O e-mail do usuário é único na base de dados, não permitindo duplicação.

Observação

Ao iniciar a aplicação, foi criada uma flyway migration para cadastrar um usuário inicial para utilização/auxilio na avaliação dos professores.

Endpoint:

POST /authenticate HTTP/1.1

Host: <https://fiaprestaurant-tc03.onrender.com>

Content-Type: application/json

Content-Length: 76

Usuário com credenciais:

```
{  
  "email": "thomas.anderson@itcompany.com",  
  "password": "@Bcd1234"  
}
```

Cadastros da aplicação

Para a utilização do aplicativo, são necessários os cadastros seguintes:

- Cadastro do usuário do aplicativo;
- Cadastro do restaurante;

Cadastro do usuário do aplicativo

Para realizar o cadastro de um novo usuário no aplicativo são necessárias as seguintes informações:

- Nome (obrigatório, com no máximo 500 caracteres).
- e-mail (obrigatório, com no máximo 500 caracteres e respeitando a regra de composição do endereço de e-mail - nome de usuário @ domínio).

- CPF (obrigatório, com 11 caracteres e respeitando a regra do número do CPF).
- Senha (obrigatório, com no mínimo 8 e máximo 20 caracteres, sendo uma letra minúscula, uma maiúscula, um número e um caractere especial [`@#$%^&+=`]).

A API de usuários permitirá as operações de cadastro de usuários, pesquisa de todos os usuários, pesquisa de usuários por ID, email ou CPF, atualização de dados já cadastrados e exclusão (lógica de usuários).

Para obter mais informações sobre a documentação da API de restaurantes, acesse o link <https://fiaprestaurant-tc03.onrender.com/swagger-ui/index.html#/UsersApi>

Cadastro dos restaurantes do aplicativo

Para realizar o cadastro de um novo restaurante são necessárias as seguintes informações:

- Nome (obrigatório, com no máximo 500 caracteres).
- CNPJ (obrigatório, com 14 caracteres e respeitando a regra do número do CNPJ).
- Tipo de cozinha (obrigatório, com no mínimo 3 e máximo 50 caracteres).
- Coordenadas da localização do restaurante, sendo latitude e longitude.
- Endereço (rua, número, bairro, cidade, Estado, CEP).
- Horário de funcionamento (hora de abertura e fechamento).
- Capacidade total de pessoas a serem servidas no restaurante, considerando que são 4 pessoas por mesa.

A API de restaurantes permitirá as operações de cadastro de restaurantes, pesquisa de restaurante (por ID, por tipo de cozinha, por nome e coordenadas), atualizar dados do restaurante e excluir um restaurante (que não possua vínculo a nenhuma reserva ou avaliação).

Para obter mais informações sobre a documentação da API de restaurantes, acesse o link <https://fiaprestaurant-tc03.onrender.com/swagger-ui/index.html#/RestaurantsApi>

Operações da aplicação

O aplicativo oferece as seguintes funcionalidades:

- Reserva de mesa em restaurante;
- Avaliação do restaurante;

Reserva de mesa em restaurante

Para realizar a reserva de uma mesa em um restaurante no aplicativo são necessárias as seguintes informações:

- ID do restaurante (obrigatório, UUID).

- ID do usuário do aplicativo e interessado na reserva de uma mesa no restaurante (obrigatório, UUID).
- Observação (opcional, descrição da observação sobre a reserva, com no mínimo 3 e máximo 500 caracteres).
- Data da reserva (obrigatório, data futura, superior a data e hora atual).

A reserva estará condicionada a capacidade do restaurante e as reservas já realizadas para a determinada data e horário.

Se a reserva for aceita, o usuário solicitante receberá um e-mail de confirmação da reserva, com os dados cadastrados.

Haverá a possibilidade de cancelamento da reserva e o recebimento da confirmação do cancelamento.

As reservas utilizadas poderão ser fechadas.

Para obter mais informações sobre a documentação da API de reservas de restaurantes, acesse o link <https://fiaprestaurant-tc03.onrender.com/swagger-ui/index.html#/RestaurantsBookingApi>

Avaliação de restaurante

Para realizar a avaliação um restaurante no aplicativo são necessárias as seguintes informações:

- ID do restaurante (obrigatório, UUID).
- ID do usuário do aplicativo e interessado na reserva de uma mesa no restaurante (obrigatório, UUID).
- Descrição (obrigatória, descrição da avaliação sobre o restaurante, com no mínimo 3 e máximo 100 caracteres).
- Nota (obrigatório, data futura, superior a data e hora atual).

A API de avaliação de restaurante permitirá realizar a consulta dos reviews por restaurante.

Para obter mais informações sobre a documentação da API de avaliação de restaurantes, acesse o link <https://fiaprestaurant-tc03.onrender.com/swagger-ui/index.html#/RestaurantsReviewsApi>

Link do projeto no Github

<https://github.com/EvolutionTeamFiapAluraPostech/fiapAluraTechChallengeFase03>

Ferramentas

Para a construção do projeto, selecionamos diversas ferramentas, onde cada uma será citada com seu respectivo objetivo:

1. Clickup <https://clickup.com/> - esta ferramenta foi selecionada para concentrarmos todas as informações do projeto, para obtermos o controle de tudo o que estamos fazendo, para que

possamos compartilhar entre os membros da equipe e manter tudo atualizado, com objetivo de nos organizar e tornar o projeto sustentável e escalável. Nesta ferramenta temos as seções:

- a. Storytelling - contém a história/motivação, conceitos, diagramas, fluxo do projeto.
 - b. Onboarding - contém o link do projeto no GitHub e orientações de como rodar o projeto.
 - c. Glossary - contém o dicionário de termos utilizados no projeto para consolidação da linguagem ubíqua.
 - d. Design Architecture, standards and conventions - contém o design da arquitetura, padrões e convenções de código. Conceitos técnicos que são aplicados no projeto. Convenções de código.
 - e. Staff - contém os integrantes do projeto.
2. Miro (miro.com) - ferramenta utilizada para desenhar os diagramas.
 3. [Draw.io](https://draw.io) - ferramenta utilizada para desenhar fluxos e o diagrama de entidade e relacionamento.
 4. Discord - ferramenta utilizada para realizar reuniões síncronas e assíncronas com os integrantes do projeto.
 5. IntelliJ IDEA - ferramenta utilizada para a implementação do código fonte do projeto.

Onboarding

Libs utilizadas no projeto

O projeto **Fiap Restaurant** foi construído utilizando as seguintes libs:

- Java 17 (<https://docs.oracle.com/en/java/javase/17>)
- Spring Boot 3.2.2 (<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>)
- Spring Web MVC (<https://docs.spring.io/spring-framework/reference/web/webmvc.html>)
- Spring Data JPA (<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>)
- Spring Security (<https://docs.spring.io/spring-security/reference/index.html>)
- JUnit (<https://junit.org/junit5/docs/current/user-guide/>)
- TestContainers (<https://java.testcontainers.org/>)
- Flyway (<https://documentation.red-gate.com/fd>)
- Docker (<https://docs.docker.com/reference/>)
- Lombok (<https://projectlombok.org/>)
- Mockito (<https://site.mockito.org/>)
- ArchUnit (<https://www.archunit.org/>)
- PostgreSQL 15.1 (<https://www.postgresql.org/docs/15/index.html>)
- Gradle (<https://docs.gradle.org/current/userguide/userguide.html>)Spring Docs Open API (<https://springdoc.org/>)
- Ethereal Fake SMTP (<https://ethereal.email/>)

Repositório do projeto no GitHub

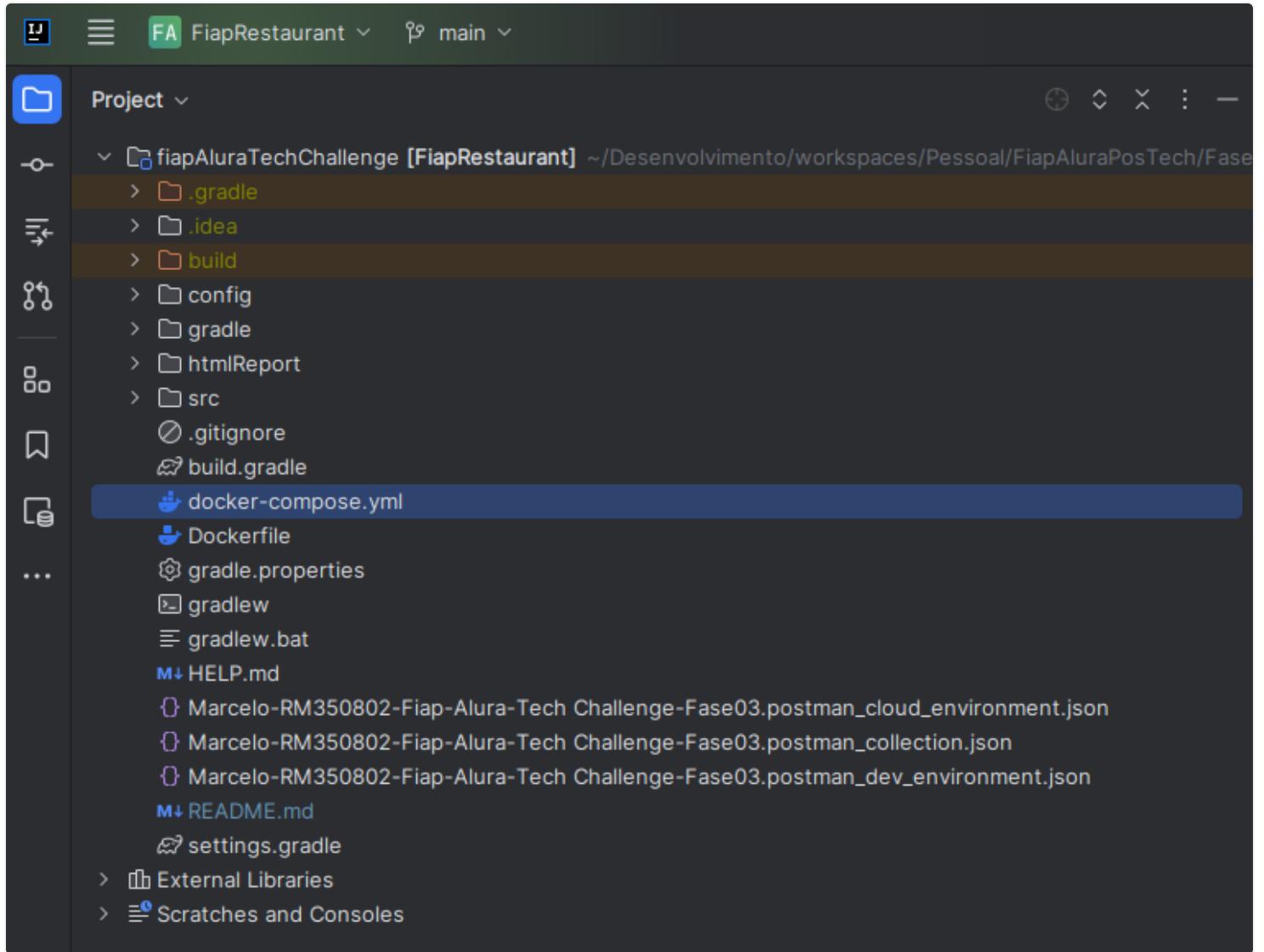
<https://github.com/EvolutionTeamFiapAluraPostech/fiapAluraTechChallengeFase03>

Setup e Start do Projeto

Para realizar o setup do projeto é necessário possuir o Java 17, docker 24 e docker-compose 1.29 instalado em sua máquina.

Faca o download do projeto (link do repositório acima) e atualize suas dependências com o gradle.

Antes de iniciar o projeto é necessário criar o banco de dados. O banco de dados está programado para ser criado em um container. Para criar o container, execute o docker-compose.



Acesse a pasta raiz do projeto (como destacado na figura acima, no mesmo local onde encontra-se o arquivo build.gradle). Neste local existe o arquivo docker-compose.yml. Para executá-lo, execute o comando docker-compose up -d (para rodar detached e não prender o terminal). Em seguida, acesse a classe [FiapRestaurantApplication.java](#) e inicie o projeto executando o start da IDE.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "fiapAluraTechChallenge [FiapRestaurant]". It includes modules like gradle, idea, build, config, gradle, htmlReport, src, main, java, resources, test, gitignore, build.gradle, Dockerfile, gradle.properties, gradlew, gradlew.bat, README.md, settings.gradle, External Libraries, and Scratches and Consoles.
- Code Editor:** The file "FiapRestaurantApplication.java" is open. The code defines a Spring Boot application with annotations @CrossOrigin, @EnableAsync, and @SpringBootApplication. It contains a static void main method that runs the application with args.

Após a inicialização do projeto, será necessário se autenticar, pois o Spring Security está habilitado. Para tanto, utilize o Postman (ou outra aplicação de sua preferência), crie um endpoint para realizar a autenticação, com a seguinte url **localhost:8080/authenticate**. No body, inclua um json contendo o atributo “email” com o valor “thomas.anderson@itcompany.com” e outro atributo “password” com o valor “@Bcd1234”. Realize a requisição para este endpoint para se obter o token JWT que deverá ser utilizado para consumir os demais endpoints do projeto. Segue abaixo o código fonte do endpoint para se autenticar na aplicação.

```
POST /authenticate HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 76

{
  "email": "[thomas.anderson@itcompany.com] (<mailto:thomas.anderson@itcompany.com>)",
  "password": "@Bcd1234"
}
```

Segue abaixo uma figura com o exemplo da requisição de autenticação.

POST [\[url\]\]/authenticate](#) Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL [JSON](#) [Beautify](#)

```
1 {  
2   ... "email": "thomas.anderson@itcompany.com",  
3   ... "password": "@Bcd1234"  
4 }
```

Body Cookies Headers (19) Test Results Status: 200 OK Time: 7.87 s Size: 826 B Save as example

Pretty Raw Preview Visualize [JSON](#)

```
1 {  
2   "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
eyJzdWIiOiJ0aG9tYXMuYW5kZXJzb25AxRj621wYh55LmNvbSIiMlcycI6IkFQSSBGSUUFQifJlc3RhdxJhbnQilCJuYW1lIjoVGhvbWFzIEFuZGVyc29uIiwiZXhwIjoxNzExMjExNDMwfQ.  
WZVqInVPzlnAsEcfaSjNjtI-agRnBsamuFvxaN376c"  
3 }
```

Glossary

- Restaurante: Estabelecimento comercial onde se servem refeições ao público, mediante pagamento.
- **Mesa:** Móvel formado por uma superfície horizontal e um ou mais pés que o sustêm, e que é usado para fazer refeições.
- **Reserva:** Ação de garantir antecipadamente a utilização de uma mesa, em determinada data e horário.
- **Avaliação:** classificar sua experiência, estimar a qualidade, dar uma nota ao restaurante.
- **Usuário do sistema:** é a pessoa interessada em realizar uma reserva de mesa no restaurante ou avaliação

Design architecture, standards and conventions

Objetivos

Esta página descreve a arquitetura, padrões e convenções utilizadas no projeto, a fim garantir que o código seja fácil de entender, manter e escalar.

Arquitetura do Projeto

A arquitetura do projeto foi definida pelos coordenadores do projeto, a fim de demonstrar na prática o entendimento da **Clean Architecture**. Para tanto, inspirada em um exemplo da apostila da disciplina de Clean Architecture (apostila número 46, aula número 4, Clean Architecture, página 6), onde é demonstrada a “cebola” e suas camadas. As camadas externas da “cebola” conhecem as camadas internas, porém as camadas internas não conhecem as camadas externas, separando assim suas dependências e responsabilidades. Cada subdomínio da aplicação possui sua separação em camadas “cebola”. Por exemplo, temos o subdomínio do usuário, e este subdomínio possui a camada web, controllers, use case e entities. Da mesma maneira ocorre com o subdomínio de restaurantes e assim por diante.

A camada mais interna, chamada **Entites**, contém as entidades do subdomínio da aplicação. Esta camada é isolada e não conhece as camadas mais externas, ou seja, não consome um use case, controller ou web. Esta camada se preocupa em “o que” o software faz.

A camada imediatamente superior, chamada **Use Case**, contém as classes que vão representar os fluxo da aplicação, sendo que cada classe de use case tem apenas uma única responsabilidade. Nela são recebidos os dados, a lógica (algoritmo) e a saída. Esta camada conhece e acessa recursos da camada entities inferior, permitindo orquestrar regras de negócio nela contidas. Porém, esta camada não conhece a camada superior. Esta camada se preocupa em “como” o software faz.

A camada superior seguinte é dos **Controllers, Gateways e Presenters**. Esta camada terá a responsabilidade de adaptar solicitações da camada mais externa, traduzir e direcionar para a camada de **Use Case**.

Por fim, temos a camada dos Devices, WEb, DB, UI e External Interfaces, que faz a intermediação entre elementos externos e internos.

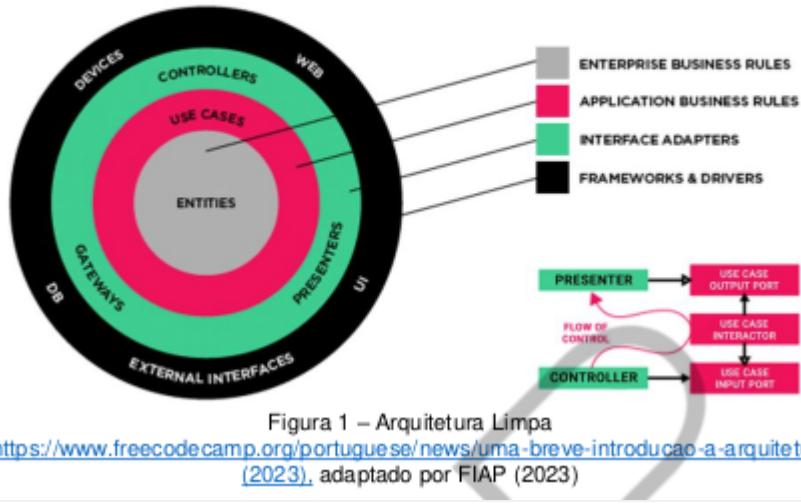


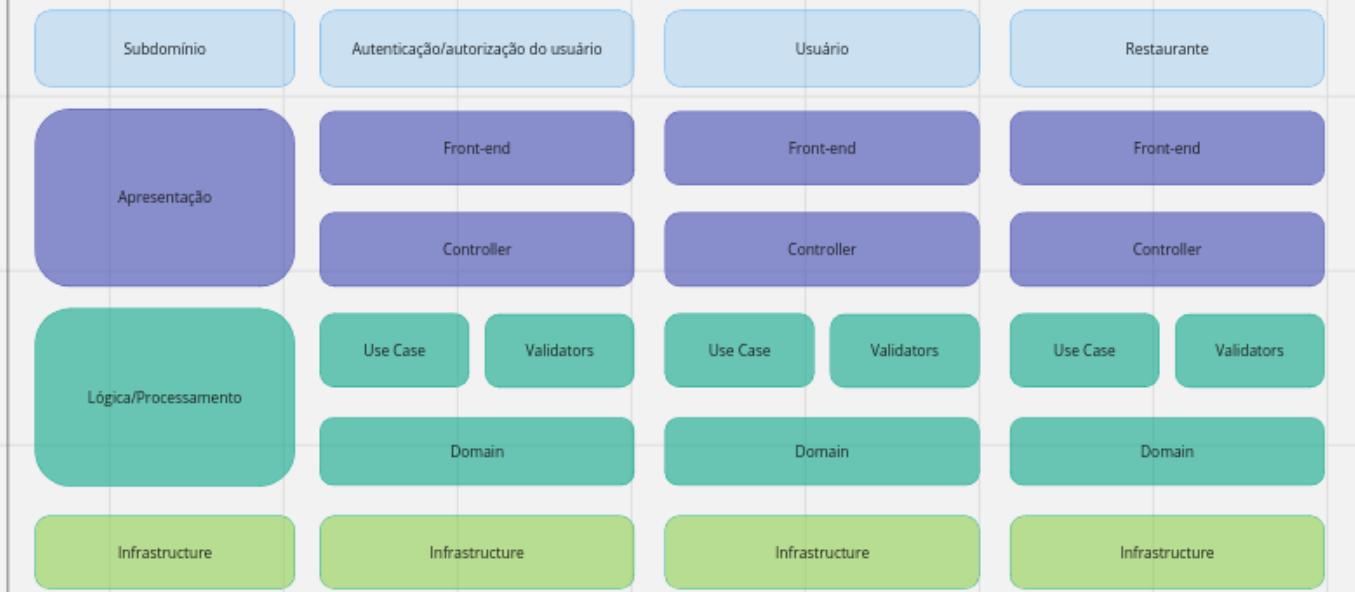
Figura 1 – Arquitetura Limpa

Fonte: <https://www.freecodecamp.org/portuguese/news/uma-breve-introducao-a-a-arquitetura-limpa/> (2023), adaptado por FIAP (2023)

O projeto foi em com um serviço, “dockerizado” e podem ser replicado, conforme a necessidade de escalabilidade.

Architecture

Container 1 - Aplicação para reserva de mesa em restaurante



Container 2 -Postgresql Database

Dados/Persistência

Database

O banco de dados utilizado foi o Postgresql 15.1, com a seguinte estrutura:

11 - Entity Relationship Diagram



Para criar a imagem docker da aplicação, está sendo utilizado o seguinte Dockerfile

```

Dockerfile
 1 FROM gradle:7.6.4-jdk17-alpine AS build
 2 COPY --chown=gradle:gradle . /home/gradle/src/producer
 3 WORKDIR /home/gradle/src/producer
 4 RUN gradle bootJar --no-daemon --stacktrace
 5 FROM openjdk:17-jdk-alpine
 6 ARG JAR_FILE=build/libs/*.jar
 7 COPY --from=build /home/gradle/src/producer/build/libs/*.jar fiaprestaurant.jar
 8 ENTRYPOINT ["java","-jar","/fiaprestaurant.jar"]
  
```

Para criar um container docker com o banco de dados postgres e executar a aplicação localmente, está sendo utilizado o seguinte docker-compose.yml.

The screenshot shows a code editor interface with a sidebar and a main panel. The sidebar on the left lists files and folders related to a project named 'FiapRestaurant'. The main panel displays the contents of a file named 'docker-compose.yml'.

```
version: '3.1'
services:
  fiaprestaurante-service-db:
    container_name: postgresql-fiaprestaurant
    image: postgres:15.1
    environment:
      POSTGRES_DB: fiaprestaurant-db
      POSTGRES_USER: fiaprestaurant-postgres-user
      POSTGRES_PASSWORD: fiaprestaurant-postgres-pwd
    ports:
      - 5432:5432
    volumes:
      - /var/lib/postgres
    networks:
      - postgres-compose-network
networks:
  postgres-compose-network:
    driver: bridge
```

Private (<https://app.clickup.com/9013160904/docs/8ckkuy8-413/8ckkuy8-573>)

Backend

Recomendamos no backend do projeto a implementação do código em inglês e de conceitos como SOLID, DRY, KISS, YAGNI e CLEAN CODE, onde cada um será resumidamente explicado abaixo:

SOLID

SOLID é um acrônimo que representa cinco princípios de design de software:

S (Single Responsibility Principle): O objeto de uma classe deve possuir apenas uma única responsabilidade, executar apenas uma tarefa bem feita, tornando o código mais simples, fácil de manter e entender.

O (Open Closed Principle): A classe deve estar aberta para a extensão e fechada para modificação, permitindo que a classe seja facilmente herdada sem a necessidade de alteração de código já implementado, tornando o conjunto de classes flexíveis, resistente a mudanças e facilitando sua testabilidade.

L (Liskov Substitution Principle): Os objetos de uma classe devem ser substituíveis por instâncias de suas subclasses, sem que isto afete a execução correta da aplicação, tornando o código modular e de fácil manutenção.

I (Interface Segregation Principle): As interfaces devem ser pequenas e possuir métodos específicos para um único objetivo, a fim de evitar a implementação de métodos desnecessários em classes concretas, tornando o código mais fácil de manter e entender.

D (Dependency Inversion Principle): Módulos de alto nível não devem depender de módulos de baixo nível, ambos devem depender de abstrações, ou seja conceitos. E conceitos não devem depender de detalhes e detalhes devem depender de conceitos, tornando o código modular e mais fácil de testar.

DRY

DRY é o acrônimo para “Don’t Repeat Yourself”, ou seja, não deve-se implementar código duplicado que realiza um objetivo. Se houver uma funcionalidade que é utilizada em vários lugares, deve-se concentrá-la em uma única classe, tornando o código mais organizado, fácil de manter e testar.

KISS

KISS é o acrônimo para “Keep It Simple, Stupid”, ou seja, deve-se implementar um código simples, sem complexidade desnecessária, tornando a compreensão e leitura simples, facilidade de manutenção, testável e reduzindo a possibilidade de erros.

YAGNI

YAGNI é o acrônimo para “You Aren’t Gonna Need It”, ou seja, deve-se adicionar apenas funcionalidades necessárias na aplicação para o momento atual, reduzindo o tempo de implementação de uma feature, implementação apenas de testes necessários, facilidade de manutenção e reduzindo a possibilidades de erros.

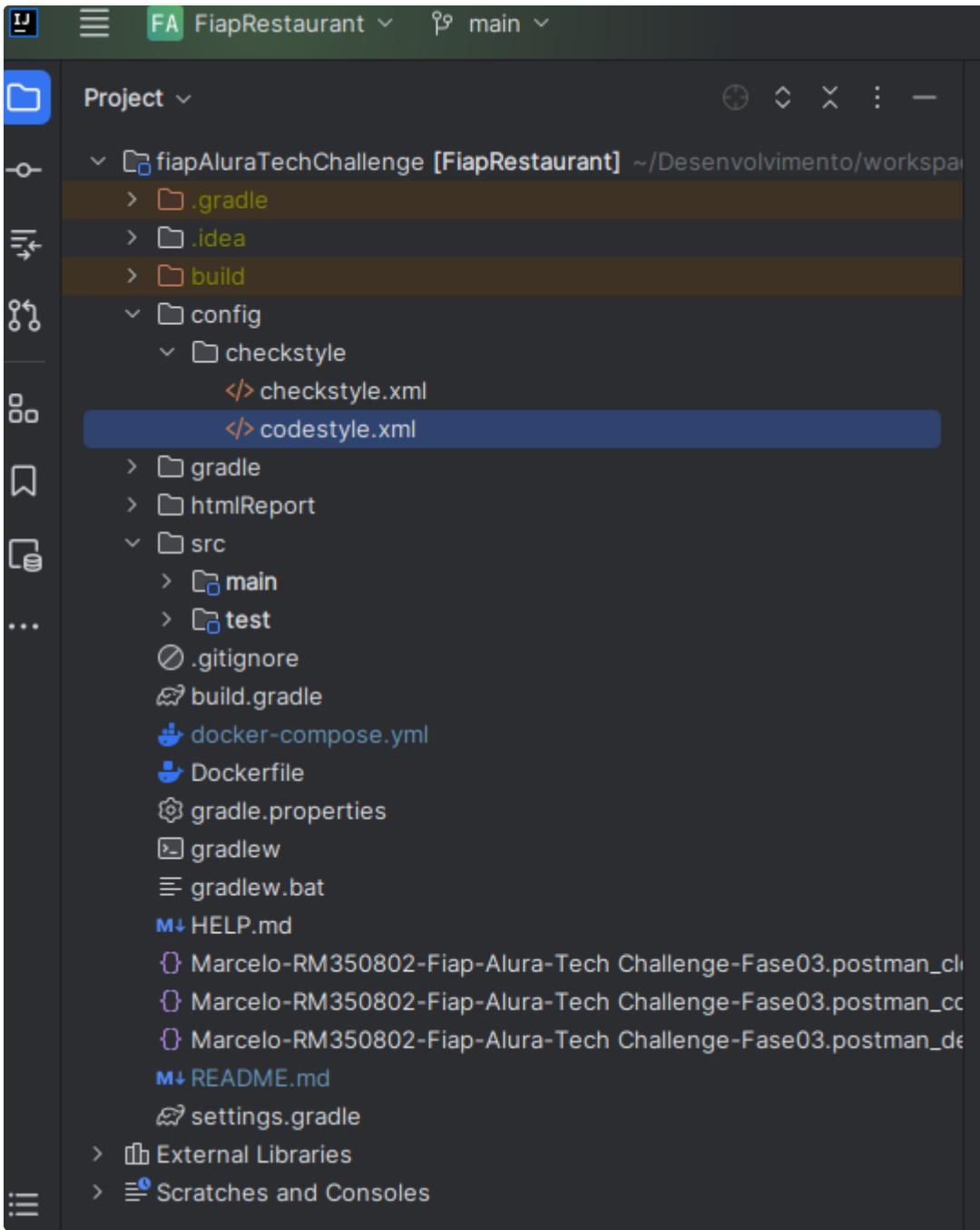
CLEAN CODE

Clean Code são técnicas que facilitam a escrita e leitura de um código, possibilitando maior compreensão, segurança em refatorações de código e testes. Resumidamente é um código limpo, testável e fácil de ser mantido, assim destacamos algumas recomendações:

1. Nomes de classes, métodos, atributos, variáveis, parâmetros, etc... devem ser claros, objetivos, definir exatamente o propósito. Não deve-se resumir um nome, mesmo que este seja extenso.
2. Prefira implementar em um método público um código que seja simples, pequeno e autoexplicativo (como se estivesse contando uma história), deixando detalhes de implementação para métodos privados que serão utilizados no método público. Evite utilizar no método público blocos condicionais, blocos de repetição, estruturas de switch, etc... que possam causar dúvida no leitor do seu código, utilize-os nos métodos privados. Crie métodos pequenos e coesos.
3. Implemente seu código como se o próprio código informe o que está sendo feito, evite comentários desnecessários.
4. Realize um tratamento de exceções de qualidade e as utilize quando necessário. Nunca realize um tratamento de exceção que não definirá uma ação.
5. Implemente todos os testes definidos pelo time (de unidade e integração), para garantir que sua alteração não afetou o comportamento da aplicação.
6. Confira os warnings da IDE e resolva-os. Por exemplo, não deixe imports de classes não utilizadas sobrando no código. Observe a quebra de linha. Utilize os atalhos CTRL + O para remover os imports desnecessários e CTRL + L para quebrar as linhas.
7. Regra do escoteiro: observar sempre a qualidade do código, se você precisou alterar o código de uma classe, método, etc... é necessário que este código seja entregue mais clean do que no momento em que você o encontrou, seguindo as recomendações anteriores.

A formatação do código está seguindo o Google Code Style. Basicamente o tab tem dois espaços e o total de colunas da área de código foi definida em 120 colunas.

Para configurar no IntelliJ utilize o arquivo contido no projeto, como mostra a figura abaixo:



CLEAN ARCHITECTURE

Injeção de dependências

Ao realizar a injeção de dependências, prefira realizá-la pelo construtor da classe, pois deixará a classe com uma clara necessidade de quais componentes são necessários, bem como sua quantidade. Se houverem muitas classes sendo injetadas no construtor, há um sinal de que a classe está com muitas responsabilidades. Pode também facilitar o mock de classes para testes e pode prevenir a dependência cíclica de classes (uma classe que depende da outra e vice-versa).

Camada Domain

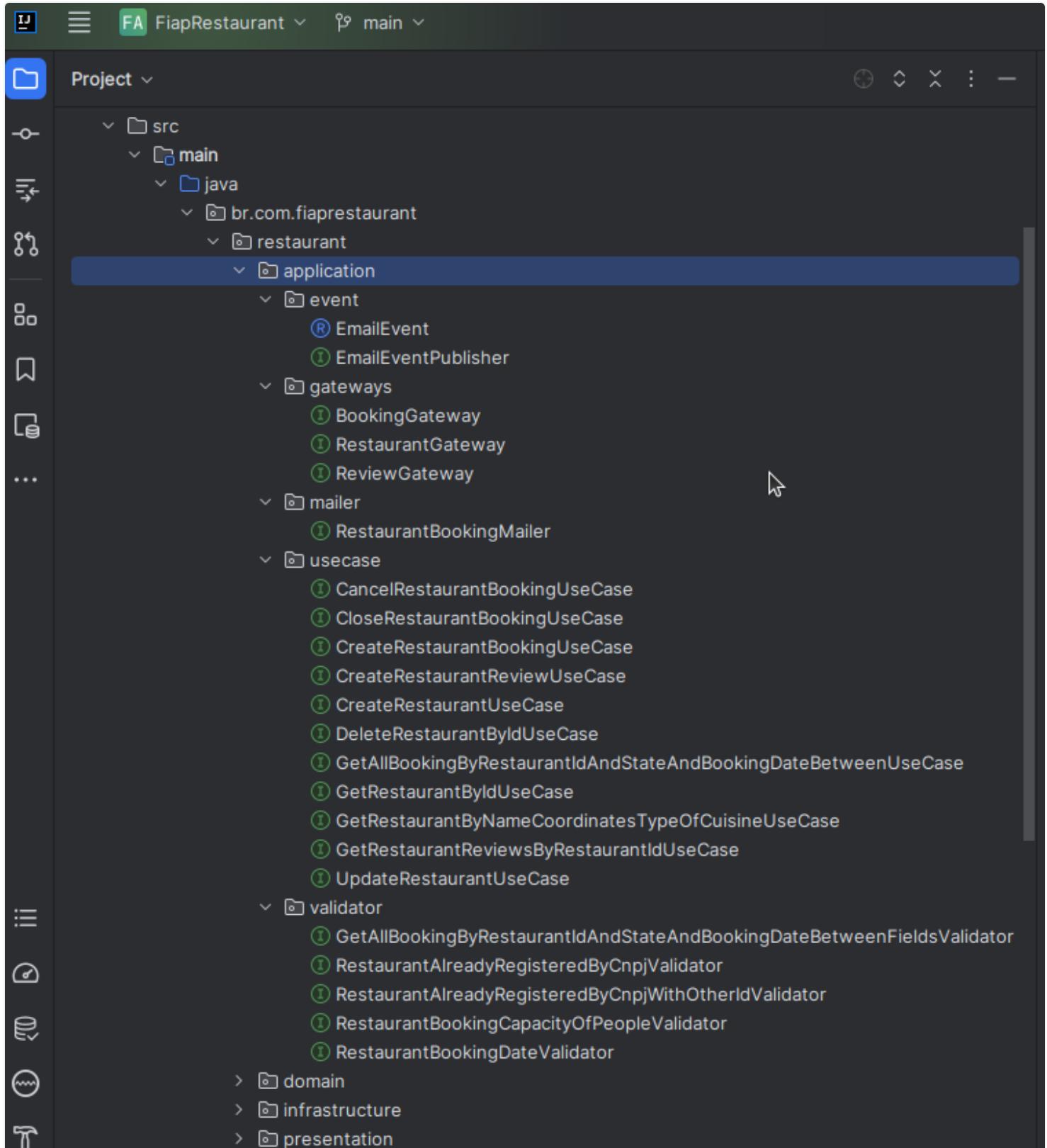
Esta camada é o coração da aplicação. Nela estão contidas as regras de negócios, implementadas nas as entidades e value objects. Contém também classes que dão suporte a classes de entidades e value objects. São classes puras, imutáveis e independente de frameworks.

The screenshot shows a Java project structure in an IDE. The project is named 'FiapRestaurant' and contains a 'src' directory with a 'main' package. Within 'main', there is a 'br.com.fiaprestaurant' package which contains a 'domain' package. The 'entity' sub-package of 'domain' contains four classes: 'Booking', 'Restaurant', 'RestaurantBuilder', and 'Review'. The 'Restaurant.java' file is open in the editor. It defines a class 'Restaurant' with various fields like name, CNPJ, type of cuisine, address, opening and closing times, and capacity. The code includes several annotations and imports. The file is annotated with 'Marcelo Akio Nishimori'.

```
1 package br.com.fiaprestaurant.restaurant.domain.entity;
2
3 > import ...
4
5 Marcelo Akio Nishimori
6
7 public class Restaurant {
8
9     2 usages
10    private UUID id;
11    2 usages
12    private final String name;
13    2 usages
14    private final Cnpj cnpj;
15    2 usages
16    private final TypeOfCuisine typeOfCuisine;
17    2 usages
18    private final Address address;
19    2 usages
20    private final OpenAt openAt;
21    2 usages
22    private final CloseAt closeAt;
23    2 usages
24    private final PeopleCapacity peopleCapacity;
25
26    ▲ Marcelo Akio Nishimori
27    public Restaurant(String name, String cnpj, String typeOfCuisine, Double latitude,
28                      Double longitude, String street, String number, String neighborhood, String city,
29                      String state, String postalCode, String openAt, String closeAt, int peopleCapacity) {
30        validateNameIsNullOrEmpty(name);
31        validateNameLength(name);
32        this.name = name.trim();
33        this.cnpj = new Cnpj(cnpj);
34        this.typeOfCuisine = new TypeOfCuisine(typeOfCuisine);
35        this.address = new Address(latitude, longitude, street, number, neighborhood, city, state,
36                                   postalCode);
37        this.openAt = new OpenAt(openAt);
38        this.closeAt = new CloseAt(closeAt);
39        this.peopleCapacity = new PeopleCapacity(peopleCapacity);
40    }
41}
```

Camada Application

Nesta camada estão contidas as classes que serão conhecidas pelas classes da camada superior, ou seja controllers, gateways ou presenters. As classes na camada application fazem a fronteira das classes do subdomínio da aplicação e que contém suas regras de negócio.



Interfaces de use case

As interfaces de use case são responsáveis por estabelecer um contrato de um fluxo de processo. As interfaces de use case devem ser criadas no pacote application/usecase de seu domínio, com o nome sufixo UseCase. Estas serão implementadas pelos respectivos Interactors contidos na camada de infrastructure.

Interfaces de validators

As interfaces de validators são responsáveis por estabelecer um contrato de validação de regra de negócio de um fluxo de processo, representado por use case. Devem ser criadas no pacote application/validator de seu domínio, com o nome sufixo Validator. Estas serão implementadas pelos respectivos Validators contidos na camada de infrastructure.

Interface de gateways

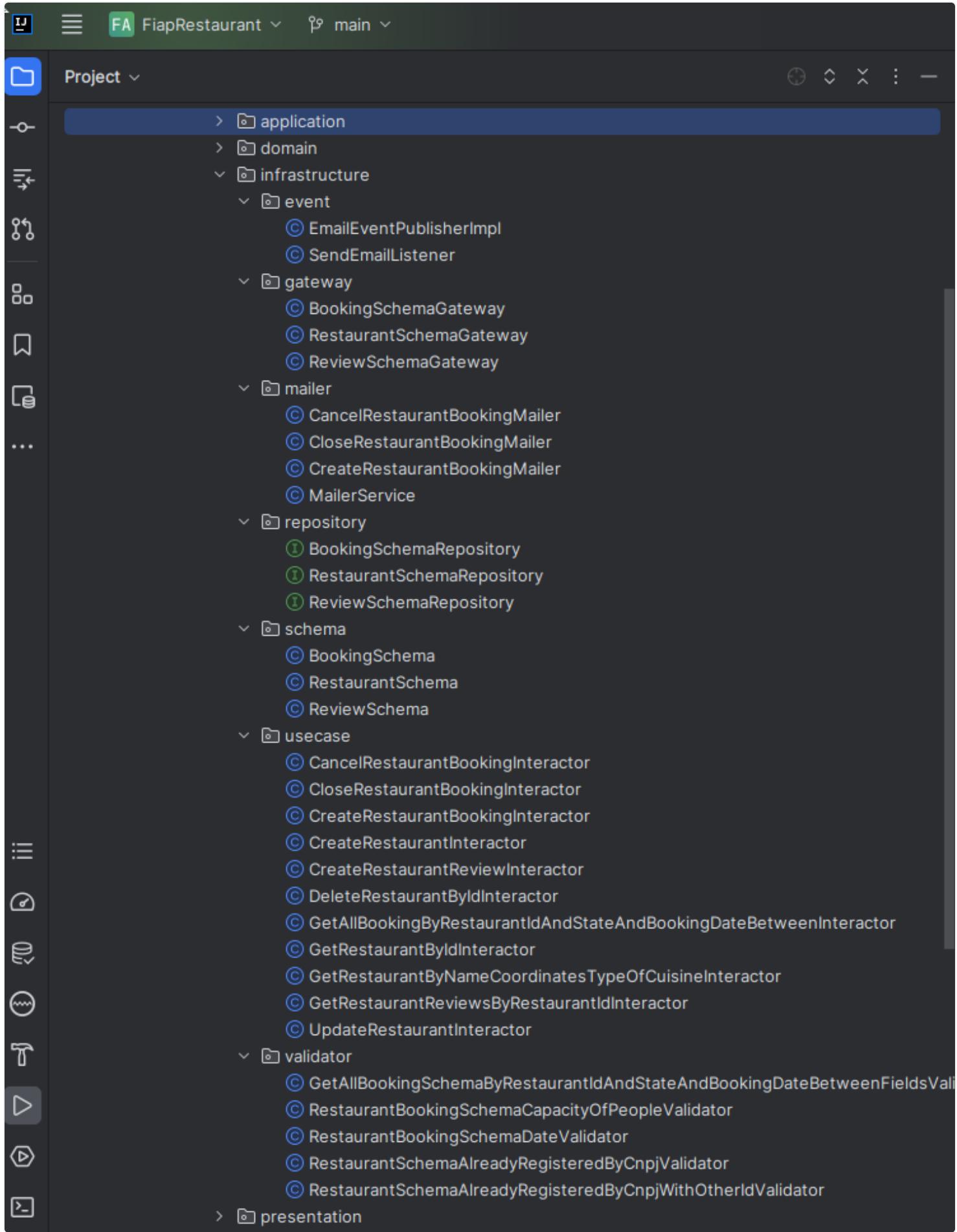
As interface de gateways são contratos que estabelecem a regra para a comunicação com os componentes externos. Estas serão implementadas pelos respectivos services contidos na camada de infrastructure.

Interface de mailers

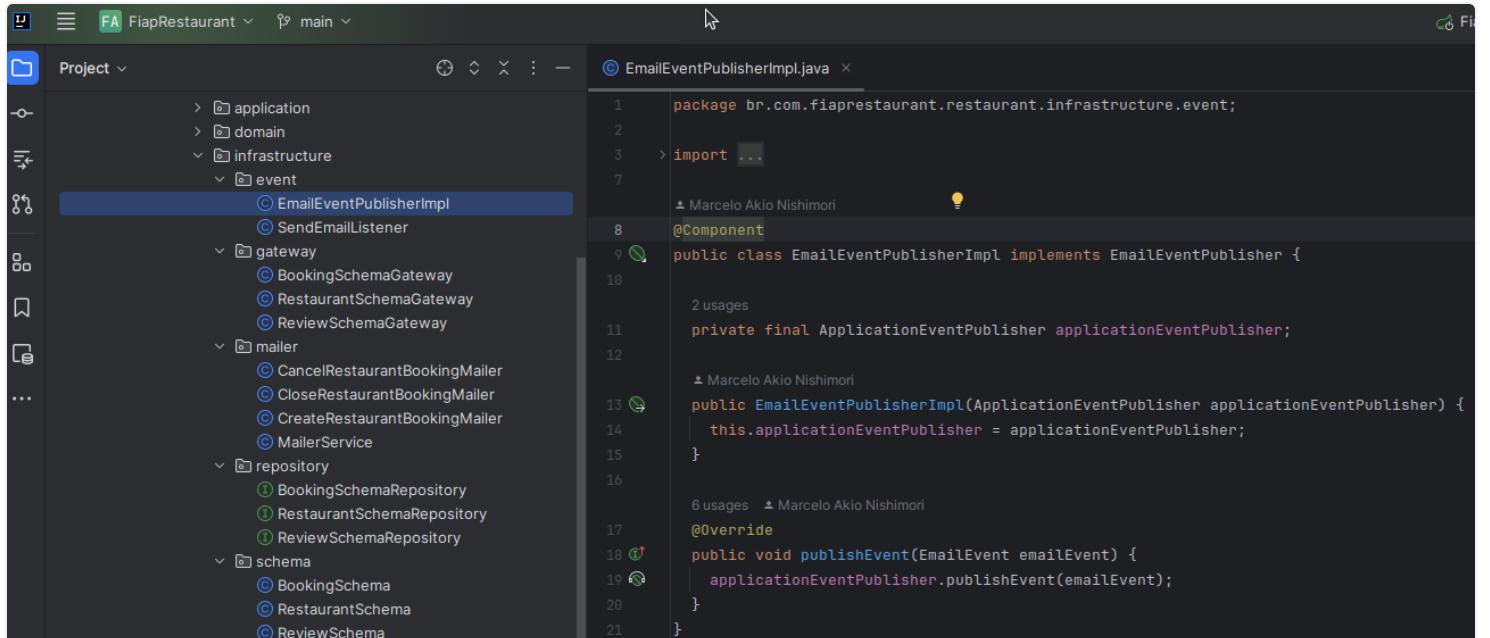
As interface de gateways são interfaces para estabelecer um contrato de envio de e-mail para comunicar a realização de algum processo representado por um use case.

Camada de infraestrutura

Nesta camada estarão as classes que conhecerão a camada de application, porém não o contrário. Aqui estão as classes que implementam interfaces da camada de application e executam operações externas ao centro da "cebola" onde estão contidas as regras de negócio. No caso da aplicação aqui desenvolvida, nesta camada estão as classes que utilizam o framework Spring para relizar tarefas, exemplo: classes que se comunicam com o banco de dados e classes que realizam envio de email. Aqui também estão os services, validators e use cases que utilizam o Spring para realizar suas operações planejadas.



Classes de event



```
package br.com.fiaprestaurant.restaurant.infrastructure.event;

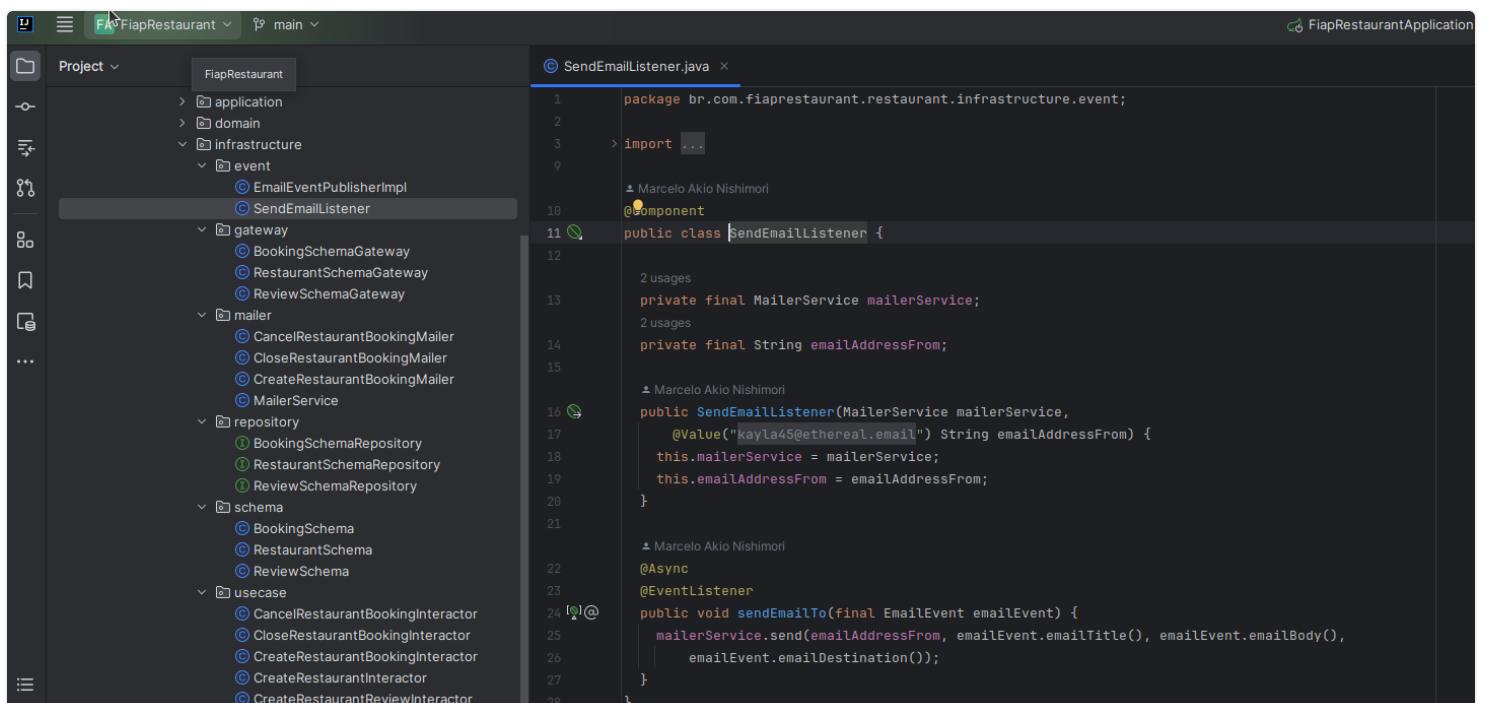
import ...;

@Marcelo Akio Nishimori
@Component
public class EmailEventPublisherImpl implements EmailEventPublisher {

    private final ApplicationEventPublisher applicationEventPublisher;

    @Marcelo Akio Nishimori
    public EmailEventPublisherImpl(ApplicationEventPublisher applicationEventPublisher) {
        this.applicationEventPublisher = applicationEventPublisher;
    }

    @Override
    public void publishEvent(EmailEvent emailEvent) {
        applicationEventPublisher.publishEvent(emailEvent);
    }
}
```



```
package br.com.fiaprestaurant.restaurant.infrastructure.event;

import ...;

@Marcelo Akio Nishimori
@Component
public class SendEmailListener {

    private final MailerService mailerService;
    private final String emailAddressFrom;

    @Marcelo Akio Nishimori
    public SendEmailListener(MailerService mailerService,
                           @Value("kayla45@ethereal.email") String emailAddressFrom) {
        this.mailerService = mailerService;
        this.emailAddressFrom = emailAddressFrom;
    }

    @Async
    @EventListener
    public void sendEmailTo(final EmailEvent emailEvent) {
        mailerService.send(emailAddressFrom, emailEvent.emailTitle(), emailEvent.emailBody(),
                           emailEvent.emailDestination());
    }
}
```

As classes de event são classes que implementam o design pattern Observer, onde temos um publicador e um ouvinte de eventos. Realiza operação assíncrona. Estas classes são fortemente acopladas com o framework Spring para realizar suas tarefas, utilizando features tais como:

- É um bean do Spring, com anotação `@component`;
- injeção de dependências;
- publicador com a interface `ApplicationEventPublisher`;
- ouvinte com a interface `@EventListener`;
- operação assíncrona com a interface `@Async`.

Classes de gateways

```
18     ▲ Marcelo Akio Nishimori
19     @Service
20     public class RestaurantSchemaGateway implements RestaurantGateway {
21
22         ▲ 1 usage
23         public static final String ENTER_AT_LEAST_ONE_PARAM_NAME_TYPE_OF_CUISINE_LATITUDE_AND_LONGITUDE = "Enter at
24             least one parameter name, type of cuisine, latitude and longitude";
25
26         ▲ Marcelo Akio Nishimori
27         private final RestaurantSchemaRepository restaurantSchemaRepository;
28
29
30         ▲ Marcelo Akio Nishimori
31         public RestaurantSchemaGateway(RestaurantSchemaRepository restaurantSchemaRepository) {
32             this.restaurantSchemaRepository = restaurantSchemaRepository;
33         }
34
35
36         ▲ Marcelo Akio Nishimori
37         @Override
38         public Restaurant save(Restaurant restaurant) {
39             var restaurantSchema = getRestaurantSchema(restaurant);
40             var restaurantSchemaSaved = restaurantSchemaRepository.save(restaurantSchema);
41             return restaurantSchemaSaved.createRestaurantFromRestaurantSchema();
42         }
43
44         ▲ 3 usages ▲ Marcelo Akio Nishimori
45         @Override
46         public Restaurant update(UUID id, Restaurant restaurant) {
47             var restaurantSchema = findRestaurantSchemaByIdRequired(id);
48             updateAttributes(restaurantSchema, restaurant);
49             var restaurantSchemaSaved = restaurantSchemaRepository.save(restaurantSchema);
50             return restaurantSchemaSaved.createRestaurantFromRestaurantSchema();
51         }
52
53         ▲ Marcelo Akio Nishimori
54         private void updateAttributes(RestaurantSchema restaurantSchema, Restaurant restaurant) {
55             restaurantSchema.setName(restaurant.getName());
56             restaurantSchema.setCnpj(restaurant.getCnpj().getCnpjValue());
57             restaurantSchema.setTypeOfCuisine(restaurant.getTypeOfCuisine().typeOfCuisineDescription());
58             restaurantSchema.setLatitude(restaurant.getAddress().getCoordinates().getLatitude());
59             restaurantSchema.setLongitude(restaurant.getAddress().getCoordinates().getLongitude());
60             restaurantSchema.setStreet(restaurant.getAddress().getStreet());
61             restaurantSchema.setNumber(restaurant.getAddress().getNumber());
62         }
63
64     }
```

As classes de gateways devem ser criadas no pacote infrastructure/gateways. São componentes que implementam a comunicação com os componentes externos, que fornecem ou armazenam dados ou efetuam atividades externas ao software, como por exemplo o acesso a repositório de dados. Devem ser criadas no pacote application/gateways de seu domínio, com o nome sufixo Gatway, contendo o mínimo necessário de métodos (considerando o conceito YAGNI).

Estas classes são fortemente acopladas com o framework Spring para realizar suas tarefas, utilizando features tais como:

- É um bean do Spring, com anotação @Service;
- injeção de dependências @Repository;

Classes de Mailer

As classes de mailer são responsáveis por enviar um e-mail e realizar uma comunicação aos interessados ao final de um fluxo de processo, representado por uma classe de use case. Neste projeto, ao realizar, cancelar ou fechar uma reserva, o usuário receberá um e-mail formalizando sua solicitação.

Devem ser criadas no pacote infrastructure/mailer de seu domínio, com o sufixo Mailer, contendo apenas o método definido em seu contrato.

The screenshot shows a Java project structure in the left panel and the code for `CreateRestaurantBookingMailer.java` in the right panel.

Project Structure:

- fiapAluraTechChallenge [FipRestaurant]
- .gradle
- idea
- build
- config
- gradle
- htmlReport
- src
- main
- br.com.fiaprestaurant
- restaurant
- application
- event
- gateways
- mailer
 - RestaurantBookingMailer
 - usecase
 - validator
- domain
- infrastructure
 - event
 - gateway
 - mailer
 - CancelRestaurantBookingMailer
 - CloseRestaurantBookingMailer
 - CreateRestaurantBookingMailer
 - MailerService
 - repository
 - schema
 - usecase
 - validator
 - presentation
- shared
- user
- FipRestaurantApplication

CreateRestaurantBookingMailer.java Code:

```
1 package br.com.fiaprestaurant.restaurant.infrastructure.mailer;
2
3 > import ...
11
12     ▲ Marcelo Akio Nishimori
13     @Component("createRestaurantBookingMailer")
14     public class CreateRestaurantBookingMailer implements RestaurantBookingMailer {
15
16         2 usages
17         ▲ Marcelo Akio Nishimori
18         private final EmailEventPublisher emailEventPublisher;
19
20         ▲ Marcelo Akio Nishimori
21         @Override
22         @
23         public void createAndSendEmail(Booking booking, Restaurant restaurant, User user) {
24             var emailTitle = "Booking made at %s restaurant ".formatted(restaurant.getName());
25             var emailBody = createEmailBody(booking, restaurant, user);
26             var emailDestination = user.getEmail().address();
27             emailEventPublisher.publishEvent(new EmailEvent(emailTitle, emailBody, emailDestination));
28         }
29
30         1 usage ▲ Marcelo Akio Nishimori
31         private String createEmailBody(Booking booking, Restaurant restaurant, User user) {
32             var formatDate = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
33             var bookingDateTime = formatDate.format(booking.getBookingDate());
34             return """
35                 |Hello %s, your reservation at the %s restaurant, for %s, was made successfully!
36                 """.formatted(user.getName(), restaurant.getName(), bookingDateTime);
37     }
```

As classes de mailer publicam o evento de email, onde o listener de email estará ouvindo estas publicações, para enfim utilizar a classe de serviço envio de email, a MailerService. Assim, o conteúdo de cada e-mail muda, mas o mecanismo de envio de e-mail é o mesmo para toda a aplicação.

The screenshot shows the IntelliJ IDEA interface with the project 'flapAluraTechChallenge' open. The left sidebar displays the project structure under 'main'. The right pane shows the code editor for 'MailerService.java'.

```

1 package br.com.fiaprestaurant.restaurant.infrastructure.mailer;
2
3 > import ...
4
5
6 ▲ Marcelo Akio Nishimori
7 @Service
8 public class MailerService {
9
10    3 usages
11    private final JavaMailSender mailSender;
12
13    ▲ Marcelo Akio Nishimori
14    public MailerService(JavaMailSender mailSender) { this.mailSender = mailSender; }
15
16    2 usages ▲ Marcelo Akio Nishimori
17    public void send(String from, String subject, String content, String to) {
18        try {
19            MimeMessage mimeMessage = mailSender.createMimeMessage();
20            MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, multipart: false, encoding: "utf-8");
21
22            helper.setFrom(from);
23            helper.setTo(to);
24            helper.setSubject(subject);
25            mimeMessage.setContent(content, type: "text/html");
26            mailSender.send(mimeMessage);
27            FiapRestaurantApplication.logger.info("e-mail sent! " + content);
28        } catch (MessagingException e) {
29            FiapRestaurantApplication.logger.error("Error by sending e-mail! {}", e.getMessage());
30        }
31    }
32
33 }
34
35
36

```

Observação: neste projeto foi utilizado um smtp server fake (<https://ethereal.email/>).

Interfaces de repository

The screenshot shows the IntelliJ IDEA interface with the project 'flapAluraTechChallenge' open. The left sidebar displays the project structure under 'main'. The right pane shows the code editor for 'RestaurantSchemaRepository.java'.

```

1 package br.com.fiaprestaurant.restaurant.infrastructure.repository;
2
3 > import ...
4
5
6 ▲ Marcelo Akio Nishimori
7 public interface RestaurantSchemaRepository extends JpaRepository<RestaurantSchema, UUID> {
8
9    5 usages ▲ Marcelo Akio Nishimori
10   Optional<RestaurantSchema> findByCnpj(String cnpjValue);
11
12   3 usages ▲ Marcelo Akio Nishimori
13   @Query("""
14       select r
15       from RestaurantSchema r
16       where r.deleted = false
17       and (:name is null or lower(r.name) like %:name%)
18       and (:typeOfCuisine is null or lower(r.typeOfCuisine) like %:typeOfCuisine%)
19       and (:latitude is null and :longitude is null) or (r.latitude = :latitude and r.longitude = :longitude)
20       order by r.name
21   """)
22   List<RestaurantSchema> queryByNameCoordinatesTypeOfCuisine(String name, String typeOfCuisine,
23                                                               Double latitude, Double longitude);
24
25
26 }

```

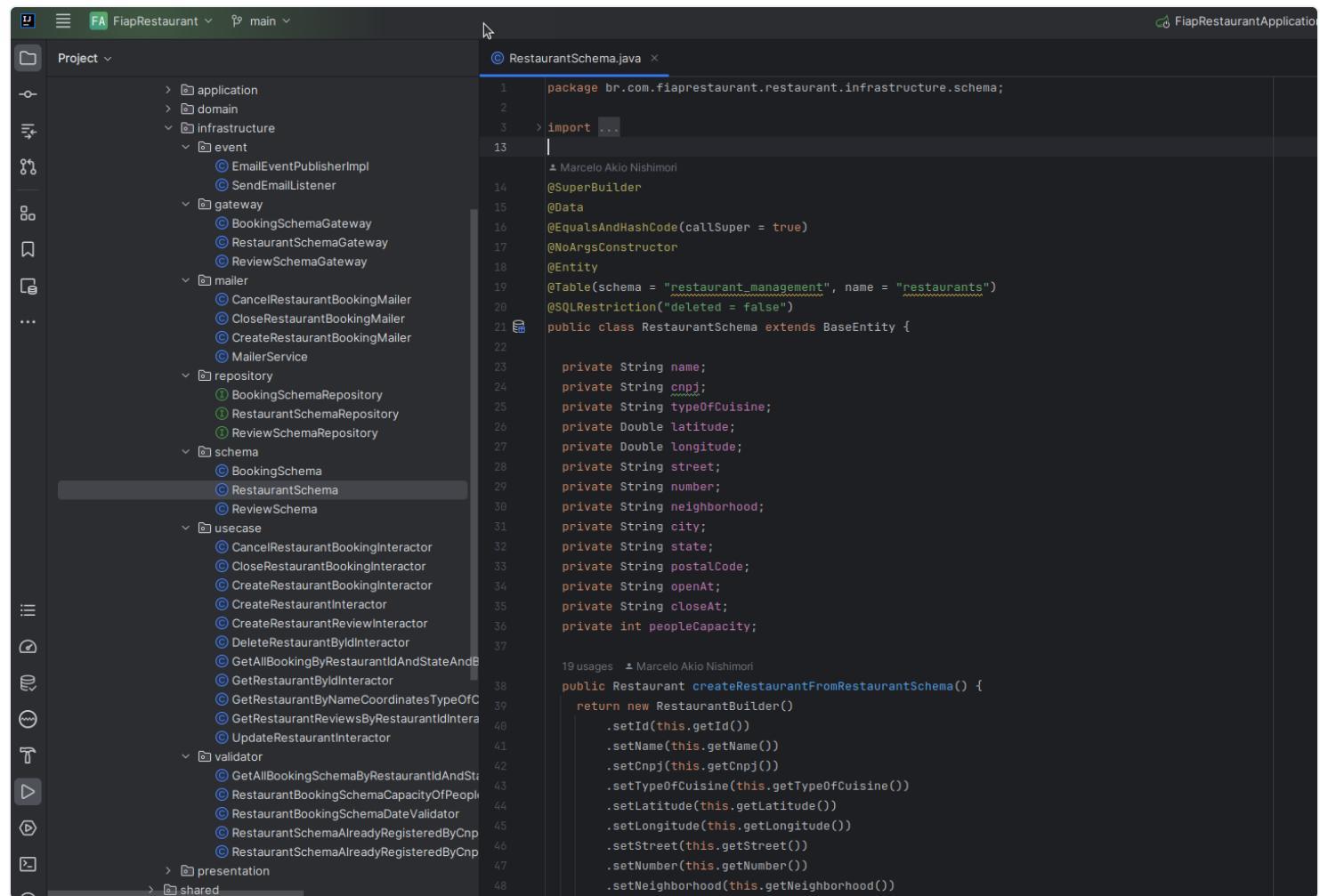
As classes de repository devem ser criadas no pacote infrastructure/repository. São componentes que implementam a comunicação com o banco de dados, com a biblioteca Spring Data JPA. Estas classes são fortemente acopladas com o framework Spring para realizar suas tarefas, utilizando features tais como:

- É um componente do Spring, com a anotação `@Repository`;

- Herança da classe JpaRepository;
- Query methods;
- Implementação de JPQL ou query SQL nativa;

Observação: A implementação de JPQL ou query nativa é utilizada neste projeto para resolver o "problema" do N+1 gerado pelo hibernate ou para implementar uma consulta complexa. Nos demais casos é utilizado querymethods do Spring Data JPA.

Classes de schema



The screenshot shows the IntelliJ IDEA interface with the project navigation bar at the top. The project structure on the left shows a package named 'br.com.fiaprestaurant.restaurant.infrastructure.schema' containing several classes under 'schema'. One class, 'RestaurantSchema', is currently selected and shown in the code editor on the right. The code editor displays Java code for a JPA entity named 'RestaurantSchema'.

```

package br.com.fiaprestaurant.restaurant.infrastructure.schema;
import ...;
import ...;

@Marcelo Akio Nishimori
@SuperBuilder
@Data
@EqualsAndHashCode(callSuper = true)
@NoArgsConstructor
@Entity
@Table(schema = "restaurant_management", name = "restaurants")
@SQLRestriction("deleted = false")
public class RestaurantSchema extends BaseEntity {

    private String name;
    private String cnpj;
    private String typeOfCuisine;
    private Double latitude;
    private Double longitude;
    private String street;
    private String number;
    private String neighborhood;
    private String city;
    private String state;
    private String postalCode;
    private String openAt;
    private String closeAt;
    private int peopleCapacity;

    public Restaurant createRestaurantFromRestaurantSchema() {
        return new RestaurantBuilder()
            .setId(this.getId())
            .setName(this.getName())
            .setCnpj(this.getCnpj())
            .setTypeOfCuisine(this.getTypeOfCuisine())
            .setLatitude(this.getLatitude())
            .setLongitude(this.getLongitude())
            .setStreet(this.getStreet())
            .setNumber(this.getNumber())
            .setNeighborhood(this.getNeighborhood());
    }
}

```

The screenshot shows a Java IDE interface with a project tree on the left and a code editor on the right. The project tree for 'FiapRestaurant' includes packages for application, domain, infrastructure (with event, gateway, mailer, repository, schema, and usecase sub-packages), and validation. The code editor displays 'BaseEntity.java' with the following content:

```
1 package br.com.fiaprestaurant.shared.domain.entity;
2
3 > import ...
4
5
6 10 usages 4 inheritors ▾ Marcelo Akio Nishimori
7 @MappedSuperclass
8 @Data
9 @SuperBuilder
10 @NoArgsConstructor
11 @AllArgsConstructor
12 @EqualsAndHashCode(onlyExplicitlyIncluded = true)
13 @EntityListeners(AuditingEntityListener.class)
14
15 public abstract class BaseEntity implements Serializable, AuditableEntity {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.UUID)
19     @EqualsAndHashCode.Include
20     protected UUID id;
21
22     @Builder.Default
23     @Column(nullable = false)
24     protected Boolean deleted = false;
25
26     @Builder.Default
27     @Version
28     @Column(nullable = false)
29     protected Long version = 0L;
30
31     @CreationTimestamp
32     protected LocalDateTime createdAt;
33
34     @UpdateTimestamp
35     protected LocalDateTime updatedAt;
36
37     protected String createdBy;
38
39     protected String updatedBy;
40
41     }
42
43     }
44
45     }
46     }
47
48     }
49
50     }
51     }
52
53     }
54 }
```

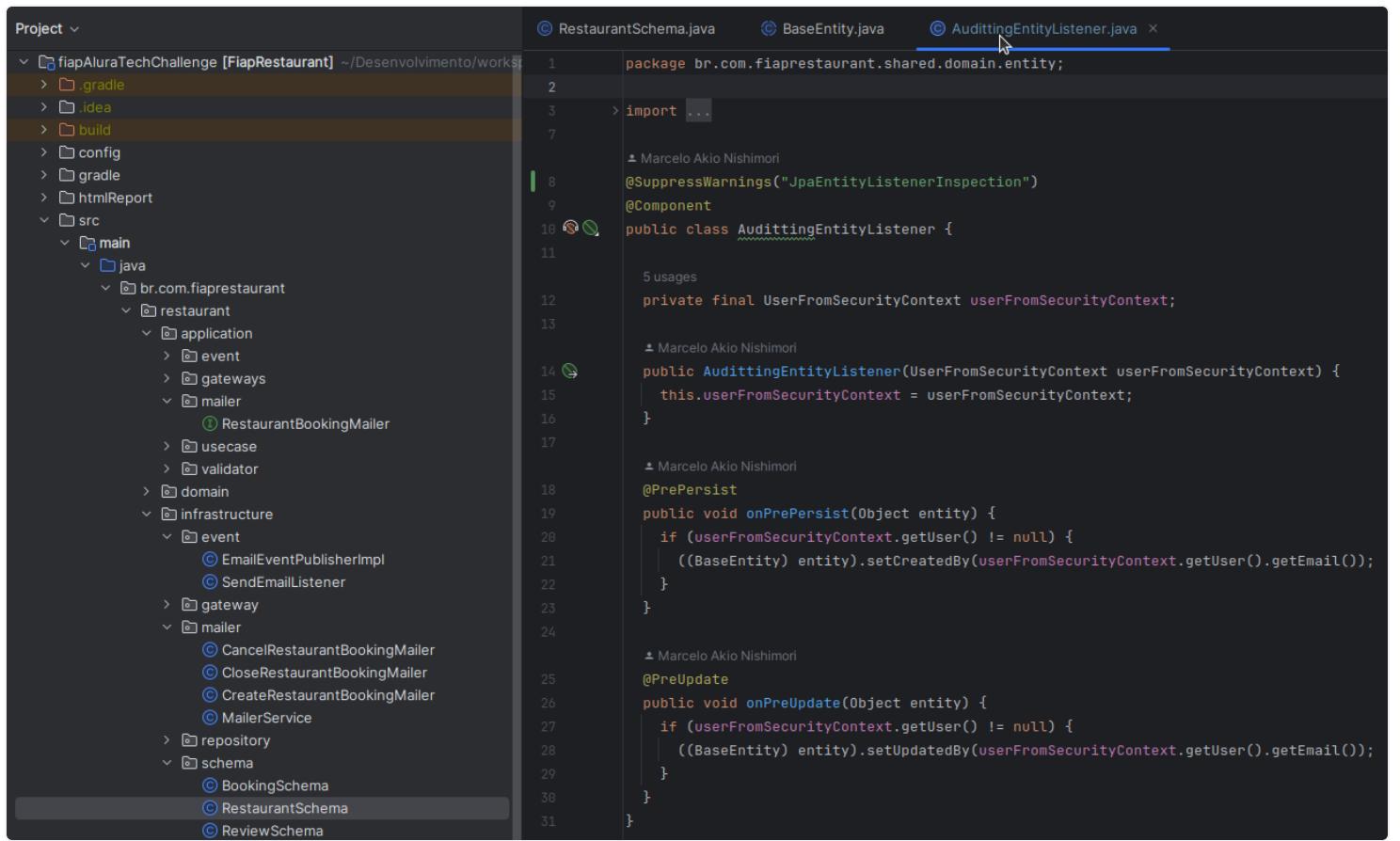
As classes de schema são as representações das tabelas no banco de dados, com anotações JPA. As classes de schema são extendidas de uma classe base, que possuem atributos comuns entre diversas as classes do projeto, de extrema importância, como o ID, se o registro é logicamente deletado, um campo para trabalhar a concorrência e informações de auditoria de primeiro nível, como por exemplo o usuário que realizou o cadastro, a data/hora que o cadastro foi realizado, o usuário que alterou o referido registro e a data/hora que o registro foi alterado.

Nestas classes, utilizamos o Lombok para oferecer ao dev construtores flexíveis, método equals e hashCode baseado no ID, getters and setters e toString.

Através de anotação fornecida pelo Hibernate (@SQLRestriction), desconsideraremos a recuperação de registros deletados logicamente.

Outra anotação interessante (@EntityListeners) indica que a classe será considerada por um listener que acionará métodos pré persistência ou pré atualização. O objetivo é recuperar o usuário autenticado e salvo na sessão do Spring e atribuir a propriedade que identifica o usuário que está realizando um cadastro (@PrePersist) ou atualizando (@PreUpdate). Assim, o dev não precisa se preocupar em identificar quem está realizando a operação de fluxo de um use case. A data de criação de um registro é atribuída automaticamente através da anotação @CreationTimestamp vinculada ao atributo createdAt. A data de atualização de um registro é

atribuída automaticamente através da anotação @UpdateTimestamp vinculada ao atributo updateAt. Os atributos createdBy, createdAt, updatedBy e updatedAt estão presentes em todas as classes de entidade JPA.



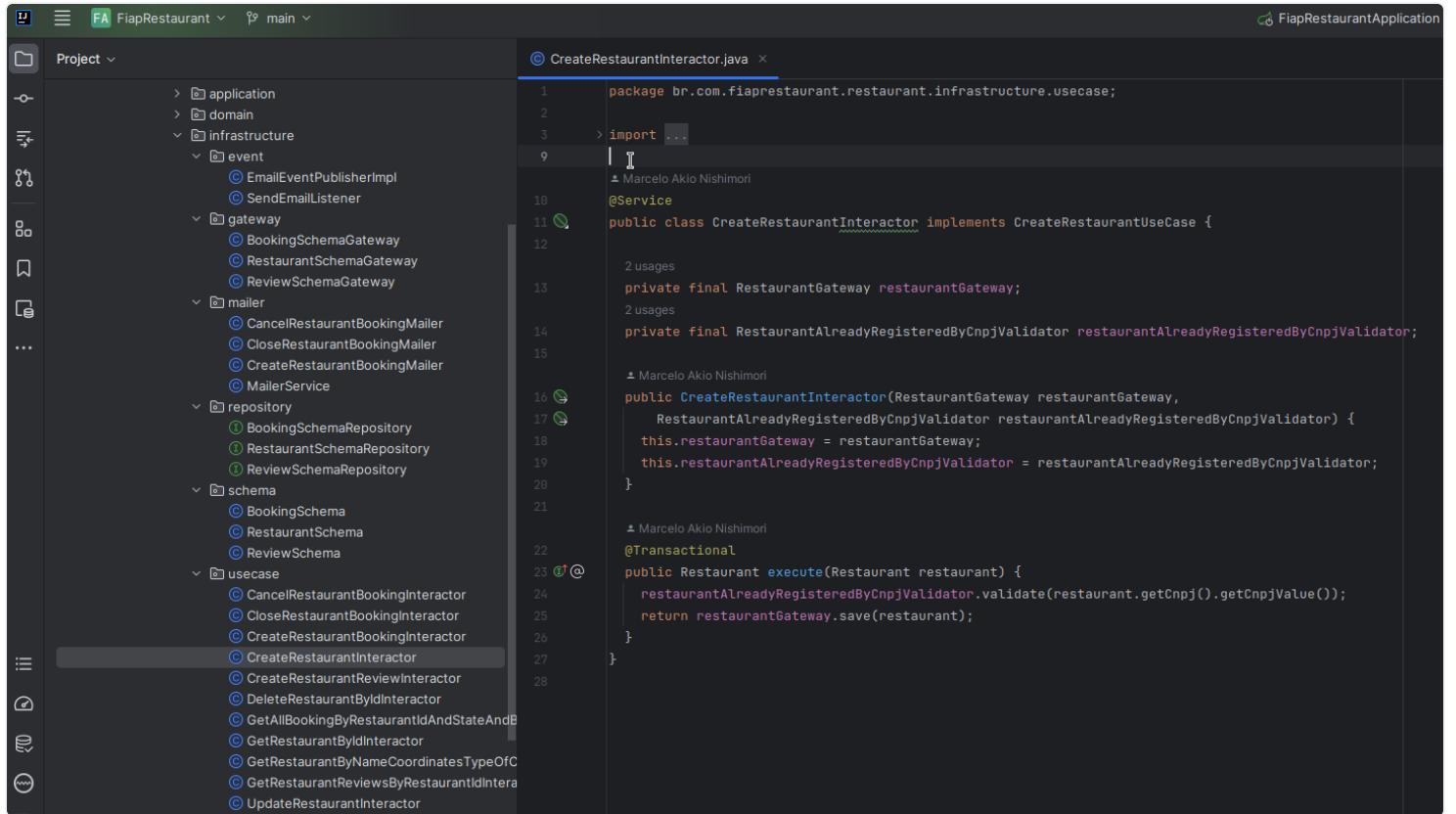
The screenshot shows a Java code editor with three tabs: RestaurantSchema.java, BaseEntity.java, and AuditingEntityListener.java. The AuditingEntityListener.java tab is active. The code implements a listener for entity persistence events. It uses annotations like @Component and @PrePersist to handle the creation and update of entities. The code references UserFromSecurityContext and BaseEntity, which are part of the Spring Security and Hibernate ecosystem respectively. The project structure on the left shows packages for application, domain, infrastructure, and various service layers like booking and review mailers.

```
1 package br.com.faprestaurant.shared.domain.entity;
2
3 import ...
4
5 usages
6
7 Marcelo Akio Nishimori
8 @SuppressWarnings("JpaEntityListenerInspection")
9 @Component
10 public class AuditingEntityListener {
11
12     private final UserFromSecurityContext userFromSecurityContext;
13
14     public AuditingEntityListener(UserFromSecurityContext userFromSecurityContext) {
15         this.userFromSecurityContext = userFromSecurityContext;
16     }
17
18     @PrePersist
19     public void onPrePersist(Object entity) {
20         if (userFromSecurityContext.getUser() != null) {
21             ((BaseEntity) entity).setCreatedBy(userFromSecurityContext.getUser().getEmail());
22         }
23     }
24
25     @PreUpdate
26     public void onPreUpdate(Object entity) {
27         if (userFromSecurityContext.getUser() != null) {
28             ((BaseEntity) entity).setUpdatedBy(userFromSecurityContext.getUser().getEmail());
29         }
30     }
31 }
```

Estas classes são fortemente acopladas com o framework Spring para realizar suas tarefas, utilizando features tais como:

- Entidade @Entity;
- Tabela @Table, identificação do schema de banco de dados e o nome da tabela;
- Restrição de pesquisa a registros deletados com @SQLRestrictions do Hibernate;
- Utilização do Lombok para simplificar o código da classe;

Classes de interactor



```
package br.com.fiaprestaurant.restaurant.infrastructure.usecase;

import ...;
import br.com.fiaprestaurant.restaurant.domain.Restaurant;
import br.com.fiaprestaurant.restaurant.infrastructure.gateway.RestaurantGateway;
import br.com.fiaprestaurant.restaurant.infrastructure.repository.RestaurantRepository;
import br.com.fiaprestaurant.restaurant.infrastructure.validator.RestaurantAlreadyRegisteredByCnpjValidator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;

@Service
public class CreateRestaurantInteractor implements CreateRestaurantUseCase {

    private final RestaurantGateway restaurantGateway;
    private final RestaurantAlreadyRegisteredByCnpjValidator restaurantAlreadyRegisteredByCnpjValidator;

    @Autowired
    public CreateRestaurantInteractor(RestaurantGateway restaurantGateway,
                                      RestaurantAlreadyRegisteredByCnpjValidator restaurantAlreadyRegisteredByCnpjValidator) {
        this.restaurantGateway = restaurantGateway;
        this.restaurantAlreadyRegisteredByCnpjValidator = restaurantAlreadyRegisteredByCnpjValidator;
    }

    @Transactional
    public Restaurant execute(Restaurant restaurant) {
        restaurantAlreadyRegisteredByCnpjValidator.validate(restaurant.getCnpj().getCnpjValue());
        return restaurantGateway.save(restaurant);
    }
}
```

As classes de interactor implementam as interfaces de casos de uso e tem como objetivo isolar cada regra de negócio em seu código único, ser testável e utilizável. Caso haja a necessidade de atualização de uma regra de negócio de um caso de uso, sabemos que não afetarão outras classes, sendo esta alteração pontual.

Estas classes são fortemente acopladas com o framework Spring para realizar suas tarefas, utilizando features tais como:

- É um bean do Spring, com anotação `@Service`;
- Injeção de dependências;
- Controle de transações do Spring como `@Transactional`;

Classes de validators

The screenshot shows a Java IDE interface with the following details:

- Project View:** Shows the package structure of the FlapRestaurant application. It includes:
 - gateway: SendEmailListener, BookingSchemaGateway, RestaurantSchemaGateway, ReviewSchemaGateway
 - mailer: CancelRestaurantBookingMailer, CloseRestaurantBookingMailer, CreateRestaurantBookingMailer, MailerService
 - repository: BookingSchemaRepository, RestaurantSchemaRepository, ReviewSchemaRepository
 - schema: BookingSchema, RestaurantSchema, ReviewSchema
 - usecase: CancelRestaurantBookingInteractor, CloseRestaurantBookingInteractor, CreateRestaurantBookingInteractor, CreateRestaurantInteractor, CreateRestaurantReviewInteractor, DeleteRestaurantByldInteractor, GetAllBookingByRestaurantIdAndStateAndBookingDateBetweenInteractor, GetRestaurantByldInteractor, GetRestaurantByNameCoordinatesTypeOfCuisineInteractor, GetRestaurantReviewsByRestaurantIdInteractor, UpdateRestaurantInteractor
 - validator: GetAllBookingSchemaByRestaurantIdAndStateAndBookingDateBetweenFieldsValidator, RestaurantBookingSchemaCapacityOfPeopleValidator, RestaurantBookingSchemaDateValidator, RestaurantSchemaAlreadyRegisteredByCnpjValidator, RestaurantSchemaAlreadyRegisteredByCnpjWithOtherIdValidator
- Code Editor:** Displays the content of `RestaurantBookingSchemaDateValidator.java`. The code implements `RestaurantBookingDateValidator` and contains logic to validate that a booking date is greater than the current date.

```

1 package br.com.flaprestaurant.restaurant.infrastructure.validator;
2
3 > import ...
4
5
6 /**
7  * Marcelo Akio Nishimori
8  * @Component
9  */
10 public class RestaurantBookingSchemaDateValidator implements RestaurantBookingDateValidator {
11
12     /**
13      * Marcelo Akio Nishimori
14      */
15
16     @Override
17     public void validate(LocalDateTime bookingDate) {
18         if (LocalDateTime.now().isAfter(bookingDate)) {
19             var actualDate = LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm"));
20             var bookingDateMessage = bookingDate.format(DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm"));
21             throw new ValidatorException(
22                 new FieldError(this.getClass(), RESTAURANT_BOOKING_DATE_FIELD,
23                               RESTAURANT_BOOKING_DATE_MUST_BE_GREATER_THAN_ACTUAL_DATE.formatted(actualDate,
24                               bookingDateMessage)));
25         }
26     }
27 }
28

```

As classes de validators são classes que aplicam uma regra de negócio do fluxo do processo do caso de uso e que não pode ser aplicada pela entidade de domínio. Por exemplo, a validação da data de reserva de uma mesa no restaurante deve ser superior a data atual. Contudo, a pesquisa do histórico de reservas de mesas em um restaurante é baseada no tempo passado, onde entidades terão a data da reserva no passado. Desta maneira, o caso de uso para criar uma reserva aplicará a validação da data de reserva ser superior a data atual, por outro lado, o caso de uso de pesquisa de históricos de reservas não terá validará a data da reserva.

Estas classes são fortemente acopladas com o framework Spring para realizar suas tarefas, utilizando features tais como:

- É um bean do Spring, com anotação `@component`;
- Injeção de dependências;

Camada de presentation/controller

As classes de controller devem ser criadas no pacote presentation/api de seu domínio, com o nome sufixo Api. Devem ser construídas seguindo os princípios e boas práticas de desenvolvimento de API REST, como identificação de recursos gerais no plural, URI legíveis, evitar citar a operação (criar, listar, etc) na URI, não citar o formato de retorno na URI, evitar alterar URI depois da feat entregue, utilizar corretamente o verbo HTTP, entre outros. O artigo que pode ajudar nestas explicações (https://www.alura.com.br/artigos/rest-principios-e-boas-praticas?gclid=CjwKCAjwpqCZBhAbEiwAa7pXeXGYyPmnLwzRctbBd_T38yHqXREcCarb_spiMnk9_8T-4BCs9iMshxoCwkYQAvD_BwE) foi publicado pela Alura.

The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the project structure for 'fiapAluraTechChallenge [FiapRestaurant]'. It includes folders for .gradle, .idea, build, config, gradle, htmlReport, src, main, java, br.com.fiaprestaurant, restaurant, application, domain, infrastructure, presentation, and api. Under api, there are subfolders for RestaurantsApi, RestaurantsBookingApi, RestaurantsBookingController, RestaurantsController, RestaurantsReviewsApi, and RestaurantsReviewsController. The code editor on the right shows the 'RestaurantsController.java' file. The code defines a controller class that implements a 'RestaurantsApi' interface. It contains several private final fields representing use cases: 'createRestaurantUseCase', 'getRestaurantByNameCoordinatesTypeOfCuisineUseCase', 'updateRestaurantUseCase', 'getRestaurantByIdUseCase', and 'deleteRestaurantByIdUseCase'. The constructor initializes these fields. A method 'postRestaurant' is annotated with @PostMapping and @ResponseStatus(HttpStatus.CREATED). It takes a RestaurantInputDto, converts it to a Restaurant entity, creates it using the use case, and returns a RestaurantOutputDto. The code is attributed to Marcelo Akio Nishimori.

```

23 |     * Marcelo Akio Nishimori
24 |     * @RestController
25 |     * @RequestMapping(value = "/restaurants")
26 |     */
27 |     public class RestaurantsController implements RestaurantsApi {
28 |
29 |         /**
30 |          * 2 usages
31 |          * Marcelo Akio Nishimori
32 |          * @private final CreateRestaurantUseCase createRestaurantUseCase;
33 |          */
34 |         private final CreateRestaurantUseCase createRestaurantUseCase;
35 |
36 |         /**
37 |          * 2 usages
38 |          * Marcelo Akio Nishimori
39 |          * @private final GetRestaurantByNameCoordinatesTypeOfCuisineUseCase getRestaurantByNameCoordinatesTypeOfCuisineUseCase;
40 |          */
41 |         private final GetRestaurantByNameCoordinatesTypeOfCuisineUseCase getRestaurantByNameCoordinatesTypeOfCuisineUseCase;
42 |
43 |         /**
44 |          * 2 usages
45 |          * Marcelo Akio Nishimori
46 |          * @private final UpdateRestaurantUseCase updateRestaurantUseCase;
47 |          */
48 |         private final UpdateRestaurantUseCase updateRestaurantUseCase;
49 |         /**
50 |          * 2 usages
51 |          * Marcelo Akio Nishimori
52 |          * @private final GetRestaurantByIdUseCase getRestaurantByIdUseCase;
53 |          */
54 |         private final GetRestaurantByIdUseCase getRestaurantByIdUseCase;
55 |         /**
56 |          * 2 usages
57 |          * Marcelo Akio Nishimori
58 |          * @private final DeleteRestaurantByIdUseCase deleteRestaurantByIdUseCase;
59 |          */
60 |         private final DeleteRestaurantByIdUseCase deleteRestaurantByIdUseCase;
61 |
62 |         /**
63 |          * Marcelo Akio Nishimori
64 |          * @PostMapping(value = "/")
65 |          * @ResponseStatus(HttpStatus.CREATED)
66 |          * @Override
67 |          */
68 |         public RestaurantOutputDto postRestaurant(@RequestBody @Valid RestaurantInputDto restaurantInputDto) {
69 |             var restaurant = restaurantInputDto.toRestaurantFrom();
70 |             var restaurantSaved = createRestaurantUseCase.execute(restaurant);
71 |             return RestaurantOutputDto.toRestaurantOutputDtoFrom(restaurantSaved);
72 |         }
73 |
74 |     }

```

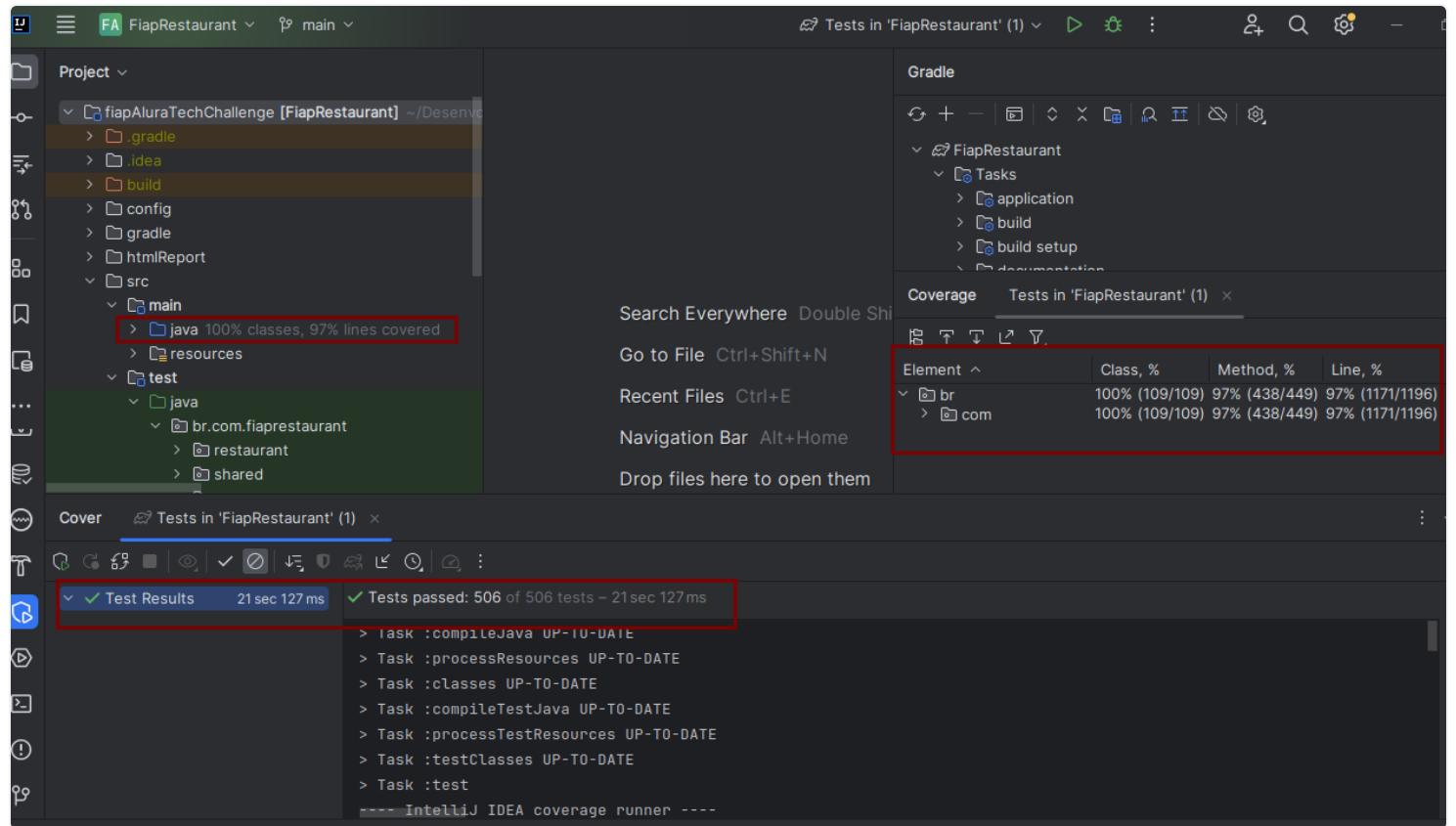
Qualidade de software

Para garantir a qualidade de software, implementamos testes de unidade e de integração na grande maioria do código. Para identificar o que foi testado, utilizamos a cobertura de testes de código do próprio IntelliJ IDEA. A decisão de utilizar o próprio IntelliJ foi motivada pela manutenção de menor número de dependências a serem adicionadas no projeto, com o objetivo de reduzir possibilidades de libs externas abrirem uma fragilidade na segurança da aplicação (lembrando do caso do Log4J) e que no cenário em que o projeto foi desenvolvido não foi necessária a adição do Jacoco.

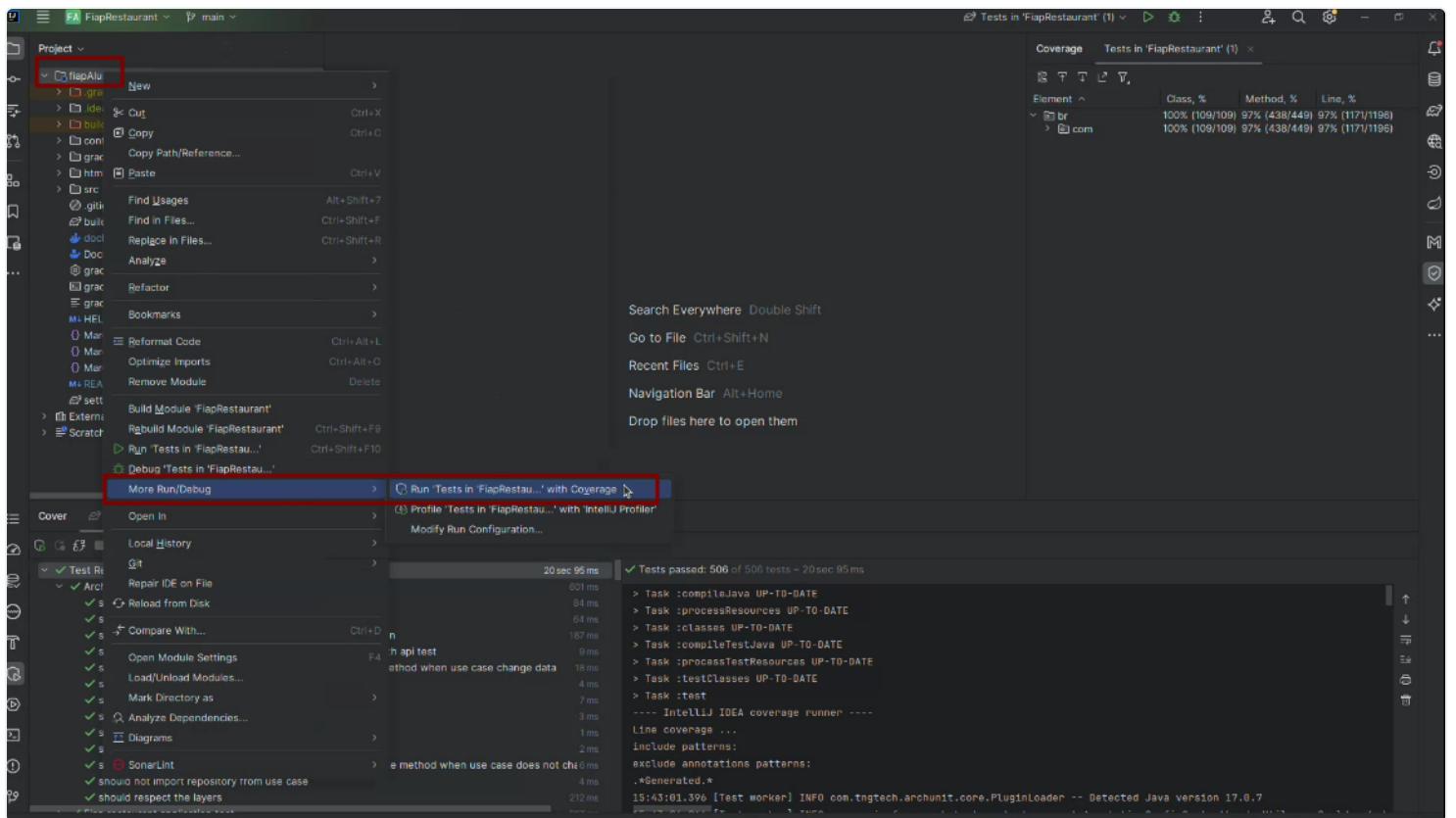
No entanto, foi adicionada a dependência do RestAssured para realizar testes, conforme aprendido na disciplina de qualidade de software. Esta foi utilizada em apenas duas classes de teste para avaliação. Contudo, na grande maioria, foram utilizadas as classes do Spring Framework Test para realizar as chamadas dos endpoints, realizar assertivas diretamente com o response capturado e realizar assertivas com o AssertJ.

Os testes de **unidade** foram implementados nas classes entidades, value objects, services, validators e use cases, testando a menor unidade de código. Os testes de **integração** foram implementados nas classes de presentation, realizando a requisição REST as endpoints em diversos cenários, testando o código por completo, da entrada dos dados, processamento e saída. O objetivo desta segregação foi considerar a eficiência dos testes versus o tempo de entrega do projeto.

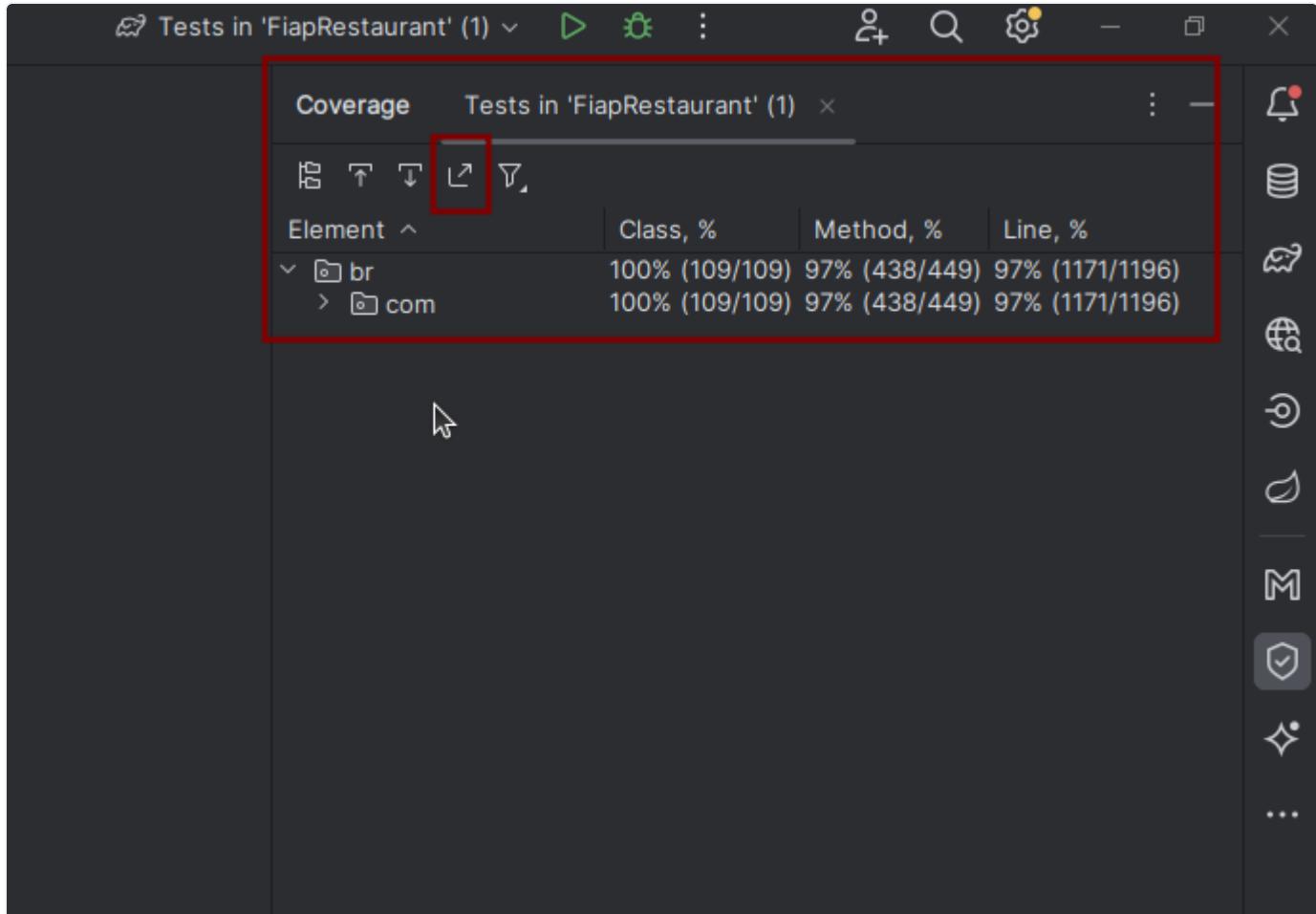
Aplicando este método, foi apurado pela cobertura de testes do IntelliJ IDEA, 97% de linhas de código testadas, sendo **269 de testes de unidade** e **237 de testes de integração**, totalizando **506 métodos de testes realizados**, para os domínios de usuário (cadastro e autenticação) e restaurantes (cadastro de restaurante, reserva de mesa, gerenciamento de reservas e avaliação do restaurante).



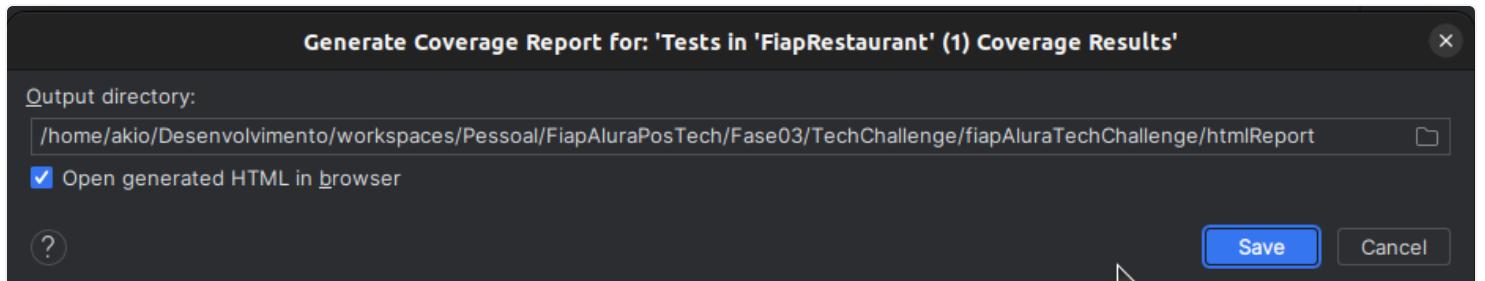
Para realizar o teste de cobertura, clique com o botão direito do mouse sobre o nome do projeto, navegue até a opção More Run/Debug, em seguida selecione a opção Run tests in <nome do projeto> with Coverage.



Ao final da execução, os resultados serão apresentados conforme a figura abaixo. Contudo, se houver o interesse em visualizar os resultados em uma janela do navegador, basta clicar no ícone destacado também na figura abaixo.



A IDE questionará o local em que o resultado deverá ser salvo.



Abaixo, o resultado no navegador.

Coverage Report > Summary

File /home/akio/Desenvolvimento/workspaces/Pessoal/FiapAluraPosTech/Fase03/TechChallenge/fiapAluraTechChallenge/htmlReport/index.html

Softplan Tech FIAP

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	100% (109/109)	97.4% (450/462)	97.8% (1171/1197)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
br.com.fiaprestaurant	100% (1/1)	66.7% (2/3)	66.7% (2/3)
br.com.fiaprestaurant.restaurant.application.event	100% (1/1)	100% (1/1)	100% (1/1)
br.com.fiaprestaurant.restaurant.domain.entity	100% (4/4)	100% (52/52)	100% (119/119)
br.com.fiaprestaurant.restaurant.domain.valueobject	100% (8/8)	100% (48/48)	100% (150/150)
br.com.fiaprestaurant.restaurant.infrastructure.event	100% (2/2)	100% (4/4)	100% (8/8)
br.com.fiaprestaurant.restaurant.infrastructure.gateway	100% (3/3)	100% (31/31)	100% (124/124)
br.com.fiaprestaurant.restaurant.infrastructure.mailer	100% (4/4)	100% (11/11)	95.3% (41/43)
br.com.fiaprestaurant.restaurant.infrastructure.schema	100% (9/9)	97.7% (43/44)	98.4% (60/61)
br.com.fiaprestaurant.restaurant.infrastructure.usecase	100% (11/11)	100% (22/22)	100% (92/92)
br.com.fiaprestaurant.restaurant.infrastructure.validator	100% (5/5)	100% (10/10)	100% (41/41)
br.com.fiaprestaurant.restaurant.presentation.api	100% (3/3)	100% (16/16)	100% (50/50)
br.com.fiaprestaurant.restaurant.presentation.dto	100% (9/9)	100% (21/21)	100% (43/43)
br.com.fiaprestaurant.shared.domain.entity	100% (3/3)	86.7% (13/15)	84.2% (16/19)
br.com.fiaprestaurant.shared.domain.exception	100% (1/1)	100% (2/2)	100% (3/3)
br.com.fiaprestaurant.shared.infrastructure.exception	100% (2/2)	100% (4/4)	100% (6/6)
br.com.fiaprestaurant.shared.infrastructure.security	100% (1/1)	100% (1/1)	100% (2/2)
br.com.fiaprestaurant.shared.infrastructure.validator	100% (2/2)	100% (4/4)	100% (10/10)
br.com.fiaprestaurant.shared.presentation.configure.jackson	100% (1/1)	100% (2/2)	100% (9/9)

Para visualizar a cobertura de uma classe específica, basta selecioná-la na lista apresentada.

Coverage Report > Create

File /home/akio/Desenvolvimento/workspaces/Pessoal/FiapAluraPosTech/Fase03/TechChallenge/fiapAluraTechChallenge/htmlReport/ns-9/sources/source-4....

Softplan Tech FIAP

Current scope: all classes | br.com.fiaprestaurant.restaurant.infrastructure.usecase

Coverage Summary for Class: CreateRestaurantInteractor (br.com.fiaprestaurant.restaurant.infrastructure.usecase)

Class	Method, %	Line, %
CreateRestaurantInteractor	100% (2/2)	100% (5/5)
CreateRestaurantInteractor\$\$SpringCGLIB\$\$0		
Total	100% (2/2)	100% (5/5)

```

1 package br.com.fiaprestaurant.restaurant.infrastructure.usecase;
2
3 import br.com.fiaprestaurant.restaurant.application.gateways.RestaurantGateway;
4 import br.com.fiaprestaurant.restaurant.application.usecase.CreateRestaurantUseCase;
5 import br.com.fiaprestaurant.restaurant.application.validator.RestaurantAlreadyRegisteredByCnpjValidator;
6 import br.com.fiaprestaurant.restaurant.domain.entity.Restaurant;
7 import org.springframework.stereotype.Service;
8 import org.springframework.transaction.annotation.Transactional;
9
10 @Service
11 public class CreateRestaurantInteractor implements CreateRestaurantUseCase {
12
13     private final RestaurantGateway restaurantGateway;
14     private final RestaurantAlreadyRegisteredByCnpjValidator restaurantAlreadyRegisteredByCnpjValidator;
15
16     public CreateRestaurantInteractor(RestaurantGateway restaurantGateway,
17             RestaurantAlreadyRegisteredByCnpjValidator restaurantAlreadyRegisteredByCnpjValidator) {
18         this.restaurantGateway = restaurantGateway;
19         this.restaurantAlreadyRegisteredByCnpjValidator = restaurantAlreadyRegisteredByCnpjValidator;
20     }
21
22     @Transactional
23     public Restaurant execute(Restaurant restaurant) {
24         restaurantAlreadyRegisteredByCnpjValidator.validate(restaurant.getCnpj().getCnpjValue());
25         return restaurantGateway.save(restaurant);
26     }
27 }

```

Para garantir a qualidade do design do software, foi realizado um teste específico (ArchUnitTest), para certificar-se de que as classes sejam implementadas respeitando cada camada da "cebola"

da Clean Architecture. No exemplo abaixo, temos o teste "shouldRespectTheLayers". O funcionamento deste teste é baseado em convenção de nomenclatura das classes e pacotes. A classe com o sufixo "Presentation" em seu nome deve estar contida em um pacote de nome presentation, e assim por diante.

The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the file structure of the 'fiapAluraTechChallenge [FiapRestaurant]' project. It includes .gradle, .idea, build, config, gradle, htmlReport, src (containing main and test), and test (containing java, br.com.fiaprestaurant, restaurant, shared, user, ArchUnitTest, and FiapRestaurantApplicationTest). Other files like .gitignore, build.gradle, docker-compose.yml, Dockerfile, gradle.properties, gradlew, gradlew.bat, HELP.md, README.md, settings.gradle, External Libraries, and Scratches and Consoles are also listed. On the right, the Editor tool window shows the content of 'ArchUnitTest.java'. The code defines a class 'ArchUnitTest' with methods for initializing imports and testing layer dependencies. An annotation '@Test' is present above a method named 'shouldRespectTheLayers'. The code uses ArchUnit annotations like 'layer' and 'mayOnlyBeAccessedByLayers' to specify layer boundaries and dependencies.

```
package br.com.fiaprestaurant;
import ...;
class ArchUnitTest {
    private static JavaClasses mainPackages;
    private static JavaClasses testPackages;
    @BeforeAll
    static void init() {
        mainPackages = new ClassFileImporter().withImportOption(new ImportOption.DoNotIncludeTests())
            .importPackages(ArchUnitTest.class.getPackage().getName());
        testPackages = new ClassFileImporter().withImportOption(new ImportOption.OnlyIncludeTests())
            .importPackages(ArchUnitTest.class.getPackage().getName());
    }
    @Test
    void shouldRespectTheLayers() {
        layeredArchitecture().consideringAllDependencies()
            .layer(name: "Presentation").definedBy(...packagIdentifiers: "...presentation..")
            .layer(name: "Application").definedBy(...packagIdentifiers: "...application..")
            .layer(name: "Domain").definedBy(...packagIdentifiers: "...domain..")
            .layer(name: "Infrastructure").definedBy(...packagIdentifiers: "...infrastructure..")
            .whereLayer(name: "Presentation").mayOnlyBeAccessedByLayers(...layerNames: "Infrastructure")
            .whereLayer(name: "Application").mayOnlyBeAccessedByLayers(...layerNames: "Infrastructure", "Presentation")
            .whereLayer(name: "Domain") LayerDependencySpecification
            .mayOnlyBeAccessedByLayers(...layerNames: "Presentation", "Application", "Infrastructure") LayeredArchitecture
            .check(mainPackages);
    }
}
```

Temos também, teste para verificar se as classes de use case, validators, service e demais possuem sua classe de teste equivalente. No caso de classes de presentation/controller, cada método que representa um endpoint deve possuir uma classe de teste equivalente. Outro teste contido no ArchUnitTeste é a verificação de classe que implementa um use case, que realiza persistência em banco de dados, deve ser anotada com @Transactional. Entre outros testes que foram implementados para garantir a qualidade do design de software.

Para facilitar a implementação dos nomes dos testes, bem como sua leitura e compreensão, foi criada uma classe **ReplaceCamelCase** para que o nome do método do teste seja recuperado e normalizado. Para tanto, configuramos o JUnit para executar esta classe no arquivo [junit-platform.properties](#). Por exemplo, veja na figura a baixo. Temos um teste chamado **shouldRespectTheLayers()**. Com esta configuração e implementação, a janela dos Tests Results apresentou o nome do teste como "**should respect the layers**". Assim, não precisamos digitar o nome do método de teste e depois acrescentar uma anotação **@DisplayName** e informar qual o nome deve ser exibido para este teste na janela de Testes Results.

The screenshot shows the Android Studio interface with the following components:

- Project View:** Shows the project structure under "flapAluraTechChallenge [FlapRestaurant]".
- Code Editors:**
 - junit-platform.properties:** Contains the line `junit.jupiter.displayname.generator.default=br.com.flaprestaurant.shared.Junit.ReplaceCamelCase` highlighted with a red box.
 - ReplaceCamelCase.java:** A Java class with two methods: `generateDisplayNameForClass` and `generateDisplayNameForNestedClass`, both annotated with `@Override`.
 - ArchUnitTest.java:** An ArchUnit test class with two test methods: `shouldRespectTheLayers` and `shouldNotExistClassesThatArentUseCases`.
- Run Tab:** Shows the "ArchUnitTest" configuration selected.
- Test Results:** A detailed view of the test results, showing 13 passed tests with execution times ranging from 6ms to 211ms.
- Output Tab:** Displays the build logs, including tasks like `:compileJava`, `:processResources`, and `:test`, and a message indicating the Java version is 17.0.7.

Além dos testes automatizados, foram realizados testes manuais com o Postman, onde foram cadastradas requisições para cada cenário de teste.

- ✓ Fiap/Alura - Tech Challenge - Fase 03
 - > 00 - Authenticate
 - > 01 - User
 - ✓ 02 - Restaurant
 - > Create
 - > Get restaurant by name/coordinates/type of cuisine
 - > Update
 - > Get restaurant by ID
 - > Delete
 - ✓ 03 - Restaurant Review
 - > Create
 - > Get restaurant reviews
 - ✓ 04 - Restaurant booking
 - > Create
 - > Get restaurant booking
 - > Cancel restaurant booking
 - > Close restaurant booking

Por exemplo, para o cadastro de restaurantes. Foram criadas requisições para cada cenário de entrada de dados. Assim, temos uma requisição para o endpoint de cadastro de um novo restaurante, com todos os dados de entrada válidos. Outro endpoint para o cadastro de um novo restaurante, mas com o número de CNPJ já cadastrado na base e que espera-se o http status de retorno Conflict. Outro endpoint para o cadastro de restaurante, mas com o nome com quantidade superior ao permitido, onde espera-se o http status de retorno Bad Request. Assim por diante. Segue abaixo uma figura com o exemplo citado.

- ✓ Fiap/Alura - Tech Challenge - Fase 03
 - ✓ 00 - Autenticate
 - POST 01** - POST authenticate
 - > 01 - User
 - ✓ 02 - Restaurant
 - ✓ Create
 - POST 01** - Return created status when create new restaurant
 - POST 02** - Return conflict status when the restaurant's CNPJ already exists in another restaurant
 - POST 03** - Return bad request status when the restaurant's name is bigger than 100 characters
 - POST 04** - Return bad request status when the restaurant's name is null
 - POST 05** - Return bad request status when the restaurant's name is empty
 - POST 06** - Return bad request status when the restaurant's CNPJ is invalid
 - POST 07** - Return bad request status when the restaurant's CNPJ is null
 - POST 08** - Return bad request status when the restaurant's CNPJ is empty
 - POST 09** - Return bad request status when the restaurant's Open At is invalid
 - POST 10** - Return bad request status when the restaurant's Open At is null
 - POST 11** - Return bad request status when the restaurant's Open At is empty
 - POST 12** - Return bad request status when the restaurant's Close At is invalid
 - POST 13** - Return bad request status when the restaurant's Close At is null
 - POST 14** - Return bad request status when the restaurant's Close At is empty
 - POST 15** - Return bad request status when the restaurant's people capacity is negative
 - POST 16** - Return bad request status when the restaurant's people capacity is null
 - POST 17** - Return bad request status when the restaurant's type of cuisine is null
 - POST 18** - Return bad request status when the restaurant's type of cuisine is empty
 - POST 19** - Return bad request status when the restaurant's type of cuisine is bigger than 50 characters
 - POST 20** - Return bad request status when the restaurant's street is null
 - POST 21** - Return bad request status when the restaurant's street is empty
 - POST 22** - Return bad request status when the restaurant's street is bigger than 255 characters
 - POST 23** - Return bad request status when the restaurant's number is null
 - POST 24** - Return bad request status when the restaurant's number is empty
 - POST 25** - Return bad request status when the restaurant's number is bigger than 100 characters

A criação de cada uma destas requests para testes manuais levou em consideração o core da aplicação, ou seja, o subdomínio de restaurantes (cadastro, reserva e gerenciamento de mesas e avaliação), devido ao grande esforço aplicado e o tempo para a entrega do projeto.

Staff

O desenvolvimento da aplicação Fiap Restaurant envolve a seguinte equipe:

- **Marcelo Akio Nishimori**
 - **Cargo:** Software Developer
 - **Email:** mnishimori@yahoo.com.br
 - **RM - 350802**