

OTH-Regensburg  
Übungen zur Vorlesung  
Softwareentwicklung

Übung Nr. 9  
Restful APIs mit JWT gesichert

**Ziele:**

- Erstellen Sie eine Spring Boot-Anwendung mit einer RESTful-API zur Benutzerverwaltung.
- Implementieren Sie Benutzerauthentifizierung und -autorisierung mit JWT-Token.
- Sichern Sie die API-Endpunkte mit JWT-Authentifizierung.
- Speichern Sie Benutzeranmeldeinformationen in einer Datenbank.

**Hinweis: Alternativ können Sie direkt die Musterlösung verwenden und die Zeit statt zum Kopieren für das Studium des Codes verwenden.**

Aufgabe 1 – Erstellen eines neuen Projekts und der Abhängigkeiten

- Importieren Sie die folgenden Abhängigkeiten for Maven:

```
Spring Web
Spring Security
Spring Data JPA
H2
Spring Security JWT:
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
</dependency>
```

Fügen Sie der Datei application.properties diese Werte hinzu:

```
spring.messages.basename=messages
server.error.include-binding-errors=always

#DB
spring.datasource.url=jdbc:h2:mem:testdb;MODE=MySQL;NON_KEYWORDS=USER;
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
#spring.data.jpa.repositories.bootstrap-mode=default

spring.jpa.defer-datasource-initialization=true
spring.jpa.hibernate.ddl-auto=update
spring.sql.init.mode=always

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.devtools.restart.log-condition-evaluation-delta=false
security.jwt.secret-key="ITR.connectToATT(PabcdeffghijklmnopqrstuvwxyzABCDEFHIJKLMNOPQRSTUVWXYZ0123456789P,
PabcdeffghijklmnopqrstuvwxyzABCDEFHIJKLMNOPQRSTUVWXYZ0123456789P)";
# 1h in millisecond
security.jwt.expiration-time=3600000
```

Fügen Sie der Datei data.sql diese Werte hinzu:

```
INSERT INTO USER (email, password, login, active) VALUES ('thomas@gmail.com', '$2a$12$8K4uC9YPI659Qnz6NUqy9e35xsoQ/OlsaVhIWRJP913VpsuIQGZNy', 'thomas', 1);
INSERT INTO USER (email, password, login, active) VALUES ('anja@gmail.com', '$2a$12$8K4uC9YPI659Qnz6NUqy9e35xsoQ/OlsaVhIWRJP913VpsuIQGZNy', 'anja', 1);

INSERT INTO AUTHORITY (description) VALUES ('ADMIN');
INSERT INTO AUTHORITY (description) VALUES ('STUDENT');

INSERT INTO USERAUTHORITY (IDUSER, IDAUTHORITY) VALUES (1, 1);
INSERT INTO USERAUTHORITY (IDUSER, IDAUTHORITY) VALUES (2, 2);
```

Aufgabe 2- Erstellen Sie die User, Authority, Repository , MyUserDetailsService und DTO classes:

---

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;
    // ... other fields as needed
}
```

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

```
public class Authority implements Serializable {

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

    private String description;

    //getters and setters
}
```

```
public interface AuthorityRepository extends JpaRepository<Authority, Long> {
}
```

```
@Service
public class MyUserDetailsService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }
        return new org.springframework.security.core.userdetails.User(user.getUsername(),
user.getPassword(), new ArrayList<>());
    }
}
```

```
public class AuthenticationRequest implements Serializable {

    private static final long serialVersionUID = 1L;

    private String username;
    private String password;

    //getters and setters
}
```

```
public class AuthenticationResponse implements Serializable {

    private static final long serialVersionUID = 1L;
    private final String jwt;

    public AuthenticationResponse(String jwt) {
        this.jwt = jwt;
    }

    public String getJwt() {
        return jwt;
    }
}
```

```
public class RegisterRequest implements Serializable{

    private static final long serialVersionUID = 1L;
    private String username;
    private String password;
    private String email;

    private ArrayList<Authority> myauthorities = new ArrayList<>();

    //getters and setters
}
```

### Aufgabe 3- Paket Config: SecurityConfig, JwtAuthenticationFilter und JwtService

---

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

    @Autowired
    MyUserDetailsService userDetailsService;

    @Autowired
    private JwtAuthenticationFilter jwtRequestFilter;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http.authorizeHttpRequests(rQ -> {
            rQ.requestMatchers("/api/register", "/api/users/authenticate").permitAll();

            rQ.requestMatchers("/api/search", "/api/demo", "/api/profile").authenticated();

        });

        http.formLogin()
            //LoginPage("/login") , if you want to have a customized login page....
            .and()
            .logout()
            //where the user goes after the logout
            .logoutSuccessUrl("/login")
            .invalidateHttpSession(true)

            .permitAll();

        http
            .headers(headers -> headers
                .frameOptions(frameOptions -> frameOptions
                    .sameOrigin()))
            );

        http.csrf(AbstractHttpConfigurer::disable);

        http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public AuthenticationManager authenticationManager(
        AuthenticationConfiguration authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    @Bean
    public PasswordEncoder getPasswordEncoder() {
        return new BCryptPasswordEncoder();
        //return NoOpPasswordEncoder.getInstance();
    }
}
```

```

//class "Utils" used to manipulate the token, extracting important information from it like username, expiration date etc.
@Service
public class JwtService {

    @Value("${security.jwt.secret-key}")
    private String SECRET_KEY;
    @Value("${security.jwt.expiration-time}")
    private long Expiration_Time;
    //private final long Expiration_Time= 1000 * 60 * 60 * 10;
    private final String PREFIX= "Bearer ";

    private Key getSigningKey() {
        byte[] keyBytes = this.SECRET_KEY.getBytes(StandardCharsets.UTF_8);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
            .parserBuilder()
            .setSigningKey(this.getSigningKey())
            .build()
            .parseClaimsJws(token)
            .getBody();
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return createToken(claims, userDetails.getUsername());
    }

    private String createToken(Map<String, Object> claims, String subject) {

        return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + Expiration_Time))
            .signWith(getSigningKey())
            .compact();
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && isTokenExpired(token));
    }

    public String getSECRET_KEY() {
        return SECRET_KEY;
    }

    public long getExpiration_Time() {
        return Expiration_Time;
    }

    public String getPREFIX() {
        return PREFIX;
    }
}

```

```

//Additional filter to intercept the request and check if there is a token on its header and validate it.
//If there is, then it will try to validate the token.
//If there is not, the user should authenticate himself in the sequence.

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private MyUserDetailsService userDetailsService;

    @Autowired
    private JwtService jwtService;

    @Override
    protected void doFilterInternal(
            jakarta.servlet.http.HttpServletRequest request,
            HttpServletResponse response,
            FilterChain chain)
            throws jakarta.servlet.ServletException, IOException {

        //this comes from the header of the request
        final String authorizationHeader = request.getHeader("Authorization");

        String username = null;
        String jwt = null;

        //if the authorization content in the header is not null and starts with "Bearer"...
        //let us extract from the header the token and the username...
        if (authorizationHeader != null && authorizationHeader.startsWith(jwtService.getPREFIX())) {
            jwt = authorizationHeader.substring(jwtService.getPREFIX().length());
            username = jwtService.extractUsername(jwt);
        }

        //if the user is not empty and the user is not yet authenticated, we will update the security context holder with the token
        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

            //trying to find the user in the DB
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

            //if the token is valid, we will proceed with the login...
            if (jwtService.validateToken(jwt, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());

                //putting the information of the request object in the usernamePasswordAuthenticationToken
                usernamePasswordAuthenticationToken
                    .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

                //informing to the SecurityContextHolder that the user is authenticated

                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }

        //continue with the chain of filters

        chain.doFilter(request, response);
    }
}

```

---

## Aufgabe 4 - Paket Controller: AuthenticationController und DemoController

---

```
@RestController
@RequestMapping("/api")
public class AuthenticationController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    MyUserDetailsService userDetailsService;

    @Autowired
    JwtService jwtServiceUtils;

    @Autowired
    BCryptPasswordEncoder encoder;

    @Autowired
    UserRepository userRepository;

    @Autowired
    AuthorityRepository authorityRepository;

    @PostMapping("/post")
    public ResponseEntity<AuthenticationResponse> register(@RequestBody RegisterRequest
        registerRequest){

        User user = new User();
        user.setActive(1);
        user.setEmail(registerRequest.getEmail());
        user.setLogin(registerRequest.getUsername());
        user.setMyauthorities(new ArrayList<Authority> ());

        for (int i = 0; i < registerRequest.getMyauthorities().size(); i++) {
            System.out.println("received auth "+i+" "+registerRequest.getMyauthorities().get(i).getId());
            Optional<Authority> authorityOp = authorityRepository.findById(registerRequest.getMyauthorities().get(i).getId());
            user.getMyauthorities().add(authorityOp.get());
            System.out.println(authorityOp.get().getDescription());

        }
        user.setPassword(encoder.encode(registerRequest.getPassword()));
        user = userRepository.save(user);
        System.out.println("saved the user per API..");
        System.out.println(user.getLogin());
        UserDetails userDetails = userDetailsService.loadUserByUsername(user.getLogin());
        String jwt = jwtServiceUtils.generateToken(userDetails);
        return ResponseEntity.ok(new AuthenticationResponse(jwt));
    }

    @RequestMapping(value = "/users/authenticate", method=RequestMethod.POST)
    public ResponseEntity<?> createAuthenticationToken (@RequestBody AuthenticationRequest authenticationRequest) throws Exception{

        try {
            authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(authenticationRequest.getUsername(),
                    authenticationRequest.getPassword())
            );
        } catch (BadCredentialsException e) {
            throw new Exception ("Incorrect username or password!", e);
        }

        UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());
        String jwt = jwtServiceUtils.generateToken(userDetails);
        return ResponseEntity.ok(new AuthenticationResponse(jwt));
    }
}
```

```

    @RequestMapping("/api")
    @RestController
    public class DemoController {

        @GetMapping("/demo")
        public ResponseEntity<String> helloDemo(){

            return ResponseEntity.ok("User is authenticated and got this answer from Demo Endpoint!");
        }
    }

```

## Aufgabe 5 – Testen Sie die API mit Postman

---

### 5.1 User erstellen

- POST

**/api/register**

**Create a User**

- Legen Sie die HTTP-Methode fest: Wählen Sie „POST“ aus dem Dropdown-Menü.
- Legen Sie die Anforderungs-URL fest: Geben Sie die URL Ihres API-Endpunkts ein, z. B. <http://localhost:8080/api/register>.
- Konfigurieren Sie den Hauptteil der Anfrage in Postman wie unten

The screenshot shows the Postman interface with a POST request to `http://localhost:8080/api/register`. The 'Body' tab is selected, displaying the following JSON payload:

```

1  {
2   "username": "diana",
3   "password": "123456",
4   "email": "diana@gmail.com",
5   "myauthorities": [ {"id": 1}, {"id": 2} ]
6 }

```

- Senden Sie die Anfrage: Klicken Sie auf die Schaltfläche „Senden“, um die Anfrage auszuführen.
- Überprüfen Sie die Antwort:
- Statuscode: Überprüfen Sie den HTTP-Statuscode im Antwortheader. Ein 200 OK zeigt eine erfolgreiche Anfrage an.
- Antworttext: Untersuchen Sie den JSON-Antworttext, um das Token zu überprüfen.
- Kopieren Sie den Text in den Anführungszeichen des Tokens und analysieren Sie ihn auf der folgenden Website:  
<https://jwt.io/>

## 5.2 User authentication

- POST [/api/users/authenticate](#) **Authenticates a User**

- Legen Sie die HTTP-Methode fest: Wählen Sie „POST“ aus dem Dropdown-Menü.
- Legen Sie die Anforderungs-URL fest: Geben Sie die URL Ihres API-Endpunkts ein, z. B. <http://localhost:8080/api/users/authenticate>
- Konfigurieren Sie den Hauptteil der Anfrage in Postman wie unten



- Untersuchen Sie den JWT-Antworttext, um das Token zu überprüfen.
- Prüfen Sie, dass auch die in 5.1 erstellte Userin Diana authentifiziert werden kann.

## 5.3 Accessing a Resource with the Token from 5.2

- GET [/api/demo](#) **Access a Demo Endpoint with a JWT Token**

- Legen Sie die HTTP-Methode fest: Wählen Sie „GET“ aus dem Dropdown-Menü.
- Legen Sie die Anforderungs-URL fest: Geben Sie die URL Ihres API-Endpunkts ein, z. B. <http://localhost:8080/api/demo>
- Wählen Sie im Tab „Autorsierung“ das Attribut „Auth Type“ als „Bearer Token“
- Kopieren Sie das Token aus 5.2 (nur Inhalt innerhalb der Anführungszeichen) in das Feld „Token“.
- Klicken Sie auf „Senden“, um eine Antwort zu erhalten:

GET | http://localhost:8080/api/demo

Send | Cookies

Docs Params Authorization ● Headers (8) Body Scripts Settings

Auth Type: Bearer Token

Token: [REDACTED]

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Body Cookies (1) Headers (11) Test Results | ⏱

200 OK | 111 ms | 362 B | 🔍 | ⚙️ | ⌂

Raw ▾ | Preview | Visualize | ⏷

User is authenticated!