



Spring Security + JWT

Logowanie + zabezpieczenie endpoint'ów



Spring Security

Spring Security to framework, który koncentruje się na zapewnieniu zarówno uwierzytelniania, jak i autoryzacji aplikacji

Trzy główne koncepty:

Authentication –
uwierzytelnianie (login, hasło)

Authorization – sprawdzenie
uprawnień zalogowanego
użytkownika.

Servlet filters – filtry, które są
uruchamiane przed przejściem
do kontrolera

JWT

Autoryzacja: najczęstszy scenariusz korzystania z JWT. Po zalogowaniu użytkownika każde kolejne żądanie będzie zawierało JWT, umożliwiając użytkownikowi dostęp do zasobów, które są dozwolone za pomocą tego tokena.

JSON Web Token (JWT) to otwarty standard (RFC 7519), który definiuje kompaktowy i samodzielny sposób bezpiecznego przesyłania informacji między stronami jako obiekt JSON. Informacje te można zweryfikować i zaufać im, ponieważ są podpisane cyfrowo. JWT mogą być podpisywane przy użyciu klucza tajnego (z algorytmem HMAC) lub pary kluczy publiczny/prywatny przy użyciu RSA lub ECDSA.

What is the JSON Web Token structure?

In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

- Header
- Payload
- Signature

Therefore, a JWT typically looks like the following:

```
xxxxx.yyyyy.zzzzz
```

Zależności

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.12.6</version>
</dependency>
```

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.12.6</version>
  <scope>runtime</scope>
</dependency>
```

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.12.6</version>
  <scope>runtime</scope>
</dependency>
```

Spring Security: Artefakt spring-boot-starter-security (zawierający Spring Security 6).

Biblioteka JJWT (do obsługi tokenów JWT): jjwt-api (interfejsy JWT).

jjwt-impl (implementacje generowania i parsowania tokenów).

jjwt-jackson (integracja JJWT z biblioteką Jackson dla serializacji JSON).

Uwaga nowe wersje SpringSecurity jak i JWT różnią się w użyciu od starszych wersji!! np. SpringSecurity<5.7

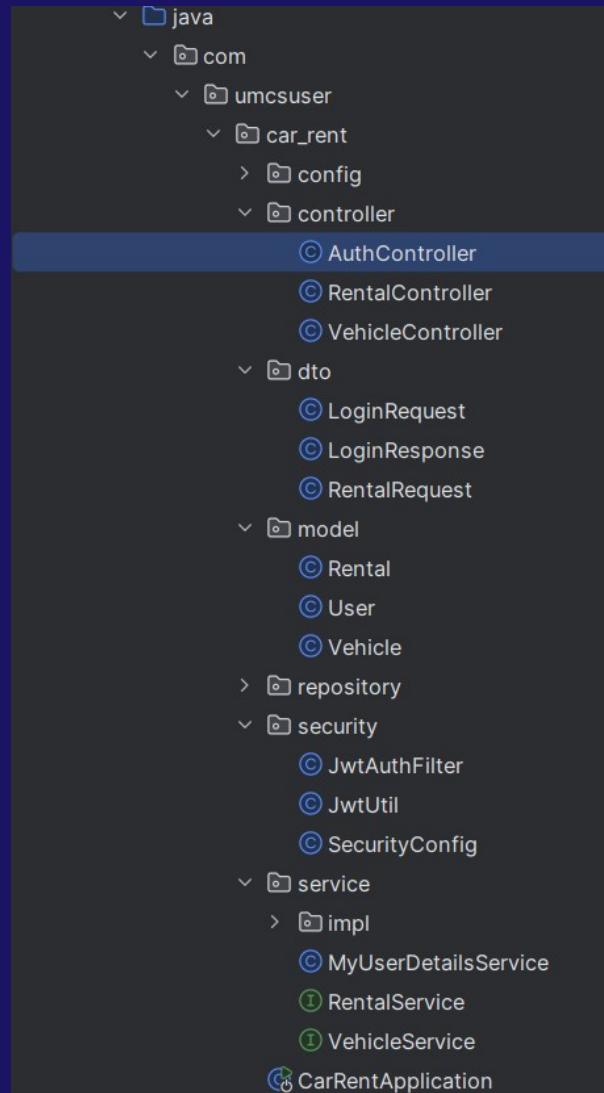
application.properties

```
jwt.secret = secret #w base64, można wygenerować ze strony: https://generate.plus/en/base64  
jwt.expiration = 3600000 #1h
```

Nowe Komponenty

SecurityConfig
JwtUtil
JwtAuthFilter
MyUserDetailsService

AuthController
LoginRequest
LoginResponse



SecurityConfig -1

```
@Configuration                //adnotacja konfiguracji - definicja beanów
@EnableWebSecurity             //włączamy mechanizm WebSecurity!
@EnableMethodSecurity          //Włączamy zabezpieczania metod - np. @PreAuthorize
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(
        HttpSecurity http,                // obiekt do łańcucha ustawień obsługi bezpieczeństwa
        JwtAuthFilter jwtAuthFilter,      // Niestandardowy filtr do implementacji - weryfikacja
        AuthenticationProvider authProvider //Dostawca uwierzytelnienia
    ) throws Exception {
        http
            .csrf(csrf -> csrf.disable()) //w rest api - brak sesji i ciasteczek, wyłączamy
            //ochronę //Cross-Site Request Forgery
            .authorizeHttpRequests(auth -> auth //konfigurator autoryzacji i ustalanie reguł:
                .requestMatchers("/api/auth/**").permitAll() //endpointy do auth(login,register)
                .requestMatchers("/api/admin/**").hasRole("ADMIN") //Endpointy dla ROLE_ADMIN!(zmienić
                //w bazie dla Usera na ROLE_USER i ROLE_ADMIN !
                .anyRequest().authenticated() //każdy inny request wymaga zalogowania
            )
            //Brak sesji, rest api z tokenami jwt!
            .sessionManagement(sess -> sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            //Dostawca uwierzytelnienia
            .authenticationProvider(authProvider)
            //Weryfikacja tokenu przed standardowym sposobem autoryzacji!
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

SecurityConfig -2

```
@Bean
public AuthenticationProvider authenticationProvider( //Bean z dostawcą autentykacji
    UserDetailsService userDetailsService, //Serwis do ładowania danych usera
    PasswordEncoder passwordEncoder //Encoder
)

{
    DaoAuthenticationProvider provider = new DaoAuthenticationProvider(); //Obiekt do Autentykacji z
    bazy
    provider.setUserDetailsService(userDetailsService); //serwis używany do
    autentykacji
    provider.setPasswordEncoder(passwordEncoder); //encoder
    return provider;
}

@Bean
public PasswordEncoder passwordEncoder() { //Używamy bcrypt do hashowania hasła
    return new BCryptPasswordEncoder();
}

@Bean
public AuthenticationManager authenticationManager(
    AuthenticationConfiguration config //konfiguracja uwierzytelniania
) throws Exception {
    return config.getAuthenticationManager(); //dostęp do AuthenticationManagera.
}
```


JwtAuthFilter -1

```
@Component
public class JwtAuthFilter extends OncePerRequestFilter { //filtr zostanie wykonany tylko raz dla
każdego przychodzącego żądania
    @Autowired
    private JwtUtil jwtUtil; //Do zaimplementowania - operacje na tokenach JWT.

    @Autowired
    private UserDetailsService userDetailsService; // Serwis do ładowania danych użytkownika.
    //Główna metoda filtra, która jest wywoływana dla każdego przychodzącego żądania HTTP:
    @Override
    protected void doFilterInternal(HttpServletRequest request, //przychodzące żądanie HTTP
                                   HttpServletResponse response, //Odpowiedź serwera
                                   FilterChain filterChain) //łańcuch filtrów - można przekazać
do następnego filtra
    {
        throws ServletException, IOException {
        String authHeader = request.getHeader("Authorization"); //token JWT jest przesyłany w tym
nagłówku
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            // Brak nagłówka Authorization lub niepoprawny format -> przekazujemy żądanie dalej (nie
autentykujemy)
            filterChain.doFilter(request, response);
            return;
        }
    }
```

JwtAuthFilter -2

```
//Gdy Istnieje nagłówek w formacie "Bearer <token>":
String token = authHeader.substring(7); //sam token
try {
    String username = jwtUtil.extractUsername(token);
    // Sprawdzamy, czy użytkownik nie jest już uwierzytelniony, np.. przez inne mechanizmy
    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

        //szczegóły użytkownika z bazy danych
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        //weryfikacja tokena na podstawie danych użytkownika
        if (jwtUtil.validateToken(token, userDetails)) {
            //Jeśli token ważny, tworzymy obiekt autentykacji (UsernamePasswordAuthenticationToken)
            UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(userDetails, null,
                    userDetails.getAuthorities());
            //Dodatkowe informacje z requesta -ip klienta(id sesji nie ma)
            authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            // Ustawiamy kontekst bezpieczeństwa - użytkownik zalogowany
            SecurityContextHolder.getContext().setAuthentication(authToken);
        }
    }
} catch (JwtException e) {
    //brak Authentication w SecurityContext
}
//żądanie idzie dalej w łańcuchu filtrów
filterChain.doFilter(request, response);
}
```

JwtUtil-1

@Component

```
public class JwtUtil {  
    @Value("${jwt.secret}")  
    private String secretKey; // Secret key - do podpisywania tokenu  
    @Value("${jwt.expiration}")  
    private long expirationMs; // czas ważności tokenu  
  
    //Metoda do generowania tokenu:  
    public String generateToken(UserDetails userDetails) {  
        Map<String, Object> claims = new HashMap<>();  
        claims.put("role", getUserRole(userDetails)); //dodanie niestandardowego Claim (oświadczenia) dla  
        roli usera.  
  
        Date now = new Date();  
        //data wygaśnięcia tokenu:  
        Date expirationDate = new Date(now.getTime() + expirationMs);  
        //budowanie tokenu:  
        return Jwts.builder()  
            .header().type("JWT").and() //opcjonalny typ tokenu  
            .claims().add(claims) //niestandardowe Claimsy - dodatkowa informacja o userze  
            .and() //dalsze budowanie:  
            .subject(userDetails.getUsername()) //podmiot tokenu - u nas username/login  
            .issuedAt(now) //data wystawienia  
            .expiration(expirationDate) //data wygaśnięcia  
            .signWith(getSigningKey()) //Podpisanie tokenu przy użyciu klucza  
            .compact(); //token jako ciąg znaków  
    }  
}
```

JwtUtil-2

```
public String extractUsername(String token) {
    return getClaims(token).getSubject();    //pobranie nazwy usera z tokenu
}
//Walidacja tokenu - czy Username i podpis się zgadza, oraz czy token nie wygasł
public boolean validateToken(String token, UserDetails userDetails) {
    try {
        final String username = extractUsername(token);

        if (!username.equals(userDetails.getUsername())) {
            return false;
        }
        return !getClaims(token).getExpiration().before(new Date());
    } catch (JwtException e) {
        return false;
    }
}

private Claims getClaims(String token) {
    Jws<Claims> jwsClaims = Jwts.parser()    //parser tokena
        .verifyWith(getSigningKey())    //SPRWADZANIE PODPISU TOKENA!!
        .build()    //budowanie parsera
        .parseSignedClaims(token);    //Gdy podpis prawidłowy sparsowanie tokena
    return jwsClaims.getPayload();    // zwrócenie claims (body) - dane usera - ciało tokenu.
}

private SecretKey getSigningKey() {
    byte[] keyBytes = Decoders.BASE64URL.decode(secretKey);    //dekoduje secretKey z base64url z pliku
    return Keys.hmacShaKeyFor(keyBytes);    // generuje SecretKey dla algorytmu HMAC-SHA
    // HMAC to metoda szyfrowania, która wykorzystuje funkcję skrótu (np. SHA-256) i klucz tajny do generowania podpisu
}

private String getUserRole(UserDetails userDetails) {    //Zwraca role usera
    return userDetails.getAuthorities().stream()
        .findFirst()
        .map(GrantedAuthority::getAuthority)
        .orElse(null);
}
```

MyUserDetailsService implements UserService

```
@Service  
@RequiredArgsConstructor
```

```
public class MyUserDetailsService implements UserDetailsService {  
  
    private final UserRepository userRepository;  
  
    @Override  
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
        // Wyszukamy użytkownika po polu 'login'  
        User user = userRepository.findByLogin(username)  
            .orElseThrow(() -> new UsernameNotFoundException("User" + username + " doesn't exist"));  
        // Zbudujemy obiekt UserDetails na podstawie danych encji User  
        // Tworzymy listę Grantów (autoryzacji) z roli użytkownika:  
        List<GrantedAuthority> authorities = List.of(new SimpleGrantedAuthority(user.getRole()));  
  
        return new org.springframework.security.core.userdetails.User(  
            user.getLogin(),  
            user.getPassword(),  
            authorities  
        );  
    }  
}
```

DTO do logowania

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

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class LoginResponse {
    private String token;
}
```

Controller - logowanie

```
@RestController
@RequestMapping("api/auth")
@RequiredArgsConstructor
public class AuthController {
    private final AuthenticationManager authenticationManager;
    private final JwtUtil jwtUtil;

    @PostMapping("/login")
    public ResponseEntity<LoginResponse> login(@RequestBody LoginRequest loginRequest) {
        // 1. Uwierzytelnienie użytkownika za pomocą podanego loginu i hasła
        Authentication auth;
        try {
            auth = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(loginRequest.getLogin(),
                    loginRequest.getPassword())
            );
            // Jeśli po wywołaniu nie rzucono wyjątku, oznacza to, że dane są poprawne
        } catch (BadCredentialsException e) {
            // Niepoprawne dane logowania
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
        // 2. Uwierzytelnienie się powiodło, można wygenerować token JWT
        UserDetails userDetails = (UserDetails) auth.getPrincipal();
        String token = jwtUtil.generateToken(userDetails);
        // 3. Zwracamy token w odpowiedzi
        LoginResponse responseBody = new LoginResponse(token);
        return ResponseEntity.ok(responseBody);
    }
}
```

Rent car

```
@PostMapping("/rent")
public ResponseEntity<Rental> rentVehicle(
    @RequestBody RentalRequest rentalRequest,
    @AuthenticationPrincipal UserDetails userDetails //Dane usera po zalogowaniu zamiast z DTO! jest
    to dostęp do informacji o zalogowanym użytkowniku - w naszym wypadku UserDetails
) {
```

```
    String login = userDetails.getUsername();
    User user = userRepository.findByLogin(login) //Bez CustomUserDetails potrzebne
    repozytorium lub serwis od naszego Usera
        .orElseThrow(() -> new UsernameNotFoundException("Użytkownik nie znaleziony: " + login));

    Rental rental = rentalService.rent(rentalRequest.getVehicleId(), user.getId());
    return ResponseEntity.status(HttpStatus.CREATED).body(rental);
}
```

```
@Getter
public class RentalRequest {
    public String vehicleId;
}
```


GET http://localhost:8080/api/

POST localhost:8080/api/auth/

POST localhost:8080/api/rent/

POST localhost:8080/api/rent/

+ ...

localhost:8080/api/auth/login

Save

POST

localhost:8080/api/auth/login

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

JSON

Beautify

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

<

Dziękuję za uwagę!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.