Aula 13: Implementando o Controlador de Autenticação

September 18, 2025

1 Introdução

Na Aula 12, configuramos o `Spring Security` para autenticação **stateless**, desabilitando o formulário de login padrão e a proteção CSRF. Nesta aula, implementaremos o processo de autenticação, criando o controlador `AutenticacaoController`, o DTO `DadosAutenticacao`, e configurando o `AuthenticationManager` e o algoritmo de hash BCrypt. Também ajustaremos a entidade `Usuario` para implementar `UserDetails` e testaremos o endpoint de login no Insomnia. As alterações serão feitas no `Visual Studio Code` ou sua IDE preferida.

2 Revisão do Processo de Autenticação

O diagrama de autenticação (visto na Aula 8) descreve o fluxo:

- O cliente (ex.: aplicativo mobile) envia uma requisição `POST /login` com um JSON contendo `login` e `senha`.
- A API valida as credenciais no banco de dados.
- Se válidas, a API gera um token JWT (a ser implementado na próxima aula).
- O token é retornado ao cliente.

3 Criando o DTO `DadosAutenticacao`

Criamos o record `DadosAutenticacao` no pacote `med.voll.api.domain.usuario` para representar o JSON enviado pelo cliente:

```
1 package med.voll.api.domain.usuario;
2
3 public record DadosAutenticacao(String login, String senha) {
4 }
```

3.1 Explicação do Código

- `String login`: Representa o e-mail do usuário (ex.: `usuario@voll.med`).
- `String senha`: Representa a senha do usuário, que será comparada com o hash armazenado no banco.

4 Criando o AutenticacaoController

No pacote med.voll.api.controller, criamos o controlador AutenticacaoController para processar requisições de login:

```java
package med.voll.api.controller;

import jakarta.validation.Valid;
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;
import med.voll.api.domain.usuario.DadosAutenticacao;

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

    @Autowired
    private AuthenticationManager manager;

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

4.1 Explicação do Código

- `@RestController` e `@RequestMapping("/login")`: Define o controlador para processar requisições na URL `/login`.
- `@Autowired AuthenticationManager manager`: Injeta o gerenciador de autenticação do Spring.
- `@PostMapping`: Mapeia requisições POST para o método `efetuarLogin`.
- `@RequestBody @Valid DadosAutenticacao dados`: Recebe o JSON com login e senha, validado pelo Bean Validation.
- `UsernamePasswordAuthenticationToken`: Converte o DTO em um token do Spring Security.
- `manager.authenticate(token)`: Dispara a autenticação, chamando `AutenticacaoService`.
- `ResponseEntity.ok().build()`: Retorna código 200 (OK), mas ainda sem o token JWT.

5 Configurando o AuthenticationManager

O AuthenticationManager não é injetado automaticamente, então adicionamos sua configuração na classe SecurityConfigurations (pacote med.voll.api.infra.security):

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

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
    org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.context.annotation.Bean;

@Configuration
@EnableWebSecurity
public class SecurityConfigurations {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
        Exception {
        return http.csrf().disable()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and().build();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration
        configuration) throws Exception {
        return configuration.getAuthenticationManager();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

5.1 Explicação do Código

- `@Bean AuthenticationManager`: Configura o `AuthenticationManager` para injeção no controlador.
- `@Bean PasswordEncoder`: Define o algoritmo BCrypt para hash de senhas.

6 Ajustando a Entidade `Usuario`

Para corrigir o erro de autenticação ("Invalid property 'accountNonLocked'"), a entidade `Usuario` (pacote `med.voll.api.domain.usuario`) deve implementar `UserDetails`:

```java
package med.voll.api.domain.usuario;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;
```

```java
import java.util.List;

@Table(name = "usuarios")
@Entity(name = "Usuario")
@Getter
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(of = "id")
public class Usuario implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String login;
    private String senha;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority("ROLE_USER"));
    }

    @Override
    public String getPassword() {
        return senha;
    }

    @Override
    public String getUsername() {
        return login;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

6.1 Explicação do Código

- `implements UserDetails`: Integra a entidade com o `Spring Security`.
- `getAuthorities()`: Retorna uma lista com o perfil $\texttt{ROLE}_USER, indicandoquenohcontroledepermisses.\texttt{getPa}$ $Mapeiamoscampos\texttt{senha}e\texttt{login}.$
- `isAccountNonExpired()`, `isAccountNonLocked()`, `isCredentialsNonExpired()`, `isEnabled()`: Retornam `true`, indicando que a conta está ativa e sem restrições.

7 Inserindo um Usuário no Banco

Para testar a autenticação, inserimos um usuário na tabela `usuarios` usando MySQL no terminal:

```
insert into usuarios values (1, 'usuario@voll.med',
    '$2a$10$Y50UaMFOxteibQEYLrwuHeehHYfcoafCopUazP12.rqB41bsolF5.');
```

7.1 Explicação do Comando

- `id`: 1 (auto-incrementado).
- `login`: `usuario@voll.med`.
- `senha`: Hash BCrypt da senha 123456.

8 Testando o Endpoint de Login

No Insomnia, criamos uma requisição `POST http://localhost:8080/login` com o JSON:

```
{
    "login": "usuario@voll.med",
    "senha": "123456"
}
```

8.1 Resultados

- **Senha incorreta** (12345678): Retorna 403 (Forbidden) devido à falha na autenticação.
- **Senha correta** (123456): Retorna 200 (OK), mas sem corpo na resposta, pois o token JWT ainda não foi implementado.

O log do Hibernate no console confirma a consulta ao banco:

```
select
    v1_0.id,
    v1_0.login,
    v1_0.senha
from
    usuarios v1_0
where
    v1_0.login=?
```

9 Próximos Passos

Concluímos a configuração básica da autenticação, mas falta retornar o token JWT no `AutenticacaoControlle`
Na próxima aula, implementaremos a geração de tokens JWT para completar o processo de autenticação. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

10 Dica do Professor

- **Aprofunde-se em Spring Security**: Consulte a documentação sobre `AuthenticationManager` (`https://docs.spring.io/spring-security/reference/features/authentication/architecture.html`) para entender o processo de autenticação.

- **Comunidade no X**: Siga perfis como @SpringSecPro e @JWTAuth no X para dicas sobre autenticação e JWT.
- **Pratique**: Insira outro usuário na tabela `usuarios` com um hash BCrypt gerado (use ferramentas como `https://www.browserling.com/tools/bcrypt`) e teste o login no Insomnia.