**Spring Security**

**Part 1 Authentication and Authorization**

Add UserEntity in Entity/DAO Package

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor

@Entity

@Table(name = "\_user")

public class User implements UserDetails {

@Id

@GeneratedValue

private Integer id;

private String firstname;

private String lastname;

private String email;

private String password;

@Enumerated(EnumType.STRING)

private Role role;

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return List.of(new SimpleGrantedAuthority(role.name()));

}

@Override

public String getPassword() {

return password;

}

@Override

public String getUsername() {

return email;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

}

Add User Repository in Repository Package

public interface UserRepository extends JpaRepository<User, Integer> {

Optional<User> findByEmail(String email);

}

Add Roles Enum in Utils Package

public enum Role {

USER,

ADMIN

}

Add Application Configuration in Config Package

@Configuration

@EnableWebSecurity

@RequiredArgsConstructor

public class SecurityConfiguration {

private final JwtAuthenticationFilter jwtAuthFilter;

private final AuthenticationProvider authenticationProvider;

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.csrf()

.disable()

.authorizeHttpRequests()

.requestMatchers("/api/v1/auth/\*\*")

.permitAll()

.anyRequest()

.authenticated()

.and()

.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authenticationProvider(authenticationProvider)

.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

return http.build();

}

}

Add Application Configuration in Config Package

@Configuration

@RequiredArgsConstructor

public class ApplicationConfig {

private final UserRepository repository;

@Bean

public UserDetailsService userDetailsService() {

return username -> repository.findByEmail(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found"));

}

@Bean

public AuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(userDetailsService());

authProvider.setPasswordEncoder(passwordEncoder());

return authProvider;

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

return config.getAuthenticationManager();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

Add Authentication Controller in Config Package

@RestController

@RequestMapping("/api/v1/auth")

@RequiredArgsConstructor

public class AuthenticationController {

private final AuthenticationService service;

@PostMapping("/register")

public ResponseEntity<AuthenticationResponse> register(

@RequestBody RegisterRequest request

) {

return ResponseEntity.ok(service.register(request));

}

@PostMapping("/authenticate")

public ResponseEntity<AuthenticationResponse> authenticate(

@RequestBody AuthenticationRequest request

) {

return ResponseEntity.ok(service.authenticate(request));

}

}

Add Authentication Request in Config Package

@Data

@Builder

@AllArgsConstructor

@NoArgsConstructor

public class AuthenticationRequest {

private String email;

String password;

}

Add Authentication Response in Config Package

@Data

@Builder

@AllArgsConstructor

@NoArgsConstructor

public class AuthenticationResponse {

private String token;

}

Add Authentication Service in Config Package

@Service

@RequiredArgsConstructor

public class AuthenticationService {

private final UserRepository repository;

private final PasswordEncoder passwordEncoder;

private final JwtService jwtService;

private final AuthenticationManager authenticationManager;

public AuthenticationResponse register(RegisterRequest request) {

var user = User.builder()

.firstname(request.getFirstname())

.lastname(request.getLastname())

.email(request.getEmail())

.password(passwordEncoder.encode(request.getPassword()))

.role(Role.USER)

.build();

repository.save(user);

var jwtToken = jwtService.generateToken(user);

return AuthenticationResponse.builder()

.token(jwtToken)

.build();

}

public AuthenticationResponse authenticate(AuthenticationRequest request) {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(

request.getEmail(),

request.getPassword()

)

);

var user = repository.findByEmail(request.getEmail())

.orElseThrow();

var jwtToken = jwtService.generateToken(user);

return AuthenticationResponse.builder()

.token(jwtToken)

.build();

}

}

Add Registry Request in Config Package

@Data

@Builder

@AllArgsConstructor

@NoArgsConstructor

public class RegisterRequest {

private String firstname;

private String lastname;

private String email;

private String password;

}

**Part 1 J W Token**

Add JWT Authentication Filter in Auth Package

@Component

@RequiredArgsConstructor

public class JwtAuthenticationFilter extends OncePerRequestFilter {

private final JwtService jwtService;

private final UserDetailsService userDetailsService;

@Override

protected void doFilterInternal(

@NonNull HttpServletRequest request,

@NonNull HttpServletResponse response,

@NonNull FilterChain filterChain

) throws ServletException, IOException {

final String authHeader = request.getHeader("Authorization");

final String jwt;

final String userEmail;

if (authHeader == null ||!authHeader.startsWith("Bearer ")) {

filterChain.doFilter(request, response);

return;

}

jwt = authHeader.substring(7);

userEmail = jwtService.extractUsername(jwt);

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

UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);

if (jwtService.isTokenValid(jwt, userDetails)) {

UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(

userDetails,

null,

userDetails.getAuthorities()

);

authToken.setDetails(

new WebAuthenticationDetailsSource().buildDetails(request)

);

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}

Add JWT Service in Auth Package

@Service

public class JwtService {

private static final String SECRET\_KEY = "404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

public String generateToken(UserDetails userDetails) {

return generateToken(new HashMap<>(), userDetails);

}

public String generateToken(

Map<String, Object> extraClaims,

UserDetails userDetails

) {

return Jwts

.builder()

.setClaims(extraClaims)

.setSubject(userDetails.getUsername())

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 24))

.signWith(getSignInKey(), SignatureAlgorithm.HS256)

.compact();

}

public boolean isTokenValid(String token, UserDetails userDetails) {

final String username = extractUsername(token);

return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);

}

private boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

private Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

private Claims extractAllClaims(String token) {

return Jwts

.parserBuilder()

.setSigningKey(getSignInKey())

.build()

.parseClaimsJws(token)

.getBody();

}

private Key getSignInKey() {

byte[] keyBytes = Decoders.BASE64.decode(SECRET\_KEY);

return Keys.hmacShaKeyFor(keyBytes);

}

}