

Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 01 – SPRING SECURITY

Spring Security **Autenticação e Autorização**



Spring

- O Spring contém um módulo específico para tratar de segurança, conhecido como Spring Security.
- O Spring Security, é um módulo dedicado para tratarmos das questões relacionadas com segurança em aplicações.
- Vamos implementar o mecanismo de autenticação e autorização da aplicação.
- Em suma, o Spring Security possui três objetivos.
- Um deles é providenciar um serviço para customizarmos como será o controle de autenticação no projeto. Isto é, como os usuários efetuam login na aplicação.

Spring

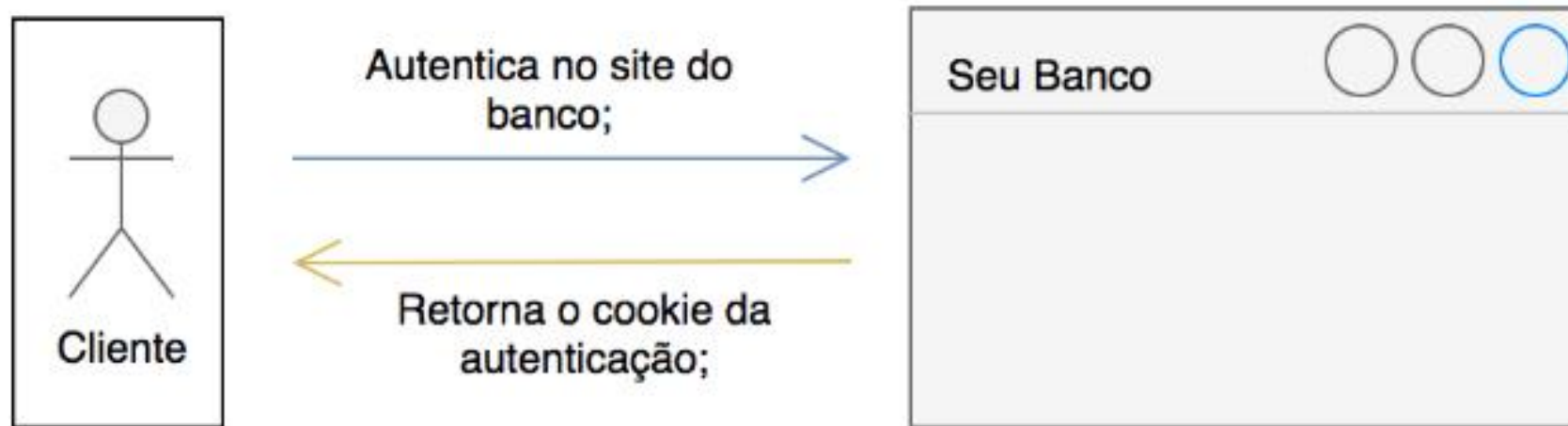
- Os usuários deverão preencher um formulário?
- É autenticação via token?
- Usa algum protocolo?
- Assim, ele possui uma maior flexibilidade para lidar com diversas possibilidades de aplicar um controle de autenticação.

Spring

- O Spring Security possui, também, **a autorização**, sendo o controle de acesso para liberarmos a requisição na API ou para fazermos um controle de **permissão**.
- Por exemplo, temos esses usuários e eles possuem a permissão "A", já estes usuários possuem a permissão "B".
- Os usuários com a permissão "A" podem acessar as URLs, os que tiverem a permissão "B", além dessas URLs, podem acessar outras URLs.
- Com isso, **conseguimos fazer um controle do acesso ao nosso sistema**.

- Há, também, um **mecanismo de proteção** contra os principais ataques que ocorre em uma aplicação, **como o CSRF** (Cross Site Request Forgery) – Falsificação de Requisição Cruzada.
- Este ataque serve para o invasor fazer movimentações financeiras ou transações online, alterar senhas de acesso do usuário vítima, alterar dados pessoais da vítima.

- 1 - Você loga no site do seu banco usando o seu navegador preferido:



- A credencial do usuário é validada, um cookie é enviado na resposta da requisição HTTP. A partir desse momento, o navegador tem salvo no disco o cookie que te mantém autenticado.

Spring

- 2 - Você recebe um e-mail (engenharia social) que te convence a clicar em um link que abre um site arbitrário
- 3 - Ao entrar nesse site, no corpo dele, tem um formulário, que irá pegar o cookie da autorização, nele tem as suas credencias.
- 4 - Esse formulário no site do atacante emula exatamente como o seu banco faz para realizar uma transferência de dinheiro.
- 5 - Em seguida, em algum momento, esse site arbitrário submete o formulário usando JavaScript:
- 6 - É realizado a transferência.

Spring

- e o **clickjacking**. (“Furto do Clique” - O invasor consegue infectar botões de um site legítimo, com isso o roubo de senhas de rede sociais, Internet Banking)
- São esses os três principais objetivos do Spring Security, nos **fornecer uma ferramenta para implementarmos** **autenticação** e **autorização** no projeto e nos **proteger** dos principais ataques.
- Isso **para não precisarmos implementar** o código que protege a aplicação, sendo que já temos disponível.
- No caso da nossa API, o que faremos é o **controle de autenticação** e **autorização**, o **controle de acesso**.

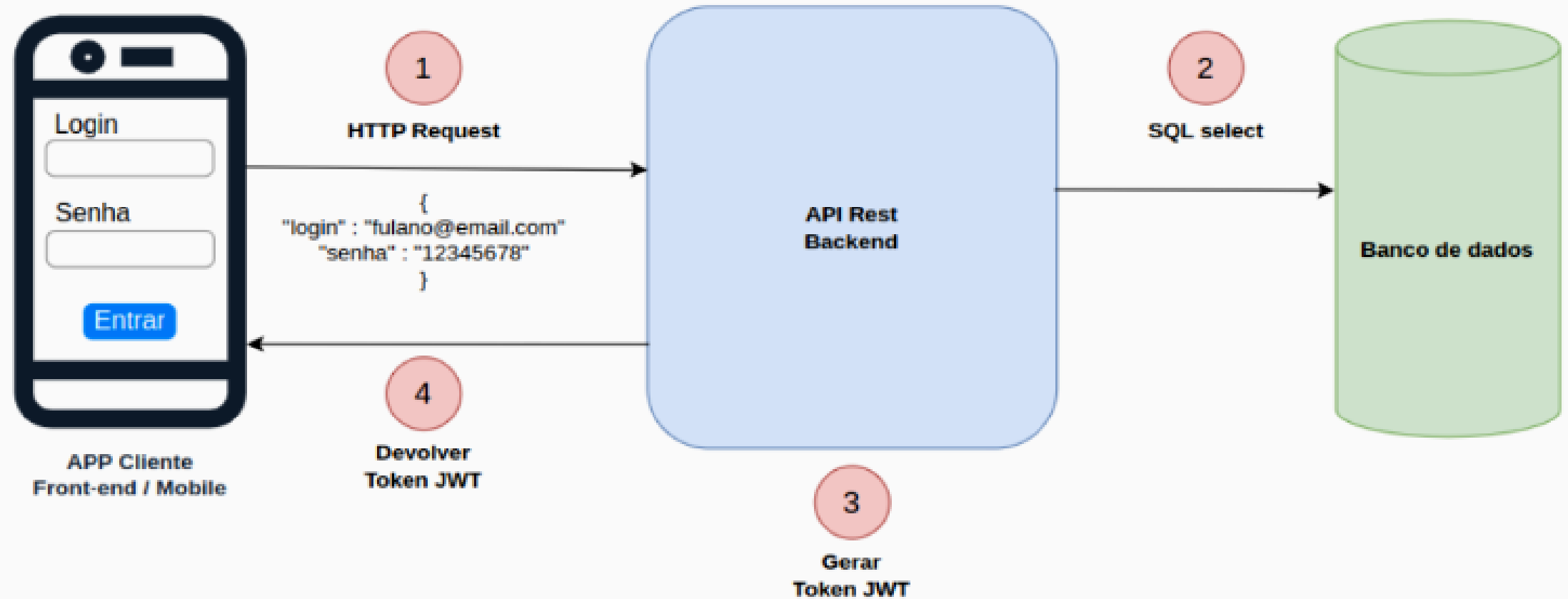
Spring

- Até o momento, não nos preocupamos com essas questões de segurança. Logo, a nossa API está pública.
- Isso significa que qualquer pessoa que tiver acesso ao endereço da API, consegue disparar requisições, assim como fizemos usando o Insomnia.
- Sabemos o endereço da API, no caso é localhost:8080 - dado que está rodando local na máquina.
- Porém, após efetuarmos o deploy da aplicação em um servidor, seria o mesmo cenário: caso alguém descubra a endereço, conseguirá enviar requisição para cadastrar um médico ou paciente, etc.

Autenticação em API Rest (Stateless)

- Não é criada a sessão, os dados do usuário não ficam em memória.
- Como será o processo de autenticação em uma API?
- Temos **diversas estratégias** para **lidarmos com a autenticação**.
- Uma delas é usando **Tokens**, e usaremos o JWT - JSON Web Tokens como protocolo padrão para lidar com o gerenciamento desses tokens - **geração e armazenamento de informações nos tokens**.

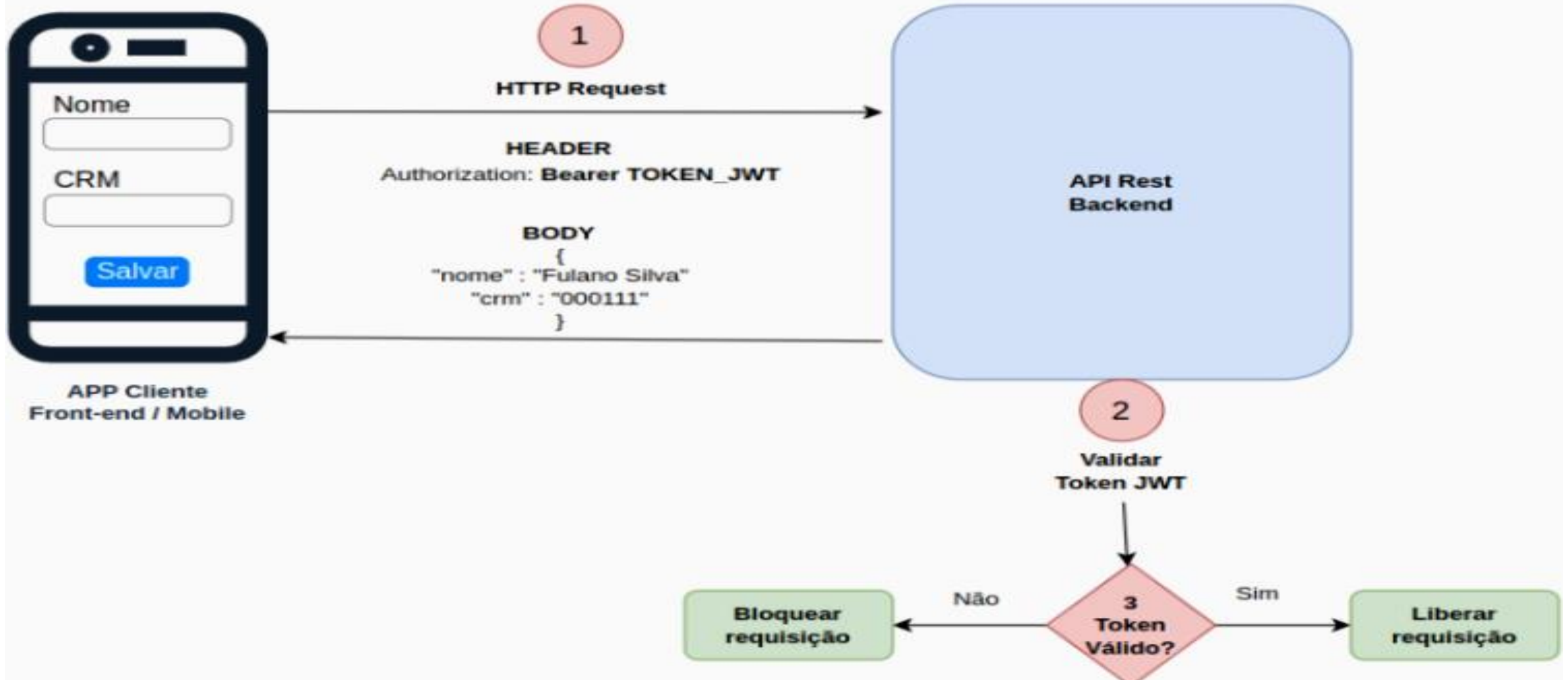
Autenticação



Spring

- Esse é o processo de uma requisição para efetuar o login e autenticar em uma API Rest, usando tokens. Será esse processo que seguiremos neste curso.
- Isto é, teremos um controller mapeando a URL de autenticação, receberemos um DTO com os dados do login e faremos uma consulta no banco de dados. Se tiver tudo certo, geramos um token e devolvemos para o front-end, para o cliente que disparou a requisição.
- Esse token deve ser armazenado pelo aplicativo mobile/front-end. Há técnicas para guardar isso de forma segura, porque esse token que identifica se o usuário está logado.
- Assim, nas requisições seguintes entra o processo de autorização, que consta no diagrama a seguir:

Autorização



Spring

- Portanto, o processo de autorização é: primeiro, chega uma requisição na API e ela lê o cabeçalho `authorization`, captura o token enviado e valida se foi gerado pela API. Teremos um código para verificar a validade do token.
- Caso não seja válido, a requisição é interrompida ou bloqueada.
- Não chamamos o controller para salvar os dados do médico no banco de dados, devolvemos um erro 403 (é um código de resposta HTTP que indica que o servidor entendeu a sua solicitação, mas recusa-se a autorizá-la) ou 401 (o servidor de um site não autorizou uma solicitação porque não tem credenciais de autenticação válidas).

Spring

- Há protocolos HTTP específicos para esse cenário de autenticação e autorização.
- Pelo fato do token estar vindo, o usuário já está logado.
- Portanto, o usuário foi logado previamente e recebeu o token.
- Este token informa se o login foi efetuado ou não.
- Caso seja válido, seguimos com o fluxo da requisição.

Spring

- Assim, nas próximas requisições o aplicativo leva na requisição, além dos dados em si, o token.
- Logo, não é necessário mandar login e senha, somente o token.
- E nesta string, contém a informação de quem é esse usuário e, com isso, a API consegue recuperar esses dados.
- Essa é uma das formas de fazer a autenticação em uma API Rest.

Spring

- Como o servidor sabe se **estamos logados** ou **não**?
- O cliente precisa enviar alguma coisa para não precisarmos enviar login e senha em toda requisição.
- Ele informa o login e a senha na requisição de logar, **recebe um token** e **nas próximas ele direciona esse mesmo token**.
- Nas próximas aulas adicionaremos o Spring Security, e implementar esse conceito de autenticação e autorização analisando cada detalhe no Spring Boot.
- <https://www.alura.com.br/artigos/tipos-de-autenticacao>

Spring

Adicionando o Spring Security



Spring

- Vamos acessar o site do [Spring Initializr](#).
- Uma vez aberto, vamos adicionar a dependência do **Spring Security**.

Project

☐ Gradle - Groovy

☐ Gradle - Kotlin ☒ **Maven**

Language

☒ **Java** ☐ Kotlin

☐ Groovy

Java ☐ 22 ☒ **21** ☐ 17

- No lado direito superior clique em

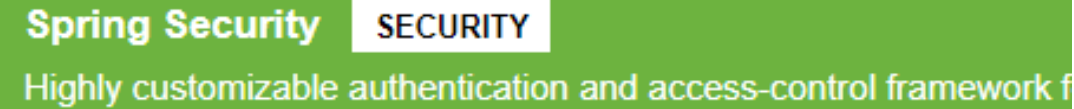
Dependencies

ADD DEPENDENCIES... CTRL + B

Spring

- Na caixa de texto digite **Security**.

- Selecione a opção.



Spring Security SECURITY
Highly customizable authentication and access-control framework f

- A dependência já foi adicionada em "Dependencies".

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Security SECURITY

Highly customizable authentication and access-control framework for
Spring applications.

- Após isso, **clicaremos no botão "Explore"** na parte inferior central.



EXPLORE CTRL + SPACE

Spring

- Note que foi incluída duas dependências: o `spring-boot-starter-security` e o `spring-security-test`.

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

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

Spring

- No ao IntelliJ, abra o pom.xml.
- Localize o MySQL.... Posicione o curso entre os <dependency> Aperte "Enter" para dar um espaço e coloque as dependências.

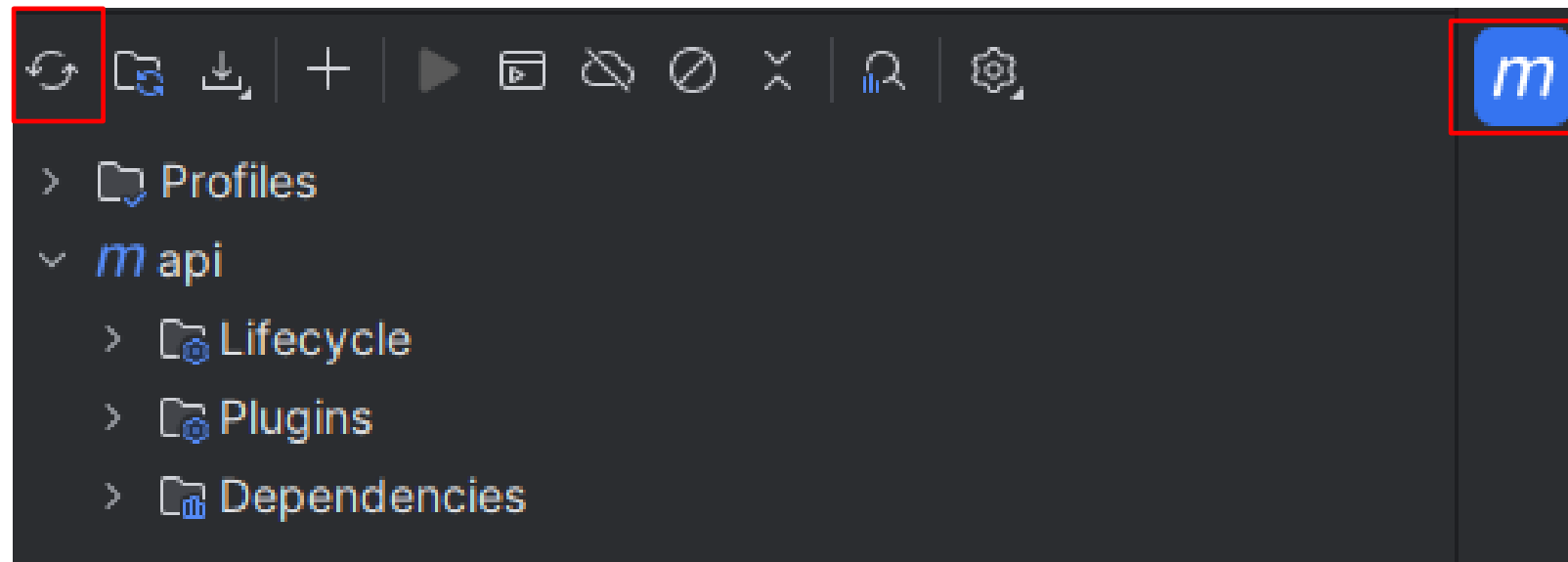


```
</dependency>  
<dependency>  
    <groupId>com.mysql</groupId>
```

- Podemos salvar o arquivo pom.xml.

Spring

- Caso você esteja rodando a aplicação o ideal é [parar a aplicação](#).
- Para garantir que as dependências do Maven foram baixadas com sucesso, selecionaremos "Maven" no canto superior direito, escrito verticalmente.
- Será exibida uma aba, em que clicaremos no primeiro ícone de reload "↺" no canto superior esquerdo, para recarregarmos todos os projetos do Maven.



Spring

- Agora vamos iniciar o projeto.
- Analisando os logs, perceba que no final foi gerado uma linha que antes não era exibida no retorno.



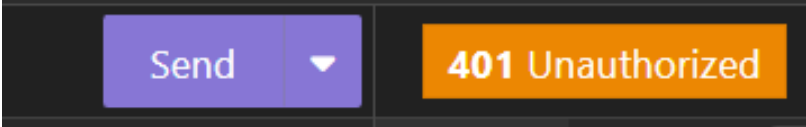
```
Using generated security password: 9806bdc3-a960-44b1-b3a3-45e3f5cd1e0b
```

- Logo em seguida essa informação

```
This generated password is for development use only.
```

- Ao adicionarmos o Spring Security como dependência do projeto, ele gera uma configuração padrão, criando um usuário e senha e exibindo no log.
- E, também, faz uma alteração no projeto, bloqueando todas as requisições.

Spring

- Logo em seguida essa informação.
- Abra o Insomnia para disparar a requisição de listagem de médicos.
- Clique na parte inferior 
-
- Com o Insomnia aberto selecione a requisição de médicos e digite o seguinte endereço: 
- Depois a faça a requisição. 
- Foi tentado realizar um requisição, mas está tudo bloqueado!!! Porquê???
- Pois não foi realizado o login

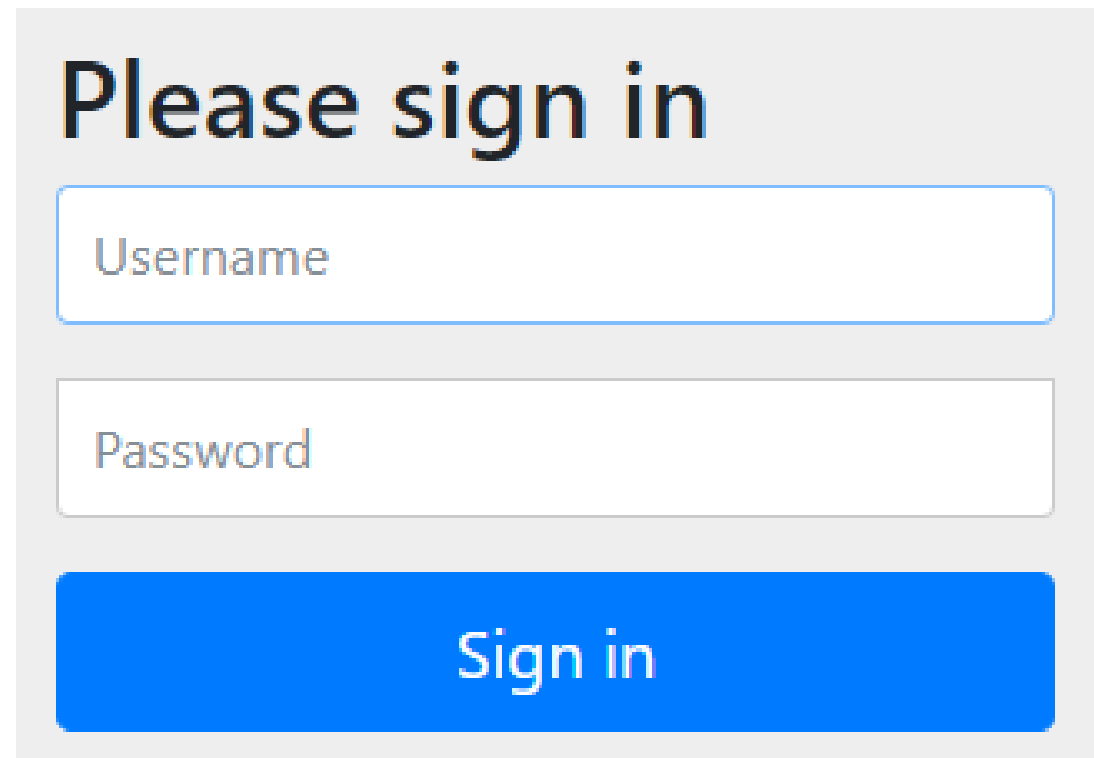
Spring

- Todas as requisições estão sendo bloqueadas pelo Spring Security.
- Portanto, esse é o comportamento padrão do Spring Security ao adicioná-lo como dependência no projeto, ele bloqueia tudo e gera uma senha aleatória.
- Onde podemos usar essa senha gerada?
- Vamos disparar a requisição pelo navegador, agora.
- Por ele conseguimos analisar melhor.

`http://localhost:8080/medicos`

Spring

- Ao clicarmos na tecla "Enter" é exibida uma tela de Login, com a frase "Please sign in" ("Por favor, inscreva-se") e os campos `username` e `password`. Abaixo, um botão azul "Sign in", centralizado.

A screenshot of a login form. At the top, the text "Please sign in" is displayed in a large, bold, black font. Below this, there are two input fields: the first is labeled "Username" and the second is labeled "Password", both in a smaller, gray font. At the bottom of the form is a large, blue button with the text "Sign in" in white, centered on it. The entire form is set against a light gray background.

Please sign in

Username

Password

Sign in

Spring

- Somos redirecionados para esse formulário, sendo do próprio Spring Security.
- Assim, esse é o padrão: bloquear todas as URLs e se tentarmos acessar do navegador, ele bloqueia e nos redireciona para o formulário de login.
- Para efetuar esse login, vamos voltar ao IntelliJ e copiar a senha gerada no console, usando o atalho "Ctrl + C".
- Após copiarmos a senha, colaremos no campo password do formulário.
- Por padrão, o Spring gera um usuário chamado user, no campo username digitamos "user".

Entidade Usuário e Migration

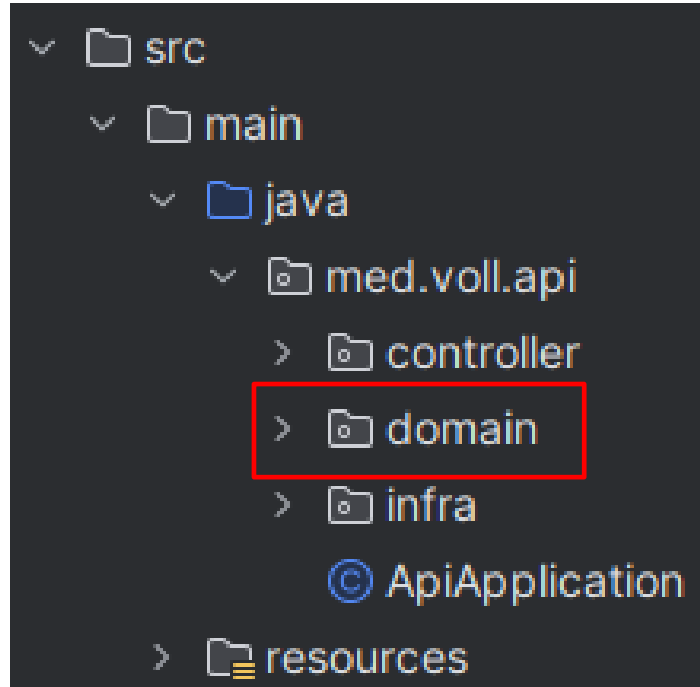


Spring

- Para autenticar o usuário na API, precisamos ter uma tabela no banco de dados em que serão armazenados os usuários e suas respectivas senhas.
- Essa é uma das modificações que vamos fazer no projeto.
- Além da tabela de médicos e pacientes, incluiremos uma nova tabela de usuário.
- Precisamos representar esse conceito de usuário no código, como estamos usando a JPA, criaremos uma entidade JPA para simbolizar esse usuário e temos que gerar a tabela no banco de dados.
- Faremos isso usando as migrations já existentes no projeto.

Spring

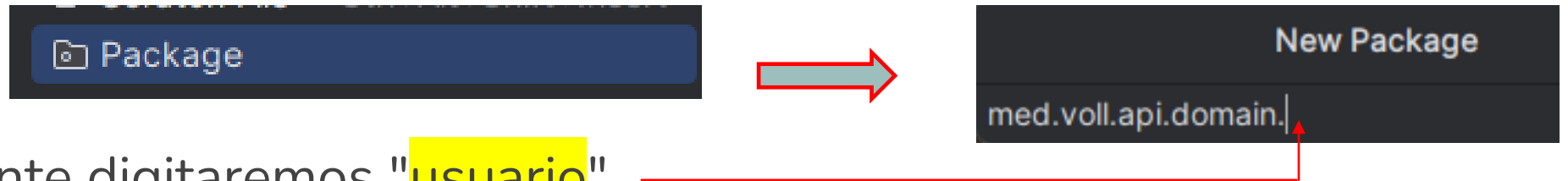
- Para criarmos a entidade JPA, vamos em "src > main > java > med.voll.api > domain", na pasta domain ficam os dados relacionados ao domínio da aplicação.



- para criar um pacote para guardar essas classes e conceitos referentes ao usuário.

Spring

- Agora, criaremos um pacote para guardar essas classes e conceitos referentes ao usuário.
- Para isso, vamos selecionar o pacote domain, e depois usamos o atalho "Alt + Insert".
- Será exibida uma aba com algumas opções, dentre elas escolheremos a "Package"
- E na aba seguinte digitaremos "usuario".
- Assim, ficamos com o seguinte pacote: med.voll.api.domain.usuario.



Spring

- Agora, dentro do pacote usuario criaremos uma classe.
- Com esse objetivo, selecionamos o pacote usuario e depois as teclas "Alt + Insert".
- Na aba exibida, vamos clicar na opção "Java Class".
- Após isso, será mostrado um pop-up em que digitaremos o nome da classe, no caso será somente "Usuario".
- Podemos selecionar "Enter" novamente, para criar.

Spring

- Agora, dentro do pacote usuario criaremos uma classe.
- Com esse objetivo, selecionamos o pacote usuario e depois as teclas "Alt + Insert".
- Na aba exibida, vamos clicar na opção "Java Class".
- Após isso, será mostrado um pop-up em que digitaremos o nome da classe, no caso será somente "Usuario".
- Podemos selecionar "Enter" novamente, para criar.

```
package med.voll.api.domain.usuario;  
no usages  
public class Usuario {  
}
```

Spring

- Para a classe Usuário teremos 3 atributos.

```
public class Usuario {  
    no usages  
    private Long id;  
    no usages  
    private String login;  
    no usages  
    private String senha;  
}
```

Spring

- São esses os atributos necessários para o objeto usuário.
- Agora, vamos inserir as **anotações do JPA** e **do Lombok**, para gerarmos os **getters** e **setters**, **construtores**, entre outras coisas.
- Para facilitar, **clicaremos no arquivo Medico** que se encontra em "src > main > java > med.voll.api > domain > medico".
- Nele, copiaremos as anotações nas linhas iniciais:
- Coloque na classe Usuário.

```
@Table(name = "medicos")
@Entity(name = "Medico")
@Getter
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(of = "id")
public class Medico {
```

Spring

- A Classe Usuário.

```
@Table(name = "usuarios")
@Entity(name = "Usuario")
@Getter
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(of = "id")
public class Usuario {
    private Long id;
    private String login;
    private String senha;
}
```

Spring

- Agora informe que o **Id do usuário** será uma **Pk e auto incremental**

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;
```

- Com isso mapeamos a entidade JPA.
- Essa **entidade** está representando **uma tabela no banco de dados** e, **por isso**, **essa tabela precisa ser criada no banco de dados**.
- Portanto, **criaremos uma migration para fazer** a **evolução do schema do banco de dados**.

Spring

- Toda vez que precisarmos **alterar algum dado no banco de dados** (**excluir coluna, incluir linha, etc**), **geramos uma nova migration.**

ATENÇÃO: sempre interrompa o projeto ao codificar as migrations!

- Essas são as migration já existentes em nosso projeto.

- `db.migration`
 - `V1__create-table-medicos.sql`
 - `V2__alter-table-medicos-add-column-telefone.sql`
 - `V3__alter-table-medicos-add-column-ativo.sql`
 - `V4__create-table-pacientes.sql`

Spring

- Vamos criar uma **quinta migration**.
- Vamos em "src > main > resources > **db.migration**". Note que temos quatro pastas.
- Seleccionaremos a primeira migration **V1__create-table-medicos.sql**, e clicaremos nas teclas "**Ctrl + C**" e depois em "**Ctrl + V**".
- **Será aberta um pop-up** copy, com os campos "**new name**" e "**to directory**". No campo "New name" **alteraremos** para **V5__create-table-usuarios.sql**.



New name: **V1__create-table-medicos.sql**

- Abra o arquivo recém criado. Precisamos alterar esse código, para....

```
create table usuarios(  
  
    id bigint not null auto_increment,  
    login varchar(100) not null,  
    senha varchar(255) not null,  
  
    primary key(id)  
);
```

Spring

- Vamos rodar o projeto, o flyway detecta a nova migration e executar esse script no banco de dados.
- Será exibida a aba "Run", procuraremos por algo que nos informe que uma nova versão subiu.

Migrating schema 'vollmedapi' to version "5 - create-table-usuarios"

- Isso significa que a tabela de usuários foi criada!
- Veja se há uma nova tabela em nosso BD.
- Com isso, concluímos o primeiro passo: criamos uma entidade JPA simbolizando o usuário e depois geramos uma migration no projeto.

Spring

Repository e Service



Spring

- Agora, criaremos uma **interface repository**.
- Em geral, cada entidade JPA terá uma interface repository, sendo ela que irá fazer o acesso ao banco de dados.
- Para criarmos o repository do usuário, vamos em "src > main > java > med.voll.api > domain > usuario".
- Com o pacote usuario selecionado, usaremos o atalho "Alt + Insert", e na aba exibida escolheremos a opção "Java Class".
- Na aba "New Java Class" digitaremos o nome "**UsuarioRepository**" e deixaremos a segunda opção "**Interface**" marcada. Logo após, clicaremos na tecla "Enter", para criar a interface.

Spring

```
package med.voll.api.domain.usuario;  
  
no usages  
  
public interface UsuarioRepository {  
}
```

- Por ser um repository precisamos herdar, por isso, acrescentaremos **extends JpaRepository**, após o nome da interface.
- Sendo **JpaRepository** a interface do Spring Data.

Spring

```
public interface UsuarioRepository  
    extends JpaRepository {  
  
}
```

- Logo após a interface `JpaRepository`, vamos abrir e fechar um sinal de menor que (" $<$ ") e outro sinal de maior que (" $>$ "), os generics.
- Dentro dos generics passaremos os dois tipos: quem é a entidade e o tipo de chave primária.
- No caso, a entidade é o `Usuario` e o tipo da chave é `Long`, para representar os IDs.

Spring

- Sempre que precisarmos consultar a tabela de usuários no banco de dados, usaremos o `UsuarioRepository`.
- Além disso, vamos criar algumas classes de configurações para o Spring Security.
- Nessa parte de autenticação, o Spring Security detecta algumas coisas de forma automática no projeto.
- Por exemplo, a classe que vai ter a lógica de usar o repository e acessar o banco de dados para realizar a consulta.

Spring

- O Spring possui um comportamento padrão, ele procura por uma classe específica no projeto.
- Portanto, precisamos criar essa classe seguindo o padrão do Spring e, com isso, ele consegue identificá-la no projeto e usá-la para fazer o processo de autenticação.
- Para isso funcionar, vamos criar uma classe dentro do pacote usuario.
- Com a pasta usuario selecionada, utilizamos o atalho "Alt + Insert", e depois escolhemos a opção "Java Class".
- No pop-up seguinte, digitamos o nome "AutenticacaoService" e selecionamos "Enter".

Spring

```
package med.voll.api.domain.usuario;  
  
no usages  
  
public class AutenticacaoService {  
    }  
}
```

- Esta classe terá o código com a lógica de autenticação no projeto.
- Até o momento é uma classe Java tradicional, não contém nenhuma anotação e, portanto, o Spring não a reconhece.
- Para o Spring identificar essa classe, usamos a anotação `@Service` em cima da classe.

Spring

- A anotação `@Service` serve para o Spring identificar essa classe como um componente do tipo serviço.
- Isto é, ela executará algum serviço no projeto, no caso, será o de autenticação.
- Assim, o Spring carrega a classe e executa o serviço ao inicializarmos o projeto.

```
@Service
public class AutenticacaoService {
}
```

Spring

- O Spring Security também precisa ser informado que esta classe é a responsável pelo serviço de autenticação.
- Para isso, teremos que implementar uma interface.
- Portanto, na assinatura da classe, antes de abrir as chaves vamos digitar `implements UserDetailsService`.
- Sendo `UserDetailsService` uma interface do Spring Security.

```
@Service
public class AutenticacaoService
    implements UserDetailsService {
}
```

Spring

- Perceba que está dando **erro de compilação** na linha da classe, está sublinhado na cor vermelha.
- Isso acontece porque ao implementarmos uma interface, **precisamos implementar os métodos dessa interface**.
- Clicando na interface **UserDetailsService**, e usando o atalho "**Atl + Enter**", será exibido um pop-up com algumas opções.
- Dentre elas, escolheremos a "**Implement methods**" ("Implementar métodos").

Spring

```
UserDetailsService {
```



Implement methods

Class 'AutenticacaoService' must either
'UserDetailsService'

Implement methods Alt+Shift+Enter

- Ou passe o mouse no erro, o Spring dirá o que deve ser implementado.

```
public class AutenticacaoService
    implements UserDetailsService {

    no usages

    @Override
    public UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException {
        return null;
    }
}
```

Spring

- Precisamos detalhar somente um método na classe dessa interface.
- No caso, o `loadUserByUsername`, sendo o método que o Spring chama de forma automática ao efetuarmos o login.
- Com isso, quando o usuário efetuar o login, o Spring busca pela classe `AutenticacaoService` - por ser a responsável por implementar a `UserDetailsService` - e por sua vez chama o método `loadUserByUsername`, passando o `username` digitado no formulário de login.
- Em suma, essa é a lógica deste método.

Spring

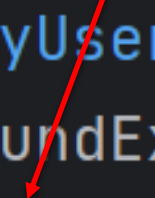
- Note que no método ele está retornando null.
- Precisamos devolver alguma coisa, no caso, precisamos acessar o banco de dados, para isso temos que usar o repository.
- Logo, vamos injetar o repository como uma dependência da classe.
- Para isso, acima do método loadUserByUsername vamos declarar o atributo private UsuarioRepository nomeando como repository.

```
public class AutenticacaoService
    implements UserDetailsService {
    no usages
    @Autowired
    private UsuarioRepository repository;
```


Spring

- Agora, no método `loadUserByUsername` vamos alterar o retorno de `null` para `repository`.
- Na sequência colocamos um ponto `(".")`, e incluímos o método `findByLogin()` (nome do nosso atributo que representa o login), passando como parâmetro o `username`.
- Teremos um erro de compilação em `findByLogin`, porque não existe esse método na interface `UsuarioRepository`.

```
public UserDetails loadUserByUsername(String username)
    throws UsernameNotFoundException {
    return repository.findByLogin(username);
}
```



Spring

- Para ajustar isso, selecionamos o método, usamos o atalho "Alt + Enter" ou passe o "mouse" e escolhemos a opção "Create method 'findByLogin' in 'UsuarioRepository'".
- Após clicarmos nesta opção

Create method 'findByLogin' in 'UsuarioRepository'

- seremos redirecionados para o arquivo:

```
public interface UsuarioRepository
    extends JpaRepository <Usuario, Long> {
    1 usage
    UserDetails findByLogin(String username);
}
```

Spring

- Foi criado na nossa interface UsuarioRepository com o método findByLogin().

```
UserDetails findByLogin(String username);
```

- Neste método, vamos alterar o parâmetro de username para login.

```
UserDetails findByLogin(String login);
```

- O findByLogin() é o método responsável por realizar a consulta do usuário no banco de dados.
- Portanto, será usado em AutenticacaoService.

Spring

- Assim, criamos a entidade JPA que representa o usuário;
- Temos a migration que gerou a tabela no banco de dados;
- A interface repository responsável pelo acesso na tabela de usuários;
- A classe AutenticacaoService que simboliza o serviço de autenticação, será chamada pelo Spring automaticamente quando efetuarmos a autenticação no projeto.
- Na próxima aula mais configurações de Segurança

Spring

