33 Deletando uma categoria

Em **CategoriaResource** criar o método **delete( )**, bastate **parecido** com o **método find( )**. Deve-se **usar** o método **HTTP DELETE** e **retornar** uma **ResponseEntity** sem **conteúdo**, mas **antes** de **retornar** deve-se **invocar** um **método delete(id) implementado** em **CategoriaService**.

```
@RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
public ResponseEntity<Void> delete(@PathVariable Integer id) { }
```

return ResponseEntity.noContent().build();

noContent

```
public static ResponseEntity.HeadersBuilder<?> noContent()
```

Create a builder with a NO_CONTENT status.

Returns:

the created builder

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html#noContent--

NO_CONTENT

```
public static final HttpStatus NO_CONTENT
```

204 No Content.

See Also:

HTTP/1.1: Semantics and Content, section 6.3.5

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpStatus.html#NO_CONTENT

Utilizando o ResponseEntity

Em situações que precisamos ter mais controle sobre a resposta HTTP em um *endpoint*, o próprio Spring nos oferece a classe `ResponseEntity` que nos permite manipular os dados HTTP da resposta.

https://medium.com/collabcode/boas-pr%C3%A1ticas-para-a-implementa%C3%A7%C3%A3o-de-apis-no-spring-boot-com-kotlin-6e77aac110da

https://www.baeldung.com/spring-response-entity

Em **CategoriaService implementar** o método **delete (Integer id)** com retorno void. Neste metódo **realizar** a **invocação do** método **deleteById(id)** de **CategoriaRepository**. É **possível chamar** o método **find(id)** antes de deleteById(id) e **lançar** uma **exceção** a **categoria** não **existir**.

```
public void delete(Integer id) {
        find(id);
        categoriaRepository.deleteById(id);
}
```

Exceção ao deletar categorias que possuem produtos cadastrados, **DataIntegrityViolationException**.

Ao tentar deletar a categoria(1) ocorre um **Internal Server Error** com código **500** porque para deletar um Objeto que possui outros Objetos associados a ele existem duas possibilidades, deletar tudo (objeto e objetos associados) ou abortar a deleção.

"Não deixar estourar o Internal Server Error para o usuario, usar uma exceção personalizada assim como ocorre em find( ) de CategoriaService."

Realizar uma solicitação em **localhost:8080/categorias/1** com **HTTP DELETE** gera uma exceção do tipo **DataIntegrityViolationException** (categoria 1 possui produtos ligados a ela), logo esta é a exceção que deve ser **capturada** no método **delete( )** de **CategoriaService**.

Criar uma **exceção personalizada** da camada de serviço chamada **DataIntegrityException** no pacote services/exceptions, a sua implementação é praticamente identica a **ObjectNotFoundException**.

Em **CategoriaService** envolve-se a instrução **categoriaRepository.delete(id)** do método **delete( )** com um **TryCatch** para **capturar exceção** do tipo **DataIntegrityViolationException.** Então quando ela acontecer lançamos a exceção personalizada **DataIntegrityException.**

Para lançar a exceção personalizada em CategoriaService delete( ):

```
        throw new DataIntegrityException("não é possível apagar categoria com produtos");
```

Lançada em **CategoriaService** a exceção **sobe** para a **camada resource** na classe **CategoriaResouce**, apesar disso o método delete( ) de CategoriaResource não implementa TryCatch. Assim como na exceção de busca por id o tratamento é feito em ResourceExceptionHandler do pacote resources/exceptions.

CategoriaService

```
public void delete(Integer id) {
    find(id);
    try {
        categoriaRepository.deleteById(id);
    } catch (DataIntegrityViolationException e) {
        throw new DataIntegrityException("Não é possivel excluir uma categoria que possui produtos.");
    }
}
```

CategoriaResource

```
@RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
public ResponseEntity<Void> delete(@PathVariable Integer id) {
    categoriaService.delete(id);
    return ResponseEntity.noContent().build();
}
```

ResourceExceptionHandler

```
@ExceptionHandler(DataIntegrityException.class)
public ResponseEntity<StandardError> objectNotFound(DataIntegrityException e, HttpServletRequest request) {
    StandardError standardError = new StandardError(HttpStatus.BAD_REQUEST.value(), e.getMessage(), System.currentTimeMillis());
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(standardError);
}
```

Classe ResourceExceptionHandler e anotação @ControllerAdvice:

https://medium.com/@jovannypcg/understanding-springs-controlleradvice-cd96a364033f