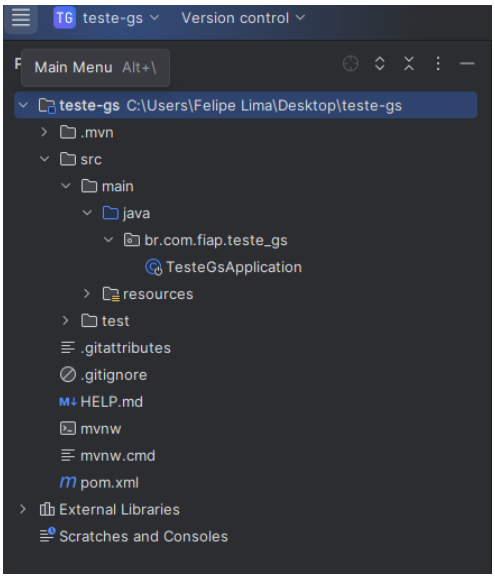
Cola da Samara – Micro Services

```
Java
        -> 'nome_do_projeto'
                -> controller
                        -> handelers
                                -> ControllerExceptionHandelers
                        -> Objeto1Controller
                        -> Objeto2Controller

                -> dto
                        -> Objeto1DTO
                        -> Objeto2DTO

                -> model
                        -> Objeto1
                        -> Objeto2
                -> repository
                        -> Objeto1Repository
                        -> Objeto2Repository
                -> service
                        -> exception
                                -> DatabaseExcepition
                                -> ResourceNotFoundException

Resource
        -> application.proprities
        -> application-teste
        -> import.sql
```

Passo a passo

1) Iniciar o projeto – Gerar no spring initializr
    a. Dependências:
        i. Spring Web
        ii. Spring Data JPA
        iii. H2 Database
        iv. Validation
        v. Lombok
        vi. Spring boot dev tools



2) Criar os models do projeto
    a. Não esquecer das anotações nos models:
        i. `@AllArgsConstructor`
        ii. `@NoArgsConstructor`
        iii. `@Getter`
        iv. `@Setter`
    b. Inserir a anotação no ID
        i. `@Id`
        ii. `@GeneratedValue(strategy = GenerationType.IDENTITY)`

c. Inserir anotações ROM na CLASSE
   i. @Entity
   ii. @Table(name = "tb_venda")

d. Fazer os relacionamentos corretamente

i.
```
@ManyToOne
@JoinColumn(name = "vendedor_id")
private Vendedor vendedor;
```

ii.
```
@OneToMany(mappedBy = "vendedor")
private List<Venda> vendas = new ArrayList<>();
```

3) Implementer application proprieties

a. PRINCIPAL

i.
```
spring.application.name=api-gestaofrota

spring.profiles.active=test
spring.jpa.open-in-view=false
```

b. TESTE

i.
```
# para expor o trace - somente para testes
# server.error.include-stacktrace = always
server.error.include-message=always

# para não expor o trace
server.error.include-stacktrace = never
# server.error.include-message=never

# Conexão com o banco H2
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

# H2 Client
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# JPA, SQL
spring.jpa.database-
platform=org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization=true
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# http://localhost:8080/h2-console/
```

4) Criar o import.sql

5) Implementar o repository (interfaces)

a.
```
extends JpaRepository<Venda, Long>
```

6) Criar os DTO
    a. Implementar as anotações

```
@AllArgsConstructor
@NoArgsConstructor
@Getter
```

    b. Inserir as restrições de cada campo (@notnull, @size)
    c. ‘Contrutor’

7) Crias as Services
    a. Adicionar a anotação

```
@Service
```

    b. Iniciar repository

    c. Fazer findAll
        i. Já posso fazer o controller (findAll())
    d. Fazer findById
    e. Fazer Insert
    f. Fazer update
    g. Fazer delete
    h. Fazer copyToEntity
        i. Iniciar o repository do outro objeto no relacionamento

8) Criar os Controller
    a. Adicionar as anotações

```
@RestController
@RequestMapping(value = "/vendas")
```

    b. Iniciar o service

```
@Autowired
private VendaService service;
```

    c. Fazer o getMapping para o findAll()
        i. Já posso testar no POSTMAN o http://localhost:8080/vendas ou http://localhost:8030/vendas

    d. Fazer o postMapping para o insert()
    e. Fazer o putMapping para o update()
    f. Fazer o deleteMapping para o delete()

9) Fazer as Exceções dos Services
    a. Dentro da package services criar uma de expection
    b. Criar a class ResourceNotFoundException
        i. Anotação
    c. Criar a class CustomErrorDTO
        i. Anotação
    d. Criar a package handlers dentro da controller
    e. Criar a class ControllerExceptionHandler
        i. Anotação
    f. Criar a class FieldMessageDTO dentro do package dto
        i. Anotação
    g. Criar a class ValidationErrorDTO
        i. Anotação
    h. Criar a class DatabaseException dentro do package exception
        i. Anotação

10) Criar documentação
    a. Adicionar dependência no pom.xml

```
b. <!--Swagger-->
c. <dependency>
d.     <groupId>org.springdoc</groupId>
e.     <artifactId>springdoc-openapi-starter-webmvc-
   ui</artifactId>
f.     <version>2.6.0</version>
g. </dependency>
```

    h. Testar
        i. http://localhost:8080/swagger-ui/index.html

    i. Adicionar Tags nos controller

```
i. @Tag(name = "Vendas", description = "Controller para
   Vendas")
```

    j. Adicionar anotações nos recursos (findAll, findById etc...)

```
i.  @Operation(
       description = "Listar vendas",
           summary = "Retorna uma lista de vendas",
           responses = {
               @ApiResponse(description = "tudo certo
moreno", responseCode = "200")
           }
)
```

    k. Adicionar no getMapping o produces = "application/json"

l. Adicionar no getMapping (findById) o produces = MediaType.*APPLICATION_JSON_VALUE*

m. Colocar @schema nos DTO (os que tem MODEL)

```
i. @Schema(description = "ID da venda gerada pelo banco de dados")
```

11) Adicionando MODEL MAPPER

a. Adicionar a dependência do model mapper

```
i.  <!--
    https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->
ii. <dependency>
iii.     <groupId>org.modelmapper</groupId>
iv.      <artifactId>modelmapper</artifactId>
v.       <version>3.2.1</version>
vi. </dependency>
```

b. Criar uma classe ConfigModelMapper dentro de uma package chamada config

c. Adicionar anotação (@Configuration) e (@RequiredArgsConstructor)

d. Adicionar o ModelMapper no retorno dos métodos das SERVICES

```
i.  @Transactional(readOnly = true)
    public List<VendedorDTO> findAll() {
        return repository.findAll().stream()
                .map(vendedor ->
    modelMapper.map(vendedor,
    VendedorDTO.class)).collect(Collectors.toList());
    }

ii. @Transactional(readOnly = true)
    public VendedorDTO findById(Long id) {
        Vendedor entity =
    repository.findById(id).orElseThrow(
                () -> new
    ResourceNotFoundException("Recurso não encontrado!
    Id: " + id)
        );

        return modelMapper.map(entity,
    VendedorDTO.class);
    }

iii.
```

Observações

1) Consumo de API - Aula_32_Microsservicos_Exceptions.pdf
2) Regra de negocio na COPYTOENTITY da SERVICE