# Roteiro
# Aula Prática

# Programação Web

## OBJETIVOS

**Definição dos objetivos da aula prática:**

- Criar projeto Spring Boot Java
- Implementar modelo de domínio
- Estruturar camadas lógicas: resource, service, repository
- Configurar banco de dados de teste (H2)
- Povoar o banco de dados
- CRUD - Create, Retrieve, Update, Delete
- Tratamento de exceções

## INFRAESTRUTURA

**Instalações:**

Spring Tools 4. Postman.

**Materiais de consumo:**

| **Descrição:** Não se aplica. | Quantidade de materiais por procedimento/atividade |
|---|---|
| | |
| | |
| | |

**Software:**

**Sim ( x ) Não ( )**

**Em caso afirmativo, qual?**

**Pago ( ) Não Pago ( x )**

**Tipo de Licença: community**

**Descrição do software:**

O Spring Tool Suite é uma IDE baseada em Eclipse, sendo utilizada para realizar projetos utilizando Spring Boot. Spring Tool se trata de uma IDE que já vem com os plugins da Spring que facilitam o desenvolvimento com o ecosistema spring.

Postman é um aplicativo usado para testes de API. É um *client HTTP* que testa requisições HTTP, utilizando uma interface gráfica com o usuário, através da qual obtemos diferentes tipos de respostas que precisam ser posteriormente validadas.

**Equipamento de Proteção Individual (EPI):**

- NSA

## PROCEDIMENTOS PRÁTICOS

## Atividade proposta:

Criar um projeto utilizando o framework Spring para o gerenciamento de usuários de um sistema. Se tratará de um projeto backend baseado em Rest API, ou seja, serão disponibilizados os endpoints necessários para realizar as operações de gerenciamento de usuários (CRUD). O projeto será MVC e será necessário realizar o modelo logico para a criação da base de dados, bem como a conexão com o mesmo. Além da logica base, será necessário realizar os tratamentos de exceções.

## Procedimentos para a realização da atividade:

Para a realização desta aula pratica você irá precisar utilizar um editor de texto, recomendável o Eclipse com Spring Tool: https://spring.io/tools.

Ademais, você precisará:

- Ter o Postman instalado: https://www.postman.com/downloads/
- Ter Git instalado: https://git-scm.com/downloads (caso queira versionar o seu projeto e/ou salvar na nuvem)

1. Crie o projeto base com a estrutura Spring a partir do seguinte link: **https://start.spring.io/**
    - Selecione as seguintes opções:
        - Maven
        - Java, versão 17
        - Versão do Spring: 3.0.0
        - JAR *packaging*
        - *Add dependencies*: Spring Web
2. Clique em "*generate*". Descompacte o projeto que foi baixado no seu *workspace*.
3. Abra o Spring Tool Suite e escolha seu *workspace*.
    - Em "*File > Import > Maven > Existing Maven projects*": escolha a pasta do projeto base gerado nos passos 1 e 2. Selecione o arquivo pom.xml que aparecerá e clique em "*Finish*".
4. Em src/main/java, crie um pacote "entities" e dentro dele crie a classe User.
    - Crie os atributos básicos: id, nome, e-mail, telefone, *password*.
    - Crie um construtor vazio e um construtor usando todos os atributos (campos).
    - Crie os *getters* e *setters*.
    - Gere a implementação para o método hashCode() e equals() (botão direito > *source* > *generate hashCode*...).
    - Faça com que sua classe implemente a interface *Serializable* e crie o atributo serialVersionUID (ele vai ser útil caso seja necessário salvar o objeto em um arquivo etc.)

- Adicione a esta classe as seguintes *annotations*:
    - `@Entity`
    - `@Table(name = "tb_user")`

5. Em src/main/java, crie um pacote "*resource*" e dentro dele crie a classe *UserResource* (esta classe ira ser o nosso controller)
    - Adicione as seguintes *annotations*:
        - @RestController
        - @RequestMapping(value = "/users")
6. Nosso banco de dados será o H2 que é um banco de dados relacional que pode ser executado no modo cliente-servidor. Iremos utilizar JPA para persistir os dados. Vamos então configurar nosso projeto para incluir a dependência destes dois recursos:
    - Para isto adicione as seguintes dependências no arquivo pom.xml:

```
<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>
```

7. Em src/main/resources, crie o arquivo *application.properties* (ele nos servirá para armazenar propriedades de escopo do aplicativo):
    - Adicione as seguintes linhas:

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

8. Em src/main/resources, crie o arquivo *application-test.properties* (ele nos servirá para armazenar propriedades de escopo do aplicativo para a realização dos testes locais):
    - Adicione as seguintes linhas:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

9. Em src/main/java, crie um pacote "*repositories*" e dentro dele crie a interface *UserRepository.*
    - Estenda a classe JpaRepository e passe como definição ao JpaRepository a entidade "User" e o tipo da chave primaria (id):
        - JpaRepository<User, Long>
10. Crie uma classe de configuração que irá rodar toda vez que executarmos o projeto no modo teste. Ela irá nos servir para popular alguns dados no banco em tempo de execução (o H2 é um banco de dados em memória, portanto quando a aplicação for finalizada seus dados deixarão de existir).

- Em src/main/java, crie um pacote "*config*" e dentro dele crie a classe *TestConfig* que irá implementar a interface *CommandLineRunner.*
- Vamos injetar um atributo do tipo *UserRepository* à esta classe utilizando a *annotation @Autowired*.
- Implemente o método ***run,*** adicionando as seguintes linhas:

```
User u1 = new User(null, "Maria Brown", "maria@gmail.com", "988888888", "123456");
User u2 = new User(null, "Alex Green", "alex@gmail.com", "977777777", "123456");

userRepository.saveAll(Arrays.asList(u1, u2));
```

11. Em src/main/java, crie um pacote "*services*" e dentro dele crie a interface *UserService,* utilize nesta classe a *annotation @Service.*
    - Injete um atributo do tipo *UserRepository* à esta classe utilizando a *annotation @Autowired*.
    - Crie os seguintes métodos e implemente cada um deles:
        - *findAll*
        - *findById*
        - *insert*
        - *delete*
        - *update*
12. Em src/main/java/resources, na classe *UserResource,* injete um atributo do tipo *UserService* à esta classe utilizando a *annotation @Autowired*. Em seguida, implemente as seguintes rotas:
        - *findAll (GET)*
        - *findById (GET)*
        - *insert (POST)*
        - *delete (DELETE)*
        - *update (PUT)*
13. Em src/main/java/service, crie um pacote "*exceptions*" e dentro dele crie a classe *ResourceNotFoundException.*
    - Estenda a classe *RuntimeException.*
    - Implemente o construtor da classe fazendo a chamada para o construtor pai, passando uma mensagem personalizada.
14. Em src/main/java/resources, crie um pacote "*exceptions*" e dentro dele crie as seguintes classes:
    - *StandardError.*
    - *ResourceExceptionHandler.*
15. Em *StandardError:*
    - Crie os atributos básicos: *timestamp*, *status, error, message, path*.
    - Crie um construtor vazio e um construtor usando todos os atributos (campos).
    - Crie os *getters* e *setters*.
    - Faça com que sua classe implemente a interface *Serializable* e crie o atributo serialVersionUID (ele vai ser útil caso seja necessário salvar o objeto em um arquivo etc.)
16. Em *ResourceExceptionHandler:*
    - Adicione a seguinte anotação à classe: *@ControllerAdvice.* Vai servir pra interceptar a exceção lançada quando não for encontrado um usuário.

- Crie o método *resourceNotFound:* ele retornará um objeto do tipo *ResponseEntity<StandardError>* e deverá receber como parâmetro o objeto *ResourceNotFoundException.*
- Adicione a anotação *@ExceptionHandler(ResourceNotFoundException.class)* ao método criado.
- Na implementação do método, além de passar uma mensagem customizada, é interessante mudar o código de erro da exceção, passando o tipo `HttpStatus status = HttpStatus.NOT_FOUND;`

17. Em src/main/java/services, na classe *UserService,* modifique os métodos *findById*, *delete* e *update* adicionando um *try/catch* que lance a exceção do tipo *ResourceNotFoundException.*
18. Teste a aplicação utilizando o postman.

## Checklist:

- Utilização de um editor de código sugerido neste documento;
- Instalação do Postman e do Git (opcional);
- Criação do projeto base;
- Adição das dependências e implementação das *properties*.
- Implementação das classes: *User*, *UserResource* e *UserService*.
- Implementação da interface *UserRepository*.
- Implementação das classes e mecanismo de exceção.
- Configuração da aplicação para testes.
- Teste da aplicação com a ferramenta **postman**.

## RESULTADOS

Uma pasta com arquivos que contenha a estrutura do projeto como informado na descrição da atividade, ou seja, os seguintes arquivos com extensão java: *User*, *UserResource*, *UserService*, *UserRepository*, *ResourceExceptionHandler*, *StandardError*, *ResourceNotFoundException* e *TestConfig*.