



PRW3 - Programação para a WEB III

Spring Security (parte final)

Conteúdo 13



O que fizemos até agora...



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

ACOMPANHAMENTO DOS PASSOS

- Adicionado o Spring Security no projeto.
- Criada a classe/entidade que vai representar um usuário de nossa API
- Criada a classe que representa o “serviço” de autenticação de usuários
- Criada a classe que define configurações de segurança.
 - util | security | SecurityConfigurations
 - criado método para gerar um objeto (@Bean) responsável pela (de)codificação BCrypt
- Criado usuário na tabela de usuários
 - senha encriptada por BCrypt (hash unidirecional)
- Criada classe controller para permitir o login: AutenticationController (/login)
- Configurar o projeto indicando que a senha está codificada com BCrypt
- **Classe Usuario implementando a interface UserDetails**
 - implementados os métodos da interface



Gerando e devolvendo JWT (Json Web Token)

JWT: ACOMPANHAMENTO DOS PASSOS

Adição de biblioteca no projeto

A página principal mudou... mas ainda tem o libraries!

The screenshot shows a web browser window with the URL `jwt.io` in the address bar. A red arrow points from the address bar to the `jwt.io` text. Another red arrow points from the `Libraries` menu item in the top navigation bar to the `SEE JWT LIBRARIES` button at the bottom of the page.

JSON Web Tokens - jwt.io

Debugger Libraries Introduction Ask

Crafted by Auth0 by Okta

JWT

JSON Web Tokens are an open, industry standard [RFC 7519](#) method for representing claims securely between two parties.

JWT.IO allows you to decode, verify and generate JWT.

LEARN MORE ABOUT JWT SEE JWT LIBRARIES

JSON Web Token Libraries [+](#)

jwt.io/libraries

JWT

Debugger Libraries Introduction Ask

Crafted by  Auth0 by Okta



Libraries for Token Signing/Verification

Filter by All


.NET

<input checked="" type="checkbox"/> Sign	<input checked="" type="checkbox"/> HS256
<input checked="" type="checkbox"/> Verify	<input checked="" type="checkbox"/> HS384
<input checked="" type="checkbox"/> iss check	<input checked="" type="checkbox"/> HS512
<input checked="" type="checkbox"/> sub check	<input checked="" type="checkbox"/> PS256


.NET

<input checked="" type="checkbox"/> Sign	<input checked="" type="checkbox"/> HS256
<input checked="" type="checkbox"/> Verify	<input checked="" type="checkbox"/> HS384
<input checked="" type="checkbox"/> iss check	<input checked="" type="checkbox"/> HS512
<input checked="" type="checkbox"/> sub check	<input checked="" type="checkbox"/> PS256


.NET

<input checked="" type="checkbox"/> Sign	<input checked="" type="checkbox"/> HS256
<input checked="" type="checkbox"/> Verify	<input checked="" type="checkbox"/> HS384
<input checked="" type="checkbox"/> iss check	<input checked="" type="checkbox"/> HS512
<input checked="" type="checkbox"/> sub check	<input checked="" type="checkbox"/> PS256

JSON Web Token Libraries [+](#)

[jwt.io/libraries?language=Java](#)

JWT

Debugger Libraries Introduction Ask

Crafted by  Auth0 by Okta ?



Libraries for Token Signing/Verification

Filter by ▾

Java	Java	Java
<input checked="" type="checkbox"/> Sign	<input checked="" type="checkbox"/> Sign	<input checked="" type="checkbox"/> Sign
<input checked="" type="checkbox"/> Verify	<input checked="" type="checkbox"/> Verify	<input checked="" type="checkbox"/> Verify
<input checked="" type="checkbox"/> iss check	<input checked="" type="checkbox"/> iss check	<input checked="" type="checkbox"/> iss check
<input checked="" type="checkbox"/> sub check	<input checked="" type="checkbox"/> sub check	<input checked="" type="checkbox"/> sub check
	<input checked="" type="checkbox"/> HS256	<input checked="" type="checkbox"/> HS256
	<input checked="" type="checkbox"/> HS384	<input checked="" type="checkbox"/> HS384
	<input checked="" type="checkbox"/> HS512	<input checked="" type="checkbox"/> HS512
	<input checked="" type="checkbox"/> PS256	<input checked="" type="checkbox"/> PS256

Java Java Java

Sign Verify iss check sub check HS256 HS384 HS512 PS256

Sign Verify iss check sub check HS256 HS384 HS512 PS256

Sign Verify iss check sub check HS256 HS384 HS512 PS256

Sign Verify iss check sub check HS256 HS384 HS512 PS256

auth0

The screenshot shows the [JWT.io Libraries page](https://jwt.io/libraries?language=Java) for Java. The page displays a grid of supported JSON Web Token (JWT) claims and their corresponding algorithms. The columns represent different claim types, and the rows represent different algorithms. A green checkmark indicates support, while a red X indicates non-support.

Claim	Algorithm	Claim	Algorithm	Claim	Algorithm
iss	HS512	iss	HS512	iss	HS512
sub	PS256	sub	PS256	sub	PS256
aud	PS384	aud	PS384	aud	PS384
exp	PS512	exp	PS512	exp	PS512
nbf	RS256	nbf	RS256	nbf	RS256
iat	RS384	iat	RS384	iat	RS384
jti	RS512	jti	RS512	jti	RS512
typ	ES256	typ	ES256	typ	ES256
	ES256K		ES256K		ES256K
	ES384		ES384		ES384
	ES512		ES512		ES512
	EdDSA		EdDSA		EdDSA

Below the grid, there are three repository cards:

- Auth0** (5438 stars) - View Repo
- Brian Campbell** - View Repo
- connect2id** - View Repo

At the bottom of each card, there is a Maven dependency line:

- maven: com.auth0 / java-jwt / 3.3.0
- maven: org.bitbucket.b_c / jose4j / 0.9.3
- maven: com.nimbusds / nimbus-jose-jwt / 5.7

auth0

The screenshot shows the [JWT.io Libraries page](https://jwt.io/libraries?language=Java) for Java. The page lists various JSON Web Token checkers and their supported algorithms. A red arrow points from the bottom of the left column to the 'View Repo' link.

Checker	Algorithm
iss check	HS512
sub check	PS256
aud check	PS384
exp check	PS512
nbf check	RS256
iat check	RS384
jti check	RS512
typ check	ES256 ES256K ES384 ES512 EdDSA

Crafted by  Auth0 by Okta

[Debugger](#) [Libraries](#) [Introduction](#) [Ask](#)

[View Repo](#)

maven: com.auth0 / java-jwt / 3.3.0

Brian Campbell [View Repo](#)

connect2id [View Repo](#)

maven: org.bitbucket.b_c / jose4j / 0.9.3

maven: com.nimbusds / nimbus-jose-jwt / 5.7

auth0 / java-jwt Public

Code Issues 6 Pull requests 1 Actions Security Insights

master 11 branches 58 tags Go to file Code

sgc109 Fix typo on a comment in JWTCreator.java (#672) ✓ a6fa0b4 on Sep 12 554 commits

.github	[SDK-4443] Use GitHub Actions for CI (#668)	3 months ago
config/checkstyle	Code formatting for CheckStyle	last year
gradle/wrapper	Update to gradle 6.9.2	last year
lib	Fix typo on a comment in JWTCreator.java (#672)	2 months ago
.codecov.yml	Documentation updates	last year
.gitignore	Update jackson dependency (#523)	last year
.shiprc	Fix .shiprc build.gradle version substitution pattern (...)	last year
CHANGELOG.md	Release 4.4.0 (#658)	8 months ago
EXAMPLES.md	Feature/cleanup (#642)	10 months ago
LICENSE	Feature/cleanup (#642)	10 months ago
MIGRATION_GUIDE.md	updates from PR review	last year
README.md	Remove CircleCI (#670)	3 months ago
build.gradle	remove jcenter	2 years ago

About

Java implementation of JSON Web Token (JWT)

java jwt dx-sdk

Readme MIT license Security policy Activity 5.5k stars 218 watching 940 forks Report repository

Releases 44

4.4.0 Latest on Mar 31 + 43 releases

GitHub - auth0/java-jwt: x +

← → C ⌂ 🔒 github.com/auth0/java-jwt

☰ README.md

Installation ↗

Add the dependency via Maven:

```
<dependency>
  <groupId>com.auth0</groupId>
  <artifactId>java-jwt</artifactId>
  <version>4.5.0</version>
</dependency>
```

Inserir no pom.xml,
como visto anteriormente com outras dependências...

JWT: ACOMPANHAMENTO DOS PASSOS

- Inserida dependência no projeto



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

Criando um “service” que vai trabalhar com os tokens



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

```
src
└── main
    └── java
        └── br.edu.ifsp.pw3.api
            ├── controller
            │   ├── AutenticacaoController.java
            │   ├── HelloController.java
            │   ├── MedicoController.java
            ├── endereco.java
            ├── medico.java
            └── usuario
                ├── AutenticacaoService.java
                ├── dadosAutenticacao.java
                ├── Usuario.java
                └── UsuarioRepository.java
            └── util
                └── security
                    ├── PW3TokenService.java
                    ├── SecurityConfigurations.java
                    ├── TratadorDeErros.java
                    └── ApiApplication.java
    └── resources
```



Nome horrível!! desculpem!



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

```
@Service ←
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;

        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }

}
```

vale
á pena
ver
de novo





O que são Beans no Spring Framework

Beans são objetos gerenciados pelo contêiner Spring IoC (Inversion of Control). Eles são os componentes fundamentais de uma aplicação Spring e possuem características específicas:

Características principais dos Beans:

- 1. Objetos gerenciados:** São instanciados, configurados e gerenciados pelo contêiner Spring, não pelo seu código diretamente.
- 2. Ciclo de vida controlado:** O Spring controla quando um bean é criado, inicializado, utilizado e destruído.
- 3. Escopo definido:** Cada bean tem um escopo específico que determina seu tempo de vida e visibilidade (singleton, prototype, request, etc.).
- 4. Injeção de dependências:** Os beans podem ter suas dependências injetadas automaticamente pelo Spring, facilitando o desacoplamento entre componentes.
- 5. Configuração centralizada:** São definidos de forma centralizada, seja por anotações, XML ou código Java.



O que são Beans no Spring Framework

Beans são objetos gerenciados pelo contêiner Spring IoC (Inversion of Control). Eles são os componentes fundamentais de uma aplicação Spring e possuem características específicas:

Características principais dos Beans:

- 1. Objetos gerenciados:** São instanciados, configurados e gerenciados pelo contêiner Spring, não pelo seu código diretamente.
- 2. Ciclo de vida controlado:** O Spring controla quando um bean é criado, inicializado, utilizado e destruído.
- 3. Escopo definido:** Cada bean tem um escopo de visibilidade (singleton, prototype, request).
- 4. Injeção de dependências:** Os beans podem ser injetados automaticamente pelo Spring, facilitando sua configuração.
- 5. Configuração centralizada:** São definidos no arquivo de configuração XML ou código Java.

Explicação da Anotação @Service no Spring Framework

A anotação `@Service` que você está usando em sua classe `AutenticacaoService` é uma das anotações estereotipadas do Spring Framework, que faz parte do mecanismo de injeção de dependências do Spring.

O que a anotação @Service faz:

- 1. Marca a classe como um componente de serviço:** Indica que a classe pertence à camada de serviço da aplicação, responsável por conter lógica de negócios.
- 2. Habilita detecção automática de componentes:** Quando o Spring escaneia os pacotes da aplicação, ele identifica automaticamente classes marcadas com `@Service` e as registra como beans no contexto da aplicação.
- 3. Permite injeção:** Uma vez registrada como um bean, a classe `AutenticacaoService` pode ser injetada em outras classes usando `@Autowired`, assim como você está injetando o `UsuarioRepository` na sua classe.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

Um JSON Web Token (JWT) é uma string que representa um objeto JSON estruturado de acordo com o padrão definido pelo RFC 7519. Portanto, em termos de tipo de dado, um JWT é uma sequência de caracteres (string) codificada em Base64 que contém informações específicas em um formato JSON.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

O usuário logado

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

```
var algoritmo = Algorithm.HMAC256("12345678");
```

- Essa linha de código está **criando uma instância de um algoritmo de assinatura HMAC** (Hash-based Message Authentication Code) **com a chave secreta "12345678"**. No contexto de JWT (JSON Web Token), o algoritmo HMAC é comumente usado para gerar e verificar assinaturas digitais, garantindo a integridade dos dados contidos no token.
 - Algorithm: É uma classe da biblioteca com.auth0.jwt.algorithms que representa algoritmos de assinatura para JWT.
 - HMAC256: Indica que o algoritmo utilizado é HMAC com SHA-256 (Secure Hash Algorithm 256 bits). SHA-256 é uma função hash criptográfica que produz um valor hash de 256 bits.
 - "12345678": É a chave secreta usada para assinar e verificar o token. **Essa chave deve ser mantida em segredo, pois é utilizada na geração da assinatura e na verificação da integridade do token.**
- Então, **a variável algoritmo** contém **uma instância do algoritmo HMAC SHA-256 configurado com a chave secreta "12345678"**.
- Essa instância será usada posteriormente para assinar o JWT, garantindo que apenas quem possui a chave secreta poderá gerar tokens válidos e verificar sua autenticidade.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

Essa linha de código está usando a biblioteca JWT para criar um token JWT com informações específicas e, em seguida, assiná-lo com o algoritmo HMAC SHA-256 e a chave secreta fornecida.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create() ←
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

        return token;

    } catch (JWTCreationException exception){
        throw new RuntimeException("Erro ao gerar token jwt", exception);
    }
}

private Instant dataExpiracao() {
    return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
}

}
```

- `JWT.create()`: Cria uma instância de construção de token usando a classe `JWT` da biblioteca.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3") ←
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

        return token;

    } catch (JWTCreationException exception){
        throw new RuntimeException("Erro ao gerar token jwt", exception);
    }
}

private Instant dataExpiracao() {
    return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
}
}
```

- `withIssuer("DISCIPLINA PW3")`: Define o emissor do token. Neste caso, o emissor é definido como "DISCIPLINA PW3". O emissor identifica a entidade que emitiu o token.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

- `withSubject(usuario.getLogin())`: Define o assunto do token. No contexto de autenticação, o assunto geralmente representa a identidade para a qual o token foi emitido. Aqui, o assunto é definido como o login do usuário.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;

        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

- `LocalDateTime.now()`: Obtém o objeto `LocalDateTime` representando a data e hora atuais.
- `plusHours(2)`: Adiciona 2 horas ao `LocalDateTime` atual. Isso está configurando a expiração do token para 2 horas após o momento atual.
- `toInstant(ZoneOffset.of("-03:00"))`: Converte o `LocalDateTime` resultante em um objeto `Instant` utilizando o deslocamento de fuso horário de -03:00. A classe `Instant` representa um ponto específico na linha do tempo, sem considerar fuso horário.

Portanto, o método `dataExpiracao()` está retornando um `Instant` que representa o momento exato em que o token JWT deve expirar, neste caso, 2 horas após a geração do token. Isso é útil para garantir que o token seja válido apenas por um período limitado de tempo, melhorando a segurança do sistema.

```
private Instant dataExpiracao() {  
    return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));  
}
```

}



Método que retorna um objeto Instant, que é a hora atual + 2 horas.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

- `.withExpiresAt(dataExpiracao())`: Define a data de expiração do token. A função `dataExpiracao()` retorna um objeto `Instant` representando o momento em que o token expirará. No exemplo, o token expirará duas horas após o momento atual.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256("12345678");
            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);
            return token;
        } catch (JWTCreationException exception) {
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

- `sign(algoritmo)` : Realiza a assinatura digital do token usando o algoritmo HMAC SHA-256 e a chave secreta fornecida. Essa é a etapa final na criação do token, garantindo sua integridade.

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;
        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }

    private Instant dataExpiracao() {
        return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

JWT: ACOMPANHAMENTO DOS PASSOS

- Inserida dependência no projeto
- **Criada classe para gerenciar tokens: PW3TokenService (Bean)**
 - método gerarToken()

Voltando e alterando o controller
que toma conta do login,
para gerar o token e colocar
na resposta da requisição ...

```
@RestController
@RequestMapping("/login")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager manager; // Objeto AuthenticationManager,
                                         // que será injetado aqui.
                                         // LEMBRAR COMENTÁRIOS UMA AULA PASSADA!!!

    @Autowired
    private PW3TokenService tokenService;

    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {

        var token = new UsernamePasswordAuthenticationToken( dados.login(), dados.senha() );

        var authentication = manager.authenticate(token);

        return ResponseEntity
            .ok( tokenService.gerarToken( (Usuario) authentication.getPrincipal() ) );
    }
}
```

```
@RestController
@RequestMapping("/login")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager manager; // Objeto AuthenticationManager,
                                            // que será injetado aqui.

    @Autowired
    private PW3TokenService tokenService;

    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {

        var token = new UsernamePasswordAuthenticationToken( dados.login(), dados.senha() );

        var authentication = manager.authenticate(token);

        return ResponseEntity
            .ok( tokenService.gerarToken( (Usuario) authentication.getPrincipal() ) );
    }
}
```

```
@RestController
@RequestMapping("/login")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager manager; // Objeto AuthenticationManager,
                                            // que será injetado aqui.

    @Autowired
    private PW3TokenService tokenService; // IDEM!!!!

    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {

        var token = new UsernamePasswordAuthenticationToken( dados.login(), dados.senha() );

        var authentication = manager.authenticate(token);

        return ResponseEntity
            .ok( tokenService.gerarToken( (Usuario) authentication.getPrincipal() ) );
    }
}
```

Injetado aqui um objeto da classe PW3TokenService
que acabamos de criar

```
@RestController
@RequestMapping("/login")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager manager; // Objeto AuthenticationManager,
                                            // que será injetado aqui.

    @Autowired
    private PW3TokenService tokenService;

    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {

        var token = new UsernamePasswordAuthenticationToken( dados.login(), dados.senha() );
        var authentication = manager.authenticate(token);

        return ResponseEntity
            .ok( tokenService.gerarToken( (Usuario) authentication.getPrincipal() ) );
    }
}
```

Lembrando... este token não é o que estamos
trabalhando agora!!
Feito aula passada!

```
@RestController
@RequestMapping("/login")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager manager; // Objeto AuthenticationManager,
                                         // que será injetado aqui.

    @Autowired
    private PW3TokenService tokenService;

    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {

        var token = new UsernamePasswordAuthenticationToken( dados.login(), dados.senha() );

        var authentication = manager.authenticate(token);

        return ResponseEntity
            .ok( tokenService.gerarToken( (Usuario) authentication.getPrincipal() ) );
    }
}
```



Usando o método `gerarToken()` para gerar o token,
e devolvendo como corpo do `.ok()`
da `ResponseEntity`.

```

@RestController
@RequestMapping("/login")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager manager; // Objeto AuthenticationManager,
                                            // que será injetado aqui.

    @Autowired
    private PW3TokenService tokenService;

    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {

        var token = new UsernamePasswordAuthenticationToken( dados.login(), dados.senha() );

        var authentication = manager.authenticate(token);

        return ResponseEntity
            .ok( tokenService.gerarToken( (Usuario) authentication.getPrincipal() ) );
    }
}

```

O objetivo dessa linha é obter o objeto '**Usuario**' associado à autenticação bem-sucedida para que ele possa ser passado como argumento para o método '**gerarToken**' do serviço '**PW3TokenService**'. Isso implica que a implementação de '**Authentication**' utilizada está configurada para armazenar um objeto '**Usuario**' como principal durante o processo de autenticação.

Testando...

POST



localhost:8080/login

Send



Params

Auth

Headers (9)

Body



Pre-req.

Tests

Settings

Cookies

raw

JSON



Beautify

```
1 {  
2   "login": "asdrubal@gmail.com",  
3   "senha": "123456"  
4 }  
5
```

Body



200 OK

719 ms

507 B



Save as example



Pretty

Raw

Preview

Visualize

Text



1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJpc3Mi0iJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIIsImV4cCI6MTY50Tc2
NjEwMX0.UIAaMgKT7vUDidZYpj-
Gv4_obnXKaH52p6IGCHsekC

Curiosidade...

Leve o token gerado
para o site jwt.io

OBS: a tela mudou um pouco,
mas o efeito é o mesmo.

Algorithm

HS256

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3Mi0iJESVNDVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIIsImV4cCI6MTc0NzQwMzQzMH0.m291SSp9RTK9SwSR5StvDQkkYahJYvpyqn17uy5ba0U
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "iss": "DISCIPLINA PW3",  
  "sub": "asdrubal@gmail.com",  
  "exp": 1747403430  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload)  
  your-256-bit-secret  
)  secret base64 encoded
```

✖ Invalid Signature

SHARE JWT

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3Mi0iJESVNDSVMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIssImV4cCI6MTY5OTgzMDQ2OH0.qgrm3TL-NzK12LcUDTjr7itm3SPRuE7JI0hvc9kAHk
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "iss": "DISCIPLINA PW3",  
  "sub": "asdrubal@gmail.com",  
  "exp": 1699830468  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  12345678  
)  secret base64 encoded
```

Signature Verified

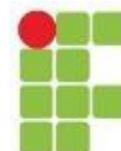
SHARE JWT

OBS: Colocar a senha primeiro,
DEPOIS colar o token aqui em cima!

```
try {  
  var algoritmo = Algorithm.HMAC256("12345678");
```

JWT: ACOMPANHAMENTO DOS PASSOS

- Inserida dependência no projeto
- Criada classe para gerenciar tokens: PW3TokenService
 - método gerarToken()
- **Alterado o AutenticacaoController**
 - alterado método efetuarLogin(), para gerar e devolver o JWT no corpo da resposta



Estamos devolvendo o token diretamente
(como uma string) na resposta da requisição.

Seguindo boas práticas,
vamos encapsular o token
em um DTO.



A recomendação de encapsular o token JWT em um DTO (Data Transfer Object) em vez de retorná-lo diretamente como uma string tem a ver com boas práticas de design e segurança.

1. Segurança:

- Ao encapsular o token em um DTO, você tem mais controle sobre como o token é manipulado antes de ser enviado como resposta. Isso pode ser útil se você precisar adicionar informações adicionais ao objeto de resposta, como data de expiração, escopo, ou qualquer outra informação relevante.

2. Evita exposição direta:

- Retornar o token diretamente como uma string pode expor mais informações do que o necessário. Um DTO pode ser configurado para incluir apenas as informações essenciais, reduzindo a exposição de detalhes sensíveis.

3. Facilita futuras alterações:

- Se, no futuro, você precisar fazer alterações na estrutura da resposta (por exemplo, adicionar campos adicionais), um DTO oferece uma camada de abstração que facilita essas modificações sem impactar diretamente os consumidores da sua API.

4. Padrão de Boas Práticas:

- Utilizar DTOs para transportar informações entre camadas é uma prática comum em arquiteturas de software. Isso ajuda na manutenção, legibilidade e na consistência do código.

DTO: DadosTokenJWT

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "main". It includes a "java" folder containing a "br.edu.ifsp.pw3.api" package with "controller", "endereco", "medico", "usuario", and "util" subfolders. The "util" folder contains a "security" subfolder which is expanded, showing three files: "DadosTokenJWT", "PW3TokenService", and "SecurityConfigurations". Two red arrows point from the bottom of the slide towards this "security" folder.
- Code Editor:** The right pane displays the code for "DadosTokenJWT.java". The code is:

```
package br.edu.ifsp.pw3.api.util.security;  
  
public record DadosTokenJWT(String token) {  
}  
|
```

A red rectangular box highlights the entire code block from line 3 to line 6.

```
public record DadosTokenJWT(String token) {  
}
```

JWT: ACOMPANHAMENTO DOS PASSOS

- Inserida dependência no projeto
- Criada classe para gerenciar tokens: PW3TokenService
 - método gerarToken()
- Alterado o AutenticacaoController
 - alterado método efetuarLogin(), para gerar e devolver o JWT no corpo da resposta
- **Criado DTO para devolver o token JWT (DadosTokenJWT)**



Alterando o controller
para usar o DTO
para devolver o token



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

Classe AutenticacaoController.java

```
@PostMapping  
public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {  
  
    var token = new UsernamePasswordAuthenticationToken( dados.login(),  
                                                        dados.senha() );  
  
    var authentication = manager.authenticate(token);  
  
    // Criando o token JWT:  
    var tokenJWT = tokenService.gerarToken( (Usuario) authentication.getPrincipal() );  
  
    // Criando o DTO DadosTokenJWT a partir do token criado acima,  
    // e devolvendo no corpo da respostas da requisição:  
    return ResponseEntity.ok( new DadosTokenJWT(tokenJWT) );  
}
```



Classe AutenticacaoController.java

```
@PostMapping  
public ResponseEntity efetuarLogin(@RequestBody @Valid dadosAutenticacao dados) {  
  
    var token = new UsernamePasswordAuthenticationToken( dados.login(),  
                                                        dados.senha() );  
  
    var authentication = manager.authenticate(token);  
  
    // Criando o token JWT:  
    var tokenJWT = tokenService.gerarToken( (Usuario) authentication.getPrincipal() );  
  
    // Criando o DTO DadosTokenJWT a partir do token criado acima,  
    // e devolvendo no corpo da respostas da requisição:  
    return ResponseEntity.ok( new DadosTokenJWT(tokenJWT) );  
}
```



Testando...



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

POST

localhost:8080/login

Send

Params Auth Headers (9) Body ● Pre-req. Tests Settings

Cookies

raw ▼

JSON ▼

Beautify

```
1 {  
2   ... "login" : "asdrubal@gmail.com",  
3   ... "senha" : "123456"  
4 }  
5
```

“antes”

Body ▼

200 OK 719 ms 507 B

Save as example

Pretty

Raw

Preview

Visualize

Text ▼



Send

```
1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIisImV4cCI6MTY50Tc2  
NjEwMX0.UIAaMgKT7vUDidZYpj-Gv4_obnXKaH52p6IGCHsek
```

String pura

```
1 {  
2   ... "login" : "asdrubal@gmail.com",  
3   ... "senha" : "123456"  
4 }  
5
```

Body ▼

200 OK 701 ms 518 B

Save as example ...

Pretty

Raw

Preview

Visualize

JSON ▼



json

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIisImV4cCI6MTY50Tg  
zMjUxNX0.-wsHA0vWB352W8uLT8ayXceD9IeN0vP5FSCADdPqMMw"  
3 }
```

POST

localhost:8080/login

Send

Params Auth Headers (9) Body Pre-req. Tests Settings

raw JSON

```
1 {  
2   ... "login" ...: "asdrubal@gmail.com",  
3   ... "senha" ...: "123456"  
4 }  
5
```

“antes”

Body

200 OK 719 ms 507 B

Save as example

Pretty Raw Preview Visualize Text  

```
1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIIsImV4cCI6MTY50Tc2  
NjEwMX0.UIAaMgKT7vUDidZYpj-Gv4_ohnXKaH52p6IGCHsekC
```



Beautify

```
1 {  
2   ... "login" ...: "asdrubal@gmail.com",  
3   ... "senha" ...: "123456"  
4 }  
5
```

String pura

json

Body 200 OK 701 ms 518 B Save as example

Pretty Raw Preview Visualize JSON  

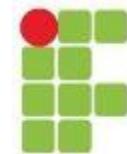
```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIIsImV4cCI6MTY50Tg  
zMjUxNX0.-wsHA0vWB352W8uLT8ayXceD9IeN0vP5FSCADdPqMMw"  
3 }
```

JWT: ACOMPANHAMENTO DOS PASSOS

- Inserida dependência no projeto
- Criada classe para gerenciar tokens: PW3TokenService
 - método gerarToken()
- Alterado o AutenticacaoController
 - alterado método efetuarLogin(), para gerar e devolver o JWT no corpo da resposta
- Criado DTO para devolver o token JWT (DadosTokenJWT)
- **Alterado o AutenticacaoController**
 - alterado método efetuarLogin(), para gerar e devolver o JWT no corpo da resposta, como DTO



Uma pequena pausa,
para falarmos de senhas
como strings “hardcoded” diretamente
no código fonte...



Criei um novo projeto Spring no IntelliJ,
com Spring WEB.

O projeto não tem nada,
apenas o código principal...

Coloquei nele uma variável String,
atribuindo um certo conteúdo nela:



c ExemploSegurancaApplication.java ×

```
1 package com.example.exemploSeguranca;  
2  
3 +import ...  
4  
5  
6 @SpringBootApplication ←  
7 ➤ public class ExemploSegurancaApplication {  
8  
9     String teste = "ASDRUBAL"; ←  
10  
11    ➤     public static void main(String[] args) {  
12        SpringApplication.run(ExemploSegurancaApplication.class, args);  
13    }  
14  
15 }
```



Vamos gerar um arquivo JAR
desse projeto...



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

1.java ×

```
e.exemploSeguranca;
```

tion

```
loSegurancaApplication {
```

```
 "ASDRUBAL";
```

```
void main(String[] args) {
```

```
ication.run(ExemploSegurancaApplication.class, args);
```

Maven



> Profiles
> exemploSeguranca
> Lifecycle

- ⚙ clean
- ⚙ validate
- ⚙ compile
- ⚙ test
- ⚙ package
- ⚙ verify
- ⚙ install
- ⚙ site
- ⚙ deploy

> Plugins
> Dependencies



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ exemploSeguranca ---
[INFO] Building jar: /home/carlao2005/HD_PRINCIPAL/Dropbox/intellijProjects/exemploSeguranca/target/exempl
[INFO]
[INFO] --- spring-boot:3.1.5:repackage (repackage) @ exemploSeguranca ---
[INFO] Replacing main artifact /home/carlao2005/HD_PRINCIPAL/Dropbox/intellijProjects/exemploSeguranca/tar
[INFO] The original artifact has been renamed to /home/carlao2005/HD_PRINCIPAL/Dropbox/intellijProjects/e
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.444 s
[INFO] Finished at: 2023-11-12T19:31:55-03:00
[INFO] -----
```

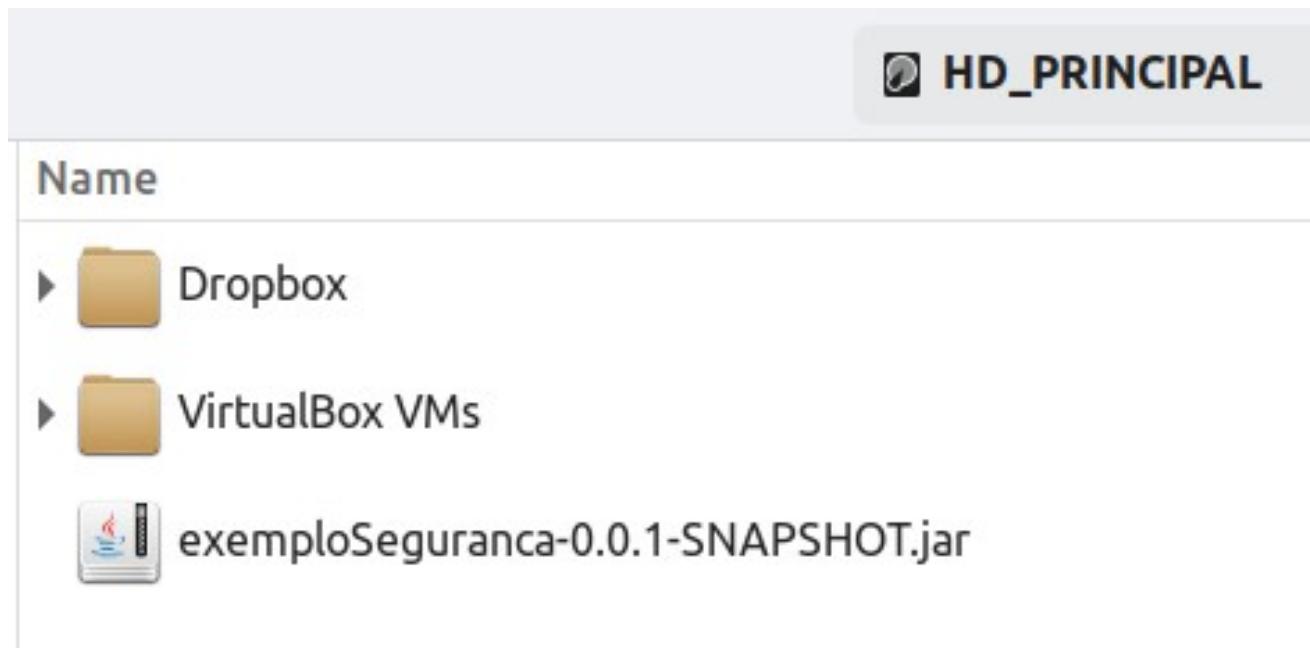
Process finished with exit code 0

Building jar:

/home/carlao2005/HD_PRINCIPAL/Dropbox
/intellijProjects/exemploSeguranca/target/exemploSeguranca-0.0.1-SNAPSHOT.jar



Copiei o jar para outro local...



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

Dá até pra executar...

Name

- ▶ Dropbox
- ▶ VirtualBox VMs
- exemploSeguranca-0.0.1-SNAPSHOT.jar

```
carlao2005@celtasnote:~/HD_PRINCIPAL$ /usr/lib/jvm/jdk-20/bin/java -jar exemploSeguranca-0.0.1-SNAPSHOT.jar
```

```
.
  _----_
 / \ / _ _ ' - _ _ - _ ( _ ) - _ _ _ _ _ \ \ \ \ \
( ( ) \ _ _ | ' _ | ' _ | ' _ \ / _ ' | \ \ \ \
\ \ / _ _ ) | [ _ ) | | | | | | ( _ | | ) ) ) )
' | _ _ _ | . _ | _ | _ | _ \ _ , | / / / /
=====| _ | ======| _ _ / = / _ / _ /
:: Spring Boot ::                               (v3.1.5)

2023-11-12T19:40:36.785-03:00  INFO 12251 --- [           main] c.e.e.ExemploSegurancaApplication      : Starting ExemploSegurancaApplication v0.0.1-SNAPSHOT using Java 20.0.1 with PID 12251 (/home/carlao2005/HD_PRINCIPAL/exemploSeguranca-0.0.1-SNAPSHOT.jar started by carlao2005 in /home/carlao2005/HD_PRINCIPAL)
2023-11-12T19:40:36.788-03:00  INFO 12251 --- [           main] c.e.e.ExemploSegurancaApplication      : No active profile set, falling back to 1 default profile: "default"
2023-11-12T19:40:37.915-03:00  INFO 12251 --- [           main] o.s.b.w.
```

Dá até pra executar...

The screenshot shows a terminal window and a web browser window side-by-side.

Terminal Window:

```
carlao2005@celtasnote:~/HD_PRINCIPAL$ /usr/lib/jvm/jdk-20/bin/java -jar exemploSeguranca-0.0.1-SNAPSHOT.jar
```

The terminal displays a Spring Boot logo followed by the text "Spring Boot ::". Below it, there is some log output:

```
2023-11-12T19:40:36.722242Z | DEBUG | main | o.s.b.w.e.WhitelabelErrorController - This application has no explicit mapping for /error, so you are seeing this as a fallback.  
2023-11-12T19:40:36.722242Z | DEBUG | main | o.s.b.w.e.WhitelabelErrorController - There was an unexpected error (type=Not Found, status=404).
```

Browser Window:

The browser shows a "Whitelabel Error Page" with the URL "localhost:8080". The page content is:

Whitelabel Error Page

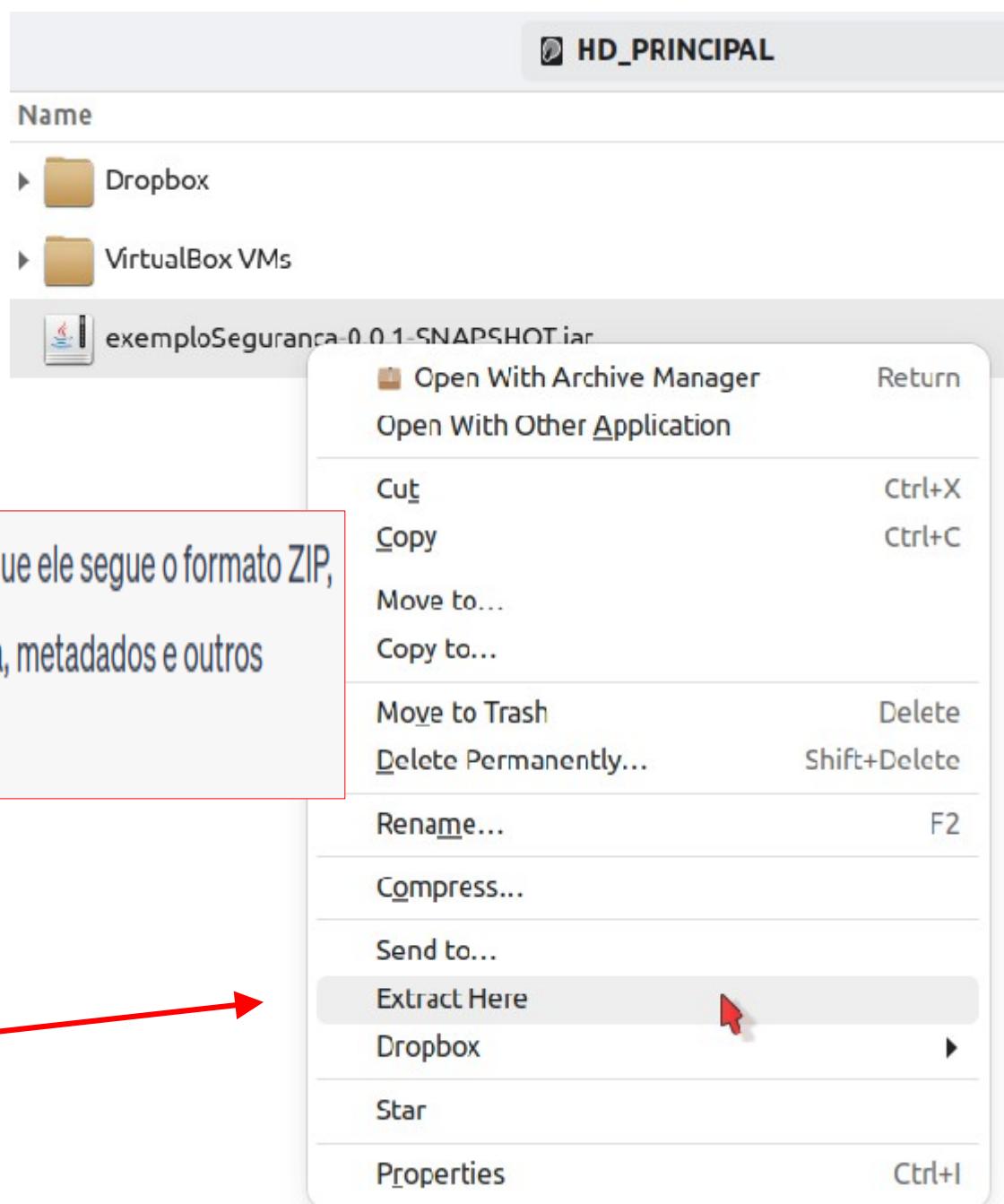
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Nov 12 19:41:38 BRT 2023

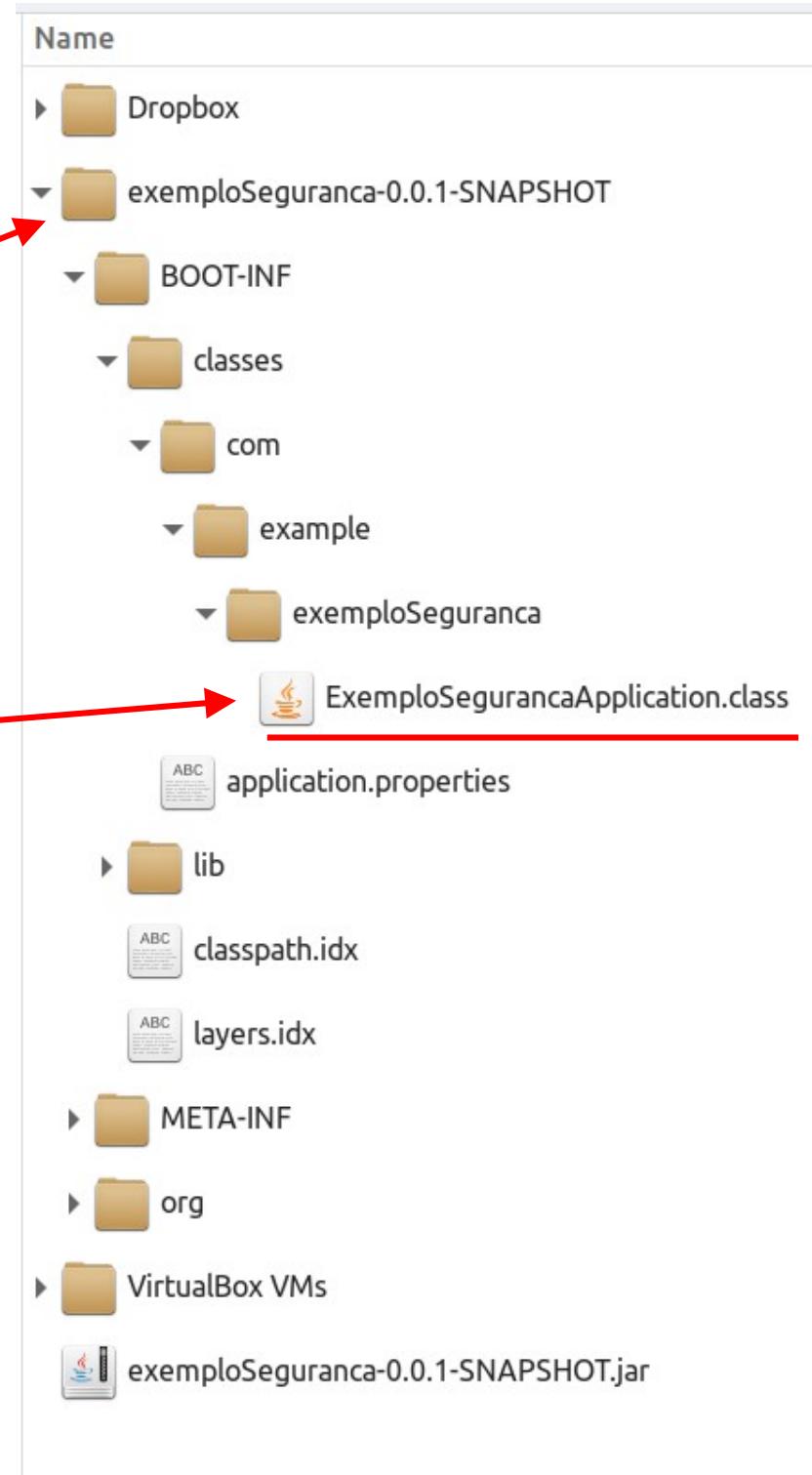
There was an unexpected error (type=Not Found, status=404).

Descompactei...

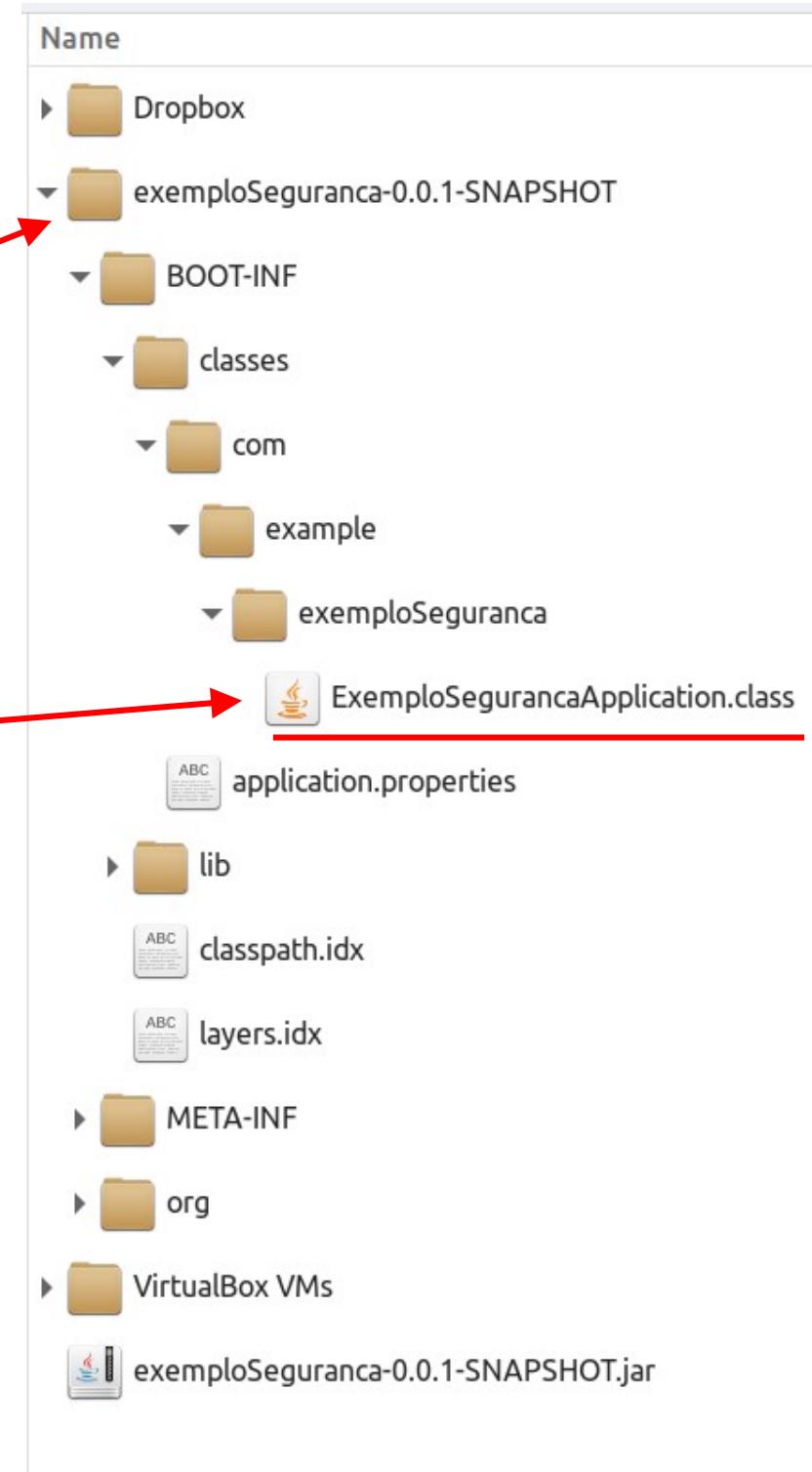
Internamente, um arquivo JAR é um arquivo ZIP, o que significa que ele segue o formato ZIP, mas com uma estrutura específica para armazenar arquivos Java, metadados e outros recursos relacionados ao Java.



Descompactei...



Descompactei...



Um arquivo `.class` contém:

1. Um cabeçalho com informações da versão da JVM.
2. Um constant pool com literais e referências.
3. Informações sobre a classe e sua hierarquia.
4. Definições dos campos (atributos).
5. Definições dos métodos e seus bytecodes.
6. Tabelas auxiliares e metadados opcionais.

Não dá pra abrir como texto puro... é formato "binário":

Vamos abrir com um editor binário...

Usei on-line:
<https://hexed.it/>

File Edit View Tools Help

Open file Save as Undo Redo Tools Settings Help

Information		-Untitled-	ExemploSegurancaAp...	
Format (Little-endian)		00000000	CA FE BA BE 00 00 00 00 3D 00 23 0A 00 02 00 03 07=.#.....
		00000010	00 04 0C 00 05 00 06 01 00 10 6A 61 76 61 2F 6Cjava/l
Assigned (+)	Signed (±)	00000020	61 6E 67 2F 4F 62 6A 65 63 74 01 00 06 3C 69 6E	ang/Object...<in
		00000030	69 74 3E 01 00 03 28 29 56 08 00 08 01 00 08 41	it>...()V.....A
26	-54	00000040	53 44 52 55 42 41 4C 09 00 0A 00 0B 07 00 0C 0C	SDRUBAL.....
	-310	00000050	00 0D 00 0E 01 00 38 63 6F 6D 2F 65 78 61 6D 708com/examp
54922	-4522294	00000060	6C 65 2F 65 78 65 6D 70 6C 6F 53 65 67 75 72 61	le/exemploSegura
9925962	-1095041334	00000070	6E 63 61 2F 45 78 65 6D 70 6C 6F 53 65 67 75 72	nca/ExemploSegur
15513239513530058		00000080	61 6E 63 61 41 70 70 6C 69 63 61 74 69 6F 6E 01	ancaApplication.
15513239513530058		00000090	00 05 74 65 73 74 65 01 00 12 4C 6A 61 76 61 2F	..teste...Ljava/
		000000A0	6C 61 6E 67 2F 53 74 72 69 6E 67 3B 0A 00 10 00	lang/String;....
valid number		000000B0	11 07 00 12 0C 00 13 00 14 01 00 2A 6F 72 67 2F*org/
16522514		000000C0	73 70 72 69 6E 67 66 72 61 6D 65 77 6F 72 6B 2F	springframework/
054324061939839e-15		000000D0	62 6F 6F 74 2F 53 70 72 69 6E 67 41 70 70 6C 69	boot/SpringAppli
1989898		000000E0	63 61 74 69 6F 6E 01 00 03 72 75 6E 01 00 62 28	cation...run..b(
1989898		000000F0	4C 6A 61 76 61 2F 6C 61 6E 67 2F 43 6C 61 73 73	Ljava/lang/Class
valid date		00000100	3B 5B 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72	;[Ljava/lang/Str
19-12-30 00:00:00.000 UTC		00000110	69 6E 67 3B 29 4C 6F 72 67 2F 73 70 72 69 6E 67	ing;)Lorg/spring
11-05-27 04:19:22 UTC		00000120	66 72 61 6D 65 77 6F 72 6B 2F 63 6F 6E 74 65 78	framework/context
11-05-26 01:19:22 Local		00000130	74 2F 43 6F 6E 66 69 67 75 72 61 62 6C 65 41 70	t/ConfigurableAp
11-05-26 04:19:22 UTC		00000140	70 6C 69 63 61 74 69 6F 6E 43 6F 6E 74 65 78 74	plicationContext
valid data		00000150	3B 01 00 04 43 6F 64 65 01 00 0F 4C 69 6E 65 4E	;...Code...LineN
		00000160	75 6D 62 65 72 54 61 62 6C 65 01 00 12 4C 6F 63	umberTable...Loc
		00000170	61 6C 56 61 72 69 61 62 6C 65 54 61 62 6C 65 01	alVariableTable.
		00000180	00 04 74 68 69 73 01 00 3A 4C 63 6F 6D 2F 65 78	..this..:Lcom/ex
		00000190	61 6D 70 6C 65 2F 65 78 65 6D 70 6C 6F 53 65 67	ample/exemploSeg
		000001A0	75 72 61 6E 63 61 2F 45 78 65 6D 70 6C 6F 53 65	uranca/ExemploSe

File Edit View Insert Tools Help

Open file Save as Undo Redo Tools Settings Help

Information

Format (Little-endian)

Unsigned (+) Signed (±)

54 -54
26 -310
54922 -4522294
9925962 -1095041334
15513239513530058
15513239513530058
Valid number
6522514
054324061939839e-15
1989898
1989898
Valid date
19-12-30 00:00:00.000 UTC
11-05-27 04:19:22 UTC
15-05-26 01:19:22 Local
15-05-26 04:19:22 UTC
Valid data

-Untitled- ExemploSegurancaAp...

CA FE BA BE 00 00 00 3D 00 23 0A 00 02 00 03 07 40 04 0C 00 05 00 06 01 00 10 6A 61 76 61 2F 6C 3 3C 69 6E L 00 08 41 7 00 0C 0C 3 61 6D 70 7 75 72 61 5 67 75 72 9 6F 6E 01 1 76 61 2F 4 00 10 00 5 72 67 2F 7 72 6B 2F 9 70 6C 69 1 00 62 28 2 61 73 73 5 53 74 72 2 69 6E 67 3 74 65 78 2 65 41 70 4 65 78 74 3 6E 65 4E 2 4C 6F 63 2 6C 65 01 1 2F 65 78 5 53 65 67 2 6F 53 65

.....#.....
.....java/l
ang/Object...<in
it>...()V.....A
SDRUBAL.....
8com/examp
le/exemploSegura
nca/ExemploSegur
ancaApplication.
.teste...Ljava/
lang/String;....
*org/
springframework/
boot/SpringAppli
cation...run..b(
Ljava/lang/Class
;[Ljava/lang/Str
ing;)Lorg/spring
framework/context
t/ConfigurableAp
plicationContext
;...Code...LineN
umberTable...Loc
alVariableTable.
..this..:Lcom/ex
ample/exemploSeg
uranca/ExemploSe

VIXE!!!

Conclusão:
É uma **falha de segurança**
colocar senhas 'hard coded' em texto puro
no código fonte.

E como fica isso ????

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("12345678");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;

        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }
}
```

E como fica isso ????

```
@Service
public class PW3TokenService {

    public String gerarToken(Usuario usuario) {

        try {
            var algoritmo = Algorithm.HMAC256("1234567890");

            String token = JWT.create()
                .withIssuer("DISCIPLINA PW3")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);

            return token;

        } catch (JWTCreationException exception){
            throw new RuntimeException("Erro ao gerar token jwt", exception);
        }
    }
}
```



Solução

- Criar a senha como uma variável de ambiente
 - variável no sistema operacional
- Configurar o arquivo **applications.properties** para ler esse valor
 - se não existir, definir uma padrão (para poder rodar localmente)
- Utilizar o valor lido no código-fonte



Arquivo application.properties

application.properties x

```
# Configuração do banco de dados H2
spring.datasource.url=jdbc:h2:file:/home/carla02005/HD_PRIN
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

server.error.include-stacktrace=never

pw3.senha.principal.geracao.token=${JWT_SECRET:12345678}
```

pw3.senha.principal.geracao.token=\${JWT_SECRET:12345678}



```
pw3.senha.principal.geracao.token=${JWT_SECRET:12345678}
```

1. `pw3.senha.principal.geracao.token`: Este é o nome da propriedade. Ela é específica para o seu projeto e pode ser usada em outras partes do código para acessar o valor associado a ela. Neste caso, parece estar relacionada à geração de tokens, talvez para autenticação ou autorização no seu aplicativo.
2. `\${JWT_SECRET:12345678}`:
 - O `\${...}` é uma notação de placeholder utilizada em arquivos de propriedades do Spring.
 - `JWT_SECRET` é uma referência a uma variável de ambiente chamada `JWT_SECRET`.
 - `12345678` é um valor padrão que será usado se a variável de ambiente `JWT_SECRET` não estiver definida.

Em resumo, essa configuração está tentando obter um valor para a propriedade

`pw3.senha.principal.geracao.token` da seguinte maneira:

- Se a variável de ambiente `JWT_SECRET` estiver definida, ela usará o valor dessa variável.
- Se a variável de ambiente não estiver definida, ela usará o valor padrão `12345678`.

Isso é útil porque permite configurar dinamicamente a senha de geração de token do seu aplicativo por meio de uma variável de ambiente, proporcionando flexibilidade e segurança.

Por exemplo, em ambientes de produção, você pode definir a variável de ambiente `JWT_SECRET` com um valor mais seguro e exclusivo.

Precisamos usar este valor
no código de geração do JWT,
como a senha principal:

Método gerarToken da classe PW3TokenService

```
@Service
```

```
public class PW3TokenService {
```

```
    @Value("${pw3.senha.principal.geracao.token}")  
    private String secret;
```

```
    @Value("${pw3.senha.principal.geracao.token}")  
    private String secret;
```

```
    public String gerarToken(Usuario usuario) {  
  
        System.out.println(secret);  
  
        try {  
            var algoritmo = Algorithm.HMAC256(secret);  
        }  
    }
```

pw3.senha.principal.geracao.token=\${JWT_SECRET:12345678}

Método gerarToken da classe PW3TokenService

```
@Service  
public class PW3TokenService {  
  
    @Value("${pw3.senha.principal.geracao.token}")  
    private String secret;  
  
    public String gerarToken(Usuario usuario) {  
  
        System.out.println(secret);  
  
        try {  
            var algoritmo = Algorithm.HMAC256(secret);  
        }  
    }  
}
```

CUIDADO COM O IMPORT DO @Value !!!!

import org.springframework.beans.factory.annotation.Value;

NÃO O DO LOMBOK!!!!

Método gerarToken da classe PW3TokenService

```
@Service  
public class PW3TokenService {  
  
    @Value("${pw3.senha.principal.geracao.token}")  
    private String secret;  
  
    public String gerarToken(Usuario usuario) {  
  
        System.out.println(secret);  
  
        try {  
            var algoritmo = Algorithm.HMAC256(secret);  
        }  
    }  
}
```

Método gerarToken da classe PW3TokenService

```
@Service  
public class PW3TokenService {  
  
    @Value("${pw3.senha.principal.geracao.token}")  
    private String secret;  
  
    public String gerarToken(Usuario usuario) {  
  
        System.out.println(secret);  
        try {  
            var algoritmo = Algorithm.HMAC256(secret);  
        } catch (Exception e) {  
            throw new RuntimeException("Erro ao gerar token", e);  
        }  
    }  
}
```

Só pra gente ver qual valor está usando...

Testando...

The screenshot shows a POST request in Postman to the endpoint `localhost:8080/login`. The request body is defined in JSON format:

```
1 {  
2   "login": "asdrubal@gmail.com",  
3   "senha": "123456"  
4 }  
5
```

The response status is `200 OK` with a response time of `795 ms` and a size of `518 B`. The response body is:

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpc3Mi0iJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLm  
NvbSIsImV4cCI6MTY50TgzODQ4MX0.  
zJdgyoQq_cP8Ab2hDhJNhRoMKdk5VEnKMAe6xviuVE0"  
3 }
```

resources
db.migration

Run: ApiApplication

Structure Bookmarks

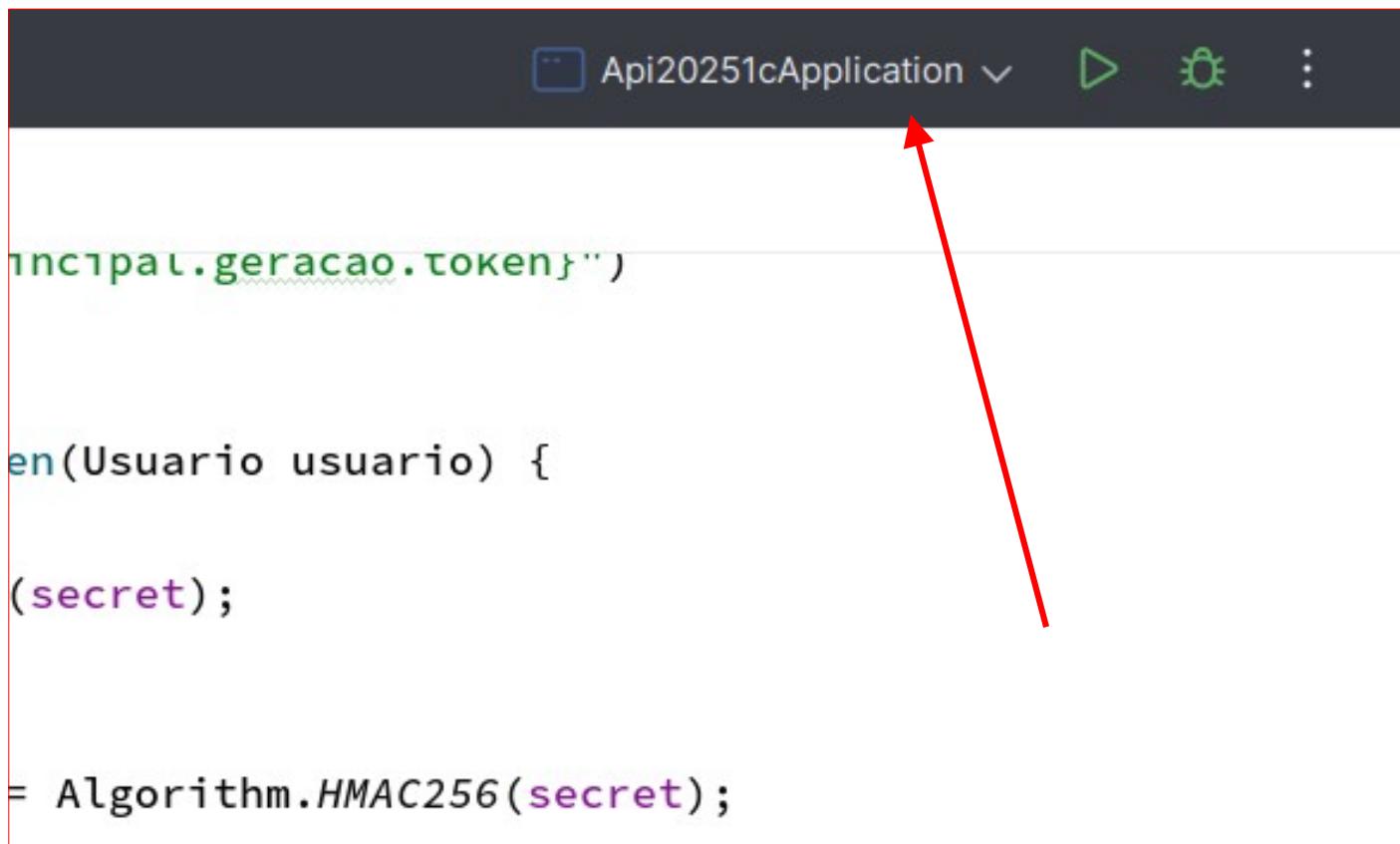
```
u1_0.id,  
u1_0.login,  
u1_0.senha  
from  
usuarios u1_0  
where  
u1_0.login=?  
12345678
```

Version Control Run TODO Problems Terminal Services

All files are up-to-date (a minute ago)

Criando (simulando) uma variável de ambiente...

"edit configurations"



```
    principal.geracao.token}"))

    en(Usuario usuario) {

        (secret);

        = Algorithm.HMAC256(secret);
    }
}
```

A screenshot of a Java IDE interface, specifically showing a code editor. The code displayed is:

```
    principal.geracao.token}"))

    en(Usuario usuario) {

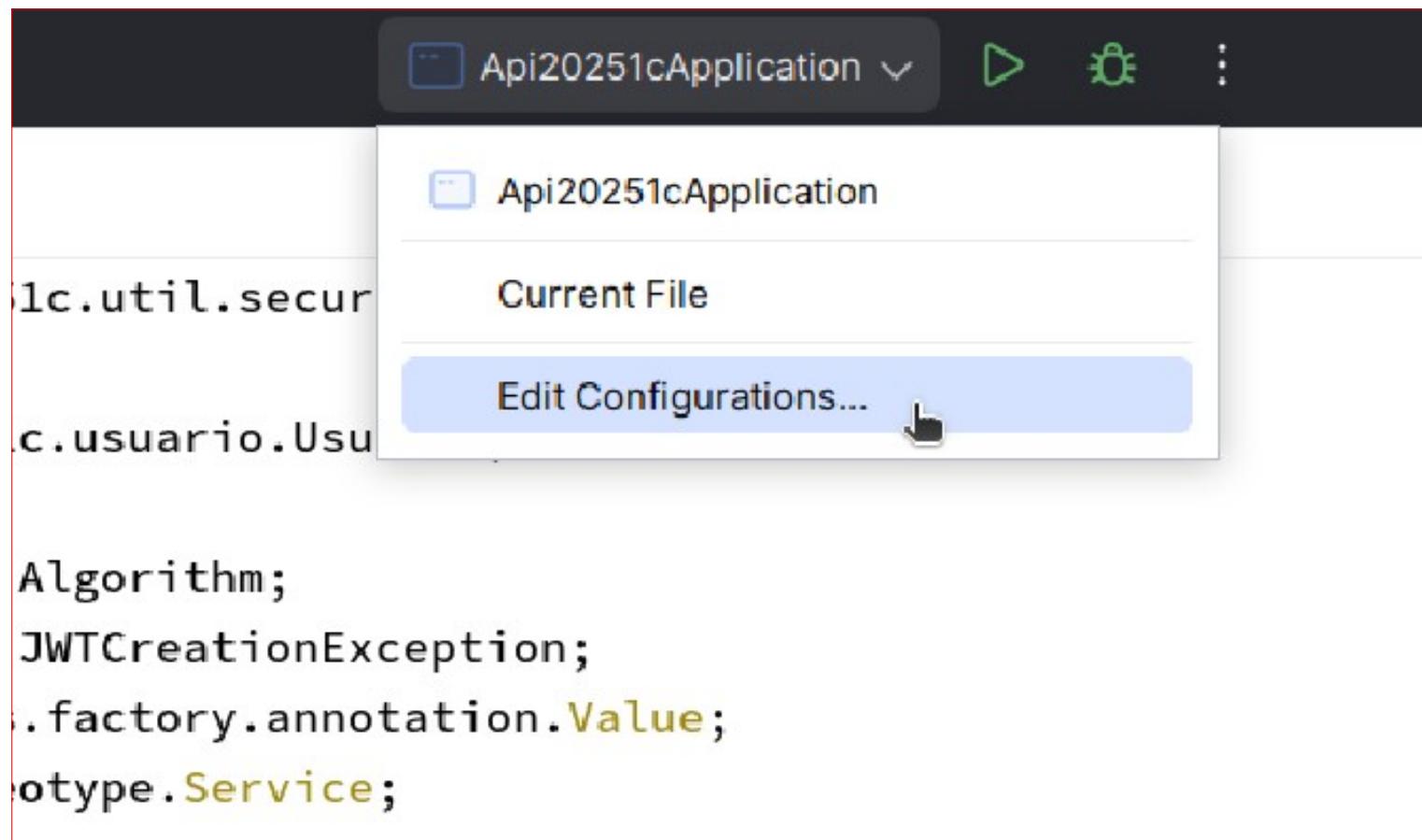
        (secret);

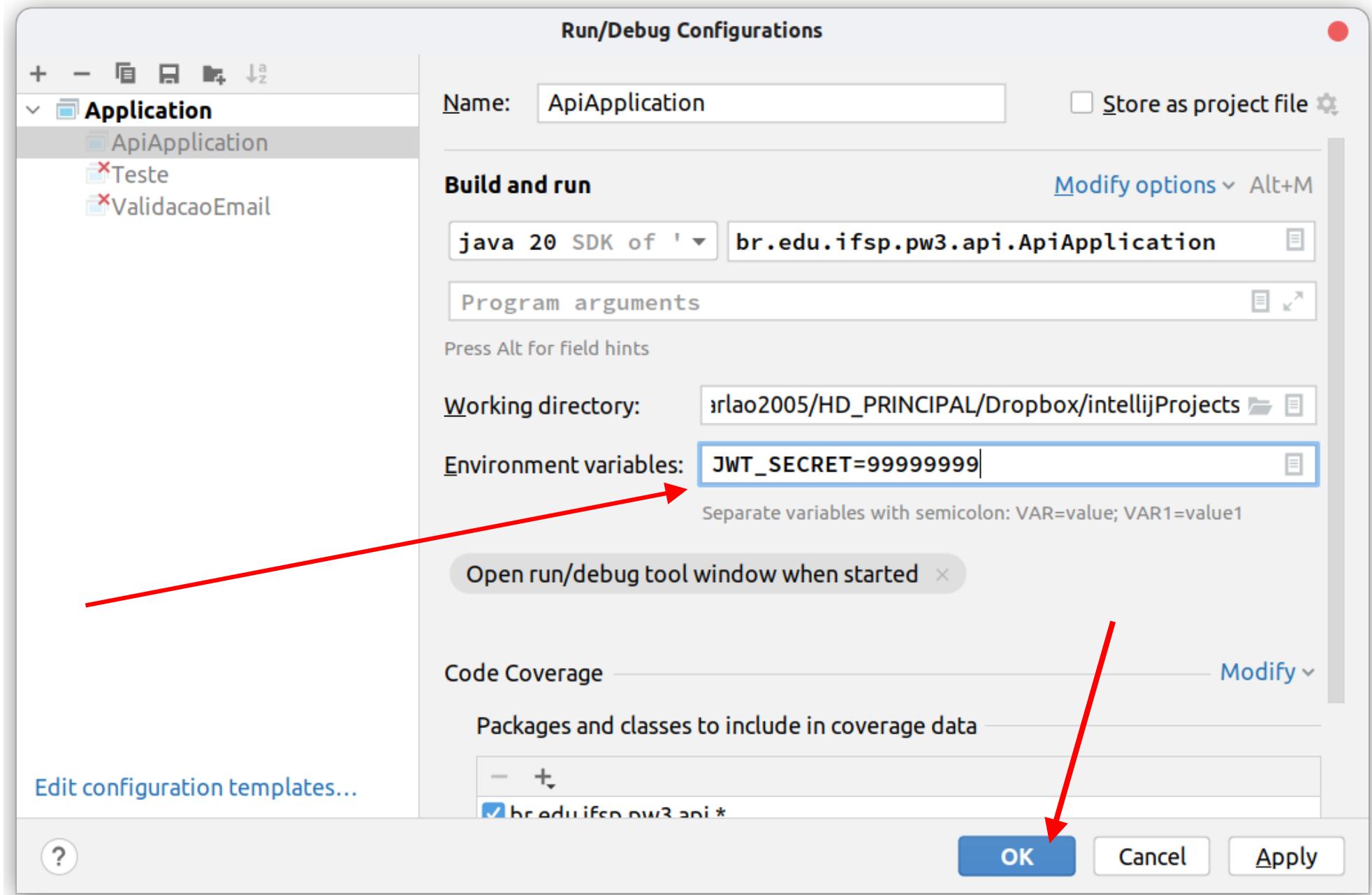
        = Algorithm.HMAC256(secret);
    }
}
```

A red arrow points from the text "Criando (simulando) uma variável de ambiente..." at the top of the slide down towards the top right corner of the code editor window.

Criando (simulando) uma variável de ambiente...

"edit configurations"





Testando novamente...

The screenshot shows a POST request to `localhost:8080/login`. The request body is a JSON object:

```
1 {  
2   "login": "asdrubal@gmail.com",  
3   "senha": "123456"  
4 }  
5
```

The response status is `200 OK` with `795 ms` duration and `518 B` size. The response body is a JSON object containing a token:

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cC  
eyJpc3Mi0iJESVNDSVBMSU5BIFBXMyIs  
NvbSIIsImV4cCI6MTY50Tgz0DQ4MX0.  
zJdgYoQq_cP8Ab2hDhJNhRoMKdk5VEnK  
3 }  
4
```

The terminal output shows a Hibernate query:

```
2023-11-12T20:29:17.007-03:00 INFO 14140 --- [nic  
Hibernate:  
    select  
        u1_0.id,  
        u1_0.login,  
        u1_0.senha  
    from  
        usuarios u1_0  
    where  
        u1_0.login=?  
99999999
```

A red arrow points from the bottom left towards the terminal window, highlighting the query parameters.

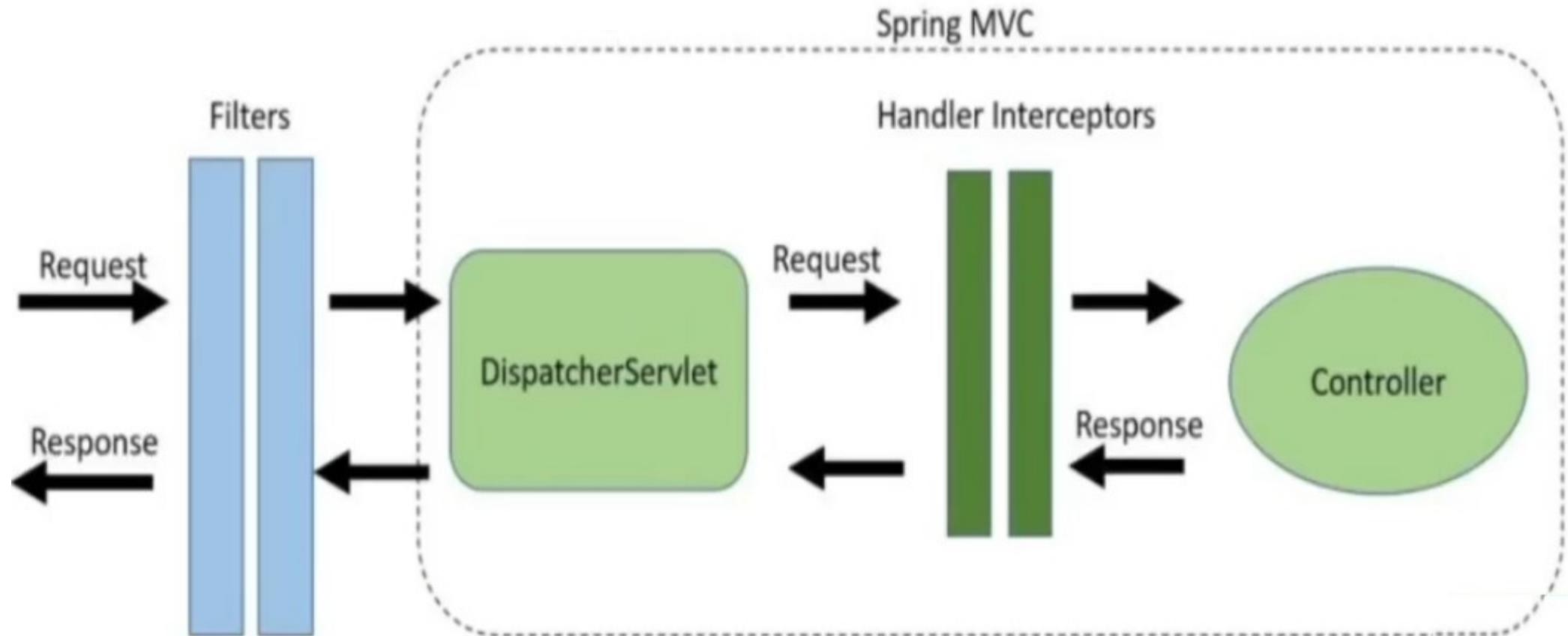
JWT: ACOMPANHAMENTO DOS PASSOS

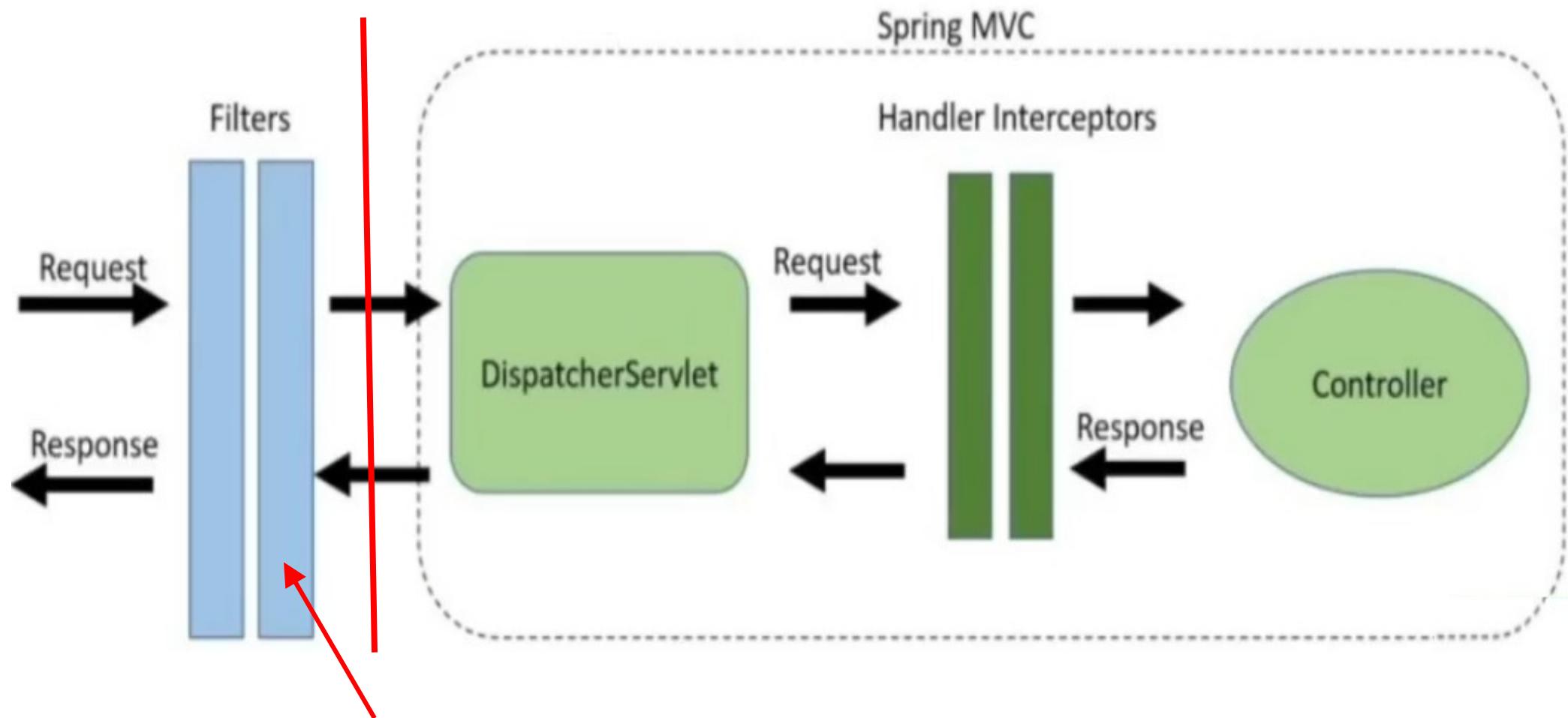
- Inserida dependência no projeto
- Criada classe para gerenciar tokens: PW3TokenService
 - método gerarToken()
- Alterado o AutenticacaoController
 - alterado método efetuarLogin(), para gerar e devolver o JWT no corpo da resposta
- Criado DTO para devolver o token JWT (**DadosTokenJWT**)
- Alterado o AutenticacaoController
 - alterado método efetuarLogin(), para gerar e devolver o JWT no corpo da resposta, como DTO
- **Criado parâmetro no application.properties para ler a senha principal de uma variável de ambiente**
- **Alterado código do método gerarToken() para usar a senha criada no item anterior.**

Processo para
verificar a validade
das requisições

(se contém um JWT válido, etc...)

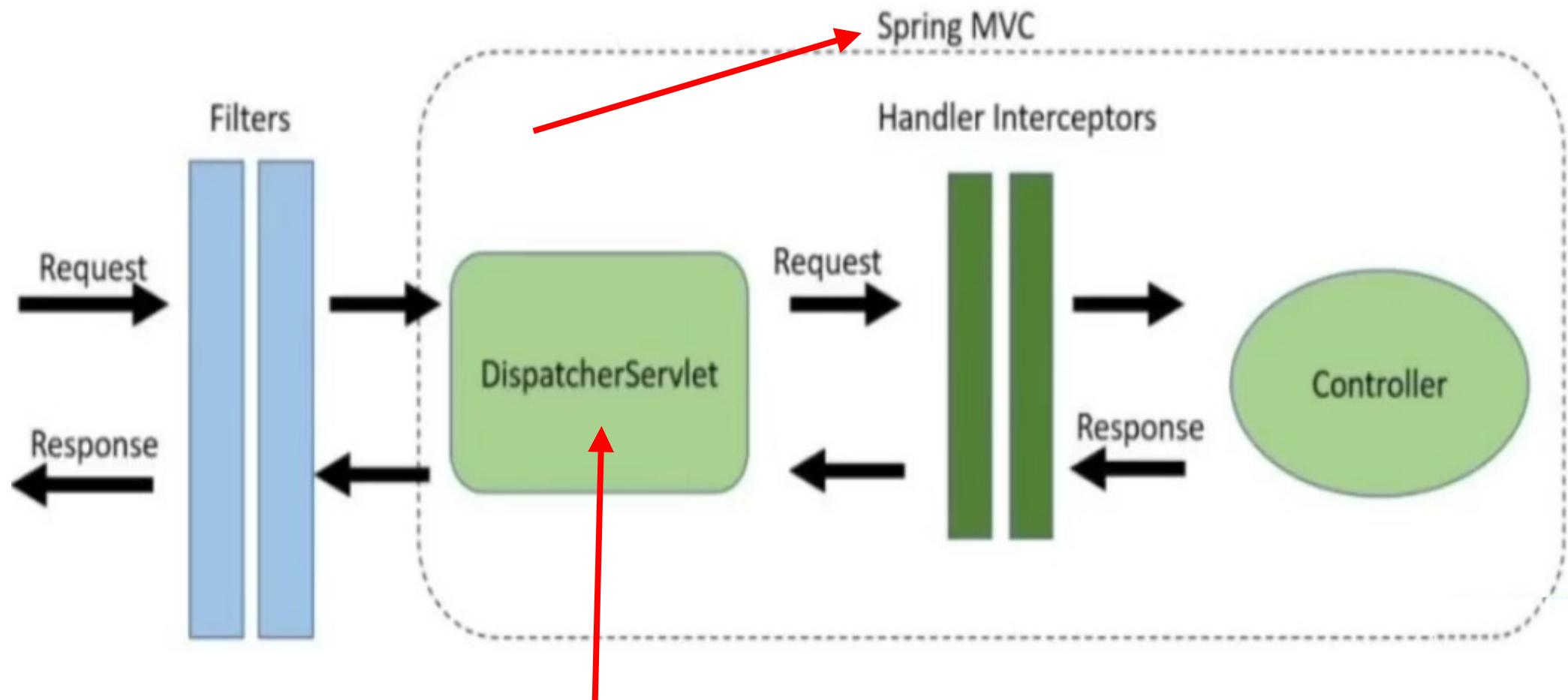
Spring MVC





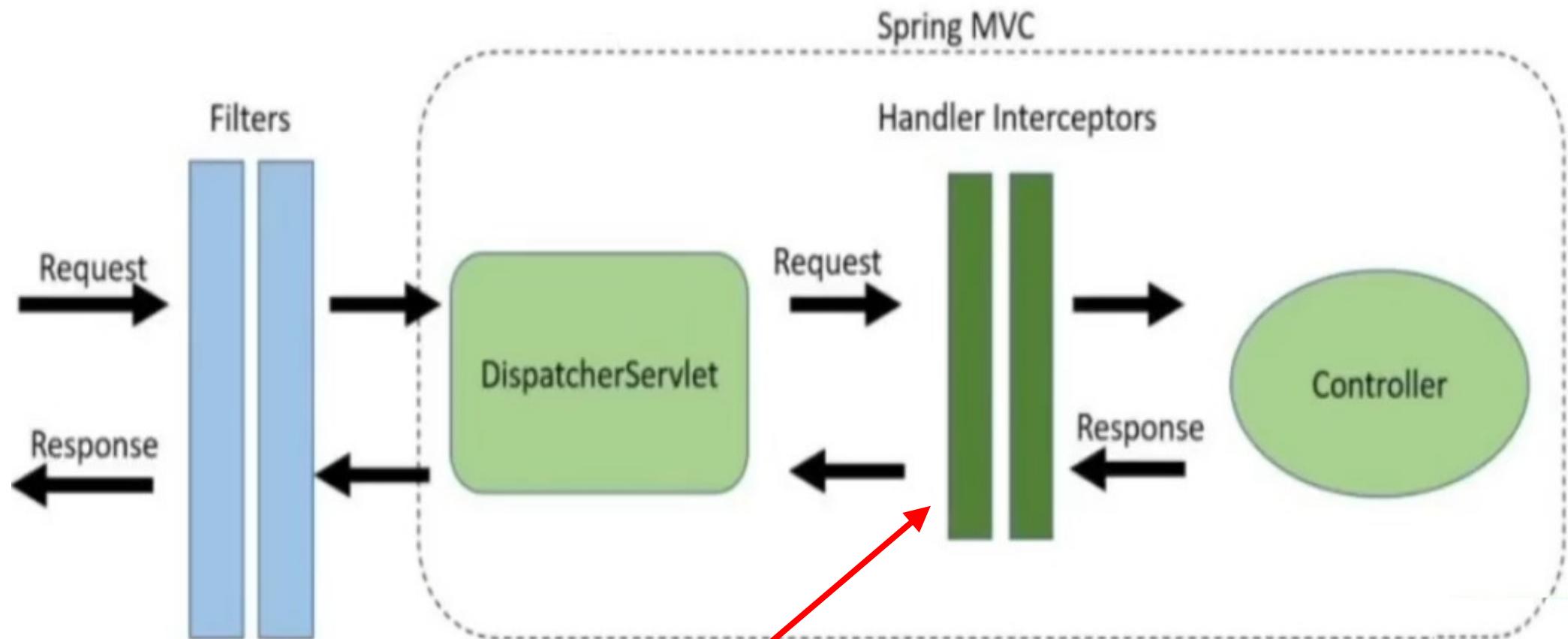
Em uma aplicação web Java, os filtros (servlet filters) são componentes que fornecem funcionalidades de pré-processamento e pós-processamento para solicitações e respostas HTTP. Eles são parte integrante da especificação Java Servlet e são configurados no arquivo `'web.xml'` da aplicação.

A função principal de um filtro é processar solicitações HTTP antes que essas solicitações alcancem um servlet ou recurso da aplicação. Eles são úteis para realizar tarefas como autenticação, autorização, manipulação de cabeçalhos, log de requisições, compressão de resposta, entre outras.



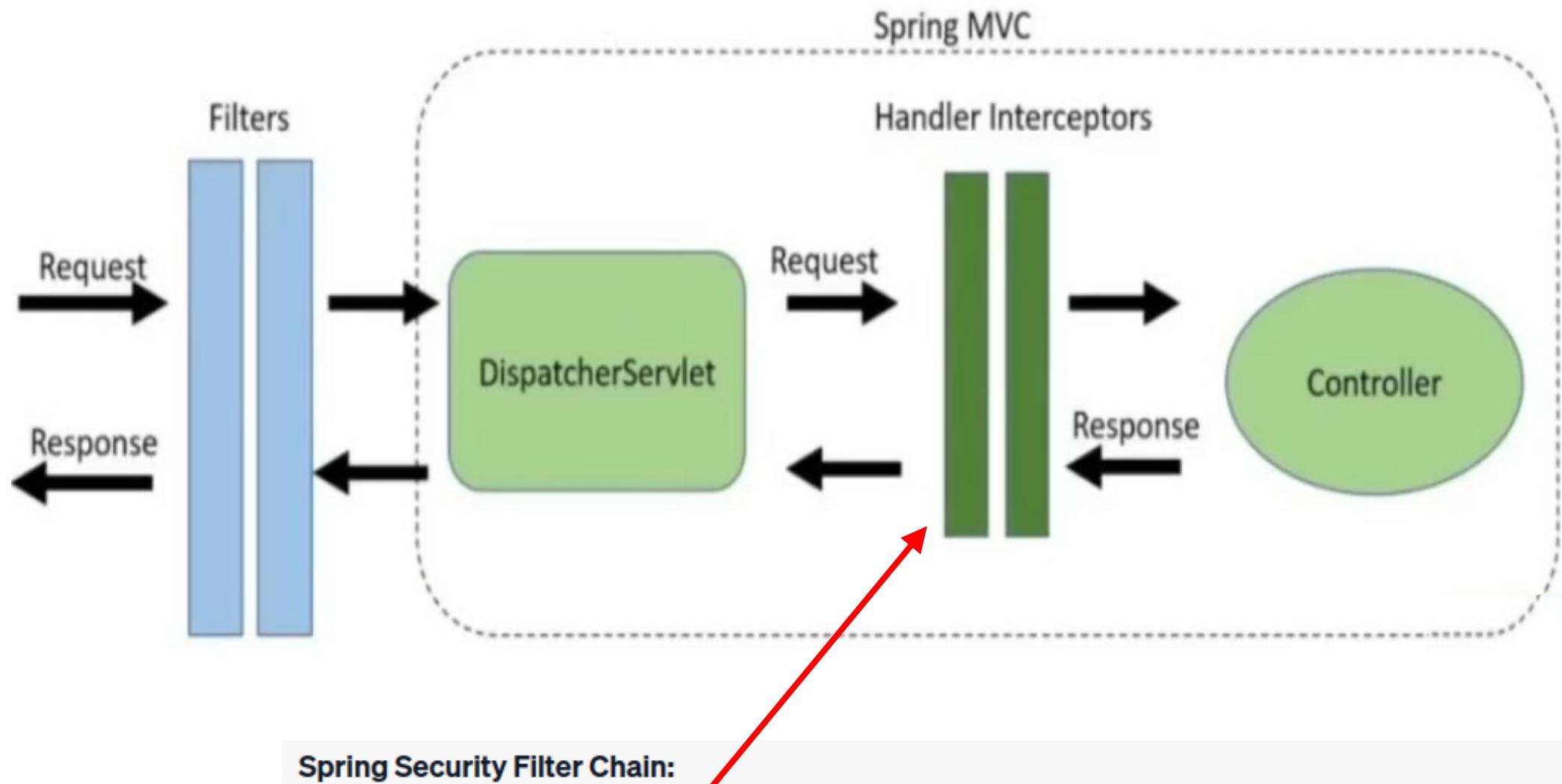
DispatcherServlet:

- O '**DispatcherServlet**' é uma parte central do framework Spring MVC. Ele atua como um controlador frontal que recebe todas as requisições HTTP para a sua aplicação.
- Quando uma requisição chega, o '**DispatcherServlet**' decide como encaminhar essa requisição com base em mapeamentos de URL definidos em seu contexto.



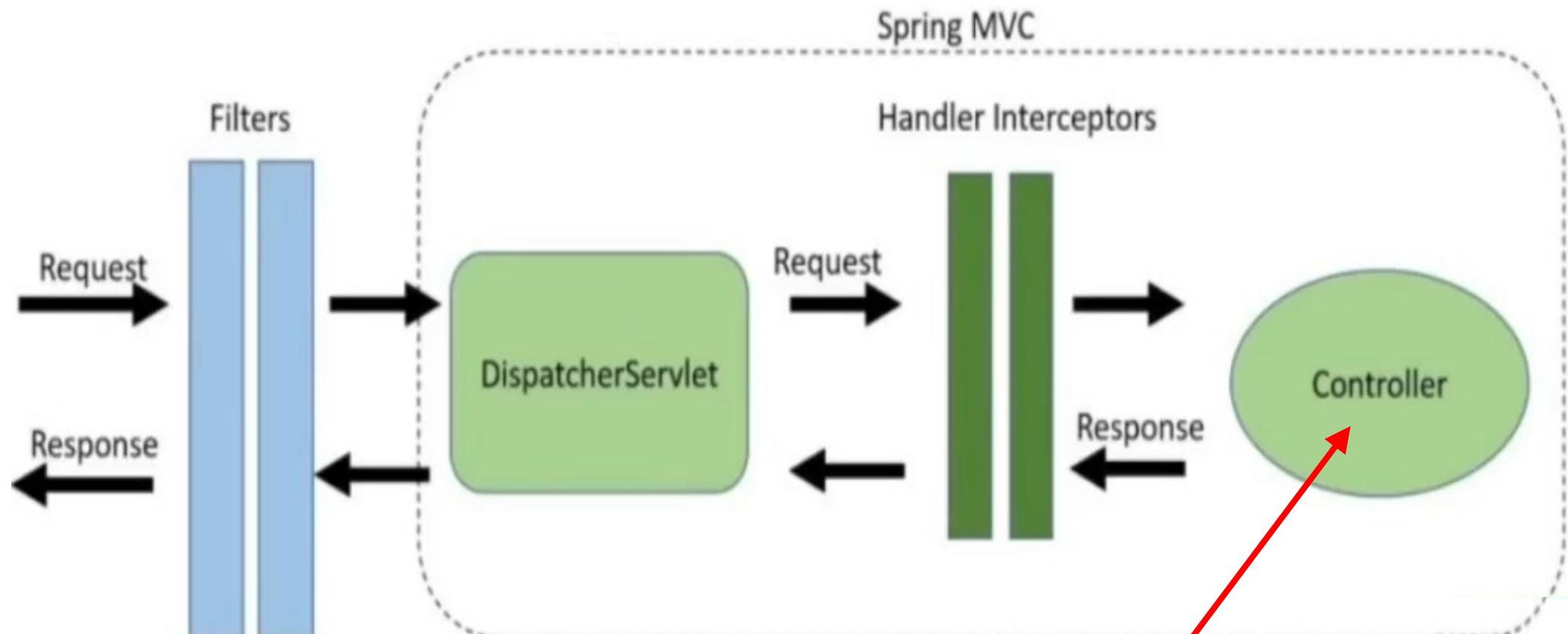
Handler Interceptors:

- Os interceptadores de manipuladores (handler interceptors) são componentes que podem ser configurados para executar operações antes ou depois que o `DispatcherServlet` envie uma requisição para o controlador ou depois que o controlador processa a requisição.
- No contexto do Spring Security, os interceptadores podem ser usados para realizar ações de segurança antes que a solicitação chegue ao controlador. Por exemplo, verificar se o usuário está autenticado, se tem as permissões necessárias, etc.



Spring Security Filter Chain:

- O Spring Security opera principalmente por meio de uma cadeia de filtros (filter chain) que são aplicados antes que a requisição alcance os controladores.
- Cada filtro na cadeia tem uma responsabilidade específica relacionada à segurança, como autenticação, autorização, prevenção contra ataques CSRF, etc.
- A configuração da cadeia de filtros é geralmente feita no arquivo de configuração de segurança (**'SecurityConfig'**), onde você pode especificar quais filtros devem ser aplicados e em que ordem.

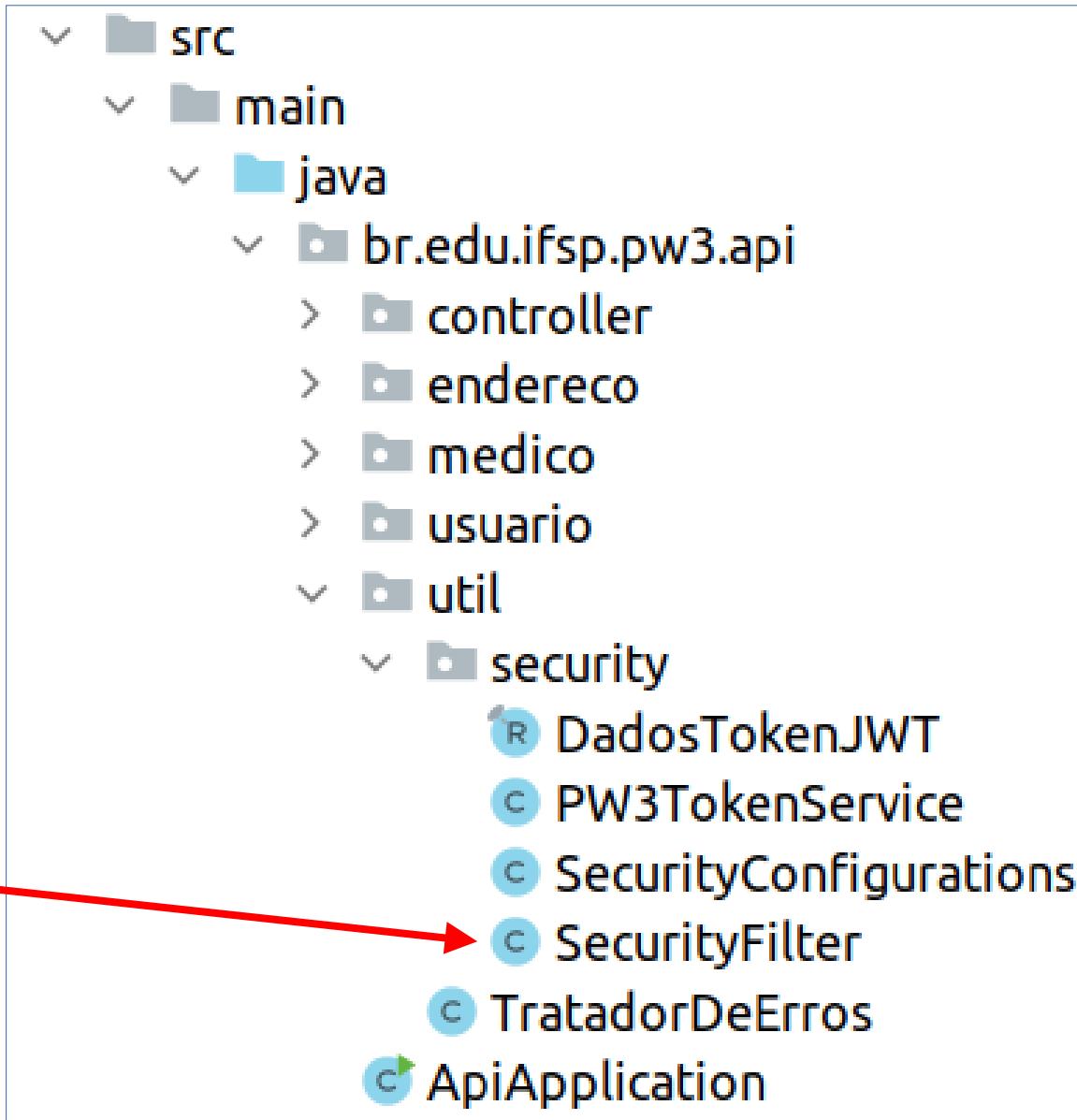


Controlador (Controller):

- O controlador é responsável por receber a requisição e processá-la. Ele contém a lógica de negócios para determinada rota ou URI.
- O controlador pode ser protegido por configurações de segurança do Spring Security. Por exemplo, você pode exigir autenticação para acessar determinadas rotas ou verificar se o usuário possui as permissões necessárias.

Vamos lá!!!





```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        System.out.println("FILTRO CHAMADO");

    }
}
```

```
@Component ←
public class SecurityFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {
        System.out.println("FILTRO CHAMADO");
    }
}
```

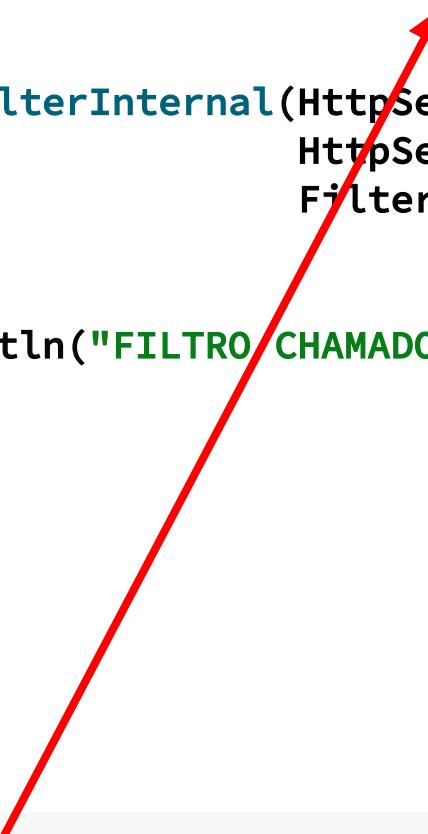
A anotação `@Component` no Spring é usada para indicar que uma classe é um **componente gerenciado pelo Spring**, ou seja, um **bean**. Ela serve para que o Spring detecte automaticamente essa classe durante o `component scan` e a registre no `ApplicationContext`.

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        System.out.println("FILTRO CHAMADO");

    }
}
```



A classe `OncePerRequestFilter` é uma classe utilitária fornecida pelo Spring Security para simplificar a implementação de filtros que precisam ser executados uma vez por solicitação. Essa classe garante que o filtro seja chamado apenas uma vez para cada solicitação HTTP.

A razão pela qual você pode querer garantir que seu filtro seja executado apenas uma vez por solicitação é evitar a execução repetida do mesmo código em uma única solicitação HTTP, o que pode ser ineficiente e causar comportamento indesejado.

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        System.out.println("FILTRO CHAMADO");

    }
}
```

Só faz sentido validar o token JWT
uma única vez!!

A classe `OncePerRequestFilter` é uma classe utilitária fornecida pelo Spring Security para simplificar a implementação de filtros que precisam ser executados uma vez por solicitação. Essa classe garante que o filtro seja chamado apenas uma vez para cada solicitação HTTP.

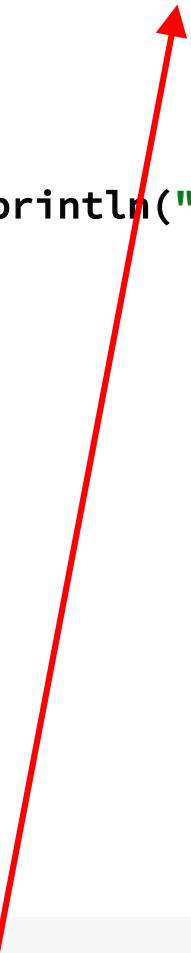
A razão pela qual você pode querer garantir que seu filtro seja executado apenas uma vez por solicitação é evitar a execução repetida do mesmo código em uma única solicitação HTTP, o que pode ser ineficiente e causar comportamento indesejado.

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        System.out.println("FILTR0 CHAMADO");

    }
}
```



O método `doFilterInternal` é um método da classe `OncePerRequestFilter` e é o lugar onde você deve colocar a lógica específica do filtro. Este método é invocado para cada solicitação HTTP que passa pelo filtro.

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        System.out.println("FILTR0 CHAMADO");

    }
}
```



Parâmetros:

- `HttpServletRequest request`: Representa a solicitação HTTP recebida pelo servidor. Contém informações sobre a solicitação, como parâmetros, cabeçalhos e corpo da solicitação.
- `HttpServletResponse response`: Representa a resposta HTTP que será enviada de volta ao cliente. Permite manipular o conteúdo e os cabeçalhos da resposta.
- `FilterChain filterChain`: É um objeto que representa a cadeia de filtros. Após a execução do código no método `doFilterInternal`, você chama `filterChain.doFilter(request, response)` para continuar o processamento da solicitação.

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        System.out.println("FILTRO CHAMADO");
    }
}
```



Só pra testar se o filtro foi chamado...

Testando

GET localhost:8080/medicos Send

Params Auth Headers (7) Body Pre-req. Tests Settings

Query Params

	Key	Value	Description	...	Bulk Ed
	Key	Value	Description		

Body 200 OK 75 ms 293 B Save as example

Pretty Raw Preview Visualize Text 1

GET localhost:8080/medicos Send

Params Auth Headers (7) Body Pre-req. Tests Settings

Query Params

	Key	Value	Description	...	Bulk Ed
	Key	Value	Description		

Body ▼ 🌐 200 OK 75 ms 293 B 💾 Save as example

Pretty Raw Preview Visualize Text ▼ 🔗 ◻

1

Mas cadê os dados??!??



GET localhost:8080/medicos Send

Params Auth Headers (7) Body Pre-req. Tests Settings

Query Params

	Key	Value	Description	...	Bulk Ed
	Key	Value	Description		

Body 200 OK 75 ms 293 B Save as example

Pretty Raw Preview Visualize Text ▾

1

Chamada do Próximo Filtro:

- Após executar a lógica do filtro, é importante chamar `'filterChain.doFilter(request, response)'` para garantir que a solicitação seja passada para o próximo filtro na cadeia ou, eventualmente, para o servlet ou controlador que manipulará a solicitação.
- Se você omitir a chamada `'filterChain.doFilter(request, response)'`, a solicitação não será continuada pela cadeia de filtros, e o processamento será interrompido.

Bookmarks

Structure

2023-11-12T23:27:25.468-03:00	WARN	8465	---	[resta
2023-11-12T23:27:25.531-03:00	INFO	8465	---	[resta
2023-11-12T23:27:25.990-03:00	INFO	8465	---	[resta
2023-11-12T23:27:26.021-03:00	INFO	8465	---	[resta
2023-11-12T23:27:26.038-03:00	INFO	8465	---	[resta
2023-11-12T23:27:30.399-03:00	INFO	8465	---	[nio-808	
2023-11-12T23:27:30.399-03:00	INFO	8465	---	[nio-808	
2023-11-12T23:27:30.401-03:00	INFO	8465	---	[nio-808	
FILTRO CHAMADO					

Version Control Run TODO Problems Terminal Services

All files are up-to-date (moments ago)



```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        filterChain.doFilter(request, response);

    }
}
```

Novo teste:

GET



localhost:8080/medicos

Send

Params Auth Headers (7) Body Pre-req. Tests Settings

0+

Query Params

	Key	Value	Description	...	Bulk Edi
	Key	Value	Description		

Body



200 OK 183 ms 1.15 KB



Save as example

Pretty

Raw

Preview

Visualize

JSON



(

```
1  []
2  {
3      "id": 1,
4      "nome": "Asdrubal Almeida",
5      "email": "asd@gmail.com",
6      "telefone": "16992481111",
7      "crm": "111111",
8      "especialidade": "ORTOPEDIA",
9      "endereco": {
10         "logradouro": "RUA UM",
11         "bairro": "BAIRRO UM",
12         "cep": "11111111",
```

GET



localhost:8080/medicos

Send

Params Auth Headers (7) Body Pre-req. Tests Settings

01

Query Params

	Key	Value	Description	...	Bulk Edi
	Key	Value	Description		

Body



200 OK 183 ms 1.15 KB



Save as example

Pretty

Raw

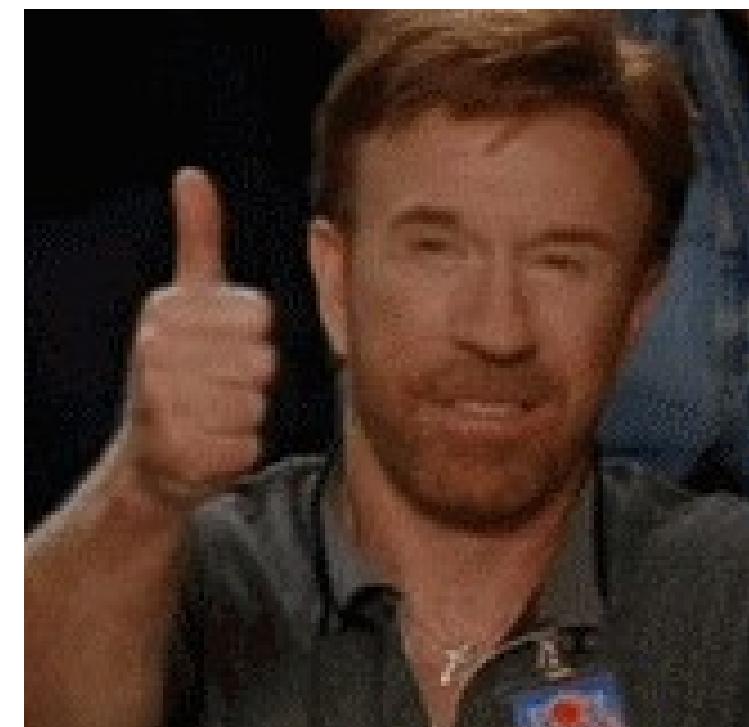
Preview

Visualize

JSON



```
1 [ {  
2   "id": 1,  
3   "nome": "Asdrubal Almeida",  
4   "email": "asd@gmail.com",  
5   "telefone": "16992481111",  
6   "crm": "111111",  
7   "especialidade": "ORTOPEDIA",  
8   "endereco": {  
9     "logradouro": "RUA UM",  
10    "bairro": "BAIRRO UM",  
11    "cep": "11111111",  
12  }
```



Envio do token na requisição.

Feito no Header (cabeçalho).

Cabeçalho Authorization.

Exemplificando no Postman...

The screenshot shows the Postman application interface. At the top, there is a header bar with 'GET' selected, a URL input field containing 'localhost:8080/medicos', a 'Send' button, and a dropdown menu. Below the header, there are tabs for 'Params', 'Auth' (which is currently selected), 'Headers (8)', 'Body', 'Pre-req.', 'Tests', 'Settings', and '...'.

The 'Auth' tab is active, and a red arrow points to the 'Type' dropdown which is set to 'Bearer Token'. Another red arrow points to the token value itself, which is a long string of characters: eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFZZHJ1YmFsQGdtYWlsLmNvbSIslmV4cCI6MTY5OTg1MDQyOX0.WXUKm6FkBpu8I5nXZadpsDrClZxiQqCGghqlHE0k3gs.

O termo "Bearer" refere-se a um tipo de esquema de autenticação utilizado em solicitações HTTP, especialmente quando se trata de enviar tokens de acesso, como tokens JWT (JSON Web Tokens). Esse esquema é conhecido como "Bearer Authentication".

Em um esquema de autenticação Bearer, o token é incluído na solicitação HTTP no cabeçalho "Authorization" de uma maneira específica. Ao usar o esquema "Bearer", o token é precedido pela palavra-chave "Bearer", seguida por um espaço em branco e, em seguida, o próprio token. Por exemplo:

No código...

Alterando método doFilterInternal()

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        // Recuperando o cabeçalho Authorization:
        var authorizationHeader = request.getHeader("Authorization");
        // Testar se o cabeçalho foi enviado:
        if (authorizationHeader == null) {
            throw new RuntimeException("Token JWT não enviado!");
        }

        // Só pra testar:
        System.out.println(authorizationHeader);

        filterChain.doFilter(request, response);
    }
}
```

Alterando método doFilterInternal()

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        // Recuperando o cabeçalho Authorization:
        var authorizationHeader = request.getHeader("Authorization");
        // Testar se o cabeçalho foi enviado:
        if (authorizationHeader == null) {
            throw new RuntimeException("Token JWT não enviado!");
        }

        // Só pra testar:
        System.out.println(authorizationHeader);

        filterChain.doFilter(request, response);
    }
}
```

A linha de código que você mencionou está recuperando o cabeçalho "Authorization" da solicitação HTTP usando o método `getHeader("Authorization")` do objeto `HttpServletRequest`.

Método doFilterInternal()

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        // Recuperando o cabeçalho Authorization:
        var authorizationHeader = request.getHeader("Authorization");
        // Testar se o cabeçalho foi enviado:
        if (authorizationHeader == null) {
            throw new RuntimeException("Token JWT não enviado!");
        }

        // Só pra testar:
        System.out.println(authorizationHeader);

        filterChain.doFilter(request, response);
    }
}
```

Método doFilterInternal()

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        // Recuperando o cabeçalho Authorization:
        var authorizationHeader = request.getHeader("Authorization");
        // Testar se o cabeçalho foi enviado:
        if (authorizationHeader == null) {
            throw new RuntimeException("Token JWT não enviado!");
        }

        // Só pra testar:
        System.out.println(authorizationHeader); ←

        filterChain.doFilter(request, response);
    }
}
```

Testando...

GET ▼ localhost:8080/medicos Send ▼

Params Auth ● Headers (8) Body Pre-req. Tests Settings ...

Type Bearer Token ▼

The authorization header will be automatically generated when you send the

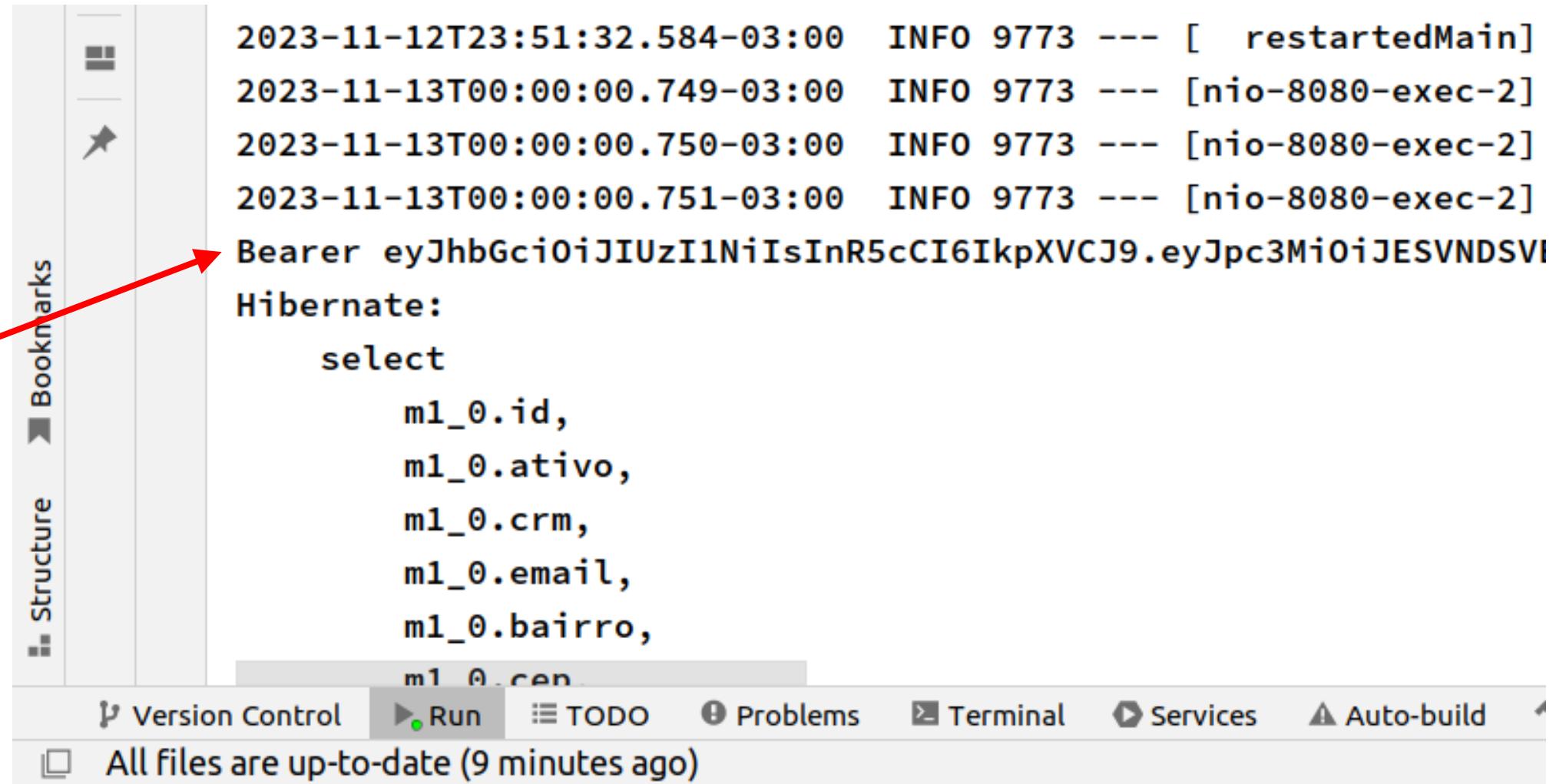
Token

```
eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJpc3MiOiJESVNDSVBMSU5BIFBXMylsInN1Yil6ImFzZHJ1YmFsQGdtYWlsLmNvbSlsImV4cCI6MTY5OTg1MDQyOX0.WXUKm6FkBpu8I5nXZadpsDrClZxiQqCGghqlHE0k3gs
```

Body ▼ 200 OK 221 ms 1.15 KB Save as example ...

Pretty Raw Preview Visualize JSON ▼ CSV ▼ Copy □ Search Q

```
1 {  
2     "id": 1,  
3     "nome": "Asdrubal Almeida",  
4     "email": "asd@gmail.com",  
5     "telefone": "16992481111",  
6     "crm": "111111",  
7     "especialidade": "ORTOPEDIA",  
8     "endereco": {  
9         "logradouro": "RUA UM",  
10    }  
}
```

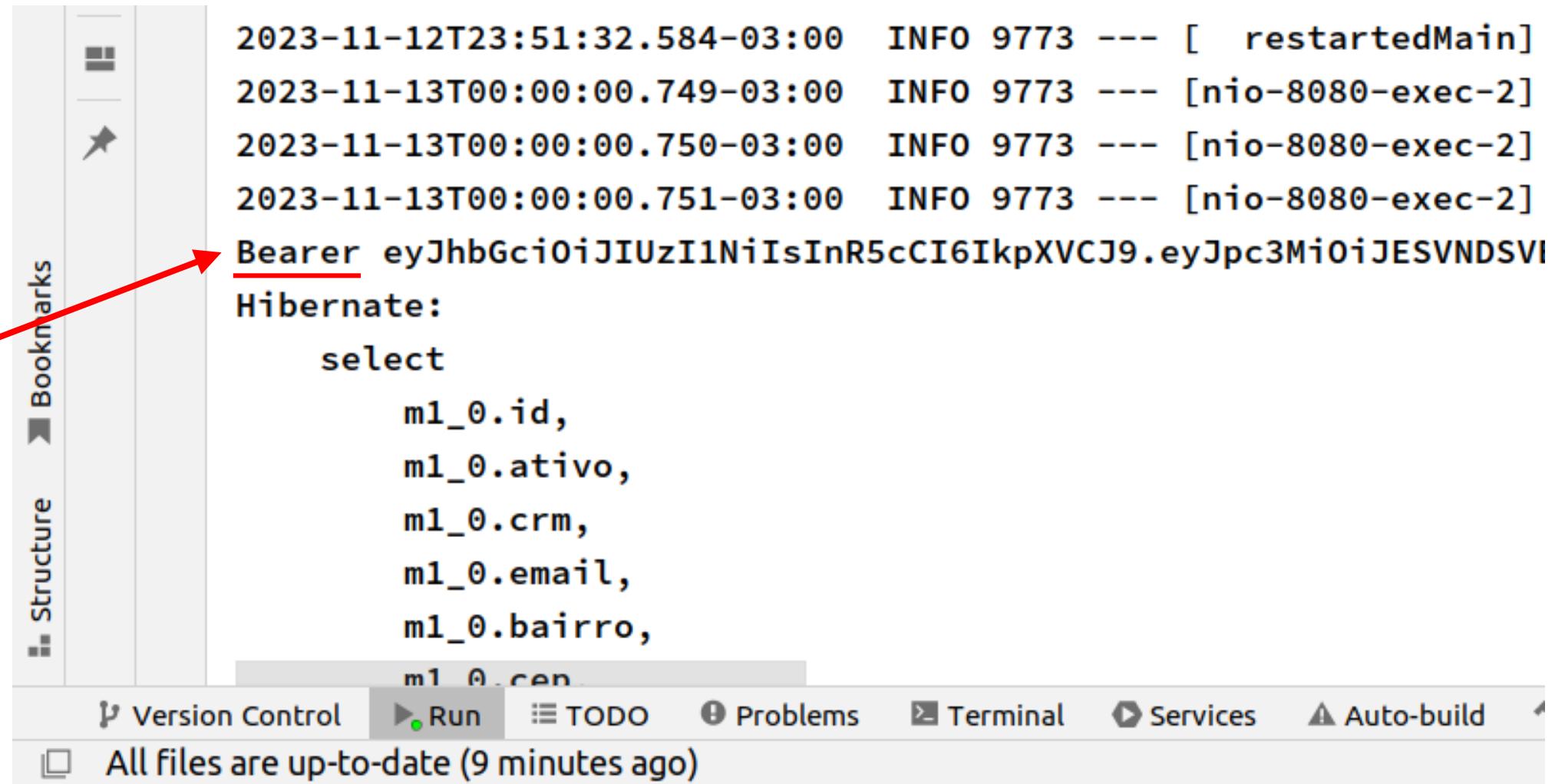


2023-11-12T23:51:32.584-03:00 INFO 9773 --- [restartedMain]
2023-11-13T00:00:00.749-03:00 INFO 9773 --- [nio-8080-exec-2]
2023-11-13T00:00:00.750-03:00 INFO 9773 --- [nio-8080-exec-2]
2023-11-13T00:00:00.751-03:00 INFO 9773 --- [nio-8080-exec-2]
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJESVNDSVI
Hibernate:
 select
 m1_0.id,
 m1_0.ativo,
 m1_0.crm,
 m1_0.email,
 m1_0.bairro,
 m1_0.cen

Version Control Run TODO Problems Terminal Services Auto-build

All files are up-to-date (9 minutes ago)

Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJESVNDSVBMSU5BIFBXMyI
sInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIslmV4cCI6MTY5OTg1MDQyOX0.WX
UKm6FkBpu8I5nXZadpsDrClZxiQqCGghqLHE0k3gs



2023-11-12T23:51:32.584-03:00 INFO 9773 --- [restartedMain]
2023-11-13T00:00:00.749-03:00 INFO 9773 --- [nio-8080-exec-2]
2023-11-13T00:00:00.750-03:00 INFO 9773 --- [nio-8080-exec-2]
2023-11-13T00:00:00.751-03:00 INFO 9773 --- [nio-8080-exec-2]
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3Mi0iJESVNDSVI
Hibernate:
 select
 m1_0.id,
 m1_0.ativo,
 m1_0.crm,
 m1_0.email,
 m1_0.bairro,
 m1_0.cen

Version Control Run TODO Problems Terminal Services Auto-build

All files are up-to-date (9 minutes ago)

Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3Mi0iJESVNDSVBMSU5BIFBXMyI
sInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIslmV4cCI6MTY5OTg1MDQyOX0.WX
UKm6FkBpu8I5nXZadpsDrClZxiQqCGghqLHE0k3gs

Vamos tirar esse 'Bearer' da string!

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        // Recuperando o cabeçalho Authorization:
        var authorizationHeader = request.getHeader("Authorization");
        // Testar se o cabeçalho foi enviado:
        if (authorizationHeader == null) {
            throw new RuntimeException("Token JWT não enviado!");
        }

        authorizationHeader = authorizationHeader.replace("Bearer ","");

        // Só pra testar:
        System.out.println(authorizationHeader);

        filterChain.doFilter(request, response);
    }
}
```

Testando novamente...

2023-11-13T00:04:39.631-03:00 INFO 9773 --- [restartedMain]
2023-11-13T00:05:23.658-03:00 INFO 9773 --- [nio-8080-exec-2]
2023-11-13T00:05:23.658-03:00 INFO 9773 --- [nio-8080-exec-2]
2023-11-13T00:05:23.659-03:00 INFO 9773 --- [nio-8080-exec-2]
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJESVNDSVBMSU5BI

Hibernate:

```
select  
    m1_0.id,  
    m1_0.ativo,  
    m1_0.crm,  
    m1_0.email,  
    m1_0.bairro,
```

m1_0.cen

Bookmarks

Structure

Version Control

Run

TODO

Problems

Terminal

Services

Auto-build

All files are up-to-date (14 minutes ago)

Se não enviar o token...

```
// Testar se o cabeçalho foi enviado:  
if (authorizationHeader == null) {  
    throw new RuntimeException("Token JWT não enviado!");  
}
```

Type

No Auth

This request does not use any authorization. Learn more about [authorization](#) ↗

Body   500 Internal Server Error 32 ms 468 B  Save as example 

Pretty Raw Preview Visualize JSON   

```
1 {  
2     "timestamp": "2025-05-16T19:11:20.306+00:00",  
3     "status": 500,  
4     "error": "Internal Server Error",  
5     "path": "/medicos"  
6 }  
7 }
```

```
java.lang.RuntimeException Create breakpoint : Token JWT não enviado!
at br.edu.ifsp.prw3.api20251c.util.security.SecurityFilter.doFilterInternal(SecurityFilter.java:26)
at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:116) ~[spr
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:164)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:140) ~[tomca
at org.springframework.web.filter.CompositeFilter$VirtualFilterChain.doFilter(CompositeFilter.java:1
at org.springframework.security.web.FilterChainProxy.lambda\$doFilterInternal\$3\(FilterChainProxy.java:1
at org.springframework.security.web.FilterChainProxy\\$VirtualFilterChain.doFilter\\(FilterChainProxy.ja
at org.springframework.security.web.access.ExceptionTranslationFilter.doFilter\\\(ExceptionTranslationF
<66 folded frames>
```



Conseguimos recuperar o token JWT
enviado no cabeçalho.

Precisamos agora validar o token!

Criamos uma classe para gerenciar tokens
(PW3TokenService).

Vamos criar um método nessa classe,
para verificar a validade do token,
e buscar o subject dentro dele
(o login do usuário).

```
@Service
public class PW3TokenService {

    @Value("${pw3.senha.principal.geracao.token}")
    private String secret;

    public String gerarToken(Usuario usuario) { ... }

    // Valida o token, e recupera o subject (login) dentro do token:
    public String getSubject(String tokenJWT) {

        try {
            var algoritmo = Algorithm.HMAC256(secret);
            JWTVerifier jwtv = JWT.require(algoritmo)
                .withIssuer("DISCIPLINA PW3")
                .build();
            return jwtv.verify(tokenJWT).getSubject();

        } catch (JWTVerificationException exception){
            throw new RuntimeException("Token JWT inválido ou expirado!");
        }
    }

    private Instant dataExpiracao() { ... }
}
```

```
@Service
public class PW3TokenService {

    @Value("${pw3.senha.principal.geracao.token}")
    private String secret;

    public String gerarToken(Usuario usuario) { ... }

    // Valida o token, e recupera o subject (login) dentro do token:
    public String getSubject(String tokenJWT) {

        try {
            var algoritmo = Algorithm.HMAC256(secret); ←
            JWTVerifier jwtv = JWT.require(algoritmo)
                .withIssuer("DISCIPLINA PW3")
                .build();
            return jwtv.verify(tokenJWT).getSubject();

        } catch (JWTVerificationException exception){
            throw new RuntimeException("Token JWT inválido ou expirado!");
        }
    }

    private Instant dataExpiracao() { ... }
}
```

```
@Service
public class PW3TokenService {

    @Value("${pw3.senha.principal.geracao.token}")
    private String secret;

    public String gerarToken(Usuario usuario) { ... }

    // Valida o token, e recupera o subject (login) dentro do token:
    public String getSubject(String tokenJWT) {

        try {
            var algoritmo = Algorithm.HMAC256(secret);
            JWTVerifier jwtv = JWT.require(algoritmo)
                .withIssuer("DISCIPLINA PW3")
                .build();

            return jwtv.verify(tokenJWT).getSubject();

        } catch (JWTVerificationException exception){
            throw new RuntimeException("Token JWT inválido ou expirado!");
        }
    }

    private Instant dataExpiracao() { ... }
}
```

~~VAR ALGORITMO = ALGORITMO.HMAC256(SECRETO);~~

```
JWTVerifier jwtv = JWT.require(algoritmo)
    .withIssuer("DISCIPLINA PW3")
    .build();

return jwtv.verify(tokenJWT).getSubject();
```

1. `JWTVerifier jwtv`: É a declaração de uma variável chamada `jwtv` que representa um verificador de tokens JWT. Esse verificador é usado para validar a estrutura e as reivindicações (claims) de um token JWT.
2. `JWT.require(algoritmo)`: O método `require` é usado para especificar os requisitos que o token JWT deve atender. Ele recebe como argumento o algoritmo de assinatura que será usado para verificar a autenticidade do token. No caso, `algoritmo` é uma instância de `Algorithm.HMAC256`, indicando que o algoritmo de HMAC SHA-256 será utilizado.
3. `.withIssuer("DISCIPLINA PW3")`: Este método especifica o "issuer" (emissor) esperado para o token JWT. O emissor é a entidade que emitiu o token. Neste caso, o emissor esperado é "DISCIPLINA PW3".
4. `.build()`: O método `build()` é chamado para finalizar a construção do objeto `JWTVerifier`. Após a chamada desse método, o objeto está pronto para ser usado para verificar a validade de um token JWT.

```

@Service
public class PW3TokenService {

    @Value("${pw3.senha.principal.geracao.token}")
    private String secret;

    public String gerarToken(Usuario usuario) { ... }

    // Valida o token, e recupera o subject (login) dentro do token:
    public String getSubject(String tokenJWT) {

        try {
            var algoritmo = Algorithm.HMAC256(secret);
            JWTVerifier jwtv = JWT.require(algoritmo)
                .withIssuer("DISCIPLINA PW3")
                .build();
            return jwtv.verify(tokenJWT).getSubject();
        }
    }
}

```

1. `jwtv.verify(tokenJWT)` : Aqui, `jwtv` é um objeto da classe `JWTVerifier` que foi criado anteriormente. O método `verify(tokenJWT)` é chamado para verificar se o token JWT (`tokenJWT`) é válido de acordo com os critérios estabelecidos no verificador. Se o token não for válido, uma exceção do tipo `JWTVerificationException` será lançada.
2. `getSubject()` : Após a verificação bem-sucedida do token, o método `getSubject()` é chamado para obter o assunto (subject) contido no token. O "subject" em um token JWT geralmente representa a entidade para a qual o token foi emitido, muitas vezes associada a um usuário ou identificador exclusivo.

Voltando para a classe SecurityFilter

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    // Injetar a classe PW3TokenService aqui:
    @Autowired
    private PW3TokenService pw3tokenservice;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException,
                                    IOException {

        // Recuperando o cabeçalho Authorization:
        var authorizationHeader = request.getHeader("Authorization");
        // Testar se o cabeçalho foi enviado:
        if (authorizationHeader == null) {
            throw new RuntimeException("Token JWT não enviado!");
        }

        authorizationHeader = authorizationHeader.replace("Bearer ","");

        // Chamar o método que valida o token, e recupera o login:
        var subject = pw3tokenservice.getSubject(authorizationHeader);

        // Só pra testar:
        System.out.println(subject);

        filterChain.doFilter(request, response);
    }
}
```

Testando...

GET

localhost:8080/medicos

Send

Params

Auth ● Headers (8)

Body

Pre-req.

Tests Settings

...

Type

Bearer Token

The authorization header will be automatically generated when you send the

Token

```
eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1Yil6ImFzZHJ1YmFsQGdtYWlsLmNvbSIslmV4cCI6MTY5OTg1MDQyOX0.WXUKm6FkBpu8I5nXZadpsDrCIZxiQqCGghqlHE0k3gs
```

Body



200 OK

329 ms

1.15 KB



Save as example

...

Pretty

Raw

Preview

Visualize

JSON



```
1
2 {
3     "id": 1,
4     "nome": "Asdrubal Almeida",
5     "email": "asd@gmail.com",
6     "telefone": "16992481111",
7     "crm": "111111",
8     "especialidade": "ORTOPEDIA",
9     "endereco": {
10         "logradouro": "RUA UM",
11         "bairro": "BAIRRO UM",
```

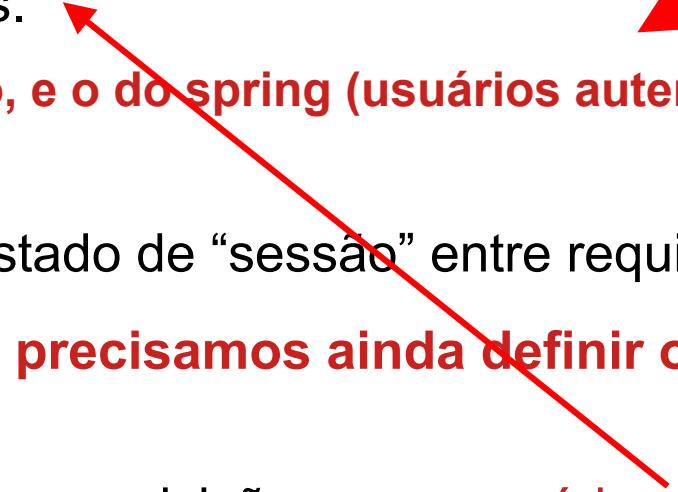
The screenshot shows a Java IDE interface with several panes:

- Left Sidebar:** Contains icons for Run, Stop, Refresh, and Bookmarks.
- Structure:** A tree view of project files.
- Log:** Displays the following log entries:
 - 2023-11-13T00:39:50.114-03:00 INFO 12097 --- [
 - 2023-11-13T00:39:50.136-03:00 INFO 12097 --- [
 - 2023-11-13T00:39:50.150-03:00 INFO 12097 --- [
 - 2023-11-13T00:39:54.233-03:00 INFO 12097 --- [n
 - 2023-11-13T00:39:54.233-03:00 INFO 12097 --- [n
 - 2023-11-13T00:39:54.235-03:00 INFO 12097 --- [n
- Code Editor:** Shows a Java code snippet using Hibernate:

```
asdrubal@gmail.com
Hibernate:
    select
        m1_0.id,
        m1_0.ativo,
        m1_0.crm,
        m1_0.email,
        m1_0.bairro,
```

A red arrow points from the email address "asdrubal@gmail.com" to the word "select".
- Bottom Navigation:** Includes tabs for Version Control, Run (highlighted), TODO, Problems, Terminal, and Services.
- Status Bar:** Shows a green icon indicating "All files are up-to-date (a minute ago)".

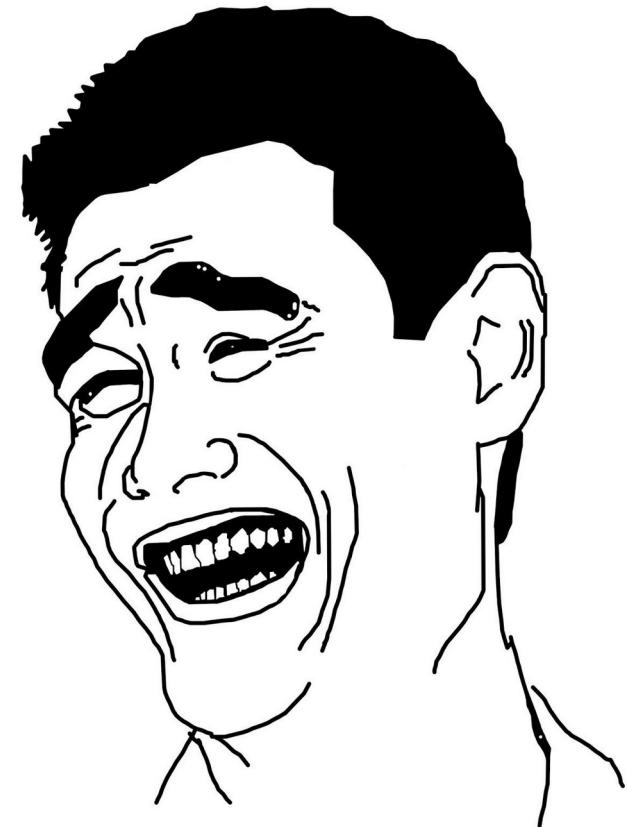
O que fizemos até agora ? (e o que precisamos fazer?)

- Criamos um filtro, que recupera um JWT no header da requisição.
 - Se o token JWT não estiver presente, é gerada uma exceção.
 - **Problema!!! /login não leva token!!!!** 
- Verificamos se o token é válido, e em caso positivo, recuperamos quem é o subject (neste caso, o email de login).
- Precisamos agora definir este usuário como “autenticado”.
 - Precisamos bloquear todas as requisições, menos a **/login**, só permitindo usuários autenticados. 
 - **Problema!! teremos 2 filtros, o nosso, e o do spring (usuários autenticados)**
 - **Em qual ordem vai executar ?** 
- Lembre-se, API REST não mantém o estado de “sessão” entre requisições.
- **Mas, se o token foi validado, por que precisamos ainda definir o usuário como “autenticado” ?**
 - Primeiro, pq o spring só vai liberar as requisições para **usuários autenticados**.
 - Também, porque depois de passar pelo filtro de segurança, as partes subsequentes do aplicativo, como os controladores, podem precisar saber quem é o usuário que está fazendo a solicitação para tomar decisões específicas com base nessa identidade.

Força aí... tá quase acabando!!!



**FOCA NO
DESENVOLVIMENTO!**



Alterações que precisamos fazer na classe SecurityConfigurations

- acesso a **/login** liberado pra todos
 - todas as outras requisições,
obrigar que usuário esteja autenticado.
- definir ordem de aplicação dos filtros de segurança
 - 1º, o nosso filtro
 - depois, o do spring (usuários autenticados)

```
@Configuration
@EnableWebSecurity
public class SecurityConfigurations {

    @Autowired
    private SecurityFilter securityFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        return http.csrf(csrf -> csrf.disable())
            .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            // configurar autorizações para as requisições:
            .authorizeHttpRequests(req -> {
                // liberar o /login
                // req.requestMatchers(antMatcher("/login")).permitAll();
                req.requestMatchers("/login").permitAll();
                // todas as outras, só para usuários autenticados
                req.anyRequest().authenticated();
            })
            // definindo a ordem de aplicação dos filtros:
            // primeiro, o nosso filtro (classe SecurityFilter injetada aqui)
            // depois, o filtro de usuarios autenticados do proprio spring
            .addFilterBefore(securityFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
    }

    @Bean
    public AuthenticationManager authenticationManager( ... )

    @Bean
    public PasswordEncoder passwordEncoder() { ...

    }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfigurations {

    @Autowired
    private SecurityFilter securityFilter;  // A red arrow points from the left margin to the 'securityFilter' field declaration.

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        return http.csrf(csrf -> csrf.disable())
            .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
 // A red arrow points from the left margin to the 'authorizeHttpRequests' block.

            // configurar autorizações para as requisições:
            .authorizeHttpRequests(req -> {
                // liberar o /login
                    req.requestMatchers(antMatcher("/login")).permitAll();
                req.requestMatchers("/login").permitAll();
                // todas as outras, só para usuários autenticados
                req.anyRequest().authenticated();
            })
            // definindo a ordem de aplicação dos filtros:
            // primeiro, o nosso filtro (classe SecurityFilter injetada aqui)
            // depois, o filtro de usuarios autenticados do proprio spring
            .addFilterBefore(securityFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
    }

    @Bean
    public AuthenticationManager authenticationManager( ... )

    @Bean
    public PasswordEncoder passwordEncoder() { ... }

}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfigurations {

    @Autowired
    private SecurityFilter securityFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        return http.csrf(csrf -> csrf.disable())
            .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            // configurar autorizações para as requisições:
            .authorizeHttpRequests(req -> {
                // liberar o /login
                // req.requestMatchers(antMatcher("/login")).permitAll();
                req.requestMatchers("/login").permitAll();
                // todas as outras, só para usuários autenticados
                req.anyRequest().authenticated();
            })
            // definindo a ordem de aplicação dos filtros:
            // primeiro, o nosso filtro (classe SecurityFilter injetada aqui)
            // depois, o filtro de usuarios autenticados do proprio spring
            .addFilterBefore(securityFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
    }

    @Bean
    public AuthenticationManager authenticationManager( ... )

    @Bean
    public PasswordEncoder passwordEncoder() { ...

    }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfigurations {

    @Autowired
    private SecurityFilter securityFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        return http.csrf(csrf -> csrf.disable())
            .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            // configurar autorizações para as requisições:
            .authorizeHttpRequests(req -> {
                // liberar o /login
                // req.requestMatchers(antMatcher("/login")).permitAll();
                req.requestMatchers("/login").permitAll();
                // todas as outras, só para usuários autenticados
                req.anyRequest().authenticated();
            })
            // definindo a ordem de aplicação dos filtros:
            // primeiro, o nosso filtro (classe SecurityFilter injetada aqui)
            // depois, o filtro de usuarios autenticados do proprio spring
            .addFilterBefore(securityFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
    }

    @Bean
    public AuthenticationManager authenticationManager( ... )

    @Bean
    public PasswordEncoder passwordEncoder() { ...

    }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfigurations {

    @Autowired
    private SecurityFilter securityFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http.csrf(csrf -> csrf.disable())
            .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            // configurar autorizações para as requisições:
            .authorizeHttpRequests(req -> {
                // liberar o /login
                // req.requestMatchers(antMatcher("/login")).permitAll();
                req.requestMatchers("/login").permitAll();
                // todas as outras, só para usuários autenticados
                req.anyRequest().authenticated();
            })
            // definindo a ordem de aplicação dos filtros:
            // primeiro, o nosso filtro (classe SecurityFilter injetada aqui)
            // depois, o filtro de usuários autenticados do próprio spring
            .addFilterBefore(securityFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
    }

    @Bean
    public AuthenticationManager authenticationManager( ... )

    @Bean
    public PasswordEncoder passwordEncoder() { ... }

}
```

Alterações na classe SecurityFilter

Antes lançava exceção se não viesse JWT,
porém /login não terá JWT!!!

Mudanças:

Se vier JWT, retira aquele 'Bearer'
Valida, e pega login (email)
Carrega usuário do BD
Autentica este usuário

Lembrando que não há sessões,
o usuário é autenticado EM CADA REQUISIÇÃO

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto Authentication que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto Authentication que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto Authentication que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```



```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto Authentication que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto Authentication que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```



```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto Authentication que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```



```
@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                                                                        usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```

```
@Component
public class SecurityFilter extends OncePerRequestFilter {

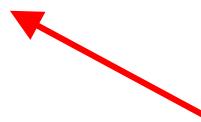
    @Autowired
    private PW3TokenService pw3tokenservice;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Recuperando o cabeçalho Authorization (o token JWT):
        var authorizationHeader = request.getHeader("Authorization");

        // Lembrando: /login não envia cabeçalho authorization!
        // Se o JWT foi enviado...
        if (authorizationHeader != null) {
            // Tira o 'Bearer':
            authorizationHeader = authorizationHeader.replace("Bearer ", "");
            // Chamar o método que valida o token, e recupera o login:
            var subject = pw3tokenservice.getSubject(authorizationHeader);
            // Carrega o usuário com esse login do BD:
            var usuario = repository.findByLogin(subject);
            // Cria um objeto que representa a identidade de um usuário autenticado.
            // Ele contém as informações do usuário, como o nome de usuário, a senha e as permissões.
            var authentication = new UsernamePasswordAuthenticationToken(usuario, null,
                usuario.getAuthorities());
            // Finalmente, manda o Spring AUTENTICAR esse usuário:
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // Para executar o próximo filtro, ou seguir o processamento:
        filterChain.doFilter(request, response);
    }
}
```



Está pronto!!!

Testando



Listagem geral de médicos, sem enviar token

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** localhost:8080/medicos
- Auth tab selected:** Bearer Token
- Type:** Bearer Token
- Token:** Token (empty)
- Description:** The authorization header will be automatically generated when you send the request.
- Body tab selected:** Pretty
- Response Status:** 403 Forbidden
- Response Time:** 67 ms
- Response Size:** 300 B
- Actions:** Save as example, Text (selected), Copy, and Search.

A red arrow points from the "Text" button in the bottom navigation bar to the "Text" dropdown menu in the response summary area.

Listagem geral de médicos, enviando token inválido

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** localhost:8080/medicos
- Auth tab selected:** Bearer Token
- Token value:** dfgdgdfgfdgfd
- Body tab selected:** Pretty
- Response status:** 403 Forbidden
- Response time:** 27 ms
- Response size:** 300 B

A red arrow points from the "Text" button in the bottom toolbar to the "403 Forbidden" status code.

```
2023-11-13T19:45:27.200-03:00  INFO 11075 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].I  
2023-11-13T19:45:27.200-03:00  INFO 11075 --- [nio-8080-exec-2] o.s.web.servlet.Disp  
2023-11-13T19:45:27.201-03:00  INFO 11075 --- [nio-8080-exec-2] o.s.web.servlet.Disp  
2023-11-13T19:46:30.911-03:00 ERROR 11075 --- [nio-8080-exec-5] o.a.c.c.C.[.][/].[c
```

java.lang.RuntimeException Create breakpoint : Token JWT inválido ou expirado!

```
    at br.edu.ifsp.pw3.api.util.security.PW3TokenService.getSubject(PW3TokenService.  
    at br.edu.ifsp.pw3.api.util.security.SecurityFilter.doFilterInternal(SecurityFil  
    at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFi  
    at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter  
    at org.springframework.security.web.authentication.logout.LogoutFilter.doFilter(LogoutF
```

Version Control Run TODO Problems Terminal Services Auto-build Build Dependencies

All files are up-to-date (2 minutes ago)

Fazendo login, senha errada

The screenshot shows the Postman interface for a POST request to `localhost:8080/login`. The request body is a JSON object with fields `login` and `senha`. The response status is `403 Forbidden`.

Request Details:

- Method: POST
- URL: localhost:8080/login
- Headers: (9)
- Body (JSON):

```
1 {  
2   "login": "asdrubal@gmail.com",  
3   "senha": "123456xxxxx"  
4 }  
5
```

Response Details:

- Status: 403 Forbidden
- Time: 604 ms
- Size: 300 B
- Save as example

A red arrow points to the `Text` button in the response preview toolbar.

Fazendo login, senha correta

The screenshot shows the Postman interface for a POST request to `localhost:8080/login`.

Body (JSON):

```
1 {  
2   "login": "asdrubal@gmail.com",  
3   "senha": "123456"  
4 }  
5
```

Response Body (Pretty):

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpc3Mi0iJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsL  
mNvbSIsImV4cCI6MTY50TkyMjkzNH0.  
Npy7X1TLhUR3u3p-KBfNkNIN-My5DPR1vo39-xN67YE"  
3 }
```

Listagem geral, usando o token criado no login

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** localhost:8080/medicos
- Auth:** Bearer Token (selected)
- Type:** Bearer Token
- Token:** eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9eyJpc3MiOiJESVNDSVBMSU5BIFBXMylsInN1Yil6ImFzZHJ1YmFsQGdtYWlsLmNvbSIslmV4cCI6MTY5OTkyMjkzNH0.Npy7X1TLhUR3u3p-KBfNkNIN-My5DPRIvo39-xN67YE
- Body:** 200 OK, 80 ms, 1.15 KB
- Body Content (Pretty):**

```
1 [  
2 {  
3     "id": 1,  
4     "nome": "Asdrubal Almeida",  
5     "email": "asd@gmail.com",  
6     "telefone": "16992481111",  
7     "crm": "111111",  
8     "especialidade": "ORTOPEDIA",
```

Dados de um médico, sem token

GET ▼ localhost:8080/medicos/2 Send ▼

Params Auth Headers (7) Body Pre-req. Tests Settings ...

Query Params

	Key	Value	Descrip...	...	Bulk Edit
	Key	Value	Description		

Body ▼ 🌐 403 Forbidden 5 ms 300 B 💾 Save as example ...

Pretty Raw Preview Visualize Text ▼ 🔗 ✖ 🔍

1

Dados de um médico, com token

The screenshot shows a Postman request configuration for a GET request to `localhost:8080/medicos/2`. The **Auth** tab is selected, indicating the use of a token for authentication. The **Type** dropdown is set to **Bearer Token**, and the token value is displayed as a long string of characters: `eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIslmV4cCI6MTY5OTkyMjkzNH0.Npy7X1TLhUR3u3p-KBfNkNIN-My5DPRlvo39-xN67YE`. The response status is **200 OK** with **24 ms** latency and **600 B** size. The response body is displayed in **Pretty** format:

```
1 {  
2   "id": 2,  
3   "nome": "Lupita Almeida",  
4   "email": "lupy@gmail.com",  
5   "crm": "222222",  
6   "telefone": "16992482222",  
7   "especialidade": "DERMATOLOGIA",  
8   "endereco": "  
  123 Rua das Flores  
  Centro  
  São Paulo - SP  
  CEP: 01234-000"  
}
```

Dados de um médico, com token, 2 horas depois

The screenshot shows a Postman API request configuration. The method is set to GET, the URL is localhost:8080/medicos/2, and the 'Auth' tab is selected. Under the 'Auth' tab, the 'Type' dropdown is set to 'Bearer Token'. A large text box labeled 'Token' contains a long, encoded string of characters: eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJpc3MiOiJESVNDSVBMSU5BIFBXMyIsInN1YiI6ImFzZHJ1YmFsQGdtYWlsLmNvbSIslmV4cCI6MTY5OTkyMjkzNH0.Npy7X1TLhUR3u3p-KBfNkNIN-My5DPRlvo39-xN67YE. Below this, a note states: 'The authorization header will be automatically generated when you send the request'.

Below the auth section, the 'Body' dropdown is set to 'Text'. The response details show a status of 403 Forbidden, 6 ms duration, and 300 B size. The response body is empty, indicated by the number '1'.

```
m1_0.uf,  
m1_0.especialidade,  
m1_0.nome,  
m1_0.telefone  
from  
medicos m1_0  
where  
m1_0.id=?
```

2023-11-13T21:54:03.766-03:00 ERROR 11075 --- [nio-8080-exec-9] o.a.c.c.C.[.[]].

java.lang.RuntimeException Create breakpoint : Token JWT inválido ou expirado!
at br.edu.ifsp.pw3.api.util.security.PW3TokenService.getSubject(PW3TokenService)
at br.edu.ifsp.pw3.api.util.security.SecurityFilter.doFilterInternal(SecurityFilter)
at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter)
at org.springframework.security.web.FilterChainProxy\$VirtualFilterChain.doFilter(FilterChainProxy)
at org.springframework.security.web.authentication.logout.LogoutFilter.doFilter(LogoutFilter)

Version Control Run TODO Problems Terminal Services Auto-build Build Dependencies

All files are up-to-date (36 minutes ago)

Criar medico, sem token:

The screenshot shows the Postman application interface. At the top, there is a header bar with a 'POST' method, a dropdown, the URL 'localhost:8080/medicos', another dropdown, and a large blue 'Send' button. Below the header, there are tabs for 'Params', 'Auth' (which is currently selected and underlined), 'Headers (8)', 'Body', 'Scripts', 'Tests', 'Settings', and 'Cookies'. Under the 'Auth' tab, a dropdown menu shows 'No Auth'. The main body of the interface displays a red '403 Forbidden' status bar with the message 'No Auth'. Below this, there are buttons for 'Body' (with a dropdown arrow), a refresh icon, and a 'Raw' dropdown set to 'Raw'. To the right of the status bar, there are performance metrics: '6 ms', '300 B', and a globe icon. Further right are buttons for 'Save Response', three dots, and icons for copy, search, and link. At the bottom left, the number '1' is displayed.

Criar medico, com token:

POST localhost:8080/medicos Send

Params Auth • Headers (9) Body • Scripts Tests Settings Cookies

Auth Type
Bearer Token Token  

The authorization header

Body  201 Created • 100 ms • 687 B •  e.g. Save Response ...

{ } JSON ▾ Preview Visualize |     

```
1 {  
2   "id": 9,  
3   "nome": "Filho do Asdrubal Zoroastro",  
4   "email": "ffasd@gmail.com",  
5   "crm": "777767",  
6   "telefone": "16999993344",  
7   "especialidade": "DERMATOLOGIA",  
8   "endereco": {  
9     "logradouro": "Rua Danilo Micalli 50",  
10    "bairro": "Sao Carlos 3",  
11    "cep": "13563264".
```



Projeto finalizado disponível no github.

https://github.com/carlaopereirasanca/prw3_api_2025_2

Exercícios



Exercícios

- Experimente com os códigos vistos na aula de hoje...



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

Avaliação 4



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

Avaliações

■ Avaliação 1

- Prática - Conteúdo: JPA.
- Vale **15%** da média.

■ Avaliação 2

- Escrita - Conteúdo: parte teórica (api root, roteo, http, arquit. spring).
- Vale **10%** da média.

■ Avaliação 3

- Prática - Conteúdo: API Root, CRUD, com Spring.
- A nota final vale **30%** da média da disciplina.

■ Avaliação 4

- Prática - Conteúdo: dotar a api criada na **av.3** de autenticação (com Spring Security).
- Vale **45%** da média.

Calendário da reta final!

- 29.out Desenvolvimento de uma API REST (parte 4). Introdução Spring Security
- 5.nov Conteúdo extra: Spring Data.
- 12.nov Spring Security parte 2.
- 19.nov Spring Security parte 3.
- 26.nov Dia para trab. na Avaliação 4. **Entrega até 23:59.**
- 3.dez Demonstração: Microservices com Spring (vídeo-aula, s/ controle presença)
- 10.dez Revisão. Preparação para IFA...



Avaliação 4

- Retome o projeto da oficina mecânica...



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

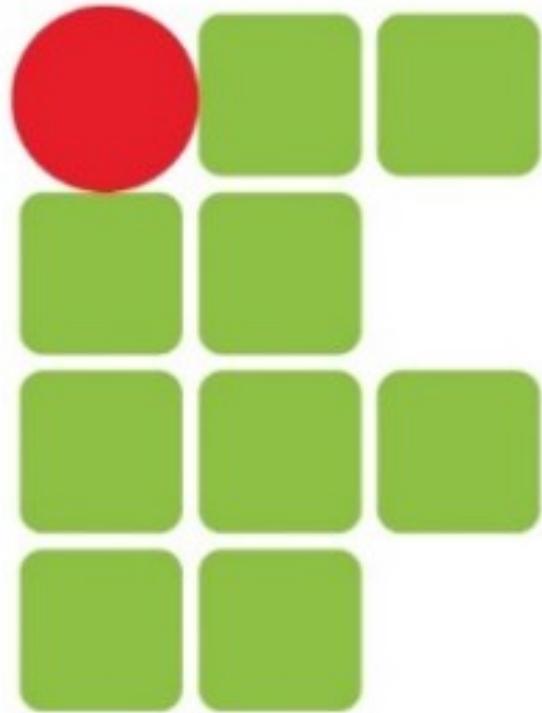
Avaliação 4

- Retome o projeto da oficina mecânica
- Implemente segurança via token, como visto nas últimas 3 aulas
 - Simples assim...





INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos



**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos**



**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos**