Let’s create a Spring Boot project demonstrating JWT (JSON Web Token) authentication. This example will show how to secure your REST APIs using JWT in a Spring Boot application.

**Overview**

We will:

1. Set up a new Spring Boot project.  
2. Add the required dependencies.  
3. Configure application properties.  
4. Create models and repositories.  
5. Implement JWT utility classes.  
6. Configure Spring Security.  
7. Create authentication controllers.  
8. Secure REST endpoints.  
9. Provide Github Repository.  
10. Test the application and provide Postman Collection for testing.

**Step 1: Set Up a New Spring Boot Project**

**Using Spring Initializr:**

1. Navigate to [Spring Initializr](https://start.spring.io/).  
2. **Project**: Maven Project  
3. **Language**: Java  
4. **Spring Boot**: Choose the latest stable version (e.g., 2.7.x or 3.x).  
5. **Project Metadata**:

* **Group**: com.example
* **Artifact**: jwt-demo

6. **Packaging**: Jar  
7. **Java**: 11 or higher  
8. **Dependencies**:

* **Spring Web**
* **Spring Security**
* **Spring Data JPA**
* **H2 Database** (for in-memory testing)
* **Lombok** (optional, for reducing boilerplate code)

9. Click **Generate** to download the project.  
10. Extract the project and open it in your favorite IDE (e.g., IntelliJ IDEA, Eclipse, VS Code).

**Step 2: Add Required Dependencies**

Open the pom.xml file and add the following dependencies for JWT support to the existing dependencies:

<dependencies>  
 <!-- Existing dependencies -->  
   
 <!-- JWT Dependencies -->  
 <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> <!-- or jjwt-gson, jjwt-orgjson -->  
 <version>0.11.5</version>  
 <scope>runtime</scope>  
 </dependency>  
</dependencies>

**Note**: Adjust the versions if newer versions are available.

**Step 3: Configure Application Properties**

In src/main/resources/application.properties, configure the H2 database and JWT properties:

# H2 Database Configuration  
spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true  
# Hibernate Configuration  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update  
# JWT Configuration  
jwt.secret=your\_secret\_key\_here  
jwt.expiration=3600000

**Replace** your\_secret\_key\_here with a strong secret key. According to the JWT specification, the key size for HS256 (HMAC with SHA-256) must be at least 256 bits (32 bytes) long and should contain **only** letters and numbers. A sample secret key you can use is: thisIsMysecregtfrdesww233eggtffeeddgkjjhhtdhttebd54ndhdhfhhhshs8877465sbbdd

**Step 4: Create Models and Repositories**

**4.1. User Entity**

Create a User model in com.example.jwt\_demo.model package:

package com.example.jwt\_demo.model;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "users")  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 @Column(unique = true)  
 private String username;  
 private String password;  
}

**4.2. User Repository**

Create a UserRepository interface in com.example.jwt\_demo.repository package:

package com.example.jwt\_demo.repository;  
  
import com.example.jwt\_demo.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
 User findByUsername(String username);  
 boolean existsByUsername(String username);  
}

**Step 5: Implement JWT Utility Classes**

**5.1. JWT Utility Class**

Create a JwtUtil class in com.example.jwt\_demo.security package:

package com.example.jwt\_demo.security;  
  
import io.jsonwebtoken.\*;  
import io.jsonwebtoken.security.Keys;  
import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
import javax.crypto.SecretKey;  
import java.nio.charset.StandardCharsets;  
import java.util.Date;  
@Component  
public class JwtUtil {  
 @Value("${jwt.secret}")  
 private String jwtSecret;  
 @Value("${jwt.expiration}")  
 private int jwtExpirationMs;  
 private SecretKey key;  
 // Initializes the key after the class is instantiated and the jwtSecret is injected,   
 // preventing the repeated creation of the key and enhancing performance  
 @PostConstruct  
 public void init() {  
 this.key = Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardCharsets.UTF\_8));  
 }  
 // Generate JWT token  
 public String generateToken(String username) {  
 return Jwts.builder()  
 .setSubject(username)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))  
 .signWith(key, SignatureAlgorithm.HS256)  
 .compact();  
 }  
 // Get username from JWT token  
 public String getUsernameFromToken(String token) {  
 return Jwts.parserBuilder()  
 .setSigningKey(key).build()  
 .parseClaimsJws(token)  
 .getBody()  
 .getSubject();  
 }  
 // Validate JWT token  
 public boolean validateJwtToken(String token) {  
 try {  
 Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);  
 return true;  
 } catch (SecurityException e) {  
 System.out.println("Invalid JWT signature: " + e.getMessage());  
 } catch (MalformedJwtException e) {  
 System.out.println("Invalid JWT token: " + e.getMessage());  
 } catch (ExpiredJwtException e) {  
 System.out.println("JWT token is expired: " + e.getMessage());  
 } catch (UnsupportedJwtException e) {  
 System.out.println("JWT token is unsupported: " + e.getMessage());  
 } catch (IllegalArgumentException e) {  
 System.out.println("JWT claims string is empty: " + e.getMessage());  
 }  
 return false;  
 }  
}

**Step 6: Configure Spring Security**

**6.1. UserDetailsService Implementation**

Create CustomUserDetailsService in com.example.jwt\_demo.service package:

package com.example.jwt\_demo.service;  
  
import com.example.jwt\_demo.model.User;  
import com.example.jwt\_demo.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.\*;  
import org.springframework.stereotype.Service;  
import java.util.Collections;  
@Service  
public class CustomUserDetailsService implements UserDetailsService {  
 @Autowired  
 private UserRepository userRepository;  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username);  
 if (user == null) {  
 throw new UsernameNotFoundException("User Not Found with username: " + username);  
 }  
 return new org.springframework.security.core.userdetails.User(  
 user.getUsername(),  
 user.getPassword(),  
 Collections.emptyList()  
 );  
 }  
}

**6.2. Authentication Entry Point**

Create AuthEntryPointJwt in com.example.jwt\_demo.security package:

package com.example.jwt\_demo.security;  
  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import org.springframework.security.core.AuthenticationException;  
import org.springframework.security.web.AuthenticationEntryPoint;  
import org.springframework.stereotype.Component;  
import java.io.IOException;  
@Component  
public class AuthEntryPointJwt implements AuthenticationEntryPoint {  
 @Override  
 public void commence(  
 HttpServletRequest request,  
 HttpServletResponse response,  
 AuthenticationException authException  
 ) throws IOException {  
 response.sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Error: Unauthorized");  
 }  
}

**6.3. JWT Authentication Filter**

Create AuthTokenFilter in com.example.jwt\_demo.security package:

package com.example.jwt\_demo.security;  
  
import com.example.jwt\_demo.service.CustomUserDetailsService;  
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.core.userdetails.UserDetails;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.authentication.\*;  
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;  
import org.springframework.stereotype.Component;  
import org.springframework.web.filter.OncePerRequestFilter;  
import java.io.IOException;  
@Component  
public class AuthTokenFilter extends OncePerRequestFilter {  
 @Autowired  
 private JwtUtil jwtUtils;  
 @Autowired  
 private CustomUserDetailsService userDetailsService;  
 @Override  
 protected void doFilterInternal(  
 HttpServletRequest request,  
 HttpServletResponse response,  
 FilterChain filterChain  
 ) throws ServletException, IOException {  
 try {  
 String jwt = parseJwt(request);  
 if (jwt != null && jwtUtils.validateJwtToken(jwt)) {  
 String username = jwtUtils.getUsernameFromToken(jwt);  
 UserDetails userDetails = userDetailsService.loadUserByUsername(username);  
 UsernamePasswordAuthenticationToken authentication =  
 new UsernamePasswordAuthenticationToken(  
 userDetails,  
 null,  
 userDetails.getAuthorities()  
 );  
 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
 SecurityContextHolder.getContext().setAuthentication(authentication);  
 }  
 } catch (Exception e) {  
 System.out.println("Cannot set user authentication: " + e);  
 }  
 filterChain.doFilter(request, response);  
 }  
 private String parseJwt(HttpServletRequest request) {  
 String headerAuth = request.getHeader("Authorization");  
 if (headerAuth != null && headerAuth.startsWith("Bearer ")) {  
 return headerAuth.substring(7);  
 }  
 return null;  
 }  
}

**6.4. Security Configuration**

Create WebSecurityConfig in com.example.jwt\_demo.security package:

package com.example.jwt\_demo.security;  
  
import com.example.jwt\_demo.service.CustomUserDetailsService;  
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.config.annotation.authentication.configuration.AuthenticationConfiguration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.http.SessionCreationPolicy;  
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  
public class WebSecurityConfig {  
 @Autowired  
 CustomUserDetailsService userDetailsService;  
 @Autowired  
 private AuthEntryPointJwt unauthorizedHandler;  
 @Bean  
 public AuthTokenFilter authenticationJwtTokenFilter() {  
 return new AuthTokenFilter();  
 }  
 @Bean  
 public AuthenticationManager authenticationManager(  
 AuthenticationConfiguration authenticationConfiguration  
 ) throws Exception {  
 return authenticationConfiguration.getAuthenticationManager();  
 }  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 // Updated configuration for Spring Security 6.x  
 http  
 .csrf(csrf -> csrf.disable()) // Disable CSRF  
 .cors(cors -> cors.disable()) // Disable CORS (or configure if needed)  
 .exceptionHandling(exceptionHandling ->  
 exceptionHandling.authenticationEntryPoint(unauthorizedHandler)  
 )  
 .sessionManagement(sessionManagement ->  
 sessionManagement.sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 )  
 .authorizeHttpRequests(authorizeRequests ->  
 authorizeRequests  
 .requestMatchers("/api/auth/\*\*", "/api/test/all").permitAll() // Use 'requestMatchers' instead of 'antMatchers'  
 .anyRequest().authenticated()  
 );  
 // Add the JWT Token filter before the UsernamePasswordAuthenticationFilter  
 http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);  
 return http.build();  
 }  
}

**Note**: Starting from Spring Boot 2.7.x and Spring Security 5.7.x, WebSecurityConfigurerAdapter is deprecated in favor of the component-based security configuration shown above.

**Step 7: Create Authentication Controllers**

**7.1. Auth Controller**

Create AuthController in com.example.jwt\_demo.controller package:

package com.example.jwt\_demo.controller;  
  
import com.example.jwt\_demo.model.User;  
import com.example.jwt\_demo.repository.UserRepository;  
import com.example.jwt\_demo.security.JwtUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.authentication.\*;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.web.bind.annotation.\*;  
@RestController  
@RequestMapping("/api/auth")  
public class AuthController {  
 @Autowired  
 AuthenticationManager authenticationManager;  
 @Autowired  
 UserRepository userRepository;  
 @Autowired  
 PasswordEncoder encoder;  
 @Autowired  
 JwtUtil jwtUtils;  
 @PostMapping("/signin")  
 public String authenticateUser(@RequestBody User user) {  
 Authentication authentication = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(  
 user.getUsername(),  
 user.getPassword()  
 )  
 );  
 UserDetails userDetails = (UserDetails) authentication.getPrincipal();  
 return jwtUtils.generateToken(userDetails.getUsername());  
 }  
 @PostMapping("/signup")  
 public String registerUser(@RequestBody User user) {  
 if (userRepository.existsByUsername(user.getUsername())) {  
 return "Error: Username is already taken!";  
 }  
 // Create new user's account  
 User newUser = new User(  
 null,  
 user.getUsername(),  
 encoder.encode(user.getPassword())  
 );  
 userRepository.save(newUser);  
 return "User registered successfully!";  
 }  
}

**Step 8: Secure REST Endpoints**

**8.1. Test Controller**

Create TestController in com.example.jwt\_demo.controller package:

package com.example.jwt\_demo.controller;  
  
import org.springframework.web.bind.annotation.\*;  
@RestController  
@RequestMapping("/api/test")  
public class TestController {  
 @GetMapping("/all")  
 public String allAccess() {  
 return "Public Content.";  
 }  
 @GetMapping("/user")  
 public String userAccess() {  
 return "User Content.";  
 }  
}

* /api/test/all: Public endpoint, accessible without authentication.
* /api/test/user: Secured endpoint, accessible only with valid JWT.