package com.kryptos.hrms.payload.response;

import java.util.List;

public class JwtResponse {

private String token;

private String type = "Bearer";

private Long id;

private String username;

private List<String> roles;

public String getToken() {

return token;

}

public void setToken(String token) {

this.token = token;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public List<String> getRoles() {

return roles;

}

public void setRoles(List<String> roles) {

this.roles = roles;

}

public JwtResponse(String token, Long id, String username, List<String> roles) {

super();

this.token = token;

this.id = id;

this.username = username;

this.roles = roles;

}

}

package com.kryptos.hrms.payload.response;

public class MessageResponse {

private String message;

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

public MessageResponse(String message) {

super();

this.message = message;

}

}

package com.kryptos.hrms.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.kryptos.hrms.model.ERole;

import com.kryptos.hrms.model.Role;

@Repository

public interface RoleRepository extends JpaRepository<Role, Long> {

Optional<Role> findByName(ERole name);

}

package com.kryptos.hrms.model;

import java.util.HashSet;

import java.util.Set;

import jakarta.persistence.Entity;

import jakarta.persistence.FetchType;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.JoinTable;

import jakarta.persistence.ManyToMany;

import jakarta.persistence.Table;

import jakarta.persistence.UniqueConstraint;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

@Entity

@Table(name = "users", uniqueConstraints = { @UniqueConstraint(columnNames = "username") })

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Size(max = 60)

@NotBlank

private String username;

@Size(max = 120)

@NotBlank

private String password;

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(name = "user\_roles", joinColumns = @JoinColumn(name = "user\_id"), inverseJoinColumns = @JoinColumn(name = "role\_id"))

private Set<Role> roles = new HashSet<>();

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

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;

}

public Set<Role> getRoles() {

return roles;

}

public void setRoles(Set<Role> roles) {

this.roles = roles;

}

public User() {

}

public User(String username, String password) {

super();

this.username = username;

this.password = password;

}

}

package com.kryptos.hrms.security;

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.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;

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

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

import org.springframework.security.core.Authentication;

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;

import com.kryptos.hrms.security.jwt.AuthEntryPointJwt;

import com.kryptos.hrms.security.jwt.AuthTokenFilter;

import com.kryptos.hrms.services.UserDetailsServiceImpl;

@Configuration

@EnableMethodSecurity

public class WebSecurityConfig {

@Autowired

private UserDetailsServiceImpl userDetailsService;

@Autowired

private AuthEntryPointJwt unathorizedHandler;

@Bean

public AuthTokenFilter authenticationJwtTokenFilter() {

return new AuthTokenFilter();

}

@Bean

public DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(userDetailsService);

authProvider.setPasswordEncoder(passwordEncoder());

return authProvider;

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {

return authConfig.getAuthenticationManager();

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

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

.exceptionHandling(exception -> exception.authenticationEntryPoint(unathorizedHandler))

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

.authorizeHttpRequests(auth -> auth.requestMatchers("/api/auth/\*\*").permitAll()

.requestMatchers("api/test/\*\*").permitAll().anyRequest().authenticated());

http.authenticationProvider(authenticationProvider());

http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);

return http.build();

}

}

package com.kryptos.hrms.security.jwt;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.http.MediaType;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.ObjectMapper;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@Component

public class AuthEntryPointJwt implements AuthenticationEntryPoint {

private static final Logger logger = LoggerFactory.getLogger(AuthEntryPointJwt.class);

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,

AuthenticationException authException) throws IOException, ServletException {

logger.error("Unauthorized errors {}", authException.getMessage());

response.setContentType(MediaType.APPLICATION\_JSON\_VALUE);

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

final Map<String, Object> body = new HashMap<>();

body.put("status", HttpServletResponse.SC\_UNAUTHORIZED);

body.put("error", "Unauthorized");

body.put("message", authException.getMessage());

body.put("path", request.getServletPath());

final ObjectMapper mapper = new ObjectMapper();

mapper.writeValue(response.getOutputStream(), body);

}

}

package com.kryptos.hrms.security.jwt;

import java.io.IOException;

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

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

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

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

import org.springframework.util.StringUtils;

import org.springframework.web.filter.OncePerRequestFilter;

import com.kryptos.hrms.services.UserDetailsServiceImpl;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

public class AuthTokenFilter extends OncePerRequestFilter {

@Autowired

private JwtUtils jwtUtils;

@Autowired

private UserDetailsServiceImpl useDetailsServiceImpl;

private String parseJwt(HttpServletRequest request) {

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

if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {

return headerAuth.substring(7);

}

return null;

}

@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.getUserNameFromJwtToken(jwt);

UserDetails userDetails = useDetailsServiceImpl.loadUserByUsername(username);

UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

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

SecurityContextHolder.getContext().setAuthentication(authentication);

}

} catch (Exception e) {

logger.error("Cannot set user authentication : {}", e);

}

filterChain.doFilter(request, response);

}

}

package com.kryptos.hrms.security.jwt;

import java.security.Key;

import java.util.Date;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.security.core.Authentication;

import org.springframework.stereotype.Component;

import com.kryptos.hrms.security.services.UserDetailsImpl;

import io.jsonwebtoken.ExpiredJwtException;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.MalformedJwtException;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.UnsupportedJwtException;

import io.jsonwebtoken.io.Decoders;

import io.jsonwebtoken.security.Keys;

@Component

public class JwtUtils {

private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

@Value("${kryptos.app.jwtSecret}")

private String jwtSecret;

@Value("${kryptos.app.jwtExpirationMs}")

private int jwtExpirationMs;

public String generateJwtToken(Authentication authentication) {

UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();

return Jwts.builder().setSubject((userPrincipal.getUsername())).setIssuedAt(new Date())

.setExpiration(new Date((new Date()).getTime() + jwtExpirationMs)).signWith(key(), SignatureAlgorithm.HS256)

.compact();

}

public String getUserNameFromJwtToken(String token) {

return Jwts.parserBuilder().setSigningKey(key()).build().parseClaimsJws(token).getBody().getSubject();

}

public Key key() {

return Keys.hmacShaKeyFor(Decoders.BASE64.decode(jwtSecret));

}

public boolean validateJwtToken(String authToken) {

try {

Jwts.parserBuilder().setSigningKey(key()).build().parse(authToken);

return true;

} catch (MalformedJwtException e) {

logger.error("Invalid JWT token : {}", e.getMessage());

} catch (ExpiredJwtException e) {

logger.error("JWT token is expired : {}", e.getMessage());

} catch (UnsupportedJwtException e) {

logger.error("JWT token is unsupported : {}", e.getMessage());

} catch (IllegalArgumentException e) {

logger.error("JWT token is excited : {}", e.getMessage());

}

return false;

}

}

package com.kryptos.hrms.security.jwt;

import java.security.Key;

import java.util.Date;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.security.core.Authentication;

import org.springframework.stereotype.Component;

import com.kryptos.hrms.security.services.UserDetailsImpl;

import io.jsonwebtoken.ExpiredJwtException;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.MalformedJwtException;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.UnsupportedJwtException;

import io.jsonwebtoken.io.Decoders;

import io.jsonwebtoken.security.Keys;

@Component

public class JwtUtils {

private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

@Value("${kryptos.app.jwtSecret}")

private String jwtSecret;

@Value("${kryptos.app.jwtExpirationMs}")

private int jwtExpirationMs;

public String generateJwtToken(Authentication authentication) {

UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();

return Jwts.builder().setSubject((userPrincipal.getUsername())).setIssuedAt(new Date())

.setExpiration(new Date((new Date()).getTime() + jwtExpirationMs)).signWith(key(), SignatureAlgorithm.HS256)

.compact();

}

public String getUserNameFromJwtToken(String token) {

return Jwts.parserBuilder().setSigningKey(key()).build().parseClaimsJws(token).getBody().getSubject();

}

public Key key() {

return Keys.hmacShaKeyFor(Decoders.BASE64.decode(jwtSecret));

}

public boolean validateJwtToken(String authToken) {

try {

Jwts.parserBuilder().setSigningKey(key()).build().parse(authToken);

return true;

} catch (MalformedJwtException e) {

logger.error("Invalid JWT token : {}", e.getMessage());

} catch (ExpiredJwtException e) {

logger.error("JWT token is expired : {}", e.getMessage());

} catch (UnsupportedJwtException e) {

logger.error("JWT token is unsupported : {}", e.getMessage());

} catch (IllegalArgumentException e) {

logger.error("JWT token is excited : {}", e.getMessage());

}

return false;

}

}

package com.kryptos.hrms.security.services;

import java.util.Collection;

import java.util.Objects;

import java.util.List;

import java.util.stream.Collectors;

import org.springframework.security.core.GrantedAuthority;

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

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

import com.kryptos.hrms.model.User;

import lombok.AllArgsConstructor;

import lombok.Data;

@Data

@AllArgsConstructor

public class UserDetailsImpl implements UserDetails {

private static final long serialVersionUID = 6767982520604222177L;

private Long id;

private String username;

private String password;

private Collection<? extends GrantedAuthority> authorities;

public static UserDetailsImpl build(User user) {

List<GrantedAuthority> authorities = user.getRoles().stream()

.map(role -> new SimpleGrantedAuthority(role.getName().name())).collect(Collectors.toList());

return new UserDetailsImpl(user.getId(), user.getUsername(), user.getPassword(), authorities);

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return authorities;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public static long getSerialversionuid() {

return serialVersionUID;

}

public void setUsername(String username) {

this.username = username;

}

public void setPassword(String password) {

this.password = password;

}

public void setAuthorities(Collection<? extends GrantedAuthority> authorities) {

this.authorities = authorities;

}

@Override

public String getPassword() {

return password;

}

@Override

public String getUsername() {

return username;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

@Override

public boolean equals(Object o) {

if (this == o)

return true;

UserDetailsImpl user = (UserDetailsImpl) o;

return Objects.equals(id, user.id);

}

}

package com.kryptos.hrms.services;

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

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

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

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

import org.springframework.stereotype.Service;

import com.kryptos.hrms.model.User;

import com.kryptos.hrms.repository.UserRepository;

import com.kryptos.hrms.security.services.UserDetailsImpl;

import jakarta.transaction.Transactional;

@Service

public class UserDetailsServiceImpl implements UserDetailsService {

@Autowired

UserRepository userRepository;

@Override

@Transactional

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByUsername(username).orElseThrow(() -> new UsernameNotFoundException(

"User not found with username : " + username));

return UserDetailsImpl.build(user);

}

}

package com.kryptos.hrms.controller;

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

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

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

@RestController

@RequestMapping("/api/test")

public class TestController {

@GetMapping("/welcome")

public String welcome() {

return "Public Access";

}

}

package com.kryptos.hrms.controller;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.stream.Collectors;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

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

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

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

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

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

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

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

import com.kryptos.hrms.model.ERole;

import com.kryptos.hrms.model.Role;

import com.kryptos.hrms.model.User;

import com.kryptos.hrms.payload.request.LoginRequest;

import com.kryptos.hrms.payload.request.SignupRequest;

import com.kryptos.hrms.payload.response.JwtResponse;

import com.kryptos.hrms.payload.response.MessageResponse;

import com.kryptos.hrms.repository.RoleRepository;

import com.kryptos.hrms.repository.UserRepository;

import com.kryptos.hrms.security.jwt.JwtUtils;

import com.kryptos.hrms.security.services.UserDetailsImpl;

import jakarta.validation.Valid;

@CrossOrigin(origins = "\*", maxAge = 3600)

@RestController

@RequestMapping("/api/auth")

public class AuthController {

@Autowired

AuthenticationManager authenticationManager;

@Autowired

UserRepository userRepository;

@Autowired

RoleRepository roleRepository;

@Autowired

PasswordEncoder encoder;

@Autowired

JwtUtils jwtUtils;

@PostMapping("/signin")

public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {

Authentication authentication = authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword()));

SecurityContextHolder.getContext().setAuthentication(authentication);

String jwt = jwtUtils.generateJwtToken(authentication);

UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();

List<String> roles = userDetails.getAuthorities().stream().map(item -> item.getAuthority())

.collect(Collectors.toList());

return ResponseEntity.ok(new JwtResponse(jwt, userDetails.getId(), userDetails.getUsername(), roles));

}

@PostMapping("/signup")

public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) {

if (userRepository.existsByUsername(signUpRequest.getUsername())) {

return ResponseEntity.badRequest().body(new MessageResponse("Error: username is already taken : "));

}

User user = new User(signUpRequest.getUsername(), encoder.encode(signUpRequest.getPassword()));

Set<String> strRoles = signUpRequest.getRole();

Set<Role> roles = new HashSet<>();

if (strRoles == null) {

Role employeeRole = roleRepository.findByName(ERole.ROLE\_EMPLOYEE)

.orElseThrow(() -> new RuntimeException("Error : Role is not found."));

roles.add(employeeRole);

} else {

strRoles.forEach(role -> {

switch (role) {

case "admin": {

Role adminRole = roleRepository.findByName(ERole.ROLE\_ADMIN)

.orElseThrow(() -> new RuntimeException("Error : Role is not found."));

roles.add(adminRole);

}

case "humanresource": {

Role humanresourceRole = roleRepository.findByName(ERole.ROLE\_HUMANRESOURCE)

.orElseThrow(() -> new RuntimeException("Error : Role is not found."));

roles.add(humanresourceRole);

}

case "manager": {

Role managerRole = roleRepository.findByName(ERole.ROLE\_MANAGER)

.orElseThrow(() -> new RuntimeException("Error : Role is not found."));

roles.add(managerRole);

break;

}

}

});

}

user.setRoles(roles);

userRepository.save(user);

return ResponseEntity.ok(new MessageResponse("User registered successfully"));

}

}