01 Projeto da aula anterior

Caso queira, você pode baixar aqui o projeto do curso no ponto em que paramos na aula anterior.

02 Adicionando a lib auth0 jwt

Transcrição

A requisição de *login* chega na classe `AutenticacaoController.java`. Nela, criamos o método `efetuarLogin`, recebendo o *DTO* `DadosAutenticacao`.

Usamos, além disso, as classes do *Spring Security* para disparar o processo de autenticação. O *DTO* do *Spring Security* é `UsernamePasswordAuthenticationToken`. Neles, passamos o *login* e a senha que chegam ao *DTO*.

Usamos, também, a classe `AuthenticationManager`, do *Spring Security*, para disparar o processo de autenticação.

Nosso foco da aula será ter o *token* como retorno no *Insomnia*.

De volta à classe `AutenticacaoController.java`, vamos remover `.build()` de `return` e adicionar o *token* entre os parênteses do parâmetro `.ok`

Vamos adicionar a biblioteca *Auth0* ao projeto. Ela será utilizada para gerar o *token*, seguindo o padrão *JWT*.

Para pegarmos a biblioteca, acessaremos o site https://jwt.io/.
Clicaremos na segunda opção do menu superior do site, "Libraries".
Lá, encontraremos uma lista com várias bibliotecas que geram *tokens* no padrão *JWT*.
À direita da tela, na parte superior, encontramos uma *combo box* que pode ser usada para filtrar as linguagens de programação. Nela, selecionaremos "Java". Serão exibidos todos os projetos que geram tokens para projetos *JWT*.

Selecionaremos a primeira, a biblioteca em *Java* para gerar *tokens* em *JWT* do *Auth0*. Vamos clicar no link "View Repo", no canto inferior direito. Com isso, seremos redirecionados para o repositório da biblioteca no *Github*.

Para instalar a biblioteca, vamos levar uma dependência para o *Maven*. Vamos copiar a *tag* de *dependency* abaixo da seção "Installation"

Obs: No momento de gravação do vídeo, a biblioteca está na versão 4.2.1. Recomendamos que você a utilize, para que consigamos fazer tudo que o instrutor faz no treinamento.

Vamos adicioná-la ao nosso projeto, pela *IDE*. Vamos parar o servidor antes de prosseguir. Depois disso, vamos acessar o arquivo "pom.xml". Abaixo da última dependência, passaremos o código abaixo:

```
<dependency>
```

```
        <groupId>com.auth0</groupId>
        <artifactId>java-jwt</artifactId>
        <version>4.2.1</version>
</dependency> COPIAR CÓDIGO
```

A biblioteca que fará a geração dos *tokens* foi importada com sucesso.

Obs: É preciso inserir o trecho de código entre as tags `<dependencies>`.

Depois, no *Maven*, clicaremos no botão de *Reload*

03 Para saber mais: JSON Web Token

JSON Web Token, ou JWT, é um padrão utilizado para a geração de *tokens*, que nada mais são do que Strings, representando, de maneira segura, informações que serão compartilhadas entre dois sistemas. Você pode conhecer melhor sobre esse padrão em seu site oficial.

Aqui na Alura temos o artigo O que é JSON Web Tokens? e o Alura+ O que é Json Web Token (JWT)?, que também explicam o funcionamento do padrão JWT.

04 Gerando tokens JWT

Transcrição

Agora faremos a geração do token para inclui-lo na resposta.

Faremos isso criando uma nova classe no projeto, para que possamos isolar o token, uma boa prática em programação. Vamos acessar "infra > security". É nessa pasta que criaremos o token.

Apertaremos "Alt + Insert" e selecionaremos a primeira opção, "Java Class". O nome dela será "TokenService".

Ela fará a geração, a validação e o que mais estiver relacionado aos tokens. No arquivo "TokenService.java", passaremos a anotação `@Service`, já que a classe representará um serviço.

Dentro disso, declararemos o método `public`, com `String` como retorno, que representa o token a ser gerado. O nome do método será `gerarToken`. Dentro dela, usaremos a biblioteca que adicionamos ao projeto na aula anterior.

Vamos copiar o trecho de código da seção "Create a JWT", para gerar nosso token. Vamos copiá-lo e colá-lo dentro do método `gerarToken`, fazendo algumas alterações.

A mais importante delas será a substituição do algoritmo padrão por `HMAC256`. Como parâmetro dela, passaremos a senha `12345678`.

Em breve, aprenderemos a ocultar a senha, para que ela não fique exposta em código aberto.

Obs: O ideal é que os tokens da API tenham data de validade.

Vamos gerar a validade chamando o método `.withExpiresAt()`, passando como parâmetro `dataExpiracao()`. Precisamos criar esse método privado clicando em "Alt + Enter".

Substituiremos `Date` por `Instant` e configuraremos o tempo de expiração:

```
import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTCreationException;
import org.springframework.stereotype.Service;

@Service
public class TokenService {

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256("12345678");
            return JWT.create()
                .withIssuer("API Voll.med")
                .withSubject(usuario.getLogin())
                .withExpiresAt(dataExpiracao())
                .sign(algoritmo);
        } catch (JWTCreationException exception){
```

```
            throw new RuntimeException("erro ao gerrar token jwt",
exception);
        }
    }

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

Agora o token estará criado.

Vamos acessar "AutenticacaoController.java" e, logo abaixo do atributo `AuthenticationManager`, passaremos `@Autowired`, e, logo abaixo, `private TokenService`.

E também `Response.Entity.ok(tokenService.gerarToken((usuario) authentication.getPrincipal()));`:

```
@Autowired
private TokenService tokenService;

@PostMapping
public ResponseEntity efetuarLogin(@RequestBody @Valid
DadosAutenticacao dados) {
    var token = new UsernamePasswordAuthenticationToken(dados.login(),
dados.senha());
    var authentication :Authentication = manager.authenticate(token);
```

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

Se voltarmos a executar a aplicação e dispararmos a requisição de login no Insomnia, teremos o tolken devolvido com sucesso.

05 Ajustes na geração do token

Transcrição

Vamos copiar o *token* que foi devolvido como resultado no *Insomnia*. De volta ao navegador, acessaremos novamente o https://jwt.io.

No site, acessaremos a seção "Debugger", na qual colaremos o *token* na caixa de texto abaixo de "Encoded". Em "Decoded", receberemos o *token* dissecado. Assim, descobriremos as informações que estão presentes nele.

A parte vermelha é o cabeçalho, onde descobrimos o algoritmo utilizado para fazer a geração do *token*, que foi o *HS256*.

A parte roxa tem as informações adicionadas dentro do *token*, *issuer*, que é "API Voll.med", a data de expiração, programada para duas horas à frente, e o *subject*, onde passamos o login do usuário que se autenticou.

Vamos voltar à IDE e acessar a classe "AutenticacaoController.java". Vamos criar um *DTO* para encapsular o *token* e não devolvê-lo solto, como fazemos no corpo da resposta.

Vamos selecionar `tokenService.gerarToekn((Usuario) authentication.getPrincipal())`. Com um "Ctrl + X", levaremos essa linha de código para antes da linha do *return*, onde criaremos outra variável, `var tokenJWT =`. Depois do `=`, colaremos a linha de código que antes estava abaixo.

Passaremos como parâmetro do `return` `(new DadosTokenJWT(tokenJWT))`. Como a classe `DadosTokenJWT` ainda não foi criada, vamos nos deparar com um erro de compilação.

Criaremos, portanto, com "Alt + Enter > Create Record DadosTokenJWT". Trocaremos o "Destination package" (pacote de destino)
de `med.voll.api.controller` para `med.voll.api.infra.security`.
Obs: No *DTO*, substitua `(String tokenJWT)` por `(String token)`.

Agora nosso *controller* chama a classe `tokenService`, responsável por gerar o *token*, recebe o *token* de volta e devolve isso em um *DTO*:

```
    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid
DadosAutenticacao dados) {
        var authenticationToken = new
UsernamePasswordAuthenticationToken(dados.login(), dados.senha());
```

```
        var authentication =
manager.authenticate(authenticationToken);

        var tokenJWT = tokenService.gerarToken((Usuario)
authentication.getPrincipal());

        return ResponseEntity.ok(new DadosTokenJWT(tokenJWT));
    } COPIAR CÓDIGO
```

Vamos voltar ao *Insomnia* para testar. Antes, quando disparávamos a requisição, recebíamos o *token* solto como resposta. Agora, ao dispararmos, perceberemos que ele virá dentro de um *JSON*, com um campo chamando "token" e a *string* representando o *token*.

Agora voltaremos à classe "TokenService.java" na IDE. Nela, precisamos passar uma senha secreta na linha de criação do algoritmo, o que é indispensável para fazer a assinatura do *token*.

Nas aulas anteriores, havíamos passado "12345678" como senha. Como passar a senha em texto dentro do código não é uma boa prática de segurança, vamos fazer a leitura dessa senha de algum lugar.

O primeiro passo será remover o "12345678" do código. No lugar dela, passaremos um atributo chamando `secret`. Vamos declarar o atributo dentro da classe `TokenService`, com a linha de código `private String secret;`.

Com o atalho "Shift + Shift", buscaremos por "application.properties". Nele, criaremos uma nova propriedade: `api.security.token.secret=`. Depois do `=`, pediremos

para que o *Spring* leia essa informação a partir de uma variável de ambiente.

Para isso, passaremos `api.security.token.secret=${JWT_SECRET}`.
Logo depois, passaremos, ainda dentro das chaves, `12345678`.

Com isso, caso o sistema não consiga acessar a variável de ambiente, ele utilizará "12345678" como senha secreta.

Se dispararmos a requisição no *Insomnia*, receberemos o erro 500 como retorno. O erro aconteceu porque, em "TokenService.java", nós declaramos o atributo `secret` mas não falamos para o *Spring* que ele deve buscá-lo em "application.properties".
Na linha acima de `private String secret;`, passaremos a anotação `@Value`.

Obs: Cuidado ao importar! Há o *Value* do *Lombok* e o value do *Spring Framework*. O que nos interessa é o segundo.

Entre aspas, como parâmetro, passaremos `"${api.security.token.secret}"`.

Para garantir que a leitura está sendo feita da maneira correta, vamos até o método `gerarToken`. Antes do `try/catch` passaremos `System.out.println(secret);`:

```
    @Value("${api.security.token.secret}")
    private String secret;


public String gerarToken(Usuario usuario) {
```

```
        System.out.println(secret); COPIAR CÓDIGO
```

Salvando o projeto e voltando ao *Insomnia*, vamos disparar a requisição. O *token* será gerado e, de volta à IDE, em "TokenService.java", veremos que foi impressa a senha "12345678", o que significa que a leitura foi feita corretamente na classe "TokenService.java", no atributo `secret`.

Como tudo deu certo, podemos remover o `System.out.println(secret);`.

Conseguimos fazer a geração correta do *token*.

06 Para saber mais: Outras informações no token

Além do Issuer, Subject e data de expiração, podemos incluir outras informações no token JWT, de acordo com as necessidades da aplicação. Por exemplo, podemos incluir o *id* do usuário no token, para isso basta utilizar o método `withClaim`:

```
return JWT.create()
    .withIssuer("API Voll.med")
    .withSubject(usuario.getLogin())

    .withClaim("id", usuario.getId())

    .withExpiresAt(dataExpiracao())
    .sign(algoritmo); COPIAR CÓDIGO
```

O método `withClaim` recebe dois parâmetros, sendo o primeiro uma String que identifica o nome do claim (propriedade armazenada no token), e o segundo a informação que se deseja armazenar.

07 Injeção de propriedades

Vimos ao longo dessa aula que podemos injetar uma propriedade declarada no arquivo `application.properties` em uma classe gerenciada pelo Spring, com a utilização da anotação `@Value`. Supondo que o arquivo `application.properties` tenha a seguinte propriedade declarada:

```
app.teste=true
COPIAR CÓDIGO
```

Qual a maneira **CORRETA** de injetá-la em um atributo de uma classe gerenciada pelo Spring?

- `@Value("${app.teste}")`
- Esse é o jeito correto de utilizar a anotação `@Value`.

- Alternativa correta

- `@Value("{app.teste}")`

- Alternativa correta

- `@Value("app.teste")`

Parabéns, você acertou!

08 Faça como eu fiz: geração de Tokens

Agora é com você! Faça o mesmo procedimento que eu fiz na aula, implementando a geração de ***tokens JWT*** quando um usuário se autenticar na API.

Opinião do instrutor

Primeiramente, você precisará adicionar a biblioteca ***Auth0 java-jwt*** no projeto, incluindo essa dependência no `pom.xml`:

```
<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>4.2.1</version>
</dependency> COPIAR CÓDIGO
```

Na sequência, será necessário criar a classe responsável pela geração dos *tokens*:

```
@Service
public class TokenService {

    @Value("${api.security.token.secret}")
    private String secret;

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256(secret);
            return JWT.create()
                    .withIssuer("API Voll.med")
                    .withSubject(usuario.getLogin())
                    .withExpiresAt(dataExpiracao())
                    .sign(algoritmo);
        } 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"));
    }

} COPIAR CÓDIGO
```

Você também vai precisar adicionar a seguinte propriedade no arquivo `application.properties`:

```
api.security.token.secret=${JWT_SECRET:12345678} COPIAR CÓDIGO
```

Por fim, será necessário criar o DTO `DadosTokenJWT` e alterar a classe `AutenticacaoController`:

```
public record DadosTokenJWT(String token) {}COPIAR CÓDIGO
@RestController
@RequestMapping("/login")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager manager;

    @Autowired
```

```
    private TokenService tokenService;


    @PostMapping
    public ResponseEntity efetuarLogin(@RequestBody @Valid
DadosAutenticacao dados) {
        var authenticationToken = new
UsernamePasswordAuthenticationToken(dados.login(), dados.senha());
        var authentication =
manager.authenticate(authenticationToken);


        var tokenJWT = tokenService.gerarToken((Usuario)
authentication.getPrincipal());


        return ResponseEntity.ok(new DadosTokenJWT(tokenJWT));
    }


} COPIAR CÓDIGO
```

09 O que aprendemos?

Nessa aula, você aprendeu como:

- Adicionar a biblioteca `Auth0 java-jwt` como dependência do projeto;
- Utilizar essa biblioteca para realizar a geração de um *token* na API;
- Injetar uma propriedade do arquivo `application.properties` em uma classe gerenciada pelo Spring, utilizando a anotação `@Value`;
- Devolver um *token* gerado na API quando um usuário se autenticar nela.