

ΕΝΟΤΗΤΑ ΕΡΓΑΣΙΩΝ 2:
ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΨΗΦΙΑΚΟΥ ΤΑΞΙΔΙΩΤΙΚΟΥ ΠΡΑΚΤΟΡΑ
ΘΕΜΑΤΙΚΟΥ ΤΟΥΡΙΣΜΟΥ & ΙΣΤΟΣΕΛΙΔΑΣ ΕΡΓΟΥ

ΠΑΡΑΔΟΤΕΟ: ΠΛΗΡΟΦΟΡΙΑΚΟ ΣΥΣΤΗΜΑ

ΚΩΔΙΚΑΣ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ THEMATTCATHENS

Περιεχόμενα

Controllers	2
UserController Documentation	2
VendorController Documentation	22
AdventuresTourismController Documentation	24
AdventuresTourismDatesController Documentation	41
TranslationController Documentation	50
HotelController Documentation	62
HotelRoomController Documentation	74
RecommendationsController Documentation	84
BookingController Documentation.....	130
API endpoints	148

Controllers

UserController Documentation

Ο UserController είναι υπεύθυνος για τη διαχείριση όλων των λειτουργιών που σχετίζονται με τους χρήστες της εφαρμογής. Αυτές περιλαμβάνουν εγγραφή, σύνδεση, αποσύνδεση, ενημέρωση προφίλ, διαχείριση κωδικών πρόσβασης, διαγραφή λογαριασμού και άλλες σχετικές λειτουργίες. Ο controller χρησιμοποιεί το BaseController για τυποποιημένες απαντήσεις και το Spatie Permission για τη διαχείριση ρόλων.

Εξαρτήσεις (Dependencies)

- **BaseController**: Βασικός controller για τυποποιημένα responses.
- **User Model**: Χρησιμοποιείται για τη διαχείριση των χρηστών στη βάση δεδομένων.
- **Spatie Permission**: Για τη διαχείριση ρόλων και δικαιωμάτων.
- **MediaFileController**: Για τη διαχείριση μεταφορτώσεων αρχείων (π.χ. avatar).
- **Mail**: Για την αποστολή email (π.χ. εγγραφή, επαναφορά κωδικού).
- **Validator**: Για την επικύρωση εισόδου.
- **Auth**: Για τη διαχείριση πιστοποίησης.
- **Log**: Για την καταγραφή σφαλμάτων και πληροφοριών.

Μέθοδοι

1. register

- **Σκοπός**: Εγγραφή νέου χρήστη.
- **Παράμετροι**:
 - username: Το όνομα χρήστη (μοναδικό).
 - first_name: Το όνομα του χρήστη.
 - last_name: Το επώνυμό του.
 - email: Το email του χρήστη (μοναδικό).
 - phone: Το τηλέφωνό του.
 - password: Ο κωδικός πρόσβασης.
 - c_password: Επιβεβαίωση κωδικού πρόσβασης.
 - roles: Ο ρόλος του χρήστη (π.χ. Customer, Vendor).
- **Λειτουργία**:

- Επικυρώνει τα δεδομένα εισόδου.
 - Δημιουργεί έναν νέο χρήστη στη βάση δεδομένων.
 - Αντιστοιχεί τον ρόλο στον χρήστη.
 - Στέλνει email επιβεβαίωσης ανάλογα με τον ρόλο (Customer ή Vendor).
 - **Response:** Μήνυμα επιτυχίας ή σφάλμα επικύρωσης.
-

2. login

- **Σκοπός:** Σύνδεση χρήστη.
 - **Παράμετροι:**
 - email: Το email του χρήστη.
 - password: Ο κωδικός πρόσβασης.
 - **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Ελέγχει αν ο κωδικός πρόσβασης είναι σωστός.
 - Δημιουργεί έναν νέο token για τον χρήστη.
 - Επιστρέφει τα στοιχεία του χρήστη και τον token.
 - **Response:** Στοιχεία χρήστη και token ή σφάλμα πιστοποίησης.
-

3. logout

- **Σκοπός:** Αποσύνδεση χρήστη.
 - **Λειτουργία:**
 - Διαγράφει το τρέχον token του χρήστη.
 - **Response:** Μήνυμα επιτυχίας.
-

4. getUserInfo

- **Σκοπός:** Λήψη πληροφοριών για τον τρέχοντα χρήστη.
 - **Λειτουργία:**
 - Επιστρέφει τα στοιχεία του χρήστη που είναι συνδεδεμένος.
 - **Response:** Στοιχεία χρήστη ή σφάλμα αν ο χρήστης δεν είναι συνδεδεμένος.
-

5. getAllUsers

- **Σκοπός:** Λήψη λίστας όλων των χρηστών (μόνο για διαχειριστές).
 - **Λειτουργία:**
 - Ελέγχει αν ο χρήστης έχει τον ρόλο "Admin".
 - Επιστρέφει όλους τους χρήστες μαζί με τους ρόλους τους.
 - **Response:** Λίστα χρηστών ή σφάλμα αν ο χρήστης δεν είναι διαχειριστής.
-

6. updateUserInfo

- **Σκοπός:** Ενημέρωση πληροφοριών χρήστη.
 - **Παράμετροι:**
 - username: Το όνομα χρήστη.
 - first_name: Το όνομα του χρήστη.
 - last_name: Το επώνυμό του.
 - email: Το email του χρήστη.
 - phone: Το τηλέφωνό του.
 - business_name: Το όνομα της επιχείρησης (για Vendors).
 - afm: Το ΑΦΜ (για Vendors).
 - bio: Μια σύντομη βιογραφία.
 - avatar_id: Εικόνα προφίλ.
 - **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Ενημερώνει τις πληροφορίες του χρήστη.
 - Διαχειρίζεται την ενημέρωση της εικόνας προφίλ (αν υπάρχει).
 - **Response:** Ενημερωμένα στοιχεία χρήστη ή σφάλμα.
-

7. changePassword

- **Σκοπός:** Αλλαγή κωδικού πρόσβασης.
- **Παράμετροι:**
 - current_password: Ο τρέχων κωδικός πρόσβασης.
 - new_password: Ο νέος κωδικός πρόσβασης.
 - confirm_new_password: Επιβεβαίωση του νέου κωδικού.
- **Λειτουργία:**

- Επικυρώνει τα δεδομένα εισόδου.
 - Ελέγχει αν ο τρέχων κωδικός είναι σωστός.
 - Ενημερώνει τον κωδικό πρόσβασης.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

8. deleteUser

- **Σκοπός:** Διαγραφή λογαριασμού χρήστη.
- **Λειτουργία:**
 - Ελέγχει αν ο χρήστης έχει ενεργές κρατήσεις (bookings).
 - Αν δεν υπάρχουν ενεργές κρατήσεις, διαγράφει τον λογαριασμό.
 - Στέλνει email ειδοποίησης για διαγραφή (για Vendors).
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

9. getVendorDetails

- **Σκοπός:** Λήψη λεπτομερειών για έναν Vendor.
- **Παράμετροι:**
 - id: Το ID του Vendor.
- **Λειτουργία:**
 - Επιστρέφει τις λεπτομέρειες του Vendor, συμπεριλαμβανομένης της εικόνας προφίλ.
- **Response:** Στοιχεία Vendor ή σφάλμα αν ο χρήστης δεν είναι Vendor.

10. getVendorServices

- **Σκοπός:** Λήψη υπηρεσιών που προσφέρει ένας Vendor.
- **Λειτουργία:**
 - Επιστρέφει όλες τις υπηρεσίες που έχει δημιουργήσει ο Vendor, ομαδοποιημένες ανά τύπο υπηρεσίας.
- **Response:** Λίστα υπηρεσιών ή σφάλμα.

```
<?php
```

```

namespace App\Http\Controllers\API;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Validator;
use Illuminate\Http\JsonResponse;
use Spatie\Permission\Models\Role;
use Illuminate\Auth\Events\Registered;
use Illuminate\Http\Response;
use Illuminate\Support\Facades\DB;
use Modules\MediaFile\Controllers\MediaFileController;
use Illuminate\Support\Facades\Log;

use Illuminate\Support\Facades>Password;
use Illuminate\Auth\Events>PasswordReset;

use Illuminate\Support\Facades\Mail;

use Modules\Booking\Models\Booking;
use App\Models\HotelRoomBooking;

//use App\Mail\SignupEmail;

use App\Mail\AccountDeletionVendorEmail;
use App\Mail>PasswordResetEmail;
use App\Mail\UserSignupEmail;
use App\Mail\VendorSignupEmail;

use Modules\AdventuresTourism\Models\AdventuresTourism;
use Modules\Agritourism\Models\Agritourism;
use Modules\AirActivitiesTourism\Models\AirActivitiesTourism;
use Modules\CityGamesTourism\Models\CityGamesTourism;
use Modules\CulturalFestivalTourism\Models\CulturalFestivalTourism;
use
Modules\CulturalUrbanExplorationTourism\Models\CulturalUrbanExploration
Tourism;
use Modules\CulturalUrbanTourism\Models\CulturalUrbanTourism;
use Modules\DarkTourism\Models\DarkTourism;
use Modules\Ecotourism\Models\Ecotourism;
use
Modules\EducationDevelopmentTourism\Models\EducationDevelopmentTourism;
use
Modules\EnrichmentSkillBuildingTourism\Models\EnrichmentSkillBuildingTo
urism;
use Modules\ExplorationTourism\Models\ExplorationTourism;

```

```

use Modules\FishingTourism\Models\FishingTourism;
use Modules\GastronomicTourism\Models\GastronomicTourism;
use Modules\Geotourism\Models\Geotourism;
use Modules\SacredSiteVisitsTourism\Models\SacredSiteVisitsTourism;
use Modules\SportingEventsTourism\Models\SportingEventsTourism;
use Modules\UrbanEntertainmentTourism\Models\UrbanEntertainmentTourism;
use Modules\WaterActivitiesTourism\Models\WaterActivitiesTourism;
use Modules\WeddingTourism\Models\WeddingTourism;
use Modules\WineTourism\Models\WineTourism;
use Modules\MedicalTourism\Models\MedicalTourism;
use Modules\SpaTourism\Models\SpaTourism;
use Modules\WellnessTourism\Models\WellnessTourism;

use Modules\AdventuresTourism\Resources\AdventuresTourismResource;
use Modules\Agritourism\Resources\AgritourismResource;
use
Modules\AirActivitiesTourism\Resources\AirActivitiesTourismResource;
use Modules\CityGamesTourism\Resources\CityGamesTourismResource;
use
Modules\CulturalFestivalTourism\Resources\CulturalFestivalTourismResource;
use
Modules\CulturalUrbanExplorationTourism\Resources\CulturalUrbanExplorationTourismResource;
use
Modules\CulturalUrbanTourism\Resources\CulturalUrbanTourismResource;
use Modules\DarkTourism\Resources\DarkTourismResource;
use Modules\Ecotourism\Resources\EcotourismResource;
use
Modules\EducationDevelopmentTourism\Resources\EducationDevelopmentTourismResource;
use
Modules\EnrichmentSkillBuildingTourism\Resources\EnrichmentSkillBuildingTourismResource;
use Modules\ExplorationTourism\Resources\ExplorationTourismResource;
use Modules\FishingTourism\Resources\FishingTourismResource;
use Modules\GastronomicTourism\Resources\GastronomicTourismResource;
use Modules\Geotourism\Resources\GeotourismResource;
use
Modules\SacredSiteVisitsTourism\Resources\SacredSiteVisitsTourismResource;
use
Modules\SportingEventsTourism\Resources\SportingEventsTourismResource;
use
Modules\UrbanEntertainmentTourism\Resources\UrbanEntertainmentTourismResource;

```

```

use
Modules\WaterActivitiesTourism\Resources\WaterActivitiesTourismResource
;
use Modules\WeddingTourism\Resources\WeddingTourismResource;
use Modules\WineTourism\Resources\WineTourismResource;
use Modules\MedicalTourism\Resources\MedicalTourismResource;
use Modules\SpaTourism\Resources\SpaTourismResource;
use Modules\WellnessTourism\Resources\WellnessTourismResource;

class UserController extends BaseController
{

    protected $mediaFileClass;

    function __construct(array $attributes = [])
    {
        $this->mediaFileClass = MediaFileController::class;
        $this->middleware(['auth:sanctum', 'permission:edit-service'],
['only' => ['updateUserInfo']]);
    }

    /**
     * Register api
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function register(Request $request): JsonResponse
    {
        $validator = Validator::make($request->all(), [
            'username' => 'required|string|unique:users|max:255',
            'first_name' => 'required|string|max:255',
            'last_name' => 'required|string|max:255',
            'email' => 'required|email|max:255|unique:users',
            'phone' => 'required|digits:10',
            'password' => 'required',
            'c_password' => 'required|same:password',
            'roles' => 'required',
            // 'role_id' => 'required|exists:roles,id',

        ]);

        if($validator->fails()){
            return $this->sendError('Validation Error.', $validator-
>errors());
        }

        // $input = $request->all();

```

```

        // $input['name'] = $input['first_name'] . ' ' .
$input['last_name'];
        // $input['password'] = bcrypt($input['password']);

        $user = User::create([
            'username' => $request->input('username'),
            'first_name' => $request->input('first_name'),
            'last_name' => $request->input('last_name'),
            'email' => $request->input('email'),
            'password' => bcrypt($request->input('password')),
            'publish' => $request->input('publish'),
            'phone' => $request->input('phone'),
            // 'role_id' => $request->input('role_id'),
        ]);

        $user->assignRole($request->input('roles'));

        $recipient = $request->input('email');
        $userName = $request->input('username');

        if ($user->hasRole('Customer')) {
            Mail::to($recipient)->send(new UserSignupEmail($userName,
$recipient));
        } else if ($user->hasRole('Vendor')) {
            Mail::to($recipient)->send(new VendorSignupEmail($request-
>input('first_name'), $recipient));
        }

        // $success['token'] = $user->createToken('authToken')-
>plainTextToken;
        // $success['username'] = $user->username;

        return $this->sendResponse('Registration Succeeded', 'User
register successfully.');
```

```

    }

    /**
     * Login api
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function login(Request $request): JsonResponse
    {

```

```

        // Validate the email and password
        $validator = Validator::make($request->all(), [
            'email' => 'required|email|exists:users,email',
            'password' => 'required',
        ]);

        // If validation fails, return an error response
        if ($validator->fails()) {
            return $this->sendError('Validation Error.', $validator-
>errors());
        }

        $user = User::where('email', $request->email)->first();

        if (!Hash::check($request->password, $user->password)) {
            return $this->sendError('Password is not
correct', ['code'=>'invalid_credentials']);
        }

        if(Auth::attempt(['email' => $request->email, 'password' =>
$request->password])){

            $success['token'] = $user->createToken('authToken')-
>plainTextToken;
            $success['role'] = $user->roles->first()->name;
            $success['user'] = $user;

            return $this->sendResponse($success, 'User login
successfully.');
```

```

        }
        else{
            return $this->sendError('Unauthorised.',
['error'=>'Unauthorised']);
        }
    }

    /**
     * Log the user out (Invalidate the token).
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function logout(Request $request): JsonResponse
    {

```

```

        $success['delete_token'] = $request->user()-
>currentAccessToken()->delete();

        return $this->sendResponse($success, 'User logout
successfully.');
```

```

    }

    /**
     * Get the authenticated User.
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function getUserInfo(): JsonResponse
    {
        // Check if the user is authenticated
        if (!Auth::check()) {
            // Return an error response if no user is authenticated
            return $this->sendError('Unauthorised.',
['error'=>'Unauthorised']);
        }

        $user = auth()->user();

        $success = $user;
        // Return the user info
        return $this->sendResponse($success, 'Getting user info
succeed');
```

```

        // return response()->json([
        //     'id' => $user->id,
        //     'name' => $user->name,
        //     'email' => $user->email,
        //     // Add other user fields as needed
        // ]);

    }

    public function getAllUsers(Request $request): JsonResponse
    {
        // Check if the authenticated user has the 'admin' role
        if (!Auth::check() || !Auth::user()->hasRole('Admin')) {
            return $this->sendError('Unauthorised.', ['error' =>
'Unauthorised']);
        }

        // Retrieve all users with their roles

```

```

        $users = User::with('roles')->get();

        // Transform the users data if necessary
        $usersData = $users->map(function ($user) {
            return [
                'id' => $user->id,
                'username' => $user->username,
                'first_name' => $user->first_name,
                'last_name' => $user->last_name,
                'email' => $user->email,
                'phone' => $user->phone,
                'business_name' => $user->business_name,
                'afm' => $user->afm,
                'bio' => $user->bio,
                'created_at' => $user->created_at,
                'updated_at' => $user->updated_at,
            ];
        });

        // Return the users data
        return $this->sendResponse($usersData, 'Users retrieved
successfully.');
```

```

    }

    /**
     * Update the authenticated User.
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function updateUserInfo(Request $request): JsonResponse
    {
        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
        }

        $user = auth()->user();

        $validator = Validator::make($request->all(), [
            'username' =>
'required|string|max:255|unique:users,username,' . $user->id,
            'first_name' => 'required|string|max:255',
            'last_name' => 'required|string|max:255',
            'email' => 'required|email|max:255|unique:users,email,' .
$user->id,
```

```

        'phone' => 'required|digits:10',
        'business_name' => 'nullable|string|max:255',
        'afm' => 'nullable|string',
        'bio' => 'nullable|string',
        'avatar_id' => 'nullable|image'
    ]);

    if($validator->fails()){
        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    try {
        // Start Transaction
        DB::beginTransaction();

        // Handle Image Update
        if ($request->hasFile('avatar_id')) {
            $file = $request->file('avatar_id');

            if ($file->isValid()) {
                Log::info('Handling media upload for avatar_id');
                $avatarId = $this->mediaFileClass::handleMediaUpload($file, 'common/users');
                $user->avatar_id = $avatarId;
                Log::info('Media upload handled successfully,
avatar_id: ' . $avatarId);
            } else {
                Log::error('Uploaded file is not valid.');
```

```

        return $this->sendError('Error occurred during update.',
['error' => $e->getMessage()]);
    }

}

/**
 * Change user's password.
 *
 * @return \Illuminate\Http\JsonResponse
 */
public function changePassword(Request $request): JsonResponse
{
    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }

    $user = auth()->user();

    $validator = Validator::make($request->all(), [
        'current_password' => 'required',
        'new_password' => 'required|min:6',
        'confirm_new_password' =>
'required|min:6|same:new_password',
    ]);

    if($validator->fails()){
        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    // Check if the current password matches
    if (!Hash::check($request->current_password, $user->password))
{
        return $this->sendError('Current password does not match.',
['error' => 'Password mismatch']);
    }

    // Check if the new password is different from the current
password
    if (Hash::check($request->new_password, $user->password)) {
        return $this->sendError('New password must be different
from current password.', ['error' => 'Same password']);
    }

    // Update the password
    $user->password = bcrypt($request->new_password);

```

```

        $user->save();

        return $this->sendResponse([], 'Password changed
successfully.');
```

```

    }

    //Send password reset email if user has forgotten
    public function sendResetLinkEmail(Request $request)
    {
        $request->validate(['email' => 'required|email']);

        $user = User::where('email', $request->email)->first();

        if (!$user) {
            return response()->json(['message' => 'Email not found'],
404);
        }

        $token = Password::createToken($user);

        $resetLink = env('SESSION_DOMAIN') . '/reset-password?token=' .
$token . '&email=' . urlencode($user->email);

        Mail::to($user->email)->send(new PasswordResetEmail($resetLink,
$user->email));

        return response()->json(['message' => 'Reset link sent'], 200);
    }

    //Reset forgotten password
    public function resetForgottenPassword(Request $request)
    {
        $request->validate([
            'token' => 'required',
            'email' => 'required|email',
            'password' => 'required|min:6|confirmed',
        ]);

        $status = Password::reset(
            $request->only('email', 'password',
'password_confirmation', 'token'),
            function ($user, $password) {
                $user->password = Hash::make($password);
                $user->save();

                event(new PasswordReset($user));
            }
        );
    }

```

```

    }
);

return $status === Password::PASSWORD_RESET
    ? response()->json(['message' => __($status)], 200)
    : response()->json(['message' => __($status)], 400);
}

/**
 * Delete authenticated User.
 *
 * @return \Illuminate\Http\JsonResponse
 */
public function deleteUser(): JsonResponse
{
    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }

    $user = auth()->user();

    $hasBookings=FALSE;

    if ($user->hasRole('Vendor')) {
        $bookingsNum=Booking::where('vendor_id', $user->id)-
>whereIn('status',['waiting', 'in_progress'])->count();
        $hotelBookingsNum=HotelRoomBooking::where('vendor_id',
$user->id)->whereIn('status',['waiting', 'in_progress'])->count();

        if ($bookingsNum>0 || $hotelBookingsNum>0){
            $hasBookings=TRUE;
        }
    }

    if ($user->hasRole('Customer')) {
        $bookingsNum=Booking::where('customer_id', $user->id)-
>whereIn('status',['waiting', 'in_progress'])->count();
        $hotelBookingsNum=HotelRoomBooking::where('customer_id',
$user->id)->whereIn('status',['waiting', 'in_progress'])->count();

        if ($bookingsNum>0 || $hotelBookingsNum>0){
            $hasBookings=TRUE;
        }
    }

    if($hasBookings) {

```

```

        return $this->sendError('You cannot delete your account
while you still have ongoing bookings.', ['error' => 'There are still
active bookings']);

    } else {
        if ($user->hasRole('Vendor')) {

            Mail::to($user->email)->send(new
AccountDeletionVendorEmail($user->email));
        }

        // Delete the user
        $user->delete();

        // Return a response indicating successful deletion
        return $this->sendResponse(['Deleted User'], 'User account
successfully deleted');
    }
}

/**
 * Display vendor's details from id.
 */
public function getVendorDetails($id): JsonResponse
{
    $user = User::find($id);

    if ($user){
        if ($user->hasRole('Vendor')) {

            $avatarUrl = $this-
>mediaFileClass::getImageUrlById($user->avatar_id);
            $user->avatarUrl = $avatarUrl;

            return $this->sendResponse($user, 'Vendor`s details
retrieved successfully.');
```

```

    }

}

public function getVendorServices(): JsonResponse
{
    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthorised']);
    }

    $user = auth()->user();

    $response = [];

    $serviceModels = [
        'adventures_tourism' => AdventuresTourism::class,
        'agritourism' => Agritourism::class,
        'air_activities_tourism' => AirActivitiesTourism::class,
        'city_games_tourism' => CityGamesTourism::class,
        'cultural_festival_tourism' =>
CulturalFestivalTourism::class,
        'cultural_urban_exp_tourism' =>
CulturalUrbanExplorationTourism::class,
        'cultural_urban_tourism' => CulturalUrbanTourism::class,
        'dark_tourism' => DarkTourism::class,
        'ecotourism' => Ecotourism::class,
        'education_development_tourism' =>
EducationDevelopmentTourism::class,
        'enrichment_skill_building_tourism' =>
EnrichmentSkillBuildingTourism::class,
        'exploration_tourism' => ExplorationTourism::class,
        'fishing_tourism' => FishingTourism::class,
        'gastronomic_tourism' => GastronomicTourism::class,
        'geotourism' => Geotourism::class,
        'sacred_site_visits_tourism' =>
SacredSiteVisitsTourism::class,
        'sporting_events_tourism' => SportingEventsTourism::class,
        'urban_entertainment_tourism'=>
UrbanEntertainmentTourism::class,
        'water_activities_tourism' =>
WaterActivitiesTourism::class,
        'wedding_tourism' => WeddingTourism::class,
        'wine_tourism' => WineTourism::class,
        'medical_tourism' => MedicalTourism::class,
        'spa_tourism' => SpaTourism::class,
    ];
}

```

```

        'wellness_tourism' => WellnessTourism::class,
    ];

    $serviceModelsResources = [
        'adventures_tourism' => AdventuresTourismResource::class,
        'agritourism' => AgritourismResource::class,
        'air_activities_tourism' =>
AirActivitiesTourismResource::class,
        'city_games_tourism' => CityGamesTourismResource::class,
        'cultural_festival_tourism' =>
CulturalFestivalTourismResource::class,
        'cultural_urban_exp_tourism' =>
CulturalUrbanExplorationTourismResource::class,
        'cultural_urban_tourism' =>
CulturalUrbanTourismResource::class,
        'dark_tourism' => DarkTourismResource::class,
        'ecotourism' => EcotourismResource::class,
        'education_development_tourism' =>
EducationDevelopmentTourismResource::class,
        'enrichment_skill_building_tourism' =>
EnrichmentSkillBuildingTourismResource::class,
        'exploration_tourism' => ExplorationTourismResource::class,
        'fishing_tourism' => FishingTourismResource::class,
        'gastronomic_tourism' => GastronomicTourismResource::class,
        'geotourism' => GeotourismResource::class,
        'sacred_site_visits_tourism' =>
SacredSiteVisitsTourismResource::class,
        'sporting_events_tourism' =>
SportingEventsTourismResource::class,
        'urban_entertainment_tourism'=>
UrbanEntertainmentTourismResource::class,
        'water_activities_tourism' =>
WaterActivitiesTourismResource::class,
        'wedding_tourism' => WeddingTourismResource::class,
        'wine_tourism' => WineTourismResource::class,
        'medical_tourism' => MedicalTourismResource::class,
        'spa_tourism' => SpaTourismResource::class,
        'wellness_tourism' => WellnessTourismResource::class,
    ];

    foreach ($serviceModels as $serviceName=>$serviceModel) {
        $services = $serviceModel::where('create_user', $user->id)-
>get();
        if ($services->isEmpty()) {
            $response[$serviceName] =
$serviceModelsResources[$serviceName]::collection($services);
        }
    }
}

```

```

        return response()->json([
            'success' => true,
            'data' => $response,
        ], Response::HTTP_OK);
    }

    public function getVendorServicesForProfile($id): JsonResponse
    {
        $user = User::find($id);

        $response = [];

        $serviceModels = [
            'adventures_tourism' => AdventuresTourism::class,
            'agritourism' => Agritourism::class,
            'air_activities_tourism' => AirActivitiesTourism::class,
            'city_games_tourism' => CityGamesTourism::class,
            'cultural_festival_tourism' =>
CulturalFestivalTourism::class,
            'cultural_urban_exp_tourism' =>
CulturalUrbanExplorationTourism::class,
            'cultural_urban_tourism' => CulturalUrbanTourism::class,
            'dark_tourism' => DarkTourism::class,
            'ecotourism' => Ecotourism::class,
            'education_tourism' => EducationDevelopmentTourism::class,
            'enrichment_tourism' =>
EnrichmentSkillBuildingTourism::class,
            'exploration_tourism' => ExplorationTourism::class,
            'fishing_tourism' => FishingTourism::class,
            'gastronomic_tourism' => GastronomicTourism::class,
            'geotourism' => Geotourism::class,
            'sacred_site_visits_tourism' =>
SacredSiteVisitsTourism::class,
            'sporting_events_tourism' => SportingEventsTourism::class,
            'urban_entertainment_tourism'=>
UrbanEntertainmentTourism::class,
            'water_activities_tourism' =>
WaterActivitiesTourism::class,
            'wedding_tourism' => WeddingTourism::class,
            'wine_tourism' => WineTourism::class,
            'medical_tourism' => MedicalTourism::class,
            'spa_tourism' => SpaTourism::class,
            'wellness_tourism' => WellnessTourism::class,
        ];
    }

```

```

        $serviceModelsResources = [
            'adventures_tourism' => AdventuresTourismResource::class,
            'agritourism' => AgritourismResource::class,
            'air_activities_tourism' =>
AirActivitiesTourismResource::class,
            'city_games_tourism' => CityGamesTourismResource::class,
            'cultural_festival_tourism' =>
CulturalFestivalTourismResource::class,
            'cultural_urban_exp_tourism' =>
CulturalUrbanExplorationTourismResource::class,
            'cultural_urban_tourism' =>
CulturalUrbanTourismResource::class,
            'dark_tourism' => DarkTourismResource::class,
            'ecotourism' => EcotourismResource::class,
            'education_tourism' =>
EducationDevelopmentTourismResource::class,
            'enrichment_tourism' =>
EnrichmentSkillBuildingTourismResource::class,
            'exploration_tourism' => ExplorationTourismResource::class,
            'fishing_tourism' => FishingTourismResource::class,
            'gastronomic_tourism' => GastronomicTourismResource::class,
            'geotourism' => GeotourismResource::class,
            'sacred_site_visits_tourism' =>
SacredSiteVisitsTourismResource::class,
            'sporting_events_tourism' =>
SportingEventsTourismResource::class,
            'urban_entertainment_tourism' =>
UrbanEntertainmentTourismResource::class,
            'water_activities_tourism' =>
WaterActivitiesTourismResource::class,
            'wedding_tourism' => WeddingTourismResource::class,
            'wine_tourism' => WineTourismResource::class,
            'medical_tourism' => MedicalTourismResource::class,
            'spa_tourism' => SpaTourismResource::class,
            'wellness_tourism' => WellnessTourismResource::class,
        ];

        foreach ($serviceModels as $serviceName=>$serviceModel) {
            $services = $serviceModel::where('create_user', $user->id)-
>get();
            if ($services->isEmpty()) {
                $response[$serviceName] =
                $serviceModelsResources[$serviceName]::collection($services);
            }
        }

        return response()->json([
            'success' => true,

```

```
        'data' => $response,  
    ], Response::HTTP_OK);  
  
    }  
  
}
```

VendorController Documentation

Ο VendorController είναι υπεύθυνος για τη διαχείριση των λειτουργιών που σχετίζονται με τους Vendors (πάροχοι). Αυτές περιλαμβάνουν την εγγραφή νέων Vendors και την αποστολή email επιβεβαίωσης. Ο controller επεκτείνει το BaseController για τυποποιημένες απαντήσεις.

Εξαρτήσεις (Dependencies)

- **BaseController:** Βασικός controller για τυποποιημένα responses.
- **User Model:** Χρησιμοποιείται για τη διαχείριση των Vendors στη βάση δεδομένων.
- **Validator:** Για την επικύρωση εισόδου.
- **Mail:** Για την αποστολή email επιβεβαίωσης.

Μέθοδοι

1. register

- **Σκοπός:** Εγγραφή νέου Vendor.
- **Παράμετροι:**
 - username: Το όνομα χρήστη (μοναδικό).
 - first_name: Το όνομα του Vendor.
 - last_name: Το επώνυμό του.
 - business_name: Το όνομα της επιχείρησης.
 - email: Το email του Vendor (μοναδικό).
 - phone: Το τηλέφωνό του.
 - password: Ο κωδικός πρόσβασης.
 - c_password: Επιβεβαίωση κωδικού πρόσβασης.
 - roles: Ο ρόλος του Vendor (π.χ. Vendor).

- **Λειτουργία:**
 - ο Επικυρώνει τα δεδομένα εισόδου.
 - ο Δημιουργεί έναν νέο Vendor στη βάση δεδομένων.
 - ο Αντιστοιχεί τον ρόλο "Vendor" στον χρήστη.
 - ο Στέλνει email επιβεβαίωσης στον Vendor.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα επικύρωσης.

```
<?php

namespace App\Http\Controllers\API;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Validator;
use Illuminate\Http\JsonResponse;

use Illuminate\Support\Facades\Mail;
use App\Mail\SignupEmail;
use App\Mail\VendorSignupEmail;

class VendorController extends BaseController
{
    /**
     * Register vendor api
     *
     * @return \Illuminate\Http\Response
     */
    public function register(Request $request): JsonResponse
    {
        $validator = Validator::make($request->all(), [
            'username' => 'required|string|unique:users|max:255',
            'first_name' => 'required|string',
            'last_name' => 'required|string',
            'business_name' => 'required|string',
            'email' => 'required|email|unique:users',
            'phone' => 'required|digits:10',
            'password' => 'required',
            'c_password' => 'required|same:password',
            'roles' => 'required',

        ]);
```

```

        if($validator->fails()){
            return $this->sendError('Validation Error.', $validator-
>errors());
        }

        // $input = $request->all();
        // $input['name'] = $input['first_name'] . ' ' .
$input['last_name'];
        // $input['password'] = bcrypt($input['password']);

        $user = User::create([
            'username' => $request->input('username'),
            'first_name' => $request->input('first_name'),
            'last_name' => $request->input('last_name'),
            'business_name' => $request->input('business_name'),
            'email' => $request->input('email'),
            'password' => bcrypt($request->input('password')),
            'publish' => $request->input('publish'),
            'phone' => $request->input('phone'),
        ]);

        $user->assignRole($request->input('roles'));

        $recipient = $request->input('email');
        $userName = $request->input('username');
        $firstName = $request->input('first_name');
        Mail::to($recipient)->send(new VendorSignupEmail($firstName,
$recipient));

        // $success['token'] = $user->createToken('authToken')-
>plainTextToken;
        // $success['username'] = $user->name;

        return $this->sendResponse('Registration Succeeded', 'Vendor
register successfully.');
```

AdventuresTourismController Documentation

Επισκόπηση

Ο `AdventuresTourismController` είναι υπεύθυνος για τη διαχείριση των λειτουργιών που σχετίζονται με `Adventures Tourism`. Αυτές περιλαμβάνουν τη δημιουργία, ενημέρωση, διαγραφή και ανάκτηση καταχωρήσεων.

Εξαρτήσεις (Dependencies)

- **BaseController**: Βασικός controller για τυποποιημένα responses.
- **AdventuresTourism Model**: Χρησιμοποιείται για τη διαχείριση των καταχωρήσεων `Adventures Tourism` στη βάση δεδομένων.
- **MediaFileController**: Για τη διαχείριση μεταφορτώσεων αρχείων (εικόνες, πιστοποιητικά κ.λπ.).
- **Validator**: Για την επικύρωση εισόδου.
- **Auth**: Για τη διαχείριση πιστοποίησης.
- **DB**: Για τη διαχείριση των transactions.

Μέθοδοι

1. index

- **Σκοπός**: Ανάκτηση όλων των καταχωρήσεων `Adventures Tourism`.
- **Λειτουργία**:
 - Επιστρέφει όλες τις καταχωρήσεις `Adventures Tourism`.
- **Response**: Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.

2. getVendorAdventuresTourism

- **Σκοπός**: Ανάκτηση καταχωρήσεων `Adventures Tourism` για έναν συγκεκριμένο `Vendor`.
- **Παράμετροι**:
 - `id`: Το ID του `Vendor`.
- **Λειτουργία**:
 - Επιστρέφει όλες τις καταχωρήσεις `Adventures Tourism` που δημιουργήθηκαν από τον `Vendor`.
- **Response**: Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.

3. store

- **Σκοπός:** Δημιουργία νέας καταχώρησης Adventures Tourism.
- **Παράμετροι:**
 - Πολλές παράμετροι, όπως title, image_id, map_lat, map_lng, address, price_per_person, κ.λπ.
- **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Διαχειρίζεται τη μεταφόρτωση εικόνων και πιστοποιητικών.
 - Δημιουργεί μια νέα καταχώρηση Adventures Tourism στη βάση δεδομένων.
- **Response:** Τα στοιχεία της νέας καταχώρησης ή σφάλμα επικύρωσης.

4. show

- **Σκοπός:** Ανάκτηση συγκεκριμένης καταχώρησης Adventures Tourism.
- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
- **Λειτουργία:**
 - Επιστρέφει τα στοιχεία της καταχώρησης.
- **Response:** Στοιχεία καταχώρησης ή σφάλμα αν δεν βρεθεί.

5. update

- **Σκοπός:** Ενημέρωση υπάρχουσας καταχώρησης Adventures Tourism.
- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - Πολλές παράμετροι, όπως title, image_id, map_lat, map_lng, address, price_per_person, κ.λπ.
- **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Διαχειρίζεται τη μεταφόρτωση εικόνων και πιστοποιητικών.
 - Ενημερώνει την καταχώρηση στη βάση δεδομένων.
- **Response:** Τα ενημερωμένα στοιχεία της καταχώρησης ή σφάλμα επικύρωσης.

6. destroy

- **Σκοπός:** Διαγραφή καταχώρησης Adventures Tourism.

- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
- **Λειτουργία:**
 - Διαγράφει την καταχώρηση από τη βάση δεδομένων.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

7. duplicate

- **Σκοπός:** Αντιγραφή υπάρχουσας καταχώρησης Adventures Tourism.
- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
- **Λειτουργία:**
 - Αντιγράφει την καταχώρηση και τις σχετικές εικόνες/πιστοποιητικά.
 - Αποθηκεύει την αντιγραφή ως νέα καταχώρηση.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

***** Η ΙΔΙΑ ΔΟΜΗ ΧΡΗΣΙΜΟΙΕΙΤΑΙ ΚΑΙ ΣΕ ΟΛΟΥΣ ΤΟΥΣ ΥΠΟΛΟΙΠΟΥΣ CONTROLLERS ΤΩΝ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΠΟΥ ΥΠΑΡΧΟΥΝ ΣΤΗΝ ΟΝΤΟΛΟΓΙΑ *****

```
<?php

namespace Modules\AdventuresTourism\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use Modules\AdventuresTourism\Resources\AdventuresTourismResource;
use App\Models\User;
use Modules\MediaFile\Controllers\MediaFileController;
use Modules\AdventuresTourism\Models\AdventuresTourism;
use Modules\AdventuresTourism\Models\AdventuresTourismDate;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Validator;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\DB;

class AdventuresTourismController extends BaseController
```

```

{

    protected $mediaFileClass;

    /**
     * Set the permissions
     *
     */
    function __construct(array $attributes = [])
    {
        $this->mediaFileClass = MediaFileController::class;
        $this->middleware(['auth:sanctum', 'permission:edit-service'],
['only' => ['store']]);
        $this->middleware(['auth:sanctum', 'permission:edit-service'],
['only' => ['update']]);
        $this->middleware(['auth:sanctum', 'permission:delete-
service'], ['only' => ['destroy']]);
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(): JsonResponse
    {
        // Get all AdventuresTourism entries
        $adventuresTourismEntries = AdventuresTourism::all();

        return $this-
>sendResponse(AdventuresTourismResource::collection($adventuresTourismE
ntries), 'Adventures Tourism entries retrieved successfully.');
```

```

}

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request): JsonResponse
{
    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }

    $user = auth()->user();

    $input = $request->all();

    // Validate the request data
    $validator = Validator::make($input, [
        'tourism_sub_category' => 'required|string|max:255',
        'tourism_sub_category_4' => 'required|string|max:255',
        'title' => 'required|string|max:255',
        'image_id' => 'required|image',
        'banner_image_id' => 'nullable|image',

        // 'category_id' =>
'nullable|integer|exists:categories,id',
        // 'location_id' => 'nullable|integer|exists:locations,id',
        'map_lat' => 'required|string',
        'map_lng' => 'required|string',

        'address' => 'required|string',
        'website_link' => 'nullable|string',
        'meeting_point_address' => 'nullable|string',
        'youtube_url' => 'nullable|string',
        'address_and_description' => 'required|string',
        'primary_town' => 'required|string',
        'additional_towns' => 'nullable|string',
        'gallery.*' => 'required|image',
        'status' => 'required|string',
        'short_description' => 'required|string',
        'program_details' => 'required|string',
        'detail_extra_questions' => 'nullable|array',
        'necessary_items' => 'nullable|string',
        'dress_code' => 'nullable|string',

```

```

        'important_booking_info' => 'nullable|string',
        'important_extra_questions' => 'nullable|array',
        'duration' => 'required|integer',
        'duration_meter' => 'required|string',
        'availability_type' => 'required|string',
        'open_hours' => 'nullable|array',
        'languages' => 'required|array',
        'services_included' => 'nullable|array',
        'price_per_person' => 'required|numeric',
        'sale_price' => 'nullable|numeric',
        'enable_extra_price' => 'nullable|boolean',
        'extra_price' => 'nullable|array',
        'max_capacity' => 'required|integer',
        'min_number_of_booking' => 'nullable|integer',
        'min_number_duration_meter' => 'nullable|string',
        'max_number_of_booking' => 'nullable|integer',
        'max_number_duration_meter' => 'nullable|string',
        'person_types' => 'nullable|array',

        'is_refundable' => 'required|boolean',
        'full_refund_time' => 'nullable|string',
        'partial_refund_time' => 'nullable|string',
        'partial_refund_time_percentage' => 'nullable|integer',
        'no_refund_after' => 'nullable|string',

        'restrictions' => 'required|array',
        'tags' => 'nullable|array',
        'has_guide' => 'required|boolean',
        'is_verified_guide' => 'nullable|boolean',
        'guide_certification_file' => 'nullable|file|mimes:pdf',
        'is_sustainable' => 'nullable|boolean',
        'sustainability_certification_file' =>
'nullable|file|mimes:pdf',
        'min_day_before_booking' => 'nullable|integer',
        'min_day_before_booking_duration_meter' =>
'nullable|string',
        'availability_state' => 'nullable|boolean',
        'start_time' => 'nullable|string',
        'max_age' => 'nullable|integer',
        'min_age' => 'nullable|integer',
        'review_score' => 'nullable|numeric',
        'exclude' => 'nullable|array',
        // ... other fields as needed
    ]);

    // If validation fails, return an error response
    if ($validator->fails()) {

```

```

        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    try {
        // Start Transaction
        DB::beginTransaction();

        if ($request->hasFile('image_id')) {
            $input['image_id'] = $this-
>mediaFileClass::handleMediaUpload($request->file('image_id'),
'services/adventures-tourism');
        }

        if ($request->hasFile('banner_image_id')) {
            $input['banner_image_id'] = $this-
>mediaFileClass::handleMediaUpload($request->file('banner_image_id'),
'services/adventures-tourism');
        }

        if ($request->hasFile('gallery')) {
            $galleryImages = $request->file('gallery');
            $galleryIds = [];

            foreach ($galleryImages as $image) {
                $galleryIds[] = $this-
>mediaFileClass::handleMediaUpload($image, 'services/adventures-
tourism');
            }

            $input['gallery'] = json_encode($galleryIds);
        }

        // Handle guide certification file upload if
is_verified_guide is true
        if ($request->input('is_verified_guide') && $request-
>hasFile('guide_certification_file')) {
            $input['guide_certification_id'] = $this-
>mediaFileClass::handleMediaUpload($request-
>file('guide_certification_file'), 'guide-certifications');
        }

        // Handle sustainability certification file upload if
is_sustainable is true
        if ($request->input('is_sustainable') && $request-
>hasFile('sustainability_certification_file')) {

```

```

        $input['sustainability_cerification_id'] = $this->mediaFileClass::handleMediaUpload($request->file('sustainability_certification_file'), 'sustainability-certifications');

    }

    unset($input['guide_certification_file']);
    unset($input['sustainability_certification_file']);

    // Encode fields that should be stored as JSON
    $fieldsToEncode = [
        'gallery',
        'detail_extra_questions',
        'important_extra_questions',
        'open_hours',
        'languages',
        'services_included',
        'extra_price',
        'person_types',
        'restrictions',
        'tags',
        'exclude',
    ];

    // Modify this part of your Laravel code
    foreach ($fieldsToEncode as $field) {
        if (isset($input[$field]) &&
!is_string($input[$field])) {
            // Only encode as JSON if it's not already a JSON
string
            $input[$field] = json_encode($input[$field]);
        }
    }

    $input['create_user'] = $user->id;
    $input['author_id'] = $user->id;

    $adventuresTourism = AdventuresTourism::create($input);

    // Commit the transaction
    DB::commit();

    return $this->sendResponse(new
AdventuresTourismResource($adventuresTourism), 'Adventures Tourism
created successfully.');
```

```

    } catch (\Exception $e) {
        // Rollback the transaction in case of an error
        DB::rollBack();

        // Return an error response
        return $this->sendError('Error occurred during creation.',
['error' => $e->getMessage()]);
    }
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id): JsonResponse
{
    $adventuresTourism = AdventuresTourism::find($id);

    if (is_null($adventuresTourism)) {
        return $this->sendError('Adventures Tourism not found.');
```

```

    }

    return $this->sendResponse(new
AdventuresTourismResource($adventuresTourism), 'Adventures Tourism
retrieved successfully.');
```

```

}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, AdventuresTourism
$adventuresTourism): JsonResponse
{
```

```

    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }
}
```

```

$this->authorize('update', $adventuresTourism);

$input = $request->all();

$newPersonType = isset($input['person_types']) ?
$input['person_types'] : null;
$currentPersonType = json_decode($adventuresTourism-
>person_types, true);

// Validate the request data
$validator = Validator::make($input, [
    'tourism_sub_category' => 'required|string|max:255',
    'tourism_sub_category_4' => 'required|string|max:255',
    'title' => 'required|string|max:255',
    'image_id' => 'nullable|image|max:5120',
    'banner_image_id' => 'nullable|image',

    // 'category_id' =>
'nullable|integer|exists:categories,id',
    // 'location_id' => 'nullable|integer|exists:locations,id',
    'map_lat' => 'required|string',
    'map_lng' => 'required|string',

    'address' => 'required|string',
    'website_link' => 'nullable|string',
    'meeting_point_address' => 'nullable|string',
    'youtube_url' => 'nullable|string',
    'address_and_description' => 'required|string',
    'primary_town' => 'required|string',
    'additional_towns' => 'nullable|string',
    'gallery.*' => 'nullable|image|max:5120',
    'status' => 'required|string',
    'short_description' => 'required|string',
    'program_details' => 'required|string',
    'detail_extra_questions' => 'nullable|array',
    'necessary_items' => 'nullable|string',
    'dress_code' => 'nullable|string',
    'important_booking_info' => 'nullable|string',
    'important_extra_questions' => 'nullable|array',
    'duration' => 'required|integer',
    'duration_meter' => 'required|string',
    'availability_type' => 'required|string',
    'open_hours' => 'nullable|array',
    'languages' => 'required|array',
    'services_included' => 'nullable|array',
    'price_per_person' => 'required|numeric',
    'sale_price' => 'nullable|numeric',
    'enable_extra_price' => 'nullable|boolean',

```

```

        'extra_price' => 'nullable|array',
        'max_capacity' => 'required|integer',
        'min_number_of_booking' => 'nullable|integer',
        'min_number_duration_meter' => 'nullable|string',
        'max_number_of_booking' => 'nullable|integer',
        'max_number_duration_meter' => 'nullable|string',
        'person_types' => 'nullable|array',
        'is_refundable' => 'required|boolean',
        'full_refund_time' => 'nullable|string',
        'partial_refund_time' => 'nullable|string',
        'partial_refund_time_percentage' => 'nullable|integer',
        'no_refund_after' => 'nullable|string',
        'restrictions' => 'required|array',
        'tags' => 'nullable|array',
        'has_guide' => 'required|boolean',
        'is_verified_guide' => 'nullable|boolean',
        'guide_certification_file' => 'nullable|file|mimes:pdf',
        'is_sustainable' => 'nullable|boolean',
        'sustainability_certification_file' =>
'nullable|file|mimes:pdf',
        'min_day_before_booking' => 'nullable|integer',
        'min_day_before_booking_duration_meter' =>
'nullable|string',
        'availability_state' => 'nullable|boolean',
        'start_time' => 'nullable|string',
        'max_age' => 'nullable|integer',
        'min_age' => 'nullable|integer',
        'review_score' => 'nullable|numeric',
        'exclude' => 'nullable|array',
        // ... other fields as needed
    ]);

    // If validation fails, return an error response
    if ($validator->fails()) {
        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    try {
        // Start Transaction
        DB::beginTransaction();

        // Check if person_types has changed
        if (!is_null($newPersonType) && $newPersonType[0]['type']
!= $currentPersonType[0]['type']) {
            // Delete dates records
            AdventuresTourismDate::where('target_id',
$adventuresTourism->id)->delete();

```

```

    }

    // Handle Image Update
    if ($request->hasFile('image_id')) {
        $file = $request->file('image_id');

        if ($file->isValid()) {

            $input['image_id'] = $this->mediaFileClass::updateMediaFile($adventuresTourism->image_id, $file,
'services/adventures-tourism');

        }
    } else {
        unset($input['image_id']);
    }

    if ($request->hasFile('banner_image_id')) {
        $file = $request->file('banner_image_id');

        if ($file->isValid()) {

            $input['banner_image_id'] = $this->mediaFileClass::updateMediaFile($adventuresTourism->banner_image_id,
$file, 'services/adventures-tourism');

        }
    } else {
        unset($input['banner_image_id']);
    }

    if ($request->hasFile('gallery')) {

        $this->mediaFileClass::deleteGallery($adventuresTourism->gallery);

        $galleryImages = $request->file('gallery');
        $galleryIds = [];

        foreach ($galleryImages as $image) {
            if ($image->isValid()) {

                $galleryIds[] = $this->mediaFileClass::handleMediaUpload($image, 'services/adventures-
tourism');

            }
        }
    }
}

```

```

    }

    $input['gallery'] = json_encode($galleryIds);
} else {
    unset($input['gallery']);
}

// Handle Guide Certification Update
if ($request->hasFile('guide_certification_file')) {
    $file = $request->file('guide_certification_file');

    if ($file->isValid()) {
        if ($adventuresTourism->guide_certification_id) {

            $input['guide_certification_id'] = $this->mediaFileClass::updateMediaFile($adventuresTourism->guide_certification_id, $file, 'guide-certifications');
        } else {
            $input['guide_certification_id'] = $this->mediaFileClass::handleMediaUpload($request->file('guide_certification_file'), 'guide-certifications');
        }
    }
}

// Handle Sustainability Certification Update
if ($request->hasFile('sustainability_certification_file'))
{
    $file = $request->file('sustainability_certification_file');

    if ($file->isValid()) {
        if ($adventuresTourism->sustainability_cerification_id) {
            $input['sustainability_cerification_id'] =
$this->mediaFileClass::updateMediaFile($adventuresTourism->sustainability_cerification_id, $file, 'sustainability-certifications');
        } else {
            $input['sustainability_cerification_id'] =
$this->mediaFileClass::handleMediaUpload($request->file('sustainability_certification_file'), 'sustainability-certifications');
        }
    }
}
}

```

```

        unset($input['guide_certification_file']);
        unset($input['sustainability_certification_file']);

        // Encode fields that should be stored as JSON
        $fieldsToEncode = [
            'gallery',
            'detail_extra_questions',
            'important_extra_questions',
            'open_hours',
            'languages',
            'services_included',
            'extra_price',
            'person_types',
            'restrictions',
            'tags',
            'exclude',
        ];

        // Modify this part of your Laravel code
        foreach ($fieldsToEncode as $field) {
            if (isset($input[$field]) &&
                !is_string($input[$field])) {
                // Only encode as JSON if it's not already a JSON
                string
                $input[$field] = json_encode($input[$field]);
            }
        }

        $input['update_user'] = auth()->id();

        $adventuresTourism->update($input);

        // Commit the transaction
        DB::commit();

        return $this->sendResponse(new
        AdventuresTourismResource($adventuresTourism), 'Adventures Tourism
        updated successfully.');
```

```

    } catch (\Exception $e) {
        // Rollback the transaction in case of an error
        DB::rollBack();

        // Return an error response
        return $this->sendError('Error occurred during update.',
        ['error' => $e->getMessage()]);
    }
}

```

```

    }

    /**
     * Remove the specified resource from storage.
     *
     * @param int $id
     * @return \Illuminate\Http\Response
     */
    public function destroy(AdventuresTourism $adventuresTourism):
    JsonResponse
    {
        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
        }

        $this->authorize('delete', $adventuresTourism);

        $adventuresTourism->delete();

        return $this->sendResponse([], 'Adventures Tourism deleted
successfully.');
```

```

    }

    public function duplicate($id): JsonResponse
    {
        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
        }

        $tourism = AdventuresTourism::find($id);

        if (is_null($tourism)) {
            return $this->sendError('Adventures Tourism not found.');
```

```

        }

        try {
            DB::beginTransaction();

            $newService = $tourism->replicate();

            //primary image

```

```

        $newImageId = $this->
>mediaFileClass::duplicateMediaFile($tourism->image_id,
'services/adventures-tourism');
        $newService->image_id = $newImageId;

        //gallery
        $newGalleryIds = [];
        foreach (json_decode($tourism->gallery) as $imageId) {
            $newGalleryIds[] = $this->
>mediaFileClass::duplicateMediaFile($imageId, 'services/adventures-
tourism');
        }

        $newService->gallery = json_encode($newGalleryIds);

        //guide certification
        if ($tourism->guide_certification_id)
        {
            $newGuideCertificationId = $this->
>mediaFileClass::duplicateMediaFile($tourism->guide_certification_id,
'guide-certifications');
            $newService->guide_certification_id =
$newGuideCertificationId;
        }

        //sustainability certification
        if ($tourism->sustainability_certification_id)
        {
            $newSustainabilityCertificationId = $this->
>mediaFileClass::duplicateMediaFile($tourism->
>sustainability_certification_id, 'sustainability-certifications');
            $newService->sustainability_certification_id =
$newSustainabilityCertificationId;
        }

        $newService->save();

        DB::commit();
        return $this->sendResponse('Adventures Tourism duplicated
successfully.', 200);
    } catch (\Exception $e) {

        DB::rollBack();
        return $this->sendError('Error occurred during duplication.',
['error' => $e->getMessage()]);
    }
}

```

```
}
```

AdventuresTourismDatesController Documentation

Επισκόπηση

Ο AdventuresTourismDatesController είναι υπεύθυνος για τη διαχείριση των ημερομηνιών και των διαθεσιμοτήτων για τα Adventures Tourism. Αυτές περιλαμβάνουν τη δημιουργία, ενημέρωση, διαγραφή και ανάκτηση ημερομηνιών για συγκεκριμένες Adventures Tourism.

Εξαρτήσεις (Dependencies)

- **BaseController**: Βασικός controller για τυποποιημένα responses.
- **AdventuresTourismDate Model**: Χρησιμοποιείται για τη διαχείριση των ημερομηνιών Adventures Tourism στη βάση δεδομένων.
- **Validator**: Για την επικύρωση εισόδου.
- **Auth**: Για τη διαχείριση πιστοποίησης.
- **DB**: Για τη διαχείριση transactions.

Μέθοδοι

1. index

- **Σκοπός**: Ανάκτηση όλων των καταχωρήσεων ημερομηνιών Adventures Tourism.
- **Λειτουργία**:
 - Επιστρέφει όλες τις καταχωρήσεις ημερομηνιών Adventures Tourism.
- **Response**: Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.

2. getAdventuresTourismDates

- **Σκοπός**: Ανάκτηση ημερομηνιών για μια συγκεκριμένη περιήγηση περιπέτειας εντός ενός εύρους ημερομηνιών.
- **Παράμετροι**:
 - adventuresTourismId: Το ID της περιήγησης.
 - start_date: Η αρχική ημερομηνία.

- end_date: Η τελική ημερομηνία.
 - **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις ημερομηνιών για τη συγκεκριμένη περιήγηση εντός του καθορισμένου εύρους ημερομηνιών.
 - **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.
-

3. getMultipleAdventuresTourismDates

- **Σκοπός:** Ανάκτηση ημερομηνιών για πολλαπλές περιηγήσεις περιπέτειας εντός ενός εύρους ημερομηνιών.
 - **Παράμετροι:**
 - tourism_ids: Λίστα με τα IDs των περιηγήσεων.
 - start_date: Η αρχική ημερομηνία.
 - end_date: Η τελική ημερομηνία.
 - **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις ημερομηνιών για τις συγκεκριμένες περιηγήσεις εντός του καθορισμένου εύρους ημερομηνιών.
 - **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.
-

4. manageAdventuresTourismDates

- **Σκοπός:** Διαχείριση ημερομηνιών για μια περιήγηση περιπέτειας.
- **Παράμετροι:**
 - target_id: Το ID της περιήγησης.
 - start_date: Η αρχική ημερομηνία.
 - end_date: Η τελική ημερομηνία.
 - price: Η τιμή ανά άτομο.
 - person_types: Οι τύποι ατόμων (π.χ. ενήλικες, παιδιά).
 - max_capacity: Η μέγιστη χωρητικότητα.
 - min_number_of_booking: Ο ελάχιστος αριθμός κρατήσεων.
 - active: Κατάσταση διαθεσιμότητας.
 - availability_type: Τύπος διαθεσιμότητας (π.χ. ανά ημέρα, ανά ώρα).
 - open_days_hours: Οι ώρες λειτουργίας για κάθε ημέρα.
- **Λειτουργία:**

- Επικυρώνει τα δεδομένα εισόδου.
 - Διαγράφει τις υπάρχουσες καταχωρήσεις ημερομηνιών για το καθορισμένο εύρος.
 - Δημιουργεί νέες καταχωρήσεις ημερομηνιών για κάθε ημέρα και ώρα που έχει οριστεί.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.
-

5. show

- **Σκοπός:** Ανάκτηση συγκεκριμένης καταχώρησης ημερομηνίας.
 - **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - **Λειτουργία:**
 - Επιστρέφει τα στοιχεία της καταχώρησης.
 - **Response:** Στοιχεία καταχώρησης ή σφάλμα αν δεν βρεθεί.
-

6. update

- **Σκοπός:** Ενημέρωση υπάρχουσας καταχώρησης ημερομηνίας.
 - **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - target_id: Το ID της περιήγησης.
 - start_date: Η αρχική ημερομηνία.
 - end_date: Η τελική ημερομηνία.
 - price: Η τιμή ανά άτομο.
 - number: Ο αριθμός διαθεσιμότητας.
 - active: Κατάσταση διαθεσιμότητας.
 - **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Ενημερώνει την καταχώρηση στη βάση δεδομένων.
 - **Response:** Τα ενημερωμένα στοιχεία της καταχώρησης ή σφάλμα επικύρωσης.
-

7. destroy

- **Σκοπός:** Διαγραφή καταχώρησης ημερομηνίας.

- **Παράμετροι:**
 - ο id: Το ID της καταχώρησης.
- **Λειτουργία:**
 - ο Διαγράφει την καταχώρηση από τη βάση δεδομένων.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

***** Η ΙΔΙΑ ΔΟΜΗ ΧΡΗΣΙΜΟΙΕΙΤΑΙ ΚΑΙ ΣΕ ΟΛΟΥΣ ΤΟΥΣ ΥΠΟΛΟΙΠΟΥΣ CONTROLLERS ΤΩΝ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΠΟΥ ΥΠΑΡΧΟΥΝ ΣΤΗΝ ΟΝΤΟΛΟΓΙΑ *****

```
<?php

namespace Modules\AdventuresTourism\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use Modules\AdventuresTourism\Resources\AdventuresTourismDatesResource;
use App\Models\User;
use Modules\AdventuresTourism\Models\AdventuresTourismDate;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Validator;
use Illuminate\Http\JsonResponse;
use Carbon\Carbon;
use Carbon\CarbonPeriod;
use Illuminate\Support\Facades\DB;

class AdventuresTourismDatesController extends BaseController
{
    /**
     * Set the permissions
     *
     */
    function __construct()
    {
        $this->middleware('permission:create-service', ['only' =>
['store']]);
        $this->middleware('permission:edit-service', ['only' =>
['update']]);
        $this->middleware('permission:delete-service', ['only' =>
['destroy']]);
    }
}
```

```

    }

    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        // Get all AdventuresTourismDate entries
        $adventuresTourismDates = AdventuresTourismDate::all();

        return $this->sendResponse(AdventuresTourismDatesResource::collection($adventuresTourismDates), 'AdventuresTourismDate entries retrieved successfully.');
```

```

    }

    /**
     * Display AdventuresTourismDates for a specific gstronomic tourism
     service within a date range.
     */
    public function getAdventuresTourismDates(Request $request,
        $adventuresTourismId): JsonResponse
    {
        // Get all AdventuresTourismDate entries for a specific Room

        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
                ['error'=>'Unauthored']);
        }

        $user = auth()->user();

        $input = $request->all();

        // Validate the request data
        $validator = Validator::make($input, [
            'start_date' => 'required|date',
            'end_date' => 'required|date|after_or_equal:start_date',
        ]);

        $existingDatesForThisMonth =
        AdventuresTourismDate::where('target_id', $adventuresTourismId)
            ->whereBetween('start_date',
                [$input['start_date'], $input['end_date']])
            ->get();

        return $this->sendResponse(AdventuresTourismDatesResource::collection($existingDates

```

```

ForThisMonth), ' AdventuresTourismDate entries retrieved
successfully.');
```

```

    }

    public function getMultipleAdventuresTourismDates(Request
$request): JsonResponse
    {
        // Get all AdventuresTourismDate entries for multiple tourisms
        //No auth since this is used for search

        $input = $request->all();

        // Validate the request data
        $validator = Validator::make($input, [
            'start_date' => 'required|date',
            'end_date' => 'required|date|after_or_equal:start_date',
            'tourism_ids' => 'required|array'
        ]);

        $adventuresTourismIdsArray = $input['tourism_ids'];

        $existingDatesForThisMonth =
AdventuresTourismDate::whereIn('target_id', $adventuresTourismIdsArray)
    ->whereBetween('start_date', [$input['start_date'],
$input['end_date']])
    ->get();

        return $this-
>sendResponse(AdventuresTourismDatesResource::collection($existingDates
ForThisMonth), ' AdventuresTourismDate entries retrieved successfully
for services.');
```

```

    }

    /**
     * Store a newly created resource in storage.
     */
    public function manageAdventuresTourismDates(Request $request)
    {
        if (!Auth::check()) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }

        $user = auth()->user();

        $input = $request->all();

```

```

// Validate the request data
$validator = Validator::make($input, [
    'target_id' => 'required|exists:adventures_tourisms,id',
    'start_date' => 'required|date',
    'end_date' => 'required|date|after_or_equal:start_date',
    'price' => 'required|numeric',
    'person_types' => 'required|array',
    'max_capacity' => 'required|numeric',
    'min_number_of_booking' => 'required|numeric',
    'active' => 'required|boolean',
    'availability_type' => 'required|in:per_day,per_hour',
    'open_days_hours' => 'nullable|array',
    'open_days_hours.*' => 'array',
    'open_days_hours.*.*' => 'date_format:H:i',
    'note_to_customer' => 'nullable|string',
    'note_to_admin' => 'nullable|string',
]);

// If validation fails, return an error response
if ($validator->fails()) {
    return $this->sendError('Validation Error.', $validator-
>errors());
}

$input['person_types'] = json_encode($input['person_types'],
true);

// Check the availability type and apply logic accordingly
$availabilityType = $input['availability_type'];

DB::beginTransaction();
try {
    $startDate = Carbon::createFromFormat('Y-m-d',
$input['start_date']);
    $endDate = Carbon::createFromFormat('Y-m-d',
$input['end_date']);

    AdventuresTourismDate::where('target_id',
$input['target_id'])
        ->whereDate('start_date', '>=',
$startDate)
        ->whereDate('end_date', '<=', $endDate)
        ->delete();

    // Iterate over each day within the date range
    while ($startDate->lte($endDate)) {
        $dayOfWeek = $startDate->format('l');

```

```

        // Check if this day of the week has defined hours
        if (array_key_exists($dayOfWeek,
$input['open_days_hours'])) {
            foreach ($input['open_days_hours'][$dayOfWeek] as
$hour) {
                $dateTime = $startDate->copy()->format('Y-m-d')
. ' ' . $hour;

                $adventuresTourismDate =
AdventuresTourismDate::updateOrCreate(
                    [
                        'target_id' => $input['target_id'],
                        'start_date' => $dateTime,
                    ],
                    [
                        'end_date' => $dateTime,
                        'price' => $input['price'],
                        'person_types' =>
$input['person_types'],
                        'max_capacity' =>
$input['max_capacity'],
                        'min_number_of_booking' =>
$input['min_number_of_booking'],
                        'active' => $input['active'],
                        'note_to_customer' =>
$input['note_to_customer'] ?? '',
                        'note_to_admin' =>
$input['note_to_admin'] ?? '',
                        'create_user' => $user->id,
                        'update_user' => $user->id,
                    ]
                );
            }
        }
        $startDate->addDay();
    }

    DB::commit();
    return $this->sendResponse(['success'], 'Adventures tourism
dates managed successfully.');
```

```

    } catch (\Exception $e) {
        DB::rollBack();
        return $this->sendError('An error occurred while updating
the dates: ' . $e->getMessage());
    }
}
}

```

```

/**
 * Display the specified resource.
 */
public function show($id): JsonResponse
{
    $AdventuresTourismDate = AdventuresTourismDate::find($id);

    if (is_null($AdventuresTourismDate)) {
        return $this->sendError('AdventuresTourismDate not
found.');
```

```

    }

    return $this->sendResponse(new
AdventuresTourismDatesResource($AdventuresTourismDate),
'AdventuresTourismDate retrieved successfully.');
```

```

}

/**
 * Update the specified resource in storage.
 */
public function update(Request $request, AdventuresTourismDate
$adventuresTourismDate): JsonResponse
{
    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }

    $this->authorize('update', $adventuresTourismDate);

    $input = $request->all();

    // Validate the request data
    $validator = Validator::make($input, [
        'target_id' =>
'required|exists:adventures_tourism_dates,id',
        'start_date' => 'required|date',
        'end_date' => 'required|date|after_or_equal:start_date',
        'price' => 'required|numeric',
        'number' => 'required|numeric',
        'active' => 'required|numeric',
    ]));

```

```

        // If validation fails, return an error response
        if ($validator->fails()) {
            return $this->sendError('Validation Error.', $validator-
>errors());
        }

        $input['update_user'] = auth()->id();

        $adventuresTourismDate->update($input);

        return $this->sendResponse(new
AdventuresTourismDatesResource($adventuresTourismDate),
'AdventuresTourismDate updated successfully.');
```

```

    /**
     * Remove the specified resource from storage.
     */
    public function destroy(AdventuresTourismDate
$adventuresTourismDate): JsonResponse
    {

        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
        }

        $this->authorize('delete', $adventuresTourismDate);

        $adventuresTourismDate->delete();

        return $this->sendResponse([], 'AdventuresTourismDate deleted
successfully.');
```

```

    }
}

```

TranslationController Documentation

Επισκόπηση

Ο TranslationController είναι υπεύθυνος για τη διαχείριση των αιτημάτων μετάφρασης και των σχετικών λειτουργιών. Αυτές περιλαμβάνουν την αποστολή αιτημάτων μετάφρασης, τη διαχείριση των εισερχόμενων μεταφράσεων και τον έλεγχο της κατάστασης των αιτημάτων.

Εξαρτήσεις (Dependencies)

- **BaseController:** Βασικός controller για τυποποιημένα responses.
- **TranslationRequest Model:** Χρησιμοποιείται για τη διαχείριση των αιτημάτων μετάφρασης στη βάση δεδομένων.
- **Validator:** Για την επικύρωση εισόδου.
- **Auth:** Για τη διαχείριση αυθεντικοποίησης.
- **DB:** Για τη διαχείριση transactions.

Μέθοδοι

1. sendTranslationRequest

- **Σκοπός:** Αποστολή αιτήματος μετάφρασης.
- **Παράμετροι:**
 - **object_model:** Το μοντέλο του αντικειμένου (π.χ. Hotel, AdventuresTourism).
 - **object_id:** Το ID του αντικειμένου.
 - **target_languages:** Οι γλώσσες προορισμού.
- **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Αποστέλλει το αίτημα μετάφρασης στον εξωτερικό API.
 - Αποθηκεύει το αίτημα στη βάση δεδομένων.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

2. sendTranslationDocumentRequest

- **Σκοπός:** Αποστολή αιτήματος μετάφρασης εγγράφου.
- **Παράμετροι:**
 - **object_model:** Το μοντέλο του αντικειμένου (π.χ. Hotel, AdventuresTourism).
 - **object_id:** Το ID του αντικειμένου.
 - **target_languages:** Οι γλώσσες προορισμού.
- **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Αποστέλλει το αίτημα μετάφρασης εγγράφου στον εξωτερικό API.
 - Αποθηκεύει το αίτημα στη βάση δεδομένων.

- **Response:** Μήνυμα επιτυχίας ή σφάλμα.
-

3. handleIncomingTranslation

- **Σκοπός:** Διαχείριση εισερχόμενης μετάφρασης.
 - **Παράμετροι:**
 - target-language: Η γλώσσα προορισμού.
 - translated-text: Το μεταφρασμένο κείμενο.
 - request-id: Το ID του αιτήματος.
 - **Λειτουργία:**
 - Ενημερώνει τη βάση δεδομένων με τη μετάφραση.
 - Αποθηκεύει τη μετάφραση στο αντίστοιχο μοντέλο.
 - **Response:** Μήνυμα επιτυχίας ή σφάλμα.
-

4. handleIncomingDocumentTranslation

- **Σκοπός:** Διαχείριση εισερχόμενης μετάφρασης εγγράφου.
 - **Παράμετροι:**
 - target-language: Η γλώσσα προορισμού.
 - translated-text: Το μεταφρασμένο κείμενο.
 - request-id: Το ID του αιτήματος.
 - **Λειτουργία:**
 - Ενημερώνει τη βάση δεδομένων με τη μετάφραση.
 - Αποθηκεύει τη μετάφραση στο αντίστοιχο μοντέλο.
 - **Response:** Μήνυμα επιτυχίας ή σφάλμα.
-

5. checkTranslationStatus

- **Σκοπός:** Έλεγχος της κατάστασης ενός αιτήματος μετάφρασης.
- **Παράμετροι:**
 - request_id: Το ID του αιτήματος.
- **Λειτουργία:**
 - Ελέγχει αν όλες οι γλώσσες έχουν μεταφραστεί.
- **Response:** Στοιχεία κατάστασης ή σφάλμα.

6. getAvailableLocales

- **Σκοπός:** Ανάκτηση διαθέσιμων γλωσσών για ένα αντικείμενο.
- **Παράμετροι:**
 - request_id: Το ID του αιτήματος.
 - request_object_model: Το μοντέλο του αντικειμένου.
- **Λειτουργία:**
 - Επιστρέφει τις διαθέσιμες γλώσσες για το αντικείμενο.
- **Response:** Λίστα γλωσσών ή σφάλμα.

```
<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\API\BaseController as BaseController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Http;
use Illuminate\Http\Client\RequestException;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Log;

use App\Models\TranslationRequest;

class TranslationController extends BaseController
{
    protected $serviceTypeMap = [
        'wine_tourism' =>
            \Modules\WineTourism\Models\WineTourism::class,
        'gastronomic_tourism' =>
            \Modules\GastronomicTourism\Models\GastronomicTourism::class,
        'cultural_urban_tourism' =>
            \Modules\CulturalUrbanTourism\Models\CulturalUrbanTourism::class,
        'wedding_tourism' =>
            \Modules\WeddingTourism\Models\WeddingTourism::class,
        'cultural_festival_tourism' =>
            \Modules\CulturalFestivalTourism\Models\CulturalFestivalTourism::class,
        'dark_tourism' =>
            \Modules\DarkTourism\Models\DarkTourism::class,
        'city_games_tourism' =>
            \Modules\CityGamesTourism\Models\CityGamesTourism::class,
```

```

        'sacred_site_visits_tourism' =>
\Modules\SacredSiteVisitsTourism\Models\SacredSiteVisitsTourism::class,
        'agritourism' =>
\Modules\Agritourism\Models\Agritourism::class,
        'ecotourism' => \Modules\Ecotourism\Models\Ecotourism::class,
        'exploration_tourism' =>
\Modules\ExplorationTourism\Models\ExplorationTourism::class,
        'geotourism' => \Modules\Geotourism\Models\Geotourism::class,
        'fishing_tourism' =>
\Modules\FishingTourism\Models\FishingTourism::class,
        'sporting_events_tourism' =>
\Modules\SportingEventsTourism\Models\SportingEventsTourism::class,
        'adventures_tourism' =>
\Modules\AdventuresTourism\Models\AdventuresTourism::class,
        'water_activities_tourism' =>
\Modules\WaterActivitiesTourism\Models\WaterActivitiesTourism::class,
        'air_activities_tourism' =>
\Modules\AirActivitiesTourism\Models\AirActivitiesTourism::class,
        'urban_entertainment_tourism' =>
\Modules\UrbanEntertainmentTourism\Models\UrbanEntertainmentTourism::cl
ass,
        'education_tourism' =>
\Modules\EducationDevelopmentTourism\Models\EducationDevelopmentTourism
::class,
        'enrichment_tourism' =>
\Modules\EnrichmentSkillBuildingTourism\Models\EnrichmentSkillBuildingT
ourism::class,
        'cultural_urban_exp_tourism' =>
\Modules\CulturalUrbanExplorationTourism\Models\CulturalUrbanExploratio
nTourism::class,
    ];

    public function sendTranslationRequest(Request $request):
JsonResponse
    {
        $translationEndpoint =
"https://webgate.ec.europa.eu/etranslation/si/translate";

        $apiUsername=env('TRANSLATION_API_USERNAME');
        $apiPassword=env('TRANSLATION_API_PASSWORD');

        // $textToTranslate = $request->input('text_to_translate');
        $objectModel = $request->input('object_model');
        $objectId = $request->input('object_id');
        $targetLanguages = json_decode($request-
>input('target_languages'));

        $model = $this->serviceTypeMap[$objectModel];

```

```

    $tourism = $model::findOrFail($objectId);

    $textToTranslate = implode('
.[nottranslate]delimiter[/nottranslate]. ', [
        $tourism->title,
        $tourism->short_description,
        $tourism->program_details,
        $tourism->address_and_description,
        $tourism->necessary_items,
        $tourism->dress_code,
        $tourism->important_booking_info,
    ]);

    $callback='http://185.138.41.199/api/incoming-translation';

    $data = [
        "textToTranslate" => $textToTranslate,
        "sourceLanguage" => "en",
        "targetLanguages" => $targetLanguages,
        "requesterCallback" => $callback,
        "callerInformation" => [
            "application" => $apiUsername
        ]
    ];

    try {
        $response = Http::withOptions([
            'auth' => [$apiUsername, $apiPassword, 'digest']
        ]->post($translationEndpoint, $data);

        if ($response->successful()) {
            $requestId = $response->body();

            DB::beginTransaction();

            $translationReq=new TranslationRequest();

            $translationReq->object_model=$objectModel;
            $translationReq->object_id=$objectId;
            $translationReq->request_id=$requestId;
            $translationReq->requested_lang=json_encode($targetLanguages);
            $translationReq->complete_lang=json_encode([]);

            $translationReq->save();
        }
    }

```

```

        DB::commit();

        return response()->json(['message' =>
$textToTranslate], 200);
    }
    else {
        $message = 'Translation Service Error '.$response-
>body();
        return response()->json(['error' => $message], 500);
    }
} catch (\Exception $e) {
    return response()->json(['error' => 'Failed to send
translation request'.$e->getMessage()], 500);;
}

}

public function sendTranslationDocumentRequest(Request $request):
JsonResponse
{
    $translationEndpoint =
"https://webgate.ec.europa.eu/etranslation/si/translate";

    $apiUsername=env('TRANSLATION_API_USERNAME');
    $apiPassword=env('TRANSLATION_API_PASSWORD');

    $objectModel = $request->input('object_model');
    $objectId = $request->input('object_id');
    $targetLanguages = json_decode($request-
>input('target_languages'));

    $model = $this->serviceTypeMap[$objectModel];

    $tourism = $model::findOrFail($objectId);

    # $textToTranslate = implode('
.[notranslate]delimiter[/notranslate]. ', [
    $textToTranslate = implode(' ', [
        "<p>".$tourism->title."</p>",
        "<p>".$tourism->short_description."</p>",
        "<p>".$tourism->program_details."</p>",
        "<p>".$tourism->address_and_description."</p>",
        "<p>".$tourism->necessary_items."</p>",
        "<p>".$tourism->dress_code."</p>",
        "<p>".$tourism->important_booking_info."</p>",
    ]);
}

```

```

        $callback='http://185.138.41.199/api/incoming-document-translation';

        $encodedText=base64_encode($textToTranslate);

        $data = [
            "documentToTranslateBase64"=> ["content"=>$encodedText,
"format"=>"html"],
            "sourceLanguage" => "EN",
            "targetLanguages" => $targetLanguages,
            "destinations"=> ["httpDestinations" => [$callback]],
            "callerInformation" => [
                "application" => $apiUsername
            ]
        ];

        try {
            $response = Http::withOptions([
                'auth' => [$apiUsername, $apiPassword, 'digest']
            ]->post($translationEndpoint, $data);

            if ($response->successful()) {
                $requestId = $response->body();

                DB::beginTransaction();

                $translationReq=new TranslationRequest();

                $translationReq->object_model=$objectModel;
                $translationReq->object_id=$objectId;
                $translationReq->request_id=$requestId;
                $translationReq->requested_lang=json_encode($targetLanguages);
                $translationReq->complete_lang=json_encode([]);

                $translationReq->save();

                DB::commit();

                // return response()->json(['message' => 'Translation request sent successfully'], 200);

                return $this->sendResponse($translationReq,
'Translation request sent successfully.');
```

```

        $message = 'Translation Service Error ' . $response-
>body();
        return response()->json(['error' => $message], 500);
    }
} catch (\Exception $e) {
    return response()->json(['error' => 'Failed to send
translation request' . $e->getMessage()], 500);
}

}

public function handleIncomingTranslation(Request $request) {
    $input = $request->all();

    DB::beginTransaction();
    try {

        $targetLang=strtolower($input['target-language']);
        $translatedText=$input['translated-text']; //remove quotes
from the beginning and end
        $requestId=$input['request-id'];

        if (!mb_check_encoding($translatedText, 'UTF-8')) {
            $translatedText = mb_convert_encoding($translatedText,
'UTF-8');
        }

        $translation = TranslationRequest::where('request_id',
$requestId)->first();

        $model=$this->serviceTypeMap[$translation->object_model];

        $tourism=$model::findOrFail($translation->object_id);

        $array = explode("[nottranslate]delimiter[/nottranslate].",
$translatedText);

        Log::info($targetLang);
        Log::info($translatedText);
        $data = [
            'title' => $array[0],
            'short_description' => $array[1],
            'program_details' => $array[2],
            'address_and_description' => $array[3],
            'necessary_items' => $array[4],

```

```

        'dress_code' => $array[5],
        'important_booking_info' => $array[6],
    ];

    $existingTranslation = $tourism->translations()-
>where('locale', $targetLang)->first();

    if ($existingTranslation) {
        $existingTranslation->update($data);
    } else {
        $tourism->translateOrNew($targetLang)->fill($data);
        $tourism->save();
    }

    $newLangs = json_decode($translation->complete_lang);
    $newLangs[] = $targetLang;

    $translation->complete_lang=json_encode($newLangs);
    $translation->save();

    DB::commit();
} catch (\Exception $e) {
    DB::rollBack();
    Log::error($e);
}

}

public function handleIncomingDocumentTranslation(Request $request)
{
    $input = $request->all();

    $content = $request->getContent();
    DB::beginTransaction();
    try {

        $targetLang=strtolower($input['target-language']);

        $incomingText=$content;
        $translatedText=base64_decode($incomingText);

        $requestId=$input['request-id'];

        if (!mb_check_encoding($translatedText, 'UTF-8')) {
            $translatedText = mb_convert_encoding($translatedText,
'UTF-8');
        }
    }
}

```

```

        $translation = TranslationRequest::where('request_id',
$requestId)->first();

        $model=$this->serviceTypeMap[$translation->object_model];

        $tourism=$model::findOrFail($translation->object_id);

        $cleanText=str_replace("</p>", "", $translatedText);
        $array = explode("<p>", $cleanText);

        $data = [
            'title' => $array[1],
            'short_description' => $array[2],
            'program_details' => $array[3],
            'address_and_description' => $array[4],
            'necessary_items' => $array[5],
            'dress_code' => $array[6],
            'important_booking_info' => $array[7],
        ];

        $existingTranslation = $tourism->translations()-
>where('locale', $targetLang)->first();

        if ($existingTranslation) {
            $existingTranslation->update($data);
        } else {
            $tourism->translateOrNew($targetLang)->fill($data);
            $tourism->save();
        }

        $newLangs = json_decode($translation->complete_lang);
        $newLangs[] = $targetLang;

        $translation->complete_lang=json_encode($newLangs);
        $translation->save();

        DB::commit();
    } catch (\Exception $e) {
        DB::rollBack();
        Log::error($e);
    }
}

public function checkTranslationStatus(Request $request):
JsonResponse
{

```

```

        $requestId = $request->input('request_id');

        $translationReq = TranslationRequest::where('request_id',
$requestId)->first();

        if (!$translationReq) {
            return response()->json(['error' => 'Invalid request ID'],
400);
        }

        $requestedLanguages = json_decode($translationReq-
>requested_lang);
        $completeLanguages = json_decode($translationReq-
>complete_lang);

        $processingLanguages = array_diff($requestedLanguages,
$completeLanguages);

        if (empty($processingLanguages)) {
            return response()->json(['message' => 'All requested
languages are complete.']);
        } else {
            return response()->json([
                'message' => 'Some translations are still pending.',
                'processing_languages' =>
array_values($processingLanguages)
            ]);
        }
    }

    public function getAvailableLocales(Request $request): JsonResponse
    {
        try {

            $objectId = $request->input('request_id');
            $objectModel = $request->input('request_object_model');

            $model = $this->serviceTypeMap[$objectModel];

            $tourism = $model::findOrFail($objectId);

            $locales = array_keys($tourism->getTranslationsArray());

            return $this->sendResponse($locales, 'All available Locales
request sent successfully.');
```

```

        } catch (\Exception $e) {

```

```

        return response()->json([
            'status' => 'error',
            'message' => 'Failed to fetch available locales: ' .
$e->getMessage(),
        ], 500);
    }
}
}
}

```

HotelController Documentation

Επισκόπηση

Ο HotelController είναι υπεύθυνος για τη διαχείριση των ξενοδοχείων (Hotels) και των σχετικών λειτουργιών. Αυτές περιλαμβάνουν τη δημιουργία, ενημέρωση, διαγραφή και ανάκτηση πληροφοριών για ξενοδοχεία.

Εξαρτήσεις (Dependencies)

- **BaseController:** Βασικός controller για τυποποιημένα responses.
- **Hotel Model:** Χρησιμοποιείται για τη διαχείριση των ξενοδοχείων στη βάση δεδομένων.
- **MediaFileController:** Για τη διαχείριση μεταφορτώσεων αρχείων (εικόνες, πιστοποιητικά κ.λπ.).
- **Validator:** Για την επικύρωση εισόδου.
- **Auth:** Για τη διαχείριση πιστοποίησης.
- **DB:** Για τη διαχείριση transactions.

Μέθοδοι

1. index

- **Σκοπός:** Ανάκτηση όλων των καταχωρήσεων ξενοδοχείων.
- **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις ξενοδοχείων.
- **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.

2. getVendorHotels

- **Σκοπός:** Ανάκτηση ξενοδοχείων για έναν συγκεκριμένο Vendor.
- **Παράμετροι:**
 - id: Το ID του Vendor.
- **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις ξενοδοχείων που δημιουργήθηκαν από τον Vendor.
- **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.

3. store

- **Σκοπός:** Δημιουργία νέας καταχώρησης ξενοδοχείου.
- **Παράμετροι:**
 - title: Ο τίτλος του ξενοδοχείου.
 - content: Περιγραφή του ξενοδοχείου.
 - star_rate: Η βαθμολογία του ξενοδοχείου (αστέρια).
 - website_link: Ο ιστότοπος του ξενοδοχείου.
 - youtube_url: Το URL του YouTube βίντεο.
 - image_id: Η κύρια εικόνα του ξενοδοχείου.
 - gallery: Οι εικόνες της γκαλερί.
 - necessary_items: Απαραίτητα αντικείμενα.
 - languages: Οι γλώσσες που υποστηρίζονται.
 - google_maps_url: Το URL του Google Maps.
 - map_lat: Το γεωγραφικό πλάτος.
 - map_lng: Το γεωγραφικό μήκος.
 - address: Η διεύθυνση του ξενοδοχείου.
 - primary_town: Η κύρια πόλη.
 - check_in_time: Η ώρα check-in.
 - check_out_time: Η ώρα check-out.
 - min_number_before_booking: Ο ελάχιστος αριθμός κρατήσεων.
 - min_number_before_booking_meter: Η μονάδα μέτρησης για τον ελάχιστο αριθμό κρατήσεων.

- min_day_stay_require: Ο ελάχιστος αριθμός ημερών διαμονής.
 - is_sustainable: Εάν το ξενοδοχείο είναι βιώσιμο.
 - sustainability_certification_file: Το πιστοποιητικό βιωσιμότητας.
 - status: Η κατάσταση του ξενοδοχείου.
 - max_child_age: Η μέγιστη ηλικία παιδιού.
 - **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Διαχειρίζεται τη μεταφόρτωση εικόνων και πιστοποιητικών.
 - Δημιουργεί μια νέα καταχώρηση ξενοδοχείου στη βάση δεδομένων.
 - **Response:** Τα στοιχεία του νέου ξενοδοχείου ή σφάλμα επικύρωσης.
-

4. show

- **Σκοπός:** Ανάκτηση συγκεκριμένης καταχώρησης ξενοδοχείου.
 - **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - **Λειτουργία:**
 - Επιστρέφει τα στοιχεία της καταχώρησης.
 - **Response:** Στοιχεία καταχώρησης ή σφάλμα αν δεν βρεθεί.
-

5. update

- **Σκοπός:** Ενημέρωση υπάρχουσας καταχώρησης ξενοδοχείου.
 - **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - Πολλές παράμετροι, όπως title, content, star_rate, image_id, gallery, κ.λπ.
 - **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Διαχειρίζεται τη μεταφόρτωση εικόνων και πιστοποιητικών.
 - Ενημερώνει την καταχώρηση στη βάση δεδομένων.
 - **Response:** Τα ενημερωμένα στοιχεία της καταχώρησης ή σφάλμα επικύρωσης.
-

6. destroy

- **Σκοπός:** Διαγραφή καταχώρησης ξενοδοχείου.
- **Παράμετροι:**
 - **id:** Το ID της καταχώρησης.
- **Λειτουργία:**
 - Διαγράφει την καταχώρηση από τη βάση δεδομένων.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

7. checkAvailability

- **Σκοπός:** Έλεγχος διαθεσιμότητας δωματίων για ένα ξενοδοχείο.
- **Παράμετροι:**
 - **hotel_id:** Το ID του ξενοδοχείου.
 - **start_date:** Η αρχική ημερομηνία.
 - **end_date:** Η τελική ημερομηνία.
 - **adults:** Ο αριθμός ενηλίκων.
- **Λειτουργία:**
 - Ελέγχει αν οι ημερομηνίες είναι έγκυρες.
 - Ελέγχει αν το ξενοδοχείο έχει διαθέσιμα δωμάτια για τις καθορισμένες ημερομηνίες.
- **Response:** Λίστα διαθέσιμων δωματίων ή σφάλμα.

```
<?php

namespace App\Http\Controllers\API;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use App\Http\Resources\HotelResource;
use Modules\MediaFile\Controllers\MediaFileController;
use App\Models\User;
use App\Models\Hotel;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Validator;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\DB;
```

```

class HotelController extends BaseController
{
    protected $hotelClass;
    protected $mediaFileClass;

    /**
     * Set the permissions
     */
    function __construct(Hotel $hotel)
    {
        $this->mediaFileClass = MediaFileController::class;
        $this->hotelClass = $hotel;
        $this->middleware('permission:create-service', ['only' =>
['store']]);
        $this->middleware('permission:edit-service', ['only' =>
['update']]);
        $this->middleware('permission:delete-service', ['only' =>
['destroy']]);
    }

    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        // Get all Hotel entries
        $hotelEntries = Hotel::all();

        return $this-
>sendResponse(HotelResource::collection($hotelEntries), 'Hotel entries
retrieved successfully.');
```

```

    }

    /**
     * Display hotels for a specific Vendor.
     */
    public function getVendorHotels($id): JsonResponse
    {
        // Get all Hotel entries for a specific Vendor

        $hotelEntries = Hotel::where('create_user', $id)->get();

        return $this-
>sendResponse(HotelResource::collection($hotelEntries), 'Vendor`s Hotel
entries retrieved successfully.');
```

```

}

/**
 * Store a newly created resource in storage.
 */
public function store(Request $request)
{
    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }

    $user = auth()->user();

    $input = $request->all();

    // Validate the request data
    $validator = Validator::make($input, [
        'title' => 'required|string|max:255',
        'content' => 'required|string',
        'star_rate' => 'required|numeric',
        'website_link' => 'nullable|string',
        'youtube_url' => 'nullable|string',
        'image_id' => 'required|image',
        'gallery.*' => 'required|image',
        'necessary_items' => 'nullable|string',
        'languages' => 'required|array',
        'google_maps_url' => 'nullable|string',
        'map_lat' => 'required|string',
        'map_lng' => 'required|string',
        'address' => 'required|string',
        'primary_town' => 'required|string',
        'check_in_time' => 'required|string',
        'check_out_time' => 'required|string',
        'min_number_before_booking' => 'required|numeric',
        'min_number_before_booking_meter' => 'required|string',
        'min_day_stay_require' => 'required|numeric',
        'is_sustainable' => 'nullable|boolean',
        'sustainability_certification_file' =>
'nullable|file|mimes:pdf',
        'status' => 'required|string',
        'max_child_age' => 'required|numeric'

    ]);

    // If validation fails, return an error response
    if ($validator->fails()) {

```

```

        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    try {

        DB::beginTransaction();

        if ($request->hasFile('image_id')) {
            $input['image_id'] = $this-
>mediaFileClass::handleMediaUpload($request->file('image_id'),
            'hotel');
        }

        if ($request->hasFile('banner_image_id')) {
            $input['banner_image_id'] = $this-
>mediaFileClass::handleMediaUpload($request->file('banner_image_id'),
            'hotel');
        }

        if ($request->hasFile('gallery')) {
            $galleryImages = $request->file('gallery');
            $galleryIds = [];

            foreach ($galleryImages as $image) {
                $galleryIds[] = $this-
>mediaFileClass::handleMediaUpload($image, 'hotel');
            }

            $input['gallery'] = json_encode($galleryIds);
        }

        // Handle sustainability certification file upload if
is_sustainable is true
        if ($request->input('is_sustainable') && $request-
>hasFile('sustainability_certification_file')) {
            $input['sustainability_cerification_id'] = $this-
>mediaFileClass::handleMediaUpload($request-
>file('sustainability_certification_file'), 'sustainability-
certifications');
        }

        unset($input['sustainability_certification_file']);

        // Encode fields that should be stored as JSON
        $fieldsToEncode = [

```

```

        'gallery',
        'languages',
    ];

    // Modify this part of your Laravel code
    foreach ($fieldsToEncode as $field) {
        if (isset($input[$field]) &&
!is_string($input[$field])) {
            // Only encode as JSON if it's not already a JSON
string
            $input[$field] = json_encode($input[$field]);
        }
    };

    $input['create_user'] = $user->id;
    // $input['author_id'] = $user->id;

    $hotel = Hotel::create($input);

    DB::commit();

    return $this->sendResponse(new HotelResource($hotel),
'Hotel created successfully.');
```

```

    } catch (\Exception $e) {
        // Rollback the transaction in case of an error
        DB::rollBack();

        // Return an error response
        return $this->sendError('Error occurred during hotel
creation.', ['error' => $e->getMessage()]);
    }

}

/**
 * Display the specified resource.
 */
public function show($id): JsonResponse
{
    $hotel = Hotel::find($id);

    if (is_null($hotel)) {
        return $this->sendError('Hotel not found.');
```

```

        return $this->sendResponse(new HotelResource($hotel), 'Hotel
retrieved successfully.');
```

```

    }

    /**
     * Update the specified resource in storage.
     */
    public function update(Request $request, Hotel $hotel):
JsonResponse
    {

        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
        }

        $this->authorize('update', $hotel);

        $input = $request->all();

        // Validate the request data
        $validator = Validator::make($input, [
            'title' => 'required|string|max:255',
            'content' => 'required|string',
            'star_rate' => 'required|numeric',
            'website_link' => 'nullable|string',
            'youtube_url' => 'nullable|string',
            'image_id' => 'nullable|image',
            'gallery.*' => 'nullable|image',
            'necessary_items' => 'nullable|string',
            'languages' => 'required|array',
            'google_maps_url' => 'nullable|string',
            'map_lat' => 'required|string',
            'map_lng' => 'required|string',
            'address' => 'required|string',
            'primary_town' => 'required|string',
            'check_in_time' => 'required|string',
            'check_out_time' => 'required|string',
            'min_number_before_booking' => 'required|numeric',
            'min_number_before_booking_meter' => 'required|string',
            'min_day_stay_require' => 'required|numeric',
            'is_sustainable' => 'nullable|boolean',
            'sustainability_certification_file' =>
'nullable|file|mimes:pdf',
            'status' => 'required|string',
            'max_child_age' => 'required|numeric'
        ]);
    }
}

```

```

    });

    // If validation fails, return an error response
    if ($validator->fails()) {
        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    try {
        // Start Transaction
        DB::beginTransaction();

        // Handle Image Update
        if ($request->hasFile('image_id')) {
            $file = $request->file('image_id');

            if ($file->isValid()) {

                $input['image_id'] = $this-
>mediaFileClass::updateMediaFile($hotel->image_id, $file, 'hotel');

            }
        } else {
            unset($input['image_id']);
        }

        if ($request->hasFile('banner_image_id')) {
            $file = $request->file('banner_image_id');

            if ($file->isValid()) {

                $input['banner_image_id'] = $this-
>mediaFileClass::updateMediaFile($hotel->banner_image_id, $file,
'hotel');

            }
        } else {
            unset($input['banner_image_id']);
        }

        if ($request->hasFile('gallery')) {

            $this->mediaFileClass::deleteGallery($hotel->gallery);

            $galleryImages = $request->file('gallery');
            $galleryIds = [];

```

```

        foreach ($galleryImages as $image) {
            if ($image->isValid()) {

                $galleryIds[] = $this->mediaFileClass::handleMediaUpload($image, 'hotel');
            }
        }

        $input['gallery'] = json_encode($galleryIds);
    } else {
        unset($input['gallery']);
    }

    $input['update_user'] = auth()->id();

    $hotel->update($input);

    // Commit the transaction
    DB::commit();

    return $this->sendResponse(new HotelResource($hotel),
    'Hotel updated successfully.');
```

```

    } catch (\Exception $e) {
        // Rollback the transaction in case of an error
        DB::rollBack();

        // Return an error response
        return $this->sendError('Error occurred during update.',
    ['error' => $e->getMessage()]);
    }

}

/**
 * Remove the specified resource from storage.
 */
public function destroy(Hotel $hotel): JsonResponse
{

    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
    ['error'=>'Unauthored']);
    }
}

```

```

        $this->authorize('delete', $hotel);

        $hotel->delete();

        return $this->sendResponse([], 'Hotel deleted successfully.');
```

 }

 public function checkAvailability(){
 \$hotel_id = \request('hotel_id');
 if(\request()->input('firstLoad') == "false") {
 \$rules = [
 'hotel_id' => 'required',
 'start_date' => 'required:date_format:Y-m-d',
 'end_date' => 'required:date_format:Y-m-d',
 'adults' => 'required',
];
 \$validator = \Validator::make(request()->all(), \$rules);
 if (\$validator->fails()) {
 return \$this->sendError(\$validator->errors()->all());
 }

 if(strtotime(\request('end_date')) -
 strtotime(\request('start_date')) < DAY_IN_SECONDS){
 return \$this->sendError(__("Dates are not valid"));
 }
 if(strtotime(\request('end_date')) -
 strtotime(\request('start_date')) > 30*DAY_IN_SECONDS){
 return \$this->sendError(__("Maximum day for booking is
 30"));
 }
 }

 \$hotel = \$this->hotelClass::find(\$hotel_id);
 if(empty(\$hotel_id) or empty(\$hotel)){
 return \$this->sendError(__("Hotel not found"));
 }

 if(\request()->input('firstLoad') == "false") {
 \$numberDays = abs(strtotime(\request('end_date')) -
 strtotime(\request('start_date'))) / 86400;
 if(!empty(\$hotel->min_day_stays) and \$numberDays < \$hotel-
 >min_day_stays){
 return \$this->sendError(__("You must to book a minimum
 of :number days",['number'=>\$hotel->min_day_stays]));
 }

 if(!empty(\$hotel->min_day_before_booking)){

```

        $minday_before = strtotime("today +".$hotel->min_day_before_booking." days");
        if( strtotime(\request('start_date')) < $minday_before){
            return $this->sendError(__("You must book the service for :number days in advance",["number"=>$hotel->min_day_before_booking]));
        }
    }

    $rooms = $hotel->getRoomsAvailability(request()->input());

    return $this->sendResponse([
        'rooms'=>$rooms
    ], 'Check availability rooms response successfully');
}
}

```

HotelRoomController Documentation

Επισκόπηση

Ο **HotelRoomController** είναι υπεύθυνος για τη διαχείριση των δωματίων ξενοδοχείων (Hotel Rooms) και των σχετικών λειτουργιών. Αυτές περιλαμβάνουν τη δημιουργία, ενημέρωση, διαγραφή και ανάκτηση πληροφοριών για τα δωμάτια των ξενοδοχείων.

Εξαρτήσεις (Dependencies)

- **BaseController**: Βασικός controller για τυποποιημένα responses.
- **HotelRoom Model**: Χρησιμοποιείται για τη διαχείριση των δωματίων στη βάση δεδομένων.
- **MediaFileController**: Για τη διαχείριση μεταφορτώσεων αρχείων (εικόνες, γκαλερί, κ.λπ.).
- **Validator**: Για την επικύρωση εισόδου.
- **Auth**: Για τη διαχείριση πιστοποίησης.
- **DB**: Για τη διαχείριση transactions.

Μέθοδοι

1. index

- **Σκοπός:** Ανάκτηση όλων των καταχωρήσεων δωματίων ξενοδοχείων.
- **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις δωματίων ξενοδοχείων.
- **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.

2. getHotelRooms

- **Σκοπός:** Ανάκτηση όλων των δωματίων για ένα συγκεκριμένο ξενοδοχείο.
- **Παράμετροι:**
 - id: Το ID του ξενοδοχείου.
- **Λειτουργία:**
 - Επιστρέφει όλα τα διαθέσιμα δωμάτια του ξενοδοχείου με το συγκεκριμένο ID.
- **Response:** Λίστα δωματίων ή σφάλμα αν δεν βρεθούν.

3. store

- **Σκοπός:** Δημιουργία νέας καταχώρησης δωματίου ξενοδοχείου.
- **Παράμετροι:**
 - title: Ο τίτλος του δωματίου.
 - content: Περιγραφή του δωματίου.
 - image_id: Η κύρια εικόνα του δωματίου.
 - gallery: Οι εικόνες της γκαλερί.
 - price: Η τιμή του δωματίου.
 - number: Ο αριθμός των διαθέσιμων δωματίων.
 - size: Το μέγεθος του δωματίου.
 - adults: Ο μέγιστος αριθμός ενηλίκων.
 - children: Ο μέγιστος αριθμός παιδιών.
 - status: Η κατάσταση του δωματίου.
 - parent_id: Το ID του ξενοδοχείου στο οποίο ανήκει το δωμάτιο.
 - is_refundable: Αν το δωμάτιο είναι επιστρέψιμο.

- full_refund_time: Χρόνος πλήρους επιστροφής.
- partial_refund_time: Χρόνος μερικής επιστροφής.
- partial_refund_time_percentage: Ποσοστό επιστροφής.
- amenities: Παροχές δωματίου.
- facilities: Εγκαταστάσεις δωματίου.
- number_of_beds: Ο αριθμός των κρεβατιών.
- **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Διαχειρίζεται τη μεταφόρτωση εικόνων.
 - Δημιουργεί μια νέα καταχώρηση στη βάση δεδομένων.
- **Response:** Τα στοιχεία του νέου δωματίου ή σφάλμα επικύρωσης.

4. show

- **Σκοπός:** Ανάκτηση συγκεκριμένης καταχώρησης δωματίου.
- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
- **Λειτουργία:**
 - Επιστρέφει τα στοιχεία της καταχώρησης.
- **Response:** Στοιχεία καταχώρησης ή σφάλμα αν δεν βρεθεί.

5. update

- **Σκοπός:** Ενημέρωση υπάρχουσας καταχώρησης δωματίου.
- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - Πολλές παράμετροι, όπως title, content, image_id, gallery, price, number, size, κ.λπ.
- **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Διαχειρίζεται τη μεταφόρτωση εικόνων.
 - Ενημερώνει την καταχώρηση στη βάση δεδομένων.
- **Response:** Τα ενημερωμένα στοιχεία της καταχώρησης ή σφάλμα επικύρωσης.

6. destroy

- **Σκοπός:** Διαγραφή καταχώρησης δωματίου.
- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
- **Λειτουργία:**
 - Διαγράφει την καταχώρηση από τη βάση δεδομένων.
- **Response:** Μήνυμα επιτυχίας ή σφάλμα.

```
<?php

namespace App\Http\Controllers\API;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use App\Http\Resources\HotelRoomResource;
use Modules\MediaFile\Controllers\MediaFileController;
use App\Models\User;
use App\Models\HotelRoom;
use Illuminate\Support\Facades\Auth;
use Validator;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\DB;

class HotelRoomController extends BaseController
{
    protected $mediaFileClass;

    /**
     * Set the permissions
     */
    function __construct()
    {
        $this->mediaFileClass = MediaFileController::class;

        $this->middleware('permission:create-service', ['only' =>
['store']]);
        $this->middleware('permission:edit-service', ['only' =>
['update']]);
    }
}
```

```

        $this->middleware('permission:delete-service', ['only' =>
['destroy']]));
    }

    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        // Get all HotelRoom entries
        $hotelRoomEntries = HotelRoom::all();

        return $this-
>sendResponse(HotelRoomResource::collection($hotelRoomEntries),
'HotelRoom entries retrieved successfully.');
```

```

    }

    /**
     * Display rooms for a specific Hotel.
     */
    public function getHotelRooms($id): JsonResponse
    {
        // Get all Hotel entries for a specific Vendor

        $roomsEntries = HotelRoom::where('parent_id', $id)->get();

        return $this-
>sendResponse(HotelRoomResource::collection($roomsEntries), 'Hotel`s
rooms entries retrieved successfully.');
```

```

    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
        }

        $user = auth()->user();

        $input = $request->all();

        // Validate the request data
        $validator = Validator::make($input, [

```

```

        'title' => 'required|string|max:255',
        'content' => 'required|string',
        'image_id' => 'required|image',
        // 'banner_image_id' => 'required|image',
        'gallery.*' => 'required|image',
        'price' => 'required|numeric',
        'number' => 'required|integer',
        // 'beds' => 'required|integer',
        'size' => 'required|integer',
        'adults' => 'required|integer',
        'children' => 'required|integer',
        'status' => 'required|string',
        'parent_id' => 'required|exists:hotels,id',

        'is_refundable' => 'required|boolean',
        'full_refund_time' => 'nullable|string',
        'partial_refund_time' => 'nullable|string',
        'partial_refund_time_percentage' => 'nullable|integer',
        // 'no_refund_after' => 'nullable|string',

        'amenities' => 'required|array',
        'facilities' => 'required|array',
        'number_of_beds' => 'required|string',

    ]);

    // If validation fails, return an error response
    if ($validator->fails()) {
        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    try {

        DB::beginTransaction();

        if ($request->hasFile('image_id')) {
            $input['image_id'] = $this-
>mediaFileClass::handleMediaUpload($request->file('image_id'), 'hotel-
room');
        }

        if ($request->hasFile('banner_image_id')) {
            $input['banner_image_id'] = $this-
>mediaFileClass::handleMediaUpload($request->file('banner_image_id'),
'hotel-room');
        }
    }

```

```

        if ($request->hasFile('gallery')) {
            $galleryImages = $request->file('gallery');
            $galleryIds = [];

            foreach ($galleryImages as $image) {
                $galleryIds[] = $this->mediaFileClass::handleMediaUpload($image, 'hotel-rooms');
            }

            $input['gallery'] = json_encode($galleryIds);

            $fieldsToEncode = [
                'amenities',
                'facilities',
                'number_of_beds'
            ];

            foreach ($fieldsToEncode as $field) {
                if (isset($input[$field]) &&
                    !is_string($input[$field])) {
                    // Only encode as JSON if it's not already a JSON
                    string
                    $input[$field] = json_encode($input[$field]);
                }
            }

            $input['create_user'] = $user->id;
            // $input['author_id'] = $user->id;

            $hotelRoom = HotelRoom::create($input);

            DB::commit();

            return $this->sendResponse(new
                HotelRoomResource($hotelRoom), 'HotelRoom created successfully.');
```

```

        } catch (\Exception $e) {
            // Rollback the transaction in case of an error
            DB::rollBack();

            // Return an error response
            return $this->sendError('Error occurred during hotel
                creation.', ['error' => $e->getMessage()]);
        }
    }
}

```

```

    }

}

/**
 * Display the specified resource.
 */
public function show($id): JsonResponse
{
    $hotelRoom = HotelRoom::find($id);

    if (is_null($hotelRoom)) {
        return $this->sendError('HotelRoom not found.');
```

```
    }

    return $this->sendResponse(new HotelRoomResource($hotelRoom),
'HotelRoom retrieved successfully.');
```

```
    }

/**
 * Update the specified resource in storage.
 */
public function update(Request $request, HotelRoom $hotelRoom):
JsonResponse
{
    if (!Auth::check()) {
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }

    $this->authorize('update', $hotelRoom);

    $input = $request->all();

    // Validate the request data
    $validator = Validator::make($input, [
        'title' => 'required|string|max:255',
        'content' => 'required|string',
        'image_id' => 'nullable|image',
        // 'banner_image_id' => 'nullable|image',
        'gallery.*' => 'nullable|image',
        'price' => 'required|numeric',
        'number' => 'required|integer',
        // 'beds' => 'required|integer',
        'size' => 'required|integer',
        'adults' => 'required|integer',
        'children' => 'required|integer',
        'status' => 'required|string',
    ]);
}
```

```

        'is_refundable' => 'required|boolean',
        'full_refund_time' => 'nullable|string',
        'partial_refund_time' => 'nullable|string',
        'partial_refund_time_percentage' => 'nullable|integer',
        //'no_refund_after' => 'nullable|string',

        'amenities' => 'required|array',
        'facilities' => 'required|array',
        'number_of_beds' => 'required|string',
    ]));

    // If validation fails, return an error response
    if ($validator->fails()) {
        return $this->sendError('Validation Error.', $validator-
>errors());
    }

    try {
        // Start Transaction
        DB::beginTransaction();

        // Handle Image Update
        if ($request->hasFile('image_id')) {
            $file = $request->file('image_id');

            if ($file->isValid()) {

                $input['image_id'] = $this-
>mediaFileClass::updateMediaFile($hotelRoom->image_id, $file, 'hotel-
room');

            }
        } else {
            unset($input['image_id']);
        }

        if ($request->hasFile('banner_image_id')) {
            $file = $request->file('banner_image_id');

            if ($file->isValid()) {

                $input['banner_image_id'] = $this-
>mediaFileClass::updateMediaFile($hotelRoom->banner_image_id, $file,
'hotel-room');

            }
        }
    }

```

```

    } else {
        unset($input['banner_image_id']);
    }

    if ($request->hasFile('gallery')) {

        $this->mediaFileClass::deleteGallery($hotelRoom->gallery);

        $galleryImages = $request->file('gallery');
        $galleryIds = [];

        foreach ($galleryImages as $image) {
            if ($image->isValid()) {

                $galleryIds[] = $this->mediaFileClass::handleMediaUpload($image, 'hotel-room');

            }
        }

        $input['gallery'] = json_encode($galleryIds);
    } else {
        unset($input['gallery']);
    }

    $fieldsToEncode = [
        'amenities',
        'facilities',
        'number_of_beds'
    ];

    foreach ($fieldsToEncode as $field) {
        if (isset($input[$field]) &&
!is_string($input[$field])) {
            // Only encode as JSON if it's not already a JSON
string
            $input[$field] = json_encode($input[$field]);
        }
    }

    $input['update_user'] = auth()->id();

    $hotelRoom->update($input);

    // Commit the transaction
    DB::commit();

```

```

        return $this->sendResponse(new
HotelRoomResource($hotelRoom), 'HotelRoom updated successfully.');
```

```

    } catch (\Exception $e) {
        // Rollback the transaction in case of an error
        DB::rollBack();

        // Return an error response
        return $this->sendError('Error occurred during update.',
['error' => $e->getMessage()]);
    }

}

/**
 * Remove the specified resource from storage.
 */
public function destroy(HotelRoom $hotelRoom): JsonResponse
{
    if (!Auth::check()) {
        \Log::error('Unauthorized request - user not logged in');
        return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
    }

    // $this->authorize('delete', $hotelRoom);
    \Log::info('Authorization passed for user: ' . auth()->id());

    $hotelRoom->delete();

    return $this->sendResponse([], 'HotelRoom deleted
successfully.');
```

```

    }
}

```

RecommendationsController Documentation

Επισκόπηση

Ο RecommendationsController είναι υπεύθυνος για τη διαχείριση των προτάσεων (recommendations) για διάφορους τύπους τουριστικών υπηρεσιών. Αυτές περιλαμβάνουν

τη δημιουργία, αναζήτηση και επιστροφή προτάσεων βασισμένων σε παραμέτρους όπως η τοποθεσία, η κατηγορία του τουρισμού, και άλλοι παράγοντες που αφορούν την υπηρεσία.

Εξαρτήσεις (Dependencies)

- **BaseController**: Βασικός controller για τυποποιημένα responses.
- **WineTourism, GastronomicTourism, κ.λπ.**: Μοντέλα για τους διάφορους τύπους τουρισμού, όπως ο οινικός τουρισμός, ο γαστρονομικός τουρισμός, κ.ά.
- **Hotel**: Μοντέλο για τη διαχείριση των ξενοδοχείων.
- **Request**: Εισαγωγή των αιτημάτων από τον χρήστη.
- **Str**: Χρησιμοποιείται για τη δημιουργία δυναμικών ονομάτων ή άλλες λειτουργίες με τη χρήση συμβολοσειρών.
- **Log**: Για τη διαχείριση καταγραφών (logs).
- **HotelResource**: Για την επιστροφή των δεδομένων του ξενοδοχείου σε συγκεκριμένο μορφότυπο.
- **Correlations και Ontology**: Χρησιμοποιούνται για τον υπολογισμό των συσχετίσεων και την κατηγοριοποίηση των τουριστικών τύπων.

Μέθοδοι

1. generateHotelRecommendations

- **Σκοπός**: Δημιουργία προτάσεων ξενοδοχείων βάσει της διαθεσιμότητας δωματίων για μια συγκεκριμένη χρονική περίοδο.
- **Παράμετροι**:
 - **start_date**: Η ημερομηνία έναρξης.
 - **end_date**: Η ημερομηνία λήξης.
 - **number_of_recommendations**: Ο αριθμός των προτάσεων.
- **Λειτουργία**:
 - Ανάκτηση όλων των ξενοδοχείων.
 - Φιλτράρισμα για δωμάτια με διαθέσιμους χώρους για τις συγκεκριμένες ημερομηνίες.
- **Απάντηση**: Λίστα με τα διαθέσιμα ξενοδοχεία ή σφάλμα αν δεν υπάρχουν διαθέσιμα δωμάτια.

2. generateRecommendations

- **Σκοπός**: Δημιουργία προτάσεων για τουριστικές υπηρεσίες βάσει του τύπου της υπηρεσίας και άλλων παραμέτρων.
- **Παράμετροι**:
 - **service_type**: Ο τύπος της υπηρεσίας (π.χ. "hotel", "wine-tourism").
 - **object_id**: Το ID του αντικειμένου για το οποίο ζητούνται οι προτάσεις.
 - **number_of_recommendations**: Ο αριθμός των προτάσεων.

- **Λειτουργία:**
 - Ανάκτηση του αντικειμένου σύμφωνα με τον τύπο της υπηρεσίας.
 - Δημιουργία προτάσεων με βάση τις συσχετίσεις (correlations) ή τυχαία αν δεν υπάρχουν συσχετίσεις.
- **Απάντηση:** Λίστα με τις προτάσεις ή σφάλμα αν δεν βρεθούν.

3. generateRandomRecommendations

- **Σκοπός:** Δημιουργία τυχαίων προτάσεων για διάφορους τύπους τουριστικών υπηρεσιών.
- **Παράμετροι:**
 - service_type: Ο τύπος της υπηρεσίας.
 - number_of_recommendations: Ο αριθμός των προτάσεων.
- **Λειτουργία:** Ανάκτηση τυχαίων προτάσεων από τη βάση δεδομένων για τον καθορισμένο τύπο υπηρεσίας.
- **Απάντηση:** Λίστα με τις τυχαίες προτάσεις.

```
<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\API\BaseController as BaseController;
use Illuminate\Http\Request;
use Modules\Booking\Models\Booking;
use Illuminate\Support\Str;
use Illuminate\Support\Facades\Log;
use Modules\Transportation\Models\Transportation;
use Modules\Transportation\Resources\TransportationResource;
use App\Models\Hotel;

class RecommendationsController extends BaseController
{
    protected $serviceTypeMap = [
        'wine-tourism' =>
            \Modules\WineTourism\Models\WineTourism::class,
        'gastronomic-tourism' =>
            \Modules\GastronomicTourism\Models\GastronomicTourism::class,
        'cultural-urban-tourism' =>
            \Modules\CulturalUrbanTourism\Models\CulturalUrbanTourism::class,
        'wedding-tourism' =>
            \Modules\WeddingTourism\Models\WeddingTourism::class,
```

```

        'cultural-festival-tourism' =>
\Modules\CulturalFestivalTourism\Models\CulturalFestivalTourism::class,
        'dark-tourism' =>
\Modules\DarkTourism\Models\DarkTourism::class,
        'city-games-tourism' =>
\Modules\CityGamesTourism\Models\CityGamesTourism::class,
        'sacred-site-visits-tourism' =>
\Modules\SacredSiteVisitsTourism\Models\SacredSiteVisitsTourism::class,
        'agritourism' =>
\Modules\Agritourism\Models\Agritourism::class,
        'ecotourism' => \Modules\Ecotourism\Models\Ecotourism::class,
        'exploration-tourism' =>
\Modules\ExplorationTourism\Models\ExplorationTourism::class,
        'geotourism' => \Modules\Geotourism\Models\Geotourism::class,
        'fishing-tourism' =>
\Modules\FishingTourism\Models\FishingTourism::class,
        'sporting-events-tourism' =>
\Modules\SportingEventsTourism\Models\SportingEventsTourism::class,
        'adventures-tourism' =>
\Modules\AdventuresTourism\Models\AdventuresTourism::class,
        'water-activities-tourism' =>
\Modules\WaterActivitiesTourism\Models\WaterActivitiesTourism::class,
        'air-activities-tourism' =>
\Modules\AirActivitiesTourism\Models\AirActivitiesTourism::class,
        'urban-entertainment-tourism' =>
\Modules\UrbanEntertainmentTourism\Models\UrbanEntertainmentTourism::class,
        'education-tourism' =>
\Modules\EducationDevelopmentTourism\Models\EducationDevelopmentTourism::class,
        'enrichment-tourism' =>
\Modules\EnrichmentSkillBuildingTourism\Models\EnrichmentSkillBuildingTourism::class,
        'cultural-urban-exp-tourism' =>
\Modules\CulturalUrbanExplorationTourism\Models\CulturalUrbanExplorationTourism::class,
        'medical-tourism' =>
\Modules\MedicalTourism\Models\MedicalTourism::class,
        'spa-tourism' => \Modules\SpaTourism\Models\SpaTourism::class,
        'wellness-tourism' =>
\Modules\WellnessTourism\Models\WellnessTourism::class,
        'hotel' => \App\Models\Hotel::class
    ];

    protected $serviceResourceMap = [
        'wine-tourism' =>
\Modules\WineTourism\Resources\WineTourismResource::class,

```

```

        'gastronomic-tourism' =>
\Modules\GastronomicTourism\Resources\GastronomicTourismResource::class
    ,
        'cultural-urban-tourism' =>
\Modules\CulturalUrbanTourism\Resources\CulturalUrbanTourismResource::c
lass,
        'wedding-tourism' =>
\Modules\WeddingTourism\Resources\WeddingTourismResource::class,
        'cultural-festival-tourism' =>
\Modules\CulturalFestivalTourism\Resources\CulturalFestivalTourismResou
rce::class,
        'dark-tourism' =>
\Modules\DarkTourism\Resources\DarkTourismResource::class,
        'city-games-tourism' =>
\Modules\CityGamesTourism\Resources\CityGamesTourismResource::class,
        'sacred-site-visits-tourism' =>
\Modules\SacredSiteVisitsTourism\Resources\SacredSiteVisitsTourismResou
rce::class,
        'agritourism' =>
\Modules\Agritourism\Resources\AgritourismResource::class,
        'ecotourism' =>
\Modules\Ecotourism\Resources\EcotourismResource::class,
        'exploration-tourism' =>
\Modules\ExplorationTourism\Resources\ExplorationTourismResource::class
    ,
        'geotourism' =>
\Modules\Geotourism\Resources\GeotourismResource::class,
        'fishing-tourism' =>
\Modules\FishingTourism\Resources\FishingTourismResource::class,
        'sporting-events-tourism' =>
\Modules\SportingEventsTourism\Resources\SportingEventsTourismResource:
:class,
        'adventures-tourism' =>
\Modules\AdventuresTourism\Resources\AdventuresTourismResource::class,
        'water-activities-tourism' =>
\Modules\WaterActivitiesTourism\Resources\WaterActivitiesTourismResourc
e::class,
        'air-activities-tourism' =>
\Modules\AirActivitiesTourism\Resources\AirActivitiesTourismResource::c
lass,
        'urban-entertainment-tourism' =>
\Modules\UrbanEntertainmentTourism\Resources\UrbanEntertainmentTourismR
esource::class,
        'education-tourism' =>
\Modules\EducationDevelopmentTourism\Resources\EducationDevelopmentTour
ismResource::class,

```

```

        'enrichment-tourism' =>
\Modules\EnrichmentSkillBuildingTourism\Resources\EnrichmentSkillBuildi
ngTourismResource::class,
        'cultural-urban-exp-tourism' =>
\Modules\CulturalUrbanExplorationTourism\Resources\CulturalUrbanExplora
tionTourismResource::class,
        'medical-tourism' =>
\Modules\MedicalTourism\Resources\MedicalTourismResource::class,
        'spa-tourism' =>
\Modules\SpaTourism\Resources\SpaTourismResource::class,
        'wellness-tourism' =>
\Modules\WellnessTourism\Resources\WellnessTourismResource::class,
        'hotel' => \App\Http\Resources\HotelResource::class,

];

protected $ontology = [
    'adventures-tourism' => [
        'target sports' => [
            'laser tag',
            'archery',
            'paintball'
        ],
        'ball-based sports' => [
            'golf',
            'tennis',
            'football',
            'basketball',
            'volleyball'
        ],
        'snow activities' => [
            'skiing',
            'snowboarding',
            'snowshoeing'
        ],
        'off-road activities' => [
            'quad bike',
            'enduro'
        ]
    ],
    'air-activities-tourism' => [
        'freefall experiences' => [
            'paraglide',
            'parachute',
            'bungee jumping'
        ],
        'aerial riding' => [
            'tours in helicopters',

```

```

        'hot air balloons'
    ],
    'plane spotting' => []
],
'sporting-events-tourism' => [
    'marathon' => [],
    'acropolis rally' => []
],
'water-activities-tourism' => [
    'motorized water sports' => [
        'jet skiing',
        'water skiing',
        'wakeboarding',
        'powerboating'
    ],
    'aquatic adventure sports' => [
        'river trekking',
        'rafting',
        'sea kayak',
        'windsurfing',
        'kite surfing',
        'paddleboarding',
        'canyoning',
        'sailing'
    ],
    'underwater activities' => [
        'snorkeling',
        'diving',
        'sea bob'
    ],
    'cruises' => []
],
'agritourism' => [
    'farm visits' => [],
    'collecting land products' => [],
    'interactions with animals' => [],
    'horse riding' => []
],
'ecotourism' => [
    'flora and fauna' => [
        'wildlife tracking',
        'bird watching',
        'wildflower watching'
    ],
    'multimodal exploration' => [
        'rafts',
        'inflatable kayaks',
        'bicycles'
    ]
]

```

```

    ],
    'volunteering' => [
        'beach cleanup',
        'animal rescue',
        'tree planting'
    ],
    'mountain hiking' => [],
    'camping' => []
],
'exploration-tourism' => [
    'hiking tours' => [],
    'flora and fauna' => [
        'wildlife tracking',
        'bird watching',
        'wildflower watching'
    ],
    'multimodal exploration' => [
        'rafts',
        'inflatable kayaks',
        'bicycles'
    ]
],
'fishing-tourism' => [
    'fishing on boat' => [],
    'local aquatic ecosystems education' => []
],
'geotourism' => [
    'geological sites' => [],
    'spelunking' => []
],
'city-games-tourism' => [
    'ancient greek games' => [],
    'escape city games' => [],
    'mythical hunts' => []
],
'cultural-festival-tourism' => [
    'festivals' => [],
    'carnivals' => []
],
'cultural-urban-exp-tourism' => [
    'museum tours' => [],
    'historical and architectural sites' => [],
    'live shows' => [],
    'tours on wheels' => [
        'hop-on hop-off bus tours',
        'electric tuk-tuk',
        'segways',
        'electric trike bikes',
    ]
]

```

```

        'electric scooters',
        'electric bicycles'
    ],
    'local arts and crafts' => [
        'ceramic and pottery artistry',
        'mosaic artistry',
        'marble carving'
    ]
],
'dark-tourism' => [
    'tragic sites' => [],
    'haunt tours' => [],
    'war museum tours' => []
],
'gastronomic-tourism' => [
    'greek gastronomy' => [],
    'cooking lessons' => [],
    'food festival and market visits' => [],
    'winery tours' => []
],
'wedding-tourism' => [
    'wedding ceremony organizations' => []
],
'wine-tourism' => [
    'cellar visits' => [],
    'vineyard and breweries tours' => [],
    'harvesting' => [],
    'wine seminars' => [],
    'wine festivals' => []
],
'cultural-urban-tourism' => [
    'cultural and artistic tours' => []
],
'urban-entertainment-tourism' => [
    'visiting clubs and bars' => [],
    'shopping tour' => [],
    'tours on wheels' => [
        'hop-on hop-off bus tours',
        'electric tuk-tuk',
        'segways',
        'electric trike bikes',
        'electric scooters',
        'electric bicycles'
    ]
],
'education-tourism' => [
    'greek language' => [],
    'study abroad programs' => []
]

```

```

],
'enrichment-tourism' => [
  'learning sessions' => [
    'cultural seminars',
    'environmental seminars',
    'art seminars',
    'educational seminars',
    'wine seminars'
  ],
  'teambuilding' => [
    'various teambuilding activities'
  ],
  'conference tourism' => [
    'events',
    'meetings',
    'incentives',
    'conferences & exhibition (mice) tourism'
  ],
  'school excursion' => []
],
'sacred-site-visits-tourism' => [
  'monastery tours' => [],
  'masses' => [],
  'biblical tours' => []
],
'medical-tourism' => [
  'surgeries' => [],
  'dental tourism' => [],
  'fertility tourism' => [],
  'cancer treatment' => [],
  'heart care' => [],
  'rehabilitation' => []
],
'spa-tourism' => [
  'natural spa' => [],
  'massage' => []
],
'wellness-tourism' => [
  'cosmetic surgeries' => [],
  'mental health' => [],
  'nature-based wellness' => []
]
];

protected $correlations = [
  'cultural-urban-tourism' => ['gastronomic-tourism', 'greek
gastronomy', 'cooking lessons', 'food festival and market visits',

```

```

'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'wine seminars', 'wine festivals', 'cultural-urban-
exp-tourism', 'museum tours', 'historical and architectural sites',
'live shows', 'tours on wheels', 'hop-on hop-off bus tours', 'electric
trike bikes', 'electric tuk-tuk', 'segways', 'electric scooters',
'electric bicycles', 'local arts and crafts', 'ceramic and pottery
artistry', 'mosaic artistry', 'marble carving', 'wedding-tourism',
'wedding ceremony organizations', 'cultural-festival-tourism',
'festivals', 'carnivals', 'dark-tourism', 'tragic sites', 'haunt
tours', 'war museum tours', 'city-games-tourism', 'ancient greek
games', 'escape city games', 'mythical hunts', 'sacred-site-visits-
tourism', 'monastery tours', 'masses', 'biblical tours', 'fertility
tourism', 'volunteering', 'sporting-events-tourism', 'marathon',
'acropolis rally'],
    'gastronomic-tourism' => ['cultural-urban-tourism', 'greek
gastronomy', 'cooking lessons', 'food festival and market visits',
'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',
'wedding-tourism', 'wedding ceremony organizations', 'cultural-
festival-tourism', 'festivals', 'carnivals', 'agritourism', 'farm
visits', 'collecting land products', 'city tourism', 'urban-
entertainment-tourism'],
    'greek gastronomy' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'cooking lessons', 'food festival and market visits', 'winery
tours', 'wine-tourism', 'cellar visits', 'vineyard and breweries
tours', 'harvesting', 'wine seminars', 'wine festivals', 'wedding-
tourism', 'wedding ceremony organizations', 'cultural-festival-
tourism', 'festivals', 'carnivals', 'agritourism', 'farm visits',
'collecting land products'],
    'cooking lessons' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits',
'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',
'agritourism', 'farm visits', 'collecting land products', 'learning
sessions'],
    'food festival and market visits' => ['cultural-urban-tourism',
'gastronomic-tourism', 'greek gastronomy', 'cooking lessons', 'winery
tours', 'wine-tourism', 'cellar visits', 'vineyard and breweries
tours', 'harvesting', 'wine seminars', 'wine festivals', 'cultural-
urban-exp-tourism', 'cultural-festival-tourism', 'festivals',
'carnivals', 'agritourism', 'farm visits', 'collecting land products'],
    'winery tours' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',
'cultural-urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city

```

```

tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'wine-tourism' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'winery tours', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',
'cultural-urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city
tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'cellar visits' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'winery tours', 'wine-tourism', 'vineyard and breweries
tours', 'harvesting', 'wine seminars', 'wine festivals', 'cultural-
urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city
tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'vineyard and breweries tours' => ['cultural-urban-tourism',
'gastronomic-tourism', 'greek gastronomy', 'cooking lessons', 'food
festival and market visits', 'winery tours', 'wine-tourism', 'cellar
visits', 'harvesting', 'wine seminars', 'wine festivals', 'cultural-
urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city
tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'harvesting' => ['gastronomic-tourism', 'greek gastronomy',
'cooking lessons', 'food festival and market visits', 'winery tours',
'wine-tourism', 'cellar visits', 'vineyard and breweries tours', 'wine
seminars', 'wine festivals', 'festivals', 'nature-based wellness',
'ecotourism', 'agritourism', 'farm visits', 'collecting land products',
'interactions with animals', 'ecotourism', 'flora and fauna', 'tree
planting'],
    'wine seminars' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'winery tours', 'wine-tourism', 'cellar visits',
'vineyard and breweries tours', 'harvesting', 'wine festivals'],
    'wine festivals' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits',
'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'cultural-urban-exp-
tourism', 'live shows', 'cultural-festival-tourism', 'festivals',
'carnivals', 'city tourism', 'urban-entertainment-tourism', 'visiting
clubs and bars', 'shopping tour'],

```

```

    'cultural-urban-exp-tourism' => ['cultural-urban-tourism',
'gastronomic-tourism', 'greek gastronomy', 'food festival and market
visits', 'wine festivals', 'museum tours', 'historical and
architectural sites', 'live shows', 'tours on wheels', 'hop-on hop-off
bus tours', 'electric trike bikes', 'electric tuk-tuk', 'segways',
'electric scooters', 'electric bicycles', 'local arts and crafts',
'ceramic and pottery artistry', 'mosaic artistry', 'marble carving',
'cultural-festival-tourism', 'festivals', 'carnivals', 'city-games-
tourism', 'ancient greek games', 'escape city games', 'mythical hunts',
'volunteering', 'exploration-tourism', 'marathon', 'acropolis rally',
'adventures-tourism', 'city tourism', 'urban-entertainment-tourism',
'visiting clubs and bars', 'shopping tour'],
    'museum tours' => ['cultural-urban-tourism'],
    'historical and architectural sites' => ['cultural-urban-
tourism', 'food festival and market visits'],
    'live shows' => ['cultural-urban-tourism', 'food festival and
market visits'],
    'tours on wheels' => ['cultural-urban-tourism', 'food festival
and market visits'],
    'hop-on hop-off bus tours' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'electric trike bikes' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'electric tuk-tuk' => ['cultural-urban-tourism', 'food festival
and market visits'],
    'segways' => ['cultural-urban-tourism', 'food festival and
market visits'],
    'electric scooters' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'electric bicycles' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'local arts and crafts' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'ceramic and pottery artistry' => ['cultural-urban-tourism'],
    'mosaic artistry' => ['cultural-urban-tourism'],
    'marble carving' => ['cultural-urban-tourism'],
    'wedding-tourism' => ['cultural-urban-tourism', 'greek
gastronomy'],
    'wedding ceremony organizations' => ['cultural-urban-tourism',
'greek gastronomy'],
    'cultural-festival-tourism' => ['cultural-urban-tourism',
'greek gastronomy', 'food festival and market visits'],
    'festivals' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits'],
    'carnivals' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits'],
    'dark-tourism' => ['cultural-urban-tourism'],
    'tragic sites' => ['cultural-urban-tourism'],

```

```

    'haunt tours' => ['cultural-urban-tourism'],
    'war museum tours' => ['cultural-urban-tourism'],
    'city-games-tourism' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'ancient greek games' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'escape city games' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'mythical hunts' => ['cultural-urban-tourism', 'food festival
and market visits'],
    'sacred-site-visits-tourism' => ['cultural-urban-tourism'],
    'monastery tours' => ['cultural-urban-tourism'],
    'masses' => ['cultural-urban-tourism'],
    'biblical tours' => ['cultural-urban-tourism'],
    'health tourism' => [],
    'medical tourism' => [],
    'surgeries' => [],
    'dental tourism' => [],
    'fertility tourism' => [],
    'cancer treatment' => [],
    'heart care' => [],
    'rehabilitation' => [],
    'spa tourism' => [],
    'natural spa' => [],
    'massage' => [],
    'cosmetic surgeries' => ['cultural-urban-tourism'],
    'nature-based wellness' => [],
    'ecotourism' => [],
    'agritourism' => [],
    'farm visits' => [],
    'collecting land products' => [],
    'interactions with animals' => [],
    'horse riding' => [],
    'flora and fauna' => [],
    'multimodal exploration' => [],
    'rafts' => [],
    'inflatable kayaks' => [],
    'volunteering' => ['cultural-urban-tourism'],
    'beach cleanup' => [],
    'animal rescue' => [],
    'tree planting' => [],
    'mountain hiking' => [],
    'camping' => [],
    'exploration-tourism' => [],
    'hiking tours' => [],
    'geotourism' => [],
    'spelunking' => [],
    'geological sites' => [],

```

```

'fishing-tourism' => [],
'fishing on boat' => [],
'local aquatic ecosystems education' => [],
'sports tourism' => [],
'sporting-events-tourism' => [],
'marathon' => ['cultural-urban-tourism'],
'acropolis rally' => ['cultural-urban-tourism'],
'adventures-tourism' => [],
'target sports' => [],
'laser tag' => [],
'archery' => [],
'paintball' => [],
'ball-based sports' => [],
'golf' => [],
'tennis' => [],
'football' => [],
'basketball' => [],
'voley ball' => [],
'snow activities' => [],
'skiing' => [],
'snowboarding' => [],
'snowshoeing' => [],
'off-road' => [],
'quad bike' => [],
'enduro' => [],
'water-activities-tourism' => [],
'motorized water sports' => [],
'jet skiing' => [],
'water skiing' => [],
'wakeboarding' => [],
'powerboating' => [],
'aquatic adventure sports' => [],
'river trekking' => [],
'rafting' => [],
'sea kayak' => [],
'windsurfing' => [],
'kite surfing' => [],
'paddleboarding' => [],
'canyoning' => [],
'sailing' => [],
'underwater activities' => [],
'snorkeling' => [],
'diving' => [],
'sea bob' => [],
'cruises' => [],
'air-activities-tourism' => [],
'freefall experiences' => [],
'paraglide' => [],

```

```

        'parachute' => [],
        'bungee jumping' => [],
        'aerial riding' => [],
        'tours in helicopters' => [],
        'hot air balloons' => [],
        'plane spotting' => [],
        'city tourism' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'urban-entertainment-tourism' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'visiting clubs and bars' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'shopping tour' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'educational tourism' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'education-tourism' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'greek language' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'study abroad programs' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'enrichment-tourism' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'learning sessions' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'seminars' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'teambuilding' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'conference tourism ' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'events' => ['cultural-urban-tourism', 'gastronomic-tourism'],
        'meetings' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'incentives' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'conferences & exhibition (mice) tourism' => ['cultural-urban-
tourism', 'gastronomic-tourism']
    ];

    public function generateHotelRecommendations(Request $request) {
        //$recs = [];
        $input = $request->all();

        $startDate = $input['start_date'];
        $endDate = $input['end_date'];
        $recNum = $input['number_of_recommendations'];
    }

```

```

        //$location = $input['location'];

        $filters = [
            'start_date' => $input['start_date'],
            'end_date' => $input['end_date'],
            'adults' => 1,
            'children' => 0
        ];

        $serviceModel=$this->serviceTypeMap['hotel'];

        $hotels=Hotel::inRandomOrder()->get();

        //$locationHotels = $hotels->where('primary_town', $location)-
>get();
        $availableHotels = [];
        //$availableAndLocationHotels =[];

        $recHotels = [];

        foreach ($hotels as $hotel) {
            //check availability here and put whichever ones have free
rooms in the list
            $availableRooms=$hotel->getRoomsAvailability($filters);
            if (count($availableRooms)>0) {
                $availableHotels[] = $hotel;
                //if ($locationHotels.contains($hotel)){
                //    $availableAndLocationHotels[] = $hotel;
                //}
            }
        }

        /*
        if ($availableAndLocationHotels<$recNum){
            foreach($availableAndLocationHotels as $avLocHotel){
                $recHotels[]=$avLocHotel;
            }
            foreach($availableHotels as $avHotel)
            {
                if (!in_array($avHotel,$recHotels)){
                    $recHotels[]=$avHotel;
                }
                if(count($recHotels)==$recNum){
                    break;
                }
            }
        }
    }

```

```

        */
        $recHotels=array_slice($availableHotels, 0, $recNum);

        return $this->sendResponse($recHotels, 'Recommendations');
    }

    public function generateRecommendations(Request $request) {
        $recs = [];

        $input = $request->all();
        $requestServiceType = $input['service_type'];
        $objectId = $input['object_id'];
        $recNum = $input['number_of_recommendations'];

        $serviceModel = $this->serviceTypeMap[$requestServiceType];

        $object = $serviceModel::find($objectId);

        $location = $object->primary_town;

        $tourism1 = $requestServiceType;
        $tourism2 = strtolower($object->tourism_sub_category);

        $subCat4Exists = false;
        $tourism3 = '';
        if (!empty($object->tourism_sub_category_4)) {
            $subCat4Exists = true;
            $tourism3 = strtolower($object->tourism_sub_category_4);
        }

        if ($subCat4Exists && array_key_exists($tourism3, $this->correlations)) {
            $correlationArray = $this->correlations[$tourism3];
        } elseif (array_key_exists($tourism2, $this->correlations)) {
            $correlationArray = $this->correlations[$tourism2];
        } else {
            $correlationArray = $this->correlations[$tourism1];
        }

        $recCount = 0;

        if (count($correlationArray) == 0) {
            // If there's no correlations at all, pick randomly

            $shuffledServices = array_keys($this->serviceTypeMap);
            shuffle($shuffledServices);
        }
    }

```

```

        $i = 0;
        $loop = false;
        $loopBreak = 0;

        while ($recCount < $recNum) {
            if ($loopBreak >= $recNum) {
                break;
            }
            if ($i < count($shuffledServices)) {
                $recServiceType = $shuffledServices[$i];
            } else {
                $loop = true;
                $loopBreak += 1;
                $i = 0;
            }

            $recModel = $this->serviceTypeMap[$recServiceType];
            $recResourceType = $this->serviceResourceMap[$recServiceType];

            // Fetch only active services
            $recInstances = $recModel::where('status', 'active')->inRandomOrder()->get();

            if (!$loop) {
                // Prefer ones in the same location during the
                first loop
                $recInstance = $recInstances->where('primary_town', $location)->first();
            } else {
                // Expand search outside of location
                $recInstance = $recInstances->first();
            }

            if ($recInstance == null) {
                $i += 1;
                continue;
            } else {
                $recResource = new $recResourceType($recInstance);

                $recData = [
                    'service_type' => $recServiceType,
                    'rec' => $recResource
                ];

                $recs[] = $recData;
                $recCount += 1;
            }
        }

```

```

        $i += 1;
    }
} else {
    // Handle correlations
    $shuffledCorrelations = $correlationArray;
    shuffle($shuffledCorrelations);

    $i = 0;
    $loop = false;
    $loopBreak = 0;

    while ($recCount < $recNum) {
        if ($loopBreak >= $recNum) {
            break;
        }
        if ($i < count($shuffledCorrelations)) {
            $corr = $shuffledCorrelations[$i];
        } else {
            $loop = true;
            $i = 0;
            $loopBreak += 1;
        }

        if (array_key_exists($corr, $this->ontology)) {
            // General tourism type, pick a random active
service
            $recModel = $this->serviceTypeMap[$corr];
            $recResourceType = $this->
>serviceResourceMap[$corr];

            $recInstances = $recModel::where('status',
'active')->inRandomOrder()->get();

            if (!$loop) {
                $recInstance = $recInstances->
>where('primary_town', $location)->first();
            } else {
                $recInstance = $recInstances->first();
            }

            if ($recInstance == null) {
                $i += 1;
                continue;
            } else {
                $recResource = new
$recResourceType($recInstance);

```

```

        $recData = [
            'service_type' => $corr,
            'rec' => $recResource
        ];

        $recs[] = $recData;
        $recCount += 1;
    }
} else {
    // Handle subtypes
    $corrHeight = 1;
    $corrServiceType = '';
    foreach ($this->ontology as $service =>
$subcategories) {
        if (array_key_exists($corr, $subcategories)) {
            $corrServiceType = $service;
            break;
        } else {
            foreach ($subcategories as $key => $value)
{
                if (in_array($corr, $value)) {
                    $corrHeight = 2;
                    $corrServiceType = $service;
                    break;
                }
            }
        }
    }

    if ($corrServiceType === '') {
        $i += 1;
        continue;
    }

    $recModel = $this->
>serviceTypeMap[$corrServiceType];
    $recResourceType = $this->
>serviceResourceMap[$corrServiceType];

    if ($corrHeight == 1) {
        $recInstances =
$recModel::where('tourism_sub_category', 'LIKE', '%' . $corr . '%')
->where('status', 'active')
->inRandomOrder();
    } else {
        $recInstances =
$recModel::whereRaw('LOWER(tourism_sub_category_4) LIKE ?', ['%' .
strtolower($corr) . '%'])

```

```

        ->where('status', 'active')
        ->inRandomOrder();
    }

    if (!$loop) {
        $recInstance = $recInstances-
>where('primary_town', $location)->first();
    } else {
        $recInstance = $recInstances->first();
    }

    if ($recInstance == null) {
        $i += 1;
        continue;
    } else {
        $recResource = new
$recResourceType($recInstance);

        $recData = [
            'service_type' => $corrServiceType,
            'rec' => $recResource
        ];

        $recs[] = $recData;
        $recCount += 1;
    }
}

$i += 1;
}

return $this->sendResponse($recs, 'Recommendations');
}

public function generateRandomRecommendations(Request $request)
{
    $recs = [];

    $input = $request->all();
    $requestType = $input['service_type'];
    $recNum = $input['number_of_recommendations'];

    $recCount=0;

    while ($recCount<$recNum)
    {

```

```

        $recServiceType = array_rand($this->serviceTypeMap);
        $recModel = $this->serviceTypeMap[$recServiceType];
        $recResourceType = $this->
>serviceResourceMap[$recServiceType];

        $recInstance = $recModel::inRandomOrder()->first();
        if ($recInstance)
        {
            $recCount=$recCount+1;
            $recResource =new $recResourceType($recInstance);

            $recData = [
                'service_type' => $recServiceType,
                'rec' => $recResource
            ];

            $recs[] = $recData;
        }
    }

    return $this->sendResponse($recs, 'Recommendations');

}

public function generateTransportationRecommendations(Request
$request) {
    $input = $request->all();

    $recNum = $input['number_of_recommendations'];
    $location = $input['location']; // Input: Location filter

    // Fetch all transportations randomly
    $transportations = Transportation::inRandomOrder()->get();

    $locationTransportations = [];
    $recommendedTransportations = [];

    // Filter transportations based on location
    foreach ($transportations as $transportation) {
        if (
            strpos($transportation->service_area, $location) !==
false
        ) {
            $locationTransportations[] = $transportation;
        }
    }
}

```

```

        // Slice results to return only the desired number of
recommendations
        $recommendedTransportations =
array_slice($locationTransportations, 0, $recNum);

        return $this->sendResponse(TransportationResource::collection($recommendedTransporta
tions), 'Transportation Recommendations');
    }
}

```

Επισκόπηση

Ο TranslationController διαχειρίζεται τα requests μετάφρασης για διάφορους τύπους τουριστικών υπηρεσιών. Υποστηρίζει τόσο την αποστολή αιτημάτων για μετάφραση κειμένων και εγγράφων όσο και την παρακολούθηση της κατάστασης των αιτημάτων μετάφρασης, την επεξεργασία των μεταφρασμένων δεδομένων και την αποθήκευση τους στη βάση δεδομένων.

Εξαρτήσεις (Dependencies)

- **BaseController:** Βασικός controller για τυποποιημένα responses.
- **TranslationRequest:** Μοντέλο για τη διαχείριση των αιτήσεων μετάφρασης.
- **Http:** Για αποστολή αιτημάτων HTTP προς την υπηρεσία μετάφρασης.
- **Log:** Για την καταγραφή συμβάντων και σφαλμάτων.
- **DB:** Χρησιμοποιείται για τις συναλλαγές στη βάση δεδομένων.

Μέθοδοι

1. sendTranslationRequest

- **Σκοπός:** Αποστολή αιτήματος μετάφρασης για κείμενα.
- **Παράμετροι:**
 - **object_model:** Ο τύπος του αντικειμένου (π.χ. wine_tourism).
 - **object_id:** Το ID του αντικειμένου που πρέπει να μεταφραστεί.
 - **target_languages:** Οι γλώσσες στις οποίες πρέπει να μεταφραστεί το κείμενο.
- **Λειτουργία:**
 - Ανάκτηση του αντικειμένου και των στοιχείων του (π.χ. τίτλος, περιγραφή, κλπ).
 - Αποστολή αίτησης στη υπηρεσία μετάφρασης μέσω API.

- Δημιουργία εγγραφής στην βάση δεδομένων για την παρακολούθηση της αίτησης.
- **Απάντηση:** Επιστρέφει το μεταφρασμένο κείμενο ή σφάλμα αν κάτι πάει στραβά.

2. `sendTranslationDocumentRequest`

- **Σκοπός:** Αποστολή αιτήματος μετάφρασης για έγγραφα (με μορφοποίηση HTML).
- **Παράμετροι:**
 - `object_model`: Ο τύπος του αντικειμένου.
 - `object_id`: Το ID του αντικειμένου.
 - `target_languages`: Οι γλώσσες προορισμού.
- **Λειτουργία:**
 - Συναρμολόγηση του κειμένου με HTML tags.
 - Κωδικοποίηση του κειμένου σε base64.
 - Αποστολή της αίτησης στη υπηρεσία μετάφρασης.
- **Απάντηση:** Επιστρέφει την επιτυχία της αίτησης ή σφάλμα.

3. `handleIncomingTranslation`

- **Σκοπός:** Επεξεργασία των μεταφρασμένων κειμένων που επιστρέφονται από την υπηρεσία μετάφρασης.
- **Παράμετροι:** Λαμβάνει τα δεδομένα του μεταφρασμένου κειμένου και τη γλώσσα.
- **Λειτουργία:**
 - Ανάκτηση της αίτησης από τη βάση δεδομένων με βάση το ID.
 - Ενημέρωση του αντικειμένου με τις μεταφρασμένες τιμές.
- **Απάντηση:** Δεν επιστρέφει κάτι, αλλά ενημερώνει τα δεδομένα της βάσης.

4. `handleIncomingDocumentTranslation`

- **Σκοπός:** Επεξεργασία των μεταφρασμένων εγγράφων.
- **Παράμετροι:** Λαμβάνει το περιεχόμενο του μεταφρασμένου εγγράφου σε μορφή base64.
- **Λειτουργία:**
 - Ανάκτηση και αποκωδικοποίηση του κειμένου.
 - Ενημέρωση του τουριστικού αντικειμένου με τις μεταφρασμένες πληροφορίες.

- **Απάντηση:** Δεν επιστρέφει κάτι, αλλά ενημερώνει τα δεδομένα της βάσης.

5. checkTranslationStatus

- **Σκοπός:** Έλεγχος της κατάστασης μιας αίτησης μετάφρασης.
- **Παράμετροι:**
 - request_id: Το ID της αίτησης μετάφρασης.
- **Λειτουργία:**
 - Επιστρέφει αν η μετάφραση είναι ολοκληρωμένη ή αν υπάρχουν εκκρεμότητες.
- **Απάντηση:** Μήνυμα που υποδεικνύει την κατάσταση της μετάφρασης.

6. getAvailableLocales

- **Σκοπός:** Ανάκτηση όλων των διαθέσιμων γλωσσών (locales) για ένα συγκεκριμένο αντικείμενο.
- **Παράμετροι:**
 - request_id: Το ID του αντικειμένου.
 - request_object_model: Ο τύπος του αντικειμένου.
- **Λειτουργία:**
 - Ανάκτηση του αντικειμένου και των διαθέσιμων γλωσσών για αυτό.
- **Απάντηση:** Λίστα με τις διαθέσιμες γλώσσες.

```
<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\API\BaseController as BaseController;
use Illuminate\Http\Request;
use Modules\Booking\Models\Booking;
use Illuminate\Support\Str;
use Illuminate\Support\Facades\Log;
use Modules\Transportation\Models\Transportation;
use Modules\Transportation\Resources\TransportationResource;
use App\Models\Hotel;

class RecommendationsController extends BaseController
{
    protected $serviceTypeMap = [
        'wine-tourism' =>
        \Modules\WineTourism\Models\WineTourism::class,
```

```

        'gastronomic-tourism' =>
\Modules\GastronomicTourism\Models\GastronomicTourism::class,
        'cultural-urban-tourism' =>
\Modules\CulturalUrbanTourism\Models\CulturalUrbanTourism::class,
        'wedding-tourism' =>
\Modules\WeddingTourism\Models\WeddingTourism::class,
        'cultural-festival-tourism' =>
\Modules\CulturalFestivalTourism\Models\CulturalFestivalTourism::class,
        'dark-tourism' =>
\Modules\DarkTourism\Models\DarkTourism::class,
        'city-games-tourism' =>
\Modules\CityGamesTourism\Models\CityGamesTourism::class,
        'sacred-site-visits-tourism' =>
\Modules\SacredSiteVisitsTourism\Models\SacredSiteVisitsTourism::class,
        'agritourism' =>
\Modules\Agritourism\Models\Agritourism::class,
        'ecotourism' => \Modules\Ecotourism\Models\Ecotourism::class,
        'exploration-tourism' =>
\Modules\ExplorationTourism\Models\ExplorationTourism::class,
        'geotourism' => \Modules\Geotourism\Models\Geotourism::class,
        'fishing-tourism' =>
\Modules\FishingTourism\Models\FishingTourism::class,
        'sporting-events-tourism' =>
\Modules\SportingEventsTourism\Models\SportingEventsTourism::class,
        'adventures-tourism' =>
\Modules\AdventuresTourism\Models\AdventuresTourism::class,
        'water-activities-tourism' =>
\Modules\WaterActivitiesTourism\Models\WaterActivitiesTourism::class,
        'air-activities-tourism' =>
\Modules\AirActivitiesTourism\Models\AirActivitiesTourism::class,
        'urban-entertainment-tourism' =>
\Modules\UrbanEntertainmentTourism\Models\UrbanEntertainmentTourism::cl
ass,
        'education-tourism' =>
\Modules\EducationDevelopmentTourism\Models\EducationDevelopmentTourism
::class,
        'enrichment-tourism' =>
\Modules\EnrichmentSkillBuildingTourism\Models\EnrichmentSkillBuildingT
ourism::class,
        'cultural-urban-exp-tourism' =>
\Modules\CulturalUrbanExplorationTourism\Models\CulturalUrbanExploratio
nTourism::class,
        'medical-tourism' =>
\Modules\MedicalTourism\Models\MedicalTourism::class,
        'spa-tourism' => \Modules\SpaTourism\Models\SpaTourism::class,
        'wellness-tourism' =>
\Modules\WellnessTourism\Models\WellnessTourism::class,
        'hotel' => \App\Models\Hotel::class

```

```

];

protected $serviceResourceMap = [
    'wine-tourism' =>
    \Modules\WineTourism\Resources\WineTourismResource::class,
    'gastronomic-tourism' =>
    \Modules\GastronomicTourism\Resources\GastronomicTourismResource::class
    ,
    'cultural-urban-tourism' =>
    \Modules\CulturalUrbanTourism\Resources\CulturalUrbanTourismResource::c
lass,
    'wedding-tourism' =>
    \Modules\WeddingTourism\Resources\WeddingTourismResource::class,
    'cultural-festival-tourism' =>
    \Modules\CulturalFestivalTourism\Resources\CulturalFestivalTourismResou
rce::class,
    'dark-tourism' =>
    \Modules\DarkTourism\Resources\DarkTourismResource::class,
    'city-games-tourism' =>
    \Modules\CityGamesTourism\Resources\CityGamesTourismResource::class,
    'sacred-site-visits-tourism' =>
    \Modules\SacredSiteVisitsTourism\Resources\SacredSiteVisitsTourismResou
rce::class,
    'agritourism' =>
    \Modules\Agritourism\Resources\AgritourismResource::class,
    'ecotourism' =>
    \Modules\Ecotourism\Resources\EcotourismResource::class,
    'exploration-tourism' =>
    \Modules\ExplorationTourism\Resources\ExplorationTourismResource::class
    ,
    'geotourism' =>
    \Modules\Geotourism\Resources\GeotourismResource::class,
    'fishing-tourism' =>
    \Modules\FishingTourism\Resources\FishingTourismResource::class,
    'sporting-events-tourism' =>
    \Modules\SportingEventsTourism\Resources\SportingEventsTourismResource:
:~class,
    'adventures-tourism' =>
    \Modules\AdventuresTourism\Resources\AdventuresTourismResource::class,
    'water-activities-tourism' =>
    \Modules\WaterActivitiesTourism\Resources\WaterActivitiesTourismResourc
e::class,
    'air-activities-tourism' =>
    \Modules\AirActivitiesTourism\Resources\AirActivitiesTourismResource::c
lass,
    'urban-entertainment-tourism' =>
    \Modules\UrbanEntertainmentTourism\Resources\UrbanEntertainmentTourismR
esource::class,

```

```

        'education-tourism' =>
\Modules\EducationDevelopmentTourism\Resources\EducationDevelopmentTour
ismResource::class,
        'enrichment-tourism' =>
\Modules\EnrichmentSkillBuildingTourism\Resources\EnrichmentSkillBuildi
ngTourismResource::class,
        'cultural-urban-exp-tourism' =>
\Modules\CulturalUrbanExplorationTourism\Resources\CulturalUrbanExplora
tionTourismResource::class,
        'medical-tourism' =>
\Modules\MedicalTourism\Resources\MedicalTourismResource::class,
        'spa-tourism' =>
\Modules\SpaTourism\Resources\SpaTourismResource::class,
        'wellness-tourism' =>
\Modules\WellnessTourism\Resources\WellnessTourismResource::class,
        'hotel' => \App\Http\Resources\HotelResource::class,

];

protected $ontology = [
    'adventures-tourism' => [
        'target sports' => [
            'laser tag',
            'archery',
            'paintball'
        ],
        'ball-based sports' => [
            'golf',
            'tennis',
            'football',
            'basketball',
            'volleyball'
        ],
        'snow activities' => [
            'skiing',
            'snowboarding',
            'snowshoeing'
        ],
        'off-road activities' => [
            'quad bike',
            'enduro'
        ]
    ],
    'air-activities-tourism' => [
        'freefall experiences' => [
            'paraglide',
            'parachute',
            'bungee jumping'
        ]
    ]
];

```

```

    ],
    'aerial riding' => [
        'tours in helicopters',
        'hot air balloons'
    ],
    'plane spotting' => []
],
'sporting-events-tourism' => [
    'marathon' => [],
    'acropolis rally' => []
],
'water-activities-tourism' => [
    'motorized water sports' => [
        'jet skiing',
        'water skiing',
        'wakeboarding',
        'powerboating'
    ],
    'aquatic adventure sports' => [
        'river trekking',
        'rafting',
        'sea kayak',
        'windsurfing',
        'kite surfing',
        'paddleboarding',
        'canyoning',
        'sailing'
    ],
    'underwater activities' => [
        'snorkeling',
        'diving',
        'sea bob'
    ],
    'cruises' => []
],
'agritourism' => [
    'farm visits' => [],
    'collecting land products' => [],
    'interactions with animals' => [],
    'horse riding' => []
],
'ecotourism' => [
    'flora and fauna' => [
        'wildlife tracking',
        'bird watching',
        'wildflower watching'
    ],
    'multimodal exploration' => [

```

```

        'rafts',
        'inflatable kayaks',
        'bicycles'
    ],
    'volunteering' => [
        'beach cleanup',
        'animal rescue',
        'tree planting'
    ],
    'mountain hiking' => [],
    'camping' => []
],
'exploration-tourism' => [
    'hiking tours' => [],
    'flora and fauna' => [
        'wildlife tracking',
        'bird watching',
        'wildflower watching'
    ],
    'multimodal exploration' => [
        'rafts',
        'inflatable kayaks',
        'bicycles'
    ]
],
'fishing-tourism' => [
    'fishing on boat' => [],
    'local aquatic ecosystems education' => []
],
'geotourism' => [
    'geological sites' => [],
    'spelunking' => []
],
'city-games-tourism' => [
    'ancient greek games' => [],
    'escape city games' => [],
    'mythical hunts' => []
],
'cultural-festival-tourism' => [
    'festivals' => [],
    'carnivals' => []
],
'cultural-urban-exp-tourism' => [
    'museum tours' => [],
    'historical and architectural sites' => [],
    'live shows' => [],
    'tours on wheels' => [
        'hop-on hop-off bus tours',

```

```

        'electric tuk-tuk',
        'segways',
        'electric trike bikes',
        'electric scooters',
        'electric bicycles'
    ],
    'local arts and crafts' => [
        'ceramic and pottery artistry',
        'mosaic artistry',
        'marble carving'
    ]
],
'dark-tourism' => [
    'tragic sites' => [],
    'haunt tours' => [],
    'war museum tours' => []
],
'gastronomic-tourism' => [
    'greek gastronomy' => [],
    'cooking lessons' => [],
    'food festival and market visits' => [],
    'winery tours' => []
],
'wedding-tourism' => [
    'wedding ceremony organizations' => []
],
'wine-tourism' => [
    'cellar visits' => [],
    'vineyard and breweries tours' => [],
    'harvesting' => [],
    'wine seminars' => [],
    'wine festivals' => []
],
'cultural-urban-tourism' => [
    'cultural and artistic tours' => []
],
'urban-entertainment-tourism' => [
    'visiting clubs and bars' => [],
    'shopping tour' => [],
    'tours on wheels' => [
        'hop-on hop-off bus tours',
        'electric tuk-tuk',
        'segways',
        'electric trike bikes',
        'electric scooters',
        'electric bicycles'
    ]
],
],

```

```

    'education-tourism' => [
        'greek language' => [],
        'study abroad programs' => []
    ],
    'enrichment-tourism' => [
        'learning sessions' => [
            'cultural seminars',
            'environmental seminars',
            'art seminars',
            'educational seminars',
            'wine seminars'
        ],
        'teambuilding' => [
            'various teambuilding activities'
        ],
        'conference tourism' => [
            'events',
            'meetings',
            'incentives',
            'conferences & exhibition (mice) tourism'
        ],
        'school excursion' => []
    ],
    'sacred-site-visits-tourism' => [
        'monastery tours' => [],
        'masses' => [],
        'biblical tours' => []
    ],
    'medical-tourism' => [
        'surgeries' => [],
        'dental tourism' => [],
        'fertility tourism' => [],
        'cancer treatment' => [],
        'heart care' => [],
        'rehabilitation' => []
    ],
    'spa-tourism' => [
        'natural spa' => [],
        'massage' => []
    ],
    'wellness-tourism' => [
        'cosmetic surgeries' => [],
        'mental health' => [],
        'nature-based wellness' => []
    ]
];

```

```

protected $correlations = [
    'cultural-urban-tourism' => ['gastronomic-tourism', 'greek
gastronomy', 'cooking lessons', 'food festival and market visits',
'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'wine seminars', 'wine festivals', 'cultural-urban-
exp-tourism', 'museum tours', 'historical and architectural sites',
'live shows', 'tours on wheels', 'hop-on hop-off bus tours', 'electric
trike bikes', 'electric tuk-tuk', 'segways', 'electric scooters',
'electric bicycles', 'local arts and crafts', 'ceramic and pottery
artistry', 'mosaic artistry', 'marble carving', 'wedding-tourism',
'wedding ceremony organizations', 'cultural-festival-tourism',
'festivals', 'carnivals', 'dark-tourism', 'tragic sites', 'haunt
tours', 'war museum tours', 'city-games-tourism', 'ancient greek
games', 'escape city games', 'mythical hunts', 'sacred-site-visits-
tourism', 'monastery tours', 'masses', 'biblical tours', 'fertility
tourism', 'volunteering', 'sporting-events-tourism', 'marathon',
'acropolis rally'],
    'gastronomic-tourism' => ['cultural-urban-tourism', 'greek
gastronomy', 'cooking lessons', 'food festival and market visits',
'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',
'wedding-tourism', 'wedding ceremony organizations', 'cultural-
festival-tourism', 'festivals', 'carnivals', 'agritourism', 'farm
visits', 'collecting land products', 'city tourism', 'urban-
entertainment-tourism'],
    'greek gastronomy' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'cooking lessons', 'food festival and market visits', 'winery
tours', 'wine-tourism', 'cellar visits', 'vineyard and breweries
tours', 'harvesting', 'wine seminars', 'wine festivals', 'wedding-
tourism', 'wedding ceremony organizations', 'cultural-festival-
tourism', 'festivals', 'carnivals', 'agritourism', 'farm visits',
'collecting land products'],
    'cooking lessons' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits',
'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',
'agritourism', 'farm visits', 'collecting land products', 'learning
sessions'],
    'food festival and market visits' => ['cultural-urban-tourism',
'gastronomic-tourism', 'greek gastronomy', 'cooking lessons', 'winery
tours', 'wine-tourism', 'cellar visits', 'vineyard and breweries
tours', 'harvesting', 'wine seminars', 'wine festivals', 'cultural-
urban-exp-tourism', 'cultural-festival-tourism', 'festivals',
'carnivals', 'agritourism', 'farm visits', 'collecting land products'],
    'winery tours' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',

```

```

'cultural-urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city
tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'wine-tourism' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'winery tours', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'wine festivals',
'cultural-urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city
tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'cellar visits' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'winery tours', 'wine-tourism', 'vineyard and breweries
tours', 'harvesting', 'wine seminars', 'wine festivals', 'cultural-
urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city
tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'vineyard and breweries tours' => ['cultural-urban-tourism',
'gastronomic-tourism', 'greek gastronomy', 'cooking lessons', 'food
festival and market visits', 'winery tours', 'wine-tourism', 'cellar
visits', 'harvesting', 'wine seminars', 'wine festivals', 'cultural-
urban-exp-tourism', 'wedding-tourism', 'wedding ceremony
organizations', 'cultural-festival-tourism', 'festivals', 'carnivals',
'agritourism', 'farm visits', 'collecting land products', 'city
tourism', 'urban-entertainment-tourism', 'visiting clubs and bars',
'shopping tour', 'seminars'],
    'harvesting' => ['gastronomic-tourism', 'greek gastronomy',
'cooking lessons', 'food festival and market visits', 'winery tours',
'wine-tourism', 'cellar visits', 'vineyard and breweries tours', 'wine
seminars', 'wine festivals', 'festivals', 'nature-based wellness',
'ecotourism', 'agritourism', 'farm visits', 'collecting land products',
'interactions with animals', 'ecotourism', 'flora and fauna', 'tree
planting'],
    'wine seminars' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'cooking lessons', 'food festival and
market visits', 'winery tours', 'wine-tourism', 'cellar visits',
'vineyard and breweries tours', 'harvesting', 'wine festivals'],
    'wine festivals' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits',
'winery tours', 'wine-tourism', 'cellar visits', 'vineyard and
breweries tours', 'harvesting', 'wine seminars', 'cultural-urban-exp-
tourism', 'live shows', 'cultural-festival-tourism', 'festivals',

```

```

'carnivals', 'city tourism', 'urban-entertainment-tourism', 'visiting
clubs and bars', 'shopping tour'],
    'cultural-urban-exp-tourism' => ['cultural-urban-tourism',
'gastronomic-tourism', 'greek gastronomy', 'food festival and market
visits', 'wine festivals', 'museum tours', 'historical and
architectural sites', 'live shows', 'tours on wheels', 'hop-on hop-off
bus tours', 'electric trike bikes', 'electric tuk-tuk', 'segways',
'electric scooters', 'electric bicycles', 'local arts and crafts',
'ceramic and pottery artistry', 'mosaic artistry', 'marble carving',
'cultural-festival-tourism', 'festivals', 'carnivals', 'city-games-
tourism', 'ancient greek games', 'escape city games', 'mythical hunts',
'volunteering', 'exploration-tourism', 'marathon', 'acropolis rally',
'adventures-tourism', 'city tourism', 'urban-entertainment-tourism',
'visiting clubs and bars', 'shopping tour'],
    'museum tours' => ['cultural-urban-tourism'],
    'historical and architectural sites' => ['cultural-urban-
tourism', 'food festival and market visits'],
    'live shows' => ['cultural-urban-tourism', 'food festival and
market visits'],
    'tours on wheels' => ['cultural-urban-tourism', 'food festival
and market visits'],
    'hop-on hop-off bus tours' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'electric trike bikes' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'electric tuk-tuk' => ['cultural-urban-tourism', 'food festival
and market visits'],
    'segways' => ['cultural-urban-tourism', 'food festival and
market visits'],
    'electric scooters' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'electric bicycles' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'local arts and crafts' => ['cultural-urban-tourism', 'food
festival and market visits'],
    'ceramic and pottery artistry' => ['cultural-urban-tourism'],
    'mosaic artistry' => ['cultural-urban-tourism'],
    'marble carving' => ['cultural-urban-tourism'],
    'wedding-tourism' => ['cultural-urban-tourism', 'greek
gastronomy'],
    'wedding ceremony organizations' => ['cultural-urban-tourism',
'greek gastronomy'],
    'cultural-festival-tourism' => ['cultural-urban-tourism',
'greek gastronomy', 'food festival and market visits'],
    'festivals' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits'],
    'carnivals' => ['cultural-urban-tourism', 'gastronomic-
tourism', 'greek gastronomy', 'food festival and market visits'],

```

```

'dark-tourism' => ['cultural-urban-tourism'],
'tragic sites' => ['cultural-urban-tourism'],
'haunt tours' => ['cultural-urban-tourism'],
'war museum tours' => ['cultural-urban-tourism'],
'city-games-tourism' => ['cultural-urban-tourism', 'food
festival and market visits'],
'ancient greek games' => ['cultural-urban-tourism', 'food
festival and market visits'],
'escape city games' => ['cultural-urban-tourism', 'food
festival and market visits'],
'mythical hunts' => ['cultural-urban-tourism', 'food festival
and market visits'],
'sacred-site-visits-tourism' => ['cultural-urban-tourism'],
'monastery tours' => ['cultural-urban-tourism'],
'masses' => ['cultural-urban-tourism'],
'biblical tours' => ['cultural-urban-tourism'],
'health tourism' => [],
'medical tourism' => [],
'surgeries' => [],
'dental tourism' => [],
'fertility tourism' => [],
'cancer treatment' => [],
'heart care' => [],
'rehabilitation' => [],
'spa tourism' => [],
'natural spa' => [],
'massage' => [],
'cosmetic surgeries' => ['cultural-urban-tourism'],
'nature-based wellness' => [],
'ecotourism' => [],
'agritourism' => [],
'farm visits' => [],
'collecting land products' => [],
'interactions with animals' => [],
'horse riding' => [],
'flora and fauna' => [],
'multimodal exploration' => [],
'rafts' => [],
'inflatable kayaks' => [],
'volunteering' => ['cultural-urban-tourism'],
'beach cleanup' => [],
'animal rescue' => [],
'tree planting' => [],
'mountain hiking' => [],
'camping' => [],
'exploration-tourism' => [],
'hiking tours' => [],
'geotourism' => [],

```

```
'spelunking' => [],
'geological sites' => [],
'fishing-tourism' => [],
'fishing on boat' => [],
'local aquatic ecosystems education' => [],
'sports tourism' => [],
'sporting-events-tourism' => [],
'marathon' => ['cultural-urban-tourism'],
'acropolis rally' => ['cultural-urban-tourism'],
'adventures-tourism' => [],
'target sports' => [],
'laser tag' => [],
'archery' => [],
'paintball' => [],
'ball-based sports' => [],
'golf' => [],
'tennis' => [],
'football' => [],
'basketball' => [],
'volley ball' => [],
'snow activities' => [],
'skiing' => [],
'snowboarding' => [],
'snowshoeing' => [],
'off-road' => [],
'quad bike' => [],
'enduro' => [],
'water-activities-tourism' => [],
'motorized water sports' => [],
'jet skiing' => [],
'water skiing' => [],
'wakeboarding' => [],
'powerboating' => [],
'aquatic adventure sports' => [],
'river trekking' => [],
'rafting' => [],
'sea kayak' => [],
'windsurfing' => [],
'kite surfing' => [],
'paddleboarding' => [],
'canyoning' => [],
'sailing' => [],
'underwater activities' => [],
'snorkeling' => [],
'diving' => [],
'sea bob' => [],
'cruises' => [],
'air-activities-tourism' => [],
```

```

        'freefall experiences' => [],
        'paraglide' => [],
        'parachute' => [],
        'bungee jumping' => [],
        'aerial riding' => [],
        'tours in helicopters' => [],
        'hot air balloons' => [],
        'plane spotting' => [],
        'city tourism' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'urban-entertainment-tourism' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'visiting clubs and bars' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'shopping tour' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'educational tourism' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'education-tourism' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'greek language' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'study abroad programs' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'enrichment-tourism' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'learning sessions' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'seminars' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'teambuilding' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'conference tourism ' => ['cultural-urban-tourism',
'gastronomic-tourism'],
        'events' => ['cultural-urban-tourism', 'gastronomic-tourism'],
        'meetings' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'incentives' => ['cultural-urban-tourism', 'gastronomic-
tourism'],
        'conferences & exhibition (mice) tourism' => ['cultural-urban-
tourism', 'gastronomic-tourism']
    ];

    public function generateHotelRecommendations(Request $request) {
        //$recs = [];
        $input = $request->all();

        $startDate = $input['start_date'];

```

```

$endDate = $input['end_date'];
$recNum = $input['number_of_recommendations'];
//$location = $input['location'];

$filters = [
    'start_date' => $input['start_date'],
    'end_date' => $input['end_date'],
    'adults' => 1,
    'children' => 0
];

$serviceModel=$this->serviceTypeMap['hotel'];

$hotels=Hotel::inRandomOrder()->get();

//$locationHotels = $hotels->where('primary_town', $location)-
>get();
$availableHotels = [];
//$availableAndLocationHotels =[];

$recHotels = [];

foreach ($hotels as $hotel) {
    //check availability here and put whichever ones have free
rooms in the list
    $availableRooms=$hotel->getRoomsAvailability($filters);
    if (count($availableRooms)>0) {
        $availableHotels[] = $hotel;
        //if ($locationHotels.contains($hotel)){
        //    $availableAndLocationHotels[] = $hotel;
        //}
    }
}

/*
if ($availableAndLocationHotels<$recNum){
    foreach($availableAndLocationHotels as $avLocHotel){
        $recHotels[]=$avLocHotel;
    }
    foreach($availableHotels as $avHotel)
    {
        if (!in_array($avHotel,$recHotels)){
            $recHotels[]=$avHotel;
        }
        if(count($recHotels)==$recNum){
            break;
        }
    }
}

```

```

    }
    */
    $recHotels=array_slice($availableHotels, 0, $recNum);

    return $this->sendResponse($recHotels, 'Recommendations');
}

public function generateRecommendations(Request $request) {
    $recs = [];

    $input = $request->all();
    $requestServiceType = $input['service_type'];
    $objectId = $input['object_id'];
    $recNum = $input['number_of_recommendations'];

    $serviceModel = $this->serviceTypeMap[$requestServiceType];

    $object = $serviceModel::find($objectId);

    $location = $object->primary_town;

    $tourism1 = $requestServiceType;
    $tourism2 = strtolower($object->tourism_sub_category);

    $subCat4Exists = false;
    $tourism3 = '';
    if (!empty($object->tourism_sub_category_4)) {
        $subCat4Exists = true;
        $tourism3 = strtolower($object->tourism_sub_category_4);
    }

    if ($subCat4Exists && array_key_exists($tourism3, $this->correlations)) {
        $correlationArray = $this->correlations[$tourism3];
    } elseif (array_key_exists($tourism2, $this->correlations)) {
        $correlationArray = $this->correlations[$tourism2];
    } else {
        $correlationArray = $this->correlations[$tourism1];
    }

    $recCount = 0;

    if (count($correlationArray) == 0) {
        // If there's no correlations at all, pick randomly

        $shuffledServices = array_keys($this->serviceTypeMap);
    }
}

```

```

        shuffle($shuffledServices);

        $i = 0;
        $loop = false;
        $loopBreak = 0;

        while ($recCount < $recNum) {
            if ($loopBreak >= $recNum) {
                break;
            }
            if ($i < count($shuffledServices)) {
                $recServiceType = $shuffledServices[$i];
            } else {
                $loop = true;
                $loopBreak += 1;
                $i = 0;
            }

            $recModel = $this->serviceTypeMap[$recServiceType];
            $recResourceType = $this->
>serviceResourceMap[$recServiceType];

            // Fetch only active services
            $recInstances = $recModel::where('status', 'active')->
>inRandomOrder()->get();

            if (!$loop) {
                // Prefer ones in the same location during the
first loop
                $recInstance = $recInstances->where('primary_town',
$location)->first();
            } else {
                // Expand search outside of location
                $recInstance = $recInstances->first();
            }

            if ($recInstance == null) {
                $i += 1;
                continue;
            } else {
                $recResource = new $recResourceType($recInstance);

                $recData = [
                    'service_type' => $recServiceType,
                    'rec' => $recResource
                ];

                $recs[] = $recData;
            }
        }
    }
}

```

```

        $recCount += 1;
    }

    $i += 1;
}
} else {
    // Handle correlations
    $shuffledCorrelations = $correlationArray;
    shuffle($shuffledCorrelations);

    $i = 0;
    $loop = false;
    $loopBreak = 0;

    while ($recCount < $recNum) {
        if ($loopBreak >= $recNum) {
            break;
        }
        if ($i < count($shuffledCorrelations)) {
            $corr = $shuffledCorrelations[$i];
        } else {
            $loop = true;
            $i = 0;
            $loopBreak += 1;
        }

        if (array_key_exists($corr, $this->ontology)) {
            // General tourism type, pick a random active
service
            $recModel = $this->serviceTypeMap[$corr];
            $recResourceType = $this->
>serviceResourceMap[$corr];

            $recInstances = $recModel::where('status',
'active')->inRandomOrder()->get();

            if (!$loop) {
                $recInstance = $recInstances->
>where('primary_town', $location)->first();
            } else {
                $recInstance = $recInstances->first();
            }

            if ($recInstance == null) {
                $i += 1;
                continue;
            } else {

```

```

        $recResource = new
$recResourceType($recInstance);

        $recData = [
            'service_type' => $corr,
            'rec' => $recResource
        ];

        $recs[] = $recData;
        $recCount += 1;
    }
} else {
    // Handle subtypes
    $corrHeight = 1;
    $corrServiceType = '';
    foreach ($this->ontology as $service =>
$subcategories) {
        if (array_key_exists($corr, $subcategories)) {
            $corrServiceType = $service;
            break;
        } else {
            foreach ($subcategories as $key => $value)
{
                if (in_array($corr, $value)) {
                    $corrHeight = 2;
                    $corrServiceType = $service;
                    break;
                }
            }
        }
    }

    if ($corrServiceType === '') {
        $i += 1;
        continue;
    }

    $recModel = $this->
>serviceTypeMap[$corrServiceType];
    $recResourceType = $this->
>serviceResourceMap[$corrServiceType];

    if ($corrHeight == 1) {
        $recInstances =
$recModel::where('tourism_sub_category', 'LIKE', '%' . $corr . '%')
            ->where('status', 'active')
            ->inRandomOrder();
    } else {

```

```

        $recInstances =
$recModel::whereRaw('LOWER(tourism_sub_category_4) LIKE ?', ['%' .
strtolower($corr) . '%'])
        ->where('status', 'active')
        ->inRandomOrder();
    }

    if (!$loop) {
        $recInstance = $recInstances-
>where('primary_town', $location)->first();
    } else {
        $recInstance = $recInstances->first();
    }

    if ($recInstance == null) {
        $i += 1;
        continue;
    } else {
        $recResource = new
$recResourceType($recInstance);

        $recData = [
            'service_type' => $corrServiceType,
            'rec' => $recResource
        ];

        $recs[] = $recData;
        $recCount += 1;
    }
}

$i += 1;
}

return $this->sendResponse($recs, 'Recommendations');
}

public function generateRandomRecommendations(Request $request)
{
    $recs = [];

    $input = $request->all();
    $requestType = $input['service_type'];
    $recNum = $input['number_of_recommendations'];

    $recCount=0;

```

```

        while ($recCount<$recNum)
        {
            $recServiceType = array_rand($this->serviceTypeMap);
            $recModel = $this->serviceTypeMap[$recServiceType];
            $recResourceType = $this->
>serviceResourceMap[$recServiceType];

            $recInstance = $recModel::inRandomOrder()->first();
            if ($recInstance)
            {
                $recCount=$recCount+1;
                $recResource =new $recResourceType($recInstance);

                $recData = [
                    'service_type' => $recServiceType,
                    'rec' => $recResource
                ];

                $recs[] = $recData;
            }
        }

        return $this->sendResponse($recs, 'Recommendations');
    }

    public function generateTransportationRecommendations(Request
$request) {
        $input = $request->all();

        $recNum = $input['number_of_recommendations'];
        $location = $input['location']; // Input: Location filter

        // Fetch all transportations randomly
        $transportations = Transportation::inRandomOrder()->get();

        $locationTransportations = [];
        $recommendedTransportations = [];

        // Filter transportations based on location
        foreach ($transportations as $transportation) {
            if (
                stripslashes($transportation->service_area, $location) !==
false
            ) {
                $locationTransportations[] = $transportation;
            }
        }
    }

```

```

    }
}

// Slice results to return only the desired number of
recommendations
$recommendedTransportations =
array_slice($locationTransportations, 0, $recNum);

return $this->
sendResponse(TransportationResource::collection($recommendedTransporta
tions), 'Transportation Recommendations');
}
}

```

BookingController Documentation

Επισκόπηση

Ο BookingController είναι υπεύθυνος για τη διαχείριση των κρατήσεων (Bookings) και των σχετικών λειτουργιών. Αυτές περιλαμβάνουν τη δημιουργία, ενημέρωση, διαγραφή και ανάκτηση πληροφοριών για κρατήσεις.

Εξαρτήσεις (Dependencies)

- **BaseController:** Βασικός controller για τυποποιημένα responses.
- **Booking Model:** Χρησιμοποιείται για τη διαχείριση των κρατήσεων στη βάση δεδομένων.
- **Validator:** Για την επικύρωση εισόδου.
- **Auth:** Για τη διαχείριση αυθεντικοποίησης.
- **DB:** Για τη διαχείριση transactions.

Μέθοδοι

1. index

- **Σκοπός:** Ανάκτηση όλων των καταχωρήσεων κρατήσεων.
- **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις κρατήσεων.
- **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.

2. store

- **Σκοπός:** Δημιουργία νέας καταχώρησης κράτησης.
- **Παράμετροι:**
 - vendor_id: Το ID του Vendor.
 - customer_id: Το ID του πελάτη.
 - package_id: Το ID του πακέτου.
 - gateway: Η πύλη πληρωμής.
 - object_id: Το ID του αντικειμένου (π.χ. ξενοδοχείο, περιήγηση).
 - object_model: Το μοντέλο του αντικειμένου (π.χ. Hotel, AdventuresTourism).
 - start_date: Η αρχική ημερομηνία.
 - end_date: Η τελική ημερομηνία.
 - total: Το συνολικό κόστος.
 - total_guests: Ο συνολικός αριθμός επισκεπτών.
 - currency: Το νόμισμα.
 - status: Η κατάσταση της κράτησης.
 - deposit: Το ποσό προκαταβολής.
 - deposit_type: Ο τύπος προκαταβολής.
 - commission: Η προμήθεια.
 - commission_type: Ο τύπος προμήθειας.
 - email: Το email του πελάτη.
 - first_name: Το όνομα του πελάτη.
 - last_name: Το επώνυμό του.
 - phone: Το τηλέφωνο του πελάτη.
 - address: Η διεύθυνση του πελάτη.
 - address2: Η δεύτερη διεύθυνση του πελάτη.
 - city: Η πόλη του πελάτη.
 - state: Η πολιτεία του πελάτη.
 - zip_code: Το ταχυδρομικό κώδικα του πελάτη.
 - country: Η χώρα του πελάτη.

- `customer_notes`: Σημειώσεις του πελάτη.
 - `vendor_service_fee_amount`: Το ποσό του τέλους υπηρεσίας του Vendor.
 - `vendor_service_fee`: Ο τύπος του τέλους υπηρεσίας του Vendor.
 - **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Δημιουργεί μια νέα καταχώρηση κράτησης στη βάση δεδομένων.
 - **Response:** Τα στοιχεία της νέας κράτησης ή σφάλμα επικύρωσης.
-

3. show

- **Σκοπός:** Ανάκτηση συγκεκριμένης καταχώρησης κράτησης.
 - **Παράμετροι:**
 - `id`: Το ID της καταχώρησης.
 - **Λειτουργία:**
 - Επιστρέφει τα στοιχεία της καταχώρησης.
 - **Response:** Στοιχεία καταχώρησης ή σφάλμα αν δεν βρεθεί.
-

4. getVendorBookings

- **Σκοπός:** Ανάκτηση κρατήσεων για έναν συγκεκριμένο Vendor.
 - **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις κρατήσεων που δημιουργήθηκαν από τον Vendor.
 - **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.
-

5. getCustomerBookings

- **Σκοπός:** Ανάκτηση κρατήσεων για έναν συγκεκριμένο πελάτη.
 - **Λειτουργία:**
 - Επιστρέφει όλες τις καταχωρήσεις κρατήσεων που δημιουργήθηκαν από τον πελάτη.
 - **Response:** Λίστα καταχωρήσεων ή σφάλμα αν δεν βρεθούν.
-

6. update

- **Σκοπός:** Ενημέρωση υπάρχουσας καταχώρησης κράτησης.

- **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - Πολλές παράμετροι, όπως vendor_id, customer_id, start_date, end_date, total, κ.λπ.
 - **Λειτουργία:**
 - Επικυρώνει τα δεδομένα εισόδου.
 - Ενημερώνει την καταχώρηση στη βάση δεδομένων.
 - **Response:** Τα ενημερωμένα στοιχεία της καταχώρησης ή σφάλμα επικύρωσης.
-

7. destroy

- **Σκοπός:** Διαγραφή καταχώρησης κράτησης.
 - **Παράμετροι:**
 - id: Το ID της καταχώρησης.
 - **Λειτουργία:**
 - Διαγράφει την καταχώρηση από τη βάση δεδομένων.
 - **Response:** Μήνυμα επιτυχίας ή σφάλμα.
-

8. checkAvailability

- **Σκοπός:** Έλεγχος διαθεσιμότητας για μια συγκεκριμένη υπηρεσία.
 - **Παράμετροι:**
 - service_type: Ο τύπος της υπηρεσίας (π.χ. wine_tourism, gastronomic_tourism).
 - service_id: Το ID της υπηρεσίας.
 - start_date: Η αρχική ημερομηνία.
 - end_date: Η τελική ημερομηνία.
 - person_types: Οι τύποι ατόμων (π.χ. ενήλικες, παιδιά).
 - **Λειτουργία:**
 - Ελέγχει αν η υπηρεσία είναι διαθέσιμη για τις καθορισμένες ημερομηνίες.
 - **Response:** Στοιχεία διαθεσιμότητας ή σφάλμα.
-

9. checkAvailabilityBeforeBooking

- **Σκοπός:** Έλεγχος διαθεσιμότητας για πολλαπλές υπηρεσίες πριν από την κράτηση.

- **Παράμετροι:**
 - items: Λίστα με τα στοιχεία των υπηρεσιών.
- **Λειτουργία:**
 - Ελέγχει αν όλες οι υπηρεσίες είναι διαθέσιμες για τις καθορισμένες ημερομηνίες.
- **Response:** Στοιχεία διαθεσιμότητας ή σφάλμα.

```
<?php

namespace Modules\Booking\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use Modules\Booking\Resources\BookingResource;
use App\Models\User;
use Modules\Booking\Models\Booking;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Validator;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\DB;

use Illuminate\Support\Str;

use Modules\AdventuresTourism\Models\AdventuresTourism;

class BookingController extends BaseController
{
    /**
     * Set the permissions
     *
     */
    function __construct(array $attributes = [])
    {
        // $this->middleware('permission:create-service', ['only' =>
        ['store']]);
        $this->middleware('permission:edit-service', ['only' =>
        ['update']]);
    }
}
```

```

        $this->middleware('permission:delete-service', ['only' =>
['destroy']]));
    }

    protected $serviceTypeMap = [
        'wine_tourism' =>
\Modes\WineTourism\Models\WineTourism::class,
        'gastronomic_tourism' =>
\Modes\GastronomicTourism\Models\GastronomicTourism::class,
        'cultural_urban_tourism' =>
\Modes\CulturalUrbanTourism\Models\CulturalUrbanTourism::class,
        'wedding_tourism' =>
\Modes\WeddingTourism\Models\WeddingTourism::class,
        'cultural_festival_tourism' =>
\Modes\CulturalFestivalTourism\Models\CulturalFestivalTourism::class,
        'dark_tourism' =>
\Modes\DarkTourism\Models\DarkTourism::class,
        'city_games_tourism' =>
\Modes\CityGamesTourism\Models\CityGamesTourism::class,
        'sacred_site_visits_tourism' =>
\Modes\SacredSiteVisitsTourism\Models\SacredSiteVisitsTourism::class,
        'agritourism' =>
\Modes\Agritourism\Models\Agritourism::class,
        'ecotourism' => \Modules\Ecotourism\Models\Ecotourism::class,
        'exploration_tourism' =>
\Modes\ExplorationTourism\Models\ExplorationTourism::class,
        'geotourism' => \Modules\Geotourism\Models\Geotourism::class,
        'fishing_tourism' =>
\Modes\FishingTourism\Models\FishingTourism::class,
        'sporting_events_tourism' =>
\Modes\SportingEventsTourism\Models\SportingEventsTourism::class,
        'adventures_tourism' =>
\Modes\AdventuresTourism\Models\AdventuresTourism::class,
        'water_activities_tourism' =>
\Modes\WaterActivitiesTourism\Models\WaterActivitiesTourism::class,
        'air_activities_tourism' =>
\Modes\AirActivitiesTourism\Models\AirActivitiesTourism::class,
        'urban_entertainment_tourism' =>
\Modes\UrbanEntertainmentTourism\Models\UrbanEntertainmentTourism::cl
ass,
        'education_tourism' =>
\Modes\EducationDevelopmentTourism\Models\EducationDevelopmentTourism
::class,
        'enrichment_tourism' =>
\Modes\EnrichmentSkillBuildingTourism\Models\EnrichmentSkillBuildingT
ourism::class,

```

```

        'cultural_urban_exp_tourism' =>
\Modules\CulturalUrbanExplorationTourism\Models\CulturalUrbanExplorationTourism::class,
    ];

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(): JsonResponse
    {
        // Get all Booking entries
        $bookingEntries = Booking::all();

        return $this->sendResponse(BookingResource::collection($bookingEntries), 'Booking
entries retrieved successfully.');
```

```

    }

    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request): JsonResponse
    {
        try {
            if (!Auth::check()) {
                return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
            }

            $user = auth()->user();
            $bookingsData = $request->input('bookings', []);

            $bookings = [];
            $errors = [];

            foreach ($bookingsData as $index => $input) {
                // Generate a unique random code for each booking
                do {
                    $code = rand(100000, 999999);
                    // Check if the code already exists in the database
                    $codeExists = Booking::where('code', $code)->exists();
                } while ($codeExists);
            }
        }
    }
}

```

```

$input['code'] = $code;

$input['create_user'] = $user->id;
$input['refund']=false;
$input['refund_multiplier'] = 0.0;
$input['source'] = 0;
$input['uuid'] = (string) Str::uuid();

// Validate each booking item
$validator = Validator::make($input, [
    'vendor_id' => 'required|integer|exists:users,id',
    'customer_id' =>
'required|integer|exists:users,id',
    'package_id' => 'required|integer',
    'gateway' => 'nullable|string|max:50',
    'object_id' => 'required|integer',
    'object_model' => 'required|string|max:255',
    'start_date' => 'required|date',
    'end_date' =>
'required|date|after_or_equal:start_date',
    'total' => 'required|numeric',
    'total_guests' => 'required|integer',
    'currency' => 'required|string|max:5',
    'status' => 'required|string|max:30',
    'deposit' => 'nullable|numeric',
    'deposit_type' => 'nullable|string|max:30',
    'commission' => 'nullable|numeric',
    'commission_type' => 'nullable|string|max:150',
    'email' => 'required|email|max:255',
    'first_name' => 'required|string|max:255',
    'last_name' => 'required|string|max:255',
    'phone' => 'required|string|max:255',
    'address' => 'required|string|max:255',
    'address2' => 'nullable|string|max:255',
    'city' => 'required|string|max:255',
    'state' => 'required|string|max:255',
    'zip_code' => 'required|string|max:255',
    'country' => 'required|string|max:255',
    'customer_notes' => 'nullable|string',
    'vendor_service_fee_amount' => 'nullable|numeric',
    'vendor_service_fee' => 'nullable|string',

]);

// If validation fails, save the error and skip to the
next booking
if ($validator->fails()) {

```

```

        $errors[] = ['index' => $index, 'errors' =>
$validator->errors()];
        continue;
    }

    // Create booking if validation passes
    $bookings[] = Booking::create($input);
}

// Check if there were any validation errors
if (!empty($errors)) {
    return $this->sendError('Validation Error.',
['validation_errors' => $errors]);
}

// Return success response with all created bookings
return $this->sendResponse(BookingResource::collection(collect($bookings)),
'Bookings created successfully.');
```

```

    } catch (\Exception $e) {
        // Return an error response
        return $this->sendError('Error occurred during creation.',
['error' => $e->getMessage()]);
    }
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show(string $id)
{
    try {
        $booking = Booking::find($id);

        if (is_null($booking)) {
            return $this->sendError('Booking not found.');
```

```

        }

        return $this->sendResponse(new BookingResource($booking),
'Booking retrieved successfully.');
```

```

    } catch (\Exception $e) {
        // Return an error response

```

```

        return $this->sendError('Error occurred during display the
specified resource.', ['error' => $e->getMessage()]);
    }
}

public function getVendorBookings():JsonResponse
{
    // Check if the user is authenticated
    if (!Auth::check()) {
        // Return an error response if no user is authenticated
        return $this->sendError('Unauthorised.',
['error'=>'Unauthorised']);
    }

    $user = auth()->user();

    $vendorBookings=Booking::where('vendor_id', $user->id)->get();
    $response = [];

    foreach ($vendorBookings as $booking) {
        $customer = User::find($booking->customer_id);
        $serviceType = str_replace('_', ' ', $booking-
>object_model);
        $serviceType = ucwords($serviceType);

        $modelName = str_replace(' ', '', $serviceType);
        $modelObject =
'Modules\\'. $modelName.'\\Models\\'. $modelName;
        $serviceObject = $modelObject::find($booking->object_id);
        $title = $serviceObject->title;

        $bookingStatus = str_replace('_', ' ', $booking->status);
        $bookingStatus = ucwords($bookingStatus);

        $item=[
            'id' => $booking->id,
            'code' => $booking->code,
            'start_date' => $booking->start_date,
            'end_date' => $booking->end_date,
            'total' => $booking->total,
            'total_guests' => $booking->total_guests,
            'customer_name' => $customer->first_name.' '.$customer-
>last_name.' '.$customer->id,
            'customer_email' => $booking->email,
            'customer_phone' => $booking->phone,
            'status' => $bookingStatus,
            'service_type' => $serviceType,
            'title' => $title,

```

```

        'created_at' => $booking->created_at
    ];

    $response[] = $item;
}

return $this->sendResponse($response, "All bookings for
vendor");
}

public function getCustomerBookings():JsonResponse
{
    // Check if the user is authenticated
    if (!Auth::check()) {
        // Return an error response if no user is authenticated
        return $this->sendError('Unauthorised.',
['error'=>'Unauthorised']);
    }

    $user = auth()->user();

    $customerBookings=Booking::where('customer_id', $user->id)-
>get();
    $response = [];

    foreach ($customerBookings as $booking) {
        $vendor = User::find($booking->vendor_id);
        $serviceType = str_replace('_', ' ', $booking-
>object_model);
        $serviceType = ucwords($serviceType);

        $modelName = str_replace(' ', '', $serviceType);
        $modelObject =
'Modules\\'.$modelName.'\\Models\\'.$modelName;
        $serviceObject = $modelObject::find($booking->object_id);
        $title = $serviceObject->title;

        $bookingStatus = str_replace('_', ' ', $booking->status);
        $bookingStatus = ucwords($bookingStatus);

        $item=[
            'id' => $booking->id,
            'code' => $booking->code,
            'start_date' => $booking->start_date,
            'end_date' => $booking->end_date,
            'total' => $booking->total,
            'total_guests' => $booking->total_guests,
            'vendor' => $title,

```

```

        'status' => $bookingStatus,
        'service_type' => $serviceType,
        'title' => $title,
        'created_at' => $booking->created_at
    ];

    $response[] = $item;
}

return $this->sendResponse($response, "All bookings for
customer");
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param Booking $booking
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, Booking $booking)
{
    try {
        if (!Auth::check()) {
            return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
        }

        $this->authorize('update', $booking);

        $user = auth()->user();

        $input = $request->all();

        // Validate the request data
        $validator = Validator::make($input, [
            'vendor_id' => 'required|integer|exists:users,id',
            'customer_id' => 'required|integer|exists:users,id',
            'payment_id' => 'nullable|integer',
            'gateway' => 'nullable|string|max:50',
            'object_id' => 'nullable|integer',
            'object_model' => 'nullable|string|max:255',
            'start_date' => 'required|date',
            'end_date' =>
'required|date|after_or_equal:start_date',
            'total' => 'required|numeric',
            'total_guests' => 'required|integer',
            'currency' => 'required|string|max:5',

```

```

        'status' => 'required|string|max:30',
        'deposit' => 'nullable|numeric',
        'deposit_type' => 'nullable|string|max:30',
        'commission' => 'nullable|numeric',
        'commission_type' => 'nullable|string|max:150',
        'email' => 'nullable|email|max:255',
        'first_name' => 'nullable|string|max:255',
        'last_name' => 'nullable|string|max:255',
        'phone' => 'nullable|string|max:255',
        'address' => 'nullable|string|max:255',
        'address2' => 'nullable|string|max:255',
        'city' => 'nullable|string|max:255',
        'state' => 'nullable|string|max:255',
        'zip_code' => 'nullable|string|max:255',
        'country' => 'nullable|string|max:255',
        'customer_notes' => 'nullable|string',
        'vendor_service_fee_amount' => 'nullable|numeric',
        'vendor_service_fee' => 'nullable|string',
    ]);

    // If validation fails, return an error response
    if ($validator->fails()) {
        return $this->sendError('Validation Error.',
$validator->errors());
    }

    $input['update_user'] = $user->id;

    $booking->update($input);

    return $this->sendResponse(new BookingResource($booking),
'Booking updated successfully.');
```

```

    } catch (\Exception $e) {
        // Return an error response
        return $this->sendError('Error occurred during update.',
['error' => $e->getMessage()]);
    }
}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy(Booking $booking): JsonResponse
{

```

```

        try {
            if (!Auth::check()) {
                return $this->sendError('Unauthorised.',
['error'=>'Unauthored']);
            }

            $this->authorize('delete', $booking);

            $booking->delete();

            return $this->sendResponse([], 'Wine Tourism deleted
successfully.');
```

```

        } catch (\Exception $e) {
            // Return an error response
            return $this->sendError('Error occurred during delete.',
['error' => $e->getMessage()]);
        }
    }

    public function checkAvailability(Request $request){

        if($request->input('firstLoad') == "false") {
            $rules = [
                'specific_datetime_flag' => 'required|boolean',
                'service_type' => 'required|string',
                'service_id' => 'required',
                'start_date' => 'required:date_format:Y-m-d',
                'end_date' => 'required:date_format:Y-m-d',
                'person_types' => 'required|array',
            ];
            $validator = \Validator::make($request->all(), $rules);
            if ($validator->fails()) {
                return $this->sendError($validator->errors()->all());
            }

            if(strtotime($request('end_date')) -
strtotime($request('start_date')) < 0 ){
                return $this->sendError(__('Dates are not valid'));
            }
            if(strtotime($request('end_date')) -
strtotime($request('start_date')) > 30*86400){
                return $this->sendError(__('Maximum day for booking is
30'));
            }
        }
    }

```

```

        $serviceType = $request->input('service_type');
        $serviceId = $request->input('service_id');
        $specificDatatimeFlag = $request-
>input('specific_datetime_flag');

        $modelClass = $this->serviceTypeMap[$serviceType];
        $service = $modelClass::find($serviceId);

        // $wine_tourism = $this-
>wineTourismClass::find($wine_tourism_id);
        if(empty($serviceId) or empty($service)){
            return $this->sendError(__("Service not found"));
        }

        if(\request()->input('firstLoad') == "false") {
            $numberDays = abs(strtotime(\request('end_date')) -
strtotime(\request('start_date'))) / 86400;
            if(!empty($service->min_day_stays) and $numberDays <
$service->min_day_stays){
                return $this->sendError(__("You must to book a minimum
of :number days",['number'=>$service->min_day_stays]));
            }

            if(!empty($service->min_day_before_booking)){
                $minday_before = strtotime("today +". $service-
>min_day_before_booking." days");
                if( strtotime(\request('start_date')) <
$minday_before){
                    return $this->sendError(__("You must book the
service for :number days in advance",["number"=>$wine_tourism-
>min_day_before_booking]));
                }
            }
        }

        if ($specificDatatimeFlag) {

            $isAvailable = $service->getAvailability(request()-
>input());

            if ($isAvailable) {
                return $this->sendResponse([
                    'is_available'=>$isAvailable
                ], 'Check service availability response successfully');
            } else {
                return $this->sendResponse([
                    'is_available'=>$isAvailable
                ], 'Check service availability response successfully');
            }
        }
    }
}

```

```

    }

    } else {

        $service_data_availability = $service->getAvailability(request()->input());

        return $this->sendResponse([
            'service_data'=>$service_data_availability
        ], 'Check service availability response successfully!');

    }

}

public function checkAvailabilityBeforeBooking(Request $request) {
    // Validate the request input
    $validator = Validator::make($request->all(), [
        'items' => 'required|array',
    ]);

    if ($validator->fails()) {
        return $this->sendError($validator->errors()->all());
    }

    $items = $request->input('items');

    $availabilityResults = [];

    foreach ($items as $item) {
        if($item['firstLoad'] == "false") {
            // Validate the input for each item
            $validator = Validator::make($item, [
                'specific_datetime_flag' => 'required|boolean',
                'service_type' => 'required|string',
                'service_id' => 'required',
                'start_date' => 'required|date_format:Y-m-d',
                'end_date' => 'required|date_format:Y-m-d',
                'person_types' => 'required|array',
            ]);

            if ($validator->fails()) {
                return $this->sendError($validator->errors()->all());
            }

```

```

        if(strtotime($item['end_date']) -
strtotime($item['start_date']) < 0 ){
            return $this->sendError(__("Dates are not valid"));
        }
        if(strtotime($item['end_date']) -
strtotime($item['start_date']) > 30*86400){
            return $this->sendError(__("Maximum day for booking
is 30"));
        }
    }

    $serviceType = $item['service_type'];
    $serviceId = $item['service_id'];
    $specificDatetimeFlag = $item['specific_datetime_flag'];

    $modelClass = $this->serviceTypeMap[$serviceType];
    $service = $modelClass::find($serviceId);

    // $wine_tourism = $this-
>wineTourismClass::find($wine_tourism_id);
    if(empty($serviceId) or empty($service)){
        return $this->sendError(__("Service not found"));
    }

    if($item['firstLoad'] == "false") {
        $numberDays = abs(strtotime($item['end_date']) -
strtotime($item['start_date'])) / 86400;
        if(!empty($service->min_day_stays) and $numberDays <
$service->min_day_stays){
            return $this->sendError(__("You must to book a
minimum of :number days",['number'=>$service->min_day_stays]));
        }

        if(!empty($service->min_day_before_booking)){
            $minday_before = strtotime("today +". $service-
>min_day_before_booking." days");
            if( strtotime($item['start_date']) <
$minday_before){
                return $this->sendError(__("You must book the
service for :number days in advance",["number"=>$wine_tourism-
>min_day_before_booking]));
            }
        }
    }

    if ($specificDatetimeFlag) {

        $isAvailable = $service->getAvailability($item);
    }

```

```

        if ($isAvailable) {

            $availabilityResults[$serviceType.'_'.$serviceId.'_' .
$item['start_date']] = ['is_available' => $isAvailable];

            // return $this->sendResponse([
            //     'is_available'=>$isAvailable
            // ], 'Check service availability response
successfully');

        } else {

            return $this->sendResponse(["availability"=>false,
"service_type"=>$serviceType, "service_id"=>$serviceId , "start_date"=>
$item['start_date']], 'Service is not available anymore. ');
            //
            $availabilityResults[$serviceType][$serviceId][$item['start_date']] =
['is_available' => $isAvailable];

            // return $this->sendResponse([
            //     'is_available'=>$isAvailable
            // ], 'Check service availability response
successfully');
        }

    } else {

        $service_data_availability = $service-
>getAvailability($item);

        return $this->sendResponse([
            'service_data'=>$service_data_availability
        ], 'Check service availability response
successfully!');

    }

}

return $this->sendResponse(['availability_results' =>
$availabilityResults], 'isAvailable');
}

}

```

API endpoints

Χρήστες και Διαχείριση Λογαριασμών:

- **register**: Εγγραφή χρήστη και προμηθευτή.
- **login**: Σύνδεση χρήστη.
- **logout**: Αποσύνδεση χρήστη (χρησιμοποιώντας middleware auth:sanctum).
- **getUserInfo**: Επιστρέφει πληροφορίες χρήστη.
- **updateUserInfo**: Ενημέρωση πληροφοριών χρήστη.
- **deleteUser**: Διαγραφή χρήστη.
- **changePassword**: Αλλαγή κωδικού χρήστη.
- **forgot-password/reset-forgotten-password**: Ανάκτηση ή επαναφορά κωδικού πρόσβασης.

Προμηθευτές:

- **registerVendor**: Εγγραφή προμηθευτή.
- **get-vendor-details**: Λήψη στοιχείων προμηθευτή.
- **get-vendor-services**: Λήψη υπηρεσιών προμηθευτή.

Πληρωμές και Webhooks:

- **make-viva-payment**: Δημιουργία παραγγελίας πληρωμής μέσω Viva.
- **viva-check-expired**: Καθαρισμός ληγμένων παραγγελιών για δοκιμές.
- **viva-refund**: Επιστροφή χρημάτων στον πελάτη.
- **viva-vendor-cancel**: Ακύρωση παραγγελίας από τον προμηθευτή.
- **viva-signup-webhook**: Webhook για επιτυχημένη εγγραφή προμηθευτή μέσω Viva.

Συστάσεις και Αναζητήσεις:

- **recommendations**: Δημιουργία συστάσεων.
- **hotel-recommendations**: Συστάσεις για ξενοδοχεία.

- **search:** Αναζητήσεις υπηρεσιών.

Υπηρεσίες Τουρισμού:

- **adventures-tourism, agritourism, air-activities-tourism κ.ά.:** APIs για διάφορες κατηγορίες τουριστικών υπηρεσιών (π.χ. αγροτουρισμός, περιπέτειες, αεροπορικές δραστηριότητες).
- **{service}-vendor/{id}:** Λήψη υπηρεσιών τουρισμού από τον προμηθευτή.

Κρατήσεις και Διαθεσιμότητα:

- **check-availability:** Έλεγχος διαθεσιμότητας υπηρεσιών (π.χ. ξενοδοχεία, μεταφορές).
- **hotel-room:** Διαχείριση δωματίων ξενοδοχείου (π.χ. προσθήκη, ενημέρωση).

Ανασκοπήσεις και Σχόλια:

- **review-vendor, review-customer:** Επιστροφή ανασκοπήσεων από πελάτες και προμηθευτές.
- **review-service:** Επιστροφή ανασκοπήσεων για υπηρεσίες.

Τουριστικές Υπηρεσίες:

1. **Ημερομηνίες adventures-tourism** - Αυτές οι διαδρομές επιτρέπουν στους χρήστες να διαχειρίζονται και να ανακτούν ημερομηνίες adventures-tourism.
 - POST /adventures-tourism-dates/dates → Λήψη πολλαπλών ημερομηνιών adventures-tourism.
 - POST /adventures-tourism-dates → Διαχείριση ημερομηνιών adventures-tourism.
 - POST /adventures-tourism-dates/{adventures_tourism_id} → Λήψη συγκεκριμένων ημερομηνιών adventures-tourism.

Άλλες Κατηγορίες Τουρισμού - Περιλαμβάνει κατηγορίες όπως τουρισμό εκπαίδευσης, ψυχαγωγίας, αλιείας, γαστρονομίας, ιατρικό τουρισμό, κ.ά.

- Κάθε κατηγορία έχει διαδρομές για ανάκτηση, διαχείριση και αντιγραφή συγκεκριμένων ημερομηνιών τουρισμού.

Διαδρομές Κρατήσεων και Διαχείρισης Δωματίων:

- **Διαχείριση Κρατήσεων και Δωματίων** - Διαδρομές για διαχείριση κρατήσεων δωματίων ξενοδοχείων, συμπεριλαμβανομένων των διαδικασιών check-out και προβολής κρατήσεων από προμηθευτές ή πελάτες.
 - POST /hotel-room-booking-checkout → Ολοκλήρωση κράτησης δωματίου.
 - POST /hotel-room-dates → Διαχείριση ημερομηνιών δωματίου.

- POST /hotel-room-dates/{room_id} → Λήψη ημερομηνιών δωματίου.
- POST /booking → Δημιουργία νέας κράτησης.
- GET /vendor-bookings → Λήψη κρατήσεων προμηθευτή.
- GET /customer-bookings → Λήψη κρατήσεων πελάτη.
- **Διαχείριση Πακέτων Ταξιδιών** - Διαδρομές για τη διαχείριση πακέτων ταξιδιών.
 - POST /package → Δημιουργία πακέτου.
- **Διαχείριση Ξενοδοχείου** - Διαδρομές για τη διαχείριση πληροφοριών για ξενοδοχεία.
 - POST /hotel → Προσθήκη νέου ξενοδοχείου.
 - PUT /hotel/{hotel} → Ενημέρωση πληροφοριών ξενοδοχείου.
 - DELETE /hotel/{hotel} → Διαγραφή ξενοδοχείου.

Μεταφράσεις:

- **Αιτήματα Μετάφρασης** - Διαδρομές που σχετίζονται με αιτήματα μετάφρασης για γενικό περιεχόμενο και έγγραφα.
 - POST /request-translation → Αποστολή αιτήματος μετάφρασης.
 - POST /request-document-translation → Αίτημα μετάφρασης εγγράφου.
 - POST /incoming-translation → Διαχείριση εισερχόμενων μεταφράσεων.
 - POST /incoming-document-translation → Διαχείριση εισερχόμενων μεταφράσεων εγγράφου.
 - GET /check-translation-status → Έλεγχος κατάστασης αιτήματος μετάφρασης.
 - POST /get-available-locales → Λήψη διαθέσιμων τοπικών γλωσσών για μετάφραση.
- **Μεταφράσεις για Τουρισμό Κρασιού** - Ειδικές διαδρομές για μεταφράσεις που σχετίζονται με τουρισμό κρασιού.
 - POST /wine-tourism-translation → Αποθήκευση μετάφρασης τουρισμού κρασιού.
 - PUT /wine-tourism-translation → Ενημέρωση μετάφρασης τουρισμού κρασιού.

Διαδρομές Αντιγραφής:

- **Αντιγραφή Υπηρεσιών Τουρισμού** - Επιτρέπει την αντιγραφή διάφορων τύπων τουριστικών υπηρεσιών όπως περιπέτειες, αγροτουρισμό, αεροπορικές δραστηριότητες, κ.ά.
 - POST /adventures-tourism/{id}/duplicate → Αντιγραφή τουρισμού περιπέτειας.

- POST /agritourism/{id}/duplicate → Αντιγραφή αγροτουρισμού.
- Και άλλες διαδρομές για άλλες κατηγορίες τουρισμού.

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

use App\Http\Controllers\API\UserController;
use App\Http\Controllers\API\VendorController;
use App\Http\Controllers\API\HotelController;
use App\Http\Controllers\API\HotelRoomController;
use App\Http\Controllers\API\HotelRoomBookingController;
use App\Http\Controllers\API\HotelRoomDatesController;

use App\Http\Controllers\API\VivaOrderCreationController;
use App\Http\Controllers\API\VivaRefundController;
use App\Http\Controllers\API\VivaExpiredOrderCleanupController;
use App\Http\Controllers\API\VivaRecurringPaymentController;

use App\Http\Controllers\API\VivaSignupController;

use App\Http\Controllers\API\TestEmailController;
use App\Http\Controllers\API\ICalendarController;

use
App\Http\Controllers\API\VivaWebhookTransactionPaymentCreatedController
;
//use
App\Http\Controllers\API\VivaWebhookAccountTransactionCreatedController
;
//use
App\Http\Controllers\API\VivaWebhookTransactionPriceCalculatedController;
use App\Http\Controllers\API\VivaSignupWebhookController;
use App\Http\Controllers\API\VivaAccountVerificationWebhookController;
use App\Http\Controllers\API\VivaTransfersController;

// -----12/01/2024-----
-----

use Modules\WineTourism\Controllers\WineTourismDatesController;
use Modules\WineTourism\Controllers\WineTourismController;
```

```

use
Modules\GastronomicTourism\Controllers\GastronomicTourismDatesController;
use
Modules\GastronomicTourism\Controllers\GastronomicTourismController;

use
Modules\CulturalUrbanTourism\Controllers\CulturalUrbanTourismDatesController;
use
Modules\CulturalUrbanTourism\Controllers\CulturalUrbanTourismController;

use Modules\WeddingTourism\Controllers\WeddingTourismDatesController;
use Modules\WeddingTourism\Controllers\WeddingTourismController;

use
Modules\CulturalFestivalTourism\Controllers\CulturalFestivalTourismDatesController;
use
Modules\CulturalFestivalTourism\Controllers\CulturalFestivalTourismController;

use Modules\DarkTourism\Controllers\DarkTourismDatesController;
use Modules\DarkTourism\Controllers\DarkTourismController;

use
Modules\CityGamesTourism\Controllers\CityGamesTourismDatesController;
use Modules\CityGamesTourism\Controllers\CityGamesTourismController;

use
Modules\SacredSiteVisitsTourism\Controllers\SacredSiteVisitsTourismDatesController;
use
Modules\SacredSiteVisitsTourism\Controllers\SacredSiteVisitsTourismController;

use Modules\Agritourism\Controllers\AgritourismDatesController;
use Modules\Agritourism\Controllers\AgritourismController;

use Modules\Ecotourism\Controllers\EcotourismDatesController;
use Modules\Ecotourism\Controllers\EcotourismController;

use
Modules\ExplorationTourism\Controllers\ExplorationTourismDatesController;
use
Modules\ExplorationTourism\Controllers\ExplorationTourismController;

```

```

use Modules\Geotourism\Controllers\GeotourismDatesController;
use Modules\Geotourism\Controllers\GeotourismController;

use Modules\FishingTourism\Controllers\FishingTourismDatesController;
use Modules\FishingTourism\Controllers\FishingTourismController;

use
Modules\SportingEventsTourism\Controllers\SportingEventsTourismDatesCon
troller;
use
Modules\SportingEventsTourism\Controllers\SportingEventsTourismControll
er;

use
Modules\AdventuresTourism\Controllers\AdventuresTourismDatesController;
use Modules\AdventuresTourism\Controllers\AdventuresTourismController;

use
Modules\WaterActivitiesTourism\Controllers\WaterActivitiesTourismDatesC
ontroller;
use
Modules\WaterActivitiesTourism\Controllers\WaterActivitiesTourismContro
ller;

use
Modules\AirActivitiesTourism\Controllers\AirActivitiesTourismDatesContr
oller;
use
Modules\AirActivitiesTourism\Controllers\AirActivitiesTourismController
;

use
Modules\UrbanEntertainmentTourism\Controllers\UrbanEntertainmentTourism
DatesController;
use
Modules\UrbanEntertainmentTourism\Controllers\UrbanEntertainmentTourism
Controller;

use
Modules\EducationDevelopmentTourism\Controllers\EducationDevelopmentTou
rismDatesController;
use
Modules\EducationDevelopmentTourism\Controllers\EducationDevelopmentTou
rismController;

```

```

use
Modules\EnrichmentSkillBuildingTourism\Controllers\EnrichmentSkillBuild
ingTourismDatesController;
use
Modules\EnrichmentSkillBuildingTourism\Controllers\EnrichmentSkillBuild
ingTourismController;

use
Modules\CulturalUrbanExplorationTourism\Controllers\CulturalUrbanExplor
ationTourismDatesController;
use
Modules\CulturalUrbanExplorationTourism\Controllers\CulturalUrbanExplor
ationTourismController;

use Modules\MedicalTourism\Controllers\MedicalTourismDatesController;
use Modules\MedicalTourism\Controllers\MedicalTourismController;

use Modules\SpaTourism\Controllers\SpaTourismDatesController;
use Modules\SpaTourism\Controllers\SpaTourismController;

use Modules\WellnessTourism\Controllers\WellnessTourismDatesController;
use Modules\WellnessTourism\Controllers\WellnessTourismController;

use Modules\Transportation\Controllers\TransportationController;

use Modules\MediaFile\Controllers\MediaFileController;

use Modules\Booking\Controllers\BookingController;

use Modules\Package\Controllers\PackageController;

use Modules\Food\Controllers\FoodController;

use Modules\Review\Controllers\ReviewController;
use App\Http\Controllers\API\SentimentController;

use App\Http\Controllers\API\RecommendationsController;
use App\Http\Controllers\API\SearchController;

use App\Http\Controllers\API\TranslationController;

use Modules\WineTourism\Controllers\WineTourismTranslationController;

/*
|-----
|
| API Routes

```

```

| -----
|
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "api" middleware group. Make something great!
|
| */
Route::controller(UserController::class)->group(function(){
    Route::post('register', 'register');
    Route::post('login', 'login');
});

Route::controller(VendorController::class)->group(function(){
    Route::post('registerVendor', 'register');
});

// Route::middleware('auth:sanctum')->get('/user', function (Request
$request) {
//     return $request->user();
// });

//Viva routes: new ones
Route::post('make-viva-payment', [VivaOrderCreationController::class,
'createOrder']);

Route::post('viva-check-expired',
[VivaExpiredOrderCleanupController::class, 'cleanupExpiredOrders']);
//Forced cleanup for tests
Route::post('viva-recurring-payments',
[VivaRecurringPaymentController::class, 'checkAllBookings']); //Forced
check for tests
Route::post('viva-transfers', [VivaTransfersController::class,
'handleTransfers']); //Forced check for transfers

Route::post('viva-refund', [VivaRefundController::class,
'refundCustomer']);
Route::post('viva-vendor-cancel', [VivaRefundController::class,
'cancelByVendor']);
Route::post('get-refund-eligibility', [VivaRefundController::class,
'checkRefundEligibility']);

Route::get('viva-transaction-payment-webhook',
[VivaWebhookTransactionPaymentCreatedController::class,
'verifyWebhook']);

```

```

Route::post('viva-transaction-payment-webhook',
[VivaWebhookTransactionPaymentCreatedController::class,
'handleWebhookResponse']);

//Route::get('viva-price-calculated-webhook',
[VivaWebhookTransactionPriceCalculatedController::class,
'verifyWebhook']);
//Route::post('viva-price-calculated-webhook',
[VivaWebhookTransactionPriceCalculatedController::class,
'handleWebhookResponse']);

//Route::get('viva-account-transaction-webhook',
[VivaWebhookAccountTransactionCreatedController::class,
'verifyWebhook']);
//Route::post('viva-account-transaction-webhook',
[VivaWebhookAccountTransactionCreatedController::class,
'handleWebhookResponse']);

Route::get('viva-signup-webhook', [VivaSignupWebhookController::class,
'verifyWebhook']);
Route::post('viva-signup-webhook', [VivaSignupWebhookController::class,
'handleSuccessfulSignupWebhook']);

Route::get('viva-account-verification-webhook',
[VivaAccountVerificationWebhookController::class, 'verifyWebhook']);
Route::post('viva-account-verification-webhook',
[VivaAccountVerificationWebhookController::class,
'handleVerificationWebhook']);

Route::post('/send-test-email', [TestEmailController::class,
'sendEmail']);
Route::post('/set-viva-wallet', [VivaSignupController::class,
'setVivaWallet']); //TEST REASONS ONLY

Route::get('generate-ical/{vendor_id}/{object_model}/{service_id}',
[ICalendarController::class, 'generateICalFileForService']);

Route::post('recommendations', [RecommendationsController::class,
'generateRecommendations']);
Route::post('hotel-recommendations', [RecommendationsController::class,
'generateHotelRecommendations']);
Route::post('random-recommendation', [RecommendationsController::class,
'generateRandomRecommendations']);
Route::post('transportation-recommendations',
[RecommendationsController::class,
'generateTransportationRecommendations']);

Route::post('search', [SearchController::class, 'searchResults']);

```

```

Route::middleware(['auth:sanctum'])->group(function () {
    Route::post('logout', [UserController::class, 'logout']);
    Route::get('getUserInfo', [UserController::class, 'getUserInfo']);
    Route::get('/admin/users', [UserController::class, 'getAllUsers']);
    Route::put('updateUserInfo', [UserController::class,
'updateUserInfo']);
    Route::delete('deleteUser', [UserController::class, 'deleteUser']);
    Route::post('changePassword', [UserController::class,
'changePassword']);

    Route::get('viva-signup-link', [VivaSignupController::class,
'generateVivaSignupLink']);
});

Route::post('forgot-password', [UserController::class,
'sendResetLinkEmail']);
Route::post('reset-forgotten-password', [UserController::class,
'resetForgottenPassword']);

Route::get('get-vendor-details/{id}', [UserController::class,
'getVendorDetails']);
Route::get('get-vendor-services/{id}', [UserController::class,
'getVendorServicesForProfile']);
Route::get('img/{id}', [MediaFileController::class, 'getImageUrl']);

//-----Services-----
Route::apiResource('adventures-tourism',
AdventuresTourismController::class);
Route::get('adventures-tourism/vendor/{id}',
[AdventuresTourismController::class, 'getVendorAdventuresTourism']);

Route::apiResource('agritourism', AgritourismController::class);
Route::get('agritourism/vendor/{id}', [AgritourismController::class,
'getVendorAgritourism']);

Route::apiResource('air-activities-tourism',
AirActivitiesTourismController::class);
Route::get('air-activities-tourism/vendor/{id}',
[AirActivitiesTourismController::class,
'getVendorAirActivitiesTourism']);

Route::apiResource('city-games-tourism',
CityGamesTourismController::class);

```

```

Route::get('city-games-tourism/vendor/{id}',
[CityGamesTourismController::class, 'getVendorCityGamesTourism']);

Route::apiResource('cultural-festival-tourism',
CulturalFestivalTourismController::class);
Route::get('cultural-festival-tourism/vendor/{id}',
[CulturalFestivalTourismController::class,
'getVendorCulturalFestivalTourism']);

Route::apiResource('cultural-urban-exp-tourism',
CulturalUrbanExplorationTourismController::class);
Route::get('cultural-urban-exp-tourism/vendor/{id}',
[CulturalUrbanExplorationTourismController::class,
'getVendorCulturalUrbanExplorationTourism']);

Route::apiResource('cultural-urban-tourism',
CulturalUrbanTourismController::class);
Route::get('cultural-urban-tourism/vendor/{id}',
[CulturalUrbanTourismController::class,
'getVendorCulturalUrbanTourism']);

Route::apiResource('dark-tourism', DarkTourismController::class);
Route::get('dark-tourism/vendor/{id}', [DarkTourismController::class,
'getVendorDarkTourism']);

Route::apiResource('ecotourism', EcotourismController::class);
Route::get('ecotourism/vendor/{id}', [EcotourismController::class,
'getVendorEcotourism']);

Route::apiResource('education-tourism',
EducationDevelopmentTourismController::class);
Route::get('education-tourism/vendor/{id}',
[EducationDevelopmentTourismController::class,
'getVendorEducationDevelopmentTourism']);

Route::apiResource('enrichment-tourism',
EnrichmentSkillBuildingTourismController::class);
Route::get('enrichment-tourism/vendor/{id}',
[EnrichmentSkillBuildingTourismController::class,
'getVendorEnrichmentSkillBuildingTourism']);

Route::apiResource('exploration-tourism',
ExplorationTourismController::class);
Route::get('exploration-tourism/vendor/{id}',
[ExplorationTourismController::class, 'getVendorExplorationTourism']);

Route::apiResource('fishing-tourism', FishingTourismController::class);

```

```

Route::get('fishing-tourism/vendor/{id}',
[FishingTourismController::class, 'getVendorFishingTourism']);

Route::apiResource('gastronomic-tourism',
GastronomicTourismController::class);
Route::get('gastronomic-tourism/vendor/{id}',
[GastronomicTourismController::class, 'getVendorGastronomicTourism']);

Route::apiResource('geotourism', GeotourismController::class);
Route::get('geotourism/vendor/{id}', [GeotourismController::class,
'getVendorGeotourism' ]);

Route::apiResource('sacred-site-visits-tourism',
SacredSiteVisitsTourismController::class);
Route::get('sacred-site-visits-tourism/vendor/{id}',
[SacredSiteVisitsTourismController::class,
'getVendorSacredSiteVisitsTourism' ]);

Route::apiResource('sporting-events-tourism',
SportingEventsTourismController::class);
Route::get('sporting-events-tourism/vendor/{id}',
[SportingEventsTourismController::class,
'getVendorSportingEventsTourism' ]);

Route::apiResource('urban-entertainment-tourism',
UrbanEntertainmentTourismController::class);
Route::get('urban-entertainment-tourism/vendor/{id}',
[UrbanEntertainmentTourismController::class,
'getVendorUrbanEntertainmentTourism' ]);

Route::apiResource('water-activities-tourism',
WaterActivitiesTourismController::class);
Route::get('water-activities-tourism/vendor/{id}',
[WaterActivitiesTourismController::class,
'getVendorWaterActivitiesTourism' ]);

Route::apiResource('wedding-tourism', WeddingTourismController::class);
Route::get('wedding-tourism/vendor/{id}',
[WeddingTourismController::class, 'getVendorWeddingTourism']);

Route::apiResource('wine-tourism', WineTourismController::class);
Route::get('wine-tourism/vendor/{id}', [WineTourismController::class,
'getVendorWineTourism' ]);
// Route::post('wine-tourism/check-availability',
[WineTourismController::class, 'checkAvailability']);

Route::apiResource('medical-tourism', MedicalTourismController::class);

```

```

Route::get('medical-tourism/vendor/{id}',
[MedicalTourismController::class, 'getVendorMedicalTourism']);

Route::apiResource('spa-tourism', SpaTourismController::class);
Route::get('spa-tourism/vendor/{id}', [SpaTourismController::class,
'getVendorSpaTourism']);

Route::apiResource('wellness-tourism',
WellnessTourismController::class);
Route::get('wellness-tourism/vendor/{id}',
[WellnessTourismController::class, 'getVendorWellnessTourism']);

Route::post('service/time-slots/check-availability',
[BookingController::class, 'checkAvailability']);
Route::post('service/time-slots/check-availability-before-booking',
[BookingController::class, 'checkAvailabilityBeforeBooking']);

Route::apiResource('food', FoodController::class);
Route::get('food/vendor/{id}', [FoodController::class,
'getVendorFood']);

Route::apiResource('transportation', TransportationController::class);
Route::get('transportation/vendor/{id}',
[TransportationController::class, 'getVendorTransportation']);

Route::get('/hotel', [HotelController::class, 'index']);
Route::get('/hotel/{hotel}', [HotelController::class, 'show']);

Route::get('hotel/vendor/{id}', [HotelController::class,
'getVendorHotels']);
Route::post('hotel/check-rooms-availability', [HotelController::class,
'checkAvailability']);

Route::apiResource('hotel-room', HotelRoomController::class)
->only(['store', 'update', 'destroy'])
->middleware('auth:sanctum');

Route::apiResource('hotel-room', HotelRoomController::class)
->except(['store', 'update', 'destroy']);

Route::get('hotel/{id}/rooms', [HotelRoomController::class,
'getHotelRooms']);

Route::get('review-vendor', [ReviewController::class,
'getVendorReview']);

```

```

Route::get('review-customer', [ReviewController::class,
'getCustomerReview']);
Route::get('review-service/{object_id}/{object_model}',
[ReviewController::class, 'getServiceReview']);

// Service date availability to be used in search
Route::post('adventures-tourism-dates/dates',
[AdventuresTourismDatesController::class,
'getMultipleAdventuresTourismDates']);
Route::post('agritourism-dates/dates',
[AgritourismDatesController::class, 'getMultipleAgritourismDates']);
Route::post('air-activities-tourism-dates/dates',
[AirActivitiesTourismDatesController::class,
'getMultipleAirActivitiesTourismDates']);
Route::post('city-games-tourism-dates/dates',
[CityGamesTourismDatesController::class,
'getMultipleCityGamesTourismDates']);
Route::post('cultural-festival-tourism-dates/dates',
[CulturalFestivalTourismDatesController::class,
'getMultipleCulturalFestivalTourismDates']);
Route::post('cultural-urban-exp-tourism-dates/dates',
[CulturalUrbanExplorationTourismDatesController::class,
'getMultipleCulturalUrbanExplorationTourismDates']);
Route::post('cultural-urban-tourism-dates/dates',
[CulturalUrbanTourismDatesController::class,
'getMultipleCulturalUrbanTourismDates']);
Route::post('dark-tourism-dates/dates',
[DarkTourismDatesController::class, 'getMultipleDarkTourismDates']);
Route::post('ecotourism-dates/dates',
[EcotourismDatesController::class, 'getMultipleEcotourismDates']);
Route::post('education-tourism-dates/dates',
[EducationDevelopmentTourismDatesController::class,
'getMultipleEducationDevelopmentTourismDates']);
Route::post('enrichment-tourism-dates/dates',
[EnrichmentSkillBuildingTourismDatesController::class,
'getMultipleEnrichmentSkillBuildingTourismDates']);
Route::post('exploration-tourism-dates/dates',
[ExplorationTourismDatesController::class,
'getMultipleExplorationTourismDates']);
Route::post('fishing-tourism-dates/dates',
[FishingTourismDatesController::class,
'getMultipleFishingTourismDates']);
Route::post('gastronomic-tourism-dates/dates',
[GastronomicTourismDatesController::class,
'getMultipleGastronomicTourismDates']);
Route::post('geotourism-dates/dates',
[GeotourismDatesController::class, 'getMultipleGeotourismDates']);

```

```

Route::post('sacred-site-visits-tourism-dates/dates',
[SacredSiteVisitsTourismDatesController::class,
'getMultipleSacredSiteVisitsTourismDates']);
Route::post('sporting-events-tourism-dates/dates',
[SportingEventsTourismDatesController::class,
'getMultipleSportingEventsTourismDates']);
Route::post('urban-entertainment-tourism-dates/dates',
[UrbanEntertainmentTourismDatesController::class,
'getMultipleUrbanEntertainmentTourismDates']);
Route::post('water-activities-tourism-dates/dates',
[WaterActivitiesTourismDatesController::class,
'getMultipleWaterActivitiesTourismDates']);
Route::post('wedding-tourism-dates/dates',
[WeddingTourismDatesController::class,
'getMultipleWeddingTourismDates']);
Route::post('wine-tourism-dates/dates',
[WineTourismDatesController::class, 'getMultipleWineTourismDates']);

Route::post('medical-tourism-dates/dates',
[MedicalTourismDatesController::class,
'getMultipleMedicalTourismDates']);
Route::post('spa-tourism-dates/dates',
[SpaTourismDatesController::class, 'getMultipleSpaTourismDates']);
Route::post('wellness-tourism-dates/dates',
[WellnessTourismDatesController::class,
'getMultipleWellnessTourismDates']);

//-----End of Services-----
---

Route::middleware('auth:sanctum')->group(function () {
    Route::apiResource('review', ReviewController::class);
    Route::get('vendor/{vendorId}/sentiment',
[SentimentController::class, 'calculateSentimentForVendor']);

    Route::get('vendor-services', [UserController::class,
'getVendorServices']);

    Route::apiResource('hotel-room-booking',
HotelRoomBookingController::class);
    Route::post('hotel-room-booking-checkout',
[HotelRoomBookingController::class, 'completeHotelRoomBooking']);

    Route::apiResource('booking', BookingController::class);

```

```

    Route::get('vendor-bookings', [BookingController::class,
'getVendorBookings']);
    Route::get('customer-bookings', [BookingController::class,
'getCustomerBookings']);

    Route::apiResource('package', PackageController::class);

    Route::post('/hotel', [HotelController::class, 'store']);
    Route::put('/hotel/{hotel}', [HotelController::class, 'update']);
    Route::delete('/hotel/{hotel}', [HotelController::class,
'destroy']);

    // Route::apiResource('hotel-room-dates',
HotelRoomDatesController::class);
    //-----Dates-----
    Route::post('hotel-room-dates', [HotelRoomDatesController::class,
'manageRoomDates']);
    Route::post('hotel-room-dates/{room_id}',
[HotelRoomDatesController::class, 'getHotelRoomDates']);

    Route::post('adventures-tourism-dates',
[AdventuresTourismDatesController::class,
'manageAdventuresTourismDates']);
    Route::post('adventures-tourism-dates/{adventures_tourism_id}',
[AdventuresTourismDatesController::class,
'getAdventuresTourismDates']);

    Route::post('agritourism-dates',
[AgritourismDatesController::class, 'manageAgritourismDates']);
    Route::post('agritourism-dates/{agritourism_id}',
[AgritourismDatesController::class, 'getAgritourismDates']);

    Route::post('air-activities-tourism-dates',
[AirActivitiesTourismDatesController::class,
'manageAirActivitiesTourismDates']);
    Route::post('air-activities-tourism-
dates/{air_activities_tourism_id}',
[AirActivitiesTourismDatesController::class,
'getAirActivitiesTourismDates']);

    Route::post('city-games-tourism-dates',
[CityGamesTourismDatesController::class,
'manageCityGamesTourismDates']);
    Route::post('city-games-tourism-dates/{city_games_tourism_id}',
[CityGamesTourismDatesController::class, 'getCityGamesTourismDates']);

```

```

Route::post('cultural-festival-tourism-dates',
[CulturalFestivalTourismDatesController::class,
'manageCulturalFestivalTourismDates']);
Route::post('cultural-festival-tourism-
dates/{cultural_festival_tourism_id}',
[CulturalFestivalTourismDatesController::class,
'getCulturalFestivalTourismDates']);

Route::post('cultural-urban-exp-tourism-dates',
[CulturalUrbanExplorationTourismDatesController::class,
'manageCulturalUrbanExplorationTourismDates']);
Route::post('cultural-urban-exp-tourism-
dates/{cultural_urban_expl_tourism_id}',
[CulturalUrbanExplorationTourismDatesController::class,
'getCulturalUrbanExplorationTourismDates']);

Route::post('cultural-urban-tourism-dates',
[CulturalUrbanTourismDatesController::class,
'manageCulturalUrbanTourismDates']);
Route::post('cultural-urban-tourism-
dates/{cultural_urban_tourism_id}',
[CulturalUrbanTourismDatesController::class,
'getCulturalUrbanTourismDates']);

Route::post('dark-tourism-dates',
[DarkTourismDatesController::class, 'manageDarkTourismDates']);
Route::post('dark-tourism-dates/{dark_tourism_id}',
[DarkTourismDatesController::class, 'getDarkTourismDates']);

Route::post('ecotourism-dates', [EcotourismDatesController::class,
'manageEcotourismDates']);
Route::post('ecotourism-dates/{ecotourism_id}',
[ecotourismDatesController::class, 'getEcotourismDates']);

Route::post('education-tourism-dates',
[EducationDevelopmentTourismDatesController::class,
'manageEducationDevelopmentTourismDates']);
Route::post('education-tourism-
dates/{education_development_tourism_id}',
[EducationDevelopmentTourismDatesController::class,
'getEducationDevelopmentTourismDates']);

Route::post('enrichment-tourism-dates',
[EnrichmentSkillBuildingTourismDatesController::class,
'manageEnrichmentSkillBuildingTourismDates']);
Route::post('enrichment-tourism-dates/{enrichment_tourism_id}',
[EnrichmentSkillBuildingTourismDatesController::class,
'getEnrichmentSkillBuildingTourismDates']);

```

```

    Route::post('exploration-tourism-dates',
[ExplorationTourismDatesController::class,
'manageExplorationTourismDates']);
    Route::post('exploration-tourism-dates/{exploration_tourism_id}',
[ExplorationTourismDatesController::class,
'getExplorationTourismDates']);

    Route::post('fishing-tourism-dates',
[FishingTourismDatesController::class, 'manageFishingTourismDates']);
    Route::post('fishing-tourism-dates/{fishing_tourism_id}',
[FishingTourismDatesController::class, 'getFishingTourismDates']);

    Route::post('gastronomic-tourism-dates',
[GastronomicTourismDatesController::class,
'manageGastronomicTourismDates']);
    Route::post('gastronomic-tourism-dates/{gastronomic_tourism_id}',
[GastronomicTourismDatesController::class,
'getGastronomicTourismDates']);

    Route::post('geotourism-dates', [GeotourismDatesController::class,
'manageGeotourismDates']);
    Route::post('geotourism-dates/{geotourism_id}',
[GeotourismDatesController::class, 'getGeotourismDates']);

    Route::post('sacred-site-visits-tourism-dates',
[SacredSiteVisitsTourismDatesController::class,
'manageSacredSiteVisitsTourismDates']);
    Route::post('sacred-site-visits-tourism-
dates/{sacred_site_visits_tourism_id}',
[SacredSiteVisitsTourismDatesController::class,
'getSacredSiteVisitsTourismDates']);

    Route::post('sporting-events-tourism-dates',
[SportingEventsTourismDatesController::class,
'manageSportingEventsTourismDates']);
    Route::post('sporting-events-tourism-
dates/{sporting_events_tourism_id}',
[SportingEventsTourismDatesController::class,
'getSportingEventsTourismDates']);

    Route::post('urban-entertainment-tourism-dates',
[UrbanEntertainmentTourismDatesController::class,
'manageUrbanEntertainmentTourismDates']);
    Route::post('urban-entertainment-tourism-
dates/{urban_entertainment_tourism_id}',
[UrbanEntertainmentTourismDatesController::class,
'getUrbanEntertainmentTourismDates']);

```

```

    Route::post('water-activities-tourism-dates',
[WaterActivitiesTourismDatesController::class,
'manageWaterActivitiesTourismDates']);
    Route::post('water-activities-tourism-
dates/{water_activities_tourism_id}',
[WaterActivitiesTourismDatesController::class,
'getWaterActivitiesTourismDates']);

    Route::post('wedding-tourism-dates',
[WeddingTourismDatesController::class, 'manageWeddingTourismDates']);
    Route::post('wedding-tourism-dates/{wedding_tourism_id}',
[WeddingTourismDatesController::class, 'getWeddingTourismDates']);

    Route::post('medical-tourism-dates',
[MedicalTourismDatesController::class, 'manageMedicalTourismDates']);
    Route::post('medical-tourism-dates/{medical_tourism_id}',
[MedicalTourismDatesController::class, 'getMedicalTourismDates']);

    Route::post('spa-tourism-dates', [SpaTourismDatesController::class,
'manageSpaTourismDates']);
    Route::post('spa-tourism-dates/{spa_tourism_id}',
[SpaTourismDatesController::class, 'getSpaTourismDates']);

    Route::post('wellness-tourism-dates',
[WellnessTourismDatesController::class, 'manageWellnessTourismDates']);
    Route::post('wellness-tourism-dates/{wellness_tourism_id}',
[WellnessTourismDatesController::class, 'getWellnessTourismDates']);

    Route::post('wine-tourism-dates',
[WineTourismDatesController::class, 'manageWineTourismDates']);
    Route::post('wine-tourism-dates/{wine_tourism_id}',
[WineTourismDatesController::class, 'getWineTourismDates']);

    // You can add more protected routes here

    Route::post('adventures-tourism/{id}/duplicate',
[AdventuresTourismController::class, 'duplicate']);
    Route::post('agritourism/{id}/duplicate',
[AgritourismController::class, 'duplicate']);
    Route::post('air-activities-tourism/{id}/duplicate',
[AirActivitiesTourismController::class, 'duplicate']);
    Route::post('city-games-tourism/{id}/duplicate',
[CityGamesTourismController::class, 'duplicate']);
    Route::post('cultural-festival-tourism/{id}/duplicate',
[CulturalFestivalTourismController::class, 'duplicate']);
    Route::post('cultural-urban-exp-tourism/{id}/duplicate',
[CulturalUrbanExplorationTourismController::class, 'duplicate']);

```

```

    Route::post('cultural-urban-tourism/{id}/duplicate',
[CulturalUrbanTourismController::class, 'duplicate']);
    Route::post('dark-tourism/{id}/duplicate',
[DarkTourismController::class, 'duplicate']);
    Route::post('ecotourism/{id}/duplicate',
[EcotourismController::class, 'duplicate']);
    Route::post('education-tourism/{id}/duplicate',
[EducationDevelopmentTourismController::class, 'duplicate']);
    Route::post('enrichment-tourism/{id}/duplicate',
[EnrichmentSkillBuildingTourismController::class, 'duplicate']);
    Route::post('exploration-tourism/{id}/duplicate',
[ExplorationTourismController::class, 'duplicate']);
    Route::post('fishing-tourism/{id}/duplicate',
[FishingTourismController::class, 'duplicate']);
    Route::post('gastronomic-tourism/{id}/duplicate',
[GastronomicTourismController::class, 'duplicate']);
    Route::post('geotourism/{id}/duplicate',
[GeotourismController::class, 'duplicate']);
    Route::post('sacred-site-visits-tourism/{id}/duplicate',
[SacredSiteVisitsTourismController::class, 'duplicate']);
    Route::post('sporting-events-tourism/{id}/duplicate',
[SportingEventsTourismController::class, 'duplicate']);
    Route::post('urban-entertainment-tourism/{id}/duplicate',
[UrbanEntertainmentTourismController::class, 'duplicate']);
    Route::post('water-activities-tourism/{id}/duplicate',
[WaterActivitiesTourismController::class, 'duplicate']);
    Route::post('wedding-tourism/{id}/duplicate',
[WeddingTourismController::class, 'duplicate']);
    Route::post('wine-tourism/{id}/duplicate',
[WineTourismController::class, 'duplicate']);

    Route::post('medical-tourism/{id}/duplicate',
[MedicalTourismController::class, 'duplicate']);
    Route::post('spa-tourism/{id}/duplicate',
[SpaTourismController::class, 'duplicate']);
    Route::post('wellness-tourism/{id}/duplicate',
[WellnessTourismController::class, 'duplicate']);
    Route::post('transportation/{id}/duplicate',
[TransportationController::class, 'duplicate']);

    // Translations
    Route::post('wine-tourism-translation',
[WineTourismTranslationController::class, 'store']);
    Route::put('wine-tourism-translation',
[WineTourismTranslationController::class, 'update']);

```

```
});  
  
Route::post('request-translation',[TranslationController::class,  
    'sendTranslationRequest']);  
Route::post('request-document-  
translation',[TranslationController::class,  
    'sendTranslationDocumentRequest']);  
Route::post('incoming-translation', [TranslationController::class,  
    'handleIncomingTranslation']);  
Route::post('incoming-document-translation',  
    [TranslationController::class, 'handleIncomingDocumentTranslation']);  
Route::get('check-translation-status', [TranslationController::class,  
    'checkTranslationStatus']);  
Route::post('get-available-locales', [TranslationController::class,  
    'getAvailableLocales']);
```