**🏗️ Estrutura geral**

rest-with-spring-boot-and-java-erudito

│

├── src

│ ├── main

│ │ ├── java

│ │ │ └── giovanycesar.com.github

│ │ │ ├── controllers

│ │ │ ├── exception

│ │ │ ├── model

│ │ │ ├── repository

│ │ │ ├── services

│ │ │ └── Startup.java

│ │ └── resources

│ │ └── application.yml

│ └── test

│ ├── java

│ └── resources

│

└── pom.xml

**📂 Pacotes e classes Java**

**🧭 Startup.java**

* É a **classe principal** do projeto Spring Boot.
* Contém a anotação @SpringBootApplication.
* Serve como **ponto de entrada**: quando você clica em “Run”, é ela que inicializa o servidor embutido (geralmente o Tomcat).
* Exemplo:
* @SpringBootApplication
* public class Startup {
* public static void main(String[] args) {
* SpringApplication.run(Startup.class, args);
* }
* }

**🧩 controllers → PersonController.java**

* Camada **Controller**, responsável por receber requisições HTTP e devolver respostas.
* Aqui você define os **endpoints da API** (/person, /person/{id}, etc.).
* Usa anotações como:
  + @RestController → indica que a classe é um controlador REST.
  + @GetMapping, @PostMapping, @PutMapping, @DeleteMapping → mapeiam os métodos HTTP.
* Ele chama os métodos do **Service** (regras de negócio).

**⚙️ services → PersonServices.java**

* Camada de **serviço**, onde ficam as **regras de negócio**.
* O controller chama o service para executar ações, e o service interage com o repositório.
* Exemplo: criar, buscar, atualizar ou excluir uma pessoa.
* Contém a anotação @Service.

**🗄️ repository → PersonRepository.java**

* Camada de **acesso a dados** (DAO).
* Interface que estende JpaRepository ou CrudRepository.
* O Spring Data JPA cria automaticamente os métodos para interagir com o banco (ex: findAll(), save(), deleteById()).
* Exemplo:
* public interface PersonRepository extends JpaRepository<Person, Long> {}

**👤 model → Person.java**

* Representa a **entidade** do banco de dados.
* Mapeia a tabela (ex: @Entity, @Table(name = "person")).
* Cada atributo representa uma coluna (ex: @Id, @GeneratedValue, @Column).
* Exemplo:
* @Entity
* public class Person {
* @Id
* @GeneratedValue(strategy = GenerationType.IDENTITY)
* private Long id;
* private String firstName;
* private String lastName;
* private String address;
* private String gender;
* }

**🚨 exception**

Contém classes para tratar erros personalizados.

**ResourceNotFoundException.java**

* Exceção lançada quando um recurso (por exemplo, uma pessoa) não é encontrado.
* Geralmente usada no service.findById().

**ExceptionResponse.java**

* Modelo (objeto) usado para retornar mensagens de erro em formato JSON.

**CustomEntityResponseHandler.java**

* Classe com @ControllerAdvice que **intercepta exceções** lançadas na aplicação e retorna mensagens customizadas.

**🧾 application.yml**

* Arquivo de configuração do Spring Boot.
* Define propriedades como:
  + Porta do servidor
  + Conexão com o banco de dados
  + Formato de logs, etc.
* Exemplo:
* server:
* port: 8080
* spring:
* datasource:
* url: jdbc:mysql://localhost:3306/testdb
* username: root
* password: root
* jpa:
* hibernate:
* ddl-auto: update

**🧪 test**

* Diretório para **testes automatizados** (JUnit).
* O Spring Boot cria uma estrutura paralela à de main para organizar testes unitários ou de integração.

**⚙️ pom.xml**

* Arquivo de configuração do **Maven**, que gerencia as dependências do projeto.
* Define bibliotecas, plugins e versões do Spring Boot.
* Exemplo:
* <dependencies>
* <dependency>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-web</artifactId>
* </dependency>
* <dependency>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-data-jpa</artifactId>
* </dependency>
* <dependency>
* <groupId>com.h2database</groupId>
* <artifactId>h2</artifactId>
* <scope>runtime</scope>
* </dependency>

</dependencies>

Spring Boot: o **tratamento global de exceções**.

Vamos destrinchar a sua classe linha por linha pra entender **quem retorna o JSON de erro** e **como isso acontece** 👇

## 💡 O que essa classe faz

@ControllerAdvice

@RestController

public class CustomEntityResponseHandler extends ResponseEntityExceptionHandler {

### 🔹 @ControllerAdvice

Diz ao Spring Boot:

“Essa classe vai **interceptar exceções lançadas por qualquer controller** do sistema.”

Ou seja, se o PersonController (ou outro controller) lançar um erro, o Spring **não vai quebrar a aplicação** — ele vai **mandar o erro pra essa classe tratar**.

### 🔹 @RestController

Permite que essa classe **retorne objetos JSON**, igual um controller comum.

## ⚙️ Quem retorna o JSON

Os **métodos anotados com @ExceptionHandler** são os responsáveis por **gerar o JSON de erro**.

### 1️⃣ Método geral:

@ExceptionHandler(Exception.class)

public final ResponseEntity<ExceptionResponse> handleAllExceptions(Exception ex, WebRequest request) {

ExceptionResponse response = new ExceptionResponse(

new Date(),

ex.getMessage(),

request.getDescription(false)

);

return new ResponseEntity<>(response, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

👉 Esse método é chamado **quando ocorre qualquer exceção genérica**, como NullPointerException, IllegalArgumentException, etc.

Ele cria um objeto ExceptionResponse (seu modelo de erro personalizado) e devolve no corpo da resposta HTTP.

🔹 **HTTP Status:** 500 – Internal Server Error  
🔹 **Corpo da resposta (JSON):**

{

"timestamp": "2025-10-25T18:41:53.123+00:00",

"message": "Null pointer exception",

"details": "uri=/person/99"

}

### 2️⃣ Método específico:

@ExceptionHandler(ResourceNotFoundException.class)

public final ResponseEntity<ExceptionResponse> handleNotFoundExceptions(Exception ex, WebRequest request) {

ExceptionResponse response = new ExceptionResponse(

new Date(),

ex.getMessage(),

request.getDescription(false)

);

return new ResponseEntity<>(response, HttpStatus.NOT\_FOUND);

}

👉 Esse método é chamado **somente quando é lançada uma ResourceNotFoundException** — por exemplo, no seu PersonServices:

return repository.findById(id)

.orElseThrow(() -> new ResourceNotFoundException("No records found for this ID."));

🔹 **HTTP Status:** 404 – Not Found  
🔹 **Corpo da resposta (JSON):**

{

"timestamp": "2025-10-25T18:42:31.456+00:00",

"message": "No records found for this ID.",

"details": "uri=/person/10"

}

## 🧱 O modelo ExceptionResponse

Essa é a classe que define **o formato do JSON de erro**:

public class ExceptionResponse {

private Date timestamp;

private String message;

private String details;

// construtor + getters

}

Ou seja, **quem “gera” o JSON é o Spring Boot**, mas **quem define o conteúdo e formato é o ExceptionResponse**, e **quem decide qual erro tratar é o CustomEntityResponseHandler**.

## ✅ Resumo da responsabilidade

| **Elemento** | **Função** |
| --- | --- |
| @ControllerAdvice | Detecta exceções lançadas em qualquer controller |
| @ExceptionHandler | Define qual tipo de erro o método vai tratar |
| ExceptionResponse | Modelo que forma o JSON de erro |
| ResponseEntity | Embala o JSON + status HTTP (404, 500 etc.) |
| CustomEntityResponseHandler | Classe que centraliza todo o tratamento de erros da API |

## 💡 O que é @RestController

A anotação:

@RestController

é uma **combinação de duas anotações** do Spring:

@Controller

@ResponseBody

Ou seja, ela transforma a classe em um **controlador REST** que:

1. É reconhecida como um controller normal do Spring (@Controller);
2. E **retorna os dados diretamente no corpo da resposta HTTP**, em vez de renderizar uma página HTML (@ResponseBody).