**Create authentication service that returns JWT**

package com.cognizant.spring\_learn.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthenticationController {

private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("START - authenticate()");

LOGGER.debug("Authorization Header: {}", authHeader);

Map<String, String> map = new HashMap<>();

map.put("token", "");

LOGGER.info("END - authenticate()");

return map;

}

}

package com.cognizant.spring\_learn.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

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

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

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.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.config.Customizer;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

// Define user details (in-memory authentication)

@Bean

public UserDetailsService userDetailsService(PasswordEncoder encoder) {

UserDetails user = User

.withUsername("user")

.password(encoder.encode("pwd"))

.roles("USER")

.build();

UserDetails admin = User

.withUsername("admin")

.password(encoder.encode("admin123"))

.roles("ADMIN")

.build();

return new InMemoryUserDetailsManager(user, admin);

}

// Define secure password encoder

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder(); // Recommended for real apps

}

// Define security filter chain

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.httpBasic(Customizer.withDefaults()) // Use basic auth

.authorizeHttpRequests(auth -> auth

.requestMatchers("/countries").hasRole("USER")

.requestMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated()

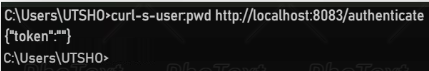
)

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

return http.build();

}

}



package com.cognizant.spring\_learn.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RestController;

import java.util.Base64;

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthenticationController {

private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("START - authenticate()");

LOGGER.debug("Authorization Header: {}", authHeader);

// Extract username

String username = getUser(authHeader);

LOGGER.debug("Extracted Username: {}", username);

Map<String, String> map = new HashMap<>();

map.put("token", "");

LOGGER.info("END - authenticate()");

return map;

}

// 🔐 Decode and extract username from Authorization header

private String getUser(String authHeader) {

LOGGER.debug("Inside getUser()");

// Strip "Basic " prefix

String encodedCredentials = authHeader.substring("Basic ".length());

LOGGER.debug("Base64 Encoded Credentials: {}", encodedCredentials);

// Decode base64 -> byte[]

byte[] decodedBytes = Base64.getDecoder().decode(encodedCredentials);

// Convert byte[] to string: "username:password"

String decodedString = new String(decodedBytes);

LOGGER.debug("Decoded Credentials: {}", decodedString);

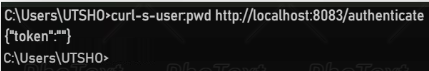
// Extract username

String username = decodedString.split(":")[0];

return username;

}

}



<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/>

</parent>

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-learn</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Security -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- Logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

</dependency>

<!-- JWT (modern version) -->

<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>

<!-- XML configuration support -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<!-- Testing -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

package com.cognizant.spring\_learn.controller;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RestController;

import javax.crypto.SecretKey;

import java.util.Base64;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthenticationController {

private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);

// At least 256-bit key (32 bytes for HS256)

private static final String SECRET = "my-secret-key-which-is-at-least-32-bytes!";

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("START - authenticate()");

LOGGER.debug("Authorization Header: {}", authHeader);

String username = getUser(authHeader);

LOGGER.debug("Extracted Username: {}", username);

String token = generateJwt(username);

LOGGER.debug("Generated JWT: {}", token);

Map<String, String> map = new HashMap<>();

map.put("token", token);

LOGGER.info("END - authenticate()");

return map;

}

// 🔐 Extract username from Basic Auth Header

private String getUser(String authHeader) {

LOGGER.debug("Inside getUser()");

String encodedCredentials = authHeader.substring("Basic ".length());

byte[] decodedBytes = Base64.getDecoder().decode(encodedCredentials);

String decodedString = new String(decodedBytes); // format: "username:password"

LOGGER.debug("Decoded Credentials: {}", decodedString);

return decodedString.split(":")[0];

}

// 🔐 Generate JWT token

private String generateJwt(String user) {

LOGGER.debug("Inside generateJwt() for user: {}", user);

SecretKey key = Keys.hmacShaKeyFor(SECRET.getBytes());

return Jwts.builder()

.setSubject(user)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 20 \* 60 \* 1000)) // 20 mins

.signWith(key, SignatureAlgorithm.HS256)

.compact();

}

}

package com.cognizant.spring\_learn.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

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

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

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.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.config.Customizer;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

// Define user details (in-memory authentication)

@Bean

public UserDetailsService userDetailsService(PasswordEncoder encoder) {

UserDetails user = User

.withUsername("user")

.password(encoder.encode("pwd"))

.roles("USER")

.build();

UserDetails admin = User

.withUsername("admin")

.password(encoder.encode("admin123"))

.roles("ADMIN")

.build();

return new InMemoryUserDetailsManager(user, admin);

}

// Define secure password encoder

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder(); // Recommended for real apps

}

// Define security filter chain

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.httpBasic(Customizer.withDefaults()) // Use basic auth

.authorizeHttpRequests(auth -> auth

.requestMatchers("/countries").hasRole("USER")

.requestMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated()

)

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

return http.build();

}

}

