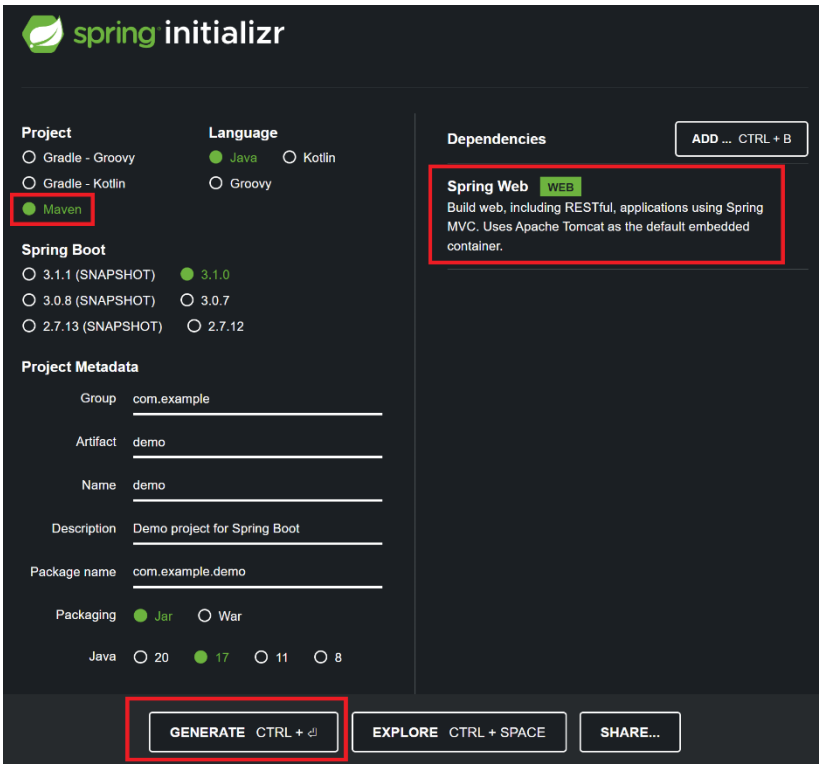
Universidade Federal de Uberlândia
Faculdade de Computação
Introdução ao Spring Boot – Prof. Daniel A. Furtado

Esta atividade tem como objetivo introduzir o framework Spring utilizando o Spring Boot para criar um serviço RESTful simplificado que permite a criação, leitura e exclusão de dados de endereço (CEP, rua, bairro e cidade). Antes de iniciar, leia os slides disponibilizados em https://furtado.prof.ufu.br/site/teaching/PPI/PPI-Modulo10-Intro-Web-Services-Spring.pdf.

Passo a passo:

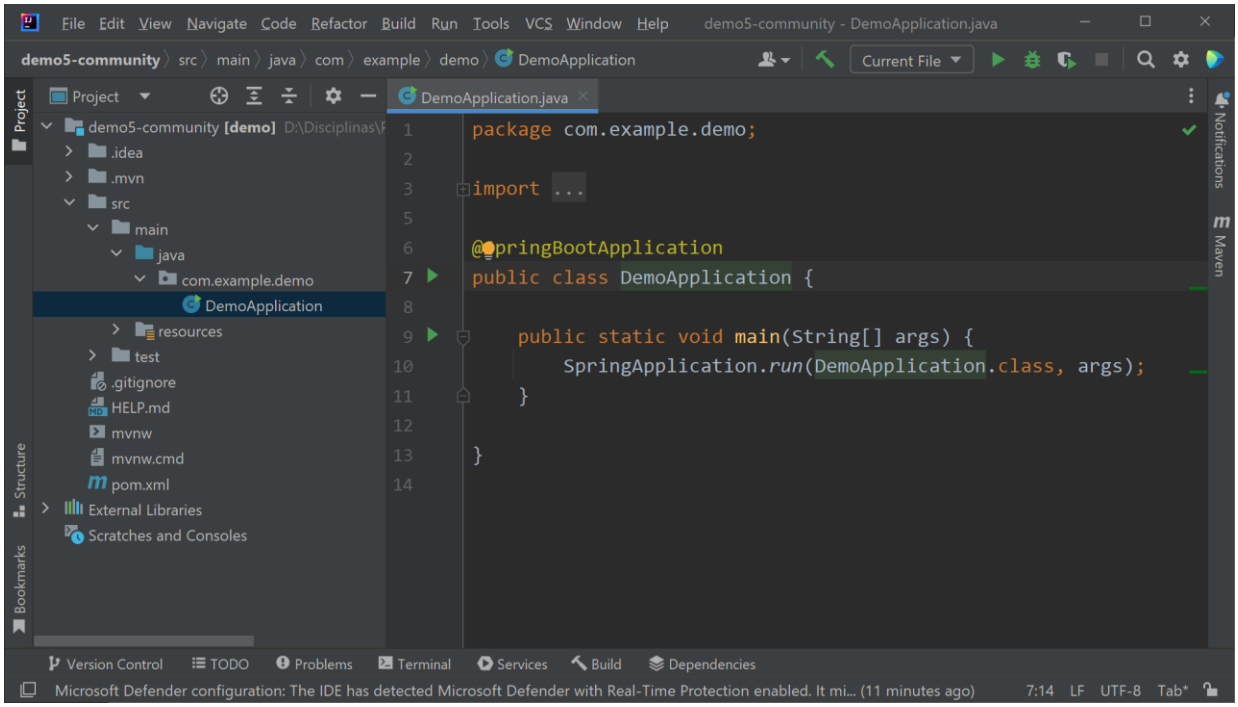
1) Baixe e instale o **IntelliJ IDEA Community Edition**. Caso tenha acesso à versão **Ultimate**, ela também pode ser utilizada (neste caso os passos 2 e 3 a seguir não serão necessários, pois um novo projeto Web utilizando o Maven pode ser criado diretamente dentro do IntelliJ IDEA Ultimate);

2) Caso esteja utilizando o **IntelliJ IDEA Community** será necessário acessar a ferramenta disponível em https://start.spring.io para criação de um projeto pré-inicializado, o qual será aberto posteriormente no IntelliJ IDEA Community. Após acessar a página acima, no lado esquerdo, escolha o **Maven** como ferramenta de automação de compilação e deixe as demais opções como estão. No lado direito, adicione a dependência **Spring Web** e posteriormente clique no botão **Generate** para exportar os dados do projeto e salvar o arquivo compactado (veja figura a seguir);



3) Descompacte o arquivo zip para um local de sua escolha e observe o conteúdo da pasta gerada. Abra o IntelliJ IDEA e escolha a opção **Open** para abrir a pasta do projeto;

4) No painel esquerdo, expanda as pastas e observe o conteúdo da classe **DemoApplication**, cujo método **main** será o ponto de entrada da nossa aplicação:



5) Caso apareceça a mensagem de warning "**Projet JDK is not defined**", clique na opção **Setup SDK** logo à frente e escolha **Download JDK → Download**;

6) Execute a aplicação clicando na aba contendo a classe Demo Application e em seguida no botão ▷. Observe as mensagens apresentadas no painel inferior. Repare que as dependências do projeto são automaticamente baixadas e que o servidor web **Apache Tomcat** é automaticamente embutido e inicializado, aguardando por requisições HTTP na porta 8080:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.4)
```

```
main] com.example.demo.DemoApplication          : Starting DemoApplication using Java 22 with PID
main] com.example.demo.DemoApplication          : No active profile set, falling back to 1 default
main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat initialized with port 8080 (http)
main] o.apache.catalina.core.StandardService    : Starting service [Tomcat]
main] o.apache.catalina.core.StandardEngine     : Starting Servlet engine: [Apache Tomcat/10.1.19]
main] o.a.c.c.C.[Tomcat].[localhost].[/]        : Initializing Spring embedded WebApplicationConte
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization compl
main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat started on port 8080 (http) with context
main] com.example.demo.DemoApplication          : Started DemoApplication in 0.883 seconds (proces
```

7) Abra o navegador de internet e digite **localhost:8080**. A seguinte página deve ser apresentada:

8) Pare a execução da aplicação clicando em . Em seguida, no painel à esquerda, procure pelo arquivo **pom.xml**. Esse é o arquivo de configuração do **Maven** contendo dados importantes sobre o projeto, como nome de identificação, versão do Java e dependências. Observe que a dependência selecionada no início desse roteiro aparece com a identificação **spring-boot-starter-web**;

9) Criação de classe do tipo Controller. No painel à esquerda, clique com o botão direito sobre **com.example.demo**, escolha **New → Java Class** e informe o nome **AddressController**. Crie um método de nome **helloWorld** que retorne a string "Hello World!". Adicione a *annotation* **@RestController** na classe **AddressController** e a *annotation* **@GetMapping("/hello")** no método **helloWorld**, conforme apresentado na figura a seguir. Em web services RESTful criados com o Spring as requisições HTTP são tratadas por um controlador, identificado por **@RestController**. Isso permitirá o mapeamento dos endereços e métodos das requisições HTTP aos métodos internos dessa classe. **@GetMapping("/hello")**, inserido antes do método **helloWorld**(), especifica que uma requisição HTTP ao endereço **/hello** utilizando o método **GET** deve disparar a execução do método **helloWorld**();

DemoApplication.java | pom.xml (demo) | AddressController.java

```
1   package com.example.demo;
2
3   import org.springframework.web.bind.annotation.GetMapping;
4   import org.springframework.web.bind.annotation.RestController;
5
    no usages
6   @RestController
7   public class AddressController {
        no usages
8       @GetMapping("/hello")
9       public String helloWorld() {
10          return "Hello World!";
11      }
12  }
```

10) Clique com o botão direito sobre **DemoApplication** e execute novamente a aplicação. Abra o navegador e digite **localhost:8080/hello** e observe o resultado. Aparecerá o texto "Hello World!" retornado pelo método.

11) Criação da classe Address. No painel à esquerda, clique com o botão direito sobre **com.example.demo** e adicione uma nova classe de nome **Address**. Adicione os atributos cep, rua, bairro e cidade, como mostrado a seguir:

```
public class Address {
    no usages
    private String cep;
    no usages
    private String rua;
    no usages
    private String bairro;
    no usages
    private String cidade;
}
```

12) Gere um construtor para a classe: **botão direito → Generate → Constructor** e selecione todos os atributos:

```
public class Address {
    1 usage
    private String cep;
    1 usage
    private String rua;
    1 usage
    private String bairro;
    1 usage
    private String cidade;

    no usages
    public Address(String cep, String rua, String bairro, String cidade) {
        this.cep = cep;
        this.rua = rua;
        this.bairro = bairro;
        this.cidade = cidade;
    }
}
```

13) Gere os métodos *getters* e *setters*: **botão direito → Generate → Getter and Setter** e selecione todos os atributos:

```
public class Address {
    3 usages
    private String cep;
    3 usages
    private String rua;
    3 usages
    private String bairro;
    3 usages
    private String cidade;

    no usages
    public Address(String cep, String rua, String bairro, String cidade) {
        this.cep = cep;
        this.rua = rua;
        this.bairro = bairro;
        this.cidade = cidade;
    }

    no usages
    public String getCep() { return cep; }

    no usages
    public void setCep(String cep) { this.cep = cep; }

    no usages
    public String getRua() { return rua; }

    no usages
    public void setRua(String rua) { this.rua = rua; }

    no usages
    public String getBairro() { return bairro; }

    no usages
    public void setBairro(String bairro) { this.bairro = bairro; }

    no usages
    public String getCidade() { return cidade; }

    no usages
    public void setCidade(String cidade) { this.cidade = cidade; }
}
```

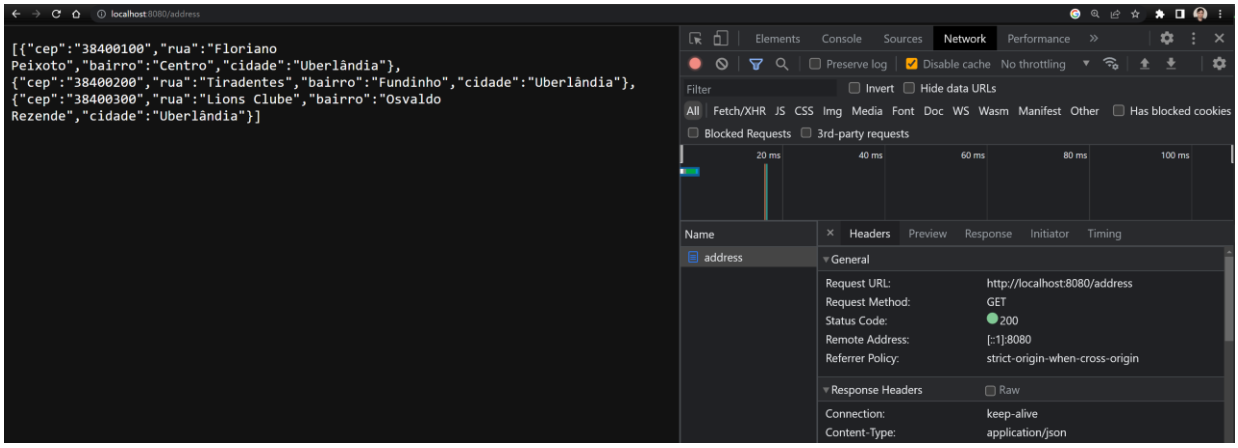
14) Volte à classe **AddressController** e crie um atributo de nome **addresses**, conforme figura a seguir. Observe que o atributo é uma lista, a qual representará nossa base de dados de endereços (**OBS**: neste exemplo introdutório não será criada uma classe de serviço nem haverá acesso a banco de dados. Também não envolverá os conceitos de Injeção de Dependência e Inversão de Controle (IoC)).

```java
@RestController
public class AddressController {

    no usages
    private final List<Address> addresses = new ArrayList<>(
            Arrays.asList(
                    new Address( cep: "38400100", rua: "Floriano Peixoto", bairro: "Centro", cidade: "Uberlândia"),
                    new Address( cep: "38400200", rua: "Tiradentes", bairro: "Fundinho", cidade: "Uberlândia"),
                    new Address( cep: "38400300", rua: "Lions Clube", bairro: "Osvaldo Rezende", cidade: "Uberlândia")
            )
    );

    no usages
    @GetMapping("/hello")
    public String helloWorld() {
        return "Hello World!";
    }
}
```

15) Retornando todos os endereços. Adicione um método de nome **getAddresses** para retornar toda a lista de endereços. Acrescente também a *annotation* **@GetMapping("/address")** para associar as requisições GET utilizando o endereço "/address" à chamada do método:

```java
no usages
@GetMapping("/address")
public List<Address> getAddresses() {
    return this.addresses;
}
```

16) Execute novamente a aplicação, abra o navegador e digite **localhost:8080/address**. Observe que a aplicação retorna uma resposta HTTP contendo os dados de todos os endereços no formato JSON (a conversão do atributo **addresses**, do tipo *List<Address>*, para JSON é feita automaticamente pela aplicação/framework):



17) Retornando um endereço específico dado o CEP. Acrescente um método de nome **getAddress** que receba o CEP por parâmetro, busque por ele na lista de endereços e retorne uma resposta HTTP contendo os respectivos dados:

```java
@GetMapping("/address/{cep}")
public ResponseEntity<Address> getAddress(@PathVariable String cep) {
    for (Address address : this.addresses)
        if (address.getCep().equals(cep))
            return ResponseEntity.ok(address);

    return ResponseEntity.notFound().build();
}
```

Neste caso é necessário resgatar o CEP direto da URL. Observe que o endereço inserido em @GetMapping contém uma variável ({cep}). A *annotation* **@PathVariable** antes do parâmetro

cep do método possibilita que o valor da variável **cep** retirado da URL seja repassado automaticamente ao parâmetro de **mesmo nome** do método. Observe também que nesse método foi utilizado a classe **ResponseEntity** para possibilitar a construção de uma resposta HTTP em situação de sucesso ou erro. Caso o cep seja localizado, será retornado uma resposta HTTP com código de status 200 e o endereço como conteúdo (**body**). Caso contrário, será retornado uma resposta com código de status 404 (not found);

18) Execute novamente a aplicação, abra o navegador e digite **localhost:8080/address/38400-100**. Observe que a aplicação retorna uma string JSON contendo os dados do endereço. Repita a requisição utilizando um cep inexistente e observe o resultado;

19) Adicionando um novo endereço. Acrescente o método **addAddress**, conforme figura a seguir, para permitir que requisições POST no endereço **/address** contendo um objeto JSON como conteúdo (request body) possam adicionar novos endereços na lista de endereços:

```
no usages
@PostMapping("/address")
public void addAddress(@RequestBody Address address) {
    this.addresses.add(address);
}
```

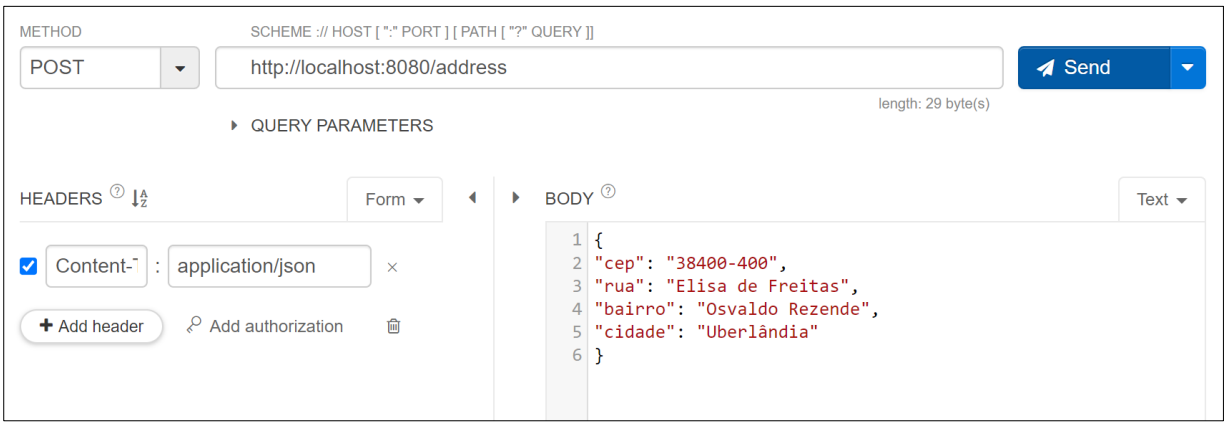
Observe que neste caso foi utilizada a *annotation* **@PostMapping**, uma vez que o objetivo é mapear requisições que utilizarem o método POST. Repare também que foi utilizada a *annotation* **@RequestBody** antes do parâmetro **address** do método. Ela permitirá que a string JSON enviada no corpo da requisição HTTP (payload) seja automaticamente carregada, convertida em um objeto do tipo **Address** e repassada ao parâmetro do método;

20) Para testar rapidamente o método **addAddress** recomenda-se a utilização de um **Cliente HTTP**. Algumas opções:

    a. Software Postman;

    b. Cliente HTTP do próprio IntelliJ IDEA Ultimate;

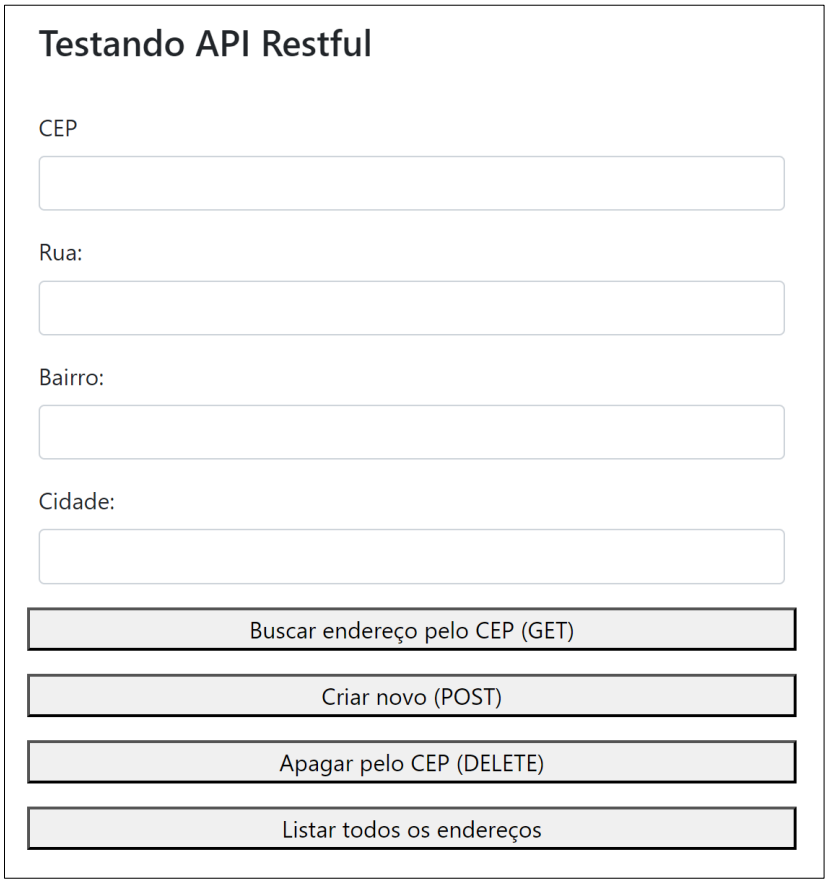
    c. Plugin **Talend API Tester** (mais simples) da Chrome Web Store. Para utilizar o plugin, basta procurar pelo respectivo nome utilizando a opção **Extensões** do navegador. Depois de instalar, clique no ícone de extensões (🧩) e selecione a extensão instalada para abrí-la no navegador. Veja a seguir um exemplo de requisição POST utilizando o Talend API Tester:



21) Adicione um método na classe para permitir que requisições utilizando o método **DELETE** possam excluir os dados de um endereço específico dado o seu cep (por exemplo, utilizando uma requisição DELETE ao endereço /address/38400-100);

22) Para testar os recursos implementados, adicione um arquivo de nome **index.html** na pasta **src/main/resources/static**. Para testar o acesso, digite **localhost:8080/index.html** no navegador.

O arquivo deve ter o formulário apresentado a seguir. As funcionalidades associadas aos botões devem ser implementadas utilizando requisições assíncronas que acessem as funcionalidades desenvolvidas nos passos anteriores (deve-se utilizar a API Fetch com async/await).



- **Buscar endereço pelo CEP**: quando o usuário digitar o CEP e clicar no botão, os demais dados do endereço devem ser preenchidos automaticamente. A requisição deve utilizar a respectiva funcionalidade da API;
- **Criar novo**: os dados preenchidos pelo usuário nos campos do formulário devem ser adicionados ao banco de endereços da API. Deve-se enviar os dados do formulário no formato JSON como parte do corpo da requisição HTTP (veja Módulo 8, slide 86);
- **Apagar pelo CEP**: permite ao usuário remover um endereço da base de endereços da API a partir do CEP digitado no campo CEP;
- **Listar todos os endereços**: todos os endereços correntemente cadastrados devem ser listados na página HTML logo após os botões. Não é necessário nenhuma formatação em especial.

Entrega

Compacte a pasta raiz do projeto e envie pelo Sistema Acadêmico de Aplicação de Testes (SAAT) até a data limite indicada pelo professor em sala de aula.

Sobre Eventuais Plágios

Este é um trabalho individual. Os alunos envolvidos em qualquer tipo de plágio, total ou parcial, seja entre equipes ou de trabalhos de semestres anteriores ou de materiais disponíveis na Internet (exceto os materiais de aula disponibilizados pelo professor), serão duramente

penalizados (art. 196 do Regimento Geral da UFU). Todos os alunos envolvidos terão seus **trabalhos anulados** e receberão **nota zero**.