package com.ratings.review;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.openfeign.EnableFeignClients;

@EnableFeignClients

@SpringBootApplication

@EnableDiscoveryClient // This enables Eureka client functionality

public class RatingsAndReviews {

public static void main(String[] args) {

SpringApplication.run(RatingsAndReviews.class, args);

}

}  
package com.ratings.review.client;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import com.ratings.review.dto.TravelPackageDTO;

@FeignClient(name = "travel-package-management")

public interface TravelPackageClient {

@GetMapping("/api/packages/{id}")

TravelPackageDTO getPackageById(@PathVariable("id") Long id);

}  
package com.ratings.review.controller;

import com.ratings.review.entity.AgentResponse;

import com.ratings.review.exception.ResourceNotFoundException;

import com.ratings.review.service.AgentResponseService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Map;

import java.time.LocalDateTime;

import java.util.HashMap;

/\*\*

\* Controller for handling agent responses to reviews.

\*/

@RestController

@RequestMapping("/api/agent-responses") //Base mapping

public class AgentResponseController {

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

@Autowired

private AgentResponseService agentResponseService;

/\*\*

\* Travel Agent Responds to a Review (Only if they own the package)

\*/

@PostMapping("/{agentId}/{reviewId}")

public ResponseEntity<Map<String, Object>> respondToReview(

@PathVariable Long agentId,

@PathVariable Long reviewId,

@RequestBody Map<String, String> requestBody) { // Accept JSON body as a map

String responseMessage = requestBody.get("responseMessage"); // Extract message from JSON

//Call the service method correctly

AgentResponse savedResponse = agentResponseService.respondToReview(agentId, reviewId, responseMessage);

//Extract packageId and review comment for response

Long packageId = savedResponse.getReview().getPackageId();

String reviewMessage = savedResponse.getReview().getComment();

//Create response with required fields

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

responseBody.put("packageId", packageId);

responseBody.put("reviewId", reviewId);

responseBody.put("reviewMessage", reviewMessage);

responseBody.put("responseMessage", savedResponse.getResponseMessage());

responseBody.put("responseTime", savedResponse.getResponseTime());

return ResponseEntity.ok(responseBody);

}

/\*\*

\* Get Responses for a Specific Review

\*/

@GetMapping("/{reviewId}")

public ResponseEntity<List<AgentResponse>> getResponsesForReview(@PathVariable Long reviewId) {

logger.info("Fetching responses for Review ID {}", reviewId);

List<AgentResponse> responses = agentResponseService.getResponsesForReview(reviewId);

return ResponseEntity.ok(responses);

}

}package com.ratings.review.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.ResponseEntity;

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

import java.time.LocalDateTime; // ✅ Used for timestamps

import java.util.HashMap; // ✅ Required for creating response maps

import java.util.Map; // ✅ Helps structure JSON response

import com.ratings.review.entity.Review;

import com.ratings.review.exception.ResourceNotFoundException;

import com.ratings.review.repository.ReviewRepository;

import com.ratings.review.service.ReviewService;

import java.util.List;

/\*\*

\* Controller for managing travel package reviews.

\*/

@RestController

@RequestMapping("/api/reviews")

public class ReviewController {

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

@Autowired

private ReviewService reviewService;

/\*\*

\* Post a new review

\*/

@PostMapping

public ResponseEntity<Map<String, Object>> postReview(@RequestBody Review review) {

logger.info("Posting a new review.");

review.setTimestamp(LocalDateTime.now());

Review savedReview = reviewService.postReview(review); // Save review through service layer

// Ensure reviewId is included in the response

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

response.put("reviewId", savedReview.getReviewId()); // Add reviewId here

response.put("userId", savedReview.getUserId());

response.put("comment", savedReview.getComment());

response.put("rating", savedReview.getRating());

response.put("timestamp", savedReview.getTimestamp());

return ResponseEntity.ok(response);

}

/\*\*

\* Retrieve all reviews associated with the specified travel package

\*/

@GetMapping("/{packageId}")

public List<Review> getReviewsByPackage(@PathVariable Long packageId) {

logger.info("Fetching reviews for package ID {}", packageId);

return reviewService.getReviewsByPackage(packageId);

}

/\*\*

\* average rating

\*/

@GetMapping("/{packageId}/average-rating")

public ResponseEntity<Double> getAverageRating(@PathVariable Long packageId) {

double avgRating = reviewService.getAverageRatingForPackage(packageId);

return ResponseEntity.ok(avgRating);

}

/\*\*

\* Update an existing review

\*/

@PutMapping("/{reviewId}")

public ResponseEntity<Review> updateReview(@PathVariable Long reviewId, @RequestBody Review updatedReview) {

logger.info("Updating Review ID {}", reviewId);

return ResponseEntity.ok(reviewService.updateReview(reviewId, updatedReview));

}

/\*\*

\* Delete a review

\*/

@DeleteMapping("/{reviewId}")

public ResponseEntity<String> deleteReview(@PathVariable Long reviewId) {

logger.info("Deleting Review ID {}", reviewId);

reviewService.deleteReview(reviewId);

return ResponseEntity.ok("Review deleted successfully.");

}

}**package** com.ratings.review.dto;

**import** lombok.Data;

@Data

**public** **class** TravelPackageDTO {

**private** Long packageId;

**private** Long agentId;

**private** String title;

}

package com.ratings.review.entity;

import jakarta.persistence.\*;

import java.time.LocalDateTime;

/\*\*

\* Entity representing an agent's response to a review.

\*/

@Entity

@Table(name = "agentResponses")

public class AgentResponse {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long responseId;

@ManyToOne

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

private Review review; // Links response to the review

@Column(name = "agent\_id", nullable = false)

private Long agentId;

// Ensures only package owner responds

@Column(nullable = false, length = 500)

private String responseMessage;

@Column(nullable = false)

private LocalDateTime responseTime;

// Getters and Setters

public Long getResponseId() { return responseId; }

public void setResponseId(Long responseId) { this.responseId = responseId; }

public Review getReview() { return review; }

public void setReview(Review review) { this.review = review; }

public Long getAgentId() { return agentId; }

public void setAgentId(Long agentId) { this.agentId = agentId; }

public String getResponseMessage() { return responseMessage; }

public void setResponseMessage(String responseMessage) {

if (responseMessage == null || responseMessage.isEmpty()) {

throw new IllegalArgumentException("Response message cannot be empty.");

}

this.responseMessage = responseMessage;

}

public LocalDateTime getResponseTime() { return responseTime; }

public void setResponseTime(LocalDateTime responseTime) { this.responseTime = responseTime; }

}  
package com.ratings.review.entity;

import jakarta.persistence.\*;

import java.time.LocalDateTime;

/\*\*

\* Entity representing a review for a travel package.

\*/

@Entity

@Table(name = "review")

public class Review {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long reviewId;

private Long userId;

@Column(name = "package\_id", nullable = false)

private Long packageId;

@Column(nullable = false)

private int rating;

@Column(nullable = false, length = 1000)

private String comment;

@Column(nullable = false)

private LocalDateTime timestamp;

// Getters and Setters

public Long getReviewId() { return reviewId; }

public void setReviewId(Long reviewId) { this.reviewId = reviewId; }

public Long getUserId() { return userId; }

public void setUserId(Long userId) { this.userId = userId; }

public Long getPackageId() { return packageId; }

public void setPackageId(Long packageId) { this.packageId = packageId; }

public int getRating() { return rating; }

public void setRating(int rating) { this.rating = rating; }

public String getComment() { return comment; }

public void setComment(String comment) { this.comment = comment; }

public LocalDateTime getTimestamp() { return timestamp; }

public void setTimestamp(LocalDateTime timestamp) { this.timestamp = timestamp; }

}package com.ratings.review.exception;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

import java.time.LocalDateTime;

import java.util.HashMap;

import java.util.Map;

/\*\*

\* Centralized exception handling across the application.

\*/

@RestControllerAdvice

public class GlobalExceptionHandler {

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

/\*\*

\* Handles `ResourceNotFoundException` when an entity is not found.

\*/

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<Map<String, Object>> handleResourceNotFound(ResourceNotFoundException ex) {

logger.error("Resource Not Found: {}", ex.getMessage());

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

errorResponse.put("timestamp", LocalDateTime.now());

errorResponse.put("status", HttpStatus.NOT\_FOUND.value());

errorResponse.put("error", "Resource Not Found");

errorResponse.put("message", ex.getMessage());

return new ResponseEntity<>(errorResponse, HttpStatus.NOT\_FOUND);

}

/\*\*

\* Handles generic exceptions.

\*/

@ExceptionHandler(Exception.class)

public ResponseEntity<Map<String, Object>> handleGenericException(Exception ex) {

logger.error("Unhandled Exception: {}", ex.getMessage());

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

errorResponse.put("timestamp", LocalDateTime.now());

errorResponse.put("status", HttpStatus.INTERNAL\_SERVER\_ERROR.value());

errorResponse.put("error", "Internal Server Error");

errorResponse.put("message", ex.getMessage());

return new ResponseEntity<>(errorResponse, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}**package** com.ratings.review.exception;

/\*\*

\* Custom exception for handling "Resource Not Found" errors.

\* Used when an entity such as Review, TravelPackage, or TravelAgent does not exist.

\*/

**public** **class** ResourceNotFoundException **extends** RuntimeException {

**public** ResourceNotFoundException(String message) {

**super**(message);

}

}

package com.ratings.review.repository;

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

import org.springframework.stereotype.Repository;

import com.ratings.review.entity.AgentResponse;

import java.util.List;

/\*\*

\* Repository for managing agent responses to reviews.

\*/

@Repository

public interface AgentResponseRepository extends JpaRepository<AgentResponse, Long> {

/\*\*

\* Find all responses related to a specific review.

\*/

List<AgentResponse> findByReview\_ReviewId(Long reviewId);

}package com.ratings.review.repository;

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

import org.springframework.stereotype.Repository;

import java.util.List;

import com.ratings.review.entity.Review;

/\*\*

\* Repository for managing travel package reviews.

\*/

@Repository

public interface ReviewRepository extends JpaRepository<Review, Long> {

List<Review> findByPackageId(Long packageId);

}package com.ratings.review.service;

import com.ratings.review.client.TravelPackageClient;

import com.ratings.review.dto.TravelPackageDTO;

import com.ratings.review.entity.AgentResponse;

import com.ratings.review.entity.Review;

import com.ratings.review.exception.ResourceNotFoundException;

import com.ratings.review.repository.AgentResponseRepository;

import com.ratings.review.repository.ReviewRepository;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Service;

import java.time.LocalDateTime;

import java.util.List;

@Service

public class AgentResponseService {

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

@Autowired

private AgentResponseRepository agentResponseRepository;

@Autowired

private ReviewRepository reviewRepository;

@Autowired

private TravelPackageClient travelPackageClient; // Feign client to communicate with Travel Package module

/\*\*

\* Allows a travel agent to respond to a review.

\* Only the agent who owns the package can respond.

\*/

public AgentResponse respondToReview(Long agentId, Long reviewId, String responseMessage) {

logger.info("Attempting to respond to review ID: {}", reviewId);

Review review = reviewRepository.findById(reviewId)

.orElseThrow(() -> {

logger.error("Review ID {} not found.", reviewId);

return new ResourceNotFoundException("Review not found.");

});

Long packageId = review.getPackageId();

// Fetch package details from Travel Package Management module

TravelPackageDTO travelPackage = travelPackageClient.getPackageById(packageId);

// Authorization check: Only package owner can respond

if (!travelPackage.getAgentId().equals(agentId)) {

logger.warn("Unauthorized response attempt by Agent ID {}", agentId);

throw new ResourceNotFoundException("Unauthorized: Only the package owner can respond.");

}

AgentResponse response = new AgentResponse();

response.setReview(review);

response.setAgentId(agentId);

response.setResponseMessage(responseMessage);

response.setResponseTime(LocalDateTime.now());

logger.info("Agent ID {} successfully responded to review ID {}", agentId, reviewId);

return agentResponseRepository.save(response);

}

/\*\*

\* Retrieves responses for a specific review.

\*/

public List<AgentResponse> getResponsesForReview(Long reviewId) {

logger.info("Fetching responses for review ID: {}", reviewId);

return agentResponseRepository.findByReview\_ReviewId(reviewId);

}

}package com.ratings.review.service;

import com.ratings.review.entity.Review;

import com.ratings.review.exception.ResourceNotFoundException;

import com.ratings.review.repository.ReviewRepository;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.time.LocalDateTime;

@Service

public class ReviewService {

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

@Autowired

private ReviewRepository reviewRepository;

/\*\*

\* Posts a new review.

\*/

public Review postReview(Review review) {

logger.info("Posting a new review.");

review.setTimestamp(LocalDateTime.now()); // Sets timestamp before saving

return reviewRepository.save(review);

}

/\*\*

\* Updates an existing review.

\*/

public Review updateReview(Long reviewId, Review updatedReview) {

logger.info("Updating review ID {}", reviewId);

Review review = reviewRepository.findById(reviewId)

.orElseThrow(() -> {

logger.error("Review ID {} not found.", reviewId);

return new ResourceNotFoundException("Review not found.");

});

review.setRating(updatedReview.getRating());

review.setComment(updatedReview.getComment());

return reviewRepository.save(review);

}

/\*\*

\* Deletes a review.

\*/

public void deleteReview(Long reviewId) {

logger.info("Deleting review ID {}", reviewId);

reviewRepository.findById(reviewId)

.orElseThrow(() -> {

logger.error("Review ID {} not found.", reviewId);

return new ResourceNotFoundException("Review not found.");

});

reviewRepository.deleteById(reviewId);

}

/\*\*

\* Retrieves all reviews for a specific travel package.

\*/

public List<Review> getReviewsByPackage(Long packageId) {

logger.info("Fetching reviews for package ID {}", packageId);

return reviewRepository.findAll().stream()

.filter(review -> review.getPackageId().equals(packageId))

.toList();

}

/\*\*

\* finding average rating for a package

\*/

public double getAverageRatingForPackage(Long packageId) {

List<Review> reviews = getReviewsByPackage(packageId);

return reviews.stream()

.mapToInt(Review::getRating)

.average()

.orElse(0.0);

}

}spring.application.name=RatingsAndReview

spring.datasource.url=jdbc:mysql://localhost:3306/ reviewsdatabase

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

server.port=8084

# Eureka client configuration

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

eureka.instance.prefer-ip-address=true