EST – IPCB

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Aplicações Distribuídas

A Minha Primeira Aplicação Multinível Spring Boot Rest MVC

3º Ano / 1º Semestre – 2022/2023

Atividade Prática n.º3 - A Minha Primeira Aplicação Multinível Spring Boot Rest MVC

Nesta atividade pretende-se que desenvolva uma aplicação Multinível Spring Boot Rest MVC.

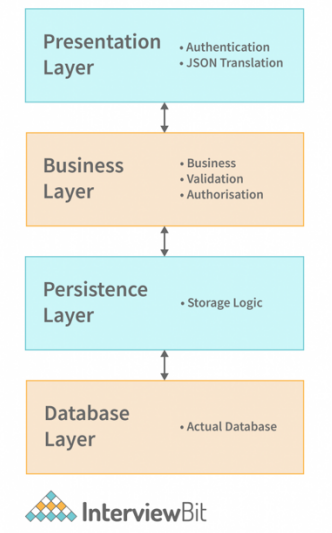
Arquitetura Spring Boot

Spring Boot é uma parte especializada do Spring Framework. É utilizada para criar aplicações de alta qualidade, de grau de produção, baseadas Spring Framework com esforço mínimo. O código Spring Framework é utilizado para criar Spring Boot.

Spring Boot utiliza uma arquitectura hierárquica na qual cada camada comunica com a camada imediatamente abaixo ou acima dela (estrutura hierárquica).

Antes de chegarmos à Arquitectura Spring Boot, temos primeiro de compreender o significado de cada uma dessas camadas e classes. As quatro camadas da Spring Boot são as seguintes:

- Camada de Apresentação
- Camada empresarial ou de negócio
- Camada de persistência
- Camada de base de dados



1. Camada de apresentação

A camada de apresentação é a camada superior da arquitetura Spring Boot. Consiste nas Vistas. Trata dos pedidos HTTP e efetua a autenticação. É responsável pela conversão do parâmetro do campo JSON para Objetos Java e vice-versa. Uma vez realizada a autenticação do pedido, passa-o para a camada seguinte, ou seja, a camada de negócio.

2. Camada de negócio

A camada negócio contém toda a lógica empresarial. É constituída por classes de serviços. É responsável pela validação e autorização.

3. Camada de persistência

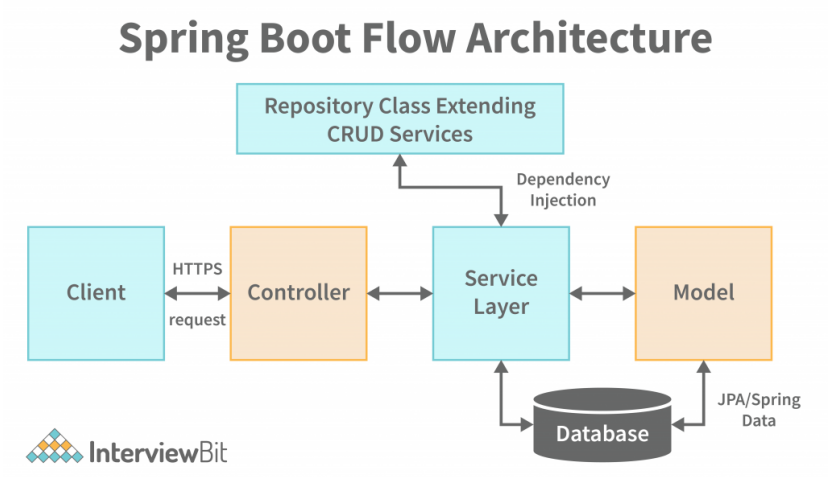
A camada de persistência contém toda a lógica de armazenamento da base de dados. É também responsável pela conversão de objetos de negócio para a base de dados e vice-versa.

4. Camada de base de dados

A camada de base de dados contém todas as bases de dados tais como MySql, MongoDB, etc. Esta camada pode conter múltiplas bases de dados. É responsável pela realização das operações CRUD.

Arquitetura do Fluxo de Trabalho do Spring Boot

Spring Boot depende fortemente da estrutura Spring, o que significa que integra quase todas as características e módulos Spring, como Spring MVC, Spring Core, etc.



Ao longo desta atividade, pretende-se que ao realizar os passos seguintes experimente diversas funcionalidades do Spring Boot e possa ficar a conhecer algumas características das aplicações Spring Boot MVC + Rest.

Mais à frente, numa das próximas atividades será aprofundado o estudo/uso do Spring Boot Rest MVC.

Para melhor compreensão dos conteúdos deste guião sugere-se a consulta do bloco de slides: AD-2023-Bloco_1_Spring_e_Spring_Boot.pdf.

Passo 1: Criar uma aplicação multinível SpringBoot MVC chamada DemoMVC-Rest:

Utilize o Spring Initializr (https://start.spring.io) para adicionar ao ficheiro pom.xml os Starter Spring Web, e Thymeleaf.

1. Escolha o **Project** do tipo **Maven Project**.
2. Escolha a linguagem **Java**.
3. Escolha a versão do Spring Boot. Utilize a última versão estável disponível.
4. Introduza os metadados do projecto – group ID, artifact ID, name of the project, project description, e package name.
5. Escolha **Packaging** as **Jar**.
6. Escolha **Java 17 ou Java 19 (a versão mínima é Java 8)**.
7. Adicione as dependências – **Spring Web** e **Actuator**.
8. Clique o botão **Generate**.
9. Descomprima o ficheiro .zip.

Passo 2: A aplicação é composta por um modelo de dados Conta, um controlador @RestController ControladorContas, um @Service ServicoContas, e um @Repository RepositorioContas.

Uma instância da classe `Conta` corresponderá a um registo de uma conta bancária num sistema de armazenamento de dados (e.g., Base de Dados Relacional ou NoSql). Na presente atividade utilizaremos um sistema de armazenamento em memória baseado num HaspMap.

```
public class Conta {
    private long id; //Gerado aleatoriamente pela aplicação.
    private String titular;
    private String morada;
    private long nif;
    private long pin;
    private double saldo;
//getters e setters
….
}

@RestController
public class ControladorContas
{
// Métodos que executam as ações
}

...
```

O Controlador REST a implementar embrulha o serviço ServicoContas com uma camada web/REST.

O serviço permite realizar as operações CRUD à conta bancária. Para tal este usa um repositório de dados.

O repositório realiza os acessos/as consultas ao sistema de dados utilizado.

Passo 3: Implementar o Repositório de Dados RepositorioContas

De momento, sugere-se que:

- O RepositorioContas simule uma base de dados criando um HashMap em memória:

```
private final Map<Long, Conta> bdcontas = new HashMap<Long, Conta>();
```

- O RepositorioContas implemente as operações CRUD sobre a base de dados/HashMap.
- Os métodos do RepositorioContas não devem retornar null. Para dar corpo a este requisito utilize sempre que possível a classe Optional introduzida no Java 8.

Exemplo parcial:

```
@Repository
public class RepositorioContas {
    private final Map<Long, Conta> bdcontas = new HashMap<Long, Conta>();
    public Conta criaConta(Conta conta) {
        bdcontas.put(conta.getId(), conta);
        return conta;
    }
    public Optional<Conta> getContaById(Long id) {
        return Optional.ofNullable(basedados.get(id));
    }
}
```

Passo 4: Implementar o serviço ServicoContas

De momento, este serviço simplesmente reencaminha os pedidos para as operações CRUD chamando os correspondentes métodos do RepositorioContas.

Passo 5: Implementar um Application/Command Line Runner

Implementar um Runner que usa o ServicoContas para inserir 5 contas na base de dados, com saldos de 1000.0, 10000.0 e sucessivamente maiores.

Passo 6: Implementar no ControladorContas os métodos ação CRUD seguintes anotados com @PostMapping, @GetMapping, @PutMapping e @Delete, respetivamente:

//URI: POST /createconta
@PostMapping
```
Conta criar(@RequestBody Conta conta);
```
Este método cria uma conta e retorna a conta criada.

//URI: GET /getconta/{id}
@GetMapping
```
Optional<Conta> consultar(@PathVariable("id" long id);
```
Este método permite a consulta do registo da conta id.

//URI: PUT /updateconta/{id}
@PutMapping
```
void editar(@RequestBody Conta conta, @PathVariable("id") long
id);
```
Este método permite atualizar o registo da conta id.

//URI: DELETE /deleteconta/{id}
@DeleteMapping
```
void remover(@PathVariable("id") long id);
```
Este método permite remover o registo da conta id.

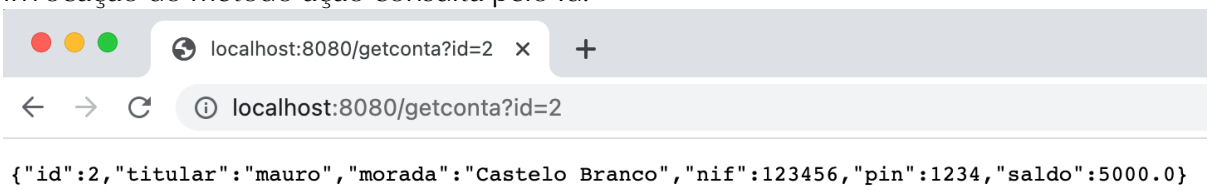
Nota: Nesta implementação ainda não são seguidas as recomendações REST durante a definição dos URIs.

Passo 7: Teste a implementação

Métodos Get

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

Na figura seguinte usando um browser Web, ilustra-se o resultado esperado para uma invocação do método ação consulta pelo Id.



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.

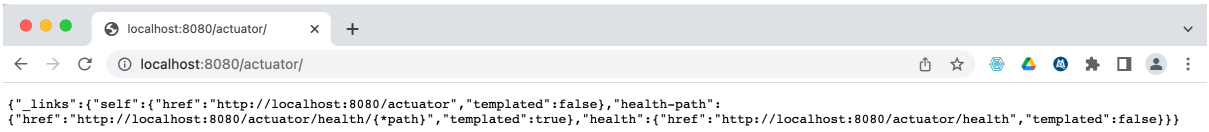
Passo 8: Adicione um Actuator à aplicação MVC e verifique a "saúde" da aplicação.

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=*
```



FIM