


Instituto Politécnico de Castelo Branco
Escola Superior de Tecnologia

APIs Spring Boot REST

Unidade Curricular

Aplicações Internet Distribuídas / Aplicações Distribuídas

Licenciatura em Engenharia Informática
Mestrado em Desenvolvimento de Software e Sistemas Interactivos
UTC Informática
Est-IPCB
Ano Letivo 2022/2023

Prof.º Doutor Alexandre Fonte
(adf@ipcb.pt)

Versão: 14 outubro de 2022

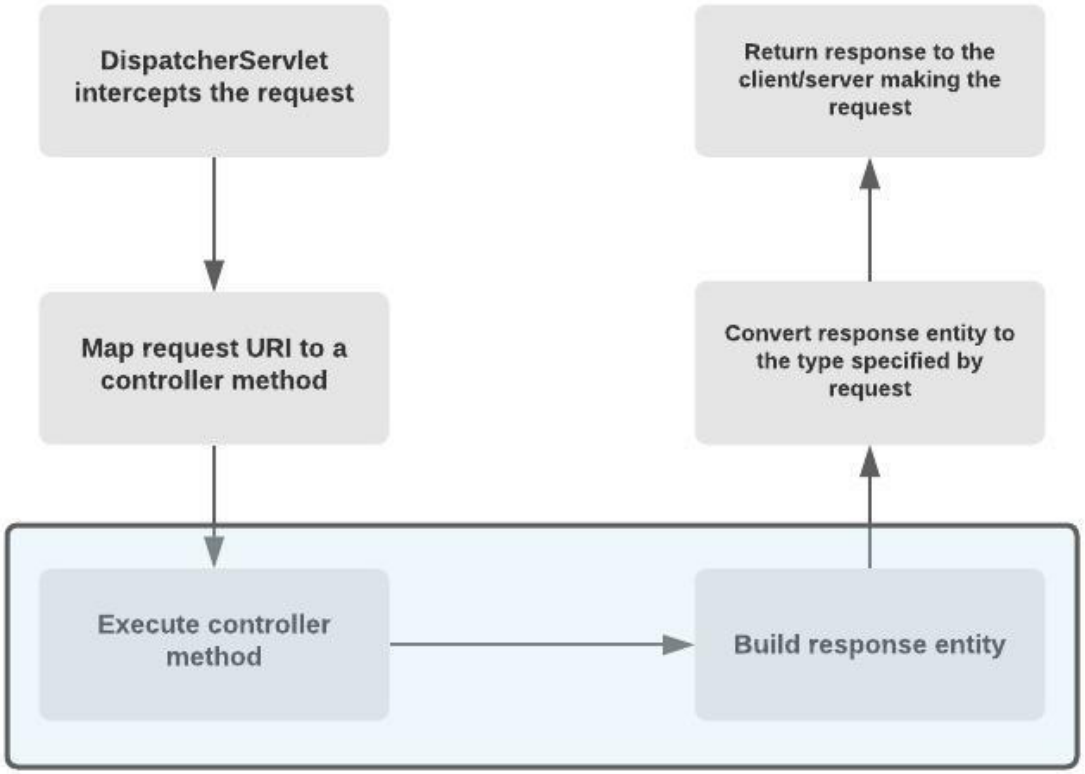
Sumário

- APIs Spring Boot REST
- Anotações MVC e REST
- Controladores REST
- Autogeração da Documentação Swagger/Open API
- Basic Authentication with Spring Security
- Security with SSL

Spring MVC e Spring Web

Workflow de uma API Spring Boot REST

- Antes de se estudar as anotações (@Controller e @RestController), vamos rapidamente percorrer o fluxo de trabalho de como a Spring Boot lida com os pedidos REST API, os processa, e devolve uma resposta:



10 Anotações Spring MVC e REST Essenciais

- @Controller
- @RequestMapping
- @RestController
- @ResponseBody
- @ResponseStatus
- @RequestParam
- @PathVariable

Anotações @Controller, @ResponseBody e @RequestMapping

- **@Controller** é uma anotação especializada que torna uma instância da classe anotada parte da camada de negócio ou de apresentação.
- Esta permite informar o Front Controller (DispatcherServlet) para incluir o novo controlador.
- Uma classe decorada com **@Controller** conjuntamente com **@ResponseBody** cria um controlador REST, uma API REST
- A **@RequestMapping** permite especificar um mapeamento entre um URI e um método do controlador (e.g., /api/tree)
- A anotação **@ResponseBody** deve decorar o retorno do método, por forma a converter a resposta em JSON / XML antes de ser retornada para o cliente.

Anotações @Controller, @ResponseBody e @RequestMapping

```
@Controller
@ResponseBody
@RequestMapping("/api/tree")
public class UserController {
    @Autowired
    private UserRepository repository;

    @GetMapping("/{id}")
    public User getUserById(@PathVariable int id) {
        return repository.findById(id);
    }

    @GetMapping
    public User getUserByNameAndAge(
                        @RequestParam String name,
                        @RequestParam int age) {
        return repository.findFirstByCommonNameIgnoreCaseAndAge(name, age);
    }
}
```

URI Base: http://localhost:8080/api/tree

Anotação @RestController (nova anotação após Spring 4.0)

- The @RestController annotation in Spring is essentially just a combination of @Controller and @ResponseBody.
- This annotation was added during Spring 4.0 to remove the redundancy of declaring the @ResponseBody annotation in your controller, which means that instead of rendering pages, it'll just respond with the data we've given it. This is natural for REST APIs - returning information once an API endpoint has been hit.

```
@RestController
@RequestMapping(“/api/user")
public class UserController {
    @Autowired
    private UserRepository repository;

    @GetMapping("/{id}")
    public User getUserById(@PathVariable int id) {
        return repository.findById(id);
    }

    @GetMapping
    public User getUserByNameAndAge(
                        @RequestParam String name,
```

Exemplo

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

    @Autowired
    private UserRepository userRepository;

    @GetMapping
    public List<User> findAllUsers() {
        // Implement
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
        // Implement
    }

    @PostMapping
    public User saveUser(@Validated @RequestBody User user) {
        // Implement
    }
}
```

Passagem de Parâmetros @RequestParam e @PathVariable

- Anotações usadas ligar/injectar os parâmetros HTTP nos parâmetros dos métodos do controlador.
- Parâmetros Query: @RequestParam

```
URL: http://localhost:8080/api/user?id=900848893
```

```
@GetMapping("api/user")
public ResponseEntity<User> findUserById(@RequestParam(value = "id") long id) {
    Optional<User> user = userRepository.findById(id);
        return ResponseEntity.ok().body(user.get());
}
```

- Parâmetros Path (ou URLs parameterizados): @PathVariable

```
URL: http://localhost:8080/api/user/900083838
```

```
@GetMapping("api/user/{id}")
public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
    Optional<User> user = userRepository.findById(id);
        return ResponseEntity.ok().body(user.get());
}
```

@RequestBody e @ResponseBody

- **@RequestBody** - converte os dados que chegam numa mensagem HTTP em objectos Java a serem passados ao método do controlar.
- **@ResponseBody** - Já esta, indica ao Spring MVC para usar um conversor para um formato de representação antes de responder ao cliente. Por omissão, converte para Json.

Exemplo

```
@PostMapping(
        value="api/user"
        consumes ="application/json")
public @ResponseBody User saveUser(@RequestBody User user) {
   return userRepository.save(user);
}
```

@RequestBody e @ResponseBody (Cont…)

- O conversor Json utilizado por omissão pelo Spring é o Jackson. No caso do XML, é preciso incluir a dependência (caso esteja disponível):

```
<dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-xml</artifactId>
    </dependency>
```

- Caso não esteja disponível pode obter por usar a JAX-B do JEE, sendo preciso também anotar as classes modelo de dados com @XmlRootElement, e os atributos XML com @XmlAttribute (ver slide seguinte):

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
</dependency>
```

@RequestBody e @ResponseBody (Cont…)

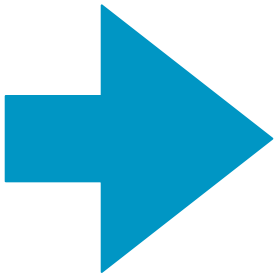
- O conversor Json utilizado por omissão pelo Spring é o Jackson. No caso do XML, é preciso incluir a dependência (caso esteja disponível):

```
@XmlRootElement
public class Conta {

    private long id;
    private String titular;
    private String morada;
    private long nif;
    private long pin;
    private double saldo;

    public Conta() {
    }

    @XmlAttribute
    public long getId() {
        return id;
    }…..}
```

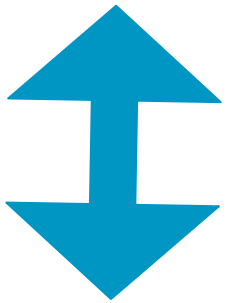


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"
<conta id="1">
    <morada>Castelo Branco</morada>
    <nif>123456</nif>
    <pin>1234</pin>
    <saldo>2000.0</saldo>
    <titular>alexandre</titular>
</conta>
```

Anotações Especializadas @GetMapping, @PostMapping …

- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping são uma especialização da anotação @RequestMapping

```
@RequestMapping(value = "/pessoa", method = RequestMethod.GET)
    public List<Pessoa> Get() {
        return _pessoaRepository.findAll();
    }
```

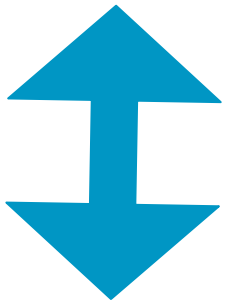


```
@GetMapping(value = “/pessoa")
    public List<Pessoa> Get() {
        return _pessoaRepository.findAll();
    }
```

Anotações Especializadas @GetMapping, @PostMapping …

- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping são uma especialização da anotação @RequestMapping

```
@RequestMapping(value = "/pessoa", method =  RequestMethod.POST)
    public Pessoa Post(@Valid @RequestBody Pessoa pessoa)
    {
        return _pessoaRepository.save(pessoa);
    }
```



```
@PostMapping(value = "/pessoa")
    public Pessoa Post(@Valid @RequestBody Pessoa pessoa)
    {
        return _pessoaRepository.save(pessoa);
    }
```

Personalização dos Códigos de Estado (Http Status Codes) - Opção 1 usando a Classe ResponseEntity

- ResponseEntity represents the whole HTTP response: status code, headers, and body. As a result, we can use it to fully configure the HTTP response.
- If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.
- ResponseEntity is a generic type. Consequently, we can use any type as the response body:

```
@GetMapping("/{id}")
public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
    Optional<User> user = userRepository.findById(id);

    if(user.isPresent()) {
        return ResponseEntity.ok().body(user.get());
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

Personalização dos Códigos de Estado (Http Status Codes) - Opção 2 usando a anotação @ResponseStatus

- If we want to specify the **response status of a controller method**, we can mark that method with @ResponseStatus. It has two interchangeable arguments for the desired response status: code, and value.
- **This annotation can be used to override the HTTP response code** for a response. You can use this annotation for error handling while developing a web application or RESTful web service using Spring.

Exemplo:

```
@ResponseStatus(HttpStatus.OK)
@RequestMapping(value = "/getconta/{id}",produces = "application/json",method =
RequestMethod.GET)
public Conta getContaById(@PathVariable("id") long id) {
   return servico.procuraContaPorId(id).orElseThrow(
           ()->new ResponseStatusException(HttpStatus.NOT_FOUND, "not found"));
}
```

A minha terceira Aplicação Spring Boot com Spring Data JPA + Spring WEB (MVC e REST)

- **Atividade Prática n.º7:** Add Controladores REST Controller to **VendasSpringDataJpa** application from Activity n.º6.

A minha terceira Aplicação Spring Boot com Spring Data JPA + Spring WEB (MVC e REST)

- **Passo 1:** Check if pom.xml has the spring-boot-starter-web.
- **Passo 2:** Create a package named rest.controllers, and the class for the REST controller, named **ControladorCliente**.
- **Passo 3:** Perform the planning of the URIs structure according to some of the REST recommendations for the case of access to the Clientes resources
- **Passo 4:** Implement **v1** version for the **ControladorCliente** REST controlador by using the non-specialised annotations and the return of the Http status codes with **ResponseEntity** class:
  - @Controller
  - @RequestMapping
  - @ResponseBody

A minha terceira Aplicação Spring Boot com Spring Data JPA + Spring WEB (MVC e REST)

- **Step 5:** Implement a **v2** version for **ControllerClient** REST controller in which the specialised annotations are used and the return of Http state codes using the **@ResponseStatus** annotation:
  - -@RestController
  - -@GetMapping, @PostMapping, etc
  - -@ResponseStatus
  - -+ Class ResponseStatusException

Note: You can use Jackson annotation **@JsonIgnore** to ignore Json of Pedidos when you return a Cliente.

A minha terceira Aplicação Spring Boot com Spring Data JPA + Spring WEB (MVC e REST)

- **Step 6:** Perform the planning of the URIs according to some of the REST recommendations for the case of access to the Pedidos resources.
  - You should consider the fact Pedido has a Many-to-One relationship with Cliente Entity.
- **Step 7:** Implement a **v2** version for **ControllerPedido** REST controller in which the specialised annotations are used and the return of Http state codes using the **@ResponseStatus** annotation.

Autogeneration of Swagger/Open API Documentation

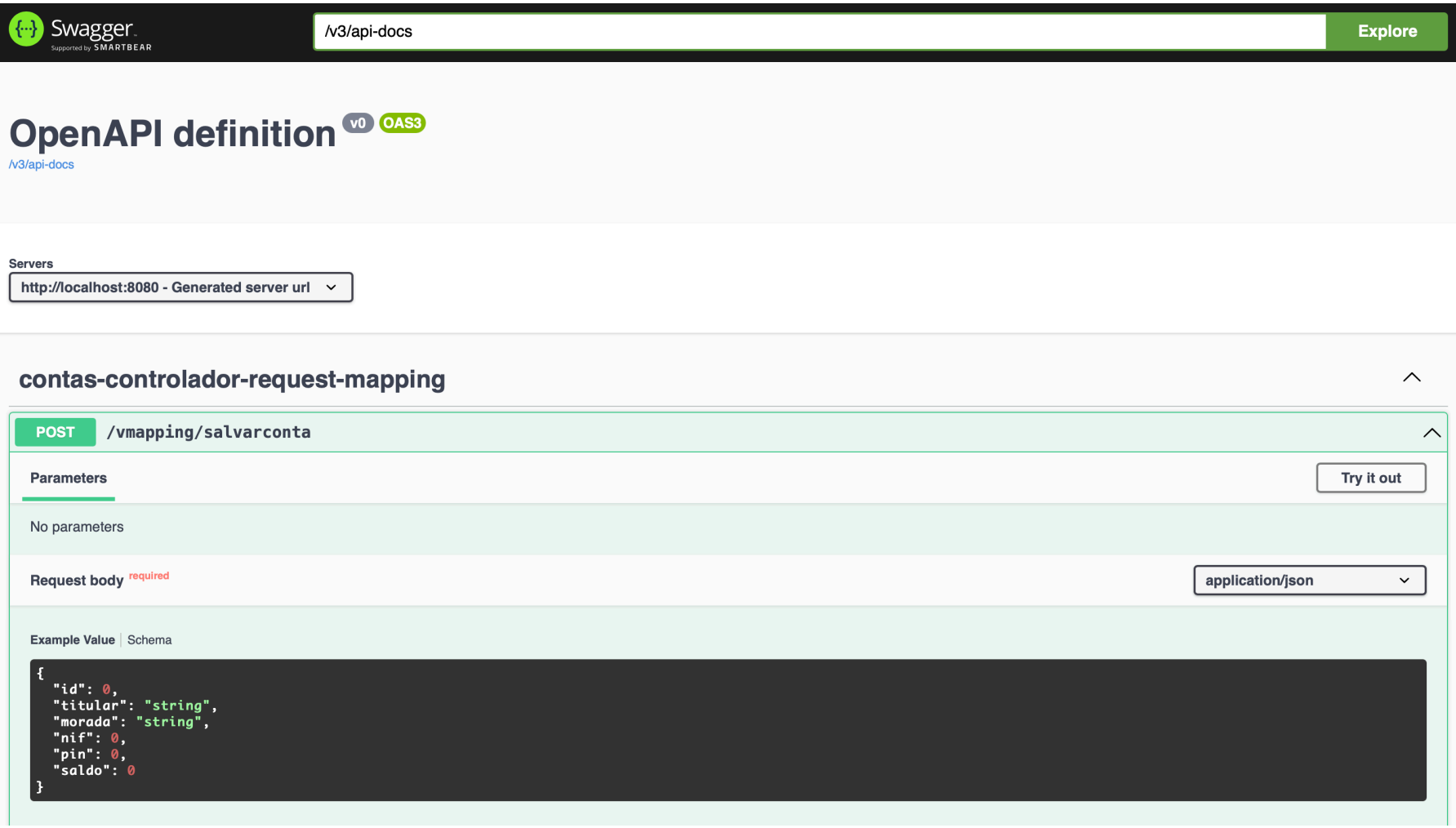
- We can use the framework springdoc-openapi
  - https://springdoc.org
- For the integration between spring-boot and swagger-ui, add the library to the list of your project dependencies:

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.6.6</version>
</dependency>
```

- The Swagger UI page will then be available at http://server:port/context-path/swagger-ui.html and the OpenAPI description will be available at the following url for json format: http://server:port/context-path/v3/api-docs
  - server: The server name or IP
  - port: The server port
  - context-path: The context path of the application

Example:
http://localhost:8080/swagger-ui/index.html

Autogeneration of Swagger/Open API Documentation



Monitoring API with Spring Boot Actuator

- Add the Actuator starter:

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

- Type: http://localhost:8080/actuator
- Para expor os links (informativos do actuator), adicionar a propriedade no ficheiro application.properties

```
management.endpoints.web.exposure.include=*
```

Nota: Se usar chrome instalar um Json Viewer: https://chrome.google.com/webstore/category/extensions

Monitoring API with Spring Boot Actuator

- Expose actuator /info by filling and enabling the info environment variable in application.properties file e inclua a informação sobre:
  - nome da aplicação, descrição, versão e versão do Java,

```
## Configuring info endpoint for Atuator
info.app.name=VendasSpringDataJpa
info.app.description=This is my first spring boot
info.app.version=1.0.0
## Expose all actuator endpoints
management.endpoints.web.exposure.include=*
## Expose info Environment Variable
management.info.env.enabled = true
info.java-vendor = ${java.specification.vendor}
```

- To test, re-run, go again Eureka web page and clique on application info link

  Nota: Se usar chrome instalar um Json Viewer: https://chrome.google.com/webstore/category/extensions

Implementing Basic Authentication with Spring Security

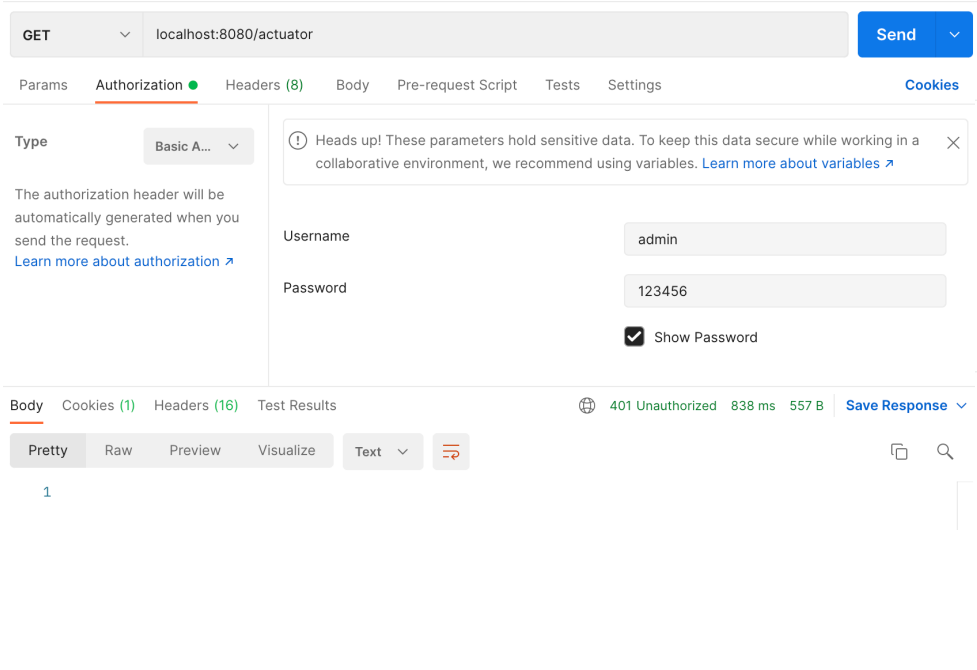
- The addition of Basic Auth in Spring Boot is almost trivial.
- First add the following dependency, and run de application. Below we can see an exemple of a default password created by Spring:

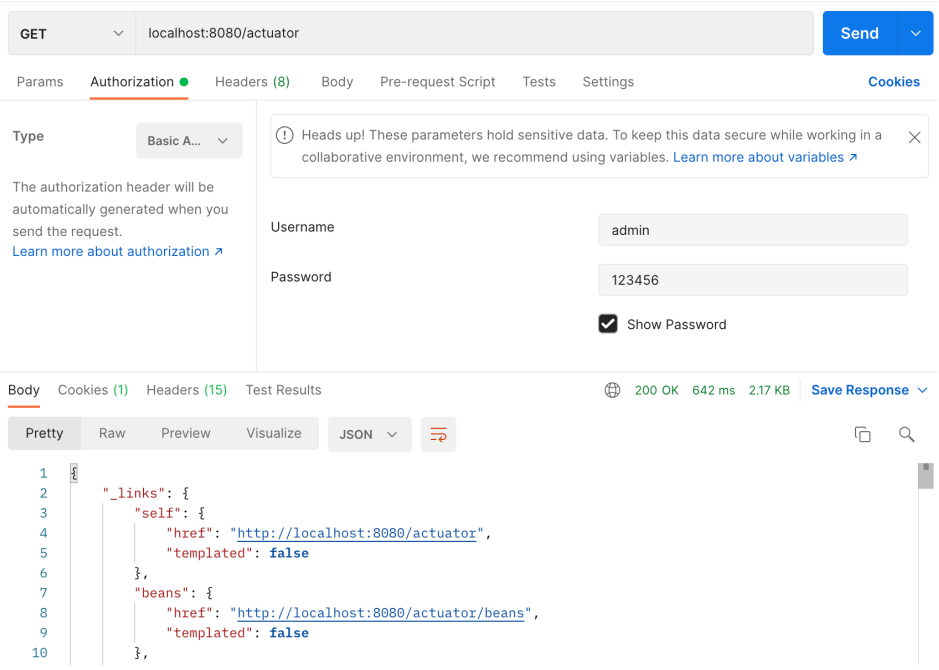
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Implementing Basic Authentication with Spring Security (2)

- If you would like to customize your Api credentials fill on the application.properties the following:

```
spring.security.user.name=admin
spring.security.user.password=123456
```





How to Secure a Rest API with HTTPs

- **Step 1:** Create a X.509 Certificate
  - In production-grade applications, certificates are issued from renowned Certification Authorities (CA) to ensure that our application is a trusted entity.
  - However, we can create a Self-Signed Certificate for our application by using Java **keytool** available the JDK_HOME/bin directory

```
keytool -genkey -keyalg RSA -alias mycertificatex509 -keystore
mycertificatex509.jks -storepass password -validity 365 -keysize 4096
-storetype pkcs12
```

- Using the RSA algorithm
- Providing an alias name as mycertificatex509
- Naming the Keystore file as mycertificatex509.jks
- Validity for one year: 365
- Storepass: password

How to Secure a Rest API with HTTPs

- **Step 2:** Add the Certificate to the Project
  - Copy to the directory src/main/resources at the classpath

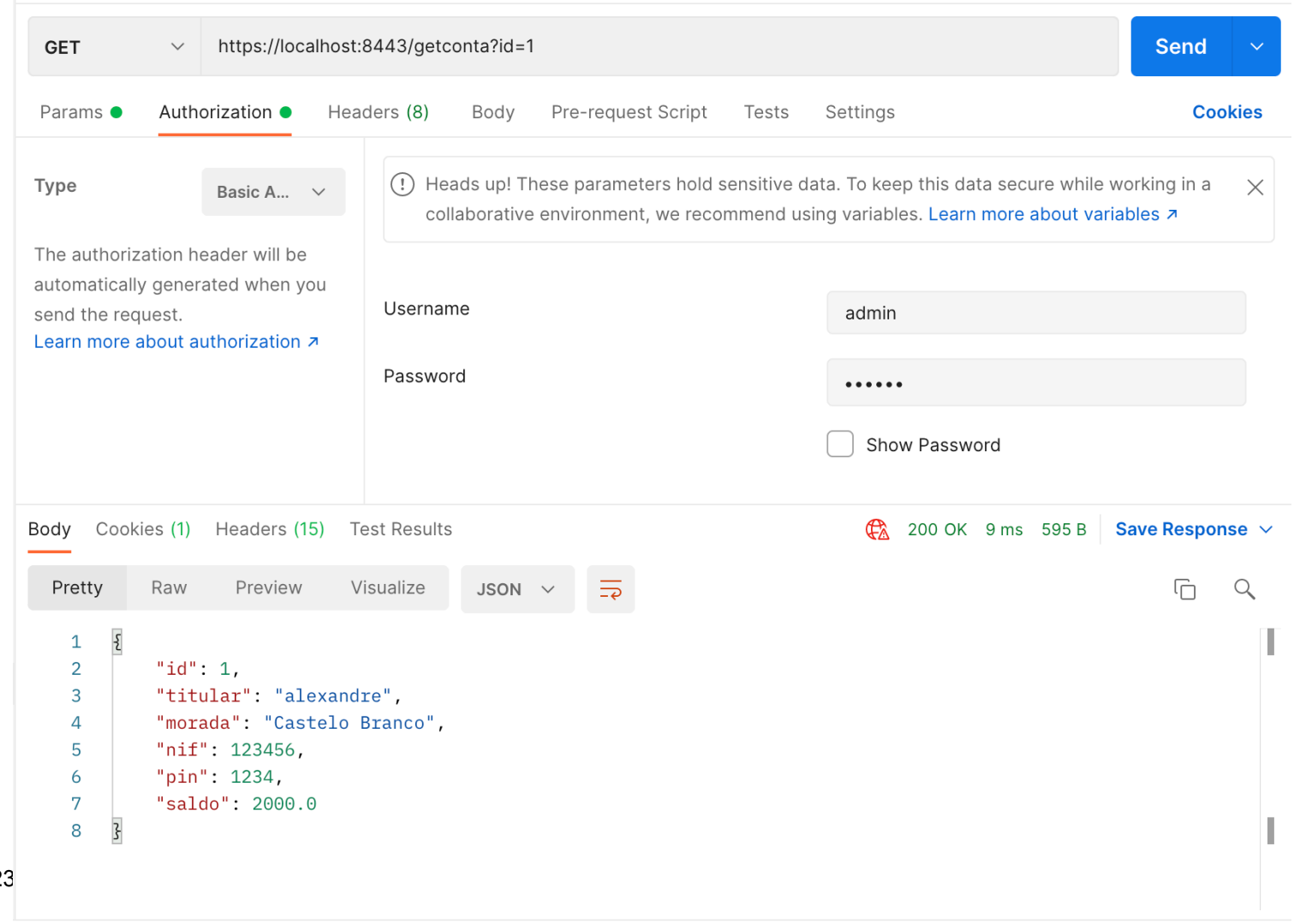
- **Step 3**: Add ssl properties to application.properties

```
server.ssl.key-store=classpath:mycertificatex509.jks
server.ssl.key-store-type=pkcs12
server.ssl.key-store-password=password
server.ssl.key-password=password
server.ssl.key-alias=mycertificatex509
server.port=8443
```

- **Step 4:** Teste your API with Postman
  - But first Postman Disable Certificate Verification

How to Secure a Rest API with HTTPs

- Ilustration:



Questões

- Maecenas aliquam maecenas ligula nostra, accumsan taciti. Sociis mauris in integer
- El eu libero cras interdum at eget habitasse elementum est, ipsum purus pede
- Aliquet sed. Lorem ipsum dol it amet, ligula suspendisse nulla pretium icus