**Отчёт: Добавление аутентификации с JWT в Spring Boot**

**Цель работы**

Настроить аутентификацию пользователей в Spring Boot приложении с использованием JSON Web Token (JWT). Реализовать защиту API, используя Spring Security.

**Этапы работы**

**1. Добавление зависимостей в build.gradle**

Для работы с JWT добавлены необходимые зависимости:

implementation 'io.jsonwebtoken:jjwt-api:0.11.5'  
implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'  
implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'

**2. Настройка фильтра JWT**

Создан фильтр JwtFilter, который перехватывает запросы и проверяет наличие JWT в заголовке:

@Component  
public class JwtFilter extends OncePerRequestFilter {  
  
 private final JwtUtil jwtUtil;  
 private final UserDetailsService userDetailsService;  
  
 public JwtFilter(JwtUtil jwtUtil, UserDetailsService userDetailsService) {  
 this.jwtUtil = jwtUtil;  
 this.userDetailsService = userDetailsService;  
 }  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request,  
 HttpServletResponse response,  
 FilterChain chain) throws ServletException, IOException {  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader == null || !authHeader.startsWith("Bearer ")) {  
 chain.doFilter(request, response);  
 return;  
 }  
  
 String token = authHeader.substring(7);  
 String email = jwtUtil.extractEmail(token);  
  
 if (email != null && SecurityContextHolder.*getContext*().getAuthentication() == null) {  
 UserDetails userDetails = userDetailsService.loadUserByUsername(email);  
  
 if (jwtUtil.validateToken(token)) {  
 UsernamePasswordAuthenticationToken authentication =  
 new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());  
 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 }  
 }  
 chain.doFilter(request, response);  
 }  
}

**3. Реализация утилиты JwtService**

Сервис для генерации и проверки токенов:

@Component  
public class JwtUtil {  
 private static final String *SECRET\_KEY* = "SuperSecretKeyForJWTTokenSuperSecretKey";  
 private static final long *EXPIRATION\_TIME* = 86400000; // 24 часа  
  
 private final Key key = Keys.*hmacShaKeyFor*(*SECRET\_KEY*.getBytes());  
  
 // Генерация токена  
 public String generateToken(String email) {  
 return Jwts.*builder*()  
 .setSubject(email)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + *EXPIRATION\_TIME*))  
 .signWith(key, SignatureAlgorithm.*HS256*)  
 .compact();  
 }  
  
 // Получение email из токена  
 public String extractEmail(String token) {  
 return Jwts.*parserBuilder*()  
 .setSigningKey(key)  
 .build()  
 .parseClaimsJws(token)  
 .getBody()  
 .getSubject();  
 }  
  
 // Проверка токена  
 public boolean validateToken(String token) {  
 try {  
 Jwts.*parserBuilder*().setSigningKey(key).build().parseClaimsJws(token);  
 return true;  
 } catch (JwtException e) {  
 return false;  
 }  
 }  
}

**4. Настройка Spring Security**

В файле SecurityConfig добавлена проверка JWT-токенов:

@Bean  
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http  
 .csrf(csrf -> csrf.disable())  
 .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.*STATELESS*))  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/login").permitAll()  
 .requestMatchers( "/register", "/css/\*\*", "/js/\*\*", "/images/\*\*","login.html","register.html","/error","error.html","/dashboard","dashboard.html").permitAll() // Разрешаем доступ  
 .requestMatchers("/api/\*\*").authenticated() // Все пользователи должны быть авторизованы  
 .anyRequest().authenticated()  
 )  
 .formLogin(form -> form  
 .loginPage("/login") // Указываем страницу входа  
 .permitAll()  
 .defaultSuccessUrl("/dashboard")  
 )  
 .logout((logout) -> logout.permitAll())  
 .sessionManagement(session -> session  
 .sessionCreationPolicy(SessionCreationPolicy.*ALWAYS*) // Открываем сессию для каждого запроса  
 )  
 .exceptionHandling(ex -> ex  
 .accessDeniedPage("/error") // Перенаправление на кастомную страницу ошибки  
 )  
 .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)  
 .build();  
}

**5. Создание контроллера аутентификации**

Контроллер для входа пользователей и генерации токенов:

@PostMapping("/login")  
public ResponseEntity<?> loginUser(@RequestParam String email, @RequestParam String password) {  
 Optional<User> userOpt = userRepository.findByEmail(email);  
  
 if (userOpt.isEmpty()) {  
 return ResponseEntity.*status*(401).body("Invalid email or password");  
 }  
  
 User user = userOpt.get();  
  
 if (!passwordEncoder.matches(password, user.getPassword())) {  
 return ResponseEntity.*status*(401).body("Invalid email or password");  
 }  
  
 String token = jwtUtil.generateToken(user.getEmail());  
 return ResponseEntity.*ok*(token);  
}

**6. Тестирование API с Postman**

Примеры запросов:

**Регистрация пользователя:**

POST http://localhost:8081/api/auth/register

{

"username": "testuser",

"password": "password123",

"email": "test@example.com"

}

**Авторизация:**

POST http://localhost:8081/api/auth/login

{

"username": "testuser",

"password": "password123"

}

Ответ:

{

"token": "eyJhbGciOiJIUzI1NiIsInR5..."

}

**Доступ к защищённому API:**

GET http://localhost:8081/api/protected

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5...

**Вывод**

Настроена аутентификация с JWT, защищены маршруты API. Теперь пользователи могут получать токены и использовать их для доступа к защищённым ресурсам.