

Spring Boot com Angular-Ionic

Capítulo: Autenticação e Autorização com tokens JWT

Objetivo geral:

- 🕒 Compreender o mecanismo de funcionamento do Spring Security
- 🕒 Implementar autenticação e autorização com JWT
- 🕒 Controlar conteúdo e acesso aos endpoints

Implementação básica (incompleta) de referência:

<https://auth0.com/blog/implementing-jwt-authentication-on-spring-boot/>

Configuração inicial do Spring Security

Checklist:

- 🕒 Incluir as dependências no pom.xml
- 🕒 Criar uma classe de configuração SecurityConfig para definir as configurações de segurança
 - Esta classe deve herdar de WebSecurityConfigurerAdapter
 - Definir as configurações básicas das URL's que necessitam ou não de autenticação/autorização

Notas:

1) De modo geral pode-se desabilitar proteção a CSRF em sistemas stateless

2) <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/cors/CorsConfiguration.html>

"By default a newly created `CorsConfiguration` does not permit any cross-origin requests and must be configured explicitly to indicate what should be allowed."

3) <https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/config/annotation/web/builders/HttpSecurity.html>

```
public CorsConfigurer<HttpSecurity> cors() throws Exception
```

"Adds a `CorsFilter` to be used. If a bean by the name of `corsFilter` is provided, that `CorsFilter` is used. Else if `corsConfigurationSource` is defined, then that `CorsConfiguration` is used. Otherwise, if Spring MVC is on the classpath a `HandlerMappingIntrospector` is used."

Adicionando senha a Cliente

Checklist:

- 🕒 Incluir um Bean de BCryptPasswordEncoder no arquivo de configuração
- 🕒 Incluir um atributo senha na classe Cliente
- 🕒 Atualizar ClienteNewDTO
- 🕒 Atualizar ClienteService
- 🕒 Atualizar DBService

Salvando perfis de usuário na base de dados

Checklist:

- 🕒 Criar o tipo enumerado Perfil (CLIENTE, ADMIN)
- 🕒 Implementar na classe Cliente:
 - Um atributo correspondente aos perfis do usuário a serem armazenados na base de dados
 - Usar @ElementCollection(fetch=FetchType.EAGER)
 - Disponibilizar os métodos:
 - public Set<Perfil> getPerfis()
 - public void addPerfil(Perfil perfil)
- 🕒 Incluir perfil padrão (CLIENTE) na instanciação de Cliente
- 🕒 Incluir mais um cliente com perfil ADMIN na carga inicial da base de dados (DBService)

Implementando autenticação e geração do token JWT

ATUALIZAÇÃO

Se você criou o projeto usando Spring Boot versão 2.x.x:

-

- Na classe JWTAuthenticationFilter, acrescentar uma implementação de AuthenticationFailureHandler e injetá-la em JWTAuthenticationFilter:

-

Checklist:

- 🕒 Criar classe de usuário conforme contrato do Spring Security (implements UserDetails)
- 🕒 Criar classe de serviço conforme contrato do Spring Security (implements UserDetailsService)
- 🕒 Em SecurityConfig, sobrescrever o método: public void configure(AuthenticationManagerBuilder auth)
- 🕒 Criar a classe CredenciaisDTO (usuário e senha)
- 🕒 Incluir as propriedades de JWT (segredo e tempo de expiração) em application.properties
- 🕒 Criar uma classe JWTUtil (@Component) com a operação String generateToken(String username)
- 🕒 Criar um filtro de autenticação
- 🕒 Em SecurityConfig, registrar o filtro de autenticação

Implementando autorização

Checklist:

- ⌚ Criar um filtro de autorização
 - Acrescentar uma função em JWTUtil para verificar se um dado token é válido
- ⌚ Em SecurityConfig, registrar o filtro de autorização

Autorizando endpoints para perfis específicos

Checklist:

- ⌚ Em SecurityConfig: `@EnableGlobalMethodSecurity(prePostEnabled = true)`
- ⌚ Nos métodos dos resources: `@PreAuthorize("hasAnyRole('ADMIN')")`

Restrição de conteúdo: cliente só recupera ele mesmo

- ⌚ IMPORTANTE: Criar um UserService com um método que retorna o usuário logado

Checklist:

- ⌚ Criar uma exceção personalizada de autorização para a camada de serviço
- ⌚ Em ClienteService, implementar a verificação: se o cliente logado não for ADMIN e não for o cliente do id solicitado, lançar uma exceção
- ⌚ Tratar a exceção na camada resource

Restrição de conteúdo: cliente só recupera seus pedidos

ATUALIZAÇÃO

Se você criou o projeto usando Spring Boot versão 2.x.x:

-

- Na classe PedidoService, usar PageRequest.of e ClienteService ao invés de ClienteRepository:

-

Checklist:

- ⌚ Criar a consulta na camada de acesso a dados
- ⌚ Criar a consulta na camada de serviço
- ⌚ Criar o endpoint

Refresh token

- ⌚ Criar um novo resource AuthResource com caminho /auth
- ⌚ Criar o endpoint (POST):

```
@RequestMapping(value="/refresh_token", method=RequestMethod.POST)
public ResponseEntity<Void> refreshToken(HttpServletResponse response) {
    UserSS user = UserService.authenticated();
    String token = jwtUtil.generateToken(user.getUsername());
    response.addHeader("Authorization", "Bearer " + token);
    return ResponseEntity.noContent().build();
}
```

Nota: por que usamos POST e não GET, PUT ou DELETE?

- GET, PUT e DELETE devem ser usados para operações IDEMPOTENTES. Uma operação idempotente é aquela que, se usada mais de uma vez, produz o mesmo resultado se usada uma única vez.

Esqueci a senha

Checklist:

- ⌚ Criar a classe AuthService com a operação sendNewPassword(String email)
- ⌚ Em EmailService, acrescentar a operação sendNewPasswordEmail(Cliente cliente, String newPass)
- ⌚ Implementar a operação em AbstractEmailService
- ⌚ Em AuthResource, criar o endpoint /auth/forgot (POST)
- ⌚ Definir o endpoint /auth/forgot como público

