Booking and Payment module

package com.booking;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication

@EnableDiscoveryClient

@EnableFeignClients(basePackages = "com.booking.client")

public class BookingApplication {

public static void main(String[] args) {

SpringApplication.run(BookingApplication.class, args);

}

}

package com.booking.client;

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.PutMapping;

import java.util.List;

@FeignClient(name = "TravelInsuranceService", url = "http://localhost:8085")

public interface TravelInsuranceClient {

@GetMapping("/api/insurance/price/user/{userId}")

double getInsurancePriceByUserId(@PathVariable("userId") Long userId);

@GetMapping("/api/insurance/price/insurance/{insuranceId}")

double getInsurancePriceByInsuranceId(@PathVariable("insuranceId") Integer insuranceId);

@GetMapping("/api/insurance/validate/{insuranceId}")

boolean validateInsurance(@PathVariable("insuranceId") Integer insuranceId);

@PutMapping("/api/insurance/{insuranceId}/booking/{bookingId}")

String updateInsuranceBookingId(@PathVariable("insuranceId") Integer insuranceId, @PathVariable("bookingId") Long bookingId);

}

package com.booking.client;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import com.booking.dto.TravelPackageDTO;

import java.util.List;

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

public interface TravelPackageClient {

@GetMapping("/api/packages/internal/all")

List<TravelPackageDTO> getAllPackages();

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

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

}

package com.booking.client;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import com.booking.dto.UserDTO;

@FeignClient(name = "usermanagement-service")

public interface UserClient {

@GetMapping("/api/users/internal/customer/{id}")

UserDTO getCustomerById(@PathVariable("id") Long userId);

//UserDTO getCustomerById(Long userId);

}

package com.booking.controller;

import com.booking.dto.BookingDTO;

import com.booking.entity.Booking;

import com.booking.service.BookingService;

import com.booking.response.ApiResponse;

import com.booking.dto.TravelPackageDTO;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.logging.Logger;

@CrossOrigin(origins = "http://localhost:4200")

@RestController

@RequestMapping("/api/bookings")

public class BookingController {

private final BookingService service;

private static final Logger logger = Logger.getLogger(BookingController.class.getName());

@Autowired

public BookingController(BookingService service) {

this.service = service;

}

/\*\*

\* Create a Booking

\*/

@PostMapping

public ResponseEntity<BookingDTO> createBooking(@RequestBody Booking booking) {

try {

BookingDTO bookingDTO = service.createBooking(booking);

return new ResponseEntity<>(bookingDTO, HttpStatus.CREATED);

} catch (RuntimeException ex) {

logger.severe("Error creating booking: " + ex.getMessage());

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

}

}

/\*\*

\* To get all Bookings

\*/

@GetMapping

public ResponseEntity<List<Booking>> getAllBookings() {

List<Booking> bookings = service.getAllBookings();

return ResponseEntity.ok(bookings);

}

/\*\*

\* To get booking by Booking ID

\*/

@GetMapping("/{id}")

public ResponseEntity<Booking> getBookingById(@PathVariable Long id) {

Booking booking = service.getBookingById(id);

if (booking == null) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.ok(booking);

}

/\*\*

\* Cancel booking by Booking ID

\*/

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

public ResponseEntity<String> cancelBooking(@PathVariable Long id) {

return service.cancelBooking(id);

}

/\*\*

\* Delete booking by Booking ID

\*/

@DeleteMapping("/{id}")

public void delete(@PathVariable Long id) {

service.deleteBooking(id);

}

/\*\*

\* User Management Module to get bookings by User ID

\*/

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

public ResponseEntity<List<Booking>> getBookingsByUserId(@PathVariable Long userId) {

List<Booking> bookings = service.getBookingsByUserId(userId);

/\*if (bookings.isEmpty()) {

return ResponseEntity.notFound().build();

}\*/

return ResponseEntity.ok(bookings);

}

/\*\*

\* Reviews and Rating Module to validate booking

\*/

@GetMapping("/user/{userId}/package/{packageId}/completed")

public boolean hasUserCompletedPackage(@PathVariable Long userId, @PathVariable String packageId) {

return service.hasUserCompletedPackage(userId, packageId);

}

/\*\*

\* Travel Insurance module to get Booking ID

\*/

@GetMapping("/internal/{id}")

public ResponseEntity<Booking> getInternalBookingById(@PathVariable Long id) {

Booking booking = service.getBookingById(id);

if (booking == null) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.ok(booking);

}

/\*\*

\* To get All packages from Travel Package Management Module

\*/

@GetMapping("/packages")

public ResponseEntity<ApiResponse> getAllPackages() {

List<TravelPackageDTO> packages = service.getAllPackages();

return ResponseEntity.ok(new ApiResponse(true, "All packages retrieved ", packages));

}

/\*\*

\* To get package by ID from Travel Package Management Module

\*/

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

public ResponseEntity<ApiResponse> getPackageById(@PathVariable("id") Long packageId) {

TravelPackageDTO packages = service.getPackageById(packageId);

return ResponseEntity.ok(new ApiResponse(true, "Package retrieved successfully ", packages));

}

}

**package** com.booking.controller;

**import** com.booking.dto.PaymentResponseDTO;

**import** com.booking.entity.Payment;

**import** com.booking.service.PaymentService;

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

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

**import** java.util.Collections;

**import** java.util.List;

**import** java.util.Map;

@RestController

@RequestMapping("/api/payments")

**public** **class** PaymentController {

@Autowired **private** PaymentService paymentService;

// Process payment with optional coupon code

@PostMapping

**public** PaymentResponseDTO process(

@RequestBody Payment payment,

@RequestParam(required = **false**) String couponCode) {

Payment savedPayment = paymentService.processPayment(payment, couponCode);

PaymentResponseDTO response = **new** PaymentResponseDTO();

response.setPaymentId(savedPayment.getPaymentId());

response.setUserId(savedPayment.getUserId());

response.setBookingId(savedPayment.getBookingId());

response.setAmount(savedPayment.getAmount());

response.setStatus(savedPayment.getStatus());

response.setPaymentMethod(savedPayment.getPaymentMethod());

response.setCurrency(savedPayment.getCurrency());

**return** response;

}

// Show total payable before actual payment

@GetMapping("/expected-total")

**public** Map<String, Double> getExpectedTotal(

@RequestParam Long bookingId,

@RequestParam(required = **false**) String couponCode) {

**double** total = paymentService.calculateExpectedTotal(bookingId, couponCode);

**return** Collections.*singletonMap*("totalPayable", total);

}

@GetMapping **public** List<Payment> getAll() { **return** paymentService.getAllPayments(); }

@GetMapping("/{id}") **public** Payment getById(@PathVariable Long id) { **return** paymentService.getPaymentById(id); }

}

package com.booking.dto;

import lombok.Data;

import java.time.LocalDate;

@Data

public class BookingDTO {

private Long bookingId;

private Long userId;

private Long packageId;

private LocalDate tripStartDate;

private LocalDate tripEndDate;

private String status;

private Long paymentId;

private Integer insuranceId;

}

package com.booking.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

/\*\*

\* DTO to expose insurance coverage plans to Booking module.

\*/

@Data

@AllArgsConstructor

public class CoveragePlanDTO {

private String coverageType; // BASIC, STANDARD, PREMIUM

private String coverageDetails; // Human-readable description

private Double price; // Fixed price per plan

private Double claimableAmount; // Max claim amount per plan

}

**package** com.booking.dto;

**import** lombok.Data;

@Data

**public** **class** FlightDTO {

**private** String airline;

**private** String fromCity;

**private** String toCity;

**private** String departureTime;

**private** String arrivalTime;

}

**package** com.booking.dto;

**import** lombok.Data;

@Data

**public** **class** HotelDTO {

**private** String name;

**private** String city;

**private** **double** rating;

**private** **int** nights;

**private** **double** costPerNight;

}

**package** com.booking.dto;

**import** lombok.Data;

/\*\*

\* DTO to expose insurance coverage plans to Booking module.

\*/

@Data

**public** **class** InsuranceRequestDTO {

**private** Integer insuranceId;

**private** Long userId;

**private** Long bookingId; // To be updated after successful payment

**private** String coverageDetails;

**private** String coverageType;

**private** **double** price;

**private** String provider;

**private** String status; // e.g., "PENDING", "ACTIVE", "CANCELLED"

}

**package** com.booking.dto;

**import** lombok.Data;

@Data

**public** **class** ItineraryDTO {

**private** **int** dayNumber;

**private** String activityTitle;

**private** String activityDescription;

}

**package** com.booking.dto;

**import** lombok.Data;

@Data

**public** **class** OfferDTO {

**private** String couponCode;

**private** String description;

**private** **int** discountPercentage;

**private** **boolean** active;

}

package com.booking.dto;

import lombok.Data;

import java.time.LocalDate;

@Data

public class PackageDTO {

private Long packageId;

private Long agentId;

private String title;

private String description;

private int duration;

private double price;

private int maxCapacity;

private LocalDate tripStartDate;

private LocalDate tripEndDate;

}

**package** com.booking.dto;

**public** **class** PaymentResponseDTO {

**private** Long paymentId;

**private** Long userId;

**private** Long bookingId;

**private** **double** amount;

**private** String status;

**private** String paymentMethod;

**private** String currency;

// Getters and Setters

**public** Long getPaymentId() {

**return** paymentId;

}

**public** **void** setPaymentId(Long paymentId) {

**this**.paymentId = paymentId;

}

**public** Long getUserId() {

**return** userId;

}

**public** **void** setUserId(Long userId) {

**this**.userId = userId;

}

**public** Long getBookingId() {

**return** bookingId;

}

**public** **void** setBookingId(Long bookingId) {

**this**.bookingId = bookingId;

}

**public** **double** getAmount() {

**return** amount;

}

**public** **void** setAmount(**double** amount) {

**this**.amount = amount;

}

**public** String getStatus() {

**return** status;

}

**public** **void** setStatus(String status) {

**this**.status = status;

}

**public** String getPaymentMethod() {

**return** paymentMethod;

}

**public** **void** setPaymentMethod(String paymentMethod) {

**this**.paymentMethod = paymentMethod;

}

**public** String getCurrency() {

**return** currency;

}

**public** **void** setCurrency(String currency) {

**this**.currency = currency;

}

}

**package** com.booking.dto;

**import** lombok.Data;

@Data

**public** **class** SightseeingDTO {

**private** String location;

**private** String description;

}

package com.booking.dto;

import java.time.LocalDate;

import java.util.List;

import lombok.Data;

@Data

public class TravelPackageDTO {

private Long packageId;

private Long agentId;

private String title;

private String description;

private int duration;

private double price;

private int maxCapacity;

private LocalDate tripStartDate;

private LocalDate tripEndDate;

private List<String> highlights;

private List<FlightDTO> flights;

private List<HotelDTO> hotels;

private List<SightseeingDTO> sightseeing;

private List<ItineraryDTO> itinerary;

private String imageUrl; //for images in front-end

private OfferDTO offer;

//newly added fields

private String country;

private String destination;

private String tripType;

}

**package** com.booking.dto;

**import** lombok.Data;

@Data

**public** **class** UserDTO {

**private** Long id;

**private** String name;

**private** String email;

**private** String role;

}

package com.booking.entity;

import java.time.LocalDate;

import jakarta.persistence.\*;

@Entity

public class Booking {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long bookingId;

private Long userId;

private Long packageId;

@Column(name= "trip\_start\_date")

private LocalDate tripStartDate;

@Column(name= "trip\_end\_date")

private LocalDate tripEndDate;

private String status;

private Long paymentId;

private Integer insuranceId;

// Getters and Setters

public Long getBookingId() {

return bookingId;

}

public void setBookingId(Long bookingId) {

this.bookingId = bookingId;

}

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 LocalDate getTripStartDate() {

return tripStartDate;

}

public void setTripStartDate(LocalDate tripStartDate) {

this.tripStartDate = tripStartDate;

}

public LocalDate getTripEndDate() {

return tripEndDate;

}

public void setTripEndDate(LocalDate tripEndDate) {

this.tripEndDate = tripEndDate;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public Long getPaymentId() {

return paymentId;

}

public void setPaymentId(Long paymentId) {

this.paymentId = paymentId;

}

public Integer getInsuranceId() {

return insuranceId;

}

public void setInsuranceId(Integer insuranceId) {

this.insuranceId = insuranceId;

}

}

**package** com.booking.entity;

**import** jakarta.persistence.\*;

@Entity

**public** **class** Payment {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long paymentId;

**private** Long userId;

**private** Long bookingId;

**private** **double** amount;

**private** String status;

@Column(nullable = **false**)

**private** String paymentMethod; // Should be "Credit Card" or "Debit Card"

**private** String currency;

@Transient

**private** String cardNumber; // Not stored in DB

@Transient

**private** String cvv; // Not stored in DB

@Transient

**private** String atmPin; // Not stored in DB

@Transient

**private** String expiryDate; // Not stored in DB

// Getters and Setters

**public** Long getUserId() {

**return** userId;

}

**public** **void** setUserId(Long userId) {

**this**.userId = userId;

}

**public** Long getBookingId() {

**return** bookingId;

}

**public** **void** setBookingId(Long bookingId) {

**this**.bookingId = bookingId;

}

**public** **double** getAmount() {

**return** amount;

}

**public** **void** setAmount(**double** amount) {

**this**.amount = amount;

}

**public** String getStatus() {

**return** status;

}

**public** **void** setStatus(String status) {

**this**.status = status;

}

**public** String getPaymentMethod() {

**return** paymentMethod;

}

**public** **void** setPaymentMethod(String paymentMethod) {

**this**.paymentMethod = paymentMethod;

}

**public** String getCurrency() {

**return** currency;

}

**public** **void** setCurrency(String currency) {

**this**.currency = currency;

}

**public** String getCardNumber() {

**return** cardNumber;

}

**public** **void** setCardNumber(String cardNumber) {

**this**.cardNumber = cardNumber;

}

**public** String getCvv() {

**return** cvv;

}

**public** **void** setCvv(String cvv) {

**this**.cvv = cvv;

}

**public** String getAtmPin() {

**return** atmPin;

}

**public** **void** setAtmPin(String atmPin) {

**this**.atmPin = atmPin;

}

**public** Long getPaymentId() {

**return** paymentId;

}

**public** **void** setPaymentId(Long paymentId) {

**this**.paymentId = paymentId;

}

**public** String getExpiryDate() {

**return** expiryDate;

}

**public** **void** setExpiryDate(String expiryDate) {

**this**.expiryDate = expiryDate;

}

}

package com.booking.repository;

import com.booking.entity.Booking;

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

import java.util.List;

public interface BookingRepository extends JpaRepository<Booking, Long> {

List<Booking> findByUserId(Long userId);

int countByUserId(Long userId);

List<Booking> findByPackageId(Long packageId);

}

package com.booking.repository;

import com.booking.entity.Payment;

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

public interface PaymentRepository extends JpaRepository<Payment, Long> {

}

package com.booking.response;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class ApiResponse {

private boolean success;

private String message;

private Object data;

}

package com.booking.service;

import com.booking.client.TravelPackageClient;

import com.booking.client.TravelInsuranceClient;

import com.booking.client.UserClient;

import com.booking.dto.BookingDTO;

import com.booking.dto.TravelPackageDTO;

import com.booking.dto.UserDTO;

import com.booking.entity.Booking;

import com.booking.repository.BookingRepository;

//import com.booking.response.ApiResponse;

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

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Service;

import java.time.LocalDate;

import java.util.List;

import java.util.logging.Logger;

@Service

public class BookingService {

@Autowired

private BookingRepository bookingRepo;

@Autowired

private TravelPackageClient travelPackageClient;

@Autowired

private TravelInsuranceClient travelInsuranceClient;

@Autowired

private UserClient userClient;

private static final Logger logger = Logger.getLogger(BookingService.class.getName());

public BookingDTO createBooking(Booking bookingRequest) {

Long userId = bookingRequest.getUserId();

Long packageId = bookingRequest.getPackageId();

Integer insuranceId = bookingRequest.getInsuranceId();

logger.info("Creating booking for userId: " + userId + " and packageId: " + packageId);

// Validate User

UserDTO user = userClient.getCustomerById(userId);

if (user == null || !"CUSTOMER".equalsIgnoreCase(user.getRole())) {

throw new RuntimeException("User is not a valid CUSTOMER.");

}

// Validate Package

TravelPackageDTO travelPackage = travelPackageClient.getPackageById(packageId);

if (travelPackage == null) {

throw new IllegalArgumentException("Invalid travel package ID.");

}

//Optional: Validate Insurance if selected

if (insuranceId != null && insuranceId > 0) {

boolean exists = travelInsuranceClient.validateInsurance(insuranceId);

if (!exists) {

throw new IllegalArgumentException("Selected Insurance ID is invalid.");

}

}

// Create and save booking

Booking booking = new Booking();

booking.setUserId(userId);

booking.setPackageId(packageId);

booking.setInsuranceId(insuranceId);

booking.setTripStartDate(travelPackage.getTripStartDate());

booking.setTripEndDate(travelPackage.getTripEndDate());

booking.setStatus("CONFIRMED");

Booking savedBooking = bookingRepo.save(booking);

// Build response DTO

BookingDTO dto = new BookingDTO();

dto.setBookingId(savedBooking.getBookingId());

dto.setUserId(userId);

dto.setPackageId(packageId);

dto.setInsuranceId(insuranceId);

dto.setTripStartDate(savedBooking.getTripStartDate());

dto.setTripEndDate(savedBooking.getTripEndDate());

dto.setStatus(savedBooking.getStatus());

dto.setPaymentId(savedBooking.getPaymentId());

logger.info("Booking created successfully with bookingId: " + savedBooking.getBookingId());

return dto;

}

public List<Booking> getAllBookings() {

return bookingRepo.findAll();

}

public Booking getBookingById(Long id) {

return bookingRepo.findById(id).orElse(null);

}

public Booking getInternalBookingById(Long id) {

return bookingRepo.findById(id).orElse(null);

}

public void deleteBooking(Long id) {

bookingRepo.deleteById(id);

}

public ResponseEntity<String> cancelBooking(Long bookingId) {

Booking booking = bookingRepo.findById(bookingId).orElse(null);

if (booking == null) {

return ResponseEntity.badRequest().body("Booking not found.");

}

LocalDate today = LocalDate.now();

if (booking.getTripStartDate().minusDays(7).isBefore(today)) {

return ResponseEntity.badRequest().body("Cancellation not allowed. Must cancel at least 7 days before departure.");

}

booking.setStatus("CANCELLED");

bookingRepo.save(booking);

return ResponseEntity.ok("Booking cancelled successfully.");

}

// User module to return Bookings by UserId

public List<Booking> getBookingsByUserId(Long userId) {

return bookingRepo.findByUserId(userId);

}

// Rating and reviews module to validate booking

public boolean hasUserCompletedPackage(Long userId, String packageId) {

List<Booking> bookings = bookingRepo.findByUserId(userId);

LocalDate today = LocalDate.now();

return bookings.stream()

.anyMatch(b ->

String.valueOf(b.getPackageId()).equals(packageId) &&

"CONFIRMED".equalsIgnoreCase(b.getStatus()) &&

!b.getTripEndDate().isAfter(today) // includes today

);

}

// Get all packages

public List<TravelPackageDTO> getAllPackages() {

return travelPackageClient.getAllPackages();

}

// Get Package by ID

public TravelPackageDTO getPackageById(Long packageId) {

return travelPackageClient.getPackageById(packageId);

}

}

package com.booking.service;

import com.booking.entity.Booking;

import com.booking.entity.Payment;

import com.booking.dto.UserDTO;

import com.booking.client.TravelPackageClient;

import com.booking.client.UserClient;

import com.booking.dto.TravelPackageDTO;

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;

@Autowired

private UserClient userClient;

@Autowired

private TravelPackageClient travelPackageClient;

private final String senderEmail = "indhaanman@gmail.com";

// Booking details are sent to customers through Email

public void notifyCustomer(Booking booking, Payment payment) {

UserDTO user = userClient.getCustomerById(booking.getUserId());

String customerEmail = user.getEmail();

String subject = "Your Travel Booking is Confirmed – Booking ID: " + booking.getBookingId();

String body = "Dear Customer,\n\n"

+ "Thank you for booking your travel with us!\n\n"

+ "Your booking has been successfully confirmed. Here are the details:\n\n"

+ "- Booking ID: " + booking.getBookingId() + "\n"

+ "- Package ID: " + booking.getPackageId() + "\n"

+ "- Travel Dates: " + booking.getTripStartDate() + " to " + booking.getTripEndDate() + "\n"

+ "- Payment Amount: " + payment.getAmount() + " " + payment.getCurrency() + "\n"

+ "- Payment Status: " + payment.getStatus() + "\n\n"

+ "We look forward to providing you with a wonderful travel experience.\n\n"

+ "Warm regards,\nTravel Booking Team";

sendEmail(customerEmail, subject, body);

System.out.println("Email is sent to " + customerEmail);

}

// Booking details are sent to travel agent through Email

public void notifyTravelAgent(Booking booking, Payment payment) {

TravelPackageDTO pkg = travelPackageClient.getPackageById(booking.getPackageId());

String agentEmail = userClient.getCustomerById(pkg.getAgentId()).getEmail();

String subject = "New Booking Received – Booking ID: " + booking.getBookingId();

String body = "Dear Travel Agent,\n\n"

+ "A new booking has been successfully made. Please find the details below:\n\n"

+ "- Booking ID: " + booking.getBookingId() + "\n"

+ "- Customer ID: " + booking.getUserId() + "\n"

+ "- Package ID: " + booking.getPackageId() + "\n"

+ "- Travel Dates: " + booking.getTripStartDate() + " to " + booking.getTripEndDate() + "\n"

+ "- Payment Amount: " + payment.getAmount() + " " + payment.getCurrency() + "\n"

+ "- Payment Status: " + payment.getStatus() + "\n\n"

+ "Please ensure all arrangements are in place for the customer's travel.\n\n"

+ "Best regards,\nTravel Booking System";

sendEmail(agentEmail, subject, body);

System.out.println("Email is sent to " + agentEmail);

}

private void sendEmail(String to, String subject, String body) {

SimpleMailMessage mail = new SimpleMailMessage();

mail.setFrom(senderEmail);

mail.setTo(to);

mail.setSubject(subject);

mail.setText(body);

mailSender.send(mail);

}

}

package com.booking.service;

import com.booking.client.TravelPackageClient;

import com.booking.client.TravelInsuranceClient;

import com.booking.dto.OfferDTO;

import com.booking.dto.TravelPackageDTO;

import com.booking.entity.Booking;

import com.booking.entity.Payment;

import com.booking.repository.BookingRepository;

import com.booking.repository.PaymentRepository;

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

import org.springframework.stereotype.Service;

import java.time.LocalDate;

import java.util.List;

@Service

public class PaymentService {

@Autowired

private PaymentRepository paymentRepo;

@Autowired

private BookingRepository bookingRepo;

@Autowired

private NotificationService notificationService;

@Autowired

private TravelPackageClient travelPackageClient;

@Autowired

private TravelInsuranceClient travelInsuranceClient;

public Payment processPayment(Payment payment, String couponCodeApplied) {

Long bookingId = payment.getBookingId();

Booking booking = bookingRepo.findById(bookingId).orElse(null);

if (booking == null) {

throw new IllegalArgumentException("Booking with ID " + bookingId + " not found.");

}

TravelPackageDTO pkg = travelPackageClient.getPackageById(booking.getPackageId());

if (pkg == null) {

throw new IllegalArgumentException("Invalid package ID associated with booking.");

}

double packagePrice = pkg.getPrice();

//System.out.println("Booking insuranceId: " + booking.getInsuranceId());

double insurancePrice = 0.0;

if (booking.getInsuranceId() != null && booking.getInsuranceId() > 0) {

try {

insurancePrice = travelInsuranceClient.getInsurancePriceByInsuranceId(booking.getInsuranceId().intValue());

} catch (Exception e) {

throw new IllegalArgumentException("Failed to fetch insurance price. Verify Insurance Service is reachable.");

}

}

double discountAmount = 0.0;

if (couponCodeApplied != null && pkg.getOffer() != null) {

OfferDTO offer = pkg.getOffer();

if (couponCodeApplied.equalsIgnoreCase(offer.getCouponCode()) && offer.isActive()) {

discountAmount = (packagePrice \* offer.getDiscountPercentage()) / 100.0;

System.out.println("Coupon applied. Discount amount: " + discountAmount);

}

}

double expectedTotal = packagePrice + insurancePrice - discountAmount;

if (Math.abs(payment.getAmount() - expectedTotal) > 0.01) {

throw new IllegalArgumentException("Payment amount mismatch. Expected: " + expectedTotal);

}

if (!payment.getPaymentMethod().equalsIgnoreCase("Credit Card") &&

!payment.getPaymentMethod().equalsIgnoreCase("Debit Card")) {

throw new IllegalArgumentException("Only Credit Card or Debit Card accepted.");

}

if (payment.getCardNumber() == null || !payment.getCardNumber().matches("\\d{16}")) {

throw new IllegalArgumentException("Invalid card number.");

}

if (payment.getCvv() == null || !payment.getCvv().matches("\\d{3}")) {

throw new IllegalArgumentException("Invalid CVV.");

}

if (payment.getAtmPin() == null || !payment.getAtmPin().matches("\\d{4}")) {

throw new IllegalArgumentException("Invalid ATM PIN.");

}

if (payment.getExpiryDate() == null || !payment.getExpiryDate().matches("\\d{2}/\\d{2}")) {

throw new IllegalArgumentException("Expiry date format invalid.");

}

String[] parts = payment.getExpiryDate().split("/");

int expMonth = Integer.parseInt(parts[0]);

int expYear = 2000 + Integer.parseInt(parts[1]);

int currentMonth = LocalDate.now().getMonthValue();

int currentYear = LocalDate.now().getYear();

if (expMonth < 1 || expMonth > 12 || expYear < currentYear ||

(expYear == currentYear && expMonth < currentMonth)) {

throw new IllegalArgumentException("Card has expired.");

}

payment.setStatus("PAID");

Payment savedPayment = paymentRepo.save(payment);

booking.setPaymentId(savedPayment.getPaymentId());

bookingRepo.save(booking);

if (insurancePrice > 0.0) {

travelInsuranceClient.updateInsuranceBookingId(booking.getInsuranceId().intValue(), booking.getBookingId());

}

notificationService.notifyCustomer(booking, savedPayment);

notificationService.notifyTravelAgent(booking, savedPayment);

return savedPayment;

}

public double calculateExpectedTotal(Long bookingId, String couponCodeApplied) {

Booking booking = bookingRepo.findById(bookingId).orElse(null);

if (booking == null) throw new IllegalArgumentException("Booking not found");

TravelPackageDTO pkg = travelPackageClient.getPackageById(booking.getPackageId());

if (pkg == null) throw new IllegalArgumentException("Package not found");

double packagePrice = pkg.getPrice();

double insurancePrice = 0.0;

if (booking.getInsuranceId() != null && booking.getInsuranceId() > 0) {

insurancePrice = travelInsuranceClient.getInsurancePriceByInsuranceId(booking.getInsuranceId().intValue());

}

double discountAmount = 0.0;

if (couponCodeApplied != null && pkg.getOffer() != null) {

OfferDTO offer = pkg.getOffer();

if (couponCodeApplied.equalsIgnoreCase(offer.getCouponCode()) && offer.isActive()) {

discountAmount = (packagePrice \* offer.getDiscountPercentage()) / 100.0;

}

}

return packagePrice + insurancePrice - discountAmount;

}

public List<Payment> getAllPayments() {

return paymentRepo.findAll();

}

public Payment getPaymentById(Long id) {

return paymentRepo.findById(id).orElse(null);

}

}spring.application.name=TravelBooking-PaymentModule

spring.datasource.url=jdbc:mysql://localhost:3306/booking\_db

server.port=8086

spring.datasource.username=root

spring.datasource.password=root

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

logging.level.org.hibernate.SQL=debug

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

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=indhaanman@gmail.com

spring.mail.password=ibzbwveckpfjibey

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

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

spring.cloud.compatibility-verifier.enabled=false

# optional:Srt timeout and logging

spring.cloud.openfeign.client.config.default.connectTimeout=5000

spring.cloud.openfeign.client.config.default.readTimeout=5000

spring.cloud.openfeign.client.config.default.loggerLevel=full

# Eureka Client Configuration

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

eureka.instance.prefer-ip-address=true

#debug=true

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

logging.level.root=DEBUG

logging.level.org.springframework.web=DEBUG

// RatingsAndReviews

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);

}

}

-------------------------------------------------------------------------------------------

//Booking Client

**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;

@FeignClient(name = "TravelBooking\_PaymentModule")

**public** **interface** BookingClient {

@GetMapping("/api/bookings/user/{userId}/package/{packageId}/completed")

**boolean** hasCompletedBooking(@PathVariable("userId") Long userId, @PathVariable("packageId") String packageId);

}

-----------------------------------------------------------------------------------------

//Travel Package Client

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);

}

------------------------------------------------------------------------------------------

//UserClient

**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.UserDTO;

@FeignClient(name = "usermanagement-service")

**public** **interface** UserClient {

@GetMapping("/api/users/internal/customer/{id}")

UserDTO getUserById(@PathVariable("id") Long id);

}

UserDTO

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

**import** lombok.Data;

@Data

**public** **class** UserDTO {

**private** Long id;

**private** String name;

**private** String email;

**private** String role;

}

**//** TravelPackageDTO

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

**import** lombok.Data;

@Data

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

**private** Long packageId;

**private** Long agentId;

**private** String title;

}

//Review

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; }

}

-------------------------------------------------------------------------------------------------------

//Review Controller

**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("packageId", savedReview.getPackageId());

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.");

}

}

// ReviewRepository

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);

boolean existsByUserIdAndPackageId(Long userId, Long packageId);

}

// **package** com.ratings.review.service;

**import** com.ratings.review.client.BookingClient;

**import** com.ratings.review.client.UserClient;

**import** com.ratings.review.dto.UserDTO;

**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;

@Autowired

**private** UserClient userClient;

@Autowired

**private** BookingClient bookingClient;

/\*\*

\* Posts a new review.

\*/

**public** Review postReview(Review review) {

// 1. Validate user exists and is CUSTOMER

UserDTO user = userClient.getUserById(review.getUserId());

**if** (user == **null** || !"CUSTOMER".equalsIgnoreCase(user.getRole())) {

**throw** **new** ResourceNotFoundException("Only customers can post reviews.");

}

// 2. Check if user has completed trip

**boolean** completed = bookingClient.hasCompletedBooking(review.getUserId(), review.getPackageId().toString());

**if** (!completed) {

**throw** **new** ResourceNotFoundException("Journey not completed for the given package.");

}

// 3. Prevent duplicate reviews

**if** (reviewRepository.existsByUserIdAndPackageId(review.getUserId(), review.getPackageId())) {

**throw** **new** ResourceNotFoundException("You have already reviewed this package.");

}

// 4. Save review

review.setTimestamp(LocalDateTime.*now*());

**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);

}

}

//Agent

package com.ratings.review.entity;

import jakarta.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "agent\_responses")

public class AgentResponse {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long responseId;

@Column(nullable = false)

private Long reviewId;

@Column(nullable = false)

private Long packageId;

@Column(nullable = false)

private Long agentId;

@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 Long getReviewId() { return reviewId; }

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

public Long getPackageId() { return packageId; }

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

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; }

}

//AgentController

package com.ratings.review.controller;

import com.ratings.review.entity.AgentResponse;

import com.ratings.review.entity.Review;

import com.ratings.review.exception.ResourceNotFoundException;

import com.ratings.review.service.AgentResponseService;

import com.ratings.review.repository.ReviewRepository;

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")

public class AgentResponseController {

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

@Autowired

private AgentResponseService agentResponseService;

@Autowired

private ReviewRepository reviewRepository;

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

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

@PathVariable Long agentId,

@PathVariable Long reviewId,

@RequestBody Map<String, String> requestBody) {

String responseMessage = requestBody.get("responseMessage");

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

Review review = reviewRepository.findById(reviewId)

.orElseThrow(() -> new ResourceNotFoundException("Review not found"));

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

responseBody.put("packageId", review.getPackageId());

responseBody.put("reviewId", reviewId);

responseBody.put("reviewMessage", review.getComment());

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

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

return ResponseEntity.ok(responseBody);

}

@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);

}

@PutMapping("/{responseId}")

public ResponseEntity<AgentResponse> updateAgentResponse(

@PathVariable Long responseId,

@RequestBody Map<String, String> requestBody) {

String updatedMessage = requestBody.get("responseMessage");

AgentResponse updatedResponse = agentResponseService.updateResponse(responseId, updatedMessage);

return ResponseEntity.ok(updatedResponse);

}

@DeleteMapping("/{responseId}")

public ResponseEntity<Map<String, String>> deleteAgentResponse(@PathVariable Long responseId) {

agentResponseService.deleteResponse(responseId);

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

response.put("message", "Agent response deleted successfully");

return ResponseEntity.ok(response);

}

}

//Agent Service

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;

import java.util.Map;

@Service

public class AgentResponseService {

@Autowired

private AgentResponseRepository agentResponseRepository;

@Autowired

private ReviewRepository reviewRepository;

@Autowired

private TravelPackageClient travelPackageClient;

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

Review review = reviewRepository.findById(reviewId)

.orElseThrow(() -> new ResourceNotFoundException("Review not found."));

Long packageId = review.getPackageId();

Map<String, Long> response = travelPackageClient.getAgentIdByPackageId(packageId);

Long actualAgentId = response.get("agentId");

if (!actualAgentId.equals(agentId)) {

throw new ResourceNotFoundException("Unauthorized: You are not the owner of this package.");

}

AgentResponse agentresponse = new AgentResponse();

agentresponse.setReviewId(reviewId);

agentresponse.setPackageId(packageId);

agentresponse.setAgentId(agentId);

agentresponse.setResponseMessage(responseMessage);

agentresponse.setResponseTime(LocalDateTime.now());

return agentResponseRepository.save(agentresponse);

}

public List<AgentResponse> getResponsesForReview(Long reviewId) {

return agentResponseRepository.findByReviewId(reviewId);

}

public AgentResponse updateResponse(Long responseId, String updatedMessage) {

AgentResponse response = agentResponseRepository.findById(responseId)

.orElseThrow(() -> new ResourceNotFoundException("Agent Response not found"));

response.setResponseMessage(updatedMessage);

response.setResponseTime(LocalDateTime.now());

return agentResponseRepository.save(response);

}

public void deleteResponse(Long responseId) {

AgentResponse response = agentResponseRepository.findById(responseId)

.orElseThrow(() -> new ResourceNotFoundException("Agent Response not found"));

agentResponseRepository.delete(response);

}

}

//AgentResponse Repository

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> findByReviewId(Long reviewId);

}

//Application.properties

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

//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>3.2.5</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>Ratings\_and\_Review\_Module-2\_final</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Ratings\_and\_Review\_Module-2\_final</name>

<description>Project Module for Ratings and Reviews</description>

<properties>

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

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>2023.0.1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Boot Starters -->

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<!-- Eureka Client -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<!-- OpenFeign -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<!-- MySQL Connector -->

<dependency>

<groupId>com.mysql</groupId>

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

<scope>runtime</scope>

</dependency>

<!--Lombok-->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.30</version>

<scope>provided</scope>

</dependency>

<!-- DevTools -->

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- JUnit -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Spring Boot Test -->

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-releases</id>

<name>Spring Releases</name>

<url>https://repo.spring.io/release</url>

</repository>

</repositories>

</project>

//src\app\admin-agent-packages\admin-agent-packages.component.css

/\* Center the heading and add spacing \*/

h2 {

    text-align: center;

    margin: 20px 0;

    color: #333;

    font-size: 24px;

  }

  /\* Style the table \*/

  table {

    width: 90%;

    margin: 0 auto;

    border-collapse: collapse;

  }

  th, td {

    padding: 12px;

    text-align: left;

    border-bottom: 1px solid #ddd;

  }

  th {

    background-color: #f8f8f8;

    color: #444;

  }

  tr:hover {

    background-color: #f1f1f1;

  }

  /\* Message if no packages found \*/

  .no-packages {

    text-align: center;

    color: gray;

    margin-top: 30px;

    font-size: 18px;

  }

//src\app\admin-agent-packages\admin-agent-packages.component.html

<div class="container">

  <h2>Packages Listed by Agent #{{ agentId }}</h2>

<!-- can add more data here to show in the frontend -->

  <table \*ngIf="packages.length > 0; else noPackages">

    <thead>

      <tr>

        <th>Title</th>

        <th>Description</th>

        <th>Price (₹)</th>

        <th>Duration (days)</th>

        <th>Start Date</th>

        <th>End Date</th>

        <th>Max Capacity</th>

      </tr>

    </thead>

    <tbody>

      <tr \*ngFor="let pkg of packages">

        <td>{{ pkg.title }}</td>

        <td>{{ pkg.description }}</td>

        <td>{{ pkg.price }}</td>

        <td>{{ pkg.duration }}</td>

        <td>{{ pkg.tripStartDate }}</td>

        <td>{{ pkg.tripEndDate }}</td>

        <td>{{ pkg.maxCapacity }}</td>

      </tr>

    </tbody>

  </table>

  <ng-template #noPackages>

    <p>No packages found for this agent.</p>

  </ng-template>

</div>

//src\app\admin-agent-packages\admin-agent-packages.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { AdminAgentPackagesComponent } from './admin-agent-packages.component';

describe('AdminAgentPackagesComponent', () => {

  let component: AdminAgentPackagesComponent;

  let fixture: ComponentFixture<AdminAgentPackagesComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [AdminAgentPackagesComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(AdminAgentPackagesComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\admin-agent-packages\admin-agent-packages.component.ts

import { Component, OnInit } from '@angular/core';

import { CommonModule } from '@angular/common';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { ActivatedRoute } from '@angular/router';

@Component({

  selector: 'app-admin-agent-packages',

  standalone: true,

  imports: [CommonModule],

  templateUrl: './admin-agent-packages.component.html',

  styleUrls: ['./admin-agent-packages.component.css']

})

export class AdminAgentPackagesComponent implements OnInit {

  agentId!: number; // ! tells ts we will initiate it later

  packages: any[] = [];

  constructor(

    private route: ActivatedRoute,

    private http: HttpClient

  ) {}

  ngOnInit(): void {

    // Get agent ID from route parameter

    this.agentId = Number(this.route.snapshot.paramMap.get('id'));

    console.log("Agent ID:",this.agentId);

    // Call backend to fetch agent's packages

    const token = localStorage.getItem('token');

    const headers = new HttpHeaders().set('Authorization', `Bearer ${token}`);

    this.http

    .get<any>(`http://localhost:8080/api/packages/agent/${this.agentId}`)

      .subscribe({

        next: (res) => {

          console.log("Fetched packages", res);

          this.packages = res.data; // res is the full object, data contains array

        },

        error: (err) => {

          console.error('Failed to fetch packages', err);

          this.packages = [];

        }

      });

  }

}

//src\app\admin-dashboard\admin-dashboard.component.css

.dashboard {

    padding: 20px;

    text-align: center;

  }

  .summary {

    margin: 20px auto;

    font-size: 18px;

    line-height: 1.6;

  }

  .buttons {

    display: flex;  /\*Arrange buttons in a row\*/

    justify-content: center; /\*Center the buttons horizontally\*/

    gap: 16px; /\* Add spaCE BETWEEN the buttons\*/

    margin-top: 20px;

  }

  .buttons button {

    margin-top: 0 10px;

    padding: 10px 20px;

    font-weight: bold;

    background-color: #4caf50;

    color: white;

    border: none;

    border-radius: 5px;

    cursor: pointer;

  }

//src\app\admin-dashboard\admin-dashboard.component.html

<div class="dashboard">

  <h2>Admin Dashboard</h2>

  <p>Welcome, Admin! Here's a summary of platform users:</p>

  <div class="summary">

    <p><strong>Total Users (excluding admin):</strong> {{ totalUsers }}</p>

    <p><strong>Total Agents:</strong> {{ totalAgents }}</p>

    <p><strong>Total Customers:</strong> {{ totalCustomers }}</p>

  </div>

  <div class="buttons">

    <button (click)="goToProfile()">Go to My Profile</button>

    <button (click)="goToManageUsers()">Manage Users</button>

  </div>

</div>

//src\app\admin-dashboard\admin-dashboard.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { AdminDashboardComponent } from './admin-dashboard.component';

describe('AdminDashboardComponent', () => {

  let component: AdminDashboardComponent;

  let fixture: ComponentFixture<AdminDashboardComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [AdminDashboardComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(AdminDashboardComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\admin-dashboard\admin-dashboard.component.ts

import { Component, OnInit } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { ToastrService } from 'ngx-toastr';

import { Router } from '@angular/router';

interface UserRoleCountResponse {

  totalUsers: number;

  agentCount: number;

  customerCount: number;

}

@Component({

  selector: 'app-admin-dashboard',

  templateUrl: './admin-dashboard.component.html',

  styleUrls: ['./admin-dashboard.component.css']

})

export class AdminDashboardComponent implements OnInit {

  totalUsers: number = 0;

  totalAgents: number = 0;

  totalCustomers: number = 0;

  constructor(

    private http: HttpClient,

    private toastr: ToastrService,

    private router: Router

  ) {}

  ngOnInit(): void {

    this.fetchUserCounts();

  }

  fetchUserCounts() {

    const token = localStorage.getItem('token');

this.http.get<UserRoleCountResponse>('http://localhost:8080/api/users/counts', {

      headers: { Authorization: `Bearer ${token}` }

    }).subscribe({

      next: (res) => {

        this.totalUsers = res.totalUsers;

        this.totalAgents = res.agentCount;

        this.totalCustomers = res.customerCount;

      },

      error: () => this.toastr.error('Failed to fetch user counts.')

    });

  }

  goToProfile() {

    this.router.navigate(['/my-profile']);

  }

  goToManageUsers() {

    this.router.navigate(['/admin-user-manage']);

  }

}

//src\app\admin-user-manage\admin-user-manage.component.css

/\* Container spacing \*/

.container {

  padding: 20px;

  font-family: Arial, sans-serif;

}

/\* Table layout \*/

table {

  width: 100%;

  border-collapse: collapse;

  margin-top: 20px;

}

/\* Table headers \*/

th {

  background-color: #f0f0f0;

  text-align: left;

  padding: 10px;

  font-weight: 600;

  border-bottom: 1px solid #ddd;

}

/\* Table rows \*/

td {

  padding: 10px;

  border-bottom: 1px solid #ddd;

  vertical-align: middle;

}

/\* Buttons in actions/view column \*/

button {

  padding: 5px 10px;

  margin-right: 5px;

  background-color: #007bff;

  color: white;

  border: none;

  border-radius: 4px;

  cursor: pointer;

}

button:hover {

  background-color: #0056b3;

}

/\* Align buttons in same row \*/

td button + button {

  margin-left: 5px;

}

/\* Make select box look nicer \*/

select {

  padding: 5px;

  border-radius: 4px;

  border: 1px solid #ccc;

}

h2 {

  text-align: center;

  margin-top: 20px;

  margin-bottom: 20px;

  font-size: 24px;

  color: #333;

}

//src\app\admin-user-manage\admin-user-manage.component.html

<div class="manage-container">

    <h2>All Registered Users</h2>

    <table border="1">

      <thead>

        <tr>

          <th>Name</th>

          <th>Email</th>

          <th>Phone</th>

          <th>View</th>

          <th>Role</th>

          <th>Actions</th>

        </tr>

        </thead>

        <tbody>

        <tr \*ngFor="let user of users">

          <td>{{ user.name }}</td>

          <td>{{ user.email }}</td>

          <td>{{ user.contactNumber }}</td>

          <td>

            <a \*ngIf="user.role === 'AGENT'"

            [routerLink]="['/admin/agent-packages', user.id]">

            <button>View Packages</button>

            </a>

          <td>

            <select [value]="user.role"

            (change)="onRoleChange(user, $event)">

              <option value="CUSTOMER">CUSTOMER</option>

              <option value="AGENT">AGENT</option>

          </select>

          </td>

          <td>

            <button (click)="deleteUser(user.id)">Delete</button>

          </td>

        </tr>

      </tbody>

    </table>

  </div>

//src\app\admin-user-manage\admin-user-manage.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { AdminUserManageComponent } from './admin-user-manage.component';

describe('AdminUserManageComponent', () => {

  let component: AdminUserManageComponent;

  let fixture: ComponentFixture<AdminUserManageComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [AdminUserManageComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(AdminUserManageComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\admin-user-manage\admin-user-manage.component.ts

import { Component, OnInit } from '@angular/core';

import { HttpClient, HttpHeaders, HttpClientModule } from '@angular/common/http';

import { ToastrService } from 'ngx-toastr';

import { CommonModule } from '@angular/common';

import { Router } from '@angular/router';

import { RouterModule } from '@angular/router';

@Component({

  selector: 'app-admin-user-manage',

  standalone: true,

  imports: [CommonModule, RouterModule, HttpClientModule],

  templateUrl: './admin-user-manage.component.html',

  styleUrls: ['./admin-user-manage.component.css']

})

export class AdminUserManageComponent implements OnInit {

  users: any[] = [];

  constructor(private router: Router, private http: HttpClient, private toastr: ToastrService) {}

  ngOnInit(): void {

    this.fetchUsers();

  }

  // 🔹 Fetch all users from backend

  fetchUsers(): void {

    const token = localStorage.getItem('token');

    this.http.get<any[]>('http://localhost:8080/api/users', {

      headers: { Authorization: `Bearer ${token}` }

    }).subscribe({

      //next: (res) => this.users = res,

      next: (res) => {

        this.users = res.filter(user => user.role && user.role.toUpperCase() !== 'ADMIN');

      },

      error: () => this.toastr.error('Failed to load users')

    });

  }

  // Called when admin changes the role of a user from dropdown

onRoleChange(user: any, event:Event): void {

  const selectedRole = (event.target as HTMLSelectElement).value;

  // Prevent manually assigning ADMIN role

  if (selectedRole === 'ADMIN') {

    this.toastr.warning('Admin role cannot be assigned manually.');

    return;

  }

  // Get token from localStorage

  const token = localStorage.getItem('token');

  // Create a copy of user with updated role

  const updatedUser = { ...user, role: selectedRole };

  // Make PUT request to update role

this.http.put(`http://localhost:8080/api/users/${user.id}`, updatedUser, {

    headers: {

      Authorization: `Bearer ${token}`

    }

  }).subscribe({

    next: () => {

      this.toastr.success('Role updated successfully');

      this.fetchUsers(); // Refresh the user list after update

    },

    error: () => {

      this.toastr.error('Failed to update role');

    }

  });

}

  // 🔹 Delete user

  deleteUser(id: number): void {

    const token = localStorage.getItem('token');

  this.http.delete(`http://localhost:8080/api/users/${id}`, {

      headers: { Authorization: `Bearer ${token}` },

      responseType: 'text', //avoids json parsing

      observe: 'response'  //  Observe the full HTTP response

    }).subscribe({

      next: (response) => {

        // Check if status is 200 OK or 204 No Content

        if (response.status === 200 || response.status === 204) {

          this.toastr.success('User deleted successfully');

          window.location.reload(); // ⟳ Refresh the page to update the UI

        } else {

          this.toastr.error('Unexpected response from server');

        }

      },

      error: (err) => {

        console.error("❌ Delete failed:", err);

        this.toastr.error('Failed to delete user');

      }

    });

  }

  //view agent packages

  viewAgentPackages(agentId: number) {

    this.router.navigate(['/admin/agent-packages',agentId]);

  }

  /\* deleteUser(id: number): void {

    const token = localStorage.getItem('token');

this.http.delete(`http://localhost:8080/api/users/${id}`, {

      headers: { Authorization: `Bearer ${token}` }

    }).subscribe({

      next: () => {

        //removes user from local list

        //this.users = this.users.filter(user => user.id !== id);

        this.toastr.success('User deleted');

        window.location.reload();

      },

      error: () => this.toastr.error('Failed to delete user')

    });

  }\*/

}

//src\app\agent-dashboard\agent-dashboard.component.css

.dashboard-container {

    padding: 20px;

    font-family: Arial, sans-serif;

  }

  button {

    margin: 5px;

    padding: 8px 14px;

    border: none;

    background-color: #2196f3;

    color: white;

    border-radius: 4px;

    cursor: pointer;

  }

  button:hover {

    background-color: #0b7dda;

  }

  ul {

    list-style: none;

    padding: 0;

  }

  li {

    margin: 10px 0;

    padding: 10px;

    background: #f1f1f1;

    border-radius: 5px;

  }

//src\app\agent-dashboard\agent-dashboard.component.html

<div class="dashboard-container">

    <h2>Welcome Agent!</h2>

    <button (click)="showCreateForm()">➕ Create New Package</button>

    <button (click)="goToProfile()">👤 My Profile</button>

    <hr>

    <div \*ngIf="packages.length > 0; else noPackages">

      <h3>Your Travel Packages:</h3>

      <ul>

        <li \*ngFor="let pkg of packages">

          <strong>{{ pkg.title }}</strong> – {{ pkg.destination }} (₹{{ pkg.price }})

          <!--view details of the package-->

          <button [routerLink]="['/view-package', pkg.packageId]">View Details</button>

          <button [routerLink]="['/edit-package', pkg.packageId]">Edit</button>

          <button (click)="deletePackage(pkg.packageId)">🗑️ Delete</button>

        </li>

      </ul>

    </div>

    <ng-template #noPackages>

      <p>No packages found. Click "Create New Package" to add one!</p>

    </ng-template>

    <hr/>

  <!-- CREATE NEW PACKAGE FORM -->

  <div \*ngIf="formVisible">

    <h3>Create New Travel Package</h3>

    <form (ngSubmit)="submitPackage()" #pkgForm="ngForm">

      <!-- Basic Info -->

      <label>Title: </label>

      <input type="text" placeholder="Title" [(ngModel)]="newPackage.title" name="title" required><br>

      <label>Description: </label><br>

      <textarea placeholder="Description" [(ngModel)]="newPackage.description" name="description" required></textarea><br>

      <label>Duration: </label>

      <input type="number" placeholder="Duration (days)" [(ngModel)]="newPackage.duration" name="duration"><br>

      <label>Price: </label>

      <input type="number" placeholder="Price" [(ngModel)]="newPackage.price" name="price"><br>

      <label>Max Capacity: </label>

      <input type="number" placeholder="Max Capacity" [(ngModel)]="newPackage.maxCapacity" name="maxCapacity"><br>

      <label>Country: </label>

      <input type="text" placeholder="Country" [(ngModel)]="newPackage.country" name="country"><br>

      <label>Destination: </label>

      <input type="text" placeholder="Destination" [(ngModel)]="newPackage.destination" name="destination"><br>

      <label>Trip Type: </label>

       <select [(ngModel)]="newPackage.tripType" name="tripType" required>

        <option value="">-- Select Trip Type --</option>

        <option value="International Trip">International Trip</option>

        <option value="Domestic Trip">Domestic Trip</option>

       </select><br>

      <label>Trip Start Date:</label>

      <input type="date" [(ngModel)]="newPackage.tripStartDate" name="tripStartDate"><br>

      <label>Trip End Date:</label>

      <input type="date" [(ngModel)]="newPackage.tripEndDate" name="tripEndDate"><br>

      <!--Offer Section-->

      <h4>Offer Details</h4>

      <label>Offer Description: </label>

      <textarea placeholder="Offer Description" [(ngModel)]="newPackage.offer.description" name="offerDesc"></textarea><br>

      <label>Coupon Code: </label>

      <input type="text" placeholder="Coupon Code" [(ngModel)]="newPackage.offer.couponCode" name="Coupon Code"><br>

      <label>Discount Percentage: </label>

      <input type="number" placeholder="Discount (%)" [(ngModel)]="newPackage.offer.discountPercentage" name="discountPercentage"><br>

      <label><input type="checkbox" [(ngModel)]="newPackage.offer.active" name="offerActive">Offer Active</label><br>

      <!--Flights-->

      <h4>Flight Details</h4>

      <div \*ngFor="let flight of newPackage.flights; let i=index">

        <label>Airline: </label>

        <input type="text" placeholder="Airline" [(ngModel)]="flight.airline" [name]="'airline'+i"><br>

        <label>From City: </label>

        <input type="text" placeholder="From City" [(ngModel)]="flight.fromCity" [name]="'fromCity'+i"><br>

        <label>To City: </label>

        <input type="text" placeholder="To City" [(ngModel)]="flight.toCity" [name]="'toCity'+i"><br>

        <label>Departire Time: </label>

        <input type="text" placeholder="Departure Time" [(ngModel)]="flight.departureTime" [name]="'departureTime'+i"><br>

        <label>Arrival Time: </label>

        <input type="text" placeholder="Arrival Time" [(ngModel)]="flight.arrivalTime" [name]="'arrivalTime'+i"><br>

      </div>

      <!-- Hotels -->

      <h4>Hotel Details</h4>

      <div \*ngFor="let hotel of newPackage.hotels; let i=index">

        <label>Hotel Name: </label>

        <input type="text" placeholder="Hotel Name" [(ngModel)]="hotel.name" [name]="'hotelName'+i"><br>

        <label>City: </label>

        <input type="text" placeholder="City" [(ngModel)]="hotel.city" [name]="'hotelCity'+i"><br>

        <label>Rating: </label>

        <input type="number" placeholder="Rating" [(ngModel)]="hotel.rating" [name]="'hotelRating'+i"><br>

        <label>Nights Staying: </label>

        <input type="number" placeholder="Nights" [(ngModel)]="hotel.nights" [name]="'hotelNights'+i"><br>

        <label>Cost per Night: </label>

        <input type="number" placeholder="Cost/Night" [(ngModel)]="hotel.costPerNight" [name]="'hotelCost'+i"><br>

      </div>

      <!-- Sightseeing -->

      <h4>Sightseeing</h4>

      <div \*ngFor="let sight of newPackage.sightseeing; let i=index">

        <label>Location: </label>

        <input type="text" placeholder="Location" [(ngModel)]="sight.location" [name]="'sightLoc'+i"><br>

        <label>Sight Description: </label>

        <input type="text" placeholder="Description" [(ngModel)]="sight.description" [name]="'sightDesc'+i"><br>

        <button type="button" (click)="removeSightseeing(i)" \*ngIf="newPackage.sightseeing.length > 1">

          "Remove Sightseeing"

        </button>

      </div>

      <button type="button" (click)="addSightseeing()">+ Add Sightseeing</button>

      <!-- Itinerary -->

      <h4>Itinerary</h4>

      <div \*ngFor="let item of newPackage.itinerary; let i=index">

        <label>Day Number: </label>

        <input type="number" placeholder="Day #" [(ngModel)]="item.dayNumber" [name]="'day'+i" placeholder="Day #" required><br>

        <label>Activity Title: </label>

        <input type="text" placeholder="Activity Title" [(ngModel)]="item.activityTitle" [name]="'activityTitle'+i" placeholder="Activity Description" required><br>

        <label>Activity Description: </label>

        <input type="text" placeholder="Activity Description" [(ngModel)]="item.activityDescription" [name]="'activityDesc'+i" required><br>

        <button type="button" (click)="removeItineraryDay(i)" \*ngIf="newPackage.itinerary.length>1">Remove Day</button>

        <hr>

      </div>

      <button type="button" (click)="addItineraryDay()">+Add Itinerary Day</button>

      <!-- Submit and Cancel -->

      <br />

      <button type="submit">Submit Package</button>

      <button type="button" (click)="cancelForm()">Cancel</button>

    </form>

  </div>

</div>

//src\app\agent-dashboard\agent-dashboard.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { AgentDashboardComponent } from './agent-dashboard.component';

describe('AgentDashboardComponent', () => {

  let component: AgentDashboardComponent;

  let fixture: ComponentFixture<AgentDashboardComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [AgentDashboardComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(AgentDashboardComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\agent-dashboard\agent-dashboard.component.ts

import { Component, OnInit, inject } from '@angular/core';

import { HttpClient, HttpClientModule } from '@angular/common/http';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { RouterLink } from '@angular/router';

@Component({

  selector: 'app-agent-dashboard',

  standalone: true,

  imports: [CommonModule, HttpClientModule, FormsModule, RouterLink],

  templateUrl: './agent-dashboard.component.html',

  styleUrl: './agent-dashboard.component.css'

})

export class AgentDashboardComponent implements OnInit{

  packages: any[] = []; //store the list of travel packages

  formVisible: boolean = false;

  agentId :number=0;

  newPackage: any = {

    title:'',

    description:'',

    duration:'',

    price:'',

    maxCapacity:'',

    country:'',

    destination:'',

    tripType:'',

    tripStartDate:'',

    tripEndDate:'',

    active: true,

    offer: { couponCode:'', description:'', discountPercentage: 0, active: false},

    flights: [

      {airline: '', fromCity:'', toCity:'', departureTime:'', arrivalTime:''}

    ],

    hotels: [

      {name:'', city:'', rating:'', nights:'', costPerNight:''}

    ],

    sightseeing: [

      {location:'', description:''}

    ],

    itinerary: [

      {dayNumber:'', activityTitle:'', activityDescription:''}

    ]

  };

  private http = inject(HttpClient);

  ngOnInit(): void {

    const storedId = localStorage.getItem('userId');

    this.agentId = storedId ? parseInt(storedId, 10) : 0;

    if (this.agentId>0) {

      this.loadPackages();

    } else {

      console.error('Agent ID not found in localStorage');

      // optionally redirect to login page or show error

    }

  }

  //Get packages created by this agent

  loadPackages(){

    const url = `http://localhost:8080/api/packages/agent/${this.agentId}`;

    this.http.get<any>(url).subscribe({

      next: (res) =>{

        this.packages = res.data;

        console.log("Loaded packages:", this.packages);//optional debug

      } ,

      error: (err) => console.error('Error loading packages', err)

    });

  }

  //Submit new package to backend

  submitPackage(){

    const url= `http://localhost:8080/api/packages`;

    const fullData = {

      ...this.newPackage,

      agentId: this.agentId

    };

    this.http.post(url, fullData).subscribe({

      next: () => {

        alert('Package created successfully!');

        this.formVisible = false;

        this.loadPackages();

        this.resetForm();

      },

      error: (err) => {

        console.error('Error correcting package', err);

        alert('Failed to create package. Check your input.');

      }

    });

  }

  //Delete a package

  deletePackage(id: number){

    if(!id){

      console.error('Invalid package ID:',id);

      alert('Error: Cannot delete package. Invalid ID.');

      return ;

    }

    const confirmDelete = confirm("Are you sure you want to delete this package?");

    if(!confirmDelete) return;

    const url = `http://localhost:8080/api/packages/${id}`;

    this.http.delete(url).subscribe({

      next: () => {

        alert('Package deleted successfully!');

        this.loadPackages();

      },

      error: (err) => {

        console.error('Error deleting package', err);

        alert('Failed to delete Package.')

      }

    });

  }

  //show or hide create form

  showCreateForm(){

    this.formVisible = true;

  }

  cancelForm() {

    this.formVisible = false;

  }

  //placeholder for edit

  editPackage(pkg: any){

    alert('Edit package: ${pkg.title} - feature coming soon!');

  }

  //Reset form after success

  resetForm() {

    this.newPackage = {

      title:'',

      description:'',

      duration:'',

      price:'',

      maxCapacity:'',

      country:'',

      destination:'',

      tripType:'',

      tripStartDate:'',

      tripEndDate:'',

      active: true,

      offer: { couponCode:'', description:'', discountPercentage: 0, active: false},

      flights: [

        {airline: '', fromCity:'', toCity:'', departureTime:'', arrivalTime:''}

      ],

      hotels: [

        {name:'', city:'', rating:'', nights:'', costPerNight:''}

      ],

      sightseeing: [

        {location:'', description:''}

      ],

      itinerary: [

        {dayNumber:'', activityTitle:'', activityDescription:''}

      ]

    };

  }

  addItineraryDay(){

    const newDay = this.newPackage.itinerary.length+1;

    this.newPackage.itinerary.push({

      dayNumber: newDay,

      activityTitle: '',

      activityDescription: ''

    });

  }

  removeItineraryDay(index:number): void{

    this.newPackage.itinerary.splice(index,1);

    this.newPackage.itinerary.forEach((item: any, i: number) =>{

      item.dayNumber = i+1;

    });

  }

  // for sight seeing

  addSightseeing(): void {

    this.newPackage.sightseeing.push({location:'', description:''});

  }

  removeSightseeing(index: number): void {

    this.newPackage.sightseeing.splice(index,1);

  }

  //Go to my profile

  goToProfile(){

    window.location.href = '/my-profile';

  }

}

//src\app\customer-dashboard\customer-dashboard.component.css

//src\app\customer-dashboard\customer-dashboard.component.html

<p>customer-dashboard works!</p>

<a routerLink="/my-profile">My Profile</a>

//src\app\customer-dashboard\customer-dashboard.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { CustomerDashboardComponent } from './customer-dashboard.component';

describe('CustomerDashboardComponent', () => {

  let component: CustomerDashboardComponent;

  let fixture: ComponentFixture<CustomerDashboardComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [CustomerDashboardComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(CustomerDashboardComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\customer-dashboard\customer-dashboard.component.ts

import { Component } from '@angular/core';

@Component({

  selector: 'app-customer-dashboard',

  imports: [],

  templateUrl: './customer-dashboard.component.html',

  styleUrl: './customer-dashboard.component.css'

})

export class CustomerDashboardComponent {

}

//src\app\edit-package\edit-package.component.css

//src\app\edit-package\edit-package.component.html

<div class="edit-container">

    <h2>Edit Travel Package</h2>

    <form (ngSubmit)="submitPackage()" #pkgForm="ngForm">

        <!-- Basic Info -->

        <label>Title: </label>

        <input type="text" placeholder="Title" [(ngModel)]="newPackage.title" name="title" required><br>

        <label>Description: </label><br>

        <textarea placeholder="Description" [(ngModel)]="newPackage.description" name="description" required></textarea><br>

        <label>Duration: </label>

        <input type="number" placeholder="Duration (days)" [(ngModel)]="newPackage.duration" name="duration"><br>

        <label>Price: </label>

        <input type="number" placeholder="Price" [(ngModel)]="newPackage.price" name="price"><br>

        <label>Max Capacity: </label>

        <input type="number" placeholder="Max Capacity" [(ngModel)]="newPackage.maxCapacity" name="maxCapacity"><br>

        <label>Country: </label>

        <input type="text" placeholder="Country" [(ngModel)]="newPackage.country" name="country"><br>

        <label>Destination: </label>

        <input type="text" placeholder="Destination" [(ngModel)]="newPackage.destination" name="destination"><br>

        <label>Trip Type: </label>

         <select [(ngModel)]="newPackage.tripType" name="tripType" required>

          <option value="">-- Select Trip Type --</option>

          <option value="International Trip">International Trip</option>

          <option value="Domestic Trip">Domestic Trip</option>

         </select><br>

        <label>Trip Start Date:</label>

        <input type="date" [(ngModel)]="newPackage.tripStartDate" name="tripStartDate"><br>

        <label>Trip End Date:</label>

        <input type="date" [(ngModel)]="newPackage.tripEndDate" name="tripEndDate"><br>

        <!--Offer Section-->

        <h4>Offer Details</h4>

        <label>Offer Description: </label>

        <textarea placeholder="Offer Description" [(ngModel)]="newPackage.offer.description" name="offerDesc"></textarea><br>

        <label>Coupon Code: </label>

        <input type="text" placeholder="Coupon Code" [(ngModel)]="newPackage.offer.couponCode" name="Coupon Code"><br>

        <label>Discount Percentage: </label>

        <input type="number" placeholder="Discount (%)" [(ngModel)]="newPackage.offer.discountPercentage" name="discountPercentage"><br>

        <label><input type="checkbox" [(ngModel)]="newPackage.offer.active" name="offerActive">Offer Active</label><br>

        <!--Flights-->

        <h4>Flight Details</h4>

        <div \*ngFor="let flight of newPackage.flights; let i=index">

          <label>Airline: </label>

          <input type="text" placeholder="Airline" [(ngModel)]="flight.airline" [name]="'airline'+i"><br>

          <label>From City: </label>

          <input type="text" placeholder="From City" [(ngModel)]="flight.fromCity" [name]="'fromCity'+i"><br>

          <label>To City: </label>

          <input type="text" placeholder="To City" [(ngModel)]="flight.toCity" [name]="'toCity'+i"><br>

          <label>Departire Time: </label>

          <input type="text" placeholder="Departure Time" [(ngModel)]="flight.departureTime" [name]="'departureTime'+i"><br>

          <label>Arrival Time: </label>

          <input type="text" placeholder="Arrival Time" [(ngModel)]="flight.arrivalTime" [name]="'arrivalTime'+i"><br>

        </div>

        <!-- Hotels -->

        <h4>Hotel Details</h4>

        <div \*ngFor="let hotel of newPackage.hotels; let i=index">

          <label>Hotel Name: </label>

          <input type="text" placeholder="Hotel Name" [(ngModel)]="hotel.name" [name]="'hotelName'+i"><br>

          <label>City: </label>

          <input type="text" placeholder="City" [(ngModel)]="hotel.city" [name]="'hotelCity'+i"><br>

          <label>Rating: </label>

          <input type="number" placeholder="Rating" [(ngModel)]="hotel.rating" [name]="'hotelRating'+i"><br>

          <label>Nights Staying: </label>

          <input type="number" placeholder="Nights" [(ngModel)]="hotel.nights" [name]="'hotelNights'+i"><br>

          <label>Cost per Night: </label>

          <input type="number" placeholder="Cost/Night" [(ngModel)]="hotel.costPerNight" [name]="'hotelCost'+i"><br>

        </div>

        <!-- Sightseeing -->

        <h4>Sightseeing</h4>

        <div \*ngFor="let sight of newPackage.sightseeing; let i=index">

          <label>Location: </label>

          <input type="text" placeholder="Location" [(ngModel)]="sight.location" [name]="'sightLoc' +i"><br>

          <label>Sight Description: </label>

          <input type="text" placeholder="Description" [(ngModel)]="sight.description" [name]="'sightDesc' +i"><br>

          <button type="button" (click)="removeSightseeing(i)" \*ngIf="newPackage.sightseeing.length >1 ">

            Remove Sightseeing

          </button>

          <hr>

        </div>

        <button type="button" (click)="addSightseeing()">+ Add Sightseeing</button>

        <!-- Itinerary -->

        <h4>Itinerary</h4>

        <div \*ngFor="let item of newPackage.itinerary; let i=index">

          <label>Day Number: </label>

          <input type="number" placeholder="Day #" [(ngModel)]="item.dayNumber" [name]="'day'+i" placeholder="Day #" required><br>

          <label>Activity Title: </label>

          <input type="text" placeholder="Activity Title" [(ngModel)]="item.activityTitle" [name]="'activityTitle'+i" placeholder="Activity Description" required><br>

          <label>Activity Description: </label>

          <input type="text" placeholder="Activity Description" [(ngModel)]="item.activityDescription" [name]="'activityDesc'+i" required><br>

          <button type="button" (click)="removeItineraryDay(i)" \*ngIf="newPackage.itinerary.length>1">Remove Day</button>

          <hr>

        </div>

        <button type="button" (click)="addItineraryDay()">+Add Itinerary Day</button>

        <!-- Submit and Cancel -->

        <br />

        <button type="submit">Submit Package</button>

        <button type="button" (click)="cancelForm()">Cancel</button>

      </form>

  </div>

//src\app\edit-package\edit-package.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { EditPackageComponent } from './edit-package.component';

describe('EditPackageComponent', () => {

  let component: EditPackageComponent;

  let fixture: ComponentFixture<EditPackageComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [EditPackageComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(EditPackageComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\edit-package\edit-package.component.ts

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, Router } from '@angular/router';

import { HttpClient } from '@angular/common/http';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { HttpClientModule } from '@angular/common/http';

import { RouterModule } from '@angular/router';

@Component({

  selector: 'app-edit-package',

  standalone: true,

  imports: [CommonModule, FormsModule, HttpClientModule, RouterModule],

  templateUrl: './edit-package.component.html',

  styleUrls: ['./edit-package.component.css']

})

export class EditPackageComponent implements OnInit {

  packageId!: number;

  // packageData: any = {};

  newPackage: any = {

    title:'',

    description:'',

    duration:'',

    price:'',

    maxCapacity:'',

    country:'',

    destination:'',

    tripType:'',

    tripStartDate:'',

    tripEndDate:'',

    active: true,

    offer: { couponCode:'', description:'', discountPercentage: 0, active: false},

    flights: [

      {airline: '', fromCity:'', toCity:'', departureTime:'', arrivalTime:''}

    ],

    hotels: [

      {name:'', city:'', rating:'', nights:'', costPerNight:''}

    ],

    sightseeing: [

      {location:'', description:''}

    ],

    itinerary: [

      {dayNumber:'', activityTitle:'', activityDescription:''}

    ]

  };

  constructor(

    private route: ActivatedRoute,

    private http: HttpClient,

    private router: Router

  ) {}

  ngOnInit(): void {

    // Capture ID from URL

    this.packageId = Number(this.route.snapshot.paramMap.get('id'));

    // Fetch existing package data

    const url=`http://localhost:8080/api/packages/${this.packageId}`;

      this.http.get<any>(url).subscribe({

        next: (res) => {

          this.newPackage = res.data;

        },

        error: (err) => {

          console.error('Failed to fetch package',err);

          alert('Error loading package data.');

        }

      });

  }

  submitPackage(){

    const url=`http://localhost:8080/api/packages/${this.packageId}`;

      this.http.put(url, this.newPackage).subscribe({

        next: () => {

          alert('Package updated successfully!');

          this.router.navigate(['/agent-dashboard']);

        },

        error: (err) =>

        { console.error('Failed to update package',err);

          alert('Error updating package');

        }

      });

  }

  addItineraryDay(){

    const nextDay = this.newPackage.itinerary.length+1;

    this.newPackage.itinerary.push({

      dayNumber: nextDay,

      activityTitle: '',

      activityDescription: ''

    });

  }

  removeItineraryDay(index:number): void{

    this.newPackage.itinerary.splice(index,1);

    this.newPackage.itinerary.forEach((item: any, i: number) =>{

      item.dayNumber = i+1;

    });

  }

  // for sight seeing

  addSightseeing(): void {

    this.newPackage.sightseeing.push({location:'', description:''});

  }

  removeSightseeing(index: number): void {

    this.newPackage.sightseeing.splice(index,1);

  }

  cancelForm(){

    this.router.navigate(['/agent-dashboard']);

  }

}

//src\app\forgot-password\forgot-password.component.css

.forgot-password-container {

    width: 300px;

    margin: 100px auto;

    padding: 20px;

    background-color: #f8f9fa;

    text-align: center;

    border-radius: 8px;

    box-shadow: 0px 0px 10px #ccc;

  }

  input {

    width: 100%;

    padding: 8px;

    margin-top: 8px;

    margin-bottom: 12px;

    border: 1px solid #ccc;

    border-radius: 4px;

  }

  button {

    padding: 10px 20px;

    background-color: #1f8ef1;

    color: white;

    border: none;

    border-radius: 4px;

    cursor: pointer;

  }

  .forgot-password-link {

    display: block;

    margin-top: 10px;

    color: #007bff;

    text-decoration: none;

    font-size: 14px;

  }

  .forgot-password-link:hover {

    text-decoration: underline;

  }

//src\app\forgot-password\forgot-password.component.html

<div class="forgot-password-container">

  <h2>Forgot Password</h2>

  <label>Email:

    <input type="email" [(ngModel)]="email" required />

  </label>

  <button (click)="sendPassword()">Send Password</button>

</div>

//src\app\forgot-password\forgot-password.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { ForgotPasswordComponent } from './forgot-password.component';

describe('ForgotPasswordComponent', () => {

  let component: ForgotPasswordComponent;

  let fixture: ComponentFixture<ForgotPasswordComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [ForgotPasswordComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(ForgotPasswordComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\forgot-password\forgot-password.component.ts

import { Component } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Router } from '@angular/router';

import { ToastrService } from 'ngx-toastr';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

@Component({

  selector: 'app-forgot-password',

  standalone: true,

  imports: [CommonModule, FormsModule],

  templateUrl: './forgot-password.component.html',

  styleUrl: './forgot-password.component.css'

})

export class ForgotPasswordComponent {

  email: string = '';

  constructor(

    private http: HttpClient,

    private router: Router,

    private toastr: ToastrService

  ) {}

  // Called when user submits the form

  sendPassword(): void {

this.http.post('http://localhost:8080/api/users/forgot-password', { email: this.email })

      .subscribe({

        next: () => {

          this.toastr.success('Password sent to your email!');

          this.router.navigate(['/login']);

        },

        error: () => {

          this.toastr.error('Email not found!');

        }

});

  }

}

//src\app\landing\landing.component.css

.landing-container {

    display: flex;

    flex-direction: column;

    align-items: center;

    justify-content: center;

    padding: 50px;

    height: 100vh;

    background-color: #f0f8ff;

  }

  h1 {

    color: #333;

    font-size: 36px;

  }

  p {

    color: #666;

    font-size: 18px;

  }

  .landing-buttons {

    margin-top: 30px;

  }

  button {

    background-color: #007bff;

    color: white;

    border: none;

    margin: 0 10px;

    padding: 12px 24px;

    font-size: 16px;

    cursor: pointer;

    border-radius: 8px;

    transition: background-color 0.3s ease;

  }

  button:hover {

    background-color: #0056b3;

  }

//src\app\landing\landing.component.html

<div class="landing-container">

    <h1>Welcome to Travel Booking System</h1>

    <p>Book your next adventure or manage your travel packages easily.</p>

    <div class="landing-buttons">

      <a  routerLink="/register"><button>Register</button></a>

      <a  routerLink="/login"><button>Login</button></a>

    </div>

  </div>

//src\app\landing\landing.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { LandingComponent } from './landing.component';

describe('LandingComponent', () => {

  let component: LandingComponent;

  let fixture: ComponentFixture<LandingComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [LandingComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(LandingComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\landing\landing.component.ts

import { Component } from '@angular/core';

import { RouterLink } from '@angular/router';

@Component({

  selector: 'app-landing',

  standalone: true,

  imports: [RouterLink], // Enable routerLink directive in template

  templateUrl: './landing.component.html',

  styleUrl: './landing.component.css'

})

export class LandingComponent {}

//src\app\login\login.component.css

.login-container {

    max-width: 400px;

    margin: 50px auto;

    padding: 20px 30px;

    border: 1px solid #ccc;

    border-radius: 8px;

    box-shadow: 0 4px 8px rgba(0,0,0,0.1);

    background-color: #fff;

    font-family: 'Segoe UI', sans-serif;

  }

  .login-container h2 {

    text-align: center;

    color: #2c3e50;

    margin-bottom: 20px;

  }

  .login-container label {

    display: block;

    margin-top: 15px;

    font-weight: 600;

  }

  .login-container input {

    width: 100%;

    padding: 8px 10px;

    margin-top: 5px;

    border: 1px solid #aaa;

    border-radius: 4px;

    outline-color: #007bff;

  }

  .login-container button {

    width: 100%;

    padding: 10px;

    background-color: #007bff;

    color: #fff;

    font-weight: bold;

    border: none;

    margin-top: 20px;

    border-radius: 4px;

    cursor: pointer;

  }

  .login-container button:hover {

    background-color: #0056b3;

  }

  .forgot-password-wrapper {

    margin-top: 12px;

    text-align: center;

  }

  .forgot-password-link {

    color: #007bff;

    text-decoration: underline;

    cursor: pointer;

  }

//src\app\login\login.component.html

<div class="login-container">

    <h2>Login</h2>

    <form (ngSubmit)="login()">

      <label>Email:</label>

  <input type="email" [(ngModel)]="credentials.email" name="email" required />

      <label>Password:</label>

      <input type="password" [(ngModel)]="credentials.password" name="password" required />

      <button type="submit">Login</button>

    </form>

//src\app\login\login.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { LoginComponent } from './login.component';

describe('LoginComponent', () => {

  let component: LoginComponent;

  let fixture: ComponentFixture<LoginComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [LoginComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(LoginComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\login\login.component.ts

interface LoginResponse {

  token: string;

  role: string;

  id: number;

}

import { FormsModule } from '@angular/forms';

import { Component } from '@angular/core';

import { Router } from '@angular/router';

import { RouterModule } from '@angular/router';

import { HttpClient } from '@angular/common/http';

import { ToastrService } from 'ngx-toastr';

@Component({

  selector: 'app-login',

  standalone: true,

  imports: [FormsModule, RouterModule],

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.css']

})

export class LoginComponent {

  credentials = {

    email: '',

    password: ''

  };

  constructor(

    private http: HttpClient,

    private router: Router,

    private toastr: ToastrService

  ) {}

  login() {

this.http.post<LoginResponse>('http://localhost:8080/api/auth/login', this.credentials)

      .subscribe({

        next: (res) => {

          this.toastr.success('✅ Login successful');

          const role = res.role;

          const token = res.token;

          const id = res.id;

          // Save token to localStorage

          localStorage.setItem('token', token);

          localStorage.setItem('role', role);

          localStorage.setItem('userId', id.toString());

          // Redirect based on role

          if (role === 'ADMIN') {

            this.router.navigate(['/admin-dashboard']);

          } else if (role === 'AGENT') {

            this.router.navigate(['/agent-dashboard']);

          } else if (role === 'CUSTOMER') {

            this.router.navigate(['/customer-dashboard']);

          }

        },

        error: () => {

          this.toastr.error('❌ Invalid credentials');

        }

      });

  }

  navigateToForgotPassword(event: Event) {

    event.preventDefault();

    this.router.navigate(['/forgot-password']);

  }

}

   <!-- Outside the <form> -->

<div class="forgot-password-wrapper">

  <a [routerLink]="['/forgot-password']" class="forgot-password-link">Forgot Password?</a>

</div>

  </div>

//src\app\login\login.component.css

.login-container {

    max-width: 400px;

    margin: 50px auto;

    padding: 20px 30px;

    border: 1px solid #ccc;

    border-radius: 8px;

    box-shadow: 0 4px 8px rgba(0,0,0,0.1);

    background-color: #fff;

    font-family: 'Segoe UI', sans-serif;

  }

  .login-container h2 {

    text-align: center;

    color: #2c3e50;

    margin-bottom: 20px;

  }

  .login-container label {

    display: block;

    margin-top: 15px;

    font-weight: 600;

  }

  .login-container input {

    width: 100%;

    padding: 8px 10px;

    margin-top: 5px;

    border: 1px solid #aaa;

    border-radius: 4px;

    outline-color: #007bff;

  }

  .login-container button {

    width: 100%;

    padding: 10px;

    background-color: #007bff;

    color: #fff;

    font-weight: bold;

    border: none;

    margin-top: 20px;

    border-radius: 4px;

    cursor: pointer;

  }

  .login-container button:hover {

    background-color: #0056b3;

  }

  .forgot-password-wrapper {

    margin-top: 12px;

    text-align: center;

  }

  .forgot-password-link {

    color: #007bff;

    text-decoration: underline;

    cursor: pointer;

  }

//src\app\login\login.component.html

<div class="login-container">

    <h2>Login</h2>

    <form (ngSubmit)="login()">

      <label>Email:</label>

  <input type="email" [(ngModel)]="credentials.email" name="email" required />

      <label>Password:</label>

      <input type="password" [(ngModel)]="credentials.password" name="password" required />

      <button type="submit">Login</button>

    </form>

   <!-- Outside the <form> -->

<div class="forgot-password-wrapper">

  <a [routerLink]="['/forgot-password']" class="forgot-password-link">Forgot Password?</a>

</div>

  </div>

//src\app\login\login.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { LoginComponent } from './login.component';

describe('LoginComponent', () => {

  let component: LoginComponent;

  let fixture: ComponentFixture<LoginComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [LoginComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(LoginComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\login\login.component.ts

interface LoginResponse {

  token: string;

  role: string;

  id: number;

}

import { FormsModule } from '@angular/forms';

import { Component } from '@angular/core';

import { Router } from '@angular/router';

import { RouterModule } from '@angular/router';

import { HttpClient } from '@angular/common/http';

import { ToastrService } from 'ngx-toastr';

@Component({

  selector: 'app-login',

  standalone: true,

  imports: [FormsModule, RouterModule],

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.css']

})

export class LoginComponent {

  credentials = {

    email: '',

    password: ''

  };

  constructor(

    private http: HttpClient,

    private router: Router,

    private toastr: ToastrService

  ) {}

  login() {

this.http.post<LoginResponse>('http://localhost:8080/api/auth/login', this.credentials)

      .subscribe({

        next: (res) => {

          this.toastr.success('✅ Login successful');

          const role = res.role;

          const token = res.token;

          const id = res.id;

          // Save token to localStorage

          localStorage.setItem('token', token);

          localStorage.setItem('role', role);

          localStorage.setItem('userId', id.toString());

          // Redirect based on role

          if (role === 'ADMIN') {

            this.router.navigate(['/admin-dashboard']);

          } else if (role === 'AGENT') {

            this.router.navigate(['/agent-dashboard']);

          } else if (role === 'CUSTOMER') {

            this.router.navigate(['/customer-dashboard']);

          }

        },

        error: () => {

          this.toastr.error('❌ Invalid credentials');

        }

      });

  }

  navigateToForgotPassword(event: Event) {

    event.preventDefault();

    this.router.navigate(['/forgot-password']);

  }

}

//src\app\my-profile\my-profile.component.css

/\* Center the profile content \*/

h2 {

    text-align: center;

    margin-top: 40px;

    font-size: 26px;

    font-weight: bold;

    color: #333;

  }

  p {

    text-align: center;

    font-size: 16px;

    margin: 10px 0;

    color: #444;

  }

  /\* Button container \*/

  .profile-buttons {

    display: flex;

    justify-content: center;

    gap: 15px;

    margin-top: 30px;

  }

  /\* Style the buttons \*/

  .profile-buttons button {

    padding: 10px 20px;

    font-size: 14px;

    font-weight: bold;

    border: none;

    border-radius: 6px;

    cursor: pointer;

    transition: background-color 0.3s ease;

  }

  /\* Different colors for each button \*/

  .profile-buttons button:nth-child(1) {

    background-color: #007bff;

    color: white;

  }

  .profile-buttons button:nth-child(2) {

    background-color: #dc3545;

    color: white;

  }

  .profile-buttons button:nth-child(3) {

    background-color: #6c757d;

    color: white;

  }

  /\* Hover effects \*/

  .profile-buttons button:hover {

    opacity: 0.9;

  }

//src\app\my-profile\my-profile.component.html

<div class="profile-card">

    <h2>My Profile</h2>

    <p><strong>Name: </strong>{{ user.name }}</p>

    <p><strong>Email: </strong>{{ user.email }}</p>

    <p><strong>Contact Number:</strong> {{ user.contactNumber }}</p>

    <p><strong>Role:</strong> {{ user.role }}</p>

    <div class="profile-buttons">

      <button (click)="goToUpdate()">Update Profile</button>

      <button (click)="deleteProfile()">Delete Profile</button>

      <button (click)="logout()">Logout</button>

    </div>

  </div>

//src\app\my-profile\my-profile.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { MyProfileComponent } from './my-profile.component';

describe('MyProfileComponent', () => {

  let component: MyProfileComponent;

  let fixture: ComponentFixture<MyProfileComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [MyProfileComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(MyProfileComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\my-profile\my-profile.component.ts

import { Component, OnInit } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { HttpClient } from '@angular/common/http';

import { Router } from '@angular/router';

import { ToastrService } from 'ngx-toastr';

@Component({

  standalone: true,

  selector: 'app-my-profile',

  imports: [FormsModule],

  templateUrl: './my-profile.component.html',

  styleUrls: ['./my-profile.component.css']

})

export class MyProfileComponent implements OnInit {

  user: any = {};

  constructor(private http: HttpClient, private router: Router, private toastr: ToastrService) {}

  ngOnInit(): void {

    this.loadProfile();

  }

  loadProfile() {

    const token = localStorage.getItem('token');

this.http.get('http://localhost:8080/api/users/myprofile', {

      headers: { Authorization: `Bearer ${token}` }

    }).subscribe(res => this.user = res);

  }

  updateProfile() {

    const token = localStorage.getItem('token');

this.http.put('http://localhost:8080/api/users/myprofile', this.user, {

      headers: { Authorization: `Bearer ${token}` }

    }).subscribe({

      next: () => this.toastr.success('Profile updated'),

      error: () => this.toastr.error('Update failed')

    });

  }

  deleteProfile() {

    const token = localStorage.getItem('token');

this.http.delete('http://localhost:8080/api/users/myprofile', {

      headers: { Authorization: `Bearer ${token}` }

    }).subscribe({

      next: () => {

        localStorage.clear();

        this.toastr.success('Profile deleted');

        this.router.navigate(['/']);

      },

      error: () => this.toastr.error('Delete failed')

    });

  }

  goToUpdate(){

    this.router.navigate(['/update-profile']);

  }

  logout() {

    localStorage.clear();

    this.toastr.success('Logged out');

    this.router.navigate(['/']);

  }

}

//src\app\view-package\view-package.component.css

//src\app\view-package\view-package.component.html

<div \*ngIf="packageData">

    <h2>{{ packageData.title }}</h2>

    <p><strong>Description:</strong> {{ packageData.description }}</p>

    <p><strong>Duration:</strong> {{ packageData.duration }} days</p>

    <p><strong>Price:</strong> ₹{{ packageData.price }}</p>

    <p><strong>Max Capacity:</strong> {{ packageData.maxCapacity }}</p>

    <p><strong>Country:</strong> {{ packageData.country }}</p>

    <p><strong>Destination:</strong> {{ packageData.destination }}</p>

    <p><strong>Trip Type:</strong> {{ packageData.tripType }}</p>

    <p><strong>Start Date:</strong> {{ packageData.tripStartDate }}</p>

    <p><strong>End Date:</strong> {{ packageData.tripEndDate }}</p>

    <h3>Offer:</h3>

    <p>Coupon Code: {{ packageData.offer?.couponCode }}</p>

    <p>Description: {{ packageData.offer?.description }}</p>

    <p>Discount: {{ packageData.offer?.discountPercentage }}%</p>

    <h3>Flights</h3>

    <ul>

        <li \*ngFor="let f of packageData.flights">

            <strong>{{ f.airline }}</strong><br>

            From: {{ f.fromCity }} → To: {{ f.toCity }}<br>

            Departure: {{ f.departureTime }} | Arrival: {{ f.arrivalTime }}

        </li>

    </ul>

    <h3>Hotels</h3>

    <ul>

        <li \*ngFor="let h of packageData.hotels">

            {{ h.name }} - {{ h.city }} ({{ h.rating }}★)<br>

            Nights: {{ h.nights }}<br>

            Cost per Night: ₹{{ h.costPerNight }}<br>

            Total Cost: ₹{{ h.nights \* h.costPerNight }}

        </li>

    </ul>

    <h3>Sightseeing</h3>

    <ul>

      <li \*ngFor="let s of packageData.sightseeing">

        {{ s.location }} - {{ s.description }}

      </li>

    </ul>

    <h3>Itinerary</h3>

    <ul>

      <li \*ngFor="let i of packageData.itinerary">

        Day {{ i.dayNumber }}: {{ i.activityTitle }} - {{ i.activityDescription }}

      </li>

    </ul>

    <br />

    <button (click)="router.navigate(['/agent-dashboard'])">Back to Dashboard</button>

</div>

//src\app\view-package\view-package.component.spec.ts

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { ViewPackageComponent } from './view-package.component';

describe('ViewPackageComponent', () => {

  let component: ViewPackageComponent;

  let fixture: ComponentFixture<ViewPackageComponent>;

  beforeEach(async () => {

    await TestBed.configureTestingModule({

      imports: [ViewPackageComponent]

    })

    .compileComponents();

    fixture = TestBed.createComponent(ViewPackageComponent);

    component = fixture.componentInstance;

    fixture.detectChanges();

  });

  it('should create', () => {

    expect(component).toBeTruthy();

  });

});

//src\app\view-package\view-package.component.ts

import { Component, OnInit} from '@angular/core';

import { ActivatedRoute, Router } from '@angular/router';

import { HttpClient, HttpClientModule } from '@angular/common/http';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { RouterModule } from '@angular/router';

@Component({

  selector: 'app-view-package',

  standalone: true,

  imports: [CommonModule, FormsModule, HttpClientModule, RouterModule],

  templateUrl: './view-package.component.html',

  styleUrl: './view-package.component.css'

})

export class ViewPackageComponent implements OnInit {

  packageId!: number;

  packageData: any;

  constructor(private route: ActivatedRoute, private http: HttpClient, public router: Router){}

  ngOnInit(): void {

    this.packageId = Number(this.route.snapshot.paramMap.get('id'));

    const url = `http://localhost:8080/api/packages/${this.packageId}`;

    this.http.get<any>(url).subscribe({

      next: (res) =>{

        this.packageData = res.data;

        console.log("Loaded packages:", this.packageData);

      },

      error: (err) =>{

        console.error('Error fetching package', err);

        alert('Failed to load package details');

      }

    });

  }

}

//src\app\app.component.html

<router-outlet></router-outlet>

//src\app\app.component.ts

import { Component } from '@angular/core';

import { RouterOutlet } from '@angular/router';

@Component({

  selector: 'app-root',

  imports: [RouterOutlet],

  templateUrl: './app.component.html',

  styleUrl: './app.component.css'

})

export class AppComponent {

  title = 'travel-booking-system-frontend';

}

//src\app\app.routes.ts

// Import Routes type from Angular Router

import { Routes } from '@angular/router';

import { LoginComponent } from './login/login.component';

import { LandingComponent } from './landing/landing.component';

import { RegisterComponent } from './register/register.component';

import { AdminUserManageComponent } from './admin-user-manage/admin-user-manage.component';

export const routes: Routes = [

  { path: '', component: LandingComponent },

  { path: 'register', component: RegisterComponent },

  { path: 'login', component: LoginComponent },

  // Dashboard placeholders

  { path: 'admin-dashboard', loadComponent: () => import('./admin-dashboard/admin-dashboard.component').then(m => m.AdminDashboardComponent) },

  { path: 'agent-dashboard', loadComponent: () => import('./agent-dashboard/agent-dashboard.component').then(m => m.AgentDashboardComponent) },

  { path: 'customer-dashboard', loadComponent: () => import('./customer-dashboard/customer-dashboard.component').then(m => m.CustomerDashboardComponent) },

  { path: 'my-profile', loadComponent: () => import('./my-profile/my-profile.component').then(m => m.MyProfileComponent) },

  { path: 'admin/users', loadComponent: () => import('./admin-user-manage/admin-user-manage.component').then(m => m.AdminUserManageComponent) },

  { path: 'update-profile', loadComponent: () => import('./update-profile/update-profile.component').then(m => m.UpdateProfileComponent) },

  { path: 'admin-user-manage' , loadComponent: () => import('./admin-user-manage/admin-user-manage.component').then(m =>AdminUserManageComponent)},

  { path: 'admin/agent-packages/:id', loadComponent: () => import('./admin-agent-packages/admin-agent-packages.component').then(m => m.AdminAgentPackagesComponent)},

  { path: 'forgot-password', loadComponent: () => import('./forgot-password/forgot-password.component').then(m => m.ForgotPasswordComponent)},

  //For package editing/updating in agent dashboard

  { path: 'edit-package/:id', loadComponent:() => import('./edit-package/edit-package.component').then(m => m.EditPackageComponent)},

  //For viewing the details of the package by the agent(retrieve)

  { path: 'view-package/:id', loadComponent:() => import('./view-package/view-package.component').then(m => m.ViewPackageComponent)}

];

//src\app\auth-expired.interceptor.spec.ts

import { TestBed } from '@angular/core/testing';

import { AuthExpiredInterceptor } from './auth-expired.interceptor';

import { Router } from '@angular/router';

import { ToastrService } from 'ngx-toastr';

describe('AuthExpiredInterceptor', () => {

  let interceptor: AuthExpiredInterceptor;

  beforeEach(() => {

    TestBed.configureTestingModule({

      providers: [

        AuthExpiredInterceptor,

        { provide: Router, useValue: { navigate: jasmine.createSpy() } },

        { provide: ToastrService, useValue: { warning: jasmine.createSpy() } }

      ]

    });

    interceptor = TestBed.inject(AuthExpiredInterceptor);

  });

  it('should be created', () => {

    expect(interceptor).toBeTruthy();

  });

});

//src\app\auth-expired.interceptor.ts

import { Injectable } from '@angular/core';

import {

  HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpErrorResponse

} from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

import { Router } from '@angular/router';

import { ToastrService } from 'ngx-toastr';

@Injectable()

export class AuthExpiredInterceptor implements HttpInterceptor {

  constructor(private router: Router, private toastr: ToastrService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    const token = localStorage.getItem('token');

    const authReq = token

      ? req.clone({

          headers: req.headers.set('Authorization', `Bearer ${token}`)

        })

      : req;

    return next.handle(authReq).pipe(

      catchError((error: HttpErrorResponse) => {

        if (error.status === 401 || error.status === 403) {

          this.toastr.warning('Session expired. Please log in again.');

          localStorage.clear();

          this.router.navigate(['/login']);

        }

        return throwError(() => error);

      })

    );

  }

}

//src\index.html

<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>TravelBookingSystemFrontend</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

  <app-root></app-root>

</body>

</html>

//src\main.ts

import { bootstrapApplication } from '@angular/platform-browser';

import { AppComponent } from './app/app.component';

// Required to provide routing

import { provideRouter } from '@angular/router';

import { routes } from './app/app.routes';

// Import standard modules for forms and HTTP

import { importProvidersFrom } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { HTTP\_INTERCEPTORS, HttpClientModule, withInterceptorsFromDi } from '@angular/common/http';

// ✅ Toast + Animations

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { provideToastr } from 'ngx-toastr';  // ✅ Use provideToastr here

//for interceptor for jwt token expiration

import { provideHttpClient } from '@angular/common/http';

import { AuthExpiredInterceptor } from './app/auth-expired.interceptor';

bootstrapApplication(AppComponent, {

  providers: [

    provideHttpClient(withInterceptorsFromDi()),

    provideRouter(routes),

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthExpiredInterceptor,

      multi: true

    },

    // ✅ Correct usage of importProvidersFrom for standard modules

    importProvidersFrom(

      FormsModule,

      HttpClientModule,

      BrowserAnimationsModule

    ),

    // ✅ provideToastr must be outside importProvidersFrom

    provideToastr({

      timeOut: 3000,

      positionClass: 'toast-bottom-center',

      preventDuplicates: true

    })

  ]

});

//src\app\review-dashboard\review-dashboard.component.css

/\* review-dashboard.component.css \*/

/\* Importing Google Font "Inter" for consistent typography \*/

@import url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700&display=swap');

/\* Apply Inter font to the component's host element or deeper if needed \*/

/\* :host is generally used for component-specific global styles, or apply to body/html if it's the root component \*/

body {

  font-family: 'Inter', sans-serif;

}

/\* Base styles for common elements to ensure consistent appearance \*/

input[type="number"],

textarea {

  transition: border-color 0.2s ease-in-out, box-shadow 0.2s ease-in-out;

}

button {

  transition: background-color 0.3s ease-in-out, transform 0.2s ease-in-out, box-shadow 0.2s ease-in-out;

}

/\* Custom focus styles for input and textarea, enhancing visibility \*/

input[type="number"]:focus,

textarea:focus {

  outline: none;

  border-color: #3B82F6; /\* Tailwind blue-500 \*/

  box-shadow: 0 0 0 3px rgba(59, 130, 246, 0.45); /\* Custom focus ring for visibility \*/

}

/\* Specific styling for action buttons for consistent appearance and touch targets \*/

.bg-blue-600,

.bg-yellow-500,

.bg-red-500,

.bg-green-600,

.bg-gray-400 {

  display: inline-flex;

  align-items: center;

  justify-content: center;

  text-align: center;

  white-space: nowrap;

  min-width: 80px;

  cursor: pointer;

}

/\* Media queries for responsiveness \*/

@media (max-width: 640px) {

  .p-6 {

    padding: 1rem;

  }

  .md\:p-8 {

    padding: 1.25rem;

  }

  .text-3xl {

    font-size: 2rem;

  }

  .text-2xl {

    font-size: 1.75rem;

  }

  .flex-col.sm\:flex-row {

    flex-direction: column;

  }

  .space-y-2.sm\:space-y-0 > \*:not(:last-child) {

    margin-bottom: 0.5rem;

  }

  .sm\:space-x-2 > \*:not(:first-child) {

    margin-left: 0;

  }

}

//src\app\review-dashboard\review-dashboard.component.html

<!-- review-dashboard.component.html -->

<div class="min-h-screen bg-gray-100 p-4 font-sans antialiased">

  <div class="max-w-4xl mx-auto bg-white rounded-xl shadow-lg p-6 md:p-8">

    <h2 class="text-3xl md:text-4xl font-extrabold text-gray-800 mb-6 text-center tracking-tight">Travel Package Reviews</h2>

    <!-- Review Submission Section -->

    <div \*ngIf="canReview && !hasReviewed" class="mb-8 p-6 bg-blue-50 rounded-lg shadow-md border border-blue-200">

      <h3 class="text-2xl font-semibold text-blue-800 mb-4">Submit Your Review</h3>

      <div class="mb-4">

        <label for="rating" class="block text-gray-700 text-sm font-medium mb-2">Rating (1-5):</label>

        <input

          type="number"

          id="rating"

          [(ngModel)]="newReview.rating"

          min="1"

          max="5"

          class="w-full p-3 border border-blue-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 transition duration-200 ease-in-out"

          placeholder="Enter rating (1-5)"

        />

      </div>

      <div class="mb-4">

        <label for="comment" class="block text-gray-700 text-sm font-medium mb-2">Your Comment:</label>

        <textarea

          id="comment"

          [(ngModel)]="newReview.comment"

          placeholder="Share your experience with this package..."

          rows="4"

          class="w-full p-3 border border-blue-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 transition duration-200 ease-in-out resize-y"

        ></textarea>

      </div>

      <button

        (click)="submitReview()"

        class="w-full bg-blue-600 text-white font-bold py-3 px-4 rounded-lg shadow-md hover:bg-blue-700 transition duration-300 ease-in-out focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 transform hover:scale-105"

      >

        Submit Review

      </button>

    </div>

    <!-- Already Reviewed Message -->

    <div \*ngIf="hasReviewed && !editingReviewId" class="mb-8 p-4 bg-yellow-100 text-yellow-800 rounded-lg shadow-md text-center font-medium border border-yellow-200">

      You have already submitted a review for this package.

    </div>

    <!-- Cannot Review Message (if not completed booking or not eligible) -->

    <div \*ngIf="!canReview && !hasReviewed" class="mb-8 p-4 bg-red-100 text-red-800 rounded-lg shadow-md text-center font-medium border border-red-200">

      You can only review this package after completing the booking.

    </div>

    <!-- User ID Not Found Message -->

    <div \*ngIf="userId === 0" class="mb-8 p-4 bg-red-100 text-red-800 rounded-lg shadow-md text-center font-medium border border-red-200">

      Please ensure you are logged in to submit or view reviews. User ID not found.

    </div>

    <!-- Reviews List Section -->

    <div \*ngIf="reviews.length > 0" class="bg-gray-50 rounded-xl shadow-md p-6 md:p-8">

      <h3 class="text-2xl font-semibold text-gray-800 mb-6">All Reviews ({{ reviews.length }})</h3>

      <div \*ngFor="let review of reviews" class="border-b border-gray-200 pb-6 mb-6 last:border-b-0 last:pb-0 last:mb-0">

        <!-- Display Review -->

        <div \*ngIf="editingReviewId !== review.reviewId">

          <p class="text-lg text-gray-900 mb-2">

            <strong class="text-blue-700">User {{ review.userId }}</strong>: {{ review.comment }} (Rating:

            <span class="font-bold text-yellow-600">{{ review.rating }}</span>/5)

          </p>

          <p class="text-sm text-gray-500 mb-4">Reviewed on: {{ review.timestamp | date:'short' }}</p>

          <!-- Agent Response -->

          <div \*ngIf="review.agentResponse" class="ml-4 p-3 bg-green-50 rounded-lg border border-green-200">

            <p class="text-base text-green-800">

              <strong class="text-green-700">Agent:</strong> {{ review.agentResponse.responseMessage }}

            </p>

            <p class="text-xs text-gray-500 mt-1">Responded on: {{ review.agentResponse.responseTime | date:'short' }}</p>

          </div>

          <!-- Action Buttons for Review Owner -->

          <div \*ngIf="review.userId === userId" class="flex flex-col sm:flex-row space-y-2 sm:space-y-0 sm:space-x-2 mt-4">

            <button

              (click)="enableEdit(review)"

              class="bg-yellow-500 text-white font-semibold py-2 px-4 rounded-lg shadow-sm hover:bg-yellow-600 transition duration-300 transform hover:scale-105"

            >

              Edit

            </button>

            <button

              (click)="deleteReview(review.reviewId!)"

              class="bg-red-500 text-white font-semibold py-2 px-4 rounded-lg shadow-sm hover:bg-red-600 transition duration-300 transform hover:scale-105"

            >

              Delete

            </button>

          </div>

        </div>

        <!-- Edit Review Form -->

        <div \*ngIf="editingReviewId === review.reviewId" class="p-4 bg-orange-50 rounded-lg shadow-inner border border-orange-200">

          <h4 class="text-xl font-semibold text-orange-800 mb-3">Edit Your Review</h4>

          <div class="mb-3">

            <label for="editRating" class="block text-gray-700 text-sm font-medium mb-1">Rating (1-5):</label>

            <input

              type="number"

              id="editRating"

              [(ngModel)]="newReview.rating"

              min="1"

              max="5"

              class="w-full p-2 border border-orange-300 rounded-md focus:outline-none focus:ring-2 focus:ring-orange-500 transition duration-200 ease-in-out"

            />

          </div>

          <div class="mb-3">

            <label for="editComment" class="block text-gray-700 text-sm font-medium mb-1">Comment:</label>

            <textarea

              id="editComment"

              [(ngModel)]="newReview.comment"

              rows="3"

              class="w-full p-2 border border-orange-300 rounded-md focus:outline-none focus:ring-2 focus:ring-orange-500 transition duration-200 ease-in-out resize-y"

            ></textarea>

          </div>

          <div class="flex flex-col sm:flex-row space-y-2 sm:space-y-0 sm:space-x-2">

            <button

              (click)="updateReview()"

              class="flex-1 bg-green-600 text-white font-bold py-2 px-4 rounded-lg shadow-md hover:bg-green-700 transition duration-300 transform hover:scale-105"

            >

              Save Changes

            </button>

            <button

              (click)="cancelEdit()"

              class="flex-1 bg-gray-400 text-white font-bold py-2 px-4 rounded-lg shadow-md hover:bg-gray-500 transition duration-300 transform hover:scale-105"

            >

              Cancel

            </button>

          </div>

        </div>

      </div>

    </div>

    <!-- No Reviews Message -->

    <div \*ngIf="reviews.length === 0 && userId !== 0" class="p-6 bg-indigo-50 text-indigo-800 rounded-lg shadow-md text-center font-medium border border-indigo-200">

      No reviews yet for this package. Be the first to share your experience!

    </div>

  </div>

</div>

//src\app\review-dashboard\review-dashboard.component.ts

// review-dashboard.component.ts

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute } from '@angular/router';

import { HttpClient, HttpErrorResponse } from '@angular/common/http';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { Observable, of } from 'rxjs';

import { catchError, map } from 'rxjs/operators';

// Define the structure for a Review as expected from your backend

interface Review {

  reviewId?: number;

  userId: number;

  packageId: number;

  rating: number;

  comment: string;

  timestamp?: string; // ISO string date

  agentResponse?: { // Optional agent response

    responseMessage: string;

    responseTime: string; // ISO string date

  };

}

@Component({

  standalone: true,

  imports: [CommonModule, FormsModule],

  selector: 'app-review-dashboard',

  templateUrl: './review-dashboard.component.html',

  styleUrl: './review-dashboard.component.css' // <-- This line now points to the separate CSS file

})

export class ReviewDashboardComponent implements OnInit {

  packageId!: number;

  userId: number = Number(sessionStorage.getItem('userId')) || 0;

  reviews: Review[] = [];

  canReview: boolean = false;

  hasReviewed: boolean = false;

  newReview: { rating: number; comment: string } = { rating: 0, comment: '' };

  editingReviewId: number | null = null;

  private readonly REVIEWS\_API\_BASE\_URL = 'http://localhost:8084/api/reviews';

  private readonly BOOKINGS\_API\_BASE\_URL = 'http://localhost:8082/api/bookings';

  constructor(private route: ActivatedRoute, private http: HttpClient) {}

  ngOnInit(): void {

    this.packageId = Number(this.route.snapshot.paramMap.get('packageId'));

    if (this.userId === 0) {

      console.warn('User ID not found in session storage. Please log in.');

      return;

    }

    this.loadReviews();

    this.checkUserEligibility();

  }

  loadReviews(): void {

    this.http.get<Review[]>(`${this.REVIEWS\_API\_BASE\_URL}/${this.packageId}`)

      .pipe(

        catchError((error: HttpErrorResponse) => {

          console.error(`Error loading reviews for package ${this.packageId}:`, error);

          return of([]);

        })

      )

      .subscribe(data => {

        this.reviews = data;

        console.log('Reviews loaded:', this.reviews);

      });

  }

  checkUserEligibility(): void {

    this.http.get<boolean>(`${this.BOOKINGS\_API\_BASE\_URL}/user/${this.userId}/package/${this.packageId}/completed`)

      .pipe(

        catchError((error: HttpErrorResponse) => {

          console.error('Error checking booking completion:', error);

          return of(false);

        })

      )

      .subscribe(hasCompleted => {

        this.canReview = hasCompleted;

        if (hasCompleted) {

          this.http.get<boolean>(`${this.REVIEWS\_API\_BASE\_URL}/exists/${this.userId}/${this.packageId}`)

            .pipe(

              catchError((error: HttpErrorResponse) => {

                console.error('Error checking if user already reviewed:', error);

                return of(false);

              })

            )

            .subscribe(alreadyReviewed => {

              this.hasReviewed = alreadyReviewed;

              console.log(`User ${this.userId} completed booking: ${this.canReview}, already reviewed: ${this.hasReviewed}`);

            });

        } else {

          console.log(`User ${this.userId} has NOT completed booking for package ${this.packageId}`);

        }

      });

  }

  submitReview(): void {

    if (this.newReview.rating < 1 || this.newReview.rating > 5) {

      console.error('Rating must be between 1 and 5.');

      return;

    }

    if (!this.newReview.comment.trim()) {

      console.error('Comment cannot be empty.');

      return;

    }

    const reviewData: Review = {

      userId: this.userId,

      packageId: this.packageId,

      rating: this.newReview.rating,

      comment: this.newReview.comment

    };

    this.http.post<Review>(this.REVIEWS\_API\_BASE\_URL, reviewData)

      .pipe(

        catchError((error: HttpErrorResponse) => {

          console.error('Error submitting review:', error);

          alert(`Failed to submit review: ${error.error.message || error.message}`);

          return of(null);

        })

      )

      .subscribe(response => {

        if (response) {

          console.log('Review submitted successfully:', response);

          this.loadReviews();

          this.checkUserEligibility();

          this.resetReviewForm();

        }

      });

  }

  enableEdit(review: Review): void {

    this.editingReviewId = review.reviewId!;

    this.newReview = {

      rating: review.rating,

      comment: review.comment

    };

  }

  updateReview(): void {

    if (this.editingReviewId !== null) {

      const updatedReviewData: Review = {

        userId: this.userId,

        packageId: this.packageId,

        rating: this.newReview.rating,

        comment: this.newReview.comment

      };

      this.http.put<Review>(`${this.REVIEWS\_API\_BASE\_URL}/${this.editingReviewId}`, updatedReviewData)

        .pipe(

          catchError((error: HttpErrorResponse) => {

            console.error('Error updating review:', error);

            alert(`Failed to update review: ${error.error.message || error.message}`);

            return of(null);

          })

        )

        .subscribe(response => {

          if (response) {

            console.log('Review updated successfully:', response);

            this.loadReviews();

            this.cancelEdit();

          }

        });

    }

  }

  deleteReview(reviewId: number): void {

    this.http.delete(`${this.REVIEWS\_API\_BASE\_URL}/${reviewId}`)

      .pipe(

        catchError((error: HttpErrorResponse) => {

          console.error('Error deleting review:', error);

          alert(`Failed to delete review: ${error.error.message || error.message}`);

          return of(null);

        })

      )

      .subscribe(response => {

        if (response) {

          console.log('Review deleted successfully.');

          this.loadReviews();

          this.checkUserEligibility();

        }

      });

  }

  cancelEdit(): void {

    this.editingReviewId = null;

    this.resetReviewForm();

  }

  private resetReviewForm(): void {

    this.newReview = {

      rating: 0,

      comment: ''

    };

  }

}

//src\app\services\review.service.ts

import { HttpClient } from '@angular/common/http';

import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })

export class ReviewService {

  private baseUrl = 'http://localhost:8084/api/reviews';

  constructor(private http: HttpClient) {}

  getReviewsByPackage(packageId: number) {

    return this.http.get<any[]>(`${this.baseUrl}/${packageId}`);

  }

  postReview(review: any) {

    return this.http.post(`${this.baseUrl}`, review);

  }

  updateReview(reviewId: number, updatedReview: any) {

    return this.http.put(`${this.baseUrl}/${reviewId}`, updatedReview);

  }

  deleteReview(reviewId: number) {

    return this.http.delete(`${this.baseUrl}/${reviewId}`);

  }

  hasCompletedBooking(userId: number, packageId: number) {

    return this.http.get(`http://localhost:8086/api/bookings/user/${userId}/package/${packageId}/completed`);

  }

  hasAlreadyReviewed(userId: number, packageId: number) {

    return this.http.get<boolean>(`${this.baseUrl}/exists/${userId}/${packageId}`);

  }

}