**Package module**

package com.example;

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 //Enables Eureka client functionality

@EnableFeignClients

public class TravelPackageManagementApplication {

public static void main(String[] args) {

SpringApplication.run(TravelPackageManagementApplication.class, args);

}

}

**package** com.example.client;

**import** org.springframework.cloud.openfeign.FeignClient;

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

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

**import** com.example.dto.UserDTO;

@FeignClient(name="USERMANAGEMENT-SERVICE")

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

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

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

}

**package** com.example.controller;

**import** com.example.model.TravelPackage;

**import** com.example.response.ApiResponse;

**import** com.example.service.TravelPackageService;

**import** jakarta.validation.Valid;

**import** org.springframework.http.ResponseEntity;

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

**import** com.example.dto.\*;

**import** java.util.List;

@RestController

@RequestMapping("/api/packages")

**public** **class** TravelPackageController {

**private** **final** TravelPackageService service;

**public** TravelPackageController(TravelPackageService service) {

**this**.service = service;

}

@GetMapping(produces = "application/json")

**public** ResponseEntity<ApiResponse> getAllPackages() {

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

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

}

@GetMapping(value = "/{id}", produces = "application/json")

**public** ResponseEntity<ApiResponse> getById(@PathVariable Long id) {

TravelPackage travelPackage = service.getPackageById(id);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Package found", travelPackage));

}

@GetMapping("/agent/{agentId}")

**public** ResponseEntity<ApiResponse> getByAgentId(@PathVariable Long agentId){

List<TravelPackage> packages = service.getPackageByAgentId(agentId);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Package by agent retrieve", packages));

}

@PostMapping

**public** ResponseEntity<ApiResponse> create(@Valid @RequestBody TravelPackage travelPackage) {

TravelPackage created = service.createPackage(travelPackage);

**return** ResponseEntity.*ok*( **new** ApiResponse(**true**, "Package created successfully", created));

}

@PutMapping("/{id}")

**public** ResponseEntity<ApiResponse> update(@PathVariable Long id, @Valid @RequestBody TravelPackage travelPackage) {

TravelPackage updated = service.updatePackage(id, travelPackage);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Package updated successfully", updated));

}

@DeleteMapping("/{id}")

**public** ResponseEntity<ApiResponse> delete(@PathVariable Long id) {

service.deletePackage(id);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Package deleted successfully", **null**));

}

//for DTO implementation

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

**public** ResponseEntity<?> getPackageWithAllDetails(@PathVariable Long id){

TravelPackage pkg = service.getPackageById(id);

TravelPackageDTO dto = service.convertToDTO(pkg);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Package with full details", dto));

}

//for booking dto purpose

@GetMapping("/search/title/{title}")

**public** ResponseEntity<ApiResponse> getByTitle(@PathVariable String title) {

List<TravelPackage> packages = service.searchByTitle(title);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Packages found by title", packages));

}

@GetMapping("/search/price/{maxPrice}")

**public** ResponseEntity<ApiResponse> getByPrice(@PathVariable **double** maxPrice) {

List<TravelPackage> packages = service.searchByPrice(maxPrice);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Packages under price", packages));

}

@GetMapping("/search/offer/{couponCode}")

**public** ResponseEntity<ApiResponse> getByOffer(@PathVariable String couponCode) {

List<TravelPackage> packages = service.searchByOffer(couponCode);

**return** ResponseEntity.*ok*(**new** ApiResponse(**true**, "Packages with offer", packages));

}

}

**package** com.example.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.example.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.example.dto;

**import** lombok.Data;

@Data

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

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

**private** String activityTitle;

**private** String activityDescription;

}

**package** com.example.dto;

**import** lombok.Data;

@Data

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

**private** String couponCode;

**private** String description;

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

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

}

**package** com.example.dto;

**import** lombok.Data;

@Data

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

**private** String location;

**private** String description;

}

**package** com.example.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** OfferDTO offer;

}

**package** com.example.dto;

**import** lombok.Data;

@Data

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

**private** Long id;

**private** String name;

**private** String email;

**private** String role;

}

package com.example.exception;

import com.example.exception.ResourceNotFoundException;

import com.example.response.ApiResponse;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.FieldError;

import org.springframework.web.bind.MethodArgumentNotValidException;

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

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

import java.util.HashMap;

import java.util.Map;

@RestControllerAdvice

public class GlobalExceptionHandler1 {

// ✅ Handle resource not found (like 404)

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<com.example.response.ApiResponse> handleResourceNotFound(ResourceNotFoundException ex) {

com.example.response.ApiResponse response = new com.example.response.ApiResponse(false, ex.getMessage(), null);

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

}

// ✅ Handle validation errors (like @Valid)

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<ApiResponse> handleValidationErrors(MethodArgumentNotValidException ex) {

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

for (FieldError error : ex.getBindingResult().getFieldErrors()) {

validationErrors.put(error.getField(), error.getDefaultMessage());

}

ApiResponse response = new ApiResponse(false, "Validation failed", validationErrors);

return ResponseEntity.badRequest().body(response);

}

// ✅ Handle all other unexpected errors

@ExceptionHandler(Exception.class)

public ResponseEntity<ApiResponse> handleGenericException(Exception ex) {

ApiResponse response = new ApiResponse(false, "An unexpected error occurred: " + ex.getMessage(), null);

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(response);

}

}

**package** com.example.exception;

/\*\*

\* Exception for handling resource not found scenarios.

\*/

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

**public** ResourceNotFoundException(String message) {

**super**(message);

}

}

package com.example.model;

import jakarta.persistence.\*;

import jakarta.validation.constraints.NotBlank;

import lombok.\*;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Flight {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotBlank(message = "Airline name is required")

private String airline;

@NotBlank(message = "From city is required")

private String fromCity;

@NotBlank(message = "To city is required")

private String toCity;

@NotBlank(message = "Departure time is required")

private String departureTime;

@NotBlank(message = "Arrival time is required")

private String arrivalTime;

}

package com.example.model;

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

import lombok.\*;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Hotel {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotBlank(message = "Hotel name is required")

private String name;

@NotBlank(message = "City is required")

private String city;

@DecimalMin(value = "0.0", inclusive = false, message = "Rating must be greater than 0")

private double rating;

@Min(value = 1, message = "Number of nights must be at least 1")

private int nights;

@DecimalMin(value = "0.0", inclusive = false, message = "Cost per night must be greater than 0")

private double costPerNight;

}

package com.example.model;

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

import lombok.\*;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Itinerary {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Min(value = 1, message = "Day number must be at least 1")

private int dayNumber;

@NotBlank(message = "Activity title is required")

private String activityTitle;

@NotBlank(message = "Activity description is required")

private String activityDescription;

}

package com.example.model;

import jakarta.persistence.Column;

import jakarta.persistence.Embeddable;

import jakarta.validation.constraints.\*;

import lombok.\*;

@Embeddable

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Offer {

@NotBlank(message = "Coupon code is required")

private String couponCode;

@NotBlank(message = "Offer description is required")

@Column(name = "offer\_description")

private String description;

@Min(value = 0, message = "Discount must be at least 0")

@Max(value = 100, message = "Discount must not exceed 100")

private int discountPercentage;

private boolean active;

}package com.example.model;

import jakarta.persistence.\*;

import jakarta.validation.constraints.NotBlank;

import lombok.\*;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Sightseeing {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotBlank(message = "Sightseeing location is required")

private String location;

@NotBlank(message = "Sightseeing description is required")

@Column(name = "sightseeing\_description")

private String description;

}

package com.example.model;

import jakarta.persistence.\*;

import jakarta.validation.Valid;

import jakarta.validation.constraints.\*;

import lombok.\*;

import java.time.LocalDate;

import java.util.List;

@Entity

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class TravelPackage {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name="package\_id")

private Long packageId;

@NotNull(message="Agent ID is required")

private Long agentId;

@NotBlank(message = "Title is mandatory")

private String title;

@NotBlank(message = "Description is mandatory")

private String description;

@Min(value = 1, message = "Duration should be at least 1 day")

private int duration;

@DecimalMin(value = "0.0", inclusive = false, message = "Price must be greater than zero")

private double price;

@Min(value=1, message="MAX capacity should be atleast 1")

@Column(name="max\_capacity", nullable=false)

private int maxCapacity;

@NotNull(message="Trip start date is required")

private LocalDate tripStartDate;

@NotNull(message="Trip end date is required")

private LocalDate tripEndDate;

@ElementCollection

@CollectionTable(name="travel\_package\_highlights", joinColumns= @JoinColumn(name="travel\_package\_package\_id"))

@Column(name="highlight")

@NotNull(message = "Highlights cannot be null")

private List<@NotBlank(message = "Highlight item cannot be blank") String> highlights;

@OneToMany(cascade = CascadeType.ALL, orphanRemoval=true)

@JoinColumn(name="travel\_package\_id")

@NotNull(message = "Flights must not be null")

private List<@Valid Flight> flights;

@OneToMany(cascade = CascadeType.ALL, orphanRemoval=true)

@JoinColumn(name="travel\_package\_id")

@NotNull(message = "Hotels must not be null")

private List<@Valid Hotel> hotels;

@OneToMany(cascade = CascadeType.ALL, orphanRemoval=true)

@JoinColumn(name="travel\_package\_id")

@NotNull(message = "Sightseeing must not be null")

private List<@Valid Sightseeing> sightseeing;

@OneToMany(cascade = CascadeType.ALL, orphanRemoval=true)

@JoinColumn(name="travel\_package\_id")

@NotNull(message = "Itinerary must not be null")

private List<@Valid Itinerary> itinerary;

@Embedded

@Valid

private Offer offer;

}

**package** com.example.repository;

**import** java.util.List;

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

**import** com.example.model.TravelPackage;

/\*\*

\* Repository for TravelPackage entity.

\*/

**public** **interface** TravelPackageRepository **extends** JpaRepository<TravelPackage, Long> {

List<TravelPackage> findByAgentId(Long agentId);

List<TravelPackage> findByTitleContainingIgnoreCase(String title);

List<TravelPackage> findByPriceLessThanEqual(**double** price);

List<TravelPackage> findByOffer\_CouponCode(String couponCode);

}

package com.example.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.example.service;

**import** com.example.client.UserClient;

**import** com.example.dto.FlightDTO;

**import** com.example.dto.HotelDTO;

**import** com.example.dto.ItineraryDTO;

**import** com.example.dto.OfferDTO;

**import** com.example.dto.SightseeingDTO;

**import** com.example.dto.TravelPackageDTO;

**import** com.example.dto.UserDTO;

**import** com.example.exception.ResourceNotFoundException;

**import** com.example.model.Flight;

**import** com.example.model.Hotel;

**import** com.example.model.Itinerary;

**import** com.example.model.Offer;

**import** com.example.model.Sightseeing;

**import** com.example.model.TravelPackage;

**import** com.example.repository.TravelPackageRepository;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

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

**import** org.springframework.stereotype.Service;

**import** java.util.ArrayList;

**import** java.util.List;

@Service

**public** **class** TravelPackageService {

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

**private** **final** TravelPackageRepository repository;

@Autowired

**private** **final** UserClient userClient;

**public** TravelPackageService(TravelPackageRepository repository,UserClient userClient) {

**this**.repository = repository;

**this**.userClient = userClient;

}

// Get all packages

**public** List<TravelPackage> getAllPackages() {

***logger***.info("Fetching all travel packages...");

**return** repository.findAll();

}

// Get package by ID

**public** TravelPackage getPackageById(Long id) {

***logger***.debug("Fetching travel package with ID: {}", id);

**return** repository.findById(id)

.orElseThrow(() -> {

***logger***.warn("Package not found with ID: {}", id);

**return** **new** ResourceNotFoundException("Package not found with ID: " + id);

});

}

**public** List<TravelPackage> getPackageByAgentId(Long agentId){

***logger***.info("Fetching packages for agent ID: {} ", agentId);

List<TravelPackage> packages = repository.findByAgentId(agentId);

**if**(packages.isEmpty()) {

***logger***.warn("No packages found for agent ID: {}", agentId);

**throw** **new** ResourceNotFoundException("No packages found for agent ID: "+agentId);

}

**return** packages;

}

// Create package

**public** TravelPackage createPackage(TravelPackage travelPackage) {

Long agentId = travelPackage.getAgentId();

***logger***.info("Validating agent ID: {}", agentId);

**try** {

UserDTO user = userClient.getUserById(agentId);

***logger***.info("User fetched from User Service: {}", user);

**if**(user == **null**) {

**throw** **new** ResourceNotFoundException("User not found for ID: "+agentId);

}

**if**(!"AGENT".equalsIgnoreCase(user.getRole())) {

***logger***.warn("User with ID {} is not an agent. Role: {}", agentId, user.getRole());

**throw** **new** IllegalArgumentException("User is not authorized to create travel packages.");

}

System.***out***.println("Agent fetched: "+ user.getName() + "Role" + user.getRole());

}**catch**(Exception e) {

***logger***.error("Feign call failed for agentId {}: {}", agentId, e.getMessage());

**throw** **new** ResourceNotFoundException("Agent with ID "+ agentId+" not found or invalid.");

}

**return** repository.save(travelPackage);

}

// logger.info("Creating new travel package: {}", travelPackage.getTitle());

//

// try {

// userClient.getUserById(travelPackage.getAgentId());

// }catch (Exception e) {

// logger.error("Agent ID {} not found in User Management Service", travelPackage.getAgentId());

// throw new IllegalArgumentException("Agent with ID"+ travelPackage.getAgentId() + " not found");

// }

// return repository.save(travelPackage);

// Update package

**public** TravelPackage updatePackage(Long id, TravelPackage updatedPackage) {

TravelPackage existing = repository.findById(id)

.orElseThrow(() -> **new** ResourceNotFoundException("Package not found with ID: " + id));

existing.setTitle(updatedPackage.getTitle());

existing.setDescription(updatedPackage.getDescription());

existing.setDuration(updatedPackage.getDuration());

existing.setPrice(updatedPackage.getPrice());

existing.setMaxCapacity(updatedPackage.getMaxCapacity());

existing.setTripStartDate(updatedPackage.getTripStartDate());

existing.setTripEndDate(updatedPackage.getTripEndDate());

existing.setHighlights(updatedPackage.getHighlights());

existing.setOffer(updatedPackage.getOffer());

// ✅ Safely update child lists — DO NOT use setFlights()

**if** (updatedPackage.getFlights() != **null**) {

existing.getFlights().clear();

existing.getFlights().addAll(updatedPackage.getFlights());

}

**if** (updatedPackage.getHotels() != **null**) {

existing.getHotels().clear();

existing.getHotels().addAll(updatedPackage.getHotels());

}

**if** (updatedPackage.getSightseeing() != **null**) {

existing.getSightseeing().clear();

existing.getSightseeing().addAll(updatedPackage.getSightseeing());

}

**if** (updatedPackage.getItinerary() != **null**) {

existing.getItinerary().clear();

existing.getItinerary().addAll(updatedPackage.getItinerary());

}

**return** repository.save(existing);

}

// Delete package

**public** **void** deletePackage(Long id) {

***logger***.info("Attempting to delete travel package with ID: {}", id);

**if** (!repository.existsById(id)) {

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

**throw** **new** ResourceNotFoundException("Package not found with ID: " + id);

}

repository.deleteById(id);

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

}

//DTO

**public** TravelPackageDTO convertToDTO(TravelPackage pkg) {

TravelPackageDTO dto = **new** TravelPackageDTO();

// Basic Fields

dto.setPackageId(pkg.getPackageId());

dto.setAgentId(pkg.getAgentId());

dto.setTitle(pkg.getTitle());

dto.setDescription(pkg.getDescription());

dto.setDuration(pkg.getDuration());

dto.setPrice(pkg.getPrice());

dto.setMaxCapacity(pkg.getMaxCapacity());

dto.setTripStartDate(pkg.getTripStartDate());

dto.setTripEndDate(pkg.getTripEndDate());

dto.setHighlights(pkg.getHighlights());

// Flights

List<FlightDTO> flightDTOs = **new** ArrayList<>();

**for** (Flight f : pkg.getFlights()) {

FlightDTO fd = **new** FlightDTO();

fd.setAirline(f.getAirline());

fd.setFromCity(f.getFromCity());

fd.setToCity(f.getToCity());

fd.setDepartureTime(f.getDepartureTime());

fd.setArrivalTime(f.getArrivalTime());

flightDTOs.add(fd);

}

dto.setFlights(flightDTOs);

// Hotels

List<HotelDTO> hotelDTOs = **new** ArrayList<>();

**for** (Hotel h : pkg.getHotels()) {

HotelDTO hd = **new** HotelDTO();

hd.setName(h.getName());

hd.setCity(h.getCity());

hd.setRating(h.getRating());

hd.setNights(h.getNights());

hd.setCostPerNight(h.getCostPerNight());

hotelDTOs.add(hd);

}

dto.setHotels(hotelDTOs);

// Sightseeing

List<SightseeingDTO> sightseeingDTOs = **new** ArrayList<>();

**for** (Sightseeing s : pkg.getSightseeing()) {

SightseeingDTO sd = **new** SightseeingDTO();

sd.setLocation(s.getLocation());

sd.setDescription(s.getDescription());

sightseeingDTOs.add(sd);

}

dto.setSightseeing(sightseeingDTOs);

// Itinerary

List<ItineraryDTO> itineraryDTOs = **new** ArrayList<>();

**for** (Itinerary i : pkg.getItinerary()) {

ItineraryDTO id = **new** ItineraryDTO();

id.setDayNumber(i.getDayNumber());

id.setActivityTitle(i.getActivityTitle());

id.setActivityDescription(i.getActivityDescription());

itineraryDTOs.add(id);

}

dto.setItinerary(itineraryDTOs);

// Offer

Offer offer = pkg.getOffer();

**if** (offer != **null**) {

OfferDTO od = **new** OfferDTO();

od.setCouponCode(offer.getCouponCode());

od.setDescription(offer.getDescription());

od.setDiscountPercentage(offer.getDiscountPercentage());

od.setActive(offer.isActive());

dto.setOffer(od);

}

**return** dto;

}

// for booking logic

**public** List<TravelPackage> searchByTitle(String title) {

**return** repository.findByTitleContainingIgnoreCase(title);

}

**public** List<TravelPackage> searchByPrice(**double** maxPrice) {

**return** repository.findByPriceLessThanEqual(maxPrice);

}

**public** List<TravelPackage> searchByOffer(String couponCode) {

**return** repository.findByOffer\_CouponCode(couponCode);

}

}

spring.application.name=travel-package-management

# Database Configuration

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

spring.datasource.username=root

spring.datasource.password=root

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

# Hibernate Configuration

spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# Server Configuration

server.port=8089

# Logging Configuration

logging.level.org.springframework=INFO

logging.level.com.example.travel=DEBUG

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

eureka.instance.prefer-ip-address=true

This is travel package management module all files are these