**Create authentication service that returns JWT**

**Code:**

Step 1: Create Spring Boot Project

1. Go to <https://start.spring.io/>
2. Choose:
   * Project: Maven
   * Spring Boot: 3.2.x
   * Group: com.example
   * Artifact: jwt-auth
   * Language: Java
   * Dependencies:
     + ✅ Spring Web
     + ✅ Spring Security
     + ✅ Spring Boot DevTools
     + ✅ Spring Boot Starter Validation
     + ✅ Spring Boot Starter JWT (we’ll add manually)
3. Click Generate, extract ZIP, open in VS Code

Step 2: Add JWT Dependency

In pom.xml, add:

xml

CopyEdit

<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>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

Then run:

bash

CopyEdit

mvn clean install

Step 3: Create User DTO

📄 model/UserDTO.java

java

CopyEdit

package com.example.jwtauth.model;

public class UserDTO {

private String username;

private String password;

public String getUsername() { return username; }

public void setUsername(String username) { this.username = username; }

public String getPassword() { return password; }

public void setPassword(String password) { this.password = password; }

}

Step 4: JWT Utility

📄 service/JwtUtil.java

java

CopyEdit

package com.example.jwtauth.service;

import io.jsonwebtoken.\*;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Component;

import java.security.Key;

import java.util.Date;

@Component

public class JwtUtil {

private final Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);

private final long EXPIRATION = 1000 \* 60 \* 60; // 1 hour

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuer("jwt-auth")

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + EXPIRATION))

.signWith(key)

.compact();

}

public String validateTokenAndGetUsername(String token) {

return Jwts.parserBuilder()

.setSigningKey(key)

.build()

.parseClaimsJws(token)

.getBody()

.getSubject();

}

}

Step 5: Create Auth Controller

📄 controller/AuthController.java

java

CopyEdit

package com.example.jwtauth.controller;

import com.example.jwtauth.model.UserDTO;

import com.example.jwtauth.service.JwtUtil;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.HashMap;

import java.util.Map;

@RestController

@RequestMapping("/auth")

public class AuthController {

private final JwtUtil jwtUtil;

private final Map<String, String> db = new HashMap<>();

public AuthController(JwtUtil jwtUtil) {

this.jwtUtil = jwtUtil;

}

@PostMapping("/register")

public ResponseEntity<String> register(@RequestBody UserDTO user) {

db.put(user.getUsername(), user.getPassword());

return ResponseEntity.ok("User registered");

}

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody UserDTO user) {

String password = db.get(user.getUsername());

if (password != null && password.equals(user.getPassword())) {

String token = jwtUtil.generateToken(user.getUsername());

return ResponseEntity.ok(Map.of("token", token));

} else {

return ResponseEntity.status(401).body("Invalid credentials");

}

}

}

Step 6: Secure API Endpoint

📄 controller/HelloController.java

java

CopyEdit

package com.example.jwtauth.controller;

import org.springframework.web.bind.annotation.\*;

@RestController

public class HelloController {

@GetMapping("/hello")

public String hello() {

return "Hello, secure world!";

}

}

Step 7: JWT Filter for Authentication

📄 config/JwtFilter.java

java

CopyEdit

package com.example.jwtauth.config;

import com.example.jwtauth.service.JwtUtil;

import jakarta.servlet.\*;

import jakarta.servlet.http.\*;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.authentication.\*;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.stereotype.Component;

import java.io.IOException;

import java.util.List;

@Component

public class JwtFilter extends GenericFilter {

private final JwtUtil jwtUtil;

public JwtFilter(JwtUtil jwtUtil) {

this.jwtUtil = jwtUtil;

}

@Override

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) req;

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

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

String token = authHeader.substring(7);

try {

String username = jwtUtil.validateTokenAndGetUsername(token);

var auth = new UsernamePasswordAuthenticationToken(

username, null, List.of(new SimpleGrantedAuthority("USER")));

SecurityContextHolder.getContext().setAuthentication(auth);

} catch (Exception e) {

SecurityContextHolder.clearContext();

((HttpServletResponse) res).sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Invalid JWT");

return;

}

}

chain.doFilter(req, res);

}

}

Step 8: Configure Spring Security

📄 config/SecurityConfig.java

java

CopyEdit

package com.example.jwtauth.config;

import org.springframework.context.annotation.\*;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.web.\*;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

private final JwtFilter jwtFilter;

public SecurityConfig(JwtFilter jwtFilter) {

this.jwtFilter = jwtFilter;

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

return http

.csrf(csrf -> csrf.disable())

.authorizeHttpRequests(auth -> auth

.requestMatchers("/auth/\*\*").permitAll()

.anyRequest().authenticated()

)

.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)

.build();

}

}

**Output:**





