Jobconnect Backend code  
  
package com.jobconnect;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

import org.springframework.http.\*;

import org.springframework.stereotype.\*;

import org.springframework.beans.factory.annotation.\*;

import javax.persistence.\*;

import org.springframework.data.jpa.repository.\*;

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

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

import org.springframework.security.config.annotation.authentication.builders.\*;

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

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.\*;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

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

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

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

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

import org.springframework.security.config.annotation.web.configuration.\*;

import io.jsonwebtoken.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.IOException;

import java.util.\*;

import java.util.stream.Collectors;

import java.time.LocalDate;

@SpringBootApplication

public class JobConnectApplication {

public static void main(String[] args) {

SpringApplication.run(JobConnectApplication.class, args);

}

// ======== Entities ========

@Entity

class User {

@Id @GeneratedValue private Long id;

private String username;

private String password;

@Enumerated(EnumType.STRING)

private Role role;

}

enum Role {

JOB\_SEEKER, EMPLOYER

}

@Entity

class Job {

@Id @GeneratedValue private Long id;

private String title;

private String description;

private String location;

private Double salary;

private LocalDate deadline;

@ManyToOne

private User postedBy;

}

// ======== DTOs ========

class UserDTO {

public String username;

public String password;

public Role role;

}

class LoginDTO {

public String username;

public String password;

}

// ======== Repositories ========

interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username);

}

interface JobRepository extends JpaRepository<Job, Long> {

List<Job> findByLocationContainingOrTitleContaining(String location, String title);

}

// ======== JWT Utils ========

@Component

class JwtUtil {

private String secret = "jobconnectsecret";

private long expiration = 3600000;

public String generateToken(String username, Role role) {

return Jwts.builder()

.setSubject(username)

.claim("role", role.name())

.setIssuedAt(new Date())

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

.signWith(SignatureAlgorithm.HS256, secret)

.compact();

}

public String extractUsername(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubject();

}

public String extractRole(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().get("role", String.class);

}

public boolean validate(String token) {

try {

Jwts.parser().setSigningKey(secret).parseClaimsJws(token);

return true;

} catch (Exception e) {

return false;

}

}

}

// ======== Security Filter ========

@Component

class JwtFilter extends OncePerRequestFilter {

@Autowired JwtUtil jwtUtil;

@Autowired UserDetailsServiceImpl userDetailsService;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

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

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

String token = header.substring(7);

if (jwtUtil.validate(token)) {

String username = jwtUtil.extractUsername(token);

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

UsernamePasswordAuthenticationToken authToken =

new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}

@Service

class UserDetailsServiceImpl implements UserDetailsService {

@Autowired UserRepository userRepo;

@Override

public UserDetails loadUserByUsername(String username) {

User user = userRepo.findByUsername(username).orElseThrow();

return new org.springframework.security.core.userdetails.User(

user.getUsername(),

user.getPassword(),

Collections.singletonList(new SimpleGrantedAuthority("ROLE\_" + user.getRole()))

);

}

}

@Configuration

@EnableWebSecurity

class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired JwtFilter jwtFilter;

@Autowired UserDetailsServiceImpl uds;

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

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

.antMatchers("/jobs/employer/\*\*").hasRole("EMPLOYER")

.anyRequest().authenticated()

.and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(uds).passwordEncoder(passwordEncoder());

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

}

// ======== Services ========

@Service

class UserService {

@Autowired UserRepository userRepo;

@Autowired PasswordEncoder encoder;

@Autowired JwtUtil jwtUtil;

@Autowired AuthenticationManager authManager;

public ResponseEntity<?> registerUser(UserDTO dto) {

User user = new User();

user.setUsername(dto.username);

user.setPassword(encoder.encode(dto.password));

user.setRole(dto.role);

userRepo.save(user);

return ResponseEntity.ok("User registered");

}

public ResponseEntity<?> authenticateUser(LoginDTO dto) {

Authentication auth = authManager.authenticate(new UsernamePasswordAuthenticationToken(dto.username, dto.password));

User user = userRepo.findByUsername(dto.username).orElseThrow();

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

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

}

}

@Service

class JobService {

@Autowired JobRepository jobRepo;

@Autowired UserRepository userRepo;

public ResponseEntity<?> createJob(Job job, String username) {

User user = userRepo.findByUsername(username).orElseThrow();

job.setPostedBy(user);

return ResponseEntity.ok(jobRepo.save(job));

}

public List<Job> getAllJobs() {

return jobRepo.findAll();

}

public List<Job> searchJobs(String keyword) {

return jobRepo.findByLocationContainingOrTitleContaining(keyword, keyword);

}

}

// ======== Controllers ========

@RestController

@RequestMapping("/auth")

class AuthController {

@Autowired UserService userService;

@PostMapping("/register")

public ResponseEntity<?> register(@RequestBody UserDTO dto) {

return userService.registerUser(dto);

}

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody LoginDTO dto) {

return userService.authenticateUser(dto);

}

}

@RestController

@RequestMapping("/jobs")

class JobController {

@Autowired JobService jobService;

@PostMapping("/employer")

public ResponseEntity<?> postJob(@RequestBody Job job, Principal principal) {

return jobService.createJob(job, principal.getName());

}

@GetMapping

public List<Job> allJobs() {

return jobService.getAllJobs();

}

@GetMapping("/search")

public List<Job> search(@RequestParam String keyword) {

return jobService.searchJobs(keyword);

}

}

}