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,haslo)

Authorization – sprawdzenie uprawnień zalogowanego użytkownika.

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

JWT

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.

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.

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

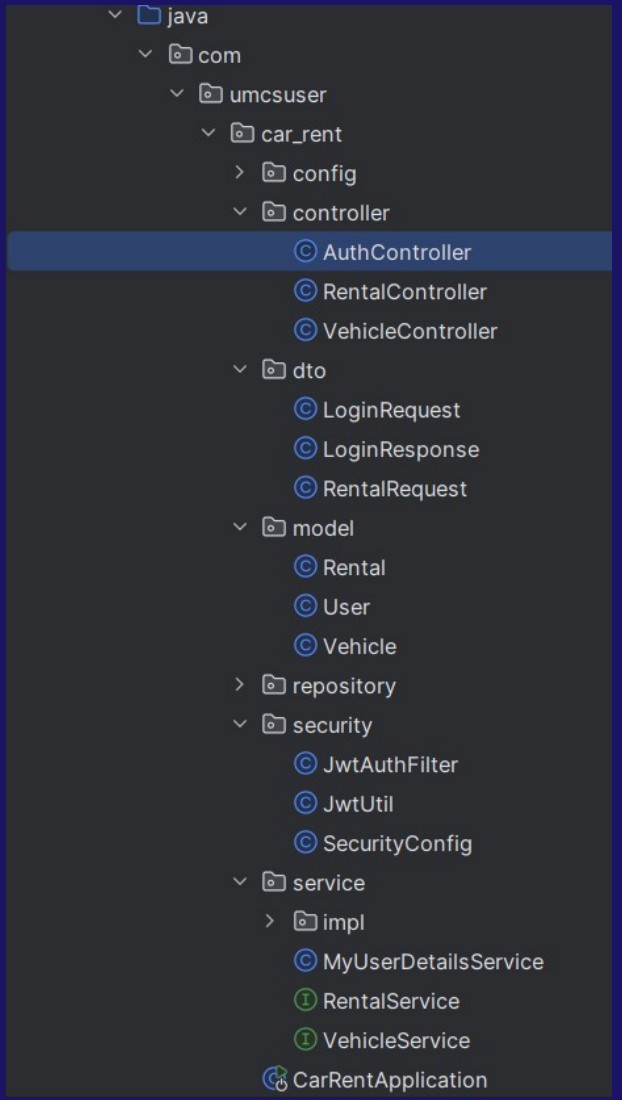
application.properties

```
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
tokenu!
            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 autentykacj
        UserDetailsService userDetailsService,       //Serwis do ładowania danych usera
        PasswordEncoder passwordEncoder              //Encoder
)
```

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

@Bean
public PasswordEncoder passwordEncoder() {     //Używamy bcrypt do hashowania hasla
    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) {
[redacted]
            //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 sparoswanie 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 algorytmuHMAC-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 UserDetailsService

```
@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, możnawygenerować 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);
    }
}
```

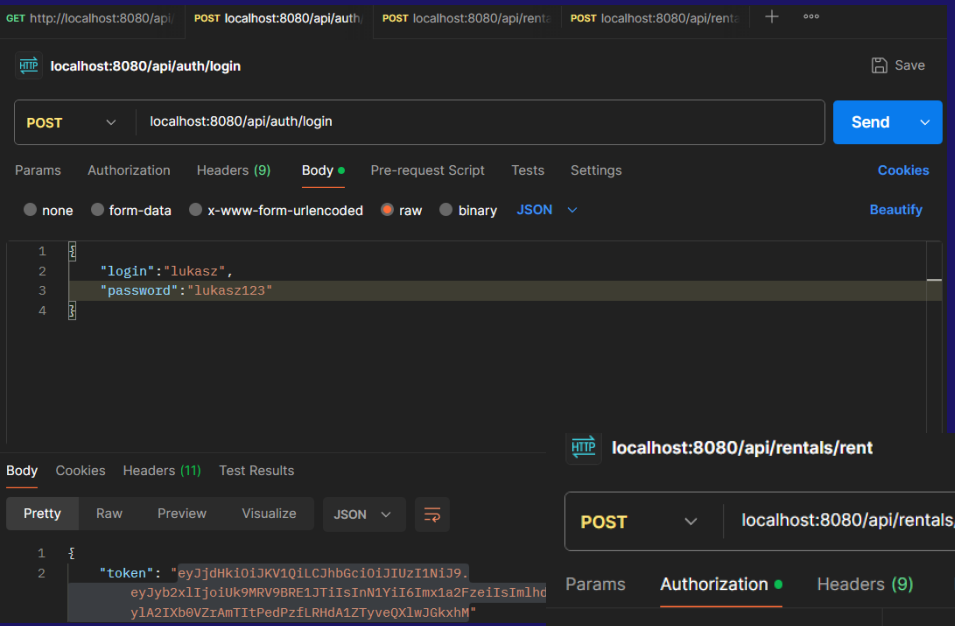
Rent car

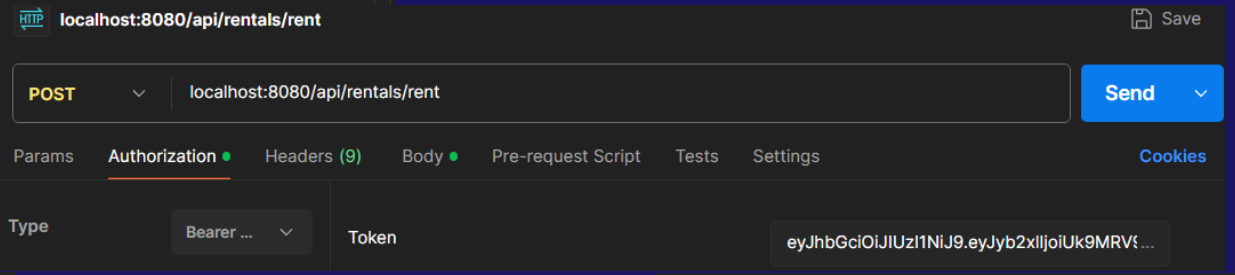
```
@PostMapping("/rent")
public ResponseEntity<Rental> rentVehicle(
        @RequestBody RentalRequest rentalRequest,
        @AuthenticationPrincipal UserDetails userDetails  //Dane usera po zalogowaniu zamiast z DTO! jest
to dostęp do informacji o zalogowanym użytkowniku – w naszym wypadku UserDetails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

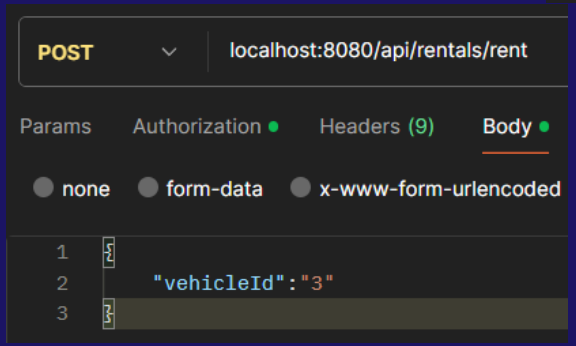
```
    String login = userDetails.getUsername();
    User user = userRepository.findByLogin(login)          //Bez CustomUserDetails potrzebne
repozytorium lub serwis od naszego Usera
            .orElseThrow(() -> new UsernameNotFoundException("Użytkownik nie znaleziony: " + login));


    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
{
"login":"lukasz",
"password":"lukasz123"
}
Body
Cookies
Headers (11)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"token": "eyJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJyb2xlIjoiUk9MRV9BRE1JTiIsInN1YiI6Imx1a2FzeiIsImlhd
ylA2IXb0VZrAmTItPedPzfLRHdA1ZTyveQXlwJGkxhM"


localhost:8080/api/rentals/rent
Save
POST
localhost:8080/api/rentals/rent
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Type
Bearer ...
Token
eyJhbGciOiJIUzI1NiJ9.eyJyb2xlIjoiUk9MRV9...


POST
localhost:8080/api/rentals/rent
Params
Authorization
Headers (9)
Body
none
form-data
x-www-form-urlencoded
{
"vehicleId":"3"
}


"id": "1e089c62-dd00-49ca-ab41-5220fa1839b3",
"vehicle": {
"id": "3",
"price": 300.0,
"category": "Car",
"brand": "Toyota",
"model": "Corolla",
"year": 2024,
"plate": "LU789",
"attributes": {},
"active": true
},
"user": {
"id": "fc6e600b-e89e-42e2-b2b7-ca18edb68924",
"login": "lukasz",
"password": "$2a$10$7MUYZR8FP8ebLzfqwMK85ejBke/TNp/Yme9NrC0XxRy.NAFde2vEm",
"role": "ROLE_ADMIN"
},
"rentDate": "2025-05-14T04:11:48.7467397",
"returnDate": null

Dziękuję za uwagę!