package com.example.usermanagementservice;

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 UsermanagementServiceApplication {

public static void main(String[] args) {

SpringApplication.run(UsermanagementServiceApplication.class, args);

}

//add a comment

}package com.example.usermanagementservice.config;

import com.example.usermanagementservice.security.JwtAuthFilter;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.security.authentication.AuthenticationManager;

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

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

import org.springframework.security.web.SecurityFilterChain;

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

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

@Configuration

public class SecurityConfig {

@Autowired

private JwtAuthFilter jwtAuthFilter;

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.csrf(csrf -> csrf.disable()) // Disable CSRF protection for APIs (stateless)

.authorizeHttpRequests(auth -> auth

.requestMatchers("/api/auth/\*\*", "/api/users/register", "/api/users/internal/{id}").permitAll()

.requestMatchers("api/users/packages/\*\*").permitAll()

.anyRequest().authenticated()

)

.sessionManagement(session -> session

.sessionCreationPolicy(SessionCreationPolicy.STATELESS) // No HTTP session saved

)

.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class); // Add our custom JWT filter

return http.build();

}

// Expose AuthenticationManager as a Spring bean

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

return config.getAuthenticationManager();

}

}package com.example.usermanagementservice.controller;

import com.example.usermanagementservice.dto.LoginRequest;

import com.example.usermanagementservice.dto.LoginResponse;

import com.example.usermanagementservice.exception.UserNotFoundException;

import com.example.usermanagementservice.model.User;

import com.example.usermanagementservice.repository.UserRepository;

import com.example.usermanagementservice.security.JwtService;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.validation.Valid;

import java.util.Collections;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/api/auth") // Base URL: /api/auth

public class AuthController {

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

@Autowired

private UserRepository userRepository;

@Autowired

private JwtService jwtService;

// Login endpoint

@PostMapping("/login")

public ResponseEntity<?> login(@Valid @RequestBody LoginRequest request) {

logger.info("Login attempt for email: {}", request.getEmail());

// 1. Find user by email

User user = userRepository.findByEmail(request.getEmail())

.orElseThrow(() -> new UserNotFoundException("User not Found with email: " + request.getEmail()));

// 2. If user not found or password doesn't match, return error

if (user == null || !user.getPassword().equals(request.getPassword())) {

logger.warn("Invalid login for email: {}", request.getEmail());

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(Collections.singletonMap("message", "Invalid email or password"));

}

// 3. Generate JWT token with user's email and role

String token = jwtService.generateToken(user.getEmail(), user.getRole());

logger.info("Login successful for email: {} with role: {}", user.getEmail(), user.getRole());

// 4. Return token + role in response

return ResponseEntity.ok(new LoginResponse(token, user.getRole()));

}

//log out endpoint

@PostMapping("/logout")

public ResponseEntity<?> logout(HttpServletRequest request) {

logger.info("Logout attempt triggered by user");

return ResponseEntity.ok(Collections.singletonMap("message", "Logout successful"));

}

}package com.example.usermanagementservice.controller;

import com.example.usermanagementservice.dto.TravelPackageDTO;

import com.example.usermanagementservice.dto.UserDTO;

import com.example.usermanagementservice.dto.UserRoleCountResponse;

import com.example.usermanagementservice.exception.RoleChangeNotAllowedException;

import com.example.usermanagementservice.model.User;

import com.example.usermanagementservice.service.UserService;

import io.jsonwebtoken.Jwts;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.validation.Valid;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.security.core.context.SecurityContextHolder;

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

import com.example.usermanagementservice.security.\*;

import java.util.Collections;

import java.util.List;

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserService userService;

@Autowired

private JwtService jwtService;

// Helper method to extract role from JWT token in Authorization header

private String extractRoleFromHeader(HttpServletRequest request) {

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

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

String token = authHeader.substring(7);

return Jwts.parserBuilder()

.setSigningKey(jwtService.getSecretKey())

.build()

.parseClaimsJws(token)

.getBody()

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

}

return null;

}

/\*\*

\* Register new user

\* @param user the user object containing register details

\* @return a ResponseEntity containing user details

\*/

@PostMapping("/register")

public ResponseEntity<User> registerUser(@Valid @RequestBody User user) {

User savedUser = userService.registerUser(user);

return ResponseEntity.ok(savedUser);

}

/\*\*

\* Get all users - only ADMIN can access

\* @param request HttpServeletRequest to extract role from JWT

\* @return ResponseEntity containing list of users if ADMIN, otherwise 403 FORBIDDEN

\*/

@GetMapping

public ResponseEntity<?> getAllUsers(HttpServletRequest request) {

String role = extractRoleFromHeader(request); //Extract role from jwt

if (!"ADMIN".equals(role)) {

return ResponseEntity.status(HttpStatus.FORBIDDEN).body("Access denied: Only ADMINs can view all users");

}

List<User> users = userService.getAllUsers();

System.out.println("Role from token = " + role);

return ResponseEntity.ok(users);

}

// Get user by ID - only ADMIN

@GetMapping("/{id}")

public ResponseEntity<?> getUserById(@PathVariable Long id, HttpServletRequest request) {

String role = extractRoleFromHeader(request);

User user = userService.getUserById(id);

if (!"ADMIN".equals(role)) {

return ResponseEntity.status(HttpStatus.FORBIDDEN).body("Access denied");

}

return ResponseEntity.ok(user);

}

// Update user profile - only admin

@PutMapping("/{id}")

public ResponseEntity<?> updateUserProfile(@PathVariable Long id, @RequestBody User updatedUser, HttpServletRequest request) {

String role = extractRoleFromHeader(request);

// Only ADMIN

if (!"ADMIN".equals(role)) {

return ResponseEntity.status(HttpStatus.FORBIDDEN).body("Access denied: You can only update your own profile");

}

User user = userService.updateUserProfile(id, updatedUser);

return ResponseEntity.ok(user);

}

// Delete user - only ADMIN can delete

@DeleteMapping("/{id}")

public ResponseEntity<?> deleteUser(@PathVariable Long id, HttpServletRequest request) {

String role = extractRoleFromHeader(request);

if (!"ADMIN".equals(role)) {

return ResponseEntity.status(HttpStatus.FORBIDDEN).body("Only ADMINs can delete users.");

}

userService.deleteUser(id);

return ResponseEntity.ok("User deleted.");

}

//get total count of users, agents, customers

@GetMapping("/counts")

public ResponseEntity<?> getUserRoleCounts(HttpServletRequest request) {

String role = extractRoleFromHeader(request);

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

return ResponseEntity.status(HttpStatus.FORBIDDEN)

.body(Collections.singletonMap("message", "Only admins can view user counts."));

}

UserRoleCountResponse response = userService.getUserRoleCounts();

return ResponseEntity.ok(response);

}

// myprofile endpoint for users(only)

//get their profile details

@GetMapping("/myprofile")

public ResponseEntity<?> getMyProfile() {

String email = SecurityContextHolder.getContext().getAuthentication().getName();

User user = userService.getUserByEmail(email);

return ResponseEntity.ok(user);

}

//update their own profile details

@PutMapping("/myprofile")

public ResponseEntity<?> updateMyProfile(@RequestBody User updatedUser) {

String email = SecurityContextHolder.getContext().getAuthentication().getName();

User existingUser = userService.getUserByEmail(email);

// Add role-protection logic here

if (!"ADMIN".equalsIgnoreCase(existingUser.getRole()) &&

"ADMIN".equalsIgnoreCase(updatedUser.getRole())) {

throw new RoleChangeNotAllowedException("You cannot assign yourself ADMIN role.");

}

User user = userService.updateUserProfile(existingUser.getId(), updatedUser);

return ResponseEntity.ok(user);

}

//delete their own profile

@DeleteMapping("/myprofile")

public ResponseEntity<?> deleteMyProfile() {

String email = SecurityContextHolder.getContext().getAuthentication().getName();

User user = userService.getUserByEmail(email);

userService.deleteUser(user.getId());

return ResponseEntity.ok(Collections.singletonMap("message", "Your profile has been deleted."));

}

/\*---------------------------------\*/

// Internal endpoint for microservices (e.g., Travel Package Service)

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

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

User user = userService.getUserById(id); // No security check, get user from DB

// ✅ Check if user exists

if (user == null) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND)

.body(Collections.singletonMap("message", "User not found with ID: " + id));

}

// ✅ Check if user is an AGENT

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

return ResponseEntity.status(HttpStatus.FORBIDDEN)

.body(Collections.singletonMap("message", "User with ID " + id + " is not an AGENT"));

}

UserDTO userDTO = userService.convertToDTO(user);

return ResponseEntity.ok(userDTO);

}

//To fetch all the packages under a travel agent

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

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

List<TravelPackageDTO> packages = userService.fetchAllPackagesByAgent(id);

return ResponseEntity.ok(packages);

}

}**package** com.example.usermanagementservice.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.usermanagementservice.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.usermanagementservice.dto;

**import** lombok.Data;

@Data

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

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

**private** String activityTitle;

**private** String activityDescription;

}

package com.example.usermanagementservice.dto;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

public class LoginRequest {

@Email(message = "Invalid email format")

@NotBlank(message = "Email is required")

private String email;

@NotBlank(message = "Email is required")

private String password;

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}**package** com.example.usermanagementservice.dto;

**public** **class** LoginResponse {

**private** String token;

**private** String role;

**public** LoginResponse(String token, String role) {

**this**.token = token;

**this**.role = role;

}

// Getters only

**public** String getToken() {

**return** token;

}

**public** String getRole() {

**return** role;

}

}

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

**import** lombok.Data;

@Data

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

**private** String couponCode;

**private** String description;

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

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

}

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

**import** lombok.Data;

@Data

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

**private** String location;

**private** String description;

}

package com.example.usermanagementservice.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.usermanagementservice.dto;

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

**private** Long id;

**private** String name;

**private** String email;

**private** String role;

// Constructors

**public** UserDTO() {

}

**public** UserDTO(Long id, String name, String email, String role) {

**this**.id = id;

**this**.name = name;

**this**.email = email;

**this**.role = role;

}

// Getters & Setters

**public** Long getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getRole() {

**return** role;

}

**public** **void** setRole(String role) {

**this**.role = role;

}

}

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

**public** **class** UserRoleCountResponse {

**private** **long** totalUsers;

**private** **long** agentCount;

**private** **long** customerCount;

**public** UserRoleCountResponse(**long** totalUsers, **long** agentCount, **long** customerCount) {

**this**.totalUsers = totalUsers;

**this**.agentCount = agentCount;

**this**.customerCount = customerCount;

}

// Getters

**public** **long** getTotalUsers() { **return** totalUsers; }

**public** **long** getAgentCount() { **return** agentCount; }

**public** **long** getCustomerCount() { **return** customerCount; }

}

**package** com.example.usermanagementservice.exception;

**public** **class** AdminRegistrationNotAllowedException **extends** RuntimeException {

**public** AdminRegistrationNotAllowedException(String message) {

**super**(message); // This allows getMessage() to work

}

}

**package** com.example.usermanagementservice.exception;

// Must extend RuntimeException or Exception

**public** **class** EmailAlreadyExistsException **extends** RuntimeException {

**public** EmailAlreadyExistsException(String message) {

**super**(message); // This allows getMessage() to work

}

}

package com.example.usermanagementservice.exception;

import org.springframework.web.bind.MethodArgumentNotValidException;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import java.util.Collections;

import java.util.stream.Collectors;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

import java.util.Collections;

//ControllerAdvice + ResponseBody

@RestControllerAdvice //intercepts exceptions thrown by any restcontroller in the app, centralises error handling logic

public class GlobalExceptionHandler {

//if email already exists in db during registration

@ExceptionHandler(EmailAlreadyExistsException.class)

public ResponseEntity<?> handleEmailExists(EmailAlreadyExistsException ex) {

return ResponseEntity.status(HttpStatus.CONFLICT) //409

.body(Collections.singletonMap("message", ex.getMessage()));

}

//if number already exists in db during registration

@ExceptionHandler(PhoneNumberAlreadyExistsException.class)

public ResponseEntity<?> handlePhoneNumberExists(PhoneNumberAlreadyExistsException ex) {

return ResponseEntity.status(HttpStatus.CONFLICT) //409

.body(Collections.singletonMap("message", ex.getMessage()));

}

//if user not found

@ExceptionHandler(UserNotFoundException.class)

public ResponseEntity<?> handleUserNotFound(UserNotFoundException ex) {

return ResponseEntity

.status(HttpStatus.NOT\_FOUND) // This sends 404

.body(Collections.singletonMap("message", ex.getMessage()));

}

//validation for email and phone number format

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<?> handleValidationErrors(MethodArgumentNotValidException ex) {

// Collect all validation error messages (e.g. "email: Invalid format")

String message = ex.getBindingResult().getFieldErrors().stream()

.map(error -> error.getField() + ": " + error.getDefaultMessage())

.collect(Collectors.joining(", "));

// Return a clean JSON response

return ResponseEntity

.status(HttpStatus.BAD\_REQUEST) //403

.body(Collections.singletonMap("message", message));

}

//if user tries to register as admin

@ExceptionHandler(AdminRegistrationNotAllowedException.class)

public ResponseEntity<?> handleAdminRegisterBlocked(AdminRegistrationNotAllowedException ex) {

return ResponseEntity

.status(HttpStatus.FORBIDDEN)

.body(Collections.singletonMap("message", ex.getMessage()));

}

//if non admin user tries to update their role

@ExceptionHandler(RoleChangeNotAllowedException.class)

public ResponseEntity<?> handleRoleChange(RoleChangeNotAllowedException ex) {

return ResponseEntity.status(HttpStatus.FORBIDDEN)

.body(Collections.singletonMap("message", ex.getMessage()));

}

}**package** com.example.usermanagementservice.exception;

**public** **class** PhoneNumberAlreadyExistsException **extends** RuntimeException{

**public** PhoneNumberAlreadyExistsException(String message) {

**super**(message);

}

}

**package** com.example.usermanagementservice.exception;

**public** **class** RoleChangeNotAllowedException **extends** RuntimeException{

**public** RoleChangeNotAllowedException(String msg) {

**super**(msg);

}

}

**package** com.example.usermanagementservice.exception;

// Must extend RuntimeException or Exception

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

**public** UserNotFoundException(String message) {

**super**(message); // This allows getMessage() to work

}

}

package com.example.usermanagementservice.feign;

import com.example.usermanagementservice.dto.TravelPackageDTO;

import org.springframework.cloud.openfeign.FeignClient;

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

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

import java.util.List;

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

public interface TravelPackageClient {

@GetMapping("/api/packages/admin/agent/{agentId}")

List<TravelPackageDTO> getPackagesByAgent(@PathVariable("agentId") Long id);

}package com.example.usermanagementservice.model;

import jakarta.persistence.\*;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Pattern;

import jakarta.validation.constraints.Size;

// this class represents a user in the system

@Entity

@Table(name = "users") //set table name as 'users'

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)// Auto-increment primary key

private Long id;

@Column(nullable=false)

private String name; //User's name

@Email(message ="Invalid email format")

@NotBlank(message="Email is required")

@Column(nullable=false, unique=true)

private String email; //Email must be unique

@NotBlank(message="Password is required")

@Size(min = 6, message = "Password must be atleast 6 characters")

@Column(nullable=false)

private String password; //User's password

@Column(nullable=false)

private String role; // Role: AGENT or CUSTOMER

@NotBlank(message="Contact number is required")

@Pattern(regexp = "\\d{10}", message = "Contact number must be 10 digits")

@Column(name ="contact\_number",nullable=false)

private String contactNumber; // phone number of the user

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

public String getContactNumber() {

return contactNumber;

}

public void setContactNumber(String contactNumber) {

this.contactNumber = contactNumber;

}}  
package com.example.usermanagementservice.repository;

import com.example.usermanagementservice.model.User;

import java.util.Optional;

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

import org.springframework.stereotype.Repository;

// This interface gives us all basic DB operations for the User entity

@Repository

//this tells spring to work with user entity and primary key is long

public interface UserRepository extends JpaRepository<User, Long>{

// we can add custom methods here if needed

//to find user by email

Optional<User> findByEmail(String email);

Optional<User> findByContactNumber(String contactNumber);

//Total users

long count();

//Count by role(Agent, customer)

long countByRole(String role);

}package com.example.usermanagementservice.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.HttpStatus;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

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

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

import java.util.Collections;

@Component

public class JwtAuthFilter extends OncePerRequestFilter {

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

@Autowired

private JwtService jwtService;

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response,

FilterChain filterChain) throws ServletException, IOException {

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

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

String token = authHeader.substring(7);

try {

Claims claims = Jwts.parserBuilder()

.setSigningKey(jwtService.getSecretKey())

.build()

.parseClaimsJws(token)

.getBody();

String email = claims.getSubject();

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

logger.info("Token validated for email: {} with role: {}", email, role);

UsernamePasswordAuthenticationToken authentication =

new UsernamePasswordAuthenticationToken(email, null, Collections.emptyList());

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

SecurityContextHolder.getContext().setAuthentication(authentication);

} catch (Exception e) {

logger.warn("❌ Invalid or expired JWT: {}", e.getMessage());

response.setStatus(HttpStatus.FORBIDDEN.value());

response.setContentType("application/json");

response.getWriter().write("{\"message\":\"You have been logged out or session expired\"}");

return; //Don't continue filter chain

}

}

filterChain.doFilter(request, response);

}

}package com.example.usermanagementservice.security;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

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

import org.springframework.stereotype.Service;

import java.security.Key;

import java.util.Date;

@Service

public class JwtService {

// Inject from application.properties

@Value("${jwt.secret}")

private String secret;

private final long EXPIRATION\_TIME = 60 \* 60 \* 1000; // 1 hour

// Generate token using dynamic key from injected secret

public String generateToken(String email, String role) {

Key key = Keys.hmacShaKeyFor(secret.getBytes());

return Jwts.builder()

.setSubject(email)

.claim("role", role)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + EXPIRATION\_TIME))

.signWith(key, SignatureAlgorithm.HS256)

.compact();

}

// Provide key for validation (used by filters/controllers)

public Key getSecretKey() {

return Keys.hmacShaKeyFor(secret.getBytes());

}

}//All the business logic like Registering, getting, updating, deleting a user

package com.example.usermanagementservice.service;

import com.example.usermanagementservice.dto.UserRoleCountResponse;

//this interface defines what functions are available

import com.example.usermanagementservice.model.User;

import jakarta.servlet.http.HttpServletRequest;

//import com.example.usermanagementservice.dto.PackageDTO;

import com.example.usermanagementservice.dto.TravelPackageDTO;

import com.example.usermanagementservice.dto.UserDTO;

import java.util.List;

public interface UserService {

User registerUser(User user); //Add new user

List<User> getAllUsers(); // Get all users

User getUserByEmail(String email); //find user by email

void deleteUser(Long id); //deletes user

User getUserById(Long id); //View profile

User updateUserProfile(Long id, User updatedUser);//Update profile like user management in their profile for editing profiles

//get user count and count by roles as well

UserRoleCountResponse getUserRoleCounts();

//get all packages created by clients

List<TravelPackageDTO> fetchAllPackagesByAgent(Long id);

UserDTO convertToDTO(User user);

}package com.example.usermanagementservice.service.impl;

import com.example.usermanagementservice.model.User;

import jakarta.servlet.http.HttpServletRequest;

import com.example.usermanagementservice.repository.UserRepository;

import com.example.usermanagementservice.service.UserService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Service;

//import com.example.usermanagementservice.dto.TravelPackageDTO;

import com.example.usermanagementservice.dto.TravelPackageDTO;

import com.example.usermanagementservice.dto.UserDTO;

import com.example.usermanagementservice.dto.UserRoleCountResponse;

import com.example.usermanagementservice.exception.AdminRegistrationNotAllowedException;

import com.example.usermanagementservice.exception.EmailAlreadyExistsException;

import com.example.usermanagementservice.exception.PhoneNumberAlreadyExistsException;

import com.example.usermanagementservice.exception.UserNotFoundException;

import com.example.usermanagementservice.feign.TravelPackageClient;

import java.util.List;

//this class implements the logic defined in Userservice

@Service

public class UserServiceImpl implements UserService {

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

@Autowired

private UserRepository userRepository;

@Autowired

private TravelPackageClient travelPackageClient;

//Register a new user

@Override

public User registerUser(User user) {

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

// check if email already exists

if (userRepository.findByEmail(user.getEmail()).isPresent() ) {

//Don't continue if already exists

throw new EmailAlreadyExistsException("Email already registered");

}

// check if number already exists

if(userRepository.findByContactNumber(user.getContactNumber()).isPresent()) {

throw new PhoneNumberAlreadyExistsException("Phone number already registered");

}

//prevent users from registering as admin

if("ADMIN".equalsIgnoreCase(user.getRole())) {

throw new AdminRegistrationNotAllowedException("You are not allowed to register as ADMIN");

}

//Proceed to save

return userRepository.save(user); //Save user to DB

}

// Get all users

@Override

public List<User> getAllUsers() {

logger.info("Fetching all users...");

return userRepository.findAll();

}

//Find user by email

@Override

public User getUserByEmail(String email) {

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

return userRepository.findByEmail(email)

.orElseThrow(() -> new UserNotFoundException("User not found with email: " + email));

}

//Delete user by ID

@Override

public void deleteUser(Long id) {

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

if(! userRepository.existsById(id)) {

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

}

userRepository.deleteById(id);

}

//view user by ID(profile view)

@Override

public User getUserById(Long id) {

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

return userRepository.findById(id)

.orElseThrow(() -> new UserNotFoundException("User not found with ID: "+ id));

}

//update user profile

@Override

public User updateUserProfile(Long id, User updatedUser) {

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

//First, get existing user

User existingUser = userRepository.findById(id).orElseThrow(() -> new UserNotFoundException("User not found with ID: " + id));

//Update fields

existingUser.setName(updatedUser.getName());

existingUser.setEmail(updatedUser.getEmail());

existingUser.setPassword(updatedUser.getPassword());

existingUser.setRole(updatedUser.getRole());

existingUser.setContactNumber(updatedUser.getContactNumber());

//save changes

return userRepository.save(existingUser);

}

//to get number of users, agents and customers

@Override

public UserRoleCountResponse getUserRoleCounts() {

long total = userRepository.count()-1;

long agents = userRepository.countByRole("AGENT");

long customers = userRepository.countByRole("CUSTOMER");

logger.info("Fetched user role counts: total={}, agents={}, customers={}", total, agents, customers);

return new UserRoleCountResponse(total, agents, customers);

}

//to get all the created packages of an agent

@Override

public List<TravelPackageDTO> fetchAllPackagesByAgent(Long agentId) {

// ✅ Pass `id` as `agentId`

return travelPackageClient.getPackagesByAgent(agentId);

}

//convert user to userDTO for the user data transfer to other services

public UserDTO convertToDTO(User user) {

return new UserDTO(user.getId(), user.getName(), user.getEmail(), user.getRole());

}

}# Set port

server.port=8081

# Service name for Eureka

spring.application.name=usermanagement-service

# MySQL database connection

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

spring.datasource.username=root

spring.datasource.password=root

# JPA table auto-creation

spring.jpa.hibernate.ddl-auto=update

# Show SQL queries in console

spring.jpa.show-sql=true

# Eureka Discovery Server location

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

eureka.instance.prefer-ip-address=true

#eureka.instance.ip-address=10.232.12.60

#JWT Secret key(atleast 32 characters for HS256)

jwt.secret=thisgoodkeyhasmademewonderhowitworkswhenitjustworkssowell