**EventManagementApplication.java**

package com.cts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EventManagementApplication {

public static void main(String[] args) {

SpringApplication.run(EventManagementApplication.class, args);

}

}

**SecurityConfig.java**

**package com.cts.config;**

**import com.cts.filter.JwtFilter;**

**import java.util.Arrays;**

**import org.springframework.web.filter.CorsFilter;**

**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.dao.DaoAuthenticationProvider;**

**import org.springframework.security.config.Customizer;**

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

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

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

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

**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 org.springframework.web.cors.CorsConfiguration;**

**import org.springframework.web.cors.CorsConfigurationSource;**

**//import org.springframework.web.cors.reactive.UrlBasedCorsConfigurationSource;**

**import org.springframework.web.cors.UrlBasedCorsConfigurationSource;**

**import com.cts.security.myuserDetailService;**

**@Configuration**

**@EnableWebSecurity**

**public class SecurityConfig {**

**@Autowired private myuserDetailService userDetailsService;**

**@Autowired private JwtFilter jwtFilter;**

**@Bean**

**public PasswordEncoder passwordEncoder() {**

**return new BCryptPasswordEncoder();**

**}**

**@Bean**

**public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {**

**return config.getAuthenticationManager();**

**}**

**@Bean**

**public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {**

**return http**

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

**.cors(Customizer.withDefaults())**

**.authorizeHttpRequests(auth -> auth**

**.requestMatchers("/auth/register", "/auth/login").permitAll()**

**.anyRequest().authenticated())**

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

**.authenticationProvider(authProvider())**

**.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)**

**.build();**

**}**

**@Bean**

**public DaoAuthenticationProvider authProvider() {**

**DaoAuthenticationProvider provider = new DaoAuthenticationProvider();**

**provider.setUserDetailsService(userDetailsService);**

**provider.setPasswordEncoder(passwordEncoder());**

**return provider;**

**}**

**// @Bean**

**// public CorsFilter corsFilter() {**

**// UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();**

**// CorsConfiguration config = new CorsConfiguration();**

**// config.setAllowedOrigins(Arrays.asList("http://localhost:4200"));**

**// config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));**

**// config.setAllowedHeaders(Arrays.asList("\*"));**

**// config.setAllowCredentials(true);**

**// source.registerCorsConfiguration("/\*\*", config);**

**// return new CorsFilter(source);**

**// }**

**@Bean**

**public CorsConfigurationSource corsConfigurationSource() { // <--- CHANGED METHOD NAME AND RETURN TYPE**

**UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();**

**CorsConfiguration config = new CorsConfiguration();**

**config.setAllowedOrigins(Arrays.asList("http://localhost:4200")); // Your Angular app's URL**

**config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));**

**config.setAllowedHeaders(Arrays.asList("\*")); // Allow all headers**

**config.setAllowCredentials(true); // Allow cookies, authorization headers etc.**

**source.registerCorsConfiguration("/\*\*", config); // Apply to all paths**

**return source; // Return the source directly**

**}**

**}**

**EventController.java**

package com.cts.controller;

import com.cts.entity.Event;

import com.cts.service.EventService;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/events")

public class EventController {

@Autowired

private EventService eventService;

@GetMapping("/all")

public ResponseEntity<List<Event>> getEvents() {

List<Event> events = eventService.getAllEvents();

return ResponseEntity.ok(events);

}

@GetMapping("/{id}")

public ResponseEntity<Event> getEventById(@PathVariable Long id) {

Event event = eventService.getEventById(id);

return ResponseEntity.ok(event);

}

@GetMapping("/category/{category}")

public ResponseEntity<Event> getEventsByCategory(@PathVariable String category) {

Event events = eventService.getEventByCategory(category);

return ResponseEntity.ok(events);

}

@PostMapping("/save")

public ResponseEntity<Event> createEvent(@RequestBody Event event) {

eventService.validateEvent(event);

Event savedEvent = eventService.saveEvent(event);

return ResponseEntity.status(201).body(savedEvent);

}

@PutMapping("/update/{id}")

public ResponseEntity<Event> updateEvent(@PathVariable Long id, @RequestBody Event updatedEvent) {

eventService.validateEvent(updatedEvent);

Event event = eventService.updateEvent(id, updatedEvent);

return ResponseEntity.ok(event);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEventById(@PathVariable Long id) {

eventService.deleteEventById(id);

return ResponseEntity.noContent().build();

}

}

**FeedbackController.java**

package com.cts.controller;

import com.cts.entity.Feedback;

import com.cts.service.FeedbackService;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/feedback")

public class FeedbackController {

@Autowired

private FeedbackService feedbackService;

@PostMapping("/submit")

public ResponseEntity<Feedback> submitFeedback(@RequestBody Feedback feedback) {

Feedback savedFeedback = feedbackService.saveFeedback(feedback);

return ResponseEntity.status(201).body(savedFeedback);

}

@GetMapping("/event/{eventId}")

public ResponseEntity<List<Feedback>> getFeedbackByEventId(@PathVariable Long eventId) {

List<Feedback> feedbackList = feedbackService.getFeedbackByEventId(eventId);

return ResponseEntity.ok(feedbackList);

}

@GetMapping("/event/{eventId}/average-rating")

public ResponseEntity<Double> getAverageRatingByEventId(@PathVariable Long eventId) {

double averageRating = feedbackService.getAverageRatingByEventId(eventId);

return ResponseEntity.ok(averageRating);

}

@GetMapping("/{id}")

public ResponseEntity<Feedback> getFeedbackById(@PathVariable Long id) {

Feedback feedback = feedbackService.getFeedbackById(id);

return ResponseEntity.ok(feedback);

}

}

**NotificationController.java**

package com.cts.controller;

import com.cts.entity.Event;

import com.cts.entity.User;

import com.cts.service.EventService;

import com.cts.service.NotificationService;

import com.cts.service.UserService;

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

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

@RestController

@RequestMapping("/notifications")

public class NotificationController {

@Autowired

private NotificationService emailService;

@Autowired

private UserService userService;

@Autowired

private EventService eventService;

// Default notification

@PostMapping("/send")

public String sendNotification(@RequestParam Long userId, @RequestParam Long eventId) {

User user = userService.getUserById(userId);

Event event = eventService.getEventById(eventId);

if (user == null || event == null) {

return "Invalid user or event ID.";

}

emailService.sendRegistrationEmail(user, event);

return "Notification sent successfully to " + user.getEmail();

}

// Custom notification

@PostMapping("/send-custom")

public String sendCustomNotification(

@RequestParam Long userId,

@RequestParam String subject,

@RequestParam String message) {

User user = userService.getUserById(userId);

if (user == null) {

return "Invalid user ID.";

}

emailService.sendCustomEmail(user.getEmail(), subject, message);

return "Custom notification sent successfully to " + user.getEmail();

}

}

**TicketController.java**

package com.cts.controller;

import com.cts.entity.Event;

import com.cts.entity.Ticket;

import com.cts.entity.User;

import com.cts.service.EventService;

import com.cts.service.TicketService;

import com.cts.service.UserService;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/tickets")

public class TicketController {

@Autowired

private TicketService ticketService;

@Autowired

private EventService eventService;

@Autowired

private UserService userService;

@PostMapping("/book")

public ResponseEntity<Ticket> bookTicket(@RequestParam Long eventId, @RequestParam Long userId) {

Event event = eventService.getEventById(eventId);

User user = userService.getUserById(userId);

Ticket ticket = ticketService.bookTicket(event,user);

return ResponseEntity.ok(ticket);

}

@GetMapping("/user/{userId}")

public ResponseEntity<List<Ticket>> getTicketsByUserId(@PathVariable Long userId) {

List<Ticket> tickets = ticketService.getTicketsByUserId(userId);

return ResponseEntity.ok(tickets);

}

@GetMapping("/event/{eventId}")

public ResponseEntity<List<Ticket>> getTicketsByEventId(@PathVariable Long eventId) {

List<Ticket> tickets = ticketService.getTicketsByEventId(eventId);

return ResponseEntity.ok(tickets);

}

@PutMapping("/cancel/{ticketId}")

public ResponseEntity<Ticket> cancelTicket(@PathVariable Long ticketId) {

Ticket ticket = ticketService.cancelTicket(ticketId);

return ResponseEntity.ok(ticket);

}

}

**UserController.java**

package com.cts.controller;

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

import org.springframework.http.ResponseEntity;

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

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

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

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

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

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

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

import com.cts.entity.User;

import com.cts.service.UserService;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService service;

@PostMapping("/register")

public User register(@RequestBody User user) {

service.validateUser(user);

return service.register(user);

}

@PostMapping("/login")

public String login(@RequestBody User user) {

return service.verify(user);

}

@PutMapping("/update")

public User update(@RequestBody User user) {

service.validateUser(user);

return service.update(user);

}

@GetMapping("/{id}")

public ResponseEntity<User> getUserById(@PathVariable Long id){

User user=service.getUserById(id);

return ResponseEntity.ok(user);

}

}

**EventDTO.java**

package com.cts.dto;

import java.sql.Date;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class EventDTO {

private Long id;

private String name;

private String category;

private String location;

private Date date;

private Long organizerId;

}

**FeedbackDTO.java**

package com.cts.dto;

import java.sql.Time;

import java.sql.Timestamp;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class FeedbackDTO {

private Long id;

private Long eventId;

private Long userId;

private int rating;

private String comments;

private Timestamp submittedTimestamp;

}

**NotificationDTO.java**

package com.cts.dto;

import java.sql.Time;

import java.sql.Timestamp;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class NotificationDTO {

private Long id;

private Long userId;

private Long eventId;

private String message;

private Timestamp sentTimestamp;

}

**TicketDTO.java**

package com.cts.dto;

import java.sql.Date;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class TicketDTO {

private Long id;

private Long eventId;

private Long userId;

private Date bookingDate;

private String status;

}

**UserDTO.java**

package com.cts.dto;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class UserDTO {

private Long id;

private String name;

private String email;

private String password;

private Long contactNumber;

}

**Event.java**

package com.cts.entity;

import java.sql.Date;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Event {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String category;

private String location;

private Date date;

private Long organizerId;

private int ticketCount;

}

**Feedback.java**

package com.cts.entity;

import java.sql.Time;

import java.sql.Timestamp;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.ManyToOne;

import jakarta.validation.constraints.NotBlank;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Feedback {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

@JoinColumn(name = "event\_id", nullable = false)

private Event event;

@ManyToOne

@JoinColumn(name = "user\_id", nullable = false)

private User user;

private int rating;

private String comments;

private Timestamp submittedTimestamp;

}

**Notification.java**

package com.cts.entity;

import java.sql.Timestamp;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.ManyToOne;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Notification {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

@JoinColumn(name = "user\_id", nullable = false)

private User user;

@ManyToOne

@JoinColumn(name = "event\_id", nullable = false)

private Event event;

private String message;

private Timestamp sentTimestamp;

}

**Role.java**

**package** com.cts.entity;

**public** **enum** Role {

***ADMIN***,

***ORGANIZER***,

***USER***

}

**Ticket.java**

package com.cts.entity;

import java.sql.Date;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.ManyToOne;

import jakarta.validation.constraints.NotBlank;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Ticket {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

@JoinColumn(name = "event\_id", nullable = false)

private Event event;

@ManyToOne

@JoinColumn(name = "user\_id", nullable = false)

private User user;

private Date bookingDate;

private String status;

}

**User.java**

package com.cts.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Pattern;

import jakarta.validation.constraints.Size;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String password;

private Long contactNumber;

@Enumerated(EnumType.STRING)

private Role role = Role.USER;

public Role getRole() {

return role;

}

}

**UserDetail.java**

package com.cts.entity;

import java.util.Collection;

import java.util.Collections;

import org.springframework.security.core.GrantedAuthority;

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

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

public class UserDetail implements UserDetails {

private User user;

public UserDetail(User user) {

this.user = user;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

// Assign authorities based on the user's role

return Collections.singleton(new SimpleGrantedAuthority(user.getRole().name()));

}

@Override

public String getPassword() {

return user.getPassword();

}

@Override

public String getUsername() {

return user.getName();

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

}

**AuthenticationException.java**

**package** com.cts.exception;

**public** **class** AuthenticationException {

}

**EventNotFoundEceptiom.java**

**package** com.cts.exception;

**public** **class** EventNotFoundException **extends** RuntimeException {

**public** EventNotFoundException(Long id) {

**super**("Event not found with id: " + id);

}

**public** EventNotFoundException(String category) {

**super**("Event not found with category: " + category);

}

}

**FeedbackNotFoundException.java**

**package** com.cts.exception;

**public** **class** FeedbackNotFoundException **extends** RuntimeException {

**public** FeedbackNotFoundException(Long id) {

**super**("Feedback not found with id: " + id);

}

}

**GlobalExceptionHandler.java**

package com.cts.exception;

import java.util.Map;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.MethodArgumentNotValidException;

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

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

import org.springframework.web.context.request.WebRequest;

import org.springframework.security.core.AuthenticationException; // Import AuthenticationException

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(AuthenticationException.class)

public ResponseEntity<String> handleAuthenticationException(AuthenticationException ex) {

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Authentication failed: " + ex.getMessage());

}

@ExceptionHandler(EventNotFoundException.class)

public ResponseEntity<String> handleEventNotFound(EventNotFoundException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(ex.getMessage());

}

@ExceptionHandler(UserNotFoundException.class)

public ResponseEntity<String> handleUserNotFound(UserNotFoundException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(ex.getMessage());

}

@ExceptionHandler(IllegalArgumentException.class)

public ResponseEntity<String> handleIllegalArgumentException(IllegalArgumentException ex) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body(ex.getMessage());

}

@ExceptionHandler(Exception.class)

public ResponseEntity<String> handleGlobalException(Exception ex, WebRequest request) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("An unexpected error occurred: " + ex.getMessage());

}

}

**NotificationNotFoundException.java**

**package** com.cts.exception;

**public** **class** NotificationNotFoundException **extends** RuntimeException {

**public** NotificationNotFoundException(Long id) {

**super**("Notification not found with id: " + id);

}

}

**TicketNotFoundException.java**

**package** com.cts.exception;

**public** **class** TicketNotFoundException **extends** RuntimeException {

**public** TicketNotFoundException(Long ticketId) {

**super**("Ticket not found with id: " + ticketId);

}

}

**UserNotFoundException.java**

**package** com.cts.exception;

**public** **class** UserNotFoundException **extends** RuntimeException {

**public** UserNotFoundException(Long id) {

**super**("User not found with id: " + id);

}

}

**JwtFilter.java**

package com.cts.filter;

import java.io.IOException;

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

import org.springframework.context.ApplicationContext;

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.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import com.cts.security.JwtService;

import com.cts.security.myuserDetailService;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@Component

public class JwtFilter extends OncePerRequestFilter {

@Autowired

private JwtService jwtService;

@Autowired

ApplicationContext context;

@Override

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

throws ServletException, IOException {

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

String token = null;

String username = null;

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

token = authHeader.substring(7);

username = jwtService.extractUserName(token);

}

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

UserDetails userDetails = context.getBean(myuserDetailService.class).loadUserByUsername(username);

if (jwtService.validateToken(token, userDetails)) {

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

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

SecurityContextHolder.getContext().setAuthentication(authToken);

// Role-based authorization

String role = jwtService.extractRolesFromToken(token);

String requestedPath = request.getRequestURI();

String method = request.getMethod();

if (!isAuthorized(role, requestedPath, method)) {

response.setStatus(HttpServletResponse.SC\_FORBIDDEN);

return;

}

}

}

filterChain.doFilter(request, response);

}

private boolean isAuthorized(String role, String path, String method) {

if ("ADMIN".equalsIgnoreCase(role)) {

return path.startsWith("/users") || path.startsWith("/feedback") || path.startsWith("/events") || path.startsWith("/tickets");

} else if ("ORGANIZER".equalsIgnoreCase(role)) {

return path.startsWith("/tickets") || path.startsWith("/events") || path.startsWith("/feedback") || (path.startsWith("/users") && method.equalsIgnoreCase("GET"));

} else if ("USER".equalsIgnoreCase(role)) {

return path.startsWith("/feedback") || (path.startsWith("/users") && (method.equalsIgnoreCase("POST") || method.equalsIgnoreCase("PUT"))) || (path.startsWith("/events//category/{category}") || path.startsWith("/tickets"));

}

return false;

}

}

**EventMapper.java**

package com.cts.mapper;

import org.springframework.stereotype.Component;

import com.cts.dto.EventDTO;

import com.cts.entity.Event;

@Component

public class EventMapper {

public EventDTO toDto(Event event) {

if (event == null) return null;

return EventDTO.builder()

.id(event.getId())

.name(event.getName())

.category(event.getCategory())

.location(event.getLocation())

.date(event.getDate())

.organizerId(event.getOrganizerId())

.build();

}

public Event toEntity(EventDTO dto) {

if (dto == null) return null;

return Event.builder()

.id(dto.getId())

.name(dto.getName())

.category(dto.getCategory())

.location(dto.getLocation())

.date(dto.getDate())

.organizerId(dto.getOrganizerId())

.build();

}

}

**FeedbackMapper.java**

package com.cts.mapper;

import org.springframework.stereotype.Component;

import com.cts.dto.FeedbackDTO;

import com.cts.entity.Feedback;

import com.cts.entity.Event;

import com.cts.entity.User;

@Component

public class FeedbackMapper {

public FeedbackDTO toDto(Feedback feedback) {

if (feedback == null) return null;

return FeedbackDTO.builder()

.id(feedback.getId())

.eventId(feedback.getEvent().getId())

.userId(feedback.getUser().getId())

.rating(feedback.getRating())

.comments(feedback.getComments())

.submittedTimestamp(feedback.getSubmittedTimestamp())

.build();

}

public Feedback toEntity(FeedbackDTO dto, Event event, User user) {

if (dto == null) return null;

return Feedback.builder()

.id(dto.getId())

.event(event)

.user(user)

.rating(dto.getRating())

.comments(dto.getComments())

.submittedTimestamp(dto.getSubmittedTimestamp())

.build();

}

}

**NotificationMapper.java**

package com.cts.mapper;

import org.springframework.stereotype.Component;

import com.cts.dto.NotificationDTO;

import com.cts.entity.Notification;

import com.cts.entity.Event;

import com.cts.entity.User;

@Component

public class NotificationMapper {

public NotificationDTO toDto(Notification notification) {

if (notification == null) return null;

return NotificationDTO.builder()

.id(notification.getId())

.userId(notification.getUser().getId())

.eventId(notification.getEvent().getId())

.message(notification.getMessage())

.sentTimestamp(notification.getSentTimestamp())

.build();

}

public Notification toEntity(NotificationDTO dto, Event event, User user) {

if (dto == null) return null;

return Notification.builder()

.id(dto.getId())

.user(user)

.event(event)

.message(dto.getMessage())

.sentTimestamp(dto.getSentTimestamp())

.build();

}

}

**TicketMapper.java**

package com.cts.mapper;

import org.springframework.stereotype.Component;

import com.cts.dto.TicketDTO;

import com.cts.entity.Ticket;

@Component

public class TicketMapper {

public TicketDTO toDto(Ticket ticket) {

if (ticket == null) return null;

return TicketDTO.builder()

.id(ticket.getId())

.eventId(ticket.getEvent().getId())

.userId(ticket.getUser().getId())

.bookingDate(ticket.getBookingDate())

.status(ticket.getStatus())

.build();

}

public Ticket toEntity(TicketDTO dto) {

if (dto == null) return null;

return Ticket.builder()

.id(dto.getId())

.bookingDate(dto.getBookingDate())

.status(dto.getStatus())

.build();

}

}

**UserMapper.java**

package com.cts.mapper;

import org.springframework.stereotype.Component;

import com.cts.dto.UserDTO;

import com.cts.entity.User;

@Component

public class UserMapper {

public UserDTO toDto(User user) {

if (user == null) return null;

return UserDTO.builder()

.id(user.getId())

.name(user.getName())

.email(user.getEmail())

.password(user.getPassword())

.contactNumber(user.getContactNumber())

.build();

}

public User toEntity(UserDTO dto) {

if (dto == null) return null;

return User.builder()

.id(dto.getId())

.name(dto.getName())

.email(dto.getEmail())

.password(dto.getPassword())

.contactNumber(dto.getContactNumber())

.build();

}

}

**EventRepository.java**

package com.cts.repository;

import java.util.List;

import java.util.Optional;

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

import com.cts.entity.Event;

public interface EventRepository extends JpaRepository<Event, Long>{

Optional<Event> findByCategory(String category);

Optional<Event> findById(Long id);

}

**FeedbackRepository.java**

package com.cts.repository;

import com.cts.entity.Feedback;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface FeedbackRepository extends JpaRepository<Feedback, Long> {

List<Feedback> findByEventId(Long eventId);

}

**NotificationRepository.java**

package com.cts.repository;

import com.cts.entity.Notification;

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

public interface NotificationRepository extends JpaRepository<Notification, Long> {

}

**TicketRepository.java**

package com.cts.repository;

import com.cts.entity.Ticket;

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

import java.util.List;

public interface TicketRepository extends JpaRepository<Ticket, Long> {

List<Ticket> findByUserId(Long userId);

List<Ticket> findByEventId(Long eventId);

}

**UserRepo.java**

package com.cts.repository;

import java.util.Optional;

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

import org.springframework.stereotype.Repository;

import com.cts.entity.User;

@Repository

public interface UserRepo extends JpaRepository<User,Long> {

User findByName(String name);

}

**JwtService.java**

package com.cts.security;

import java.security.NoSuchAlgorithmException;

import java.util.Base64;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

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

import org.springframework.stereotype.Service;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.io.Decoders;

import io.jsonwebtoken.security.Keys;

@Service

public class JwtService {

private String Secure\_Key;

public JwtService() {

try {

KeyGenerator keyGenerator = KeyGenerator.getInstance("HmacSHA256");

SecretKey sk = keyGenerator.generateKey();

Secure\_Key = Base64.getEncoder().encodeToString(sk.getEncoded());

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

}

@SuppressWarnings("deprecation")

public String generateToken(String Username) {

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

return Jwts.builder()

.setClaims(claims)

.setSubject(Username)

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + (60 \* 10 \* 60 \* 30)))

.signWith(getKey())

.compact();

}

private SecretKey getKey() {

byte[] ByteKey = Decoders.BASE64.decode(Secure\_Key);

return Keys.hmacShaKeyFor(ByteKey);

}

public String extractUserName(String token) {

return extractClaim(token, Claims::getSubject);

}

public String extractRolesFromToken(String token) {

Claims claims = extractAllClaims(token);

return claims.get("role", String.class);

}

private <T> T extractClaim(String token, Function<Claims, T> claimResolver) {

final Claims claims = extractAllClaims(token);

return claimResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser()

.verifyWith(getKey())

.build()

.parseSignedClaims(token)

.getPayload();

}

public boolean validateToken(String token, UserDetails userDetails) {

final String userName = extractUserName(token);

return (userName.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

private boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

private Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

}

**myuserDetailService.java**

package com.cts.security;

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.cts.entity.User;

import com.cts.entity.UserDetail;

import com.cts.repository.UserRepo;

@Service

public class myuserDetailService implements UserDetailsService {

@Autowired

private UserRepo userRepo;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user=userRepo.findByName(username);

if(user == null) {

System.out.println("Not an active user");

throw new UsernameNotFoundException("User not found");

}

return new UserDetail(user);

}

}

**EventService.java**

package com.cts.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cts.entity.Event;

import com.cts.exception.EventNotFoundException;

import com.cts.repository.EventRepository;

@Service

public class EventService {

@Autowired

private EventRepository eventRepository;

public List<Event> getAllEvents() {

return eventRepository.findAll();

}

public Event getEventById(Long id) {

return eventRepository.findById(id).orElseThrow(() -> new EventNotFoundException(id));

}

public Event getEventByCategory(String category) {

return eventRepository.findByCategory(category).orElseThrow(() -> new EventNotFoundException(category));

}

public Event saveEvent(Event event) {

validateEvent(event);

return eventRepository.save(event);

}

public Event updateEvent(Long id, Event updatedEvent) {

validateEvent(updatedEvent);

Event existingEvent = eventRepository.findById(id).orElseThrow(() -> new EventNotFoundException(id));

existingEvent.setName(updatedEvent.getName());

existingEvent.setCategory(updatedEvent.getCategory());

existingEvent.setLocation(updatedEvent.getLocation());

existingEvent.setDate(updatedEvent.getDate());

existingEvent.setOrganizerId(updatedEvent.getOrganizerId());

existingEvent.setTicketCount(updatedEvent.getTicketCount());

return eventRepository.save(existingEvent);

}

public void deleteEventById(Long id) {

if (!eventRepository.existsById(id)) {

throw new EventNotFoundException(id);

}

eventRepository.deleteById(id);

}

public void decreaseTicketCount(Long eventId) {

Event event = getEventById(eventId);

if (event.getTicketCount() <= 0) {

throw new IllegalArgumentException("No tickets available");

}

event.setTicketCount(event.getTicketCount() - 1);

eventRepository.save(event);

}

public void increaseTicketCount(Long eventId) {

Event event = getEventById(eventId);

event.setTicketCount(event.getTicketCount() + 1);

eventRepository.save(event);

}

public void validateEvent(Event event) {

if (event.getName() == null || event.getName().isEmpty()) {

throw new IllegalArgumentException("Event name is required");

}

if (event.getCategory() == null || event.getCategory().isEmpty()) {

throw new IllegalArgumentException("Category is required");

}

if (event.getLocation() == null || event.getLocation().isEmpty()) {

throw new IllegalArgumentException("Location is required");

}

if (event.getDate() == null) {

throw new IllegalArgumentException("Date is required");

}

if (event.getOrganizerId() == null) {

throw new IllegalArgumentException("Organizer ID is required");

}

if (event.getTicketCount() <= 0) {

throw new IllegalArgumentException("Ticket count must be positive");

}

}

}

**FeedbackService.java**

package com.cts.service;

import com.cts.entity.Feedback;

import com.cts.exception.FeedbackNotFoundException;

import com.cts.repository.FeedbackRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class FeedbackService {

@Autowired

private FeedbackRepository feedbackRepository;

public Feedback saveFeedback(Feedback feedback) {

return feedbackRepository.save(feedback);

}

public List<Feedback> getFeedbackByEventId(Long eventId) {

return feedbackRepository.findByEventId(eventId);

}

public double getAverageRatingByEventId(Long eventId) {

List<Feedback> feedbackList = feedbackRepository.findByEventId(eventId);

return feedbackList.stream()

.mapToInt(Feedback::getRating)

.average()

.orElse(0.0);

}

public Feedback getFeedbackById(Long id) {

return feedbackRepository.findById(id).orElseThrow(() -> new FeedbackNotFoundException(id));

}

}

**NotificationService.java**

package com.cts.service;

import com.cts.entity.Event;

import com.cts.entity.User;

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

import org.springframework.mail.SimpleMailMessage;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.stereotype.Service;

@Service

public class NotificationService {

@Autowired

private JavaMailSender mailSender;

// Default registration email

public void sendRegistrationEmail(User user, Event event) {

SimpleMailMessage message = new SimpleMailMessage();

message.setTo(user.getEmail());

message.setSubject("Event Registration Confirmation");

message.setText("Hello " + user.getName() + ",\n\n" +

"You have successfully registered for the event:\n" +

"Event: " + event.getName() + "\n" +

"Location: " + event.getLocation() + "\n" +

"Date: " + event.getDate() + "\n\n" +

"Thank you for registering!\nEvent Management Team");

mailSender.send(message);

}

// Custom email

public void sendCustomEmail(String toEmail, String subject, String messageBody) {

SimpleMailMessage message = new SimpleMailMessage();

message.setTo(toEmail);

message.setSubject(subject);

message.setText(messageBody);

mailSender.send(message);

}

}

**TicketService.java**

package com.cts.service;

import com.cts.entity.Event;

import com.cts.entity.Ticket;

import com.cts.entity.User;

import com.cts.exception.TicketNotFoundException;

import com.cts.repository.TicketRepository;

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

import org.springframework.stereotype.Service;

import java.sql.Date;

import java.util.List;

@Service

public class TicketService {

@Autowired

private TicketRepository ticketRepository;

@Autowired

private EventService eventService;

public Ticket bookTicket(Event event,User user) {

eventService.decreaseTicketCount(event.getId());

Ticket ticket = Ticket.builder()

.event(event)

.user(user)

.bookingDate(new Date(System.currentTimeMillis()))

.status("Confirmed")

.build();

return ticketRepository.save(ticket);

}

public List<Ticket> getTicketsByUserId(Long userId) {

return ticketRepository.findByUserId(userId);

}

public List<Ticket> getTicketsByEventId(Long eventId) {

return ticketRepository.findByEventId(eventId);

}

public Ticket cancelTicket(Long ticketId) {

Ticket ticket = ticketRepository.findById(ticketId).orElseThrow(() -> new TicketNotFoundException(ticketId));

ticket.setStatus("Cancelled");

eventService.increaseTicketCount(ticket.getEvent().getId());

return ticketRepository.save(ticket);

}

}

**UserService.java**

package com.cts.service;

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

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

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

import org.springframework.stereotype.Service;

import com.cts.entity.User;

import com.cts.exception.UserNotFoundException;

import com.cts.repository.UserRepo;

import com.cts.security.JwtService;

@Service

public class UserService {

@Autowired

private UserRepo repo;

@Autowired

private AuthenticationManager authman;

@Autowired

private JwtService service;

private BCryptPasswordEncoder encrypt=new BCryptPasswordEncoder(12);

public User register(User user) {

validateUser(user);

user.setPassword(encrypt.encode(user.getPassword()));

return repo.save(user);

}

public String verify(User user) {

Authentication authentication=authman.authenticate(new UsernamePasswordAuthenticationToken(user.getName(),user.getPassword()));

if(authentication.isAuthenticated()) {

return "Login Success";

}

else {

return "Failure";

}

}

public User update(User user) {

User temp\_user=repo.findByName(user.getName());

temp\_user.setName(user.getName());

temp\_user.setPassword(encrypt.encode(user.getPassword()));

repo.save(temp\_user);

return temp\_user;

}

public User getUserById(Long id) {

return repo.findById(id).orElseThrow(() -> new UserNotFoundException(id));

}

public void validateUser(User user) {

if (user.getName() == null || user.getName().isEmpty()) {

throw new IllegalArgumentException("Name is required");

}

if (user.getEmail() == null || user.getEmail().isEmpty() || !user.getEmail().matches("^[A-Za-z0-9+\_.-]+@(.+)$")) {

throw new IllegalArgumentException("Valid email is required");

}

if (user.getPassword() == null || user.getPassword().isEmpty() || user.getPassword().length() < 8) {

throw new IllegalArgumentException("Password must be at least 8 characters long");

}

if (user.getContactNumber() == null || !String.valueOf(user.getContactNumber()).matches("\\d{10}")) {

throw new IllegalArgumentException("Contact number must be 10 digits");

}

}

}

**application.properties**

spring.application.name=EventManagement

server.port=2061

#====================

# DATABASE CONNECTION

#====================

spring.datasource.url=jdbc:mysql://localhost:3306/event\_management

spring.datasource.username=root

spring.datasource.password=420007

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#====================

# JPA / HIBERNATE

#====================

#spring.security.user.name=arun07

#spring.security.user.password=1234567

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.properties.hibernate.format\_sql=true

#====================

# EMAIL CONFIGURATION

#====================

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=nedunurisahitya7@gmail.com

spring.mail.password=pwrbvjolkgdcoyfm

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

spring.mail.properties.mail.smtp.connectiontimeout=10000

spring.mail.properties.mail.smtp.timeout=10000

spring.mail.properties.mail.smtp.writetimeout=10000

spring.mvc.throw-exception-if-no-handler-found=true

spring.mvc.dispatch-options-request=true

**pom.xml**

<?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>3.4.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cts</groupId>

<artifactId>Spring\_Security</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Spring\_Security</name>

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

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.12.6</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.12.6</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.12.6</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.8.6</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>