**WEEK-4**

**JWT Tokens**

**Create authentication service that returns JWT   
Code:**

**JwtAuthApplication.java:**

package com.example.jwt\_auth;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JwtAuthApplication {

    public static void main(String[] args) {

        SpringApplication.run(JwtAuthApplication.class, args);

    }

}

**SecurityConfig.java:**

package com.example.jwt\_auth.config;

import com.example.jwt\_auth.filter.JwtAuthFilter;

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.web.SecurityFilterChain;

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

@Configuration

public class SecurityConfig {

    @Autowired

    private JwtAuthFilter jwtAuthFilter;

    @Bean

    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http

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

            .authorizeHttpRequests(auth -> auth

                .requestMatchers("/authenticate").permitAll()

                .anyRequest().authenticated()

            )

            .sessionManagement(sess -> sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();

    }

    @Bean

    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

        return config.getAuthenticationManager();

    }

}

**AuthController.java:**

package com.example.jwt\_auth.controller;

import com.example.jwt\_auth.service.JwtService;

import org.springframework.beans.factory.annotation.Autowired;

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

import com.example.jwt\_auth.dto.AuthRequest;

@RestController

public class AuthController {

    @Autowired

    private JwtService jwtService;

    @PostMapping("/authenticate")

public String authenticate(@RequestBody AuthRequest request) {

    if ("oviya".equals(request.getUsername()) && "1234".equals(request.getPassword())) {

        return jwtService.generateToken(request.getUsername());

    } else {

        throw new RuntimeException("Invalid credentials");

    }

}

}

**AuthRequest.java:**

package com.example.jwt\_auth.dto;

public class AuthRequest {

    private String username;

    private String password;

    public AuthRequest() {}

    public AuthRequest(String username, String password) {

        this.username = username;

        this.password = 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;

    }

}

**JwtAuthFilter.java:**

package com.example.jwt\_auth.filter;

import com.example.jwt\_auth.util.JwtUtil;

import io.jsonwebtoken.Claims;

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.authentication.UsernamePasswordAuthenticationToken;

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

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

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

import java.util.Collections;

@Component

public class JwtAuthFilter extends OncePerRequestFilter {

    @Autowired

    private JwtUtil jwtUtil;

    @Override

    protected void doFilterInternal(HttpServletRequest request,

                                    HttpServletResponse response,

                                    FilterChain filterChain)

                                    throws ServletException, IOException {

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

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

            String token = authHeader.substring(7);

            Claims claims = jwtUtil.extractAllClaims(token);

            String username = claims.getSubject();

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

                UsernamePasswordAuthenticationToken authToken =

                        new UsernamePasswordAuthenticationToken(username, null, Collections.emptyList());

                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authToken);

            }

        }

        filterChain.doFilter(request, response);

    }

    @Override

    protected boolean shouldNotFilter(HttpServletRequest request) {

        return request.getServletPath().equals("/authenticate");

    }

}

**JwtService.java:**

package com.example.jwt\_auth.service;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Service;

import javax.crypto.SecretKey;

import java.util.Date;

@Service

public class JwtService {

    private final SecretKey key = Keys.secretKeyFor(SignatureAlgorithm.HS256); // 256-bit key

    public String generateToken(String username) {

        return Jwts.builder()

                .setSubject(username)

                .setIssuedAt(new Date())

                .setExpiration(new Date(System.currentTimeMillis() + 3600000)) // 1 hour

                .signWith(key)

                .compact();

    }

    public boolean validateToken(String token) {

        try {

            Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);

            return true;

        } catch (Exception e) {

            return false;

        }

    }

    public String extractUsername(String token) {

        return Jwts.parserBuilder()

                .setSigningKey(key)

                .build()

                .parseClaimsJws(token)

                .getBody()

                .getSubject();

    }

}

**JwtUtil.java:**

package com.example.jwt\_auth.util;

import io.jsonwebtoken.\*;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.Base64;

import java.security.Key;

import javax.crypto.spec.SecretKeySpec;

@Component

public class JwtUtil {

    private final String SECRET\_KEY = "mysecretkeymysecretkeymysecretkey"; // should be 256-bit for HS256

    private Key getSignKey() {

        byte[] keyBytes = Base64.getEncoder().encode(SECRET\_KEY.getBytes());

        return new SecretKeySpec(keyBytes, SignatureAlgorithm.HS256.getJcaName());

    }

    public String generateToken(String username) {

        long currentTimeMillis = System.currentTimeMillis();

        return Jwts.builder()

                .setSubject(username)

                .setIssuedAt(new Date(currentTimeMillis))

                .setExpiration(new Date(currentTimeMillis + 1000 \* 60 \* 10)) // 10 minutes

                .signWith(getSignKey(), SignatureAlgorithm.HS256)

                .compact();

    }

    public Claims extractAllClaims(String token) {

        return Jwts.parserBuilder()

                .setSigningKey(getSignKey())

                .build()

                .parseClaimsJws(token)

                .getBody();

    }

    public String extractUsername(String token) {

        return extractAllClaims(token).getSubject();

    }

    public boolean validateToken(String token, String username) {

        String extractedUsername = extractUsername(token);

        return extractedUsername.equals(username) && !isTokenExpired(token);

    }

    private boolean isTokenExpired(String token) {

        return extractAllClaims(token).getExpiration().before(new Date());

    }

}

**Output:**



