Modul 223

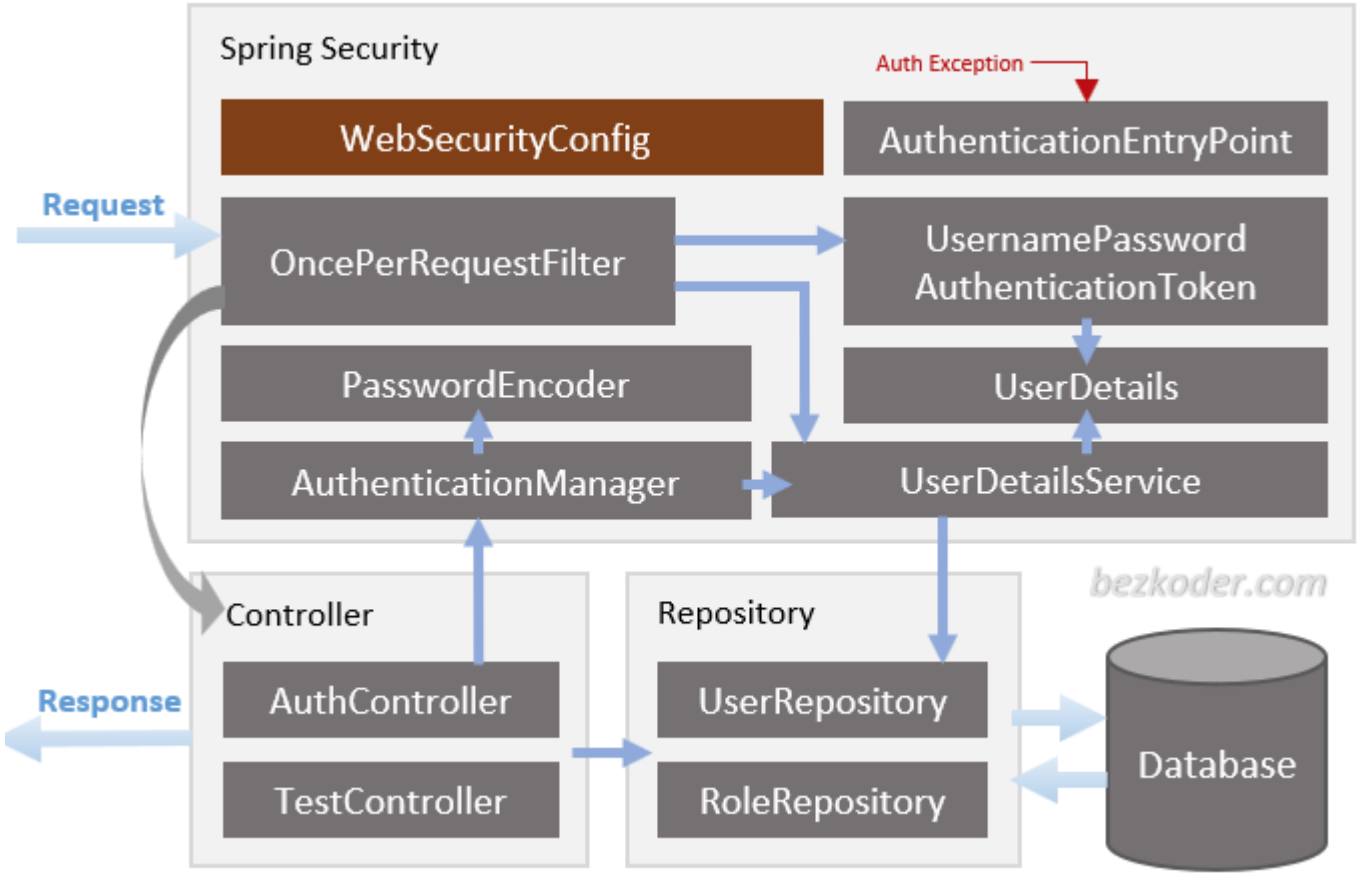
Multiuser - Apps

Unterrichtsziel heute

- Begriffsklärung JWT
- Erklären Ablauf Authentifizierung mit Spring-Security
- Erklären Architektur Spring-Security mit JWT
- Implementieren Session-basierte Auth² mit Spring-Security
- Implementieren JWT Token-basierte Auth² mit Spring-Security

Spring Security – interner Nachrichtenfluss

https://www.bezkoder.com/spring-boot-jwt-authentication/



Implementierung: User

```java
@Entity
@Table(name="User")
public class User {

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

    @NotBlank
    private String username;
    @NotBlank
    private String email;
    @NotBlank
    private String password;

    public User(){ }

    public User(String name, String email, String password) {
        this.username = name;
        this.email = email;
        this.password = password;
    }

    @ManyToMany(fetch = FetchType.LAZY) //das ist der spannende ORM Teil: automatisches Mapping von M-N Beziehungen :-)
    private Set<Role> roles = new HashSet<>();
   /* all setters and getters */
}
```

Implementierung: ERole und Role

```java
public enum ERole {
    ROLE_USER,
    ROLE_ADMIN
}
```

```java
@Entity
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private ERole name;

    public Role() { }

    public Role(ERole name) {
      this.name = name;
    }
    public String toString(){
      return name.toString();
    }
    /* setter und getter */
}
```

Implementierung: Repositories

```
@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
    Optional<Role> findByName(ERole name);
}
```

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
  Optional<User> findByUsername(String username);
  Boolean existsByUsername(String username);
  Boolean existsByEmail(String email);
}
```

Implementierung: Rollen vorbelegen

```
@SpringBootApplication
public class WissQuiz2024Application implements CommandLineRunner{

    @Autowired
    RoleRepository roleRepository;

    public static void main(String[] args) {
        SpringApplication.run(WissQuiz2024Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        if (roleRepository.count() == 0) {
            roleRepository.save(new Role(ERole.ROLE_USER));
            roleRepository.save(new Role(ERole.ROLE_ADMIN));
        }
    }
}
```

Implementierung: UserDetailsImpl.java

aka der Kit zwischen unseren User-Objekten und Springboot-Security

```java
public class UserDetailsImpl implements UserDetails {
private static final long serialVersionUID = 1L;

private Long id;
private String username;
private String email;

@JsonIgnore
private String password;

private Collection<? extends GrantedAuthority> authorities;

public UserDetailsImpl(Long id, String username, String email, String password,
    Collection<? extends GrantedAuthority> authorities) {
  this.id = id;
  this.username = username;
  this.email = email;
  this.password = password;
  this.authorities = authorities;
}

public static UserDetailsImpl build(User user) {
  List<GrantedAuthority> authorities = user.getRoles().stream()
      .map(role -> new SimpleGrantedAuthority(role.toString()))
      .collect(Collectors.toList());

  return new UserDetailsImpl(
      (long)user.getId(),
      user.getUsername(),
      user.getEmail(),
      user.getPassword(),
      authorities);
}
```

```java
@Override
  public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
  }

  public Long getId() {  return id;   }

  public String getEmail() {  return email;   }

  @Override
  public String getPassword() {  return password;  }

  @Override
  public String getUsername() { return username;  }

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

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

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

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

  @Override
  public boolean equals(Object o) {
    if (this == o)     return true;
    if (o == null || getClass() != o.getClass()) return false;
    UserDetailsImpl user = (UserDetailsImpl) o;
    return Objects.equals(id, user.id);
  }
}
```

Imports für UserDetailsImpl.java

```
import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

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

import com.fasterxml.jackson.annotation.JsonIgnore;
```

Implementierung: UserDetailsServiceImpl

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService{
  @Autowired UserRepository userRepository;
  @Override
  @Transactional
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: " + username));
    List<GrantedAuthority> authorities = new ArrayList<>(); // important role to authorities mapping
    for (Role r : user.getRoles()) {
      authorities.add(new SimpleGrantedAuthority(r.toString()));
    }
    return UserDetailsImpl.build(user);
  }
}
```

Imports für UserDetailsServiceImpl.java

```
import java.util.ArrayList;import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
```

User, Role und Repositories aus eigenen Packages importieren!

Implementierung: AuthController.java

API-Endpoint für Benutzerregistrierung und -anmeldung

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

  @Autowired AuthenticationManager authenticationManager;
  @Autowired UserRepository userRepository;
  @Autowired RoleRepository roleRepository;
  @Autowired PasswordEncoder encoder;

  @PostMapping("/login")
  public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {
    Authentication authentication = authenticationManager.authenticate(
        new UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
                                                loginRequest.getPassword()));
    SecurityContextHolder.getContext().setAuthentication(authentication);
    return ResponseEntity.ok("OK");
  }
```


Hinweis: List, Set und HashSet aus *java.util* importieren, unterstrichene Klassen siehe nächste Folie, alles andere aus *org.springframework*

Implementierung: AuthController.java

WISS Schulen für Wirtschaft Informatik Immobilien

API-Endpoint für Benutzerregistrierung und -anmeldung

```
@PostMapping("/signup")
public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) {
  if (userRepository.existsByUsername(signUpRequest.getUsername())) {
    return ResponseEntity.badRequest().body("Error: Username is already taken!"); }
  if (userRepository.existsByEmail(signUpRequest.getEmail())) {
    return ResponseEntity.badRequest().body("Error: Email is already in use!"); }
  // create new user account
  User user = new User(signUpRequest.getUsername(),
      signUpRequest.getEmail(), encoder.encode(signUpRequest.getPassword()));
  // handle roles for the new user
  Set<String> strRoles = signUpRequest.getRoles();
  Set<Role> roles = new HashSet<>();
  if ( strRoles==null ) { roles.add(roleRepository.findByName(ERole.ROLE_USER).get());
  } else {
    strRoles.forEach(role -> {
      switch (role) {
        case "admin":
          roles.add(roleRepository.findByName(ERole.ROLE_ADMIN).get());
          break;
        default:
          roles.add(roleRepository.findByName(ERole.ROLE_USER).get());
      }
    });
  }
  user.setRoles(roles);
  userRepository.save(user);
  return ResponseEntity.ok("User registered successfully!");
 }
}
```


Hinweis: List, Set und HashSet aus *java.util* importieren, unterstrichene Klassen siehe nächste Folie, alles andere aus *org.springframework*

Imports für AuthController.java

```
import java.util.HashSet;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

SignupRequest, LoginRequest siehe nächste Folie

User, Role und Repositories aus eigenen Packages importieren!

Signup-, LoginRequest und MessageResponse

mappen JSON-formatierte Request- bzw. Response-Parameter zu Objekten und umgekehrt

```java
@Setter @Getter
public class SignupRequest {
    @NotBlank
    @Size(min = 3, max = 20)
    private String username;

    @NotBlank
    @Size(max = 50)
    @Email
    private String email;

    private Set<String> roles;

    @NotBlank
    @Size(min = 6, max = 40)
    private String password;

}
```

```java
@Setter @Getter
public class LoginRequest {

    private String username;
    private String password;

}
```

Am Besten ins dto package speichern (*data transfer object*)

Imports für AuthController.java

```
import java.util.HashSet;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

SignupRequest, LoginRequest siehe nächste Folie

User, Role und Repositories aus eigenen Packages importieren!

SecurityConfiguration anpassen

```java
@Configuration
public class SecurityConfiguration {

  @Autowired private MyUserDetailsService userDetailsService;

  @Bean
  public DaoAuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
    authProvider.setUserDetailsService(userDetailsService);
    authProvider.setPasswordEncoder(passwordEncoder());
    return authProvider;
  }

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

  @Bean
  public PasswordEncoder passwordEncoder() {  return new BCryptPasswordEncoder();  }

  private static final String[] EVERYONE = { "/api/auth/**", "/hello" };

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf(csrf -> csrf.disable())
        .cors(Customizer.withDefaults())
        .authorizeHttpRequests(auth -> auth.requestMatchers(EVERYONE).permitAll()
            .anyRequest().authenticated()) //add anything u need
        .formLogin(Customizer.withDefaults()) // für Login-Form
        .httpBasic(Customizer.withDefaults()); // für CURL, Postman
    return http.build();
  }
}
```

Imports für SecurityConfiguration

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
```

& that's how to run basic Auth

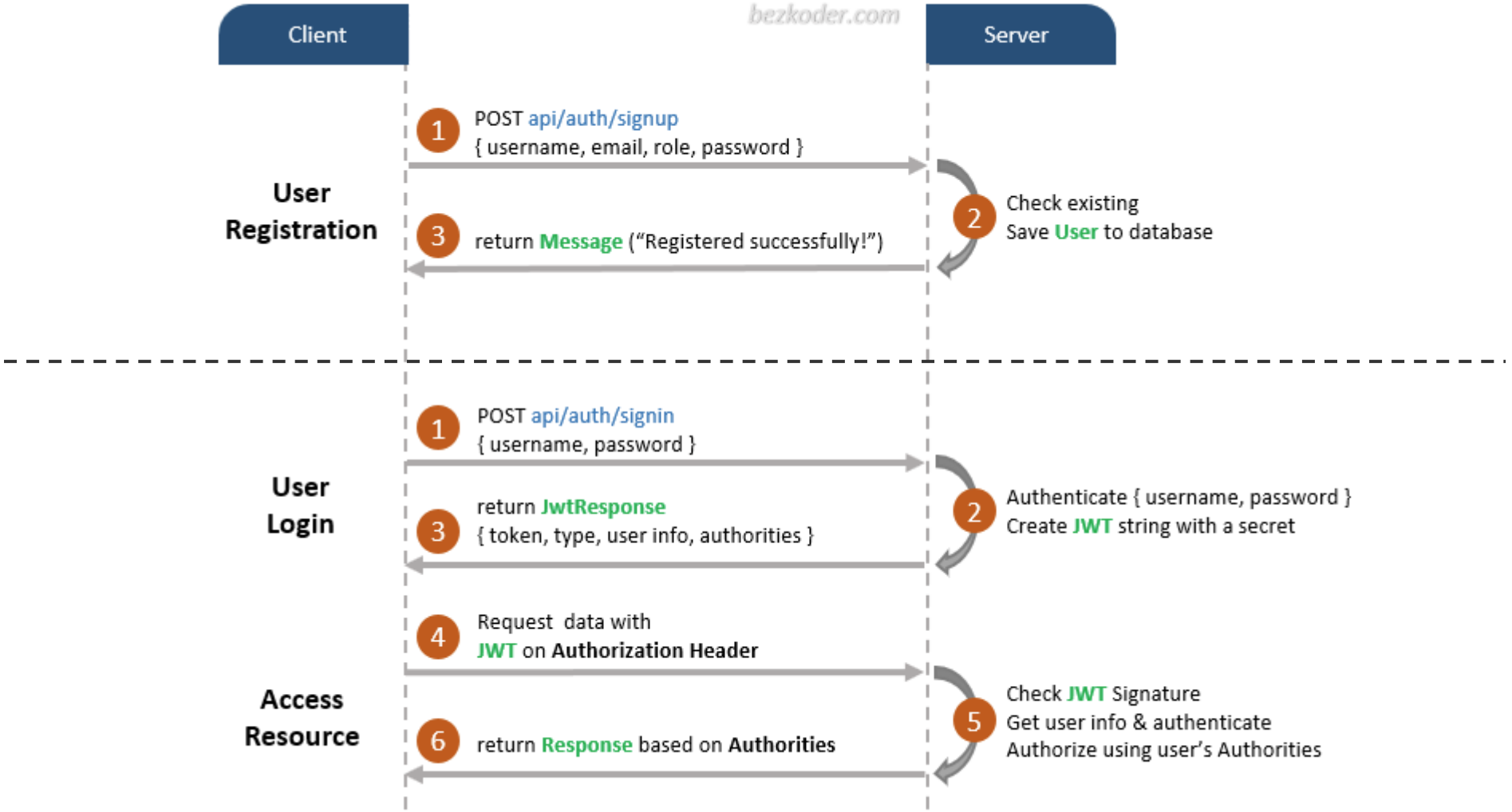
How about...

- running the app?
- check the demo data exists in the database?
- create users and test login?
- writing some Unit-Tests?

Next level: Token basiert

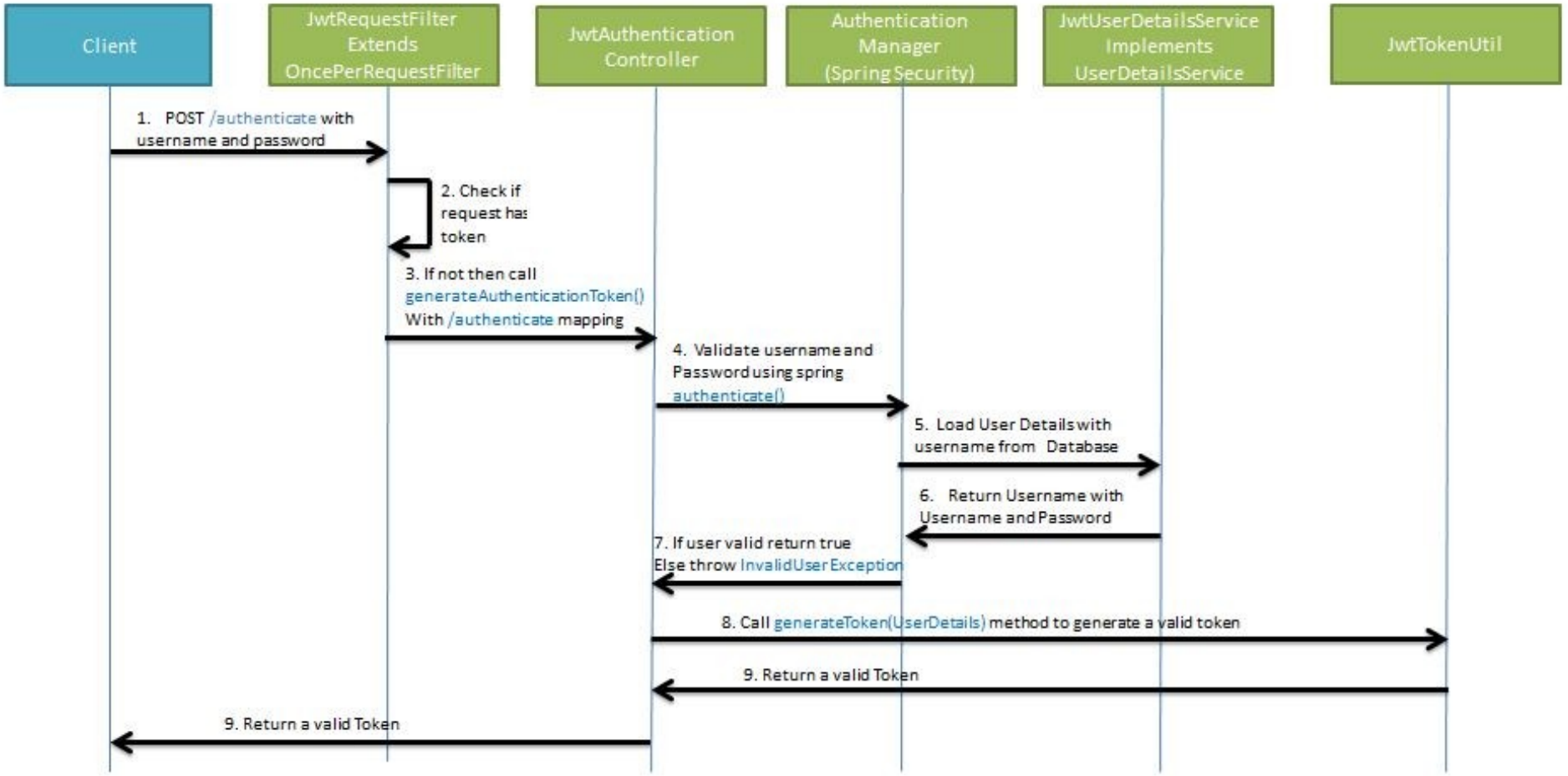
geplante Abläufe: – Signup + SignIn

Folien und Code basieren auf dem Tutorial von
https://www.bezkoder.com/spring-boot-jwt-authentication/



Wie funktioniert die JWT-Generierung?

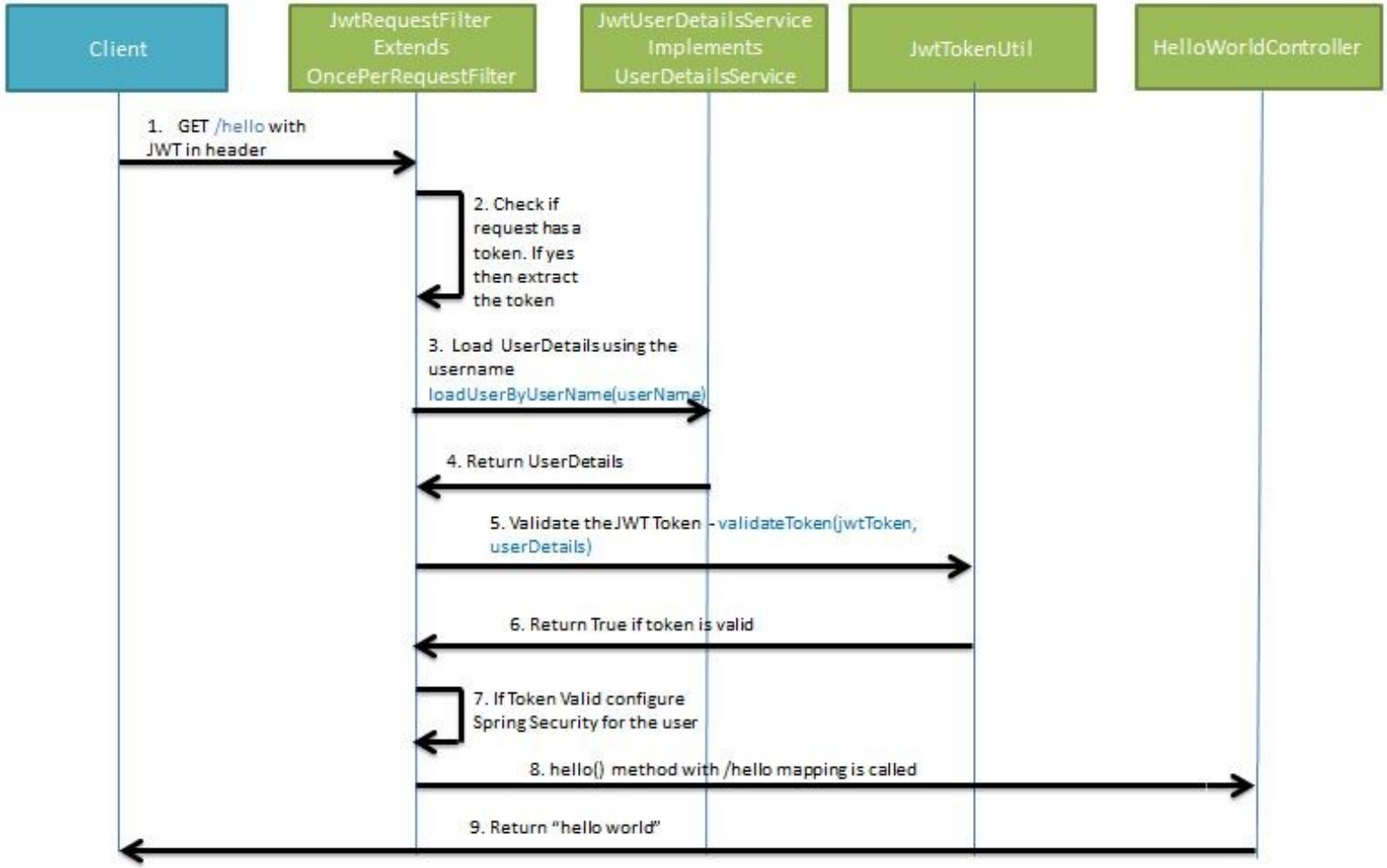
(im Backend - beim Login, (2) vorherige Folie



https://www.javainuse.com/spring/boot-jwt

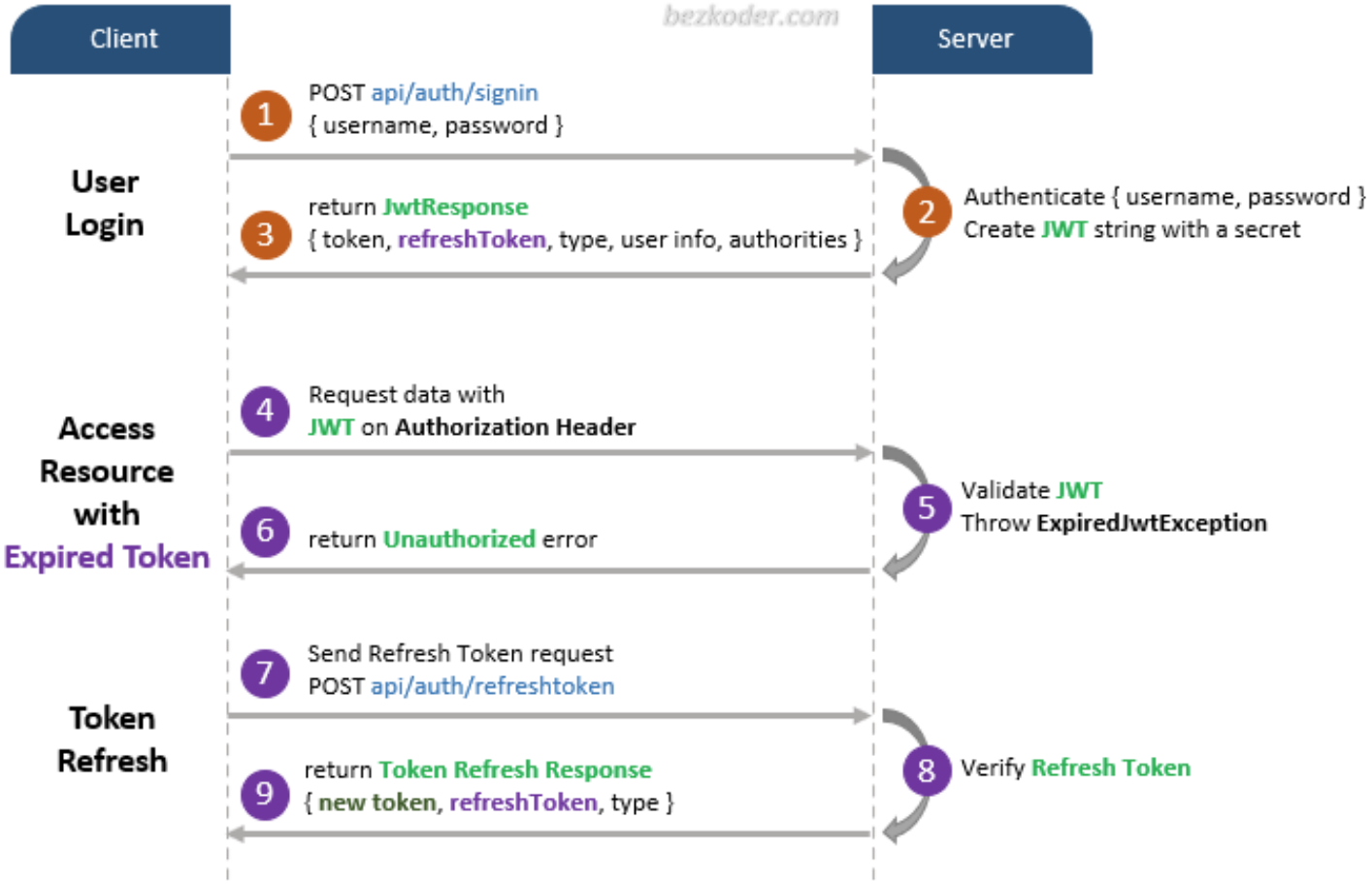
Wie funktioniert JWT - Validierung?

(im Backend bei weiteren Requests, (5) vorletzte Folie



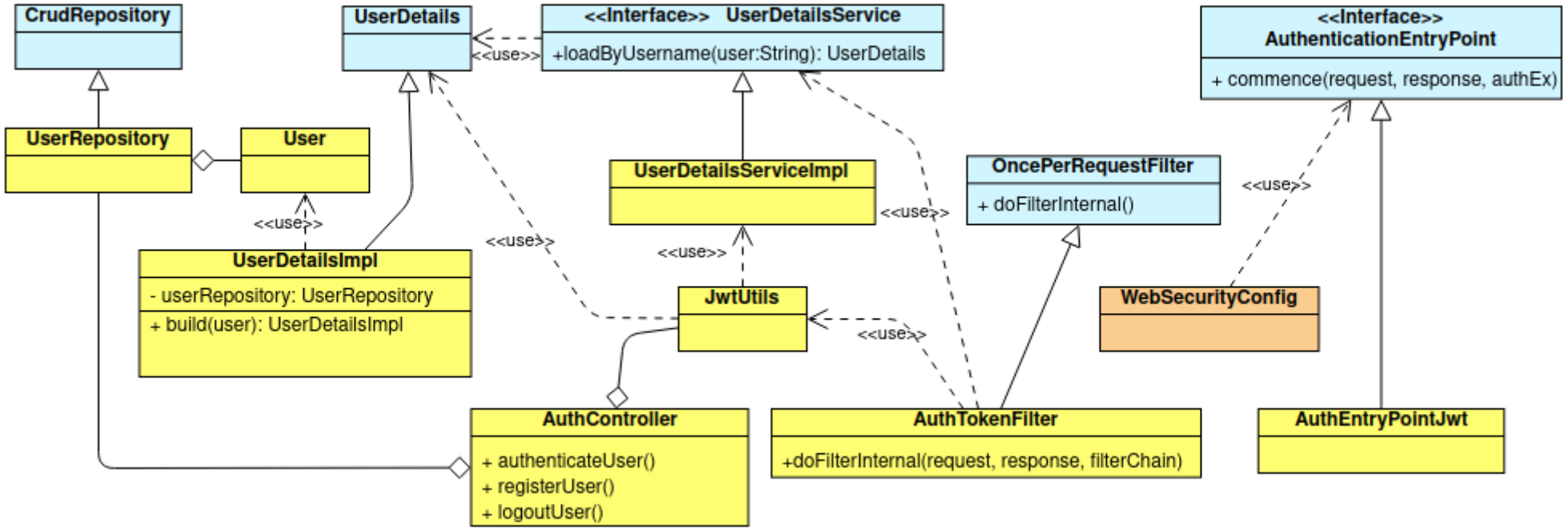
Spring Security – Login

https://www.bezkoder.com/spring-boot-jwt-authentication/



Spring Security: Architektur

https://www.bezkoder.com/spring-boot-jwt-authentication/



Benötigte dependencies in pom.xml

works with <parent><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-parent</artifactId>**<version>3.4.5</version>**<relativePath/></parent>

```
<dependency>
 <groupId>javax.servlet</groupId>
 <artifactId>servlet-api</artifactId>
 <version>2.5</version>
 <scope>provided</scope>
</dependency>

<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-data-jpa</artifactId>
 <version>3.0.0</version>
</dependency>

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

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

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

<dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <scope>runtime</scope>
</dependency>

<dependency>
 <groupId>org.projectlombok</groupId>
 <artifactId>lombok</artifactId>
 <version>1.18.30</version>
 <scope>provided</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
<dependency>
<groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.3</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
 <dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt-impl</artifactId>
<version>0.12.3</version>
<scope>runtime</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->
<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt-jackson</artifactId>
<version>0.12.3</version>
<scope>runtime</scope>
</dependency>
```

Implementierung: JwtUtils.java

```java
/**
 * This class has 3 main functions:
 *  generateToken: create JWT Token from Auth object
 *  extractUserName: get username from JWT
 *  validateToken: validate a JWT with a secret
 */
@Component
public class JwtUtils {
private static final Logger logger =
LoggerFactory.getLogger(JwtUtils.class);

//aus application.properties
@Value("${myapp.jwtSecret}")
private String jwtSecret;
@Value("${myapp.jwtExpirationMs}")
private int jwtExpirationMs;

public String generateToken(String username) {
 Map<String, Object> claims = new HashMap<>();
 return Jwts.builder() .claims(claims)
   .subject(username)
   .issuedAt(new Date())
   .expiration(new Date(System.currentTimeMillis()
       +jwtExpirationMs))
   .signWith(getSignKey()) .compact();
}
```

```java
  private Key getSignKey() {
    byte[] keyBytes = Decoders.BASE64.decode(SECRET);
    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.parser()
    .verifyWith((SecretKey)getSignKey())
    .build()
    .parseSignedClaims(token)
    .getPayload();
  }
  private Boolean isTokenExpired(String token) {
   return extractExpiration(token).before(new Date());
  }
  public Boolean validateToken(String token){
   return !isTokenExpired(token);
  }
 }
```


Hinweis import: *org.springframework* bzw. *io.jsonwebtoken*

Imports für JwtUtils-Klasse

```
import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

import javax.crypto.SecretKey;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
```

Implementierung: AuthTokenFilter

```java
/**
 * a filter that executes once per request. AuthTokenFilter class that extends OncePerRequestFilter and overrides doFilterInternal() method.
 */
public class AuthTokenFilter extends OncePerRequestFilter {
  @Autowired
  private JwtUtils jwtUtils;

  @Autowired
  private UserDetailsServiceImpl userDetailsService;
  private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);

  @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
    try {
        String jwt = parseJwt(request);
        if (jwt != null && jwtUtils.validateToken(jwt)) {
            String username = jwtUtils.extractUsername(jwt);
            UserDetails userDetails = userDetailsService.loadUserByUsername(username);
            UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    } catch (Exception e) {
        logger.error("Cannot set user authentication: {}", e);
    }
    filterChain.doFilter(request, response);
  }

  private String parseJwt(HttpServletRequest request) {
    String headerAuth = request.getHeader("Authorization");
    if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
      return headerAuth.substring(7, headerAuth.length());
    }
    logger.error("Cannot extract Bearer Token");
    return null;
  }
}
```

Imports AuthTokenFilter

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
```

Implementierung: AuthEntryPointJwt.java

```java
/**
 * Handle Authentication Exception
 * override the commence() method.
 * This method is triggered anytime an unauthenticated User requests a secured HTTP resource
 * and an AuthenticationException is thrown
 */
@Component
public class AuthEntryPointJwt implements AuthenticationEntryPoint {
    private static final Logger logger = LoggerFactory.getLogger(AuthEntryPointJwt.class);

    /**
     * commence heisst "anfangen"
     */
    @Override
    public void commence(HttpServletRequest request,
            HttpServletResponse response, AuthenticationException authException)
            throws IOException, jakarta.servlet.ServletException {
        logger.error("Unauthorized error: {}", authException.getMessage());
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Error: Unauthorized");
    }
}
```


Hinweis: Logger und LoggerFactory aus *org.slf4j* importieren

Imports für AuthEntryPointJwt

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
```

Implementierung: AuthController.java

API-Endpoint für Benutzerregistrierung und -anmeldung

```java
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/auth")
public class AuthController {
  @Autowired
  AuthenticationManager authenticationManager;

  @Autowired UserRepository userRepository;

  @Autowired RoleRepository roleRepository;

  @Autowired PasswordEncoder encoder;

  @Autowired JwtUtils jwtUtils;

  @PostMapping("/signin")
public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {

Authentication authentication = authenticationManager.authenticate(
new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword()));

SecurityContextHolder.getContext().setAuthentication(authentication);
String jwt = jwtUtils.generateJwtToken(authentication);

UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();
List<String> roles = userDetails.getAuthorities().stream()
.map(item -> item.getAuthority())
.collect(Collectors.toList());

return ResponseEntity.ok(new JwtResponse(jwt,
 userDetails.getId(),
 userDetails.getUsername(),
 userDetails.getEmail(),
 roles));
}
```

```java
@PostMapping("/signup")
public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) {
if (userRepository.existsByUsername(signUpRequest.getUsername())) {
return ResponseEntity
.badRequest()
.body(new MessageResponse("Error: Username is already taken!"));
}

if (userRepository.existsByEmail(signUpRequest.getEmail())) {
return ResponseEntity
.badRequest()
.body(new MessageResponse("Error: Email is already in use!"));
}

// Create new user's account
User user = new User(signUpRequest.getUsername(),
 signUpRequest.getEmail(), encoder.encode(signUpRequest.getPassword()));

Set<String> strRoles = signUpRequest.getRole();
Set<Role> roles = new HashSet<>();

if (strRoles == null) {
Role userRole = roleRepository.findByName(ERole.ROLE_USER)
.orElseThrow(() -> new RuntimeException("Error: Role is not found."));
roles.add(userRole);
} else {
strRoles.forEach(role -> {
switch (role) {
case "admin":
Role adminRole = roleRepository.findByName(ERole.ROLE_ADMIN)
.orElseThrow(() -> new RuntimeException("Error: Role is not found."));
roles.add(adminRole);
break;
default:
Role userRole = roleRepository.findByName(ERole.ROLE_USER)
.orElseThrow(() -> new RuntimeException("Error: Role is not found."));
roles.add(userRole);
}
});
}

user.setRoles(roles);
userRepository.save(user);

return ResponseEntity.ok(new MessageResponse("User registered successfully!"));
}
}
```


Hinweis: List, Set und HashSet aus *java.util* importieren, unterstrichene Klassen siehe nächste Folie, alles andere aus *org.springframework*

Imports für AuthController

```
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import jakarta.validation.Valid;
```

Alles andere aus den selbst erstellten packages importieren!

JwtResponse und MessageResponse

mappt JWT zu JSON-formatierten Response-Objekt

```java
@Setter @Getter
public class JwtResponse{

    private String token;
    private String type = "Bearer";
    private Long id;
    private String username;
    private String email;
    private List<String> roles;

    public JwtResponse(String accessToken,
                       Long id, String userName,
                       String email, List<String> roles) {
      this.token = accessToken;
      this.id = id;
      this.username = userName;
      this.email = email;
      this.roles = roles;
    }
}
```

```java
@Setter @Getter @AllArgsConstructor
public class MessageResponse {

    private String message;
}
```

Was will ich schützen?

- REST-API Endpoints!
- auf Rollenbasis, z.B. für WissQuiz-Webapp

Rolle	Endpunkt
keine	/api/auth, /quiz, /category (ohne PUT)
MODERATOR	/api/auth, /quiz, /question, /category
ADMIN	/api/auth, /quiz, /question, /category, /user/admin

wird in WebSecurityConfig implementiert (nächste Folie)

Implementierung: WebSecurityConfig I

Globale Konfiguration der geschützen API-Endpoints und Passwortverschlüsselung.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
  @Autowired private UserDetailsServiceImpl userDetailsService;
  @Autowired private AuthenticationEntryPoint unauthorizedHandler;

  private final static String[] EVERYONE = { "/api/auth/**", "/category", "/quiz" };
  private final static String[] SECURE = { "/question" };
  private final static String[] ROLES = { "MODERATOR", "ADMIN"};

  @Bean
  public AuthTokenFilter authenticationJwtTokenFilter() {
    return new AuthTokenFilter();
  }

  @Bean
  public DaoAuthenticationProvider authenticationProvider() {
      DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
      authProvider.setUserDetailsService(userDetailsService);
      authProvider.setPasswordEncoder(passwordEncoder());
      return authProvider;
  }

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

Implementierung: WebSecurityConfig 2

Globale Konfiguration der geschützen API-Endpoints und Passwortverschlüsselung.

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
   http.csrf(csrf -> csrf.disable()).cors(Customizer.withDefaults())
       .exceptionHandling(exception -> exception.authenticationEntryPoint(unauthorizedHandler))
       .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
       .authorizeHttpRequests(auth ->
         auth.requestMatchers(EVERYONE).permitAll()
             .anyRequest().authenticated()
       );
   http.authenticationProvider(authenticationProvider());
   http.addFilterBefore(authenticationJwtTokenFilter(),
                UsernamePasswordAuthenticationFilter.class);
   return http.build();
  }
}
```

Implementierung: WebSecurityConfig 3

Globale CORS – Konfiguration (erlaubt alle Endpoints mit REACT Frontend vom localhost)

```
....
@Bean
 public WebMvcConfigurer corsConfigurer() {
     return new WebMvcConfigurer() {
         @Override
         public void addCorsMappings(CorsRegistry registry) {
             registry.addMapping("/**")
                 .allowedOrigins("http://localhost:5173");
         }
     };
}
```

Imports für SecurityConfiguration

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
```

Logging erweitern

1. application.properties

```
logging.level.org.springframework.web=DEBUG
logging.level.org.springframework.security=DEBUG
```

2. spezielle Log-Klasse: logt eingehende Request-Parameter

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.filter.CommonsRequestLoggingFilter;

@Configuration
public class RequestLoggingFilterConfig {

   @Bean
   public CommonsRequestLoggingFilter logFilter() {
      CommonsRequestLoggingFilter filter = new CommonsRequestLoggingFilter();
      filter.setIncludeQueryString(true);
      filter.setIncludePayload(true);
      filter.setMaxPayloadLength(10000);
      filter.setIncludeHeaders(false);
      filter.setAfterMessagePrefix("REQUEST DATA : ");
      return filter;
   }
}
```

… and that's how we make Multi user apps

on the backend :-)

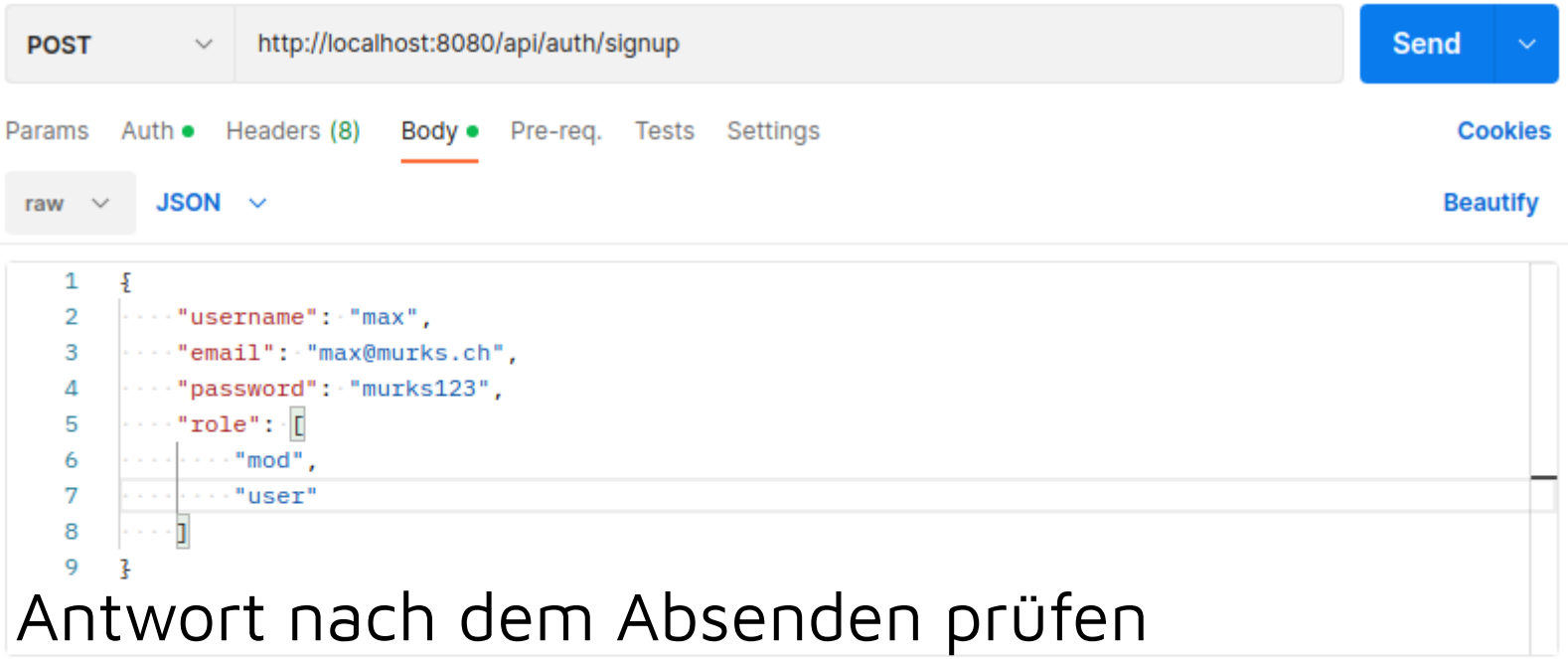
Wait, there's a few more things...

- Wie kann ich im Backend herausfinden, welcher Benutzer den aktuellen Request sendet?

```
Authentication auth =
    SecurityContextHolder.getContext()
                         .getAuthentication();

if (auth.isAuthenticated()){

    Optional<User> currentUser =
        userRepository.findByName(auth.getName());

    if (currentUser.isPresent()){ /* ... */ }

}
```

Benutzer mit Postman (oder ply*) registrieren

1. Parameter im Body als JSON übertragen

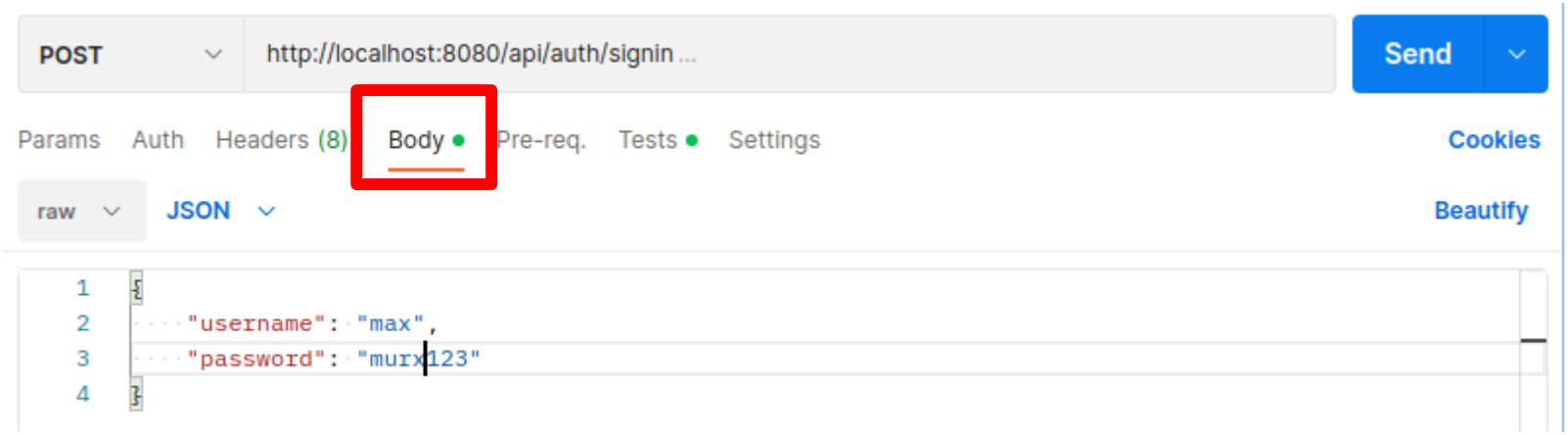


2. Antwort nach dem Absenden prüfen

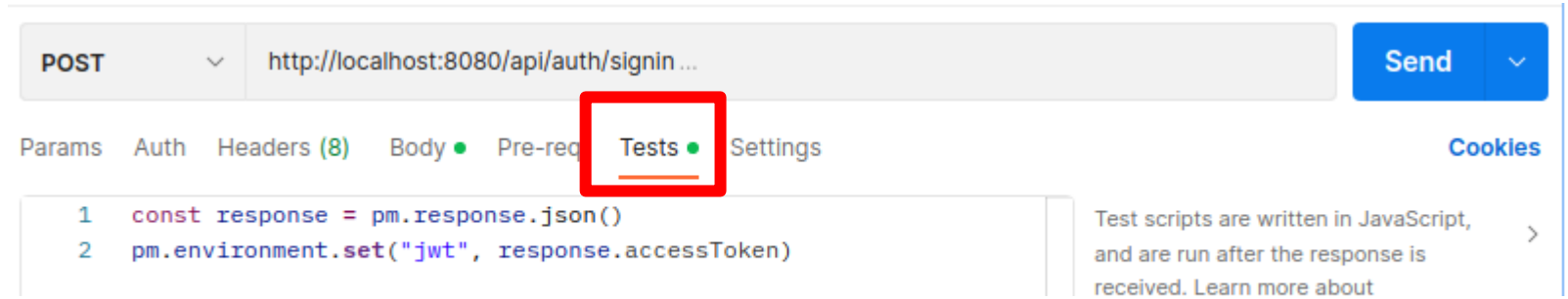
*ply: Addon für VSCode, siehe https://ply-ct.org/ply/

Login mit Postman oder ply* testen

Benutzername und Passwort als JSON im Body übergeben



und JWT als Variable *jwt* zur Weiterverwendung speichern!

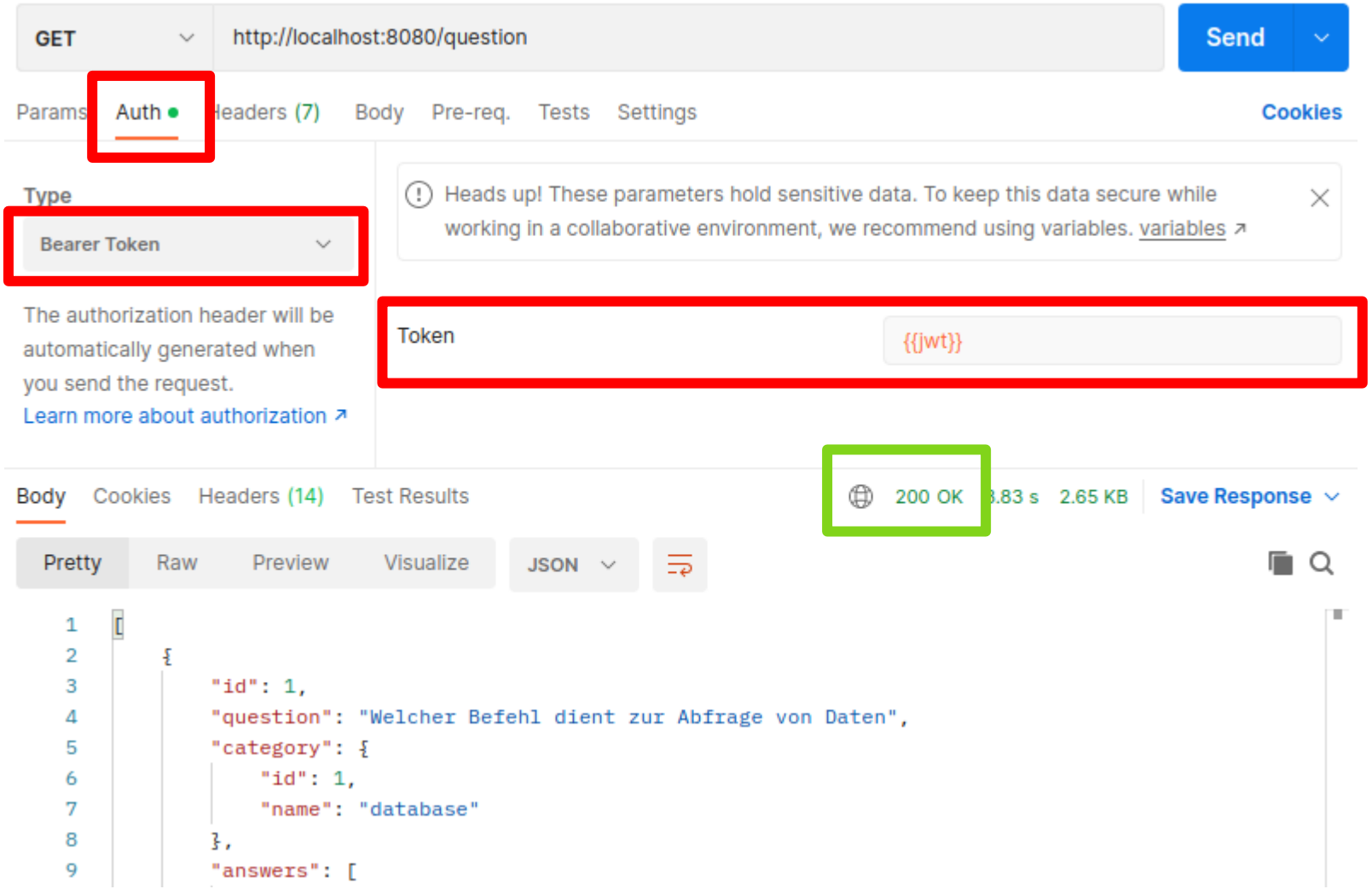


*ply: Addon für VSCode, siehe https://ply-ct.org/ply/

erfolgreiche Login-Response

Zugriff auf geschützten Endpunkt

mit JWT - Postman-Variable



Well done!

Bravo!