Spring Security: preparando a parte do usuário

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Considerações iniciais
- Revisão do que será implementado
- Laboratório
- Considerações finais

Adicionando as dependências

- Adicionar as dependências do Spring Security

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>5.0.7.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.0.7.RELEASE</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.7.0</version>
</dependency>
```

Continuando a preparação da parte de usuário

- Criar a classe Role
  - implementando a interface GrantedAuthority
  - criar os getters e setters, usar o (lombok)
  - sobrescrever o método getAuthority

Configuração do Spring Security: Classe Role

```java
package br.com.futurodev.primeiraapi.model;

import org.springframework.security.core.GrantedAuthority;

import javax.persistence.*;

3 usages
@Entity
@Table(name = "role")
public class Role implements GrantedAuthority {

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

    3 usages
    private String nomeRole; /* Papel, ROLE_GERENTE, ROLE_ADMINISTRADOR*/

    @Override
    public String getAuthority() { return this.nomeRole; }
```

Continuando a preparação da parte de usuário

- Refatorar classe Usuario, considerando:
  - Implementar a interface UserDetails
  - sobrescrever todos os métodos herdados
  - passar para true os métodos com retorno boolean
  - retorno os atributos corretos nos métodos:
    - getPassword
    - getUsername
  - Fazer o mapeamento do atributo roles

Configuração do Spring Security: Classe Usuario

- import org.springframework.security.core.userdetails.UserDetails;

```
@Entity
@Table(name = "usuario")
public class Usuario implements UserDetails {
```

```
@OneToMany(fetch = FetchType.EAGER)
@JoinTable(name = "usuarios_role", uniqueConstraints = @UniqueConstraint(
        columnNames = {"usuario_id", "role_id"}, name = "unique_role_usuario"),
        joinColumns = @JoinColumn(name = "usuario_id", referencedColumnName = "id", table = "usuario",
                foreignKey = @ForeignKey(name = "usuario_fk", value = ConstraintMode.CONSTRAINT)),

        inverseJoinColumns = @JoinColumn(name = "role_id", referencedColumnName = "id", table = "role",
                updatable = false,
foreignKey = @ForeignKey(name ="role_fk", value = ConstraintMode.CONSTRAINT)))
private List<Role> roles; /* Os papeis ou acessos do usuário*/
```

Configuração do Spring Security: Classe Usuario

```
public Collection<? extends GrantedAuthority> getAuthorities() { return this.roles; }

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

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

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

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

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

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

Continuando a preparação da parte de usuário

- Na interface UsuarioRepository:
  - implementar consulta para obter usuário por login

```
2 usages   bruno
@Query(value = "select u from Usuario u where u.login = ?1")
Usuario findUserByLogin(String login);
```

Continuando a preparação da parte de usuário

- Na classe CadastroUsuarioService
  - adicionar implementação da interface UserDetailsService

Configuração da Classe CadastroUsuarioService

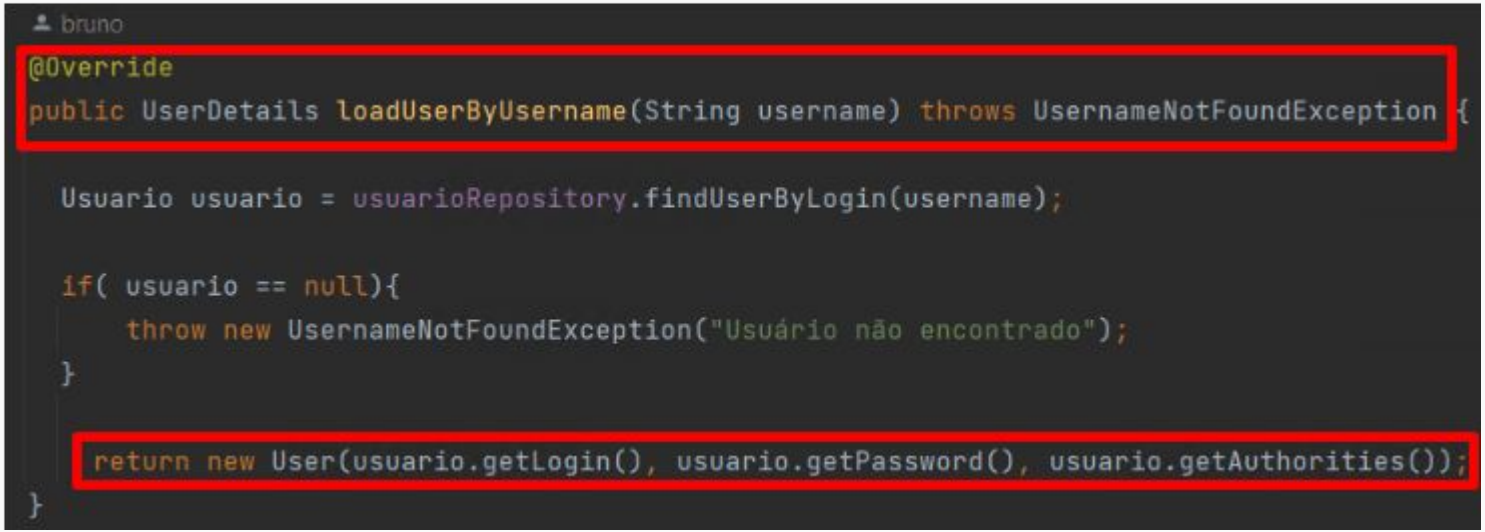
- import org.springframework.security.core.userdetails.UserDetailsService;

```
@Service
public class CadastroUsuarioService implements UserDetailsService {

    6 usages
    @Autowired
    private UsuarioRepository usuarioRepository;
```

Configuração da Classe CadastroUsuarioService

- import org.springframework.security.core.userdetails.UserDetailsService;



Configuração da Classe CadastroUsuarioService

- Usar a anotação **@Transactional(readOnly = true)**, o Spring Security necessita de uma transação para realizar as operações

```
@Transactional(readOnly = true)
public List<Usuario> getUsers(){

    List<Usuario> usuarios = usuarioRepository.findAll();

    for (Usuario usuario: usuarios) {
        usuario.getTelefones().isEmpty();
    }

    return usuarios;
}
```

Laboratório: projeto spring-security

- Etapas da preparando o projeto
  - Ajustando a parte do usuário para Spring Security
    - Vídeo 1 - (https://youtu.be/bcViGF3jZ_w)
  - Criando a classe central de configuração do Spring Security
    - Vídeo 2 - (https://youtu.be/xcY5KCRucUA)
  - Adicionando autenticação com token JWT
    - Vídeo 3 - (https://youtu.be/BNjmHzDcnFc)
    - Vídeo 4 - (https://youtu.be/AjzfYIOqxuo)

INTERVALO DE AULA

DEV!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>