Assignment 1:
Backend Development with Spring Boot and MySQL
Create a UserController

```
import com.ey.springboot3security.entity.AuthRequest;
import com.ey.springboot3security.entity.UserInfo;
import com.ey.springboot3security.service.JwtService;
import com.ey.springboot3security.service.UserInfoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(&quot;/auth&quot;)
public class UserController {

    @Autowired
    private UserInfoService service;

    @Autowired
    private JwtService jwtService;

    @Autowired
    private AuthenticationManager authenticationManager;

    @GetMapping(&quot;/welcome&quot;)
    public String welcome() {
        return &quot;Welcome this endpoint is not secure&quot;;
    }

    @PostMapping(&quot;/addNewUser&quot;)
    public String addNewUser(@RequestBody UserInfo userInfo) {
        return service.addUser(userInfo);
    }

    @GetMapping(&quot;/user/userProfile&quot;)
    @PreAuthorize(&quot;hasAuthority('ROLE_USER')&quot;)
    public String userProfile() {
```

```
        return &amp;quot;Welcome to User Profile&amp;quot;;
    }

    @GetMapping(&amp;quot;/admin/adminProfile&amp;quot;)
    @PreAuthorize(&amp;quot;hasAuthority('ROLE_ADMIN')&amp;quot;)
    public String adminProfile() {
        return &amp;quot;Welcome to Admin Profile&amp;quot;;
    }

    @PostMapping(&amp;quot;/generateToken&amp;quot;)
    public String authenticateAndGetToken(@RequestBody AuthRequest authRequest) {
        Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(authRequest.getUsername(),
authRequest.getPassword()));
        if (authentication.isAuthenticated()) {
            return jwtService.generateToken(authRequest.getUsername());
        } else {
            throw new UsernameNotFoundException(&amp;quot;invalid user request !&amp;quot;);
        }
    }

}
```

Create a SecurityConfig Class

```
import com.ey.springboot3security.filter.JwtAuthFilter;
import com.ey.springboot3security.service.UserInfoService;
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.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.Authenticati
onConfiguration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecu
rity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
```

```
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter
;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    @Autowired
    private JwtAuthFilter authFilter;

    // User Creation
    @Bean
    public UserDetailsService userDetailsService() {
        return new UserInfoService();
    }

    // Configuring HttpSecurity
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http.csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(auth -> auth.requestMatchers("/auth/welcome",
"/auth/addNewUser", "/auth/generateToken").permitAll())
                .authorizeHttpRequests(auth ->
auth.requestMatchers("/auth/user/**").authenticated())
                .authorizeHttpRequests(auth ->
auth.requestMatchers("/auth/admin/**").authenticated())
                .sessionManagement(sess ->
sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider())
                .addFilterBefore(authFilter, UsernamePasswordAuthenticationFilter.class)
                .build();
    }

    // Password Encoding
    @Bean
    public PasswordEncoder passwordEncoder() {
```

```
        return new BCryptPasswordEncoder();
    }

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

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


}
```

Create Entity Classes

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class UserInfo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String email;
    private String password;
```

```
    private String roles;

}
```

Create Filter Class

```
import com.ey.springboot3security.service.JwtService;
import com.ey.springboot3security.service.UserInfoService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
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.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

// This class helps us to validate the generated jwt token
@Component
public class JwtAuthFilter extends OncePerRequestFilter {

    @Autowired
    private JwtService jwtService;

    @Autowired
    private UserInfoService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain) throws ServletException, IOException {
        String authHeader = request.getHeader(&amp;quot;Authorization&amp;quot;);
        String token = null;
        String username = null;
        if (authHeader != null &amp;amp;&amp;amp; authHeader.startsWith(&amp;quot;Bearer &amp;quot;)) {
            token = authHeader.substring(7);
            username = jwtService.extractUsername(token);
```

```
        }

        if (username != null &amp;amp;&amp;amp;
SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userDetailsService.loadUserByUsername(username);
            if (jwtService.validateToken(token, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Create a Repository Interface

```
import com.ey.springboot3security.entity.UserInfo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface UserInfoRepository extends JpaRepository&amp;lt;UserInfo, Integer&amp;gt; {
    Optional&amp;lt;UserInfo&amp;gt; findByName(String username);
}
```

Create Service Classes

```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

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

```
import java.util.function.Function;

@Component
public class JwtService {

    public static final String SECRET =
&amp;quot;5367566B59703373367639792F423F4528482B4D6251655468576D5A71347437
&amp;quot;;
    public String generateToken(String userName) {
        Map&amp;lt;String, Object&amp;gt; claims = new HashMap&amp;lt;&amp;gt;();
        return createToken(claims, userName);
    }

    private String createToken(Map&amp;lt;String, Object&amp;gt; claims, String userName) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userName)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 30))
                .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
    }

    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 &amp;lt;T&amp;gt; T extractClaim(String token, Function&amp;lt;Claims, T&amp;gt;
claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
```

```
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

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

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


}
```