Spring Security - role i rejestracja

Wiele ról

W systemie jeden użytkownik może pełnić naraz kilka funkcji (np. być zarówno klientem, jak i administratorem)

Migracja danych

1. Utworzenie tabeli roles i wstawienie domyślnych ról

```
CREATE TABLE roles (
  id   TEXT        PRIMARY KEY,
  name TEXT        NOT NULL UNIQUE
);
```

```
INSERT INTO roles(id, name)
VALUES
  ('1', 'ROLE_ADMIN'),
  ('2', 'ROLE_USER');
```

2. Utworzenie tabeli łączącej użytkowników z rolami

```
CREATE TABLE user_roles (
  user_id TEXT NOT NULL,
  role_id TEXT    NOT NULL,
  PRIMARY KEY (user_id, role_id),
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
  FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE CASCADE
);
```

3. Przeniesienie istniejących danych

```
INSERT INTO user_roles(user_id, role_id)
SELECT u.id, r.id
FROM users u
JOIN roles r
  ON u.role = r.name;
```

4. Usunięcie starej kolumny

```
ALTER TABLE users
  DROP COLUMN role;
```

Migracja danych

Testowo dodanie roli adminowi lukasz dodatkowej roli:

```
INSERT INTO user_roles (user_id, role_id)
VALUES (
  (SELECT id FROM users WHERE login = 'lukasz'),
  (SELECT id FROM roles WHERE name = 'ROLE_USER')
);
```

@ManyToMany

Relacja wiele-do-wielu: wielu użytkowników może mieć wiele ról, i każda rola może należeć do wielu użytkowników.

W JPA bez dodatkowych ustawień ta adnotacja tworzy tabelę pośrednią (join table), która będzie przechowywać same pary kluczy obcych.

W klasie User:

```
@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(
    name = "user_roles",
    joinColumns = @JoinColumn(name = "user_id"),
    inverseJoinColumns = @JoinColumn(name = "role_id")
)
private Set<Role> roles;
```

@ManyToMany

Klasa Role:

```
@Entity
@Table(name = "roles")
@Getter
@Setter
@ToString
@NoArgsConstructor
@AllArgsConstructor
@Builder

public class Role {
    @Id
    @Column(nullable = false, unique = true)
    private String id;

    @Column(nullable = false, unique = true)
    private String name;

    @ManyToMany(mappedBy = "roles")
    @JsonIgnore
    // odwrotna strona relacji:
    private Set<User> users;
}
```

@Role w Spring Security

Zamiast pojedyńczej roli, należy zmapować wszystkie role:

W MyUserDetailsService, loadUserByUsername:

```
var authorities =
user.getRoles().stream()
        .map(role -> new
SimpleGrantedAuthority(role.getName()))
        .toList();
```

W JwtUtil:

```
private List<String> getUserRoles (UserDetails
 userDetails){
    return userDetails.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority)
            .collect(Collectors.toList());
}
```

Role w JWT

Przykładowy token jwt po logowaniu z 2 rolami

DECODED HEADER

JSON CLAIMS TABLE


```
{
  "typ": "JWT",
  "alg": "HS256"
}
```


DECODED PAYLOAD

JSON CLAIMS TABLE


```
{
  "roles": [
    "ROLE_ADMIN",
    "ROLE_USER"
  ],
  "sub": "lukasz",
  "iat": 1747789751,
  "exp": 1747793351
}
```

Rejestracja

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class UserRequest {
    private String login;
    private String password;
}
```

Dodajemy endpoint:

```
@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody UserRequest req) {
    try {
        userService.register(req);
        return ResponseEntity
                .status(HttpStatus.CREATED)
                .body("Registered successfully");
    } catch (IllegalArgumentException ex) {
        return ResponseEntity
                .badRequest()
                .body(ex.getMessage());
    }
}
```

Rejestracja

Tworzymy RoleRepository:

```
public interface RoleRepository extends JpaRepository<Role, String> {
    Optional<Role> findByName(String name);
}
```

Tworzymy UserService:

```
public interface UserService {
    void register(UserRequest req);
    Optional<User> findByLogin(String login);
}
```

...do rozbudowania...

Rejestracja

Znajdujemy role, Tworzymy nowego użytkownika, dodajemy mu role(USER_ROLE) i zapisujemy

```
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final RoleRepository roleRepository;
    private final PasswordEncoder passwordEncoder;

    @Override
    public void register(UserRequest req) {
        if (userRepository.findByLogin(req.getLogin()).isPresent()) {
            throw new IllegalArgumentException("Error...");
        }
        Role userRole = roleRepository.findByName("ROLE_USER")
                .orElseThrow(() ->
                      new IllegalStateException("There is no role... ROLE_USER"));

        User u = User.builder()
                .id(UUID.randomUUID().toString())
                .login(req.getLogin())
                .password(passwordEncoder.encode(req.getPassword()))
                .roles(Set.of(userRole))
                .build();
        userRepository.save(u);
    }

    @Override
    public Optional<User> findByLogin(String login) {
        return userRepository.findByLogin(login);
    }
}
```

Rejestracja

Zabezpieczenie metod:
np. Zabezpieczenie poszczególnych metod,np:

```
.requestMatchers(HttpMethod.POST,"/api/vehicles").hasRole("ADMIN")

//do testu:
.requestMatchers(HttpMethod.GET,"/api/vehicles/available").hasRole("USER")
```

Lub przeniesienie metod do kontrolera ze zmapowanym endpointem (np. dla admina):

```
.requestMatchers("/api/admin/**").hasRole("ADMIN")
```

Rejestracja

Pytanie:

Jak zabezpieczyć endpoint, kiedy chcemy pokazać dane dotyczące konkrentego zalogowanego Użytkownika?

Dziękuję za uwagę!