EST – IPCB

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Aplicações Distribuídas

A Minha Segunda Aplicação Multinível Spring Boot Rest MVC com Spring Data JPA – Controladores REST

(Versão 3 novembro 2022)

3º Ano / 1º Semestre – 2022/2023

Atividade Prática n.º6 - A Minha Segunda Aplicação Multinível Spring Boot Rest MVC com Spring Data JPA – Controladores REST em Detalhe

Nesta atividade pretende-se que devolva e adicione controladores REST à segunda aplicação multinível renomeada como **AppVendas-SpringData-Jpa-Rest**.

Esta atividade diferencia-se da atividade 5, por aprofundar o estudo sobre a criação de APIs REST em particular:

- Planeamento detalhado da estrutura dos URIs + decisões sobre os verbos Http + decisões sobre os códigos Http status a adotar para cada caso.
- Utilização de anotações REST.
- Formas alternativas de retorno dos códigos Http status.

Para melhor compreensão dos conteúdos deste guião sugere-se a consulta do bloco de slides: AD-2023-Modulo-REST_APIs_com_Spring_WEB.pdf.

Passo 1: Clonar a aplicação multinível Spring Boot chamada AppVendas-SpringData-Jpa para **AppVendas-SpringData-Jpa-Rest**.

Passo 2: Adicione no ficheiro pom.xml a dependência do projecto lombok (https://projectlombok.org) para que os getters e setters sejam gerados.

```
<dependency>
     <groupId>org.projectlombok</groupId>
     <artifactId>lombok</artifactId>
     <scope>provided</scope>
</dependency>
```

***Lombok**: It is a Java component library that injects plugins to the editor and builds tools that don't require us to write methods like `Getters` and `Setters` for property variables specified in class and parameterized or no-argument constructors. Instead, we can use some annotations that take care of those methods. These help in removing the boilerplate code.*

- Remova das classes dos Clientes e dos Pedidos os setters e getters e anote as classes com **@Data** ou **@Getter / @Setter**:

  `@Data`

  All together now: A shortcut for `@ToString`, `@EqualsAndHashCode`, `@Getter` on all fields, and `@Setter` on all non-final fields, and `@RequiredArgsConstructor`!

  `@Getter` / `@Setter`
  You can annotate any field with `@Getter` and/or `@Setter`, to let lombok generate the default getter/setter automatically.

Passo 3: Realize o planeamento detalhado da estrutura dos URIs, Verbos Http e código de estado, conforme os princípios e recomendações REST para acesso às coleções de recursos dos Clientes e dos Pedidos.

Considere que a API REST oferece as seguintes ações:

- Criar um novo Cliente
- Consultar um Cliente pelo Id
- Consultar todos os Clientes
- Atualizar um Cliente
- Criar um novo Pedido para um Cliente
- Consultar todos os Pedidos de um Cliente
- Consulta um Pedido pelo Id
- Atualizar um Pedido pelo Id
- Apagar um Pedido pelo Id
- Apagar um Cliente pelo Id e todos os seus Pedido. Nota: é necessário adicionar na @OneToMany a propriedade cascadeType: cascade= CascadeType.ALL
- Apagar todos os Pedidos de um Cliente

Considere o facto da entidade Cliente ter um relacionamento a One-to-Many- com a entidade Pedido e vice-versa.

Durante a implementação defina os tipos de *fetch* (trazer junto) recomendados para o lado Cliente (`fetch=FetchType.LAZY)` e para lado do Pedido(`fetch=FetchType.EAGER).` Significa que o fetch dos pedidos para junto do cliente poderá ter que ser realizado numa consulta subsequente à parte usando uma Query apropriada:

Exemplo:

```
@Query("select c from Cliente c left join fetch c.pedidos
where c.id =:id")
```

```
Cliente findClienteByIdFetchPedidos(@Param("id") BigInteger
id);
```

Um relacionamento pode ser *optional* ou obrigatório.

- Considerando o lado One-to-Many - é sempre opcional, e não podemos fazer nada quanto a isso.
- O lado Many-to-One, por outro lado, oferece-nos a opção de o tornar obrigatório.
  @ManyToOne(optional = false)
  @ManyToOne(optional = true)

Passo 4: Limpe os Repositórios existentes ou crie NOVAS Interfaces Repositórios de Dados para os Clientes e Pedidos que cumprem com os requisitos indicados no Passo 3.

Passo 5: Num package chamado **rest.controllers**, crie a classe do controlador REST, chamado **ControladorClienteV1.**

Passo 6: Implemente a versão v1 do ControladorCliente REST decorando os métodos ação CRUD usando as anotações não-especializadas (@Controller, @ResponseBody, @RequestMapping, etc), e o retorno dos códigos Http Status usando a classe ResponseEntity.

- Utilize a estratégia de controlo de versões baseada num segmento na Path.
- A classe ResponseEntity pode ser utilizada de duas formas, conforme os padrões nos exemplos seguintes:
  - `return ResponseEntity.ok.body("Ola mundo")`
  - `return ResponseEntity.notFound().build();`
  - `return new ResponseEntity<>(HttpStatus.NO_CONTENT)`
  - `return new ResponseEntity<>("Ola mundo", httpStatus.OK)`

Passo 7: Implemente a versão v2 do ControladorCliente REST decorando os métodos ação CRUD usando as anotações especializadas @PostMapping, @GetMapping, @PutMapping e @Delete, e o retorno dos códigos Http Status usando a anotação @ResponseStatus + a classe + Class ResponseStatusException.

Passo 8: Teste a implementação usando o Browser e/ou a Ferramenta Postman.

Métodos Get

Para testar os métodos Get precisa apenas de um simples browser.

Métodos Post, Put, e Delete

Para testar estes métodos precisará de usar a ferramenta de testes de APIs REST (e.g., Postman) ou de usar o comando cURL.

Passo 9: Implemente desde logo, uma versão v2 para o ControllerPedido REST utilizando as anotações especializadas e o retorno dos códigos Http Status usando a anotação @ResponseStatus.

Passo 10: Para integração do Spring Boot e o Swagger-ui, adicione a seguinte dependência à lista de dependências do projeto:

```
<dependency>
     <groupId>org.springdoc</groupId>
     <artifactId>springdoc-openapi-ui</artifactId>
     <version>1.6.6</version>
</dependency>
```

A página do Swagger UI ficará disponível no url http://server:port/context-path/swagger-ui.html e a documentação OpenAPI no formato json no url: http://server:port/context-path/v3/api-docs

Experimente a ferramenta de testes disponibilizada.
No nosso caso deve utilizar o URI:
http://localhost:8080/swagger-ui.html
http://localhost:8080/v3/api-docs

Passo 11: Adicione um Actuator à aplicação e verifique a "saúde" da aplicação.

Para ativar um Spring Boot Actuator, apenas precisamos de adicionar a dependência do spring-boot-actuator ao gestor de pacotes (pom.xml):

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Efetue as seguintes configurações no ficheiro application.properties com as informações gerais a mostrar sobre a aplicação :

```
## Configuring info endpoint for Atuator
info.app.name= Minha Primeira Aplicação Multinível Spring Boot
Rest MVC
info.app.description=Esta aplicação ilustra o desenvolvimento de
uma aplicação multinível Spring Boot
```

```
info.app.version=1.0.0
## Expose all actuator endpoints
management.endpoints.web.exposure.include=*
```

Consulte os links disponíveis invocando o link:

http://localhost:8080/actuator

Nota: Deverão ser apresentados 13 links.

Observe as informações da aplicação invocando o link:

http://localhost:8080/actuator/info

Teste invocado o link actuator/health:

http://localhost:8080/actuator/health

{"status":"UP"}

We can have a detailed view of the health by having the following application configuration:

management.endpoint.health.show-details=always

Teste invocando novamente o link actuator/health:

{"status":"UP","components":{"db":{"status":"UP","details":{"database":"MySQL","validationQuery":"isValid()"}},"diskSpace":{"status":"UP","details":{"total":209190907904,"free":105208123392,"threshold":10485760,"exists":true}},"ping":{"status":"UP"}}}

ANEXO – POSSÍVEL SOLUÇÃO URIs

Métodos	Urls	Ação
POST	/clientes	Criar um novo Cliente
POST	/clientes/{id}/pedidos	Criar um novo Pedido para um Cliente
GET	/clientes	Consultar todos os Clientes
PUT	/clientes/{id}	Atualizar um Cliente

Métodos	Urls	Ação
GET	/clientes/{id}/pedidos	Consultar todos os Pedidos de um Cliente
GET	/pedidos/{id}	Consultar um Pedido pelo id
PUT	/pedidos/{id}	Atualizar um Pedido pelo id
DELETE	/pedidos/{id}	Apagar um Pedido pelo Id
DELETE	/clientes/{id}	Apagar um Cliente (e todos os seus Pedidos) pelo Id
DELETE	/clientes/{id}/pedidos	Apagar todos os Pedidos de um Cliente

FIM