



EST – IPCB

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Aplicações Distribuídas

A Minha Primeira Aplicação Spring Boot
3º Ano / 1º Semestre – 2022/2023

Atividade Prática n.º1 - A Minha Primeira Aplicação Spring Boot

De seguida, pretende-se que desenvolva a primeira aplicação Spring Boot web-based com um REST endpoint.

Ao longo desta atividade, pretende-se que ao realizar os passos seguintes experimente diversas funcionalidades do Spring Boot e conheça algumas das características essenciais das aplicações Spring Boot.

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 – Aceder à plataforma Spring Initializr e criar o projecto da Aplicação Spring Boot Demo “Hello World”.

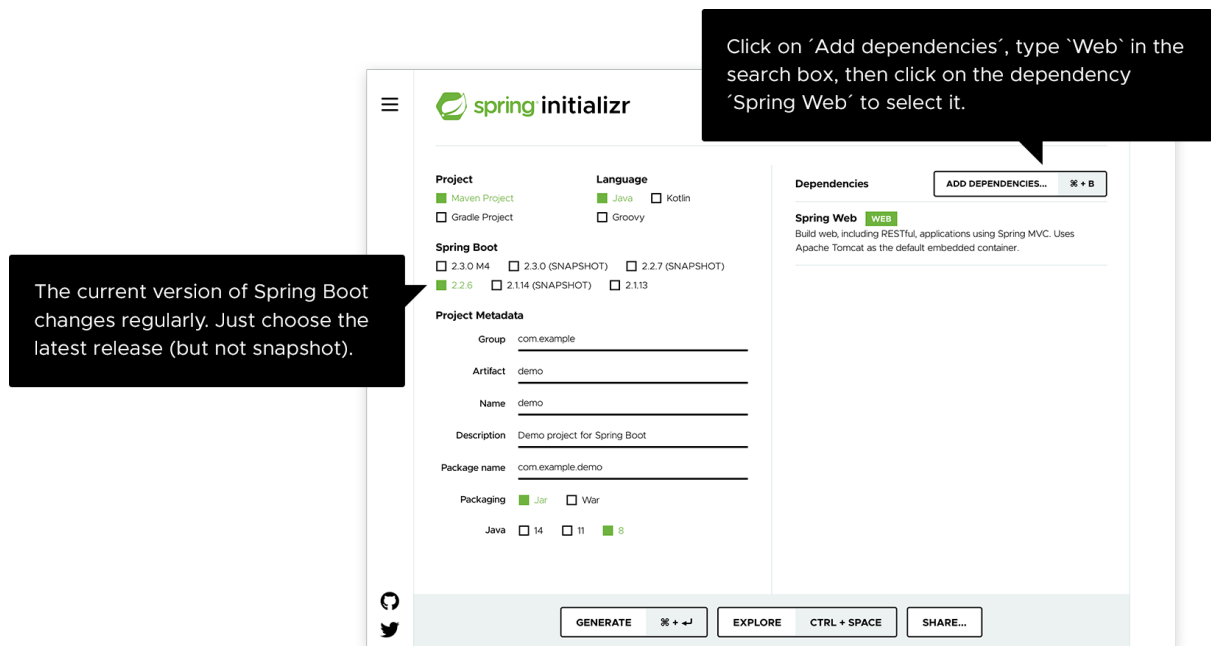
Para desenvolver uma aplicação Spring Boot a partir zero, pode utilizar a ferramenta Spring Initializr fornecida pela equipa Spring, que lhe oferece as dependências mais utilizadas e a pode configurar com essas dependências.

Para utilizar a Plataforma Spring Initializr, realizar os seguintes passos:

1. Aceda ao seguinte website usando o Chrome ou outro browser:

<https://start.spring.io>

A imagem de ecrã seguinte mostra a ferramenta Spring Initializr a configurar/estabelecer a nossa primeira aplicação:



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

Passo 2 - Observar a Classe principal da Aplicação e executar a aplicação.

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

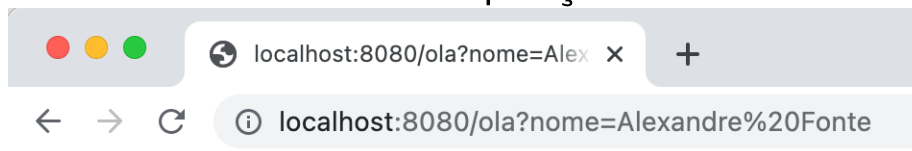
Passo 3 - Criar um controlador REST, decorando a classe da aplicação com @RestController.

```
@SpringBootApplication
@RestController
//URI Base: http://localhost:8080
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    //Http GET /ola?nome=alexandre%20fonte
    @GetMapping(value = "/ola")
    public String olamundo(@RequestParam(value = "nome", defaultValue = "Mundo!") String nome){
        return "Olá " + nome;
    }
}
```

Passo 4 - Executar e Testar a Aplicação



Olá Alexandre Fonte

Serialização/Conversão de um Objeto Java-> Json

Nos passos seguintes, pode experimentar que o Spring Boot permite a conversão *automática* de um objeto Java para Json (e vice-versa) recorrendo ao módulo Jackson, mais corretamente ao Jackson ObjectMapper.

Passo 5 - Criar uma classe POJO que representa recurso Conta. Inclua na implementação:

- Um construtor que permita inicializar as propriedades.
- Todos os Getters e Setters.
- O método toString()

```
public class Conta {
    private long id;
    private String titular;
    private String morada;
    private long nif;
    private long pin;
    private double saldo;
    private LocalDateTime data_atual;
    ...
}
```

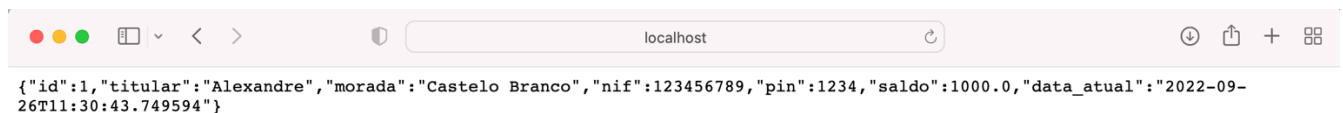
```
}
```

Passo 6 - Incluir no controlador REST um método que retorna uma instância da Conta. Por exemplo:

```
@GetMapping(value = "/getconta")
public Conta getContaById(@RequestParam("id") long id) {
    return new Conta(id, "Alexandre", "Castelo
Branco", 123456789, 1234, 1000.0, LocalDateTime.now());
}
```

O Spring Boot recorre ao módulo Jackson para converter a Conta para o formato Json. Nalgumas versões é necessário acrescentar explicitamente na anotação GetMapping o atributo: `produces = "Application/Json"`.

Passo 7 - Executar e Testar.

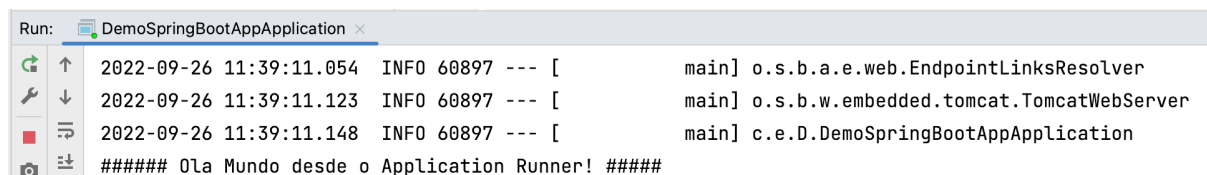


É possível configurar o Jackson de várias formas (em application.properties ou no código) para personalizar as conversões para Json, como no problema comum do formato de data, por agora aspetos que não serão abordados em profundidade.

Ainda assim, experimente colocar a anotação `@JsonFormat(pattern = "dd:MM:yyyy")` para definir o padrão de data indicado: dia-mês-ano.

Interface Application Runner

Passo 8 - Experimente a utilização de um Application Runner, que imprime uma mensagem no ecrã durante o arranque da aplicação. Por exemplo:



Para realizar este passo precisa de indicar que a classe da aplicação Spring Boot implementa a interface **ApplicationRunner**, e de implementar o método **run()** da interface. Neste inclui:

```
System.out.println("##### Ola Mundo desde o Application Runner!
#####");
```

Ver também a utilização de um `CommandLineRunner`.

Configurações Personalizadas de uma Aplicação Spring Boot usando uma Classe `@Configuration` e `@Bean`

Passo 9 - Crie uma configuração Personalizada criando uma nova classe `MinhaConfiguracao`. Nesta será definido o nome da aplicação, declarando um método que instancia um `@Bean` do tipo `String` chamado `nome.app`. Por exemplo:

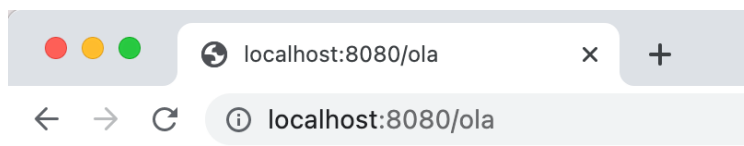
```
@Configuration
public class MinhaConfiguracao {

    @Bean(name = "nome.app")
    public String getNomeAplicacao(){
        return "App Spring Boot Ola Mundo!";
    }
}
```

Para utilizar e testar esta configuração, defina na classe principal da aplicação uma propriedade/campo anotada com `@Autowired` e `@Qualifier(<nome do bean>)` e adapte o método `olamundo()`. Por exemplo:

```
@Autowired
@Qualifier("nome.app")
private String nomeaplicacao;

//Http GET /ola?nome=Alexandre%20Fonte
@GetMapping(value = "/ola")
public String olamundo(@RequestParam(value = "nome", defaultValue = "Mundo!") String nome){
    return "Olá " + nome + " da app "+nomeaplicacao;
}
```



Olá App Ola Mundo!

Repita o processo declarando um novo `@Bean` chamado `nome.autor`, implemente e teste a sua utilização.

Configurações Externalizadas de uma Aplicação Spring Boot usando o ficheiro `application.properties`

Passo 10 - Crie uma configuração Personalizada, recorrendo as propriedades definidas no ficheiro `application.properties`, externo ao código da aplicação.

Corresponde a definir as configurações no ficheiro das propriedades da aplicação **`application.properties`**. As propriedades seguem o modelo chave-valor.

Experimentar o seguinte exemplo:

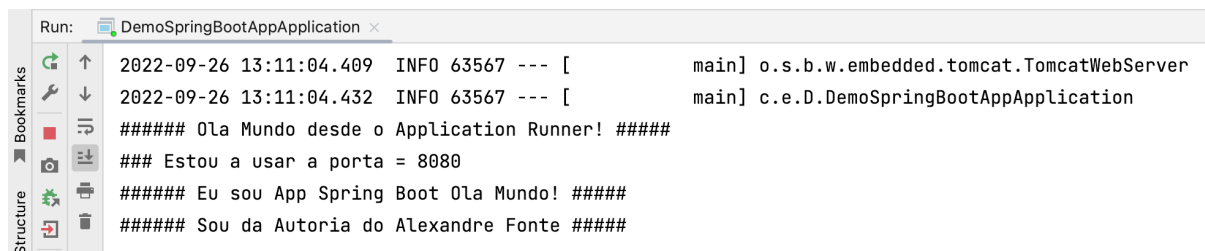
1. Colocando no ficheiro **`application.properties`** as propriedades:

```
server.port=8080
nome.autor=Alexandre Fonte
nome.aplicacao=App Spring Boot Ola Mundo!
```

2. Lendo as propriedades, criando na classe da aplicação três novos campos anotados com `@Value("${nome configuracao a ler}")`. Exemplo para o caso do autor:

```
@Value("${nome.autor}")
private String autoraplicacao;
```

3. Atualize o **`ApplicationRunner`** por forma a produzir um output semelhante:



```
Run: DemoSpringBootApplication x
2022-09-26 13:11:04.409 INFO 63567 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer
2022-09-26 13:11:04.432 INFO 63567 --- [main] c.e.D.DemoSpringBootApplication
##### Ola Mundo desde o Application Runner! #####
### Estou a usar a porta = 8080
##### Eu sou App Spring Boot Ola Mundo! #####
##### Sou da Autoria do Alexandre Fonte #####
```

Beans, IoC e DI

Passo 12 - Implemente um Bean Fibo anotado com `@Service` que disponibiliza um método que imprime os primeiros N números da sequência de Fibonacci.

Neste caso tratando-se de um Bean do tipo `@Service` precisa de implementar uma classe que dará corpo ao Bean.

Ajuda:

ver: https://en.wikipedia.org/wiki/Fibonacci_number

ver: <https://www.geeksforgeeks.org/different-ways-to-print-fibonacci-series-in-java/>

Passo 13 – Usando o mecanismo de Injeção de Dependências, adicione à classe da aplicação um método REST que utiliza o serviço implementado.

FIM