Curso: Spring Boot com Angular 7

https://www.udemy.com/user/hugo-silva

Prof. Hugo Silva

Capítulo: Construindo API REST

Objetivo geral:

- Importar o projeto Back end na IDE Intellij IDEA
- Criar as operações de CRUD

Importando o projeto na IDE Intellij IDEA

Checklist:

- Abrir o projeto na sua IDE de preferência
- Rodar o projeto e testar: http://localhost:8080
- Se quiser mudar a porta padrão do projeto, incluir em application.properties: `server.port=${port:8081}`

Entity User e REST funcionando

Checklist para criar entidades:

- Atributos básicos
- Construtores (não inclua coleções no construtor com parâmetros)
- Getters e setters
- hashCode e equals (implementação padrão: somente id)
- Serializable (padrão: 1L) : `private static final long serialVersionUID = 1L;`

Checklist:

- No subpacote **domain**, criar a classe **User**
- No subpacote **resources**, criar uma classe **UserResource** e implementar nela o endpoint GET padrão:

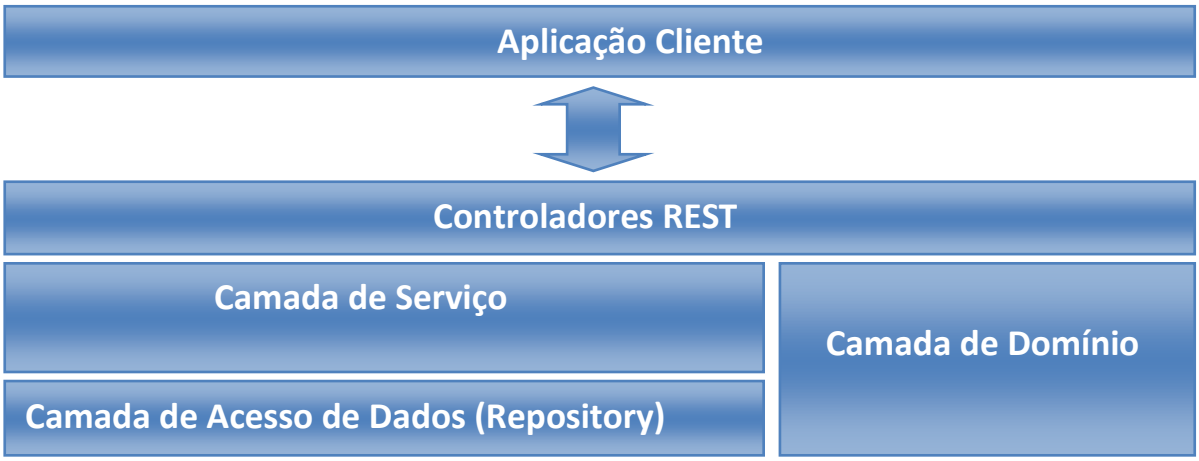
```
@RestController
@RequestMapping("/api")
public class UserResource {

    @GetMapping("/users")
    public ResponseEntity<List<User>> findAll() {

        List<User> users = new ArrayList<>();
        User joao = new User("João", "Souza", "joao@gmail.com");
        User maria = new User("Maria", "Teixeira", "maria@gmail.com");
        users.addAll(Arrays.asList(joao, maria));
        return ResponseEntity.ok().body(users);
    }

}
```

Conectando ao MongoDB com repository e service



Checklist:

- Em pom.xml, incluir a dependência do MongoDB:

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

- No pacote **repository**, criar a interface **UserRepository**
- No pacote **services**, criar a classe **UserService** com um método **findAll**
- Em **User**, incluir a anotação *@Document* e *@Id* para indicar que se trata de uma coleção do MongoDB
- Em **UserResource**, refatorar o código, usando o **UserService** para buscar os usuários
- Em *application.properties*, incluir os dados de conexão com a base de dados:

Atualização: MongoDB Atlas

```
spring.data.mongodb.uri=mongodb+srv://wsuser:1234ws@wsoauth2-
ub8pa.mongodb.net/test?retryWrites=true
```

Versão Antiga do mLab:

```
spring.data.mongodb.uri=mongodb://wsuser:1234ws@ds211875.mlab.com:11875/ws
oauth2
```

- Usando o **MongoDB Compass**:
  - Criar alguns documentos **user** manualmente
- Testar o endpoint /users

Operação de instanciação da base de dados

Checklist:

- No subpacote **config**, criar uma classe de configuração **SetupDataLoader** que implemente **CommandlLineRunner**
- Implementar o método para verificar se o usuário existe

```
private  User createUserIfNotFound(final User user)
```

- Dados para copiar:

```
User joao = new User("João", "Souza", "joao@gmail.com");
User maria = new User("Maria", "Teixeira", "maria@gmail.com");
```

- Deletar todos os usuários da base de dados pelo Mongo Compass
- Inicializar a aplicação e verificar se os usuários foram inicializados;

Usando padrão DTO para retornar usuários

DTO (*Data Transfer Object*): é um objeto que tem o papel de carregar dados das entidades de forma simples, podendo inclusive "projetar" apenas alguns dados da entidade original. Vantagens:

- Otimizar o tráfego (trafegando menos dados)

- Evitar que dados de interesse exclusivo do sistema fiquem sendo expostos (por exemplo: senhas, dados de auditoria como data de criação e data de atualização do objeto, etc.)

Checklist:

- No subpacote **dto**, criar **UserDTO**
- Implements **Serializable**: *private static final long serialVersionUID = 1L;*
- Em **UserResource**, refatorar o método **findAll**

Obtendo um usuário por *ID*

Checklist:

- No subpacote **service**.**exception**, criar **ObjectNotFoundException**
- Serializable --> *private static final long serialVersionUID = 1L;*
- Em **UserService**, implementar o método **findById**
- Em **UserResource**, implementar o método **findById** (retornar **DTO**)
- No subpacote **resources.exception**, criar as classes:
  - **StandardError**
  - **RestResponseEntityExceptionHandler**

Criação de usuário com *POST*

Checklist:

- Em **UserService**, implementar os métodos **create** e **fromDTO**

- Em **UserResource**, implementar o método **create**

Atualização de usuário com *PUT*

Checklist:

- Em **UserService**, implementar os métodos **update**
- Em **UserResource**, implementar o método **update**

Deleção de usuário com *DELETE*

Checklist:

- Em **UserService**, implementar o método **delete**
- Em **UserResource**, implementar o método **delete**

Criando entity Role

Checklist:

- Criar classe **Role**
- Criar **RoleRepository**
- Inserir alguns roles na carga inicial (classe **SetupDataLoader**) da base de dados
- Criar o método:

```
private  Role  createRoleIfNotFound(String name)
```

Referenciando os Roles do usuário

@**DBRef** é uma convenção pra representar um documento relacional, ou seja, semelhante a chave estrangeira vista em banco relacional.

- **$ref:** nome da coleção a ser referenciada.
- **$id:** o ObjectId do documento referenciado.
- **$db:** a database onde a coleção referenciada se encontra.

Checklist:

- Em **User**, criar o atributo "**roles**", usando *@**DBRef***
  - Sugestão: incluir o parâmetro (lazy = true)
- Refatorar a carga inicial inicial (classe **SetupDataLoader**) da base de dados, incluindo as associações dos roles

Endpoint para retornar os Roles de um usuário

Checklist:

- Em **UserResource**, criar o método para retornar os **roles** de um dado usuário