API GATEWAY

package com.example.Apigateway;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

import org.springframework.web.reactive.config.EnableWebFlux;

@SpringBootApplication

public class ApiGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(ApiGatewayApplication.class, args);

}

}

package com.example.Apigateway.filter;

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

import org.springframework.cloud.gateway.filter.GatewayFilter;

import org.springframework.cloud.gateway.filter.factory.AbstractGatewayFilterFactory;

import org.springframework.http.HttpStatus;

import org.springframework.http.server.reactive.ServerHttpResponse;

import org.springframework.stereotype.Component;

import com.example.Apigateway.util.JwtUtil;

import com.google.common.net.HttpHeaders;

import reactor.core.publisher.Mono;

*@Component*

public class AuthenticationFilter extends AbstractGatewayFilterFactory<AuthenticationFilter.Config> {

*@Autowired*

private RouteValidator validator;

*@Autowired*

private JwtUtil util;

public static class Config {

}

public AuthenticationFilter() {

super(Config.class);

}

*@Override*

public GatewayFilter apply(Config config) {

return (exchange, chain) -> {

if (validator.isSecured.test(exchange.getRequest())) {

if (!exchange.getRequest().getHeaders().containsKey(HttpHeaders.***AUTHORIZATION***)) {

return handleUnauthorized(exchange.getResponse(), "Missing authorization header");

}

String authHeader = exchange.getRequest().getHeaders().getFirst(HttpHeaders.***AUTHORIZATION***);

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

authHeader = authHeader.substring(7);

}

try {

String role = util.extractRolesFromToken(authHeader);

String requestedPath = exchange.getRequest().getPath().toString();

String method = exchange.getRequest().getMethod().name();

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

return handleUnauthorized(exchange.getResponse(), "Unauthorized access");

}

} catch (Exception e) {

return handleUnauthorized(exchange.getResponse(), "Invalid token");

}

}

return chain.filter(exchange);

};

}

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

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

return path.startsWith("/users") || path.startsWith("/feedbackandratings") || path.startsWith("/events")

||path.startsWith("/tickets");

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

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

|| (path.startsWith("/feedbackandratings") && method.equalsIgnoreCase("GET")) || (path.startsWith("/users")&& method.equalsIgnoreCase("GET"));

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

return (path.startsWith("/feedbackandratings")&& (method.equalsIgnoreCase("PUT") || method.equalsIgnoreCase("POST")))||(path.startsWith("/users") && (method.equalsIgnoreCase("POST") || method.equalsIgnoreCase("PUT")))

|| path.startsWith("/events") && method.equalsIgnoreCase("GET") || path.startsWith("/ticket");

}

return false;

}

private Mono<Void> handleUnauthorized(ServerHttpResponse response, String message) {

response.setStatusCode(*HttpStatus*.***FORBIDDEN***);

return response.setComplete();

}

}

package com.example.Apigateway.filter;

import java.util.function.Predicate;

import org.springframework.http.server.reactive.ServerHttpRequest;

import org.springframework.stereotype.Component;

@Component

public class RouteValidator {

public static final String[] OPEN\_API\_ENDPOINTS = { "/auth/register", "/auth/new", "/auth/validate", "/eureka" };

public Predicate<ServerHttpRequest> isSecured = request -> {

String path = request.getPath().toString();

for (String endpoint : OPEN\_API\_ENDPOINTS) {

if (path.contains(endpoint)) {

return false; // Endpoint does not require authorization

}

}

return true; // Endpoint requires authorization

};

}

package com.example.Apigateway.util;

import java.security.Key;

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 JwtUtil {

public static final String SECRET = "d3780ec3d1cfaba271e0538d4fae686d8367e10155ee424691fbf191eabec53d";

public void validateToken(final String token) {

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

}

public String extractRolesFromToken(final String token) {

Claims claims = Jwts.parserBuilder().setSigningKey(getSignKey()).build().parseClaimsJws(token).getBody();

System.out.println("Step2"+ claims);

String authorities = (String) claims.get("roles"); // Assuming roles are stored as a List<String> in "roles" claim

System.out.println("Step3"+ authorities);

// Convert authorities to roles in the format "ROLE\_ADMIN" or "ROLE\_USER"

return authorities;

}

private Key getSignKey() {

byte[] keyBytes = Decoders.BASE64.decode(SECRET);

return Keys.hmacShaKeyFor(keyBytes);

}

}

CONFIG SERVER

package com.cts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication

@EnableConfigServer

public class ConfigServerApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigServerApplication.class, args);

}

}

EUREKA SERVER

package com.cts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class EurekaServerApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaServerApplication.class, args);

}

}

EVENT MANAGEMENT

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);

}

}

package com.cts.controller;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

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

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

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.exceptions.EventNotFoundException;

import com.cts.exceptions.InvalidDataException;

import com.cts.model.EventManagement;

import com.cts.service.EventManagementService;

@RestController

@RequestMapping("/events")

public class EventManagementController {

@Autowired

private EventManagementService service;

@PostMapping("/saveevent")

public String saveEvent(@RequestBody EventManagement event) throws InvalidDataException {

return service.saveEvent(event);

}

@GetMapping("/fetcheventbyid/{eid}")

public EventManagement getEventById(@PathVariable("eid") int id) throws EventNotFoundException {

return service.getEventById(id);

}

@PutMapping("/updateevent/{eid}")

public String updateEvent(@PathVariable("eid") int id, @RequestBody EventManagement updatedEvent) throws InvalidDataException {

return service.updateEvent(id, updatedEvent);

}

@DeleteMapping("/deleteevent/{eid}")

public String deleteEventById(@PathVariable("eid") int id) throws EventNotFoundException {

return service.deleteEventById(id);

}

@GetMapping("/fetchallevents")

public List<EventManagement> getAllEvents() {

return service.getAllEvents();

}

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

public List<EventManagement> getEventByCategory(@PathVariable("category") String category)

throws EventNotFoundException {

return service.getEventByCategory(category);

}

@PostMapping("decreaseTicketCount/{id}")

public void decreaseTicketCount(@PathVariable("id") int id) throws EventNotFoundException {

service.decreaseTicketCount(id);

}

@PostMapping("increaseTicketCount/{id}")

public void increaseTicketCount(@PathVariable("id") int id) throws EventNotFoundException {

service.increaseTicketCount(id);

}

@GetMapping("/fetcheventbylocation/{location}")

public List<EventManagement> getEventByLocation(@PathVariable("location") String location) throws EventNotFoundException {

return service.getEventByLocation(location);

}

@GetMapping("/fetcheventbydate/{date}")

public List<EventManagement> getEventByDate(@PathVariable("date") String date) throws EventNotFoundException, ParseException {

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");

Date formattedDate = formatter.parse(date);

System.out.println(formattedDate);

return service.getEventByDate(formattedDate);

}

}

package com.cts.exceptions;

public class EventNotFoundException extends Exception{

public EventNotFoundException(String msg) {

// **TODO** Auto-generated constructor stub

super(msg);

}

}

package com.cts.exceptions;

public class InvalidDataException extends Exception {

public InvalidDataException(String msg) {

super(msg);

}

}

package com.cts.model;

import java.sql.Date;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.PrePersist;

import jakarta.persistence.PreUpdate;

import jakarta.validation.constraints.Future;

import jakarta.validation.constraints.Min;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.NotNull;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class EventManagement {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@NotBlank(message = "Event name is mandatory")

private String eventName;

@NotBlank(message = "Event category is mandatory")

private String eventCategory;

@NotBlank(message = "Event location is mandatory")

private String eventLocation;

@Future(message = "Event date must be in the future")

@NotNull(message = "Event date is mandatory")

private Date eventDate;

@Min(value = 1, message = "Event organizer ID must be at least 1")

private int eventOrganizerId;

@Min(value = 1, message = "Ticket count must be at least 1")

private int ticketCount;

@PrePersist

@PreUpdate

public void formatToLowerCase() {

if (eventName != null) {

eventName = eventName.toLowerCase();

}

if (eventCategory != null) {

eventCategory = eventCategory.toLowerCase();

}

if (eventLocation != null) {

eventLocation = eventLocation.toLowerCase();

}

}

}

package com.cts.repository;

import java.util.Date;

import java.util.List;

import java.util.Optional;

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

import com.cts.model.EventManagement;

public interface EventManagementRepository extends JpaRepository<EventManagement,Integer> {

List<EventManagement> findByEventCategory(String category);

List<EventManagement> findByEventDate(Date date);

List<EventManagement> findByEventLocation(String location);}

package com.cts.repository;

import java.util.Date;

import java.util.List;

import java.util.Optional;

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

import com.cts.model.EventManagement;

public interface EventManagementRepository extends JpaRepository<EventManagement,Integer> {

List<EventManagement> findByEventCategory(String category);

List<EventManagement> findByEventDate(Date date);

List<EventManagement> findByEventLocation(String location);

}

package com.cts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication

@EnableFeignClients

public class FeedbackAndRatingsApplication {

public static void main(String[] args) {

SpringApplication.run(FeedbackAndRatingsApplication.class, args);

}

}

package com.cts.controller;

import java.util.List;

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

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

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.exceptions.FeedbackAndRatingsNotFoundException;

import com.cts.model.FeedbackAndRatings;

import com.cts.service.FeedbackAndRatingsService;

@RestController

@RequestMapping("/feedbackandratings")

public class FeedbackAndRatingsController {

@Autowired

FeedbackAndRatingsService service;

@PostMapping("/save")

public String saveFeedback(@RequestBody FeedbackAndRatings feedback) {

return service.saveFeedback(feedback);

}

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

public String updateFeedback(@PathVariable("id") int id, @RequestBody FeedbackAndRatings feedback)

throws FeedbackAndRatingsNotFoundException {

return service.updateFeedback(id, feedback);

}

@DeleteMapping("/delete/{id}")

public String deleteFeedback(@PathVariable("id") int id) throws FeedbackAndRatingsNotFoundException {

return service.deleteFeedback(id);

}

@GetMapping("getfeedbackbyid/{id}")

public FeedbackAndRatings getFeedbackById(@PathVariable("id") int id) throws FeedbackAndRatingsNotFoundException {

return service.getFeedbackById(id);

}

@GetMapping("/getfeedbackbyeventid/{eventid}")

public List<FeedbackAndRatings> getAllFeedbacksByEvent(@PathVariable("eventid") int eventId) {

return service.getAllFeedbacksByEvent(eventId);

}

@GetMapping("/eventavg/{eventid}")

public Double getAverageRatingForEvent(@PathVariable("eventid") int eventId) {

return service.getAverageRatingForEvent(eventId);

}

@GetMapping("/fetchallfeedback")

public List<FeedbackAndRatings> getAllfeedbacks() {

return service.getAllfeedbacks();

}

}

package com.cts.dto;

import lombok.Data;

*@Data*

public class EventManagement {

private int id;

}

package com.cts.dto;

public class UserRegistration {

private int userId;

}

package com.cts.exceptions;

public class FeedbackAndRatingsNotFoundException extends Exception {

public FeedbackAndRatingsNotFoundException(String msg) {

super(msg);

}

}

package com.cts.feignclient;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import com.cts.dto.EventManagement;

@FeignClient(name="EVENTMANAGEMENT",path="events")

public interface EventManagementClient {

@GetMapping("/fetcheventbyid/{eid}")

public EventManagement getEventById(@PathVariable("eid") int id);}

package com.cts.feignclient;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import com.cts.dto.UserRegistration;

@FeignClient(name = "USERREGISTRATION", path = "users")

public interface UserRegistrationClient {

@GetMapping("/{id}")

public UserRegistration getUserById(@PathVariable int id);

}

package com.cts.model;

import java.time.LocalDateTime;

import org.hibernate.validator.constraints.Range;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Tuple;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Size;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class FeedbackAndRatings {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@NotNull(message = "Event ID cannot be null")

private int eventId;

@NotNull(message = "User ID cannot be null")

private int userId;

@Range(min = 1, max = 5, message = "Rating must be between 1 and 5")

private Double rating;

@Size(max = 500, message = "Comments cannot exceed 500 characters")

private String comments;

private LocalDateTime submittedTimestamp;

}

package com.cts.repository;

import java.util.List;

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

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

import com.cts.model.FeedbackAndRatings;

public interface FeedbackAndRatingsRepository extends JpaRepository<FeedbackAndRatings, Integer> {

List<FeedbackAndRatings> findByUserId(int userId);

List<FeedbackAndRatings> findByEventId(int eventId);

@Query("SELECT AVG(f.rating) FROM FeedbackAndRatings f WHERE f.eventId = :eventId")

Double findAverageRatingByEventId(int eventId);

//Optional<FeedbackAndRatings> findById();

}

package com.cts.service;

import java.util.List;

import com.cts.exceptions.FeedbackAndRatingsNotFoundException;

import com.cts.model.FeedbackAndRatings;

import jakarta.persistence.Tuple;

public interface FeedbackAndRatingsService {

public abstract String saveFeedback(FeedbackAndRatings feedback);

public abstract String updateFeedback(int feedbackId, FeedbackAndRatings feedback) throws FeedbackAndRatingsNotFoundException;

public abstract String deleteFeedback(int feedbackId) throws FeedbackAndRatingsNotFoundException;

public abstract FeedbackAndRatings getFeedbackById(int feedbackId) throws FeedbackAndRatingsNotFoundException;

public abstract List<FeedbackAndRatings> getAllfeedbacks();

public abstract List<FeedbackAndRatings> getAllFeedbacksByEvent(int eventId);

public abstract Double getAverageRatingForEvent(int eventId);

}

package com.cts.service;

import java.time.LocalDateTime;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.cts.dto.EventManagement;

import com.cts.dto.UserRegistration;

import com.cts.exceptions.FeedbackAndRatingsNotFoundException;

import com.cts.feignclient.EventManagementClient;

import com.cts.feignclient.UserRegistrationClient;

import com.cts.model.FeedbackAndRatings;

import com.cts.repository.FeedbackAndRatingsRepository;

@Service

public class FeedbackAndRatingsServiceImpl implements FeedbackAndRatingsService {

// Logger instance for logging

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

@Autowired

FeedbackAndRatingsRepository repository;

@Autowired

EventManagementClient eventClient;

@Autowired

UserRegistrationClient userClient;

@Override

public String saveFeedback(FeedbackAndRatings feedback) {

logger.info("Saving feedback for event ID: {} and user ID: {}", feedback.getEventId(), feedback.getUserId());

// Check if the event exists

EventManagement eventId = eventClient.getEventById(feedback.getEventId());

// Check if the user exists

UserRegistration userId = userClient.getUserById(feedback.getUserId());

if (userId == null) {

logger.error("User not found with ID: {}", feedback.getUserId());

return "Enter valid userId";

}

// Set the submitted timestamp and save the feedback

feedback.setSubmittedTimestamp(LocalDateTime.now());

repository.save(feedback);

logger.info("Feedback saved successfully for event ID: {} and user ID: {}", feedback.getEventId(), feedback.getUserId());

return "Feedback saved successfully";

}

@Override

public String updateFeedback(int feedbackId, FeedbackAndRatings feedback)

throws FeedbackAndRatingsNotFoundException {

logger.info("Updating feedback with ID: {}", feedbackId);

FeedbackAndRatings existingFeedback = getFeedbackById(feedbackId);

FeedbackAndRatings updatedFeedback = existingFeedback;

updatedFeedback.setRating(feedback.getRating());

updatedFeedback.setComments(feedback.getComments());

repository.save(updatedFeedback);

logger.info("Feedback updated successfully with ID: {}", feedbackId);

return "Feedback updated successfully";

}

@Override

public String deleteFeedback(int feedbackId) throws FeedbackAndRatingsNotFoundException {

FeedbackAndRatings feedback = getFeedbackById(feedbackId);

logger.info("Deleting feedback with ID: {}", feedbackId);

repository.deleteById(feedbackId);

logger.info("Feedback deleted successfully with ID: {}", feedbackId);

return "Feedback deleted successfully";

}

@Override

public FeedbackAndRatings getFeedbackById(int feedbackId) throws FeedbackAndRatingsNotFoundException {

logger.info("Fetching feedback with ID: {}", feedbackId);

Optional<FeedbackAndRatings> optional = repository.findById(feedbackId);

if (optional.isPresent()) {

logger.info("Feedback found with ID: {}", feedbackId);

return optional.get();

} else {

logger.error("Feedback not found with ID: {}", feedbackId);

throw new FeedbackAndRatingsNotFoundException("Feedback not found with id: " + feedbackId);

}

}

@Override

public List<FeedbackAndRatings> getAllFeedbacksByEvent(int eventId) {

logger.info("Fetching all feedbacks for event ID: {}", eventId);

return repository.findByEventId(eventId);

}

@Override

public Double getAverageRatingForEvent(int eventId) {

logger.info("Fetching average rating for event ID: {}", eventId);

Double avgRating = repository.findAverageRatingByEventId(eventId);

return (avgRating != null) ? avgRating : 0.0;

}

@Override

public List<FeedbackAndRatings> getAllfeedbacks() {

logger.info("Fetching all feedbacks");

return repository.findAll();

}

}

package com.cts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class GateWayApplication {

public static void main(String[] args) {

SpringApplication.run(GateWayApplication.class, args);

}

}

package com;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringSecurityLatestApplication {

public static void main(String[] args) {

SpringApplication.run(SpringSecurityLatestApplication.class, args);

}}

package com.config;

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.AuthenticationProvider;

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.annotation.web.configuration.EnableWebSecurity;

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

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.web.SecurityFilterChain;

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

import com.filter.JwtAuthFilter;

@Configuration

@EnableWebSecurity

@EnableMethodSecurity

public class SecurityConfig {

@Autowired

private JwtAuthFilter authFilter;

//authentication

@Bean

UserDetailsService userDetailsService() {

return new UserInfoUserDetailsService();

}

@Bean

SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

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

.authorizeHttpRequests(requests -> requests

.requestMatchers("/auth/authenticate", "/auth/getroles/\*\*", "/auth/\*\*").permitAll())

.authorizeHttpRequests(requests -> requests.requestMatchers("/users/\*\*","/events/\*\*","/tickets/\*\*","/feedbackandratings/\*\*")

.authenticated())

.sessionManagement(management -> management

.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

.authenticationProvider(authenticationProvider())

.addFilterBefore(authFilter, UsernamePasswordAuthenticationFilter.class)

.build();

}

@Bean

PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

AuthenticationProvider authenticationProvider(){

DaoAuthenticationProvider authenticationProvider=new DaoAuthenticationProvider();

authenticationProvider.setUserDetailsService(userDetailsService());

authenticationProvider.setPasswordEncoder(passwordEncoder());

return authenticationProvider;

}

@Bean

AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

return config.getAuthenticationManager();

}

}

package com.config;

import java.util.Arrays;

import java.util.Collection;

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.entity.UserInfo;

public class UserInfoUserDetails implements UserDetails {

private String name;

private String password;

private List<GrantedAuthority> authorities;

public UserInfoUserDetails(UserInfo userInfo) {

name=userInfo.getName();

password=userInfo.getPassword();

authorities= Arrays.stream(userInfo.getRoles().split(","))

.map(SimpleGrantedAuthority::new)

.collect(Collectors.toList());

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return authorities;

}

@Override

public String getPassword() {

return password;

}

@Override

public String getUsername() {

return name;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

}

package com.config;

import com.entity.UserInfo;

import com.repository.UserInfoRepository;

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.Component;

import java.util.Optional;

@Component

public class UserInfoUserDetailsService implements UserDetailsService {

@Autowired

private UserInfoRepository repository;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

Optional<UserInfo> userInfo = repository.findByName(username);

return userInfo.map(UserInfoUserDetails::new)

.orElseThrow(() -> new UsernameNotFoundException("user not found " + username));

}

}

package com.controller;

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.core.userdetails.UsernameNotFoundException;

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

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.RequestBody;

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

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

import com.dto.AuthRequest;

import com.entity.UserInfo;

import com.repository.UserInfoRepository;

import com.service.JwtService;

import com.service.UserService;

@RestController

@RequestMapping("/auth")

@CrossOrigin("\*")

public class AuthController {

@Autowired

private UserService service;

@Autowired

private JwtService jwtService;

@Autowired

private UserInfoRepository repo;

@Autowired

private AuthenticationManager authenticationManager;

@GetMapping("/welcome") //http://localhost:9090/auth/welcome

public String welcome() {

return "Welcome this endpoint is not secure";

}

@PostMapping("/new") //http://localhost:9090/auth/new

public String addNewUser(@RequestBody UserInfo userInfo) {

return service.addUser(userInfo);

}

@PostMapping("/authenticate") //http://localhost:9090/auth/authenticate

public String authenticateAndGetToken(@RequestBody AuthRequest authRequest) {

Authentication authentication = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword()));

if (authentication.isAuthenticated()) {

UserInfo obj = repo.findByName(authRequest.getUsername()).orElse(null);

return jwtService.generateToken(authRequest.getUsername(),obj.getRoles());

} else {

throw new UsernameNotFoundException("invalid user request !");

}

}

@GetMapping("/getroles/{username}") //http://localhost:9090/auth/getroles/{username}

public String getRoles(@PathVariable String username)

{

return service.getRoles(username);

}

}

package com.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class AuthRequest {

private String username;

private String 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;

}

}

package com.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@AllArgsConstructor

@NoArgsConstructor

public class UserInfo {

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Id

private int id;

private String name;

private String email;

private String password;

private String roles;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getRoles() {

return roles;

}

public void setRoles(String roles) {

this.roles = roles;

}

}

package com.filter;

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.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import com.config.UserInfoUserDetailsService;

import com.service.JwtService;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@Component

public class JwtAuthFilter extends OncePerRequestFilter {

@Autowired

private JwtService jwtService;

@Autowired

private UserInfoUserDetailsService userDetailsService;

@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 = userDetailsService.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);

}

}

filterChain.doFilter(request, response);

}

}

package com.repository;

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

import com.entity.UserInfo;

import java.util.List;

import java.util.Optional;

public interface UserInfoRepository extends JpaRepository<UserInfo, Integer> {

Optional<UserInfo> findByName(String username);

}

package com.service;

import java.security.Key;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

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

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.io.Decoders;

import io.jsonwebtoken.security.Keys;

@Component

public class JwtService {

public static final String SECRET = "d3780ec3d1cfaba271e0538d4fae686d8367e10155ee424691fbf191eabec53d";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

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

}

private Boolean isTokenExpired(String token) {

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

}

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

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

}

public String generateToken(String userName, String roles) {

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

claims.put("roles", roles);

System.out.println(claims);

return createToken(claims, userName);

}

private String createToken(Map<String, Object> claims, String userName) {

return Jwts.builder().setClaims(claims).setSubject(userName).setIssuedAt(new Date(System.currentTimeMillis()))

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

.signWith(getSignKey(), SignatureAlgorithm.HS256).compact();

}

private Key getSignKey() {

byte[] keyBytes = Decoders.BASE64.decode(SECRET);

return Keys.hmacShaKeyFor(keyBytes);

}

}

package com.service;

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

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

import org.springframework.stereotype.Service;

import com.entity.UserInfo;

import com.repository.UserInfoRepository;

@Service

public class UserService {

@Autowired

private UserInfoRepository repository;

@Autowired

private PasswordEncoder passwordEncoder;

public String addUser(UserInfo userInfo) {

String name = userInfo.getName();

UserInfo obj1 = repository.findByName(name).orElse(null);

System.out.println(obj1);

if (obj1 == null) {

userInfo.setPassword(passwordEncoder.encode(userInfo.getPassword()));

repository.save(userInfo);

return "Registration Successfully ";

} else {

return "This UserName is Already Registered.";

}

}

public String getRoles(String username) {

UserInfo obj2 = repository.findByName(username).orElse(null);

if (obj2 != null) {

return obj2.getRoles();

}

return "Not Found";

}

}

package com.cts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication

@EnableFeignClients

public class TicketBookingApplication {

public static void main(String[] args) {

SpringApplication.run(TicketBookingApplication.class, args);

}

}

package com.cts.controller;

import java.util.List;

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

import org.springframework.http.ResponseEntity;

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

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.RequestBody;

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

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

import com.cts.exceptions.TicketNotFoundException;

import com.cts.model.TicketBooking;

import com.cts.service.TicketBookingService;

@RestController

@RequestMapping("/tickets")

public class TicketBookingController {

@Autowired

private TicketBookingService ticketBookingService;

@PostMapping("/book")

public TicketBooking bookTicket(@RequestBody TicketBooking ticket ) {

return ticketBookingService.bookTicket(ticket);

}

@GetMapping("getticketbyid/{ticketId}")

public TicketBooking getTicketById(@PathVariable("ticketId") int ticketId) throws TicketNotFoundException {

return ticketBookingService.getTicketById(ticketId);

}

@GetMapping("/getalltickets")

public List<TicketBooking> getAllTickets() {

return ticketBookingService.getAllTickets();

}

@GetMapping("/getticketsbyuserid/{userid}")

public List<TicketBooking> getTicketsByUserId(@PathVariable("userid") int userId) {

return ticketBookingService.getTicketsByUserId(userId);

}

@GetMapping("/getticketsbyeventid/{eventid}")

public List<TicketBooking> getTicketsByEventId(@PathVariable("eventid") int userId) {

return ticketBookingService.getTicketsByEventId(userId);

}

@DeleteMapping("/cancelticket/{ticketid}")

public String cancelTicket(@PathVariable("ticketid") int ticketid) throws TicketNotFoundException {

return ticketBookingService.cancelTicket(ticketid);

//

// @PostMapping("/sendemail")

// public String sendEmail(@RequestBody EmailRequest request) {

// return ticketBookingService(request);

// }

//

}

}

package com.cts.dto;

import java.sql.Date;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class EventManagement {

private int eventid;

private String eventName;

private String eventCategory;

private String eventLocation;

private Date eventDate;

}

package com.cts.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class UserRegistration {

private int id;

private String userEmail;}

package com.cts.exceptions;

public class TicketNotFoundException extends Exception{

public TicketNotFoundException(int id) {

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

}

}

package com.cts.feignclient;

import org.springframework.cloud.openfeign.FeignClient;

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

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

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

import com.cts.dto.EventManagement;

@FeignClient(name="EVENTMANAGEMENT",path="/events")

public interface EventManagementClient {

@GetMapping("/fetcheventbyid/{eid}")

public EventManagement getEventById(@PathVariable("eid") int id);

@PostMapping("increaseTicketCount/{eid}")

public void increaseTicketCount(@PathVariable("eid") int id);

@PostMapping("decreaseTicketCount/{eid}")

public void decreaseTicketCount(@PathVariable("eid") int id);

}

package com.cts.feignclient;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import com.cts.dto.UserRegistration;

@FeignClient(name = "USERREGISTRATION", path = "/users")

public interface UserRegistrationClient {

@GetMapping("/{id}")

public UserRegistration getUserById(@PathVariable int id);

}

package com.cts.model;

import java.time.LocalDateTime;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Version;

import jakarta.validation.constraints.NotNull;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@Entity

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class TicketBooking {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int ticketId;

@NotNull(message = "Event ID cannot be null")

private int eventId;

@NotNull(message = "User ID cannot be null")

private int userId;

private LocalDateTime ticketBookingDate;

private Status ticketStatus;

public enum Status {

BOOKED, CANCELLED;

}

}

package com.cts.repository;

import java.util.List;

import java.util.Optional;

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

import com.cts.model.TicketBooking;

public interface TicketBookingRepository extends JpaRepository<TicketBooking,Integer> {

List<TicketBooking> findByUserId(int userId);

List<TicketBooking> findByEventId(int eventId);

// List<TicketBooking> findByUserId(int userId);

}

package com.cts.service;

import java.util.List;

import com.cts.exceptions.TicketNotFoundException;

import com.cts.model.TicketBooking;

public interface TicketBookingService {

public abstract TicketBooking bookTicket(TicketBooking ticket);

public abstract TicketBooking getTicketById(int ticketId) throws TicketNotFoundException;

public abstract List<TicketBooking> getAllTickets();

public abstract List<TicketBooking> getTicketsByUserId(int userId);

public abstract List<TicketBooking> getTicketsByEventId(int eventId);

public abstract String cancelTicket(int ticketId) throws TicketNotFoundException;

//public String sendEmailForBooking(String emailId,EmailRequest request);

}

package com.cts.service;

import java.time.LocalDateTime;

import java.util.List;

import java.util.Optional;

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

import org.springframework.mail.SimpleMailMessage;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.stereotype.Service;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.cts.dto.EventManagement;

import com.cts.dto.UserRegistration;

import com.cts.exceptions.TicketNotFoundException;

import com.cts.feignclient.EventManagementClient;

import com.cts.feignclient.UserRegistrationClient;

import com.cts.model.TicketBooking;

import com.cts.model.TicketBooking.Status;

import com.cts.repository.TicketBookingRepository;

import jakarta.transaction.Transactional;

@Service

@Transactional

public class TicketBookingServiceImpl implements TicketBookingService {

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

@Autowired

private TicketBookingRepository repository;

@Autowired

private EventManagementClient eventClient;

@Autowired

private UserRegistrationClient userClient;

@Autowired

private JavaMailSender javaMailSender;

@Override

public TicketBooking bookTicket(TicketBooking ticket) {

logger.info("Booking ticket for event ID: {} and user ID: {}", ticket.getEventId(), ticket.getUserId());

EventManagement eventDetails = eventClient.getEventById(ticket.getEventId());

UserRegistration userDetails = userClient.getUserById(ticket.getUserId());

eventClient.decreaseTicketCount(ticket.getEventId());

logger.info("Decreased ticket count for event ID: {}", ticket.getEventId());

ticket.setTicketBookingDate(LocalDateTime.now());

ticket.setTicketStatus(TicketBooking.Status.BOOKED);

UserRegistration user = userClient.getUserById(ticket.getUserId());

String emailId = user.getUserEmail();

String subject = "Ticket booking status";

String message = "Congratulations! Your ticket has been booked successfully."+ "\n" +"Ticket Details:\n" +

"User ID : " +ticket.getUserId() + "\n" +

"Ticket ID : " + ticket.getTicketId() + "\n" +

"Event ID : " + ticket.getEventId() + "\n" +

"Event Name : " + eventDetails.getEventName().toUpperCase() + "\n" +

"Event Locatoin : "+ eventDetails.getEventLocation().toUpperCase() + "\n" +

"Event Date : "+ eventDetails.getEventDate();

sendEmail(emailId, subject, message);

logger.info("Sent email to user ID: {}", ticket.getUserId());

// Save the ticket booking

TicketBooking savedTicket = repository.save(ticket);

logger.info("Ticket booked successfully for event ID: {} and user ID: {}", ticket.getEventId(), ticket.getUserId());

return savedTicket;

}

@Override

public TicketBooking getTicketById(int ticketId) throws TicketNotFoundException {

logger.info("Fetching ticket with ID: {}", ticketId);

Optional<TicketBooking> optional = repository.findById(ticketId);

if (optional.isPresent()) {

logger.info("Ticket found with ID: {}", ticketId);

return optional.get();

} else {

logger.error("Ticket not found with ID: {}", ticketId);

throw new TicketNotFoundException(ticketId);

}

}

@Override

public List<TicketBooking> getAllTickets() {

logger.info("Fetching all tickets");

return repository.findAll();

}

@Override

public List<TicketBooking> getTicketsByUserId(int userId) {

logger.info("Fetching tickets for user ID: {}", userId);

UserRegistration userDetails = userClient.getUserById(userId);

return repository.findByUserId(userId);

}

@Override

public List<TicketBooking> getTicketsByEventId(int eventId) {

logger.info("Fetching tickets for event ID: {}", eventId);

EventManagement eventDetails = eventClient.getEventById(eventId);

return repository.findByEventId(eventId);

}

@Override

public String cancelTicket(int ticketId) throws TicketNotFoundException {

logger.info("Cancelling ticket with ID: {}", ticketId);

TicketBooking ticket = getTicketById(ticketId);

ticket.setTicketStatus(Status.CANCELLED);

eventClient.increaseTicketCount(ticket.getEventId());

repository.save(ticket);

logger.info("Ticket cancelled with ID: {}", ticketId);

return "Ticket cancelled";

}

public String sendEmail(String to, String subject, String body) {

logger.info("Sending email to: {}", to);

SimpleMailMessage message = new SimpleMailMessage();

message.setTo(to);

message.setSubject(subject);

message.setText(body);

javaMailSender.send(message);

logger.info("Email sent to: {}", to);

return "final";

}

}

package com.cts;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class UserRegistrationApplication {

public static void main(String[] args) {

SpringApplication.run(UserRegistrationApplication.class, args);

}

}

package com.cts.controller;

import java.util.List;

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

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

import com.cts.exceptions.InvalidDataException;

import com.cts.exceptions.UserNotFoundException;

import com.cts.model.UserRegistration;

import com.cts.service.UserRegistrationService;

@RestController

@RequestMapping("/users")

public class UserRegistrationController {

@Autowired

private UserRegistrationService userService;

@PostMapping("/signup")

public UserRegistration signup(@RequestBody UserRegistration user) throws InvalidDataException {

return userService.saveUser(user);

}

@PostMapping("/login")

public String login(@RequestBody UserRegistration user) {

boolean isValidUser = userService.validateUser(user.getUserEmail(), user.getUserPassword());

return isValidUser ? "Login successful" : "Invalid credentials";

}

@GetMapping("/all")

public List<UserRegistration> getUsers() {

return userService.getAllUsers();

}

@GetMapping("/{id}")

public UserRegistration getUserById(@PathVariable int id) throws UserNotFoundException {

return userService.getUserById(id);

}

@PostMapping("/save")

public UserRegistration createUser(@RequestBody UserRegistration user) throws InvalidDataException {

return userService.saveUser(user);

}

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

public String updateUser(@PathVariable int id, @RequestBody UserRegistration updatedUser) throws UserNotFoundException {

return userService.updateUser(id, updatedUser);

}

@DeleteMapping("/delete/{id}")

public String deleteUserById(@PathVariable int id) throws UserNotFoundException {

userService.deleteUserById(id);

return "User deleted successfully";

}

}

package com.cts.exceptions;

public class InvalidDataException extends Exception {

public InvalidDataException(String msg) {

// **TODO** Auto-generated constructor stub

super(msg);

}

}

package com.cts.exceptions;

public class UserNotFoundException extends RuntimeException {

public UserNotFoundException(String string) {

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

}

}

package com.cts.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

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

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class UserRegistration {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@NotBlank(message = "User name cannot be blank")

@Size(min = 4, max =20 , message = "User name must be between 4 and 20 characters")

private String userName;

@NotBlank(message = "User email cannot be blank")

@Email(message = "User email should be valid")

private String userEmail;

@NotBlank(message = "User password cannot be blank")

@Size(min = 8, message = "User password must be at least 8 characters long")

private String userPassword;

@NotBlank(message = "User contact number cannot be blank")

@Pattern(regexp = "^\\+?[0-9. ()-]{7,25}$", message = "User contact number should be valid")

private String userContactNumber;

}

package com.cts.repository;

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

import com.cts.model.UserRegistration;

public interface UserRegistrationRepository extends JpaRepository<UserRegistration, Integer> {

UserRegistration findByUserEmail(String email);

}

package com.cts.service;

import java.util.\*;

import com.cts.exceptions.InvalidDataException;

import com.cts.exceptions.UserNotFoundException;

import com.cts.model.UserRegistration;

public interface UserRegistrationService {

public abstract List<UserRegistration> getAllUsers();

public abstract UserRegistration getUserById(int id) throws UserNotFoundException;

public abstract UserRegistration saveUser(UserRegistration user) throws InvalidDataException;

public abstract String updateUser(int id, UserRegistration updatedUser);

public abstract void deleteUserById(int id);

public abstract boolean validateUser(String email, String password);

}

package com.cts.service;

import java.util.List;

import java.util.Optional;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Service;

import com.cts.exceptions.InvalidDataException;

import com.cts.exceptions.UserNotFoundException;

import com.cts.model.UserRegistration;

import com.cts.repository.UserRegistrationRepository;

@Service

public class UserRegistrationServiceImpl implements UserRegistrationService {

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

@Autowired

private UserRegistrationRepository repository;

@Override

public List<UserRegistration> getAllUsers() {

logger.info("Fetching all users");

return repository.findAll();

}

@Override

public UserRegistration getUserById(int id) throws UserNotFoundException {

logger.info("Fetching user with ID: {}", id);

Optional<UserRegistration> optional = repository.findById(id);

if (optional.isPresent()) {

logger.info("User found with ID: {}", id);

return optional.get();

} else {

logger.error("User not found with ID: {}", id);

throw new UserNotFoundException("Invalid ID");

}

}

@Override

public UserRegistration saveUser(UserRegistration user) throws InvalidDataException {

logger.info("Saving user: {}", user);

UserRegistration u = repository.save(user);

if(u!=null) {

return u;

}

else {

throw new InvalidDataException("Invalid user data...");

}

}

@Override

public String updateUser(int id, UserRegistration updatedUser) throws UserNotFoundException {

logger.info("Updating user with ID: {}", id);

Optional<UserRegistration> optionalUser = repository.findById(id);

if (optionalUser.isPresent()) {

UserRegistration existingUser = optionalUser.get();

existingUser.setUserName(updatedUser.getUserName());

existingUser.setUserEmail(updatedUser.getUserEmail());

existingUser.setUserPassword(updatedUser.getUserPassword());

existingUser.setUserContactNumber(updatedUser.getUserContactNumber());

repository.save(existingUser);

logger.info("User updated successfully with ID: {}", id);

return "User updated successfully!";

} else {

logger.error("User not found with ID: {}", id);

throw new UserNotFoundException("User not found with id: " + id);

}

}

@Override

public void deleteUserById(int id) throws UserNotFoundException {

logger.info("Deleting user with ID: {}", id);

if (repository.existsById(id)) {

repository.deleteById(id);

logger.info("User deleted successfully with ID: {}", id);

} else {

logger.error("User not found with ID: {}", id);

throw new UserNotFoundException("User not found with id: " + id);

}

}

@Override

public boolean validateUser(String email, String password) {

logger.info("Validating user with email: {}", email);

UserRegistration user = repository.findByUserEmail(email);

if (user != null && user.getUserPassword().equals(password)) {

logger.info("User validated successfully with email: {}", email);

return true;

} else {

logger.error("User validation failed with email: {}", email);

return false;

}

}

}