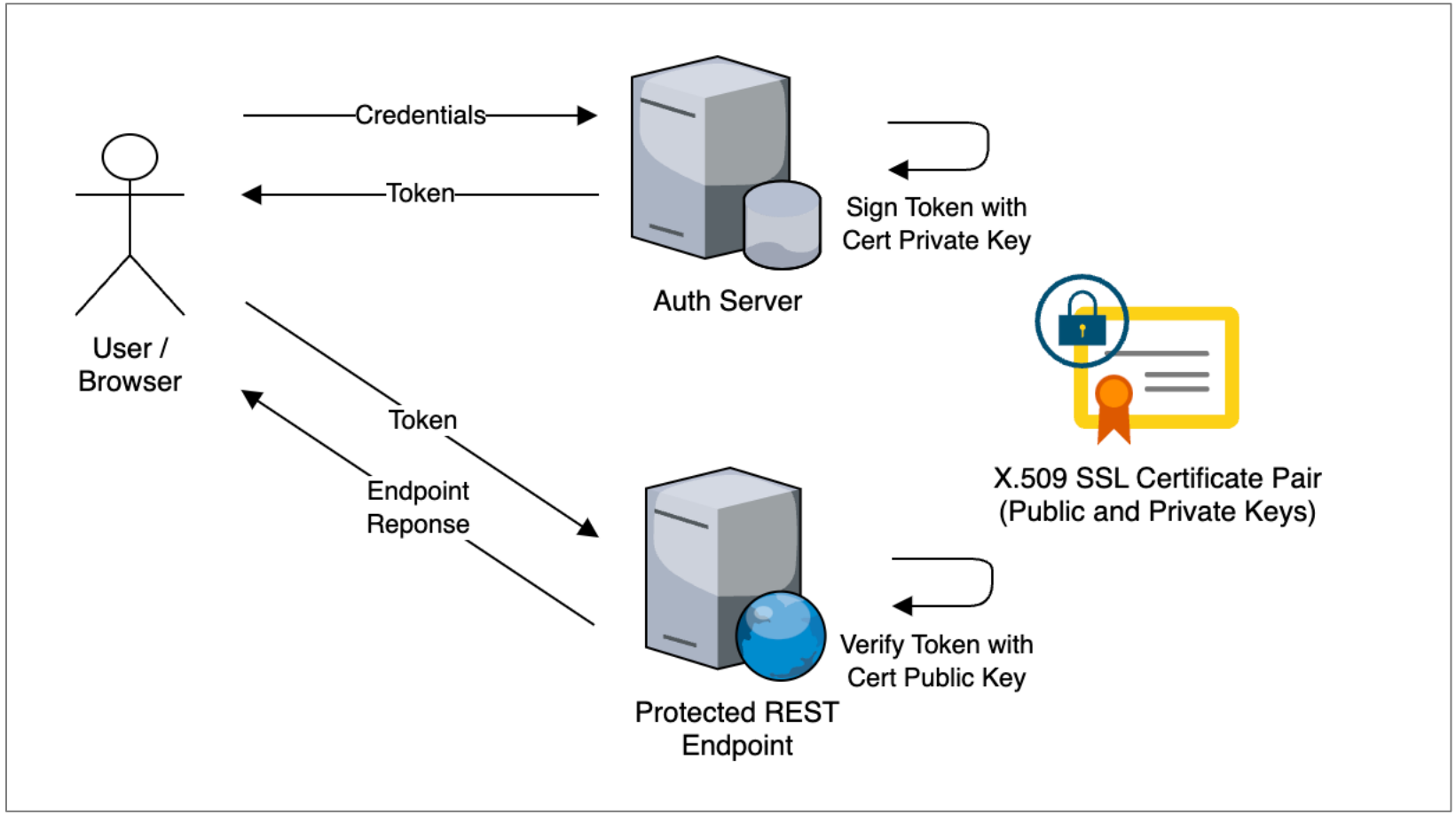
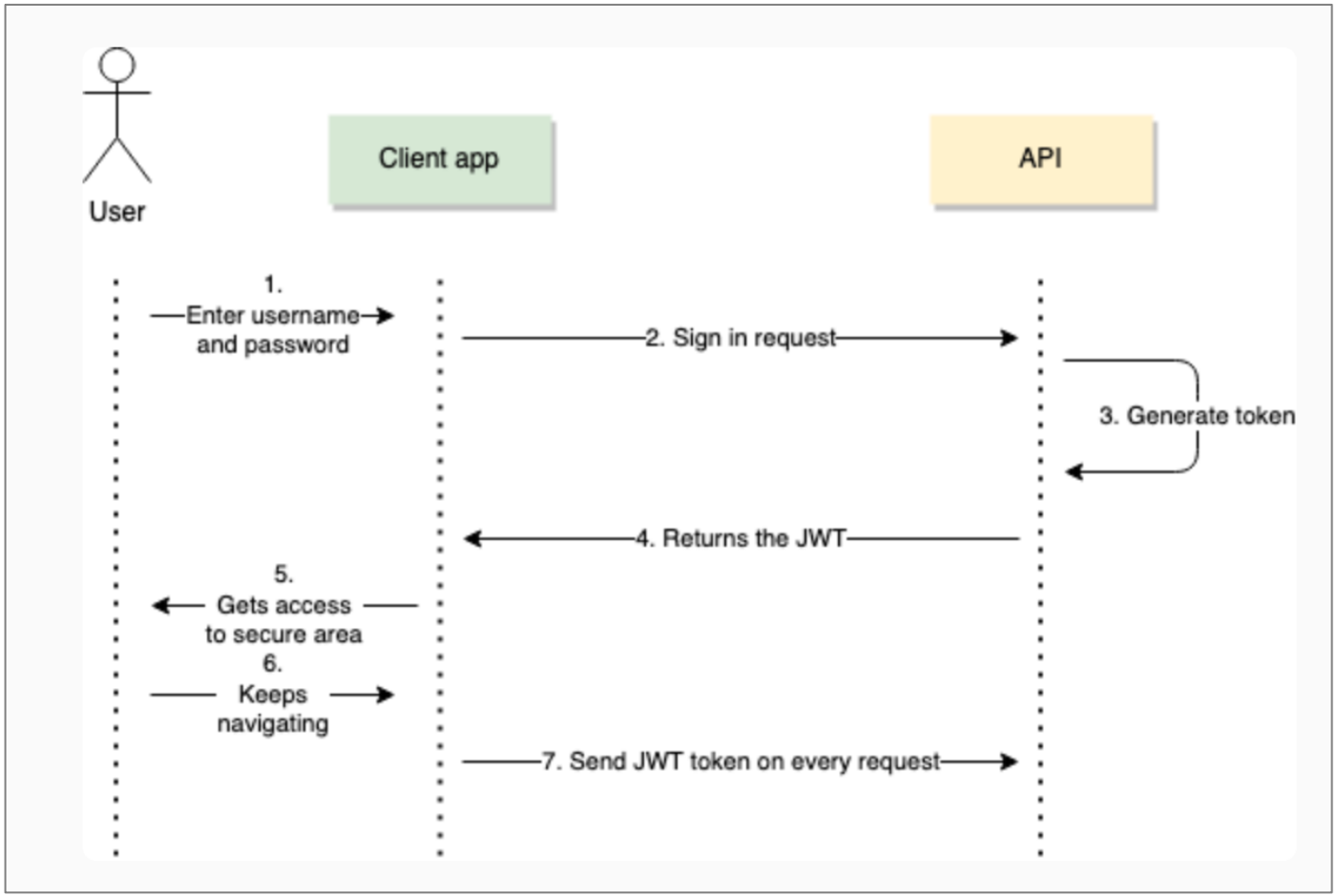
Spring Boot
JWT-based Authentication

JWT-based Authentication

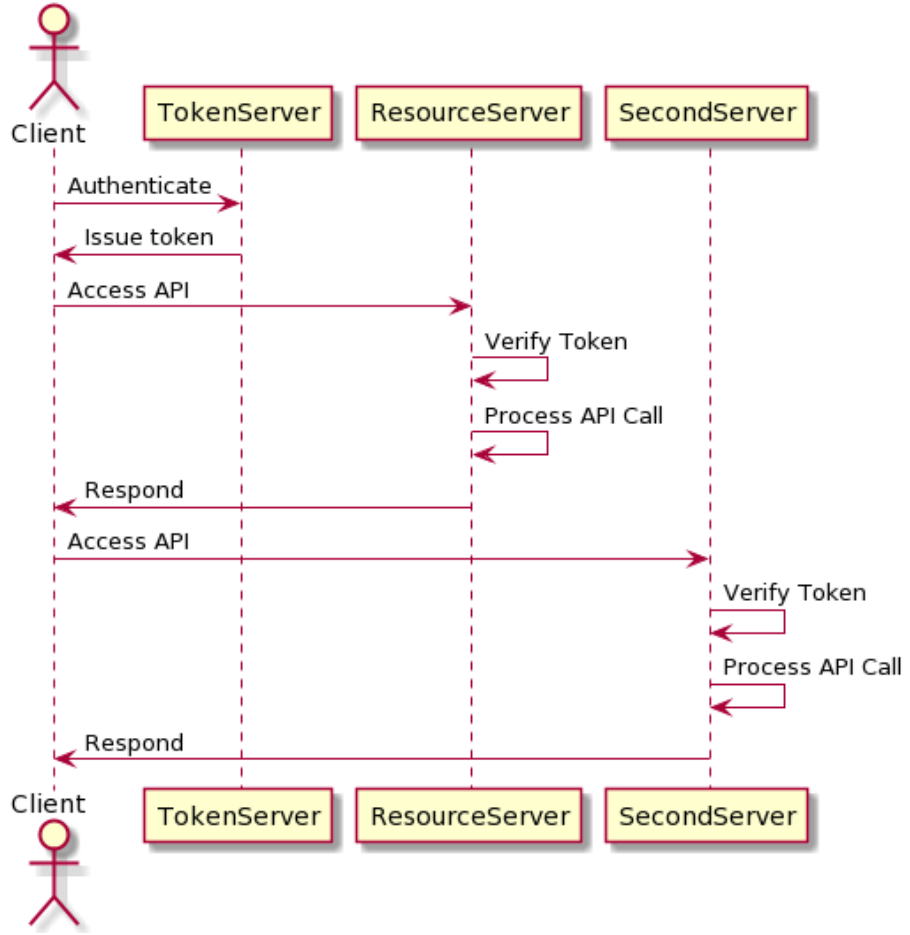


JWT-based Authentication



Session based vs Token based

- Scalability/Micro Service
  - A session often lives on a server or a cluster of servers.
- Some data that needs to be remembered across various parts of a website.



What is JSON Web Token?

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- This information can be verified and trusted because it is digitally signed.
- JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.
- Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens.
- Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties.
- When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

What is the JSON Web Token structure?

- In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:
  - Header
    - The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.
  - Payload
    - Which contains the claims. Claims are statements about an entity (typically, the user) and additional data.
  - Signature
    - The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.
  - Putting all together

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

JSON Web Token structure

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJh
QGdtYWkuY29tIiwiaWF0IjoxNzExM
zQzNzM1LCJleHAiOjE3MTEzNTA5M
zV9.nrKL5p1b2Q_lrs1d2dPvJoIzW5
4-YihoEdCoZhBDbcc
```

Online Decoder: https://jwt.io

Decoded

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

How do JSON Web Tokens work?

- In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned. Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required.
- Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema. The content of the header should look like the following:

```
Authorization: Bearer <token>
```

Spring Boot JWT Example

Spring security

- Spring Security is a framework that enables a programmer to impose security restrictions to Spring-framework–based Web applications through JEE components.
- Its primary area of operation is to handle authentication and authorization at the Web request level as well as the method invocation level.
- The greatest advantage of this framework is that it is powerful yet highly customizable in its implementation.
- Although it follows Spring’s convention over configuration, programmers can choose between default provisions or customizing them according to their needs.

Authentication

- The form of key, Grant the user who have the proper credentials
- Challenges the user to validate credentials (for example, through passwords, answers to security questions, or facial recognition)
- Usually done before authorization. First, we verify the user then check other things.
- For example, Employees in a company are required to authenticate through the network before accessing their company email

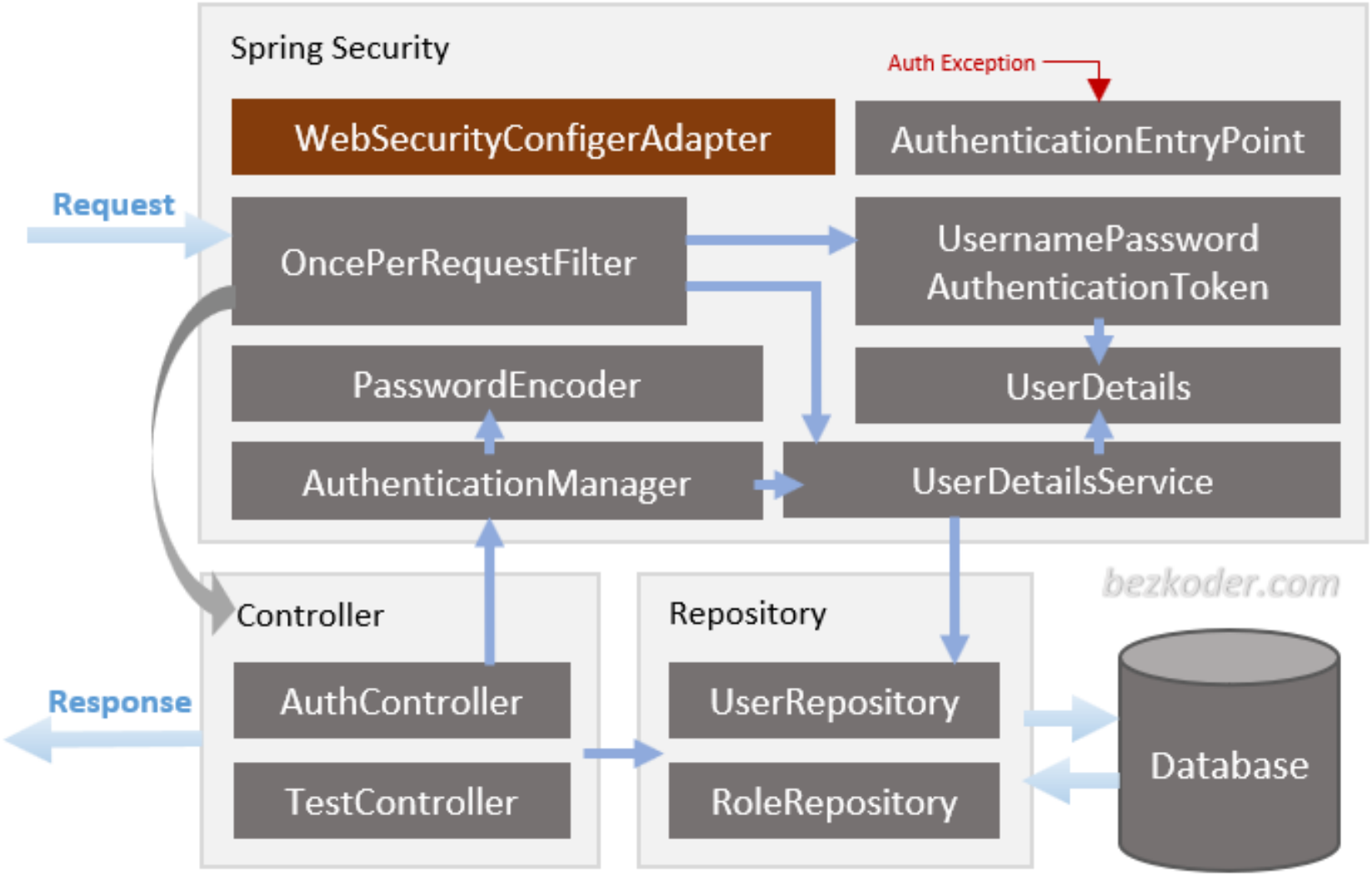
Authorization

- Determines what users can and cannot access
- In the form of permissions. In this term, only denotes validate the access permission.
- Verifies whether access is allowed through policies and rules
- Usually done after successful authentication

Best Practices for Securing REST APIs

- Use HTTPS all the time: With the use of SSL, all the authentication credentials can be cut down to an arbitrarily produced access-token, which uses the HTTP Basic Auth technique.
- Use Hashed Password: hashing of the password is vital to shield RESTful services because even when your password gets compromised by hackers in a hacking attempt, they will not be able to read them out. Various hashing algorithms make this approach a fruitful one. Some of them are MD5, PBKDF2, bcrypt, SHA algorithms, etc.
- Considering OAuth: If the basic auth is implemented to most of the APIs correctly, then it is a great choice, which is more secure also. With the introduction of the OAuth 2.0 authorization framework, all third-party applications get enabled to attain the partial right of entry to HTTP service(s).
- Validating Input Parameter: Security can be well executed of the request parameters get validated in the very beginning, before reaching in the application logic.

Spring Security



Dependencies

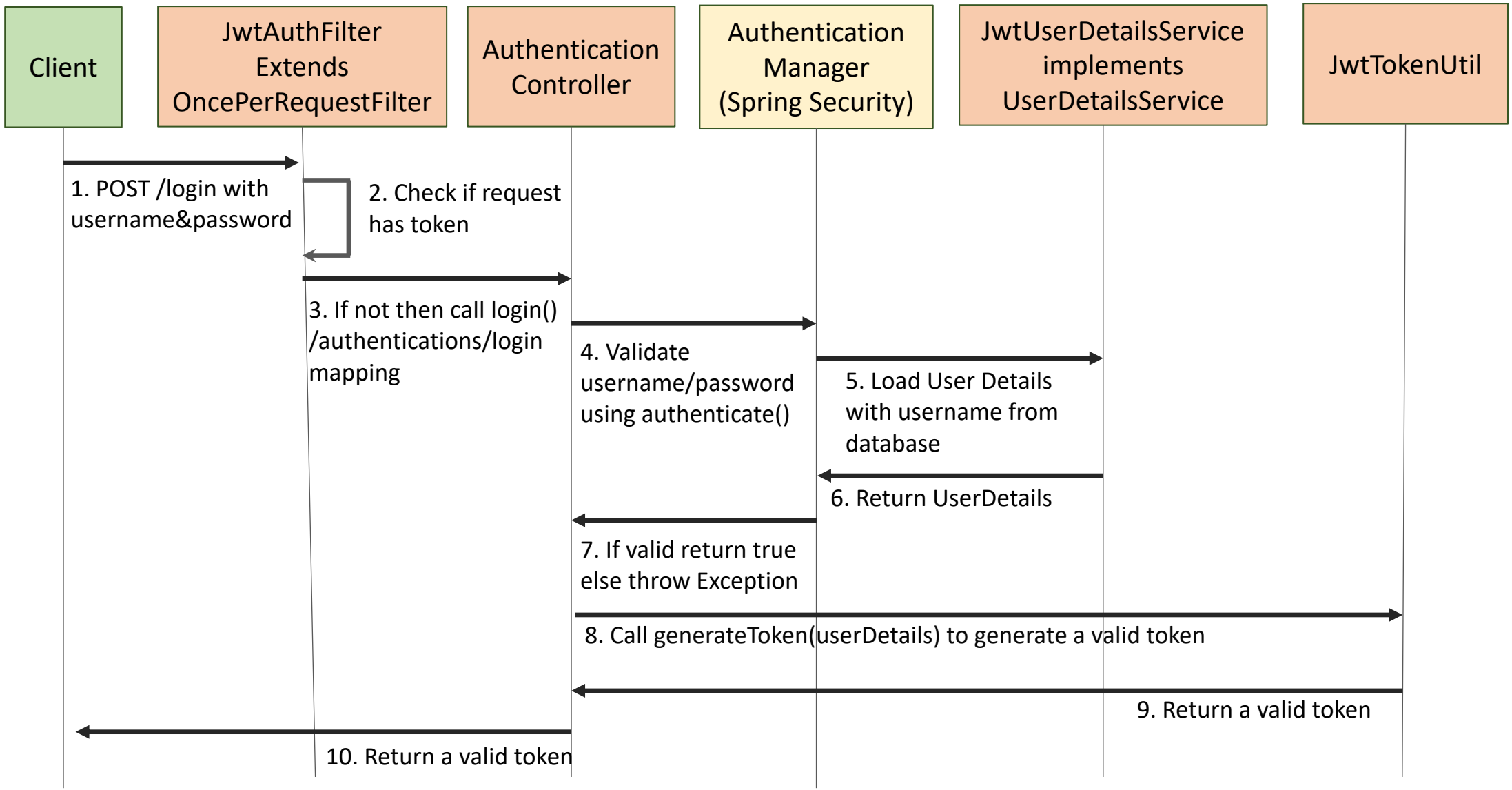
```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
   <version>3.2.2</version>
</dependency>
```

```
<dependency>
   <groupId>de.mkammerer</groupId>
   <artifactId>argon2-jvm</artifactId>
   <version>2.11</version>
</dependency>
```

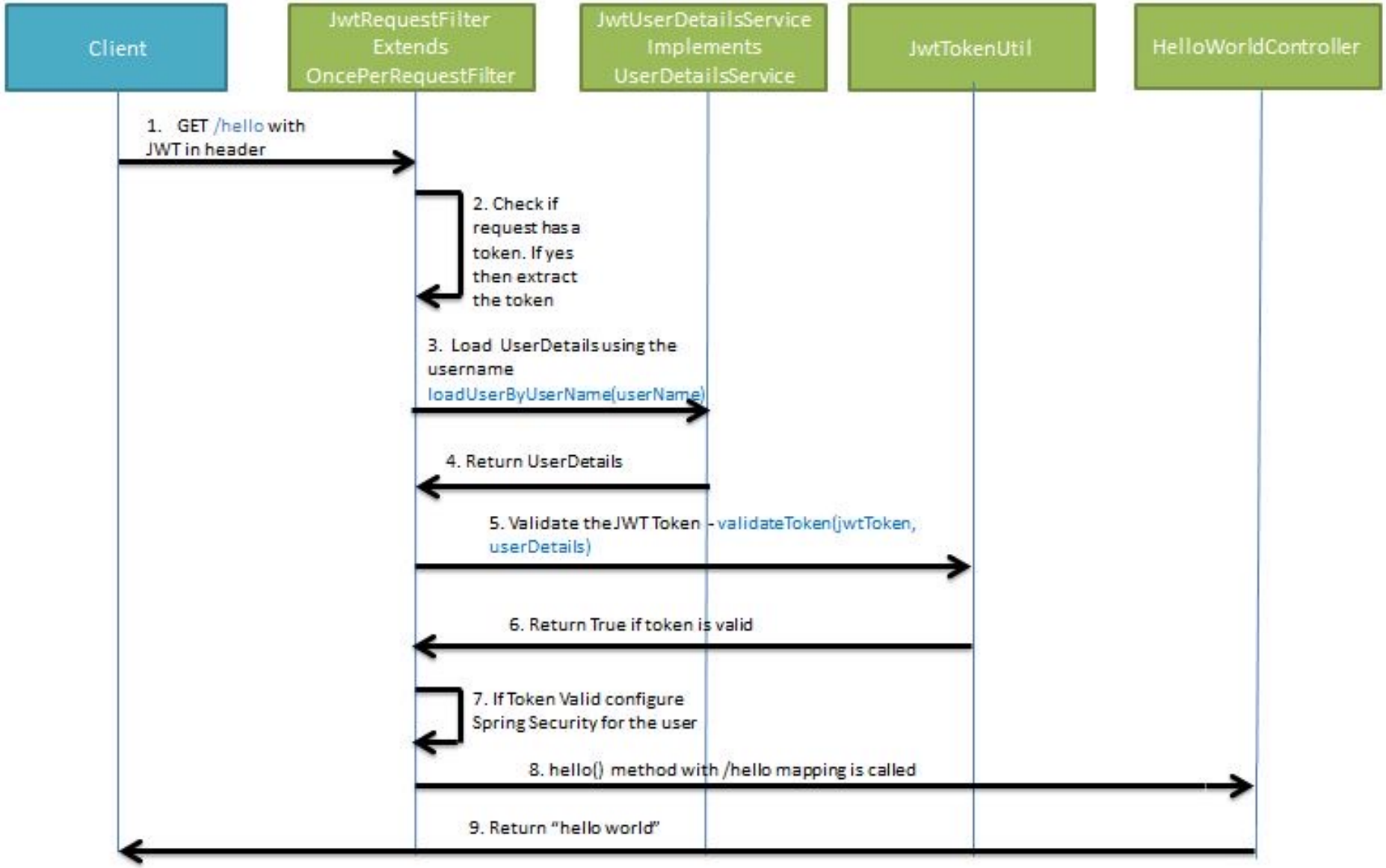
```
<dependency>
   <groupId>org.bouncycastle</groupId>
   <artifactId>bcprov-jdk15on</artifactId>
   <version>1.64</version>
</dependency>
```

```
<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>
   <scope>runtime</scope>
</dependency>
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt-jackson</artifactId>
   <version>0.11.5</version>
   <scope>runtime</scope>
</dependency>
```

Generating JWT – Sequence diagram



Request Resource – Sequence Diagram

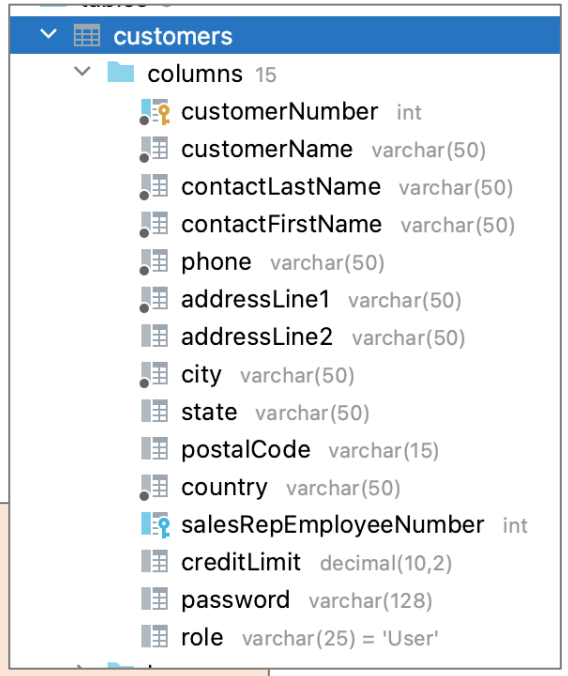


Implementations - Setting

1. Add dependencies
2. Preparing database
3. Edit Customer entity
4. Edit Customer repositoy
5. Create WebSecurityConfig.java

password	role
e2a1c23b6d431b840327ce8233…	USER
$argon2d$v=19$m=16,t=2,p=1…	ADMIN
e2a1c23b6d431b840327ce8233…	USER
$argon2d$v=19$m=16,t=2,p=1…	USER
$argon2d$v=19$m=16,t=2,p=1…	USER

https://argon2.online




```
package sit.int204.classicmodelservice.config;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.csrf(csrf -> csrf.disable())
                .authorizeRequests(authorize -> authorize.requestMatchers("/authentications/**").permitAll()
                        .anyRequest().authenticated()
                )
                .httpBasic(withDefaults());
        return httpSecurity.build();
    }
```

```
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
    @Query("select c from Customer c where concat(c.contactFirstName,' ',c.contactLastName) = :name")
    Customer findByName(String name);
}
```

Implementations (2)

6. Create JwtRequestIUser.java
7. Create Spring Security User AuthUser.java

```
package sit.int204.classicmodelservice.dtos;

@Data
public class JwtRequestUser {
    @NotBlank
    private String userName;
    @Size(min = 8)
    @NotBlank
    private String password;
}
```

```
@Getter
@Setter
public class AuthUser extends User implements Serializable {
    public AuthUser() {
        super("anonymous", "", new ArrayList<GrantedAuthority>());
    }
    public AuthUser(String userName, String password) {
        super(userName, password, new ArrayList<GrantedAuthority>());
    }

    public AuthUser(String userName, String password, Collection<? extends
            GrantedAuthority> authorities) {
        super(userName, password, authorities);
    }
}
```

Implementations (3) - Create UserDetailsService: JwtUserDetailsService.java

```
@Service
public class JwtUserDetailsService implements UserDetailsService {
   @Autowired
   private CustomerRepository customerRepository;
   @Override
   public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {
      Customer customer = customerRepository.findByName(userName);
      if(customer == null) {
         throw new ResponseStatusException(HttpStatus.NOT_FOUND, userName+ " does not exist !!");
      }
      List<GrantedAuthority> roles = new ArrayList<>();
      GrantedAuthority grantedAuthority = new GrantedAuthority() {
         @Override
         public String getAuthority() {
            return customer.getRole();
         }
      };
      roles.add(grantedAuthority);
      UserDetails userDetails = new AuthUser(userName, customer.getPassword(), roles);
      return userDetails;
   }
}
```

Implementations (4) – Create JwtHelper: JwtTokenUtil.java (1/2)

application.properties

```
jwt.secret=N7KgseMPtJ26AEved0ahUKEwj4563eioyFAxUyUGwGHbTODx0Q4dUDCBA
jwt.max-token-interval-hour=2
```

```
@Component
public class JwtTokenUtil implements Serializable {
   @Value("${jwt.secret}")
   private String SECRET_KEY;
   @Value("#{${jwt.max-token-interval-hour}*60*60*1000}")
   private long JWT_TOKEN_VALIDITY;
   SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

   public String getUsernameFromToken(String token) {
      return getClaimFromToken(token, Claims::getSubject);
   }
   public Date getExpirationDateFromToken(String token) {
      return getClaimFromToken(token, Claims::getExpiration);
   }
   public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
     final Claims claims = getAllClaimsFromToken(token);
      return claimsResolver.apply(claims);
   }
   public Claims getAllClaimsFromToken(String token) {
      Claims claims = Jwts.parser().setSigningKey(SECRET_KEY)
            .parseClaimsJws(token).getBody();
      return claims;
   }
```

Implementations (4) – Create JwtHelper: JwtTokenUtil.java (2/2)

```
  private Boolean isTokenExpired(String token) {
      final Date expiration = getExpirationDateFromToken(token);
      return expiration.before(new Date());
   }
   public String generateToken(UserDetails userDetails) {
      Map<String, Object> claims = new HashMap<>();
      claims.put("info#1", "claim-objec 1");
      claims.put("info#2", "claim-objec 2");
      claims.put("info#3", "claim-objec 3");
      return doGenerateToken(claims, userDetails.getUsername());
   }
   private String doGenerateToken(Map<String, Object> claims, String subject) {
      return Jwts.builder().setHeaderParam("typ", "JWT").setClaims(claims).setSubject(subject)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALIDITY))
            .signWith(signatureAlgorithm, SECRET_KEY).compact();
   }
   public Boolean validateToken(String token, UserDetails userDetails) {
      final String username = getUsernameFromToken(token);
      return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
   }
}
```

Implementations (5) - Create AuthenticationController.java (Test-1)

```
@RestController
@RequestMapping("/authentications")
public class AuthenticationController {
    @Autowired
    JwtUserDetailsService jwtUserDetailsService;

    @Autowired
    JwtTokenUtil jwtTokenUtil;

    @PostMapping("/login")
    public ResponseEntity<Object> login(@RequestBody @Valid JwtRequestUser jwtRequestUser) {
        UserDetails userDetails = jwtUserDetailsService.loadUserByUserName(jwtRequestUser.getUserName());
        String token = jwtTokenUtil.generateToken(userDetails);
        return ResponseEntity.ok(token);
    }
}
```

Implementations (5) - Create AuthenticationController.java (Test-2)

```
@GetMapping("/validate-token")
public ResponseEntity<Object> validateToken(@RequestHeader("Authorization") String requestTokenHeader) {
   Claims claims = null;
   String jwtToken = null;
   if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
      jwtToken = requestTokenHeader.substring(7);
      try {
           claims = jwtTokenUtil.getAllClaimsFromToken(jwtToken);
      } catch (IllegalArgumentException e) {
         System.out.println("Unable to get JWT Token");
      } catch (ExpiredJwtException e) {
         System.out.println("JWT Token has expired");
      }
   } else {
      throw new ResponseStatusException(HttpStatus.EXPECTATION_FAILED,
            "JWT Token does not begin with Bearer String");
   }
   return ResponseEntity.ok(claims);
}
```

JwtAuthFilter (1/2)

```
@Component
public class JwtAuthFilter extends OncePerRequestFilter {
   @Autowired    private JwtUserDetailsService jwtUserDetailsService;
   @Autowired.   private JwtTokenUtil jwtTokenUtil;
   @Override
   protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
         throws ServletException, IOException {
      final String requestTokenHeader = request.getHeader("Authorization");
      String username = null;
      String jwtToken = null;
      if (requestTokenHeader != null) {
         if (requestTokenHeader.startsWith("Bearer ")) {
            jwtToken = requestTokenHeader.substring(7);
            try {
               username = jwtTokenUtil.getUsernameFromToken(jwtToken);
            } catch (IllegalArgumentException e) {
               throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
            } catch (ExpiredJwtException e) {
               throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
            }
         } else {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "JWT Token does not begin with Bearer String");
         }
      }
```

JwtAuthFilter.java (2/2)

```
        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

            UserDetails userDetails = this.jwtUserDetailsService.loadUserByUsername(username);

            if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new
                 UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                usernamePasswordAuthenticationToken
                        .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }
        chain.doFilter(request, response);
    }
}
```

Authentication with Spring AuthenticationManager (1/2)

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
   @Autowired private JwtUserDetailsService jwtUserDetailsService;
   @Autowired  private JwtAuthFilter jwtAuthFilter;
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
      httpSecurity.csrf(csrf -> csrf.disable())
            .authorizeRequests(authorize -> authorize
                  .requestMatchers("/authentications/login").permitAll()
                  .requestMatchers("/authentications/validate-token").hasAuthority("ADMIN")
                  .anyRequest().authenticated()
            )
            .httpBasic(withDefaults());
      httpSecurity.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
      return httpSecurity.build();
   }
```

Authentication with Spring AuthenticationManager (2/2)

```
@Bean
  public PasswordEncoder passwordEncoder() {
     return Argon2PasswordEncoder.defaultsForSpringSecurity_v5_8();
  }

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

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