Aula 16: Encapsulando o Token JWT e Configurando a Chave Secreta

September 18, 2025

1 Introdução

Na Aula 15, criamos a classe `TokenService` para gerar tokens JWT e atualizamos o `AutenticacaoController` para retorná-los. Nesta aula, encapsularemos o token em um DTO (`DadosTokenJWT`) para uma resposta mais estruturada e configuraremos a leitura da chave secreta do JWT a partir do arquivo `application.properties` usando uma variável de ambiente. Também testaremos o endpoint no Insomnia para verificar o token encapsulado. As alterações serão feitas no `Visual Studio Code` ou sua IDE preferida.

2 Verificando o Token JWT

No Insomnia, copiamos o token retornado pela requisição `POST http://localhost:8080/login`. No site `https://jwt.io/`, na seção "Debugger", colamos o token na caixa "Encoded". A seção "Decoded" exibe:

- **Header** (vermelho): Algoritmo `HS256` (HMAC256).
- **Payload** (roxo): Informações como `iss` ("API Voll.med"), `sub` (login do usuário, ex.: `usuario@voll.med`), e `exp` (data de expiração, 2 horas à frente).

3 Criando o DTO `DadosTokenJWT`

Para encapsular o token na resposta, criamos o record `DadosTokenJWT` no pacote `med.voll.api.infra.securi`

```
1 package med.voll.api.infra.security;
2
3 public record DadosTokenJWT(String token) {
4 }
```

3.1 Explicação do Código

- `String token`: Campo que armazena o token JWT gerado.

4 Atualizando o `AutenticacaoController`

No `AutenticacaoController` (pacote `med.voll.api.controller`), atualizamos o método `efetuarLogin` para retornar o token encapsulado no DTO:

```
1 package med.voll.api.controller;
2
3 import jakarta.validation.Valid;
4 import med.voll.api.domain.usuario.DadosAutenticacao;
5 import med.voll.api.domain.usuario.Usuario;
```

```java
import med.voll.api.infra.security.DadosTokenJWT;
import med.voll.api.infra.security.TokenService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import
    org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@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));
    }
}
```

4.1 Explicação do Código

- `var tokenJWT = tokenService.gerarToken(...)`: Gera o token JWT usando o `TokenService`.
- `new DadosTokenJWT(tokenJWT)`: Encapsula o token no DTO.
- `ResponseEntity.ok(...)`: Retorna o DTO com status 200 (OK).

5 Configurando a Chave Secreta no `TokenService`

Na classe `TokenService` (pacote `med.voll.api.infra.security`), substituímos a chave secreta hardcoded (“12345678”) por uma propriedade lida do `application.properties`:

```java
package med.voll.api.infra.security;

import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTCreationException;
import med.voll.api.domain.usuario.Usuario;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;

@Service
public class TokenService {
```

```
15
16      @Value("${api.security.token.secret}")
17      private String secret;
18
19      public String gerarToken(Usuario usuario) {
20          try {
21              var algoritmo = Algorithm.HMAC256(secret);
22              return JWT.create()
23                      .withIssuer("API Voll.med")
24                      .withSubject(usuario.getLogin())
25                      .withExpiresAt(dataExpiracao())
26                      .sign(algoritmo);
27          } catch (JWTCreationException exception) {
28              throw new RuntimeException("Erro ao gerar token JWT", exception);
29          }
30      }
31
32      private Instant dataExpiracao() {
33          return LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
34      }
35  }
```

5.1 Explicação do Código

- `@Value("$api.security.token.secret")`: Lê a propriedade `api.security.token.secret` do `application.properties`.
- `private String secret`: Armazena a chave secreta.
- `Algorithm.HMAC256(secret)`: Usa a chave secreta lida para assinar o token.

6 Configurando o `application.properties`

No arquivo `src/main/resources/application.properties`, adicionamos a propriedade para a chave secreta:

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

6.1 Explicação

- `api.security.token.secret`: Define a chave secreta, lida da variável de ambiente $\texttt{JWT}_{SECRET}$.12345678 : $Valorpadrocasoavariveldeambientenoestejadefinida.$

7 Testando o Endpoint de Login

No Insomnia, testamos a requisição `POST http://localhost:8080/login` com o JSON:

```
1  {
2      "login": "usuario@voll.med",
3      "senha": "123456"
4  }
```

7.1 Resultado

- **Antes**: O token era retornado como uma string solta (ex.: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...`
- **Agora**: Retorna código 200 (OK) com o token encapsulado em um JSON:

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Se a leitura da propriedade falhar, um erro 500 (Internal Server Error) será retornado. Para verificar a leitura da chave secreta, adicionamos temporariamente um `System.out.println(secret)` no método `gerarToken`, confirmando que o valor "12345678" é lido corretamente.

8 Próximos Passos

Com o token JWT encapsulado e a chave secreta configurada via propriedades, o próximo passo é implementar um filtro no `Spring Security` para validar o token em requisições protegidas. Na próxima aula, configuraremos a validação de tokens e protegeremos os endpoints da API. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

9 Dica do Professor

- **Aprofunde-se em Configurações**: Consulte a documentação do Spring sobre propriedades (`https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config`) para entender variáveis de ambiente.
- **Comunidade no X**: Siga perfis como @SpringConfig e @SecurityPro no X para dicas sobre configurações seguras e JWT.

- **Pratique**: Configure a variável de ambiente `JWT`$_{S}ECRETnosistemaoperacionalounaIDEetestearequisionoInsomn$