BEM-VINDOS!!

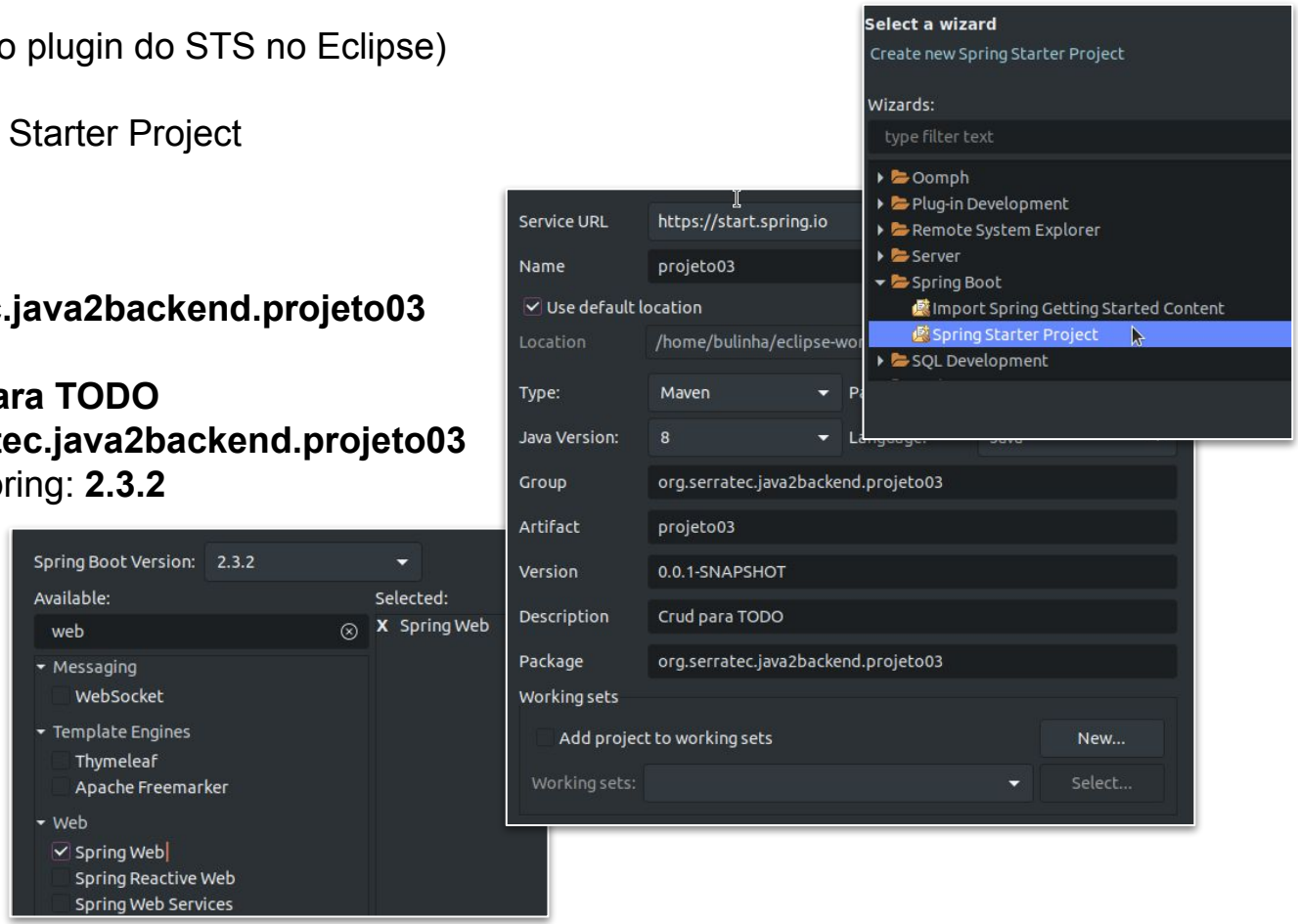
Residência de Software 2022/1

Disciplina: Desenvolvimento de API RestFull

Crud - Projeto

Usando o Spring Tool Suite (ou o plugin do STS no Eclipse)

- File > New > Other …
- Spring Boot > Spring Boot Starter Project
- Preencher com os valores
  - Name: **projeto03**
  - Java Version: **8**
  - Group: **org.serratec.java2backend.projeto03**
  - Artifact: **projeto03**
  - Description: **Crud para TODO**
  - Package: **org.serratec.java2backend.projeto03**
- Selecionar a versão do Spring: **2.3.2**
- Selecionar a dependência
  Spring Web



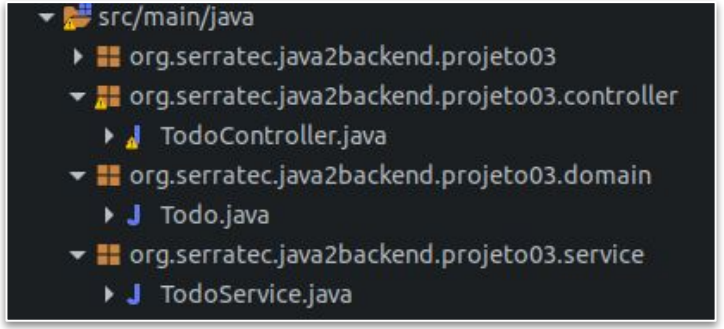
Crud - Componentes

- Criar a classe de dominio Todo com os atributos **id**, **titulo** e **descricao**
- Criar a classe **TodoService** com a anotação **@Service**
- Criar a classe **TodoController** com a anotação **@RestController**

```java
public class Todo {

    private Integer id;
    private String titulo;
    private String descricao;

    public Todo() {
        super();
    }

    public Todo(Integer id, String titulo, Str
        super();
```

```java
@RestController()
@RequestMapping("/todo")
public class TodoController {
    @Autowired
    private TodoService todoService;
```

```java
@Service
public class TodoService {

    private List<Todo> todos = new ArrayList<Todo>();
```



Crud - GET e POST

- Implementação do Service
  - getAll - retorna a própria lista
  - getTodo - iterar a lista para encontrar o “todo” que tenha o mesmo ID passado. Existem várias maneiras:
    - “ForEach” (exemplo ao lado)
    - for(int i; i<lista.size(); i++) {...
    - Iterator it = lista.iterator(); while(it.hasNext()) {...
  - addTodo - adiciona um novo “todo” na lista

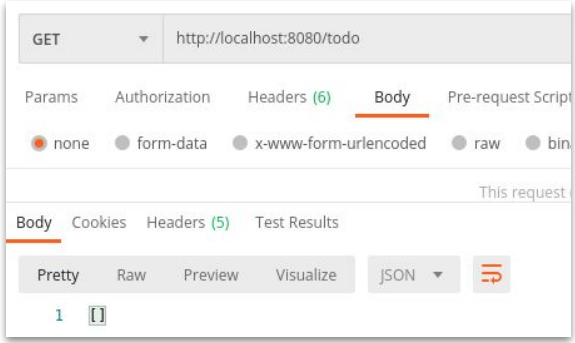
- Implementação do Controller
  - Utilizar GetMapping e PostMapping
    - Possível utilizar RequestMapping com method=GET/POST
    - @PathVariable pode ser utilizado ou não nesse caso
  - Utilizar RequestBody para converter o corpo da requisição HTTP em objeto java

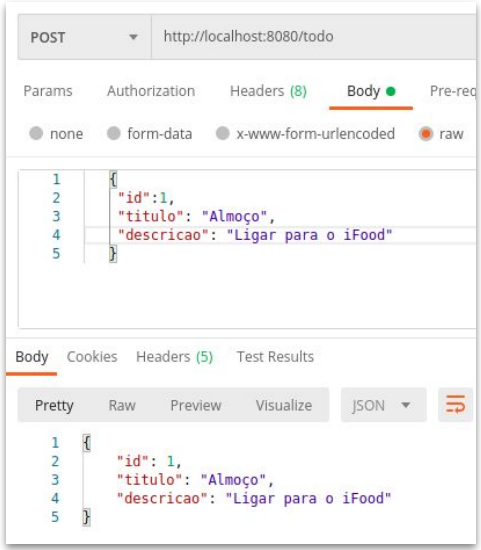
```java
public List<Todo> getAll(){
    return todos;
}
public Todo getTodo(Integer id) {
    Todo achado = null;
    for(Todo todo: todos) {
        if (todo.getId().equals(id)) {
            achado = todo;
            break;
        }
    }
    return achado;
}
public Todo addTodo(Todo todo) {
    todos.add(todo);
    return todo;
}
```

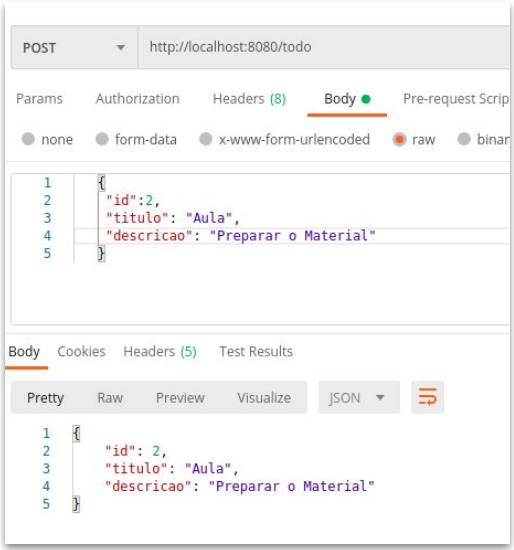
```java
@GetMapping
public List<Todo> getAll() {
    return todoService.getAll();
}
@GetMapping("/{id}")
public Todo getTodo(@PathVariable Integer id) {
    return todoService.getTodo(id);
}
@PostMapping
public Todo postTodo(@RequestBody Todo todo) {
    return todoService.addTodo(todo);
}
```

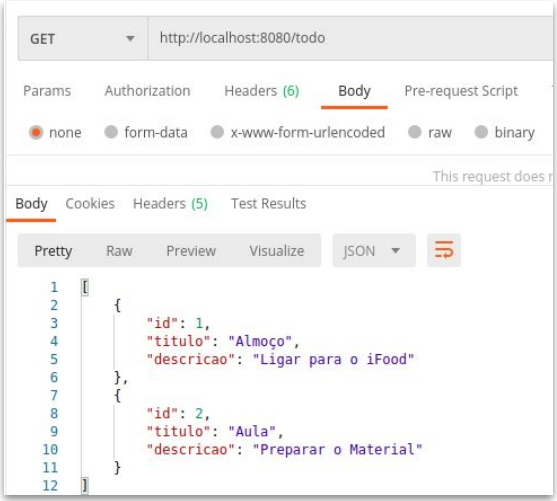
Crud - Teste GET e POST

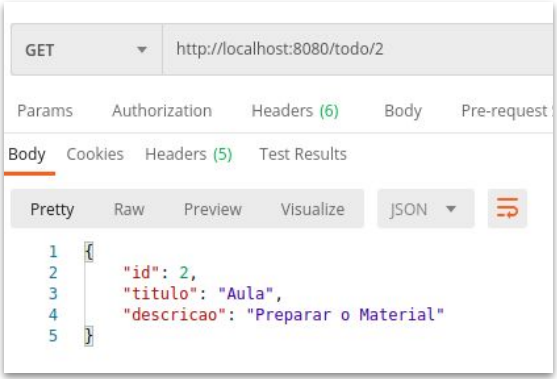
Firjan SENAI | RESIDÊNCIA DE SOFTWARE | serratec Parque Tecnológico Região Serrana











Crud - PUT e DELETE

Implementação no Service dos métodos updateTodo e deleteTodo (assim como o get, é necessário encontrar o “todo” a ser alterado ou excluído iterando a lista.

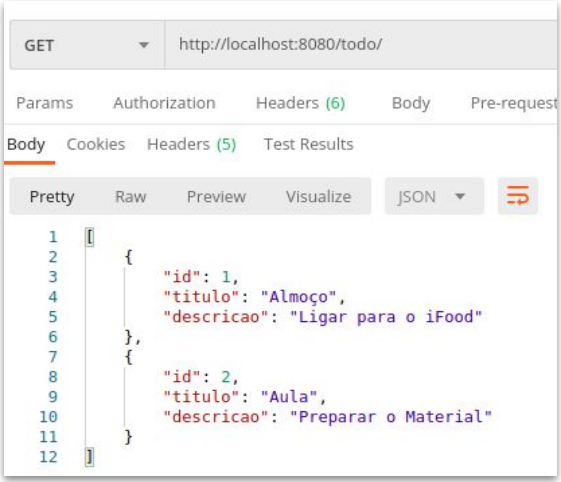
- No Service
  - No update verificar quais atributos foram passados (!=nulll) e somente alterar estes.
    - Atualizar os atributos
    - Retornar o objeto atualizado
  - No delete basta remover o objeto da lista
- No Controller apenas é necessário chamar os métodos do service.

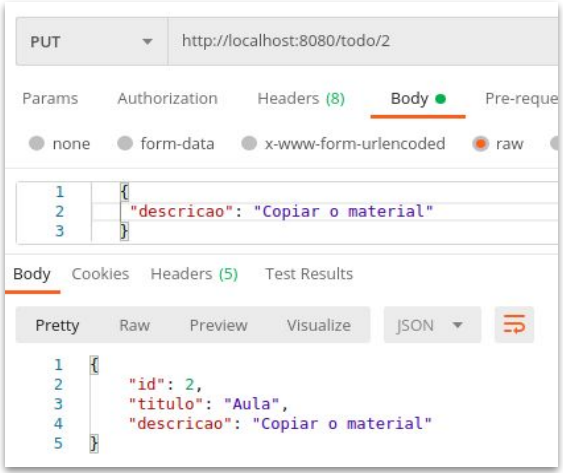
```java
@PutMapping("/{id}")
public Todo putTodo(@PathVariable Integer id, @RequestBody Todo todo) {
    return todoService.updateTodo(id, todo);
}

@DeleteMapping("/{id}")
public void deleteTodo(@PathVariable Integer id) {
    todoService.deleteTodo(id);
}
```

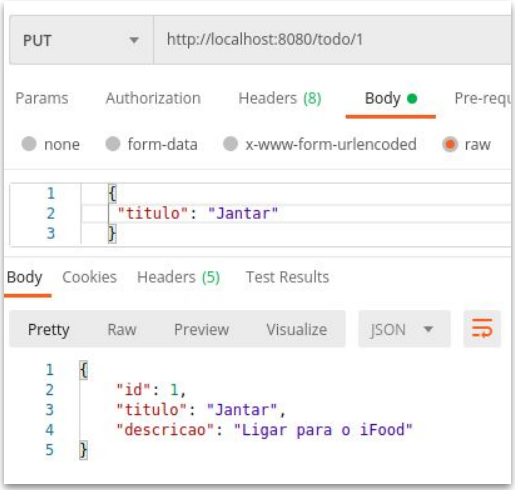
```java
public Todo updateTodo(Integer id, Todo todoNovo) {
    Todo achado = null;
    for(Todo todo: todos) {
        if (todo.getId().equals(id)) {
            achado = todo;
            if (todoNovo.getId()!=null)
                achado.setId(todoNovo.getId());
            if (todoNovo.getTitulo()!=null)
                achado.setTitulo(todoNovo.getTitulo());
            if (todoNovo.getDescricao()!=null)
                achado.setDescricao(todoNovo.getDescricao());
            break;
        }
    }
    return achado;
}
public void deleteTodo(Integer id) {
    for(Todo todo: todos) {
        if (todo.getId().equals(id)) {
            todos.remove(todo);
            break;
        }
    }
}
```

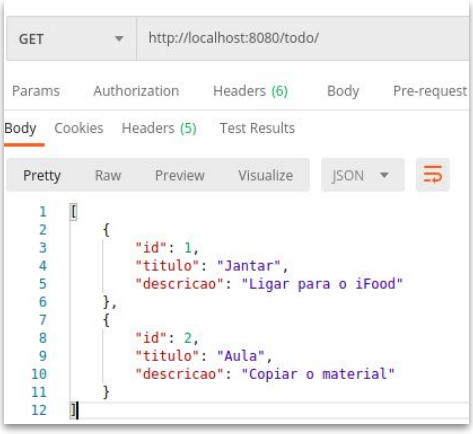
Crud - Teste PUT e DELETE

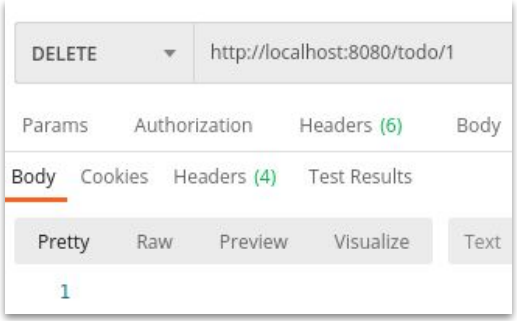
Firjan SENAI | RESIDÊNCIA DE SOFTWARE | serratec Parque Tecnológico Região Serrana

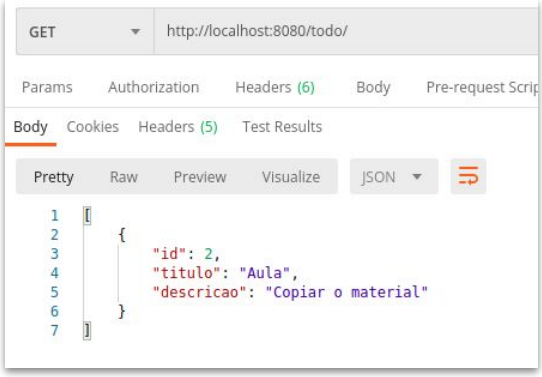




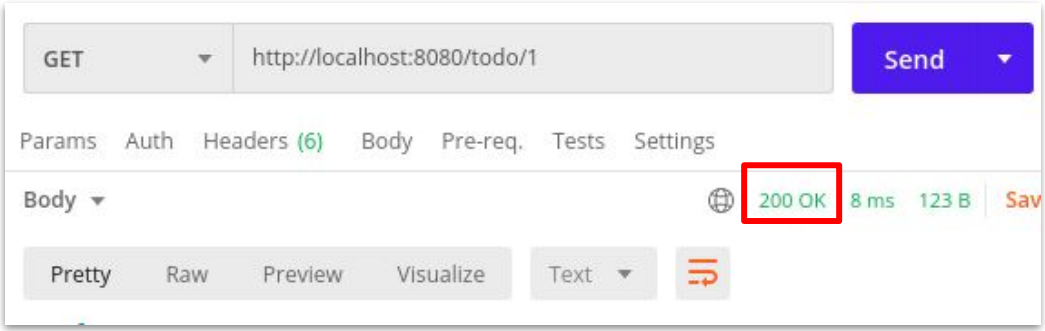


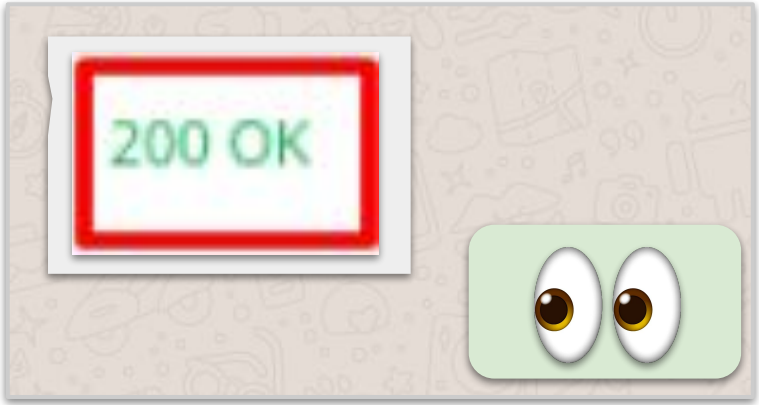


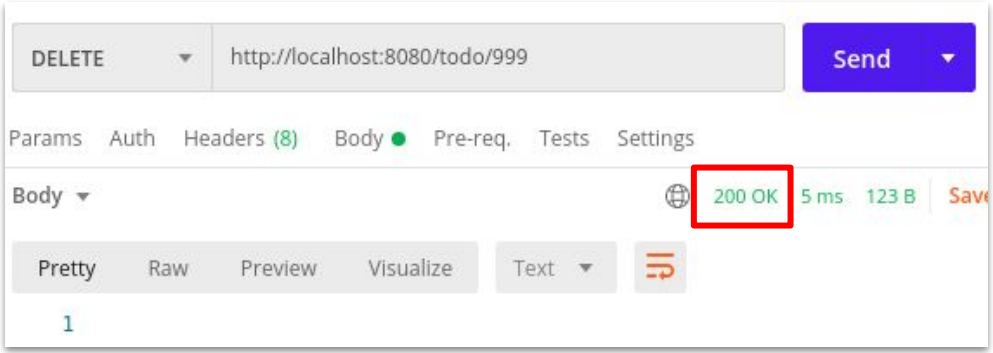


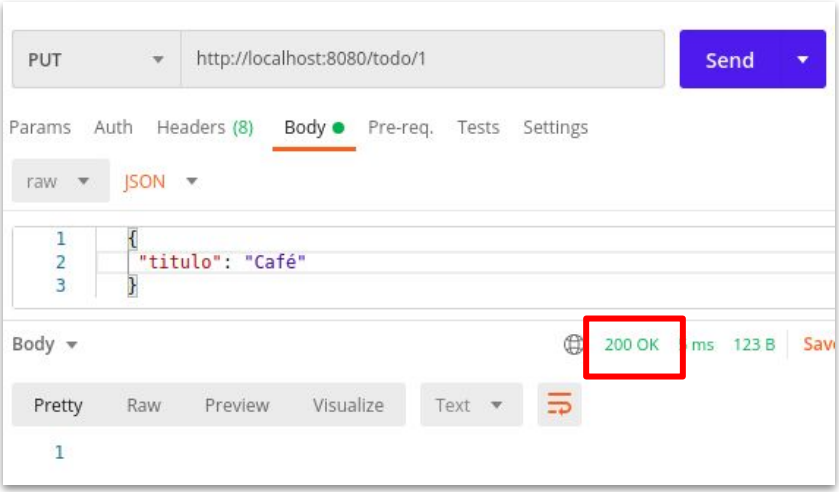


Response Status OK??









ResponseEntity

ResponseEntity é uma classe utilitária do Spring Web para auxiliar na resposta HTTP

Ela é uma classe “Genérica” (suporta generics do java) e recebe como parâmetros no construtor o objeto a ser retornado e um objeto do tipo HttpStatus (que já possui diversas instâncias constantes)

```
@GetMapping("/hello")
public ResponseEntity<String> hello(){
    return new ResponseEntity<>("Olá", HttpStatus.OK);
}
```

Ela possui também uma sintaxe com métodos estáticos alternativos para os principais status:

```
@GetMapping("/hello")
public ResponseEntity<String> hello(){
    return ResponseEntity.ok("Olá");
}
```

ResponseEntity - Bad Request

Exemplo: Realizar tratamento/validação de entrada de dados

```
@GetMapping("/idade")
ResponseEntity<String> calculaIdade(@RequestParam("ano-de-nascimento") int anoDeNascimento) {

    if (anoDeNascimento>LocalDate.now().getYear()) {
        return new ResponseEntity<>("Você não pode nascer no futuro!!!",HttpStatus.BAD_REQUEST);
    }

    int idade = (LocalDate.now().getYear() - anoDeNascimento);
    return new ResponseEntity<>(String.format("Você tem %d anos", idade), HttpStatus.OK);
}
```

É possível também utilizar um método estático da classe ResponseEntity

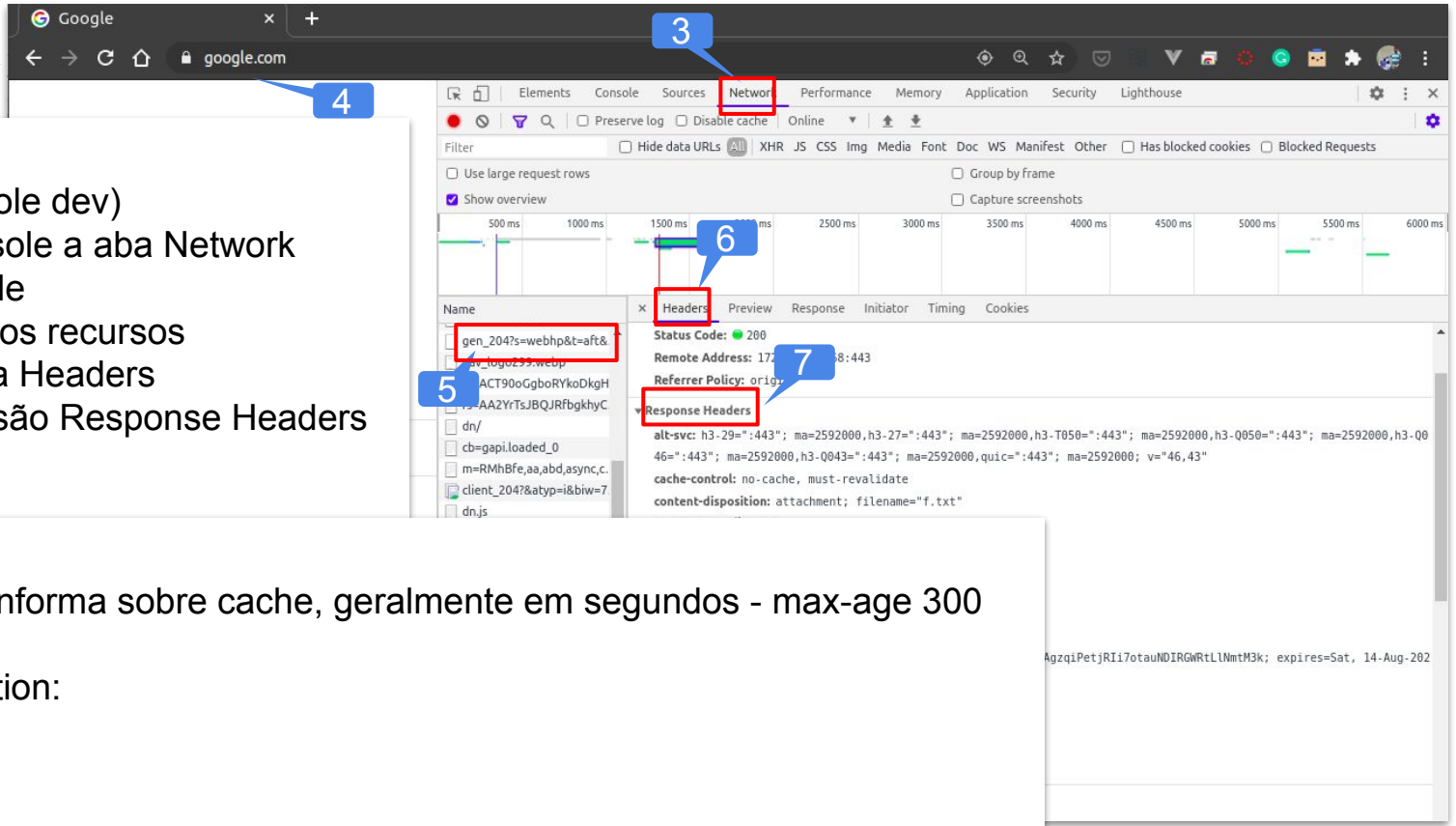
```
@GetMapping("/idade")
ResponseEntity<String> calculaIdade(@RequestParam("ano-de-nascimento") int anoDeNascimento) {

    if (anoDeNascimento>LocalDate.now().getYear()) {
        return ResponseEntity.badRequest().body("Você não pode nascer no futuro!!!");
    }

    int idade = (LocalDate.now().getYear() - anoDeNascimento);
    return ResponseEntity.ok(String.format("Você tem %d anos", idade));
}
```

ResponseEntity

Código	ResponseEntity Method
Padrões	new ResponseEntity<>(HttpStatus.<STATUS>) ResponseEntity.status(HttpStatus.CREATED).build() new ResponseEntity<>(objeto, HttpStatus.<STATUS>) ResponseEntity.status(HttpStatus.CREATED).body(objeto)
200 - Ok	ResponseEntity.ok().build() ResponseEntity.ok(objeto)
201 - Criado HttpStatus.CREATED	*ResponseEntity.created(uri).body(objeto)* *
204 - Sem Conteúdo	ResponseEntity.noContent().build()
400 - Bad Request	ResponseEntity.badRequest().build() ResponseEntity.badRequest().*body(objeto)*
404 - não encontrado	ResponseEntity.notFound().build()

Método created com parâmetro uri deve ser usado com *HATEOAS* tutorial aqui https://www.baeldung.com/spring-hateoas-tutorial
Mais informações sobre os códigos http https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

Cabeçalhos HTTP



1. Abrir Chrome
2. F12 (abre console dev)
3. Acesse no console a aba Network
4. Acesse o Google
5. Selecione um dos recursos
6. Selecione a aba Headers
7. Encontre a sessão Response Headers

Cabeçalhos comuns:

- cache-control: informa sobre cache, geralmente em segundos - max-age 300
- pragma:
- content-disposition:
- content-type:
- date:
- expires:
- status:

Cabeçalhos HTTP

Cabeçalhos comuns:

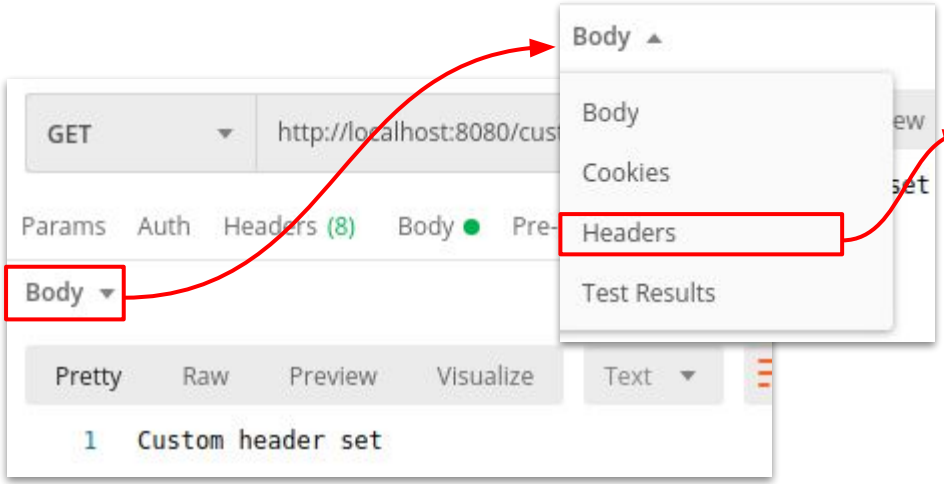
- *cache-control*: informa sobre cache, geralmente em segundos
  - max-age 3600
- *pragma*: geralmente usado para informar que o dado não deve ficar no cache
  - no-cache
- content-disposition: indica se o conteúdo deve ser “baixado” (janela de download) e informa o nome do arquivo
  - attachment; filename="fname.ext"
- content-type: tipo MIME (mídia de internet) do conteúdo
  - text/html; charset=utf-8
- content-length: tamanho em bytes do conteúdo
- date: data e hora do conteúdo
  - Fri, 14 Aug 2020 08:12:31 GMT
- *expires*: quando recurso expira do cache
  - Sun, 16 Aug 2020 16:00:00 GMT
- status: status da requisição

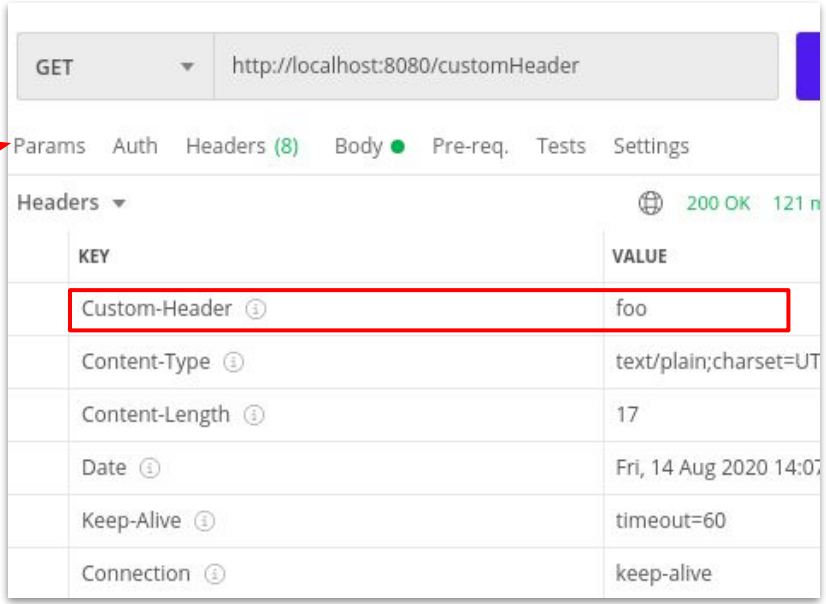
Mais informações https://pt.wikipedia.org/wiki/Lista_de_campos_de_cabe%C3%A7alho_HTTP

ResponseEntity - Header


```java
@GetMapping("/customHeader")
ResponseEntity<String> customHeader() {
    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-Header", "foo");
    String body = "Custom header set";
    return new ResponseEntity<String>( body, headers,  HttpStatus.OK);
}
```

```java
@GetMapping("/customHeader")
ResponseEntity<String> customHeader() {
    return ResponseEntity.ok()
        .header("Custom-Header", "foo")
        .body("Custom header set");
}
```





Refatorando a aplicação

Refactoring is a controlled technique for improving the design of an existing code base.

Fowler, Martin

Refatorar é uma técnica controlada para melhorar o design de uma base de código existente.

Translator, Google

Exception (1)

- Criar Exception Customizada
- Refatorar o método getTodo do TodoService
  - Extraindo o conteúdo dele para um novo método findTodo
  - Alterá-lo para quando não encontrar um “todo” para o Id, ele “dispare” a exception
- Alterar os métodos que buscavam um Todo pelo ID para utilizar o novo método

```
public Todo getTodo(Integer id) {
    Todo achado = null;
    for(Todo todo: todos) {
        if (todo.getId().equals(id)) {
            achado = todo;
            break;
        }
    }
    return achado;
}
```

```
public class TodoNotFoundException extends Exception {

    private static final long serialVersionUID = -7580404419897048916L;
    private Integer id;

    public TodoNotFoundException(Integer id) {
        this.id = id;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

}
```

```
private Todo findTodo(Integer id) throws TodoNotFoundException {
    Todo achado = null;
    for(Todo todo: todos) {
        if (todo.getId().equals(id)) {
            achado = todo;
            break;
        }
    }
    if (achado==null) {
        throw new TodoNotFoundException(id);
    }
    return achado;
}

public Todo getTodo(Integer id) throws TodoNotFoundException {
    return findTodo(id);
}
```

Exception (2)

Os métodos updateTodo e deleteTodo devem utilizar o método findTodo e também propagarem a exception customizada.

```
public Todo updateTodo(Integer id, Todo todoNovo) throws TodoNotFoundException {
    Todo achado = findTodo(id);
    if (todoNovo.getId()!=null)
        achado.setId(todoNovo.getId());
    if (todoNovo.getTitulo()!=null)
        achado.setTitulo(todoNovo.getTitulo());
    if (todoNovo.getDescricao()!=null)
        achado.setDescricao(todoNovo.getDescricao());
    return achado;
}
public void deleteTodo(Integer id) throws TodoNotFoundException {
    Todo todo = findTodo(id);
    todos.remove(todo);
}
```

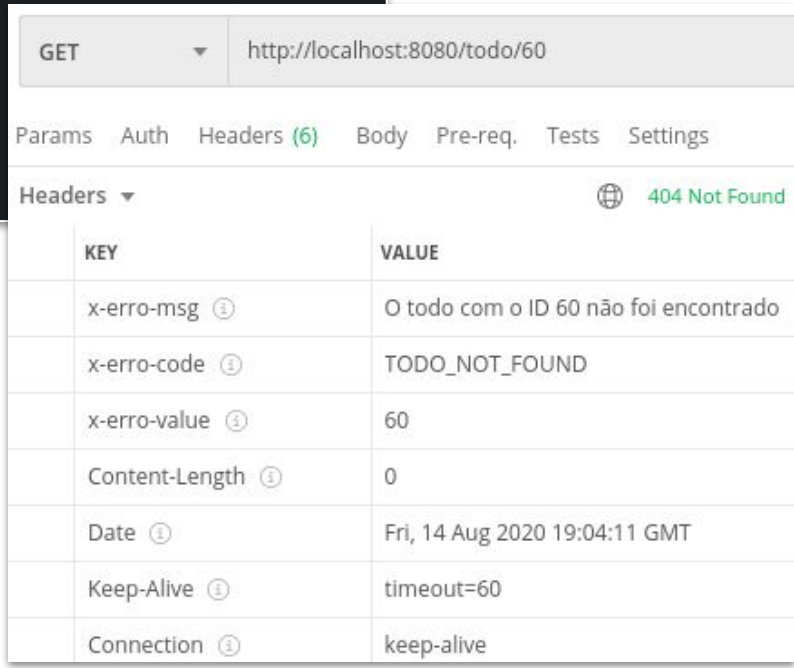
Exceptions (3)

Os métodos do TodoController devem também propagar a exception

```
@GetMapping("/{id}")
public Todo getTodo(@PathVariable Integer id) throws TodoNotFoundException {
    return todoService.getTodo(id);
}

@PutMapping("/{id}")
public Todo putTodo(@PathVariable Integer id, @RequestBody Todo todo) throws TodoNotFoundException {
    return todoService.updateTodo(id, todo);
}

@DeleteMapping("/{id}")
public void deleteTodo(@PathVariable Integer id) throws TodoNotFoundException {
    todoService.deleteTodo(id);
}
```

Exception (4)

```java
@RestControllerAdvice
public class ExceptionsController {

    @ExceptionHandler(TodoNotFoundException.class)
    public ResponseEntity<String> trataTodoNotFound(TodoNotFoundException exception){
        String msg = String.format("O todo com o ID %d não foi encontrado", exception.getId());
        return ResponseEntity.notFound()
                .header("x-erro-msg", msg)
                .header("x-erro-code", "TODO_NOT_FOUND")
                .header("x-erro-value", exception.getId().toString())
                .build();
    }

}
```



- @RestControllerAdvice - “Controller” específico para realizar o tratamento das exceptions
- @ExceptionHandler - indica um método específico para tratar exceções

Arquivo de propriedades

Localizado na pasta resources da aplicação, o arquivo “application.properties” tem o seguinte formato:

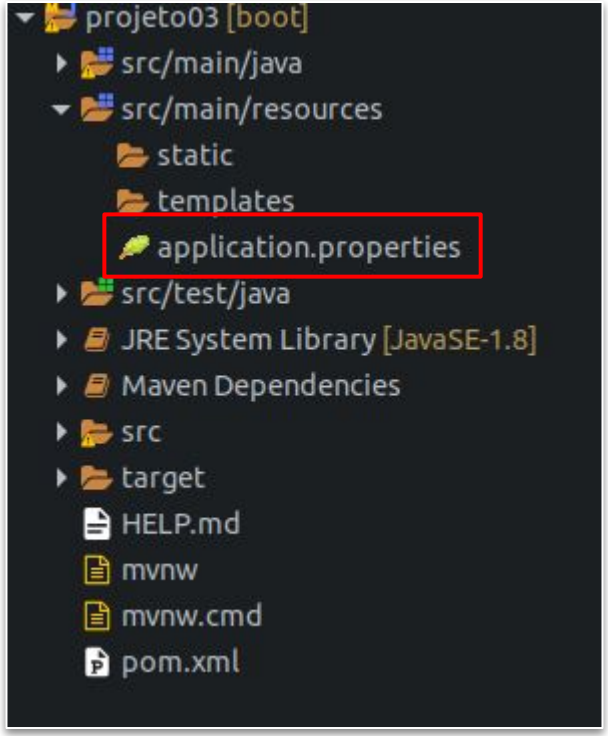
```
propriedade=valor
```

Sendo que “propriedade” pode ser uma “string” simples ou com separação por pontos, como nos nomes de pacotes:

```
prop.subprop=valor
```

Pode ser utilizado o padrão YAML, neste caso o nome do arquivo seria “application.yaml” e teria o formato:

```
prop:
   subprop:  valor
```



Arquivo propriedades

Propriedades comuns do Spring Boot:

server.port	Porta do servidor (8080)
spring.datasource.url	URL jdbc do banco
spring.datasource.username	Usuário para acesso ao banco
spring.datasource.password	Senha para acesso ao banco

application.properties

```
1 server.port=9090
2
```

```
 .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.2.RELEASE)

2020-08-14 16:47:22.914  INFO 1156273 --- [           main] o.s.j.projeto03.Projeto03Application     : Starting Projeto03Application on bu
2020-08-14 16:47:22.916  INFO 1156273 --- [           main] o.s.j.projeto03.Projeto03Application     : No active profile set, falling back
2020-08-14 16:47:23.428  INFO 1156273 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 9090 (http)
2020-08-14 16:47:23.433  INFO 1156273 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-08-14 16:47:23.433  INFO 1156273 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.37]
2020-08-14 16:47:23.462  INFO 1156273 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationCont
2020-08-14 16:47:23.463  INFO 1156273 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization compl
2020-08-14 16:47:23.564  INFO 1156273 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExe
2020-08-14 16:47:23.660  INFO 1156273 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 9090 (http) with cont
2020-08-14 16:47:23.665  INFO 1156273 --- [           main] o.s.j.projeto03.Projeto03Application     : Started Projeto03Application in 0.947 seconds (
```

Arquivo de propriedades

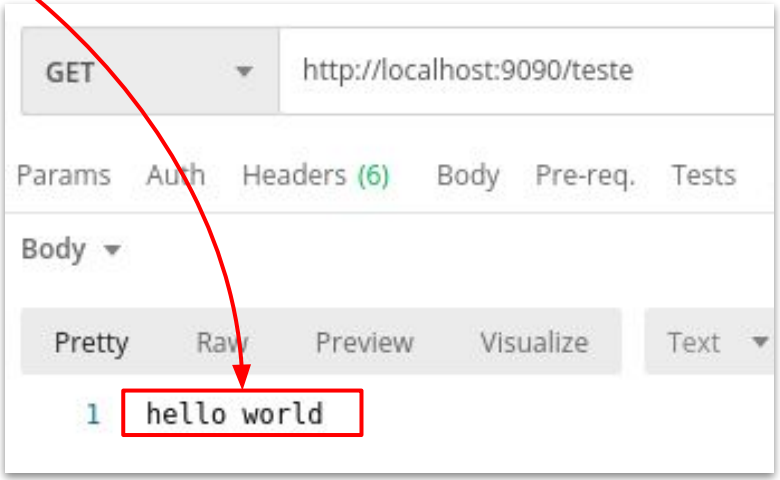
Propriedades customizadas

```
server.port=9090

minha-propriedade=hello world
```

```
@Value("${minha-propriedade}")
private String propriedadeCustomizada;

@GetMapping("/teste")
public String retornaMinhaPropriedade() {
    return propriedadeCustomizada;
}
```

IMPORTANTE

O nome da propriedade deve estar entre ${..} na anotação @Value.



Arquivo de Propriedades

É possível ter uma classe complexa preenchida pelo arquivo de configuração do Spring Boot

application.properties

```
aplicacao.valor1=1500
aplicacao.valor2=Teste de valor
aplicacao.array1=a,b,c
aplicacao.array2[0]=1500
aplicacao.array2.1=1300
aplicacao.map.chave1=Primeira chave
aplicacao.map.chave2=Segunda Chave
aplicacao.lista=1,2,3
aplicacao.todos[0].id=1
aplicacao.todos[0].titulo=gravar
aplicacao.todos[0].descricao=gravar a aula
```

"Bean" de configuração

```
@Component
@ConfigurationProperties("aplicacao")
public class AplicacaoConfig {

    private int valor1;
    private String valor2;
    private String[] array1;
    private int[] array2;
    private Map<String,String> map;
    private List<Integer> lista;
    private List<Todo> todos;
```

Controller

```
@Autowired
private AplicacaoConfig aplicacaoConfig;

@GetMapping("/config")
public AplicacaoConfig retornaConfig() {
    return aplicacaoConfig;

}
```

GET http://localhost:9090/config

Params Auth Headers (6) Body Pre-req. Tests Settings

Body

Pretty Raw Preview Visualize JSON


```
{
    "valor1": 1500,
    "valor2": "Teste de valor",
    "array1": [
        "a",
        "b",
        "c"
    ],
    "array2": [
        1500,
        1300
    ],
    "map": {
        "chave1": "Primeira chave",
        "chave2": "Segunda Chave"
    },
    "lista": [
        1,
        2,
        3
    ],
    "todos": [
        {
            "id": 1,
            "titulo": "gravar",
            "descricao": "gravar a aula"
        }
    ]
}
```

Prioridade de Propriedades

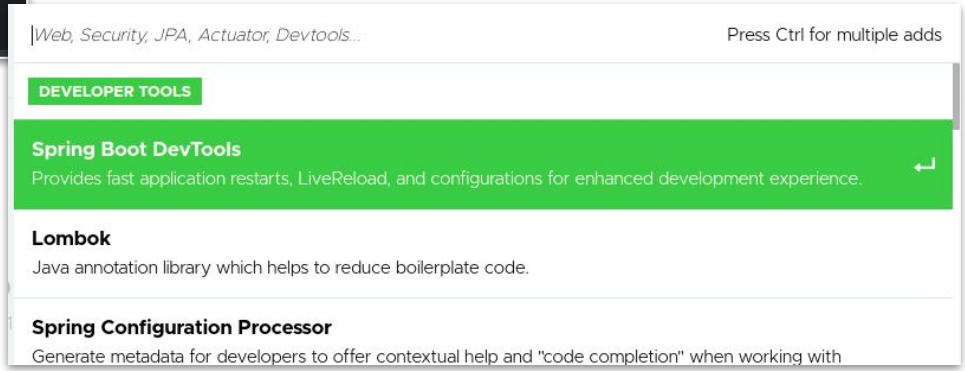
1. **Parâmetros de linha de comando**
   java -jar minhaApp.jar --minha-propriedade=valor
1. Atributos JNDI em java:comp/env (apenas para aplicações em servidores JEE)
2. Java System properties (System.getProperties()).
   System.setProperty(<prop>,<valor>)
1. **Variáveis do Sistema Operacional**.
   Win: set propriedade=valor / ou Painel de Controle - Variáveis do ambiente…
   Linux: propriedade=valor / export propriedade=valor
1. **Um arquivo de profile específico *fora* do JAR quando for ativado profile**
   application-{profile}.properties / application-dev.properties
1. **Um arquivo de profile específico *dentro* do JAR quando for ativado profile**
   application-{profile}.properties / application-dev.properties
1. **O arquivo application.properties *fora* do jar**
2. **O arquivo application.properties *dentro* do jar**
3. Valor da propriedade padrão do Spring

DevTools

DevTools é uma feature do Spring Boot com o foco em aumentar a produtividade durante o desenvolvimento de aplicações

projeto03/pom.xml

```
20
21    <dependencies>
22        <dependency>
23            <groupId>org.springframework.boot</groupId>
24            <artifactId>spring-boot-starter-web</artifactId>
25        </dependency>
26
27        <dependency>
28            <groupId>org.springframework.boot</groupId>
29            <artifactId>spring-boot-devtools</artifactId>
30        </dependency>
31
```



Diversas propriedades são incluídas em tempo de execução para acelerar o desenvolvimento

- Desabilitar cache de recursos estáticos (páginas html, imagens, etc…)

Restart automático - Após iniciar a aplicação, ao alterar alguma classe ou arquivo o Spring Boot irá reinicializar seu servidor interno recarregando a aplicação

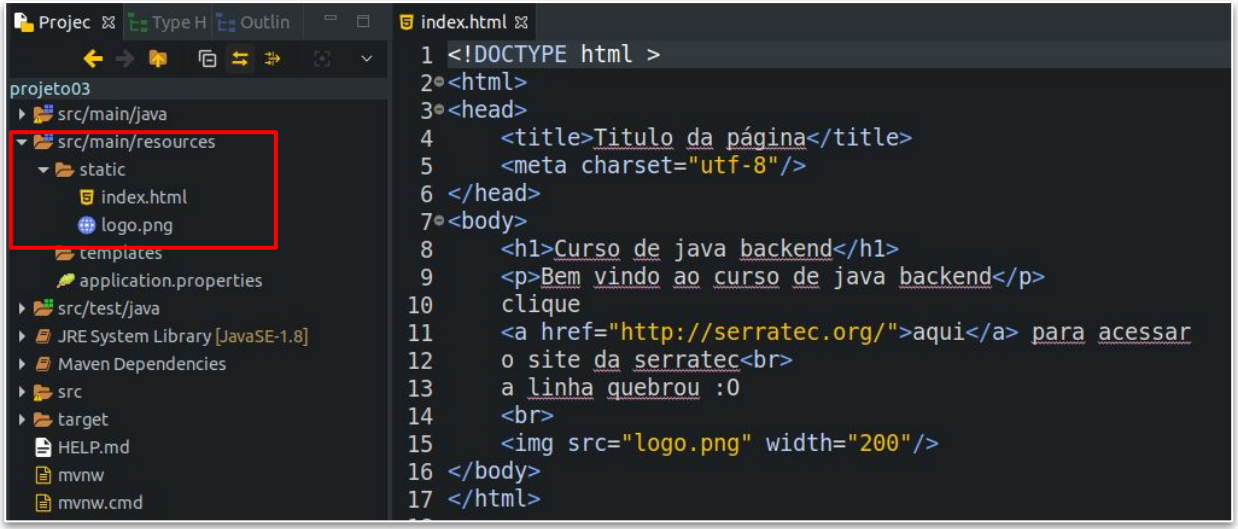
Livereload - Para aplicações com páginas web, ele pode forçar o browser a recarregar uma página (necessita de extensão no browser - na ChromeWebStore: RemoteLiveReload)

Mais informações https://www.baeldung.com/spring-boot-devtools

Conteúdo estático

Arquivos e páginas html estáticas podem ser disponibilizadas pela aplicação se estiverem em uma destas pastas (/src/main/resources):

- /META-INF/resources/
- /resources/
- /static/
- /public/



Exercício

Crie uma aplicação para gerenciar contas bancárias:

- Entidade: Conta
  - numero
  - titular
  - saldo
- Operacao
  - Tipo (debito / credito)
  - valor
- API
  - GET /conta - lista todas as contas
  - GET /conta/<numero> - retorna a conta com o número passado
  - POST /conta - insere uma nova conta
  - PUT /conta/numero - atualiza a conta (somente nome do titular e numero)
  - POST /conta/numero/operacao - recebe uma operacao com parâmetro e a partir dela atualiza o saldo
  - DELETE /conta/numero - remove a conta cujo número foi passado
- Tratamento de erro para saldo insuficiente (operação de débito com valor maior do que o saldo)