JWT HANDS ON

Superset ID : 6384831

Name : Mohana Priya N

E-mail : mohanapriya.2205056@srec.ac.in

Mandatory Question:

1) Hands on 3: Create authentication service that returns JWT

Solution:

//AuthenticationController.java

```
package com.cognizant.springsecurity.controller;
import java.util.Base64;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RestController;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
@RestController
public class AuthenticationController {
    @GetMapping("/authenticate")
    public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {
        String user = getUser(authHeader);
        String token = generateJwt(user);
        Map<String, String> map = new HashMap<>();
        map.put("token", token);
        return map;
    }
    private String getUser(String authHeader) {
```

```
        String base64Credentials = authHeader.substring("Basic ".length());
        byte[] decoded = Base64.getDecoder().decode(base64Credentials);
        String decodedString = new String(decoded);
        return decodedString.split(":")[0];
    }
    private String generateJwt(String user) {
        return Jwts.builder()
                .setSubject(user)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + 1200000)) // 20 mins
                .signWith(SignatureAlgorithm.HS256, "secretkey")
                .compact();
    }
}
```

Other Questions:

2) Hands on 1: Securing RESTful Web Services with Spring Security

Solution:

//pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

3) Hands on 2: Creating users and roles in Spring Security

Solution:

//SecurityConfig.java

```
package com.cognizant.springsecurity.security;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
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;
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public AuthenticationManager authManager(HttpSecurity http, PasswordEncoder encoder) throws
Exception {
        return http.getSharedObject(AuthenticationManagerBuilder.class)
            .inMemoryAuthentication()
            .withUser("user").password(encoder.encode("pwd")).roles("USER")
            .and()
            .withUser("admin").password(encoder.encode("pwd")).roles("ADMIN")
            .and()
            .passwordEncoder(encoder)
            .and().build();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
```

```
        return new BCryptPasswordEncoder();
    }
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
            .antMatchers("/authenticate").permitAll()
            .anyRequest().authenticated()
            .and().httpBasic();
        return http.build();
    }
}
```

4) Hands on 4: Create authentication controller and configure it in SecurityConfig

Solution:

//JwtAuthorizationFilter.java

```
package com.cognizant.springsecurity.security;
import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.JwtException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.authentication.AuthenticationManager;
```

```
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;
public class JwtAuthorizationFilter extends BasicAuthenticationFilter {
   private static final Logger LOGGER = LoggerFactory.getLogger(JwtAuthorizationFilter.class);
   public JwtAuthorizationFilter(AuthenticationManager authenticationManager) {
      super(authenticationManager);
   }
   @Override
   protected void doFilterInternal(HttpServletRequest request,
                                   HttpServletResponse response,
                                   FilterChain chain)
         throws IOException, ServletException {
      String header = request.getHeader("Authorization");
      if (header == null || !header.startsWith("Bearer ")) {
         chain.doFilter(request, response);
         return;
      }
      UsernamePasswordAuthenticationToken auth = getAuthentication(header);
      SecurityContextHolder.getContext().setAuthentication(auth);
      chain.doFilter(request, response);
   }
   private UsernamePasswordAuthenticationToken getAuthentication(String header) {
      try {
         String token = header.replace("Bearer ", "");
         Jws<Claims> parsedToken = Jwts.parser()
            .setSigningKey("secretkey")
            .parseClaimsJws(token);
         String user = parsedToken.getBody().getSubject();
         if (user != null) {
```

```
            return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());
        }
      } catch (JwtException e) {
        return null;
      }
      return null;
    }
}
```

5) Hands on 5: Read Authorization header and decode the username and password

Solution:

//SecurityConfig.java

```
package com.cognizant.springsecurity.security;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder
;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
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;
@Configuration
@EnableWebSecurity
public class SecurityConfig {
  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }
```

```
    @Bean
    public AuthenticationManager authManager(HttpSecurity http, PasswordEncoder encoder) throws
Exception {
        return http.getSharedObject(AuthenticationManagerBuilder.class)
            .inMemoryAuthentication()
            .withUser("user").password(encoder.encode("pwd")).roles("USER")
            .and()
            .withUser("admin").password(encoder.encode("pwd")).roles("ADMIN")
            .and()
            .passwordEncoder(encoder)
            .and().build();
    }
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
            .antMatchers("/authenticate").permitAll()
            .anyRequest().authenticated()
            .and()
            .httpBasic();
        return http.build();
    }
}
```

//AuthenticationController.java

```
package com.cognizant.springsecurity.controller;
import java.util.Base64;
import java.util.HashMap;
import java.util.Map;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class AuthenticationController {
   @GetMapping("/authenticate")
   public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {
      System.out.println("Authorization Header: " + authHeader);
      String username = extractUsername(authHeader);
      System.out.println("Decoded username: " + username);
      Map<String, String> response = new HashMap<>();
      response.put("token", "");
      return response;
   }
   private String extractUsername(String authHeader) {
      String base64Credentials = authHeader.substring("Basic ".length());
      byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);
      String decoded = new String(decodedBytes);
      return decoded.split(":")[0];
   }}
```

6) Hands on 6: Generate token based on the user

Solution:

//AuthenticationController.java

```
package com.cognizant.springsecurity.controller;
import java.util.Base64;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RestController;
import io.jsonwebtoken.Jwts;
```

```
import io.jsonwebtoken.SignatureAlgorithm;
@RestController
public class AuthenticationController {
   @GetMapping("/authenticate")
   public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {
      String user = extractUserFromHeader(authHeader);
      String token = generateJwtToken(user);
      Map<String, String> response = new HashMap<>();
      response.put("token", token);
      return response;
   }
   private String extractUserFromHeader(String authHeader) {
      String base64Credentials = authHeader.substring("Basic ".length());
      byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);
      String decoded = new String(decodedBytes);
      return decoded.split(":")[0];
   }
   private String generateJwtToken(String username) {
      return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 20 * 60 * 1000))
            .signWith(SignatureAlgorithm.HS256, "secretkey")
            .compact();
   }}
```

7) Hands on 7: Authorize based on JWT

Solution:

//JwtAuthorizationFilter.java

```
package com.cognizant.springsecurity.security;
```

```
import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.JwtException;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;
public class JwtAuthorizationFilter extends BasicAuthenticationFilter {
   public JwtAuthorizationFilter(AuthenticationManager authenticationManager) {
      super(authenticationManager);
   }
   @Override
   protected void doFilterInternal(HttpServletRequest request,
                                   HttpServletResponse response,
                                   FilterChain chain)
         throws IOException, ServletException {
      String header = request.getHeader("Authorization");
      if (header == null || !header.startsWith("Bearer ")) {
         chain.doFilter(request, response);
         return;
      }
      UsernamePasswordAuthenticationToken authentication = getAuthentication(header);
      SecurityContextHolder.getContext().setAuthentication(authentication);
```

```
        chain.doFilter(request, response);
    }
    private UsernamePasswordAuthenticationToken getAuthentication(String header) {
        try {
            String token = header.replace("Bearer ", "");
            Jws<Claims> parsedToken = Jwts.parser()
                    .setSigningKey("secretkey")
                    .parseClaimsJws(token);
            String username = parsedToken.getBody().getSubject();
            if (username != null) {
                return new UsernamePasswordAuthenticationToken(username, null, new ArrayList<>());
            }
        } catch (JwtException e) {
            return null;
        }
        return null;
    }
}
```

//SecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    AuthenticationManager authManager = authManager(http, passwordEncoder());
    http.csrf().disable()
        .authorizeRequests()
        .antMatchers("/authenticate").permitAll()
        .anyRequest().authenticated()
        .and()
        .addFilter(new JwtAuthorizationFilter(authManager)); // Add JWT Filter
    return http.build();
}
```