Overview + Database

Hone rah kun ballashet bl project taba3 internship.

System Overview:
BabiVoyage is a RESTful API backend for a flight booking platform targeting West African air travel. The system allows users to **search** and **book** flights, **manage** their reservations, and **receive** booking confirmations — all through a secure, token-based API.

Core Features:

1- Authentication & Users:
- User registration with secure password hashing
- Token-based login/logout via Laravel Sanctum
- User profile management

2- Flights:
- Admin can create/update/delete flights
- Users can browse and search available flights
- Flights have origin, destination, departure/arrival times, price, and available seats

3- Bookings:
- Users can book available flights
- Each booking holds passenger details and seat count
- Booking status management (pending → confirmed → cancelled)

4- Passengers:
- Each booking can include multiple passengers
- Linked to the booking, not the user account directly

Database Design:

Users Table built-in in Laravel

Airports Table:

Column	Type	Notes
id	bigint PK	auto-increment
name	string	e.g. "Murtala Muhammed International"
code	string	unique, IATA code e.g. "LOS"
city	string	e.g. "Lagos"
country	string	e.g. "Nigeria"
timestamps	—	created_at, updated_at

Airlines Table:

Column	Type	Notes
id	bigint PK	auto-increment
name	string	e.g. "Air Côte d'Ivoire"
code	string	unique, IATA code e.g. "HF"
logo_url	string	nullable
timestamps	—	created_at, updated_at

Flights Table:

Column	Type	Notes
Id	bigint PK	auto-increment
flight_number	string	unique, e.g. "HF204"
airline_id	FK → airlines	required
origin_airport_id	FK → airports	required
destination_airport_id	FK → airports	required
departure_time	datetime	required
arrival_time	datetime	required
price	decimal(10,2)	required
total_seats	integer	required
available_seats	integer	required
status	enum	'scheduled', 'cancelled', 'completed'
timestamps	———	created_at, updated_at

Bookings Table:

Column	Type	Notes
id	bigint PK	auto-increment
user_id	FK → users	required
flight_id	FK → flights	required
booking_reference	string	unique, e.g. "BV-20260330-XXXX"
seat_count	integer	how many seats booked
total_price	decimal(10,2)	seat_count × flight price
status	enum	'pending', 'confirmed', 'cancelled'
timestamps	—	created_at, updated_at

Passengers Table:

Column	Type	Notes
id	bigint PK	auto-increment
booking_id	FK → bookings	required
first_name	string	required
last_name	string	required
passport_number	string	nullable
nationality	string	nullable
date_of_birth	date	nullable
timestamps	—	created_at, updated_at

Saved_flights Table:

Column	Type	Notes
id	bigint PK	auto-increment
user_id	FK → users	**required**
flight_id	FK → flights	**required**
unique(user_id, flight_id)	index	composite unique key
timestamps	—	created_at, updated_at

The Relationship Map:

Users ———————————————— Bookings
(one user → many bookings) 1 : N

Flights ———————————————— Bookings
(one flight → many bookings) 1 : N

Airlines ———————————————— Flights
(one airline → many flights) 1 : N

Airports ———————————————— Flights (as origin)
(one airport → many flights) 1 : N

Airports ———————————————— Flights (as destination)
(one airport → many flights) 1 : N

Bookings ———————————————— Passengers
(one booking → many passengers) 1 : N

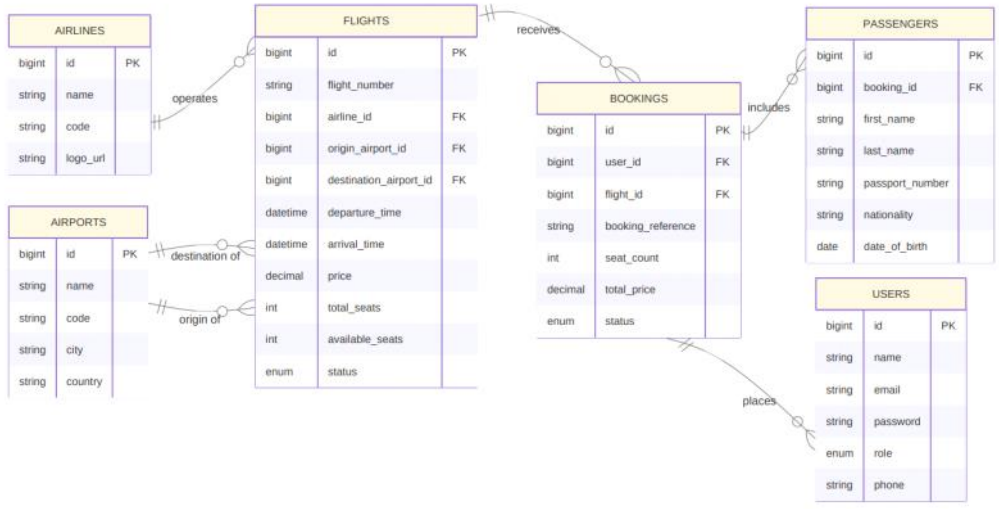
Users———————————————— Flights
(many users → many flights) N : N

Relationship Types Summary:

User ->Bookings	One-to-Many	bookings.user_id
Flight ->Bookings	One-to-Many	bookings.flight_id
Airline -> Flights	One-to-Many	flights.airline_id
Airport -> Flights (origin)	One-to-Many	flights.origin_airport_id
Airport -> Flights (destination)	One-to-Many	flights.destination_airport_id
Booking -> Passengers	One-to-Many	passengers.booking_id
User <-> Flights	Many-to-Many	saved_flights.user_id and saved_flights.flight_id

#	Table	Depends On
1	airlines	—
2	airports	—
3	users	—
4	flights	airlines, airports
5	bookings	users, flights
6	passengers	bookings
7	saved_flights	users, flights

Visual ERD Diagram



Repository Setup

First, I turn on the Wamp server, it should be green

Then I open the Vscode.
We can check the php and composer version to make sure all is running correctly.

In the Vscode:

- cd C:\wamp64\www
- composer create-project --prefer-dist laravel/laravel webInternshipProj

It may take some time.

```
PS C:\wamp64\www> composer create-project --prefer-dist laravel/laravel webInternshipProj

  Creating migration table ..................................................... 16.78ms DONE

  INFO  Running migrations.

  0001_01_01_000000_create_users_table ......................................... 40.85ms DONE
  0001_01_01_000001_create_cache_table ......................................... 23.49ms DONE
  0001_01_01_000002_create_jobs_table .......................................... 33.61ms DONE

PS C:\wamp64\www>
```

In PHPMyAdmin (in wamp), we create a new database: webInternship_db

Then, in the .env file we change the name of the database

run migrations: php artisan migrate
It gives me this error:
SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length is 1000 bytes (Connection: mysql, Host: 127.0.0.1, Port: 3306, Database: webInternship_db, SQL: alter table `users` add unique `users_email_unique`(`email`))

How to solve it?
in the app/Providers/AppServiceProvider.php

```
public function boot(): void
{
    Schema::defaultStringLength(191);
}
```

I set the default String Length to 191, so it does not exceed the 1000 bytes limit (utf8mb4)

After doing: php artisan migrate:fresh => runs all migrations.

The application runs perfectly.

Repository Creation:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj
$ git init
Initialized empty Git repository in C:/wamp64/www/webInternshipProj/.git/

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (master)
$ git add .

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (master)
$ git commit -m "Creating the Repository"
[master (root-commit) cb88df7] Creating the Repository
```

I create a new github repository "webInternshipProj" in the github website online

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (master)
$ git remote add origin https://github.com/MedianeOz/webInternshipProj.git

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (master)
$ git branch -M main

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (main)
$ git push -u origin main
Enumerating objects: 76, done.
Counting objects: 100% (76/76), done.
Delta compression using up to 4 threads
Compressing objects: 100% (59/59), done.
Writing objects: 100% (76/76), 72.42 KiB | 2.41 MiB/s, done.
Total 76 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/MedianeOz/webInternshipProj.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (main)
$ git checkout -b develop
Switched to a new branch 'develop'

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ |
```

I create a new repository online, and I link it to our project
And then I create a new branch called develop

Migrations

Now, I will create the migrations and database tables.

We already have the users table built in Laravel

This will be the creation order of the tables:
1. **airlines** (no dependencies)
2. **airports** (no dependencies)
3. **users** (already exists in Laravel)
4. **flights** (depends on: airlines, airports)
5. **bookings** (depends on: users, flights)
6. **passengers** (depends on: bookings)

Creation of Table:

```
$ php artisan make:migration create_airlines_table
php artisan make:migration create_airports_table
php artisan make:migration create_flights_table
php artisan make:migration create_bookings_table
php artisan make:migration create_passengers_table

  INFO  Migration [C:\wamp64\www\webInternshipProj\database\migrations\2026_04_24
_073046_create_airlines_table.php] created successfully.

  INFO  Migration [C:\wamp64\www\webInternshipProj\database\migrations\2026_04_24
_073050_create_airports_table.php] created successfully.

  INFO  Migration [C:\wamp64\www\webInternshipProj\database\migrations\2026_04_24
_073050_create_flights_table.php] created successfully.

  INFO  Migration [C:\wamp64\www\webInternshipProj\database\migrations\2026_04_24
_073051_create_bookings_table.php] created successfully.

  INFO  Migration [C:\wamp64\www\webInternshipProj\database\migrations\2026_04_24
_073052_create_passengers_table.php] created successfully.
```

Now I change the up function of each file generated based on the tables we've created earlier.

I will only put 2 pictures

Airlines Table:

```php
    public function up(): void
    {
        Schema::create('airlines', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('code', 10)->unique();  // IATA code e.g. "HF"
            $table->string('logo_url')->nullable();
            $table->timestamps();
    });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('airlines');
    }
```

Flights Table:

```php
public function up(): void
{
    Schema::create('flights', function (Blueprint $table) {
        $table->id();
        $table->string('flight_number')->unique();  // e.g. "BV-101"

        $table->foreignId('airline_id')
            ->constrained('airlines')
            ->onDelete('cascade');

        $table->foreignId('origin_airport_id')
            ->constrained('airports')
            ->onDelete('restrict');

        $table->foreignId('destination_airport_id')
            ->constrained('airports')
            ->onDelete('restrict');

        $table->dateTime('departure_time');
        $table->dateTime('arrival_time');
        $table->decimal('price', 10, 2);
        $table->unsignedInteger('total_seats'); //seats can never be negative
        $table->unsignedInteger('available_seats');
        $table->enum('status', ['scheduled', 'delayed', 'cancelled', 'completed'])
            ->default('scheduled');
        $table->timestamps();
    });
}
```

I want to add 2 fields to the users table:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:migration add_fields_to_users_table

  INFO  Migration [C:\wamp64\www\webInternshipProj\database\migrations\2026_04_24
_074803_add_fields_to_users_table.php] created successfully.
```

I create a new migration and I link it to the users table like this:

```php
public function up(): void
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('phone')->nullable()->after('email');
        $table->enum('role', ['user', 'admin'])->default('user')->after('phone');
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn(['phone', 'role']);
    });
}
```

I want to add 2 fields to the existing users table:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:migration add_fields_to_users_table
```

Then in the up and down function:

```php
public function up(): void
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('phone')->nullable()->after('email');
        $table->enum('role', ['user', 'admin'])->default('user')->after('phone');
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn(['phone', 'role']);
    });
}
```

Since I only have one-to-many relationships and not many-to-many : I don't need pivot tables

But I will add a relationship between users and flights, just to add the pivot table.
A user can save many flights, a flight can be saved by many users. (many-to-many)

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:migration create_saved_flights_table

  INFO  Migration [C:\wamp64\www\webInternshipProj\database\migrations\2026_04_
24_080948_create_saved_flights_table.php] created successfully.
```

Then:

```php
public function up(): void
{
    Schema::create('saved_flights', function (Blueprint $table) {
        $table->id();

        $table->foreignId('user_id')
            ->constrained('users')
            ->onDelete('cascade');

        $table->foreignId('flight_id')
            ->constrained('flights')
            ->onDelete('cascade');

        // Prevent a user from saving the same flight twice
        $table->unique(['user_id', 'flight_id']);

        $table->timestamps();
    });
}
```

Since I added a new table, I need to adjust the ERD diagram and the previous tables

In the end, I do the migration:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan migrate

  INFO  Running migrations.

  2026_04_24_073046_create_airlines_table ..................... 204.28ms DONE
  2026_04_24_073050_create_airports_table ...................... 45.10ms DONE
  2026_04_24_073050_create_flights_table ....................... 226.70ms DONE
  2026_04_24_073051_create_bookings_table ...................... 100.66ms DONE
  2026_04_24_073052_create_passengers_table ..................... 41.15ms DONE
  2026_04_24_074803_add_fields_to_users_table ................... 65.72ms DONE
  2026_04_24_080948_create_saved_flights_table ................. 105.54ms DONE
```

The migrations done successfully

Models

Now I will create the Eloquent models

First of all, the model is singular, not like the table.
Second the User model is built-in in laravel

Now I will create 5 models: flight, Booking, Passenger, Airline and Airport

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (main)
$ php artisan make:model Flight
php artisan make:model Booking
php artisan make:model Passenger

  INFO  Model [C:\wamp64\www\webInternshipProj\app\Models\Flight.php] created s
uccessfully.

  INFO  Model [C:\wamp64\www\webInternshipProj\app\Models\Booking.php] created
successfully.

  INFO  Model [C:\wamp64\www\webInternshipProj\app\Models\Passenger.php] create
d successfully.

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:model Airline

  INFO  Model [C:\wamp64\www\webInternshipProj\app\Models\Airline.php] created
successfully.

Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:model Airport

  INFO  Model [C:\wamp64\www\webInternshipProj\app\Models\Airport.php] created
successfully.
```

Created successfully

In the User model:
I need to adjust the fillable, add use HasMany, and add the relations

```php
use Database\Factories\UserFactory;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
```

```php
protected $fillable = [
    'name',
    'email',
    'password',
    'phone',
    'role',
];
```

```php
// One user can have many bookings
public function bookings(): HasMany
{
    return $this->hasMany(Booking::class);
}

// Many-to-many: saved flights (pivot table: saved_flights)
public function savedFlights(): BelongsToMany
{
    return $this->belongsToMany(Flight::class, 'saved_flights')
                ->withTimestamps();
}
```

Now the Flight model:

```php
class Flight extends Model
{
    protected $fillable = [
        'flight_number',
        'airline_id',
        'origin_airport_id',
        'destination_airport_id',
        'departure_time',
        'arrival_time',
        'price',
        'total_seats',
        'available_seats',
        'status',
    ];

    protected function casts(): array
    {
        return [
            'departure_time'  => 'datetime',
            'arrival_time'    => 'datetime',
            'price'           => 'decimal:2',
            'total_seats'     => 'integer',
            'available_seats' => 'integer',
        ];
    }
```

```php
// One flight can have many bookings
public function bookings(): HasMany
{
    return $this->hasMany(Booking::class);
}

public function airline(): BelongsTo
{
    return $this->belongsTo(Airline::class);
}

public function originAirport(): BelongsTo
{
    return $this->belongsTo(Airport::class, 'origin_airport_id');
}

public function destinationAirport(): BelongsTo
{
    return $this->belongsTo(Airport::class, 'destination_airport_id');
}

public function savedByUsers(): BelongsToMany
{
    return $this->belongsToMany(User::class, 'saved_flights')
                ->withTimestamps();
}
```

I put all fillables from the table I created earlier
"casts()" to convert string (departure_at and arrival_at) to a carbon datetime objects.

For Booking model:

```php
// A booking belongs to one user
public function user(): BelongsTo
{
    return $this->belongsTo(User::class);
}

// A booking belongs to one flight
public function flight(): BelongsTo
{
    return $this->belongsTo(Flight::class);
}

// A booking can have many passengers
public function passengers(): HasMany
{
    return $this->hasMany(Passenger::class);
}
```

For Passenger model:

```php
class Passenger extends Model
{
    protected $fillable = [
        'booking_id',
        'first_name',
        'last_name',
        'passport_number',
        'nationality',
        'date_of_birth',
    ];

    protected function casts(): array
    {
        return [
            'date_of_birth' => 'date',
        ];
    }

    // A passenger belongs to one booking
    public function booking(): BelongsTo
    {
        return $this->belongsTo(Booking::class);
    }
}
```

Airline model:

```php
class Airline extends Model
{
    protected $fillable = [
        'name',
        'code',
        'logo_url'
        ];

    public function flights(): HasMany
    {
        return $this->hasMany(Flight::class);
    }
}
```

Airport model:

```php
class Airport extends Model
{
    protected $fillable = [
        'name',
        'code',
        'city',
        'country'
        ];

    // Flights departing from this airport
    public function originatingFlights(): HasMany
    {
        return $this->hasMany(Flight::class, 'origin_airport_id');
    }

    // Flights arriving at this airport
    public function arrivingFlights(): HasMany
    {
        return $this->hasMany(Flight::class, 'destination_airport_id');
    }
```

A small note:
In both Flight and User model:
we've put a function saved_flights that belongsToMany (many-to-many), that refers to the other table, as shown in the pictures.

```
User
 ├── hasMany ──────────→ Bookings
 |                          ├── belongsTo → User
 |                          ├── belongsTo → Flight
 |                          └── hasMany  → Passengers
 |                                            └── belongsTo → Booking
 └── belongsToMany ──→ Flights (via saved_flights)

Flight
 ├── belongsTo ────────→ Airline
 ├── belongsTo ────────→ Airport (origin)
 ├── belongsTo ────────→ Airport (destination)
 ├── hasMany ──────────→ Bookings
 └── belongsToMany ──→ Users (via saved_flights)
```

```
Airline
 └── hasMany ──────────→ Flights

Airport
 ├── hasMany ──────────→ Flights (as origin)
 └── hasMany ──────────→ Flights (as destination)
```

Registration API

Here's what we need to build:

1. Create AuthController
2. Create a RegisterRequest (Form Request for validation)
3. Write the register function
4. Add the route in api.php
5. Test in Postman

AuthController:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:controller API/AuthController

  INFO  Controller [C:\wamp64\www\webInternshipProj\app\Http\Controllers\API\Au
thController.php] created successfully.
```

Claude suggestion:
Instead of writing validation directly inside the controller, we use a **Form Request** — this is the professional Laravel way. It keeps the controller clean.

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:request Auth/RegisterRequest

  INFO  Request [C:\wamp64\www\webInternshipProj\app\Http\Requests\Auth\Registe
rRequest.php] created successfully.
```

This is for the validation rules, and error messages

```php
// true = everyone (since registration is public)
public function authorize(): bool
{
    return true;
}

public function rules(): array
{
    return [
        'name'     => ['required', 'string', 'max:255'],
        'email'    => ['required', 'string', 'email', 'max:255', 'unique:users,email'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
        'phone'    => ['nullable', 'string', 'max:20'],
    ];
}

// Custom error messages (optional but professional)
public function messages(): array
{
    return [
        'name.required'      => 'Full name is required.',
        'email.required'     => 'Email address is required.',
        'email.email'        => 'Please provide a valid email address.',
        'email.unique'       => 'This email is already registered.',
        'password.required'  => 'Password is required.',
        'password.min'       => 'Password must be at least 8 characters.',
        'password.confirmed' => 'Password confirmation does not match.',
    ];
}
```

Why confirmed on password? Laravel's confirmed rule automatically looks for a password_confirmation field in the request and checks they match. So in Postman I'll send both password and password_confirmation.

Then I adjust the register funtion in AuthController

```php
// by RegisterRequest — no need to validate manually here

// Create the user
$user = User::create([
    'name'     => $request->name,
    'email'    => $request->email,
    'password' => $request->password, // hashed automatically via cast in User model
    'phone'    => $request->phone,
    'role'     => 'user',
]);

// Generate Sanctum token for immediate login after registration
$token = $user->createToken('auth_token')->plainTextToken;

return response()->json([
    'success' => true,
    'message' => 'Registration successful.',
    'data'    => [
        'user'  => [
            'id'    => $user->id,
            'name'  => $user->name,
            'email' => $user->email,
            'phone' => $user->phone,
            'role'  => $user->role,
        ],
        'token' => $token,
    ],
], 201);
```

In routes/api.php :

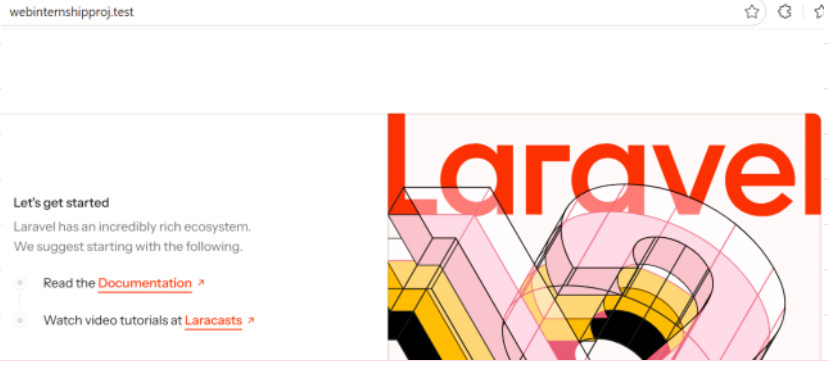
```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\API\AuthController;

// Public routes (no token needed)
Route::post('/register', [AuthController::class, 'register']);
```

We add the route of the registration

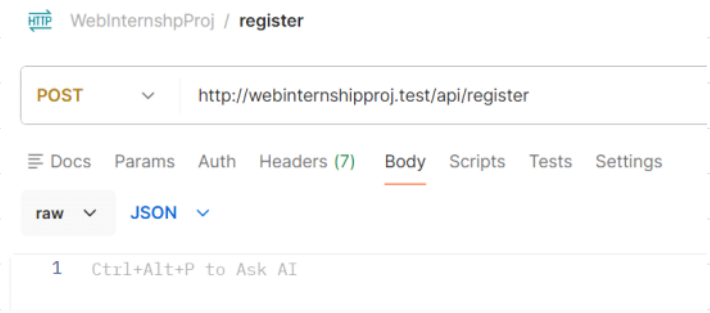
Then I test it in postman, to make sure all is running.

First I create a new collection named WebInternshipProj
"Add a request" to create new route.
I choose post request, and name it register

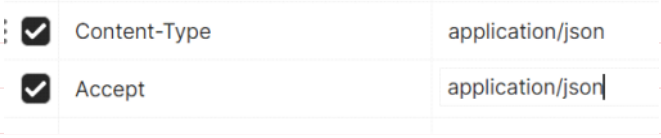
Then I need to change the url of project,
from localhost/udemyWebInter/public/ , to webinternshipproj.test



Now I put http//webinternshipproj.test/api/register in the url of the new post request



I check that:
Header: Content-Type is application/json |~~| Header: Accept is application/json

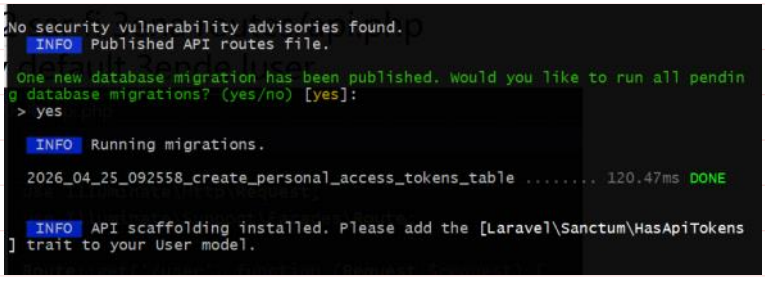


Now I will be adding the data as json, and test it in postman

When I run this json data:

```
{
    "name": "Mediane Ozeir",
    "email": "mediane@gmail.com",
    "password": "password123",
    "password_confirmation": "password123",
    "phone": "12345678"
}
```

It gives me error, why? It tells me that there is no api/register.
So I deleted the api file I created manually,
and I used this command: **php artisan install:api**
This will create the api file in the routes, and install all dependencies needed.



It tells me to add Laravel\Sanctum\HasApiToken to the User model (already did)

Now this is the routes/api.php (user route created automatically)

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\API\AuthController;

Route::get('/user', function (Request $request) {
    return $request->user();
})->middleware('auth:sanctum');


// Public routes (no token needed)
Route::post('/register', [AuthController::class, 'register']);
```

Now when I rerun the json code:


```json
"success": true,
"message": "Registration successful.",
"data": {
    "user": {
        "id": 1,
        "name": "Mediane Ozeir",
        "email": "mediane@gmail.com",
        "phone": "12345678",
        "role": "user"
    },
    "token": "1|2fJkAZ7lC1XMg26NEUeEVY4jSoVrPEV6dKg1tPHUa6902c8b"
}
```


I will send bad data and test it
no name:


```json
"email": "medianeO@gmail.com",
"password": "password123",
"password_confirmation": "password123",
"phone": "12345678"
```


422 Unprocessable Cor

Preview Debug with AI


```json
"message": "Full name is required.",
"errors": {
    "name": [
        "Full name is required."
    ]
```


wrong email format:


```json
"email": "medianegmail.com",
"password": "password123",
"password_confirmation": "password123",
"phone": "12345678"
```


422 Unprocessable Content 91 ms 45

Preview Debug with AI


```json
"message": "Please provide a valid email address.",
"errors": {
    "email": [
        "Please provide a valid email address."
    ]
```


existing password:

```
"name": "Mediane Ozeir",
"email": "mediane@gmail.com",
"password": "password123",
"password_confirmation": "password123",
"phone": "12345678"
```


422 Unprocessable Content · 115 ms

Preview Debug with AI


```
"message": "This email is already registered.",
"errors": {
    "email": [
        "This email is already registered."
    ]
```


passwords unmatch:


```
"name": "Mediane Ozeir",
"email": "medianeOz@gmail.com",
"password": "password123",
"password_confirmation": "password",
"phone": "12345678"
```


422 Unprocessable Content · 104 ms · 46

Preview Debug with AI


```
"message": "Password confirmation does not match.",
"errors": {
    "password": [
        "Password confirmation does not match."
    ]
```

Authentication API

Here's what we need to build:

1. Create LoginRequest (and Form Request for validation)
2. Write the login function in AuthController
3. Write the logout function in AuthController
4. Add the routes in api.php
5. Test in Postman

I will start with the Login Form Request

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:request Auth/LoginRequest

  INFO  Request [C:\wamp64\www\webInternshipProj\app\Http\Requests\Auth\LoginRequest.php] created successfully.
```

In LoginRequest.php

```php
public function rules(): array
{
    return [
        'email'    => ['required', 'string', 'email'],
        'password' => ['required', 'string'],
    ];
}

public function messages(): array
{
    return [
        'email.required'     => 'Email address is required.',
        'email.email'        => 'Please provide a valid email address.',
        'password.required'  => 'Password is required.',
    ];
}
```

Now I create the logic of login and logout in AuthController

Login Function:

```php
public function login(LoginRequest $request)
{
    // Check if credentials are correct
    if (!Auth::attempt($request->only('email', 'password'))) {
        return response()->json([
            'success' => false,
            'message' => 'Invalid credentials. Please check your email and password.',
        ], 401);
    }

    // Get the authenticated user
    $user = Auth::user();

    // Revoke all old tokens (optional but clean — one session at a time)
    $user->tokens()->delete();

    // Generate a fresh token
    $token = $user->createToken('auth_token')->plainTextToken;

    return response()->json([
        'success' => true,
        'message' => 'Login successful.',
        'data'    => [
            'user'  => [
                'id'    => $user->id,
                'name'  => $user->name,
                'email' => $user->email,
                'phone' => $user->phone,
                'role'  => $user->role,
            ],
            'token' => $token,
        ],
    ], 200);
}
```

Logout Function

```php
public function logout()
{
    // Revoke all tokens for the currently authenticated user
    Auth::user()->tokens()->delete();

    return response()->json([
        'success' => true,
        'message' => 'Logged out successfully.',
    ], 200);
}
```

In routes/api:

```php
// ----------------------------------------------------
// Public routes (no token needed)
// ----------------------------------------------------

Route::post('/register', [AuthController::class, 'register']);

Route::post('/login', [AuthController::class, 'login']);



// ----------------------------------------------------
// Protected Routes (token required)
// ----------------------------------------------------

Route::middleware('auth:sanctum')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);
});
```

In the login, the token is created, and then it is used to do any action, and then it is deleted in the logout (to do the logout we need the token)

Why middleware('auth:sanctum') on logout? Because I need to be logged in to log out. If a user sends a request to /logout without a valid token, Sanctum automatically returns 401 Unauthenticated — I don't need to handle that manually.

```
CLIENT                          SERVER                          DATABASE
  |                               |                                |
  |  POST /api/login              |                                |
  |  {email, password}            |                                |
  |------------------------------>|                                |
  |                               |                                |
  |                               |  Check credentials             |
  |                               |  Hash password, compare        |
  |                               |                                |
  |                               |  CREATE token in               |
  |                               |  personal_access_tokens        |
  |                               |------------------------------>|
  |                               |                                |
  |  200 OK                       |                                |
  |  {token: "1|abc123..."}       |                                |
  |<------------------------------|                                |
  |                               |                                |
  |  GET /api/bookings            |                                |
  |  Authorization: Bearer 1|abc..|                                |
  |------------------------------>|                                |
  |                               |  Validate token                |
  |                               |  Find user from token          |
  |                               |------------------------------>|
  |                               |                                |
  |  200 OK                       |                                |
  |  {bookings: [...]}            |                                |
  |<------------------------------|                                |
```

Now I will test it in Postman:
I create a new post request with url /api/login

Login Successful

```
{
    "email": "mediane@gmail.com",
    "password": "password123"
```

200 OK · 496 ms · 543 B

ON · Preview · Visualize

```
    "success": true,
    "message": "Login successful.",
    "data": {
        "user": {
            "id": 1,
            "name": "Mediane Ozeir",
            "email": "mediane@gmail.com",
            "phone": "12345678",
            "role": "user"
        },
        "token": "4|bLLap00iOhEqOLOYuAC4UOkNmfixoAXOmPFpaJVz9b5b94b6"
```

Wrong Password:

```
    "email": "mediane@gmail.com",
    "password": "password"
```

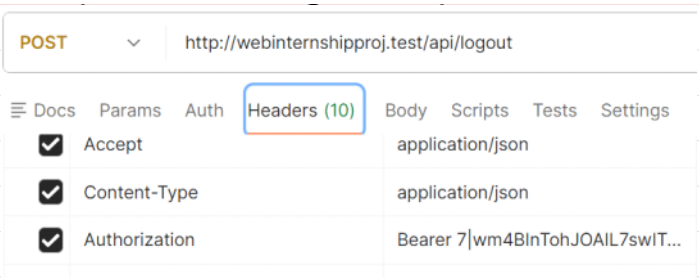
401 Unauthorized · 469 ms · 425 B · Save

N · Preview · Pass the correct auth credentials

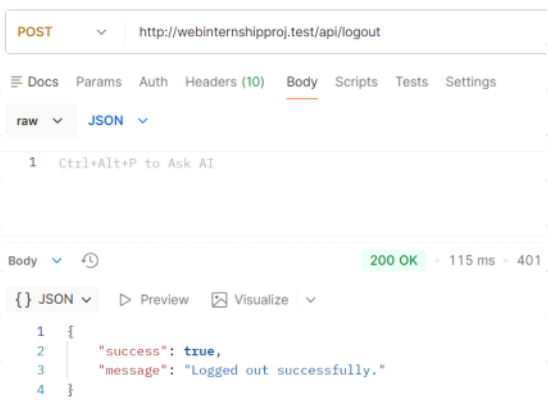
```
{
    "success": false,
    "message": "Invalid credentials. Please check your email and password."
}
```

Now for Logout:
I create a new post logout request with url /api/logout
But here I need to add one thing to the header

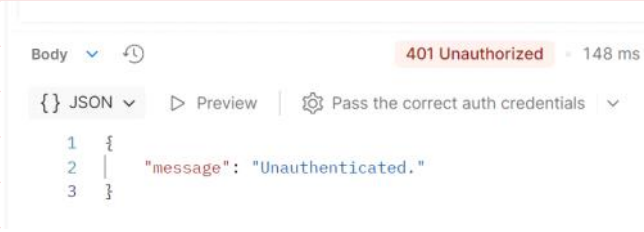
When I login, it creates a specific token, I will copy it
And paste it the logout request header, like this:



So when I send the logout request, I don't send anything in the body, it does it based on the token of the login.



If I remove the token from the header, it will give me this error:



How the token flow works:

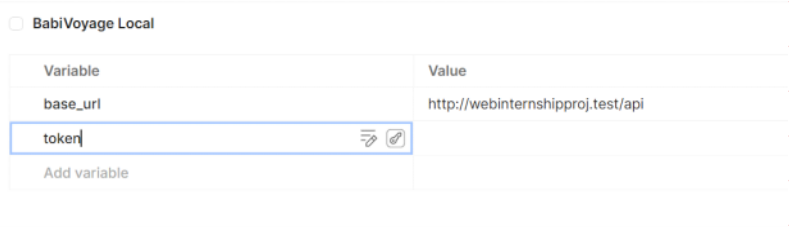
```
POST /register  →  creates user  →  returns token
POST /login     →  checks credentials  →  deletes old tokens  →  returns new token
POST /logout    →  requires token in header  →  deletes all tokens  →  logged out
```

Postman Collection

Now I'll adjust the postman collection, and create all endpoints needed

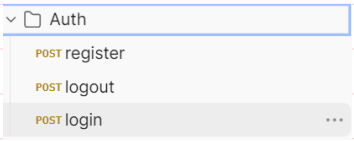
I will use environment variables (base_url, token).

I create a postman environmnent: it stores variables that change based on where I run the API

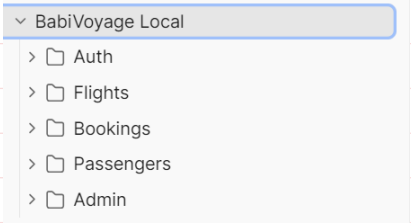


Like this, I don't need to every time right the whole url, I can just write base_url
And for the token, it will be saved and deleted automatically (based on a code I will write it later on).

I create a folder Auth, and drag the three requests: login, logout and register into it.



I create the folders:




```
├── Flights
│   ├── GET    /api/flights              (list all flights)
│   ├── GET    /api/flights/{id}         (get one flight)
│   ├── GET    /api/flights/search       (search flights)
│   ├── POST   /api/admin/flights        (admin: create)
│   ├── PUT    /api/admin/flights/{id}   (admin: update)
│   └── DELETE /api/admin/flights/{id}  (admin: delete)
│
├── Bookings
│   ├── GET    /api/bookings             (my bookings)
│   ├── POST   /api/bookings             (create booking)
│   ├── GET    /api/bookings/{id}        (get one booking)
│   └── PATCH  /api/bookings/{id}/cancel (cancel booking)
│
└── Passengers
    ├── GET    /api/bookings/{id}/passengers       (list passengers)
    ├── POST   /api/bookings/{id}/passengers       (add passenger)
    └── DELETE /api/bookings/{id}/passengers/{pid} (remove passenger)
```

BabiVoyage Local > Auth > **register**

POST {{base_url}} /register

Docs Params Authorization Headers 10 Body Scripts Settings

Pre-request

Post-response

```
if (pm.response.code === 201) {
    const json = pm.response.json();
    pm.environment.set("token", json.data.token);
    pm.test("Token stored from registration", () => {
        pm.expect(pm.environment.get("token")).to.not.be.null;
    });
}
```

Like this, when the user register, the token is stored automatically in the token variable created in the env.
It should be done in the Post-response and not Pre-request (when I write it first in the pre-request it gave me error).

The same for login:

POST {{base_url}} /login

Docs Params Authorization Headers 10 Body Scripts Settings

Pre-request

Post-response

```
if (pm.response.code === 200) {
    const json = pm.response.json();
    pm.environment.set("token", json.data.token);
    pm.test("Token stored from login", () => {
        pm.expect(pm.environment.get("token")).to.not.be.null;
    });
}
```

For the logout:

Authorization Bearer {{token}}

POST {{base_url}} /logout

Docs Params Authorization Headers 11 Body Scripts Settings

Pre-request

Post-response

```
if (pm.response.code === 200) {
    pm.environment.unset("token");
    pm.test("Token cleared on logout", () => {
        pm.expect(pm.environment.get("token")).to.be.undefined;
    });
}
```

It will automatically get the token from the Beared {{token}}, and delete it.

Now I will test it:
Register:

```
{
    "name": "Jad Ozeir",
    "email": "jadozeir@gmail.com",
    "password": "password123",
    "password_confirmation": "password123",
    "phone": "12345678"
```

Cookies Headers 9 Test Results 1/1

JSON Preview Visualize

```
{
    "success": true,
    "message": "Registration successful.",
    "data": {
        "user": {
            "id": 4,
            "name": "Jad Ozeir",
            "email": "jadozeir@gmail.com",
            "phone": "12345678",
            "role": "user"
        },
```

Login:

```
{
    "email": "jadozeir@gmail.com",
    "password": "password123"
```

Cookies Headers 9 Test Results 1/1 200 OK 434 ms

JSON Preview Visualize

```
{
    "success": true,
    "message": "Login successful.",
    "data": {
        "user": {
            "id": 4,
            "name": "Jad Ozeir",
            "email": "jadozeir@gmail.com",
            "phone": "12345678",
            "role": "user"
        },
        "token": "10|VqTJyzHBN3k5A3PDCk9lNtgXBWPoAFkS5aca7XJu06dfd46b"
    }
```

Logout:

Body Cookies Headers 10 Test Results 1/1

{} JSON Preview Visualize

```
{
    "success": true,
    "message": "Logged out successfully."
}
```

I created 2 requests in Auth folder, getProfile and changeProfile, and it takes token in the variables.

I created 3 requests in the Flights folder, that doesn't need token.

- POST getProfile
- PUT changeProfile
- Flights
  - GET listFlights
  - GET getOneFlight
  - GET searchFlights

These 3 requests are done only by the admin, and requires token:

POST addFlights
PUT updateFlight
DEL deleteFlight

Since all requests in the Bookings and Passengers folder requires token, I will add it into the authorization of the whole folder like this:

BabiVoyage Local > **Bookings**

Overview | Authorization | Scripts

This authorization method will be used for every request in this folder. You can override this by specifying one in the request.

Auth Type

Bearer Token

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

Token {{token}}

Bookings
- GET myBookings
- POST createBooking
- GET getOneBooking
- PATCH cancelBooking

Passengers
- GET listPassengers
- POST addPassengers
- DEL deletePassenger

I will add a sample of the json code:

Create Flights (Admin):

POST {{base_url}} /admin/flights

Docs | Params | Authorization | Headers 11 | Body | Scr

none | form-data | x-www-form-urlencoded | raw

```
{
  "flight_number": "HF204",
  "airline_id": 1,
  "origin_airport_id": 1,
  "destination_airport_id": 2,
  "departure_time": "2026-05-01 08:00:00",
  "arrival_time": "2026-05-01 10:30:00",
  "price": 150.00,
  "total_seats": 120,
  "available_seats": 120,
  "status": "scheduled"
}
```

Create Booking:

POST {{base_url}} /bookings

Docs Params Authorization Headers 1

none form-data x-www-form-urle

```
{
    "flight_id": 1,
    "seat_count": 2
}
```

Add Passenger:

POST {{base_url}} /bookings/{id}/passengers

Docs Params Authorization Headers 11 Body

none form-data x-www-form-urlencoded

```
{
    "first_name": "Mediane",
    "last_name": "Ozeir",
    "passport_number": "A123456",
    "nationality": "Lebanese",
    "date_of_birth": "2002-05-15"
}
```

WebInter
nshpPro...

Full CRUD API

- Phase 1: Flight CRUD (FlightController + Form Requests + Routes)
- Phase 2: Booking CRUD (BookingController + Form Requests + Routes)
- Phase 3: Passenger CRUD (PassengerController + Form Requests + Routes)
- Phase 4: Update Postman (Test everything)

Phase 1: Flight CRUD

First, I will create the FlightController:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:controller API/FlightController

  INFO  Controller [C:\wamp64\www\webInternshipProj\app\Http\Controllers\API\Fl
ightController.php] created successfully.
```

Flight Form Request:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:request Flight/StoreFlightRequest
php artisan make:request Flight/UpdateFlightRequest

  INFO  Request [C:\wamp64\www\webInternshipProj\app\Http\Requests\Flight\Store
FlightRequest.php] created successfully.


  INFO  Request [C:\wamp64\www\webInternshipProj\app\Http\Requests\Flight\Updat
eFlightRequest.php] created successfully.
```

In the **StoreFlightRequest:**

Create Flights:

```php
public function authorize(): bool
{
    return auth()->check() && auth()->user()->role === 'admin'; //only admins can create flights
}


public function rules(): array
{
    return [
        'flight_number'          => ['required', 'string', 'max:20', 'unique:flights,flight_number'],
        'airline_id'             => ['required', 'integer', 'exists:airlines,id'],
        'origin_airport_id'      => ['required', 'integer', 'exists:airports,id'],
        'destination_airport_id' => ['required', 'integer', 'exists:airports,id', 'different:origin_airport_id'],
        'departure_time'         => ['required', 'date', 'after:now'],
        'arrival_time'           => ['required', 'date', 'after:departure_time'],
        'price'                  => ['required', 'numeric', 'min:0'],
        'total_seats'            => ['required', 'integer', 'min:1'],
        'available_seats'        => ['required', 'integer', 'min:0', 'lte:total_seats'],
        'status'                 => ['required', 'in:scheduled,delayed,cancelled,completed'],
    ];
}
```

```php
public function messages(): array
{
    return [
        'flight_number.unique'               => 'This flight number already exists.',
        'destination_airport_id.different'   => 'Destination must be different from origin.',
        'departure_time.after'               => 'Departure time must be in the future.',
        'arrival_time.after'                 => 'Arrival time must be after departure time.',
        'available_seats.lte'                => 'Available seats cannot exceed total seats.',
        'status.in'                          => 'Status must be: scheduled, delayed, cancelled, or completed.',
    ];
}
```

UpdateFlightRequest:

```php
public function authorize(): bool
{
    return auth()->check() && auth()->user()->role === 'admin'; //only admin can update flights
}


public function rules(): array
{
    $flightId = $this->route('id'); //updating specific flight

    return [
        'flight_number'          => ['sometimes', 'string', 'max:20', Rule::unique('flights', 'flight_number')->ignore($flightId)],
        'airline_id'             => ['sometimes', 'integer', 'exists:airlines,id'],
        'origin_airport_id'      => ['sometimes', 'integer', 'exists:airports,id'],
        'destination_airport_id' => ['sometimes', 'integer', 'exists:airports,id', 'different:origin_airport_id'],
        'departure_time'         => ['sometimes', 'date'],
        'arrival_time'           => ['sometimes', 'date', 'after:departure_time'],
        'price'                  => ['sometimes', 'numeric', 'min:0'],
        'total_seats'            => ['sometimes', 'integer', 'min:1'],
        'available_seats'        => ['sometimes', 'integer', 'min:0'],
        'status'                 => ['sometimes', 'in:scheduled,delayed,cancelled,completed'],
    ];
}
```

Why sometimes? On update, the admin might only want to change the price. sometimes means: only validate this field if it's present in the request.

Now I will create the **FlightController**.
It contains: list, search, get specific, create, update and delete.

1- List Flights {index}:

```php
// ----------------------------------------------
// PUBLIC: GET /api/flights
// List all available (scheduled) flights
// ----------------------------------------------
public function index()
{
    $flights = Flight::with(['airline', 'originAirport', 'destinationAirport'])
        ->where('status', 'scheduled')
        ->get();

    return response()->json([
        'success' => true,
        'data'    => $flights,
    ], 200);
}
```

2- Search Fights {search}:
(Flights can be filtered with: Origin, Destination, Departure time, Airport code or Price)

```php
public function search(Request $request)
{
    $request->validate([
        'origin'      => ['nullable', 'string'],
        'destination' => ['nullable', 'string'],
        'date'        => ['nullable', 'date'],
        'min_price'   => ['nullable', 'numeric', 'min:0'],
        'max_price'   => ['nullable', 'numeric', 'min:0'],
    ]);

    $query = Flight::with(['airline', 'originAirport', 'destinationAirport'])
        ->where('status', 'scheduled');

    // Filter by origin city or airport code
    if ($request->filled('origin')) {
        $query->whereHas('originAirport', function ($q) use ($request) {
            $q->where('city', 'like', '%' . $request->origin . '%')
              ->orWhere('code', 'like', '%' . $request->origin . '%');
        });
    }

    // Filter by destination city or airport code
    if ($request->filled('destination')) {
        $query->whereHas('destinationAirport', function ($q) use ($request) {
            $q->where('city', 'like', '%' . $request->destination . '%')
              ->orWhere('code', 'like', '%' . $request->destination . '%');
        });
    }

    // Filter by departure date
    if ($request->filled('date')) {
        $query->whereDate('departure_time', $request->date);
    }
```

```php
// Filter by price range
if ($request->filled('min_price')) {
    $query->where('price', '>=', $request->min_price);
}

if ($request->filled('max_price')) {
    $query->where('price', '<=', $request->max_price);
}

$flights = $query->get();

return response()->json([
    'success' => true,
    'data'    => $flights,
], 200);
```

3- Get a specific flight {show}:

```php
// ---------------------------------------------
// PUBLIC: GET /api/flights/{id}
// Get a specific flight
// ---------------------------------------------
public function show($id)
{
    $flight = Flight::with(['airline', 'originAirport', 'destinationAirport'])
        ->find($id);

    if (!$flight) {
        return response()->json([
            'success' => false,
            'message' => 'Flight not found.',
        ], 404);
    }

    return response()->json([
        'success' => true,
        'data'    => $flight,
    ], 200);
}
```

4- Create a new Flight {store}:

```php
// ---------------------------------------------
// ADMIN: POST /api/admin/flights
// Create a new flight
// ---------------------------------------------
public function store(StoreFlightRequest $request)
{
    $flight = Flight::create($request->validated());

    return response()->json([
        'success' => true,
        'message' => 'Flight created successfully.',
        'data'    => $flight->load(['airline', 'originAirport', 'destinationAirport']),
    ], 201);
}
```

The work is simplified and well-organized because of the request forms.

5- Update a Flight {update}:

```php
// ---------------------------------------------
// ADMIN: PUT /api/admin/flights/{id}
// Update a flight
// ---------------------------------------------
public function update(UpdateFlightRequest $request, $id)
{
    $flight = Flight::find($id);

    if (!$flight) {
        return response()->json([
            'success' => false,
            'message' => 'Flight not found.',
        ], 404);
    }

    $flight->update($request->validated());

    return response()->json([
        'success' => true,
        'message' => 'Flight updated successfully.',
        'data'    => $flight->load(['airline', 'originAirport', 'destinationAirport']),
    ], 200);
}
```

6- Delete a Flight {destroy}:

```php
// ---------------------------------------------
// ADMIN: DELETE /api/admin/flights/{id}
// Delete a flight
// ---------------------------------------------
public function destroy($id)
{
    $flight = Flight::find($id);

    if (!$flight) {
        return response()->json([
            'success' => false,
            'message' => 'Flight not found.',
        ], 404);
    }

    $flight->delete();

    return response()->json([
        'success' => true,
        'message' => 'Flight deleted successfully.',
    ], 200);
}
```

I've finished this part for now:

- Flights
  - GET listFlights
  - GET getOneFlight
  - GET searchFlights
  - POST addFlights
  - PUT updateFlight
  - DEL deleteFlight

Before creating the booking and passenger controllers,
I want to create the routes for the FlightController functions:

```php
// Public flight browsing (anyone can browse)
Route::get('/flights',        [FlightController::class, 'index']);
Route::get('/flights/search', [FlightController::class, 'search']);
Route::get('/flights/{id}',   [FlightController::class, 'show']);

// -----------------------------------------------------
// Protected Routes (token required)
// -----------------------------------------------------

Route::middleware('auth:sanctum')->group(function () {

    // Auth
    Route::post('/logout', [AuthController::class, 'logout']);

    // Admin: Flight management
    Route::post('/admin/flights',        [FlightController::class, 'store']);
    Route::put('/admin/flights/{id}',    [FlightController::class, 'update']);
    Route::delete('/admin/flights/{id}', [FlightController::class, 'destroy']);
```

Now I will check all functions in postman, to make sure everything is running successfully.

For the data of Airlines and Airports:
I will manually use Database Seeders.

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:seeder AirlineSeeder

  INFO  Seeder [C:\wamp64\www\webInternshipProj\database\seeders\AirlineSeeder.
php] created successfully.


Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:seeder AirportSeeder

  INFO  Seeder [C:\wamp64\www\webInternshipProj\database\seeders\AirportSeeder.
php] created successfully.
```

database/seeders/AirlineSeeder:

```php
class AirlineSeeder extends Seeder
{

    public function run(): void
    {
        Airline::insert([
            ['name' => "Air Côte d'Ivoire",        'code' => 'HF', 'logo_url' => null],
            ['name' => 'Asky Airlines',            'code' => 'KP', 'logo_url' => null],
            ['name' => 'Air Senegal',              'code' => 'HC', 'logo_url' => null],
            ['name' => 'Arik Air',                 'code' => 'W3', 'logo_url' => null],
            ['name' => 'Air Burkina',              'code' => '2J', 'logo_url' => null],
            ['name' => 'Ceiba Intercontinental',   'code' => 'C2', 'logo_url' => null],
            ['name' => 'Cronos Airlines',          'code' => 'C8', 'logo_url' => null],
        ]);
    }
```

AirportSeeder:

```php
class AirportSeeder extends Seeder
{

    public function run(): void
    {
        Airport::insert([
            [
                'name'    => "Félix Houphouët-Boigny International Airport",
                'code'    => 'ABJ',
                'city'    => 'Abidjan',
                'country' => "Côte d'Ivoire",
            ],
            [
                'name'    => 'Kotoka International Airport',
                'code'    => 'ACC',
                'city'    => 'Accra',
                'country' => 'Ghana',
            ],
            [
                'name'    => 'Murtala Muhammed International Airport',
                'code'    => 'LOS',
                'city'    => 'Lagos',
                'country' => 'Nigeria',
            ],
            [
```

DatabaseSeeder:

```php
class DatabaseSeeder extends Seeder
{
    use WithoutModelEvents;

    /**
     * Seed the application's database.
     */
    public function run(): void
    {

        $this->call([
            AirlineSeeder::class,
            AirportSeeder::class,
        ]);
```

Database created successfully

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan db:seed

  INFO  Seeding database.

  Database\Seeders\AirlineSeeder ..................................... RUNNING
  Database\Seeders\AirlineSeeder ................................... 65 ms DONE

  Database\Seeders\AirportSeeder ..................................... RUNNING
  Database\Seeders\AirportSeeder ................................... 5 ms DONE
```

Now let's continue

TEST in Postman:

- Admin create a new flight:

```json
{
  "flight_number": "HF204",
  "airline_id": 1,
  "origin_airport_id": 1,
  "destination_airport_id": 2,
  "departure_time": "2026-05-01 08:00:00",
  "arrival_time": "2026-05-01 10:30:00",
  "price": 150.00,
  "total_seats": 120,
  "available_seats": 120,
  "status": "scheduled"
}
```

It gives me:

Body Cookies Headers 10 Test Results

{} JSON Preview Pass the correct auth credentials


```
{
    "message": "Unauthenticated."
}
```


Unauthenticated because I tried to create a new flight as a user not admin.
I changed the role manually from user to admin and logged-in again.

Now:


```
"success": true,
"message": "Login successful.",
"data": {
    "user": {
        "id": 1,
        "name": "Mediane Ozeir",
        "email": "mediane@gmail.com",
        "phone": "12345678",
        "role": "admin"
```



```
"success": true,
"message": "Flight created successfully.",
"data": {
    "flight_number": "HF204",
    "airline_id": 1,
    "origin_airport_id": 1,
    "destination_airport_id": 2,
    "departure_time": "2026-05-01T08:00:00.000000Z",
    "arrival_time": "2026-05-01T10:30:00.000000Z",
    "price": "150.00",
    "total_seats": 120,
    "available_seats": 120,
    "status": "scheduled",
    "updated_at": "2026-04-29T10:54:02.000000Z",
    "created_at": "2026-04-29T10:54:02.000000Z",
    "id": 2,
    "airline": {
        "id": 1,
        "name": "Air Côte d'Ivoire",
        "code": "HF",
        "logo_url": null,
        "created_at": null,
        "updated_at": null
    },
    "origin_airport": {
        "id": 1,
        "name": "Félix Houphouët-Boigny International Airport",
        "code": "ABJ",
        "city": "Abidjan",
        "country": "Côte d'Ivoire",
        "created_at": null,
        "updated_at": null
    },
    "destination_airport": {
        "id": 2,
        "name": "Kotoka International Airport",
        "code": "ACC",
        "city": "Accra",
```


Flight added successfully


```
    "price": 145.00,
    "total_seats": 100,
    "available_seats": 85,
    "status": "nothing"
}
```


dy Cookies Headers 10 Test Results 422 Unprocessable Content

JSON Preview Debug with AI


```
{
    "message": "Status must be: scheduled, delayed, cancelled, or completed.",
    "errors": {
        "status": [
```

It gives me error, because the status is wrong, as shown in the message.

Update Flight:
(I can send only the data that I want to change, not necessary all data as mentioned before)
Also I need to put the flight_id in the url

PUT {{base_url}} /admin/flights/1

Docs Params Authorization Headers 11 Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQl

```
{
  "price": 135.00,
  "available_seats": 110
}
```

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize

```
{
    "success": true,
    "message": "Flight updated successfully.",
    "data": {
        "id": 1,
        "flight_number": "HF204",
        "airline_id": 1,
        "origin_airport_id": 1,
        "destination_airport_id": 2,
        "departure_time": "2026-05-01T08:00:00.000000Z",
        "arrival_time": "2026-05-01T10:30:00.000000Z",
        "price": "135.00",
        "total_seats": 120,
        "available_seats": 110,
```

If I put the price as string it gives me error:

```
{
  "price": "abc",
  "available_seats": 110
}
```

Cookies Headers 10 Test Results 42

Preview Debug with AI

```
{
    "message": "The price field must be a number.",
    "errors": {
```

Now delete this flight:
Also flight_id in the url:

DELETE {{base_url}} /admin/flights/1

Docs Params Authorization Headers 9 Body Scripts

none form-data x-www-form-urlencoded raw

```
1
```

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize

```
{
    "success": true,
    "message": "Flight deleted successfully."
}
```

List Flights: (No JSON code needed)

Body Cookies Headers 9 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "data": [
        {
            "id": 3,
            "flight_number": "HF204",
            "airline_id": 1,
            "origin_airport_id": 1,
            "destination_airport_id": 2,
            "departure_time": "2026-05-01T08:00:00.000000Z",
            "arrival_time": "2026-05-01T10:30:00.000000Z",
            "price": "150.00",
            "total_seats": 120,
            "available_seats": 120,
            "status": "scheduled",
            "created_at": "2026-04-29T11:07:04.000000Z",
```


Body Cookies Headers 9 Test Results

{} JSON Preview Visualize


```
        },
        {
            "id": 4,
            "flight_number": "EF306",
            "airline_id": 2,
            "origin_airport_id": 2,
            "destination_airport_id": 1,
            "departure_time": "2026-05-04T08:00:00.000000Z",
            "arrival_time": "2026-05-04T10:30:00.000000Z",
            "price": "145.00",
            "total_seats": 100,
            "available_seats": 85,
            "status": "scheduled",
            "created_at": "2026-04-29T11:08:04.000000Z",
            "updated_at": "2026-04-29T11:08:04.000000Z",
            "airline": {
```


Get one Flight, by id:

GET {{base_url}} /flights/4

Docs Params Authorization Headers 8 Body Scripts Settings

none form-data x-www-form-urlencoded raw binary

This request does

Body Cookies Headers 9 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "data": {
        "id": 4,
        "flight_number": "EF306",
        "airline_id": 2,
        "origin_airport_id": 2,
        "destination_airport_id": 1,
        "departure_time": "2026-05-04T08:00:00.000000Z",
        "arrival_time": "2026-05-04T10:30:00.000000Z",
        "price": "145.00",
        "total_seats": 100,
        "available_seats": 85,
```


Search Flights:

There is 2 methods to get data:

1- Put the parameters in the Param tab in postman

2- Put the parameters manually in the url

Method 1:

Query Params

Key	Value
✓ origin	ACC
Key	Value

Body Cookies Headers 9 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "data": [
        {
            "id": 4,
            "flight_number": "EF306",
            "airline_id": 2,
            "origin_airport_id": 2,
            "destination_airport_id": 1,
            "departure_time": "2026-05-04T08:00:00.000000Z",
            "arrival_time": "2026-05-04T10:30:00.000000Z",
```


Method 2:

GET {{base_url}} /flights/search?origin=ACC

Docs Params Authorization Headers 8 Body Scripts Settings

none form-data x-www-form-urlencoded raw binary Gr

This request does not hav

Body Cookies Headers 9 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "data": [
        {
            "id": 4,
            "flight_number": "EF306",
            "airline_id": 2,
            "origin_airport_id": 2,
            "destination_airport_id": 1,
            "departure_time": "2026-05-04T08:00:00.000000Z",
            "arrival_time": "2026-05-04T10:30:00.000000Z",
            "price": "145.00",
```


Method 1 is better and more professional.

For now, all these requests are working successfully:

- Auth
  - POST register
  - POST logout
  - POST login
  - POST getProfile
  - PUT changeProfile
- Flights
  - GET listFlights
  - GET getOneFlight
  - GET searchFlights
  - POST addFlights
  - PUT updateFlight
  - DEL deleteFlight

Phase 2: Booking CRUD

First I create the BookingController and the StoreBookingRequest

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:controller API/BookingController

  INFO  Controller [C:\wamp64\www\webInternshipProj\app\Http\Controllers\API\Bo
okingController.php] created successfully.


Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:request Booking/StoreBookingRequest

  INFO  Request [C:\wamp64\www\webInternshipProj\app\Http\Requests\Booking\Stor
eBookingRequest.php] created successfully.
```

StoreBookingRequest:

```php
public function authorize(): bool
{
    return auth()->check(); //authenticated user can book a flight, not a guest
}


public function rules(): array
{
    return [
        'flight_id'  => ['required', 'integer', 'exists:flights,id'],
        'seat_count' => ['required', 'integer', 'min:1', 'max:10'],
    ];
}

public function messages(): array
{
    return [
        'flight_id.exists'   => 'The selected flight does not exist.',
        'seat_count.min'     => 'You must book at least 1 seat.',
        'seat_count.max'     => 'You cannot book more than 10 seats at once.',
    ];
}
```

BookingController:

1- Get all flights of a user {index}

```php
// ----------------------------------------------
// GET /api/bookings
// List all bookings for the authenticated user
// ----------------------------------------------
public function index()
{
    $bookings = Booking::with(['flight.airline', 'flight.originAirport', 'flight.destinationAirport', 'passengers'])
        ->where('user_id', auth()->id())
        ->latest()
        ->get();

    return response()->json([
        'success' => true,
        'data'    => $bookings,
    ], 200);
}
```

2- Create new booking {store}

```php
// POST /api/bookings
// Create a new booking
// ----------------------------------------------
public function store(StoreBookingRequest $request)
{
    $flight = Flight::find($request->flight_id);

    // Check flight is available
    if ($flight->status !== 'scheduled') {
        return response()->json([
            'success' => false,
            'message' => 'This flight is not available for booking.',
        ], 422);
    }

    // Check enough seats available
    if ($flight->available_seats < $request->seat_count) {
        return response()->json([
            'success' => false,
            'message' => 'Not enough seats available. Only ' . $flight->available_seats . ' seats left.',
        ], 422);
    }

    // Calculate total price
    $totalPrice = $flight->price * $request->seat_count;
```

```php
    // Create the booking
    $booking = Booking::create([
        'user_id'           => auth()->id(),
        'flight_id'         => $flight->id,
        'booking_reference' => 'BV-' . strtoupper(Str::random(8)),
        'seat_count'        => $request->seat_count,
        'total_price'       => $totalPrice,
        'status'            => 'confirmed',
    ]);

    // Reduce available seats on the flight
    $flight->decrement('available_seats', $request->seat_count);

    return response()->json([
        'success' => true,
        'message' => 'Booking created successfully.',
        'data'    => $booking->load(['flight.airline', 'flight.originAirport', 'flight.destinationAirport']),
    ], 201);
}
```

3- Get specific booking {show}:
only owner of the booking can see it.

```php
// Get a specific booking (only owner can see it)
// ----------------------------------------------
public function show($id)
{
    $booking = Booking::with(['flight.airline', 'flight.originAirport', 'flight.destinationAirport', 'passengers'])
        ->find($id);

    if (!$booking) {
        return response()->json([
            'success' => false,
            'message' => 'Booking not found.',
        ], 404);
    }

    // Authorization: only the owner can view their booking
    if ($booking->user_id !== auth()->id()) {
        return response()->json([
            'success' => false,
            'message' => 'You are not authorized to view this booking.',
        ], 403);
    }

    return response()->json([
        'success' => true,
        'data'    => $booking,
    ], 200);
}
```

4- Cancel a booking

```php
// PATCH /api/bookings/{id}/cancel
// Cancel a booking
// ----------------------------------------------
public function cancel($id)
{
    $booking = Booking::find($id);

    if (!$booking) {
        return response()->json([
            'success' => false,
            'message' => 'Booking not found.',
        ], 404);
    }

    // Authorization check
    if ($booking->user_id !== auth()->id()) {
        return response()->json([
            'success' => false,
            'message' => 'You are not authorized to cancel this booking.',
        ], 403);
    }

    // Check it's not already cancelled
    if ($booking->status === 'cancelled') {
        return response()->json([
            'success' => false,
            'message' => 'This booking is already cancelled.',
        ], 422);
    }
```

Now I will add the routes for the booking functions, and then test it in Postman

```
// Bookings
Route::get('/bookings',              [BookingController::class, 'index']);
Route::post('/bookings',             [BookingController::class, 'store']);
Route::get('/bookings/{id}',         [BookingController::class, 'show']);
Route::patch('/bookings/{id}/cancel', [BookingController::class, 'cancel']);
```

Testing:

```
1 {
2     "flight_id": 5,
3     "seat_count": 3
4 }
```

Body Cookies Headers 10 Test Results 422

{} JSON Preview Debug with AI

```
1 {
2     "message": "The selected flight does not exist.",
3     "errors": {
4         "flight_id": [
5             "The selected flight does not exist."
6         ]
7     }
8 }
```

It gives me error because there is no flight with id 5.

Booking created successfully

```
1 {
2     "flight_id": 4,
3     "seat_count": 3
4 }
```

ly Cookies Headers 10 Test Results

JSON Preview Visualize

```
1 {
2     "success": true,
3     "message": "Booking created successfully.",
4     "data": {
5         "user_id": 1,
6         "flight_id": 4,
7         "booking_reference": "BV-SVIP93UP",
8         "seat_count": 3,
9         "total_price": "405.00",
10        "status": "confirmed",
11        "updated_at": "2026-04-29T11:59:50.000000Z",
12        "created_at": "2026-04-29T11:59:50.000000Z",
```

Listing the booking of an authenticated user:

GET {{base_url}} /bookings

Docs Params Authorization Headers 9 Body Scripts

none form-data x-www-form-urlencoded raw

```
1
```

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize

```
1 {
2     "success": true,
3     "data": [
4         {
5             "id": 1,
6             "user_id": 1,
7             "flight_id": 4,
8             "booking_reference": "BV-SVIP93UP",
9             "seat_count": 3,
10            "total_price": "405.00",
```

Get one booking with id:2

GET {{base_url}} /bookings/2

Docs Params Authorization Headers 9 Body Scripts Setting

none form-data x-www-form-urlencoded raw bin

```
1
```

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
1  {
2      "success": true,
3      "data": {
4          "id": 2,
5          "user_id": 1,
6          "flight_id": 3,
7          "booking_reference": "BV-TUZP8CZ3",
8          "seat_count": 3,
9          "total_price": "450.00",
10         "status": "confirmed",
11         "created_at": "2026-04-29T12:03:47.000000Z",
```


Cancel a booking: /2/cancel

PATCH {{base_url}} /bookings/2/cancel

Docs Params Authorization Headers 10 Body Scripts Setting

none form-data x-www-form-urlencoded raw binar

This request do

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
1  {
2      "success": true,
3      "message": "Booking cancelled successfully.",
4      "data": {
5          "id": 2,
6          "user_id": 1,
7          "flight_id": 3,
8          "booking_reference": "BV-TUZP8CZ3",
```


Phase 3: Passenger CRUD

Same steps:

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:controller API/PassengerController

  INFO  Controller [C:\wamp64\www\webInternshipProj\app\Http\Controllers\API\Pa
ssengerController.php] created successfully.


Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:request Passenger/StorePassengerRequest

  INFO  Request [C:\wamp64\www\webInternshipProj\app\Http\Requests\Passenger\St
orePassengerRequest.php] created successfully.
```

StorePassengerRequest:

```php
public function authorize(): bool
{
    return auth()->check();
}


public function rules(): array
{
    return [
        'first_name'      => ['required', 'string', 'max:100'],
        'last_name'       => ['required', 'string', 'max:100'],
        'passport_number' => ['nullable', 'string', 'max:20'],
        'nationality'     => ['nullable', 'string', 'max:100'],
        'date_of_birth'   => ['nullable', 'date', 'before:today'],
    ];
}

public function messages(): array
{
    return [
        'first_name.required'  => 'First name is required.',
        'last_name.required'   => 'Last name is required.',
        'date_of_birth.before' => 'Date of birth must be in the past.',
    ];
}
```

PassengerController:

1- List all Passengers for a booking:

```php
// ----------------------------------------------
// GET /api/bookings/{id}/passengers
// List all passengers for a booking
// ----------------------------------------------
public function index($bookingId)
{
    $booking = Booking::find($bookingId);

    if (!$booking) {
        return response()->json([
            'success' => false,
            'message' => 'Booking not found.',
        ], 404);
    }

    // Only the booking owner can see passengers
    if ($booking->user_id !== auth()->id()) {
        return response()->json([
            'success' => false,
            'message' => 'You are not authorized to view passengers for this booking.',
        ], 403);
    }

    return response()->json([
        'success' => true,
        'data'    => $booking->passengers,
    ], 200);
}
```

2- Add new Passengers:

only owner of the booking can add passengers:

```php
// POST /api/bookings/{id}/passengers
// Add a passenger to a booking
// ----------------------------------------------
public function store(StorePassengerRequest $request, $bookingId)
{
    $booking = Booking::find($bookingId);

    if (!$booking) {
        return response()->json([
            'success' => false,
            'message' => 'Booking not found.',
        ], 404);
    }

    // Authorization check
    if ($booking->user_id !== auth()->id()) {
        return response()->json([
            'success' => false,
            'message' => 'You are not authorized to add passengers to this booking.',
        ], 403);
    }

    // Cannot add passengers to a cancelled booking
    if ($booking->status === 'cancelled') {
        return response()->json([
            'success' => false,
            'message' => 'Cannot add passengers to a cancelled booking.',
        ], 422);
    }
```

```php
    // Check passenger count does not exceed seat count
    $currentPassengers = $booking->passengers()->count();
    if ($currentPassengers >= $booking->seat_count) {
        return response()->json([
            'success' => false,
            'message' => 'Cannot add more passengers than booked seats (' . $booking->seat_count . ').',
        ], 422);
    }

    $passenger = Passenger::create([
        'booking_id'      => $booking->id,
        'first_name'      => $request->first_name,
        'last_name'       => $request->last_name,
        'passport_number' => $request->passport_number,
        'nationality'     => $request->nationality,
        'date_of_birth'   => $request->date_of_birth,
    ]);

    return response()->json([
        'success' => true,
        'message' => 'Passenger added successfully.',
        'data'    => $passenger,
    ], 201);
}
```

3- Delete Passenger:

```php
// ----------------------------------------------
// DELETE /api/bookings/{id}/passengers/{pid}
// Remove a passenger from a booking
// ----------------------------------------------
public function destroy($bookingId, $passengerId)
{
    $booking = Booking::find($bookingId);

    if (!$booking) {
        return response()->json([
            'success' => false,
            'message' => 'Booking not found.',
        ], 404);
    }

    if ($booking->user_id !== auth()->id()) {
        return response()->json([
            'success' => false,
            'message' => 'You are not authorized to remove passengers from this booking.',
        ], 403);
    }

    $passenger = Passenger::where('id', $passengerId)
        ->where('booking_id', $bookingId)
        ->first();

    if (!$passenger) {
        return response()->json([
            'success' => false,
```

```php
    if (!$passenger) {
        return response()->json([
            'success' => false,
            'message' => 'Passenger not found in this booking.',
        ], 404);
    }

    $passenger->delete();

    return response()->json([
        'success' => true,
        'message' => 'Passenger removed successfully.',
    ], 200);
}
```

The routes:

```php
// Passengers
Route::get('/bookings/{id}/passengers',              [PassengerController::class, 'index']);
Route::post('/bookings/{id}/passengers',             [PassengerController::class, 'store']);
Route::delete('/bookings/{id}/passengers/{pid}',     [PassengerController::class, 'destroy']);
```

Passenger requests **Testing**:

Error: First Name required

POST {{base_url}} /bookings/1/passengers

Docs Params Authorization Headers 11 Body

none form-data x-www-form-urlencoded

```
1 {
2
3     "last_name": "Ozeir",
4     "passport_number": "A123456",
5     "nationality": "Lebanese",
6     "date_of_birth": "2002-05-15"
```

Body Cookies Headers 10 Test Results

{} JSON Preview Debug with AI

```
1 {
2     "message": "First name is required.",
3     "errors": {
4         "first_name": [
5             "First name is required."
6         ]
7     }
8 }
```

The booking with id:1 , it contains 3 seat_counts.
I tried to add 4:

none form-data x-www-form-urlencoded raw binary GraphQL JSC

```
{
    "first_name": "Elissar",
    "last_name": "Ozeir",
    "passport_number": "A123789",
    "nationality": "Lebanese",
    "date_of_birth": "1996-05-13"
}
```

Body Cookies Headers 10 Test Results 422 Unprocessable C

{} JSON Preview Debug with AI

```
{
    "success": false,
    "message": "Cannot add more passengers than booked seats (3)."
}
```

Invalid date_of_birth:

```
    "nationality": "Lebanese",
    "date_of_birth": "1996-13-05"
}
```

Cookies Headers 10 Test Results 422 Unproces

Preview Debug with AI

```
{
    "message": "The date of birth field must be a valid date.",
    "errors": {
        "date_of_birth": [
            "The date of birth field must be a valid date."
        ]
    }
```

Passenger Deleted:

DELETE {{base_url}} /bookings/1/passengers/3

Docs Params Authorization Headers 9 Body Scripts Set

none form-data x-www-form-urlencoded raw b

```
1
```

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize

```
{
    "success": true,
    "message": "Passenger removed successfully."
}
```

Adding the saved_flights concept (many-to-many relationship):

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:controller API/SavedFlightController

  INFO  Controller [C:\wamp64\www\webInternshipProj\app\Http\Controllers\API\Sa
vedFlightController.php] created successfully.
```

SavedFlightController:

1- List all saved flights {index}

```php
// ------------------------------------------------
// GET /api/saved-flights
// List all saved flights for the authenticated user
// ------------------------------------------------
public function index(Request $request)
{
    $savedFlights = $request->user()···
    // If the user hasn't saved any flights yet
    if ($savedFlights->isEmpty()) {
        return response()->json([
            'success' => true,
            'message' => "You haven't saved any flight yet.",
            'data'    => [],
        ], 200);
    }

    // Normal response with the list
    return response()->json([
        'success' => true,
        'data'    => $savedFlights,
    ], 200);
}
```

2- Save a specific flight {save}

```php
// ------------------------------------------------
// POST /api/flights/{id}/save
// Save a specific flight
// ------------------------------------------------
public function save(Request $request, $id)
{
    $flight = Flight::with(['airline', 'originAirport', 'destinationAirport'])
                    ->find($id);

    if (!$flight) {
        return response()->json([
            'success' => false,
            'message' => 'Flight not found.',
        ], 404);
    }

    $request->user()->savedFlights()->syncWithoutDetaching([$flight->id]);

    return response()->json([
        'success' => true,
        'message' => 'Flight saved successfully.',
        'data'    => $flight,
    ], 200);
}
```

Why syncWithoutDetaching?
It adds the flight if not already saved, but doesn't remove others. Thanks to the unique(user_id, flight_id) constraint, we can never insert duplicates.

3- Unsave a specific flight

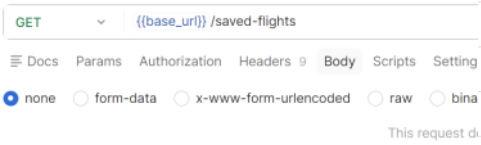
```php
// ------------------------------------------------
// DELETE /api/flights/{id}/unsave
// Delete a specific flight
// ------------------------------------------------
public function unsave(Request $request, $id)
{
    $flight = Flight::find($id);

    if (!$flight) {
        return response()->json([
            'success' => false,
            'message' => 'Flight not found.',
        ], 404);
    }

    $request->user()->savedFlights()->detach($flight->id);

    return response()->json([
        'success' => true,
        'message' => 'Flight removed from saved list.',
    ], 200);
}
```

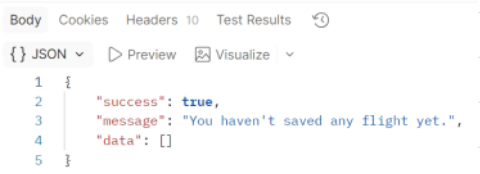
Routes:

```php
// Saved flights
Route::get('/saved-flights',                [SavedFlightController::class, 'index']);
Route::post('/flights/{id}/save',           [SavedFlightController::class, 'save']);
Route::delete('/flights/{id}/unsave',       [SavedFlightController::class, 'unsave']);
```

Testing:

- List Flights:





- Save Flight:

```php
public function save(Request $request, $id)
{
    $flight = Flight::with(['airline', 'originAirport', 'destinationAirport'])
                    ->find($id);

    if (!$flight) {
        return response()->json([
            'success' => false,
            'message' => 'Flight not found.',
        ], 404);
    }

    $user = $request->user();
    $alreadySaved = $user->savedFlights()->where('flight_id', $flight->id)->exists();
    if ($alreadySaved) {
        return response()->json([
            'success' => true,
            'message' => 'Flight is already in your saved list.',
            'data'    => $flight,
        ], 200);
    }

    $user->savedFlights()->attach($flight->id);// Not saved yet → attach it

    return response()->json([
        'success' => true,
        'message' => 'Flight saved successfully.',
        'data'    => $flight,
    ], 200);
}
```

When I save the same flight another time, it gives me:

POST {{base_url}} /flights/1/save

Docs Params Authorization Headers 10 Body Scripts Settings

none form-data x-www-form-urlencoded raw binary

This request does

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "message": "Flight is already in your saved list.",
    "data": {
        "id": 1,
        "flight_number": "HF204",
        "airline_id": 1,
        "origin_airport_id": 1,
```


A user can save many flights:

POST {{base_url}} /flights/2/save

Docs Params Authorization Headers 10 Body Scripts S

none form-data x-www-form-urlencoded raw

This requ

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "message": "Flight saved successfully.",
    "data": {
        "id": 2,
        "flight_number": "EF306",
        "airline_id": 2,
        "origin_airport_id": 2,
```


This user has booked 2 flights, with id 1 and 2.

I will login as another user, and try to save these 2 same flights.

```
UPDATE `saved_flights` SET `id` = '4' WHERE `saved_flights
```

Show all | Number of rows: 25 Filter rows: Searc

Extra options

				id	user_id	flight_id	create
Edit	Copy	Delete		1	1	1	2026-0
Edit	Copy	Delete		2	1	2	2026-0
Edit	Copy	Delete		3	3	1	2026-0
Edit	Copy	Delete		4	3	2	2026-0

This photo of the saved_flights database shows that:
The user with id:1, saved 2 flights (id 1 and 2)
Also the user with id:3, saved the same 2 flights (id 1 and 2)

It shows clearly the many-to-many relationship between the 2 tables: user and flights.

Now I will create the ProfileController and the requests (getProfile, updateProfile)

```
Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:controller API/ProfileController

  INFO  Controller [C:\wamp64\www\webInternshipProj\app\Http\Controllers\API\Pr
ofileController.php] created successfully.


Administrator@Mediane MINGW64 /c/wamp64/www/webInternshipProj (develop)
$ php artisan make:request Profile/UpdateProfileRequest

  INFO  Request [C:\wamp64\www\webInternshipProj\app\Http\Requests\Profile\Upda
teProfileRequest.php] created successfully.
```

UpdateProfileRequest:

```php
public function authorize(): bool
{
    return auth()->check();
}


public function rules(): array
{
    return [
        'name'     => ['sometimes', 'string', 'min:1', 'max:255'],
        'phone'    => ['nullable', 'string', 'max:14'],
        'password' => ['nullable', 'string', 'min:8', 'confirmed'],
    ];
}

public function messages(): array
{
    return [
        'name.required'         => 'Your name cannot be empty.',
        'name.max'              => 'Name must not exceed 255 characters.',
        'phone.max'             => 'Phone number is too long.',
        'password.min'          => 'New password must be at least 8 characters.',
        'password.confirmed'    => 'Password confirmation does not match.',
    ];
}
```

ProfileController:

1- Get the Profile {show}

```php
// ----------------------------------------------
// GET /api/profile
// Get profile of the authenticated user
// ----------------------------------------------
public function show(Request $request)
{
    return response()->json([
        'success' => true,
        'data'    => $request->user(),
    ], 200);
}
```

2- Change the Profile {update}

```php
// ----------------------------------------------
// PUT /api/profile
// Change the profile
// ----------------------------------------------
public function update(UpdateProfileRequest $request)
{
    $user = $request->user();

    // Collect only the fields we allow to update
    $data = $request->only(['name', 'phone']);

    // If a new password is provided, hash it automatically via User model cast
    if ($request->filled('password')) {
        $data['password'] = $request->password;
    }

    $user->update($data);

    return response()->json([
        'success' => true,
        'message' => 'Profile updated successfully.',
        'data'    => $user,
    ], 200);
}
```

Routes:

```php
// Profile
Route::get('/profile', [ProfileController::class, 'show']);
Route::put('/profile', [ProfileController::class, 'update']);
```

Testing:

Get Profile:

GET {{base_url}} /profile

Docs Params Authorization Headers 10 Body Scripts Setting

none form-data x-www-form-urlencoded raw binar

This request d

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "data": {
        "id": 3,
        "name": "Jad Ozeir",
        "email": "jadozeir@gmail.com",
        "phone": "12345678",
        "role": "user",
        "email_verified_at": null,
        "created_at": "2026-04-26T10:49:06.000000Z",
        "updated_at": "2026-04-26T10:49:06.000000Z"
    }
```


Another user:

GET {{base_url}} /profile

Docs Params Authorization Headers 10 Body Scripts

none form-data x-www-form-urlencoded raw

This re

Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "data": {
        "id": 1,
        "name": "Mediane Ozeir",
        "email": "mediane@gmail.com",
        "phone": "12345678",
        "role": "admin",
        "email_verified_at": null,
```


Change Profile:

PUT {{base_url}} /profile

Docs Params Authorization Headers 11 Body Scripts

none form-data x-www-form-urlencoded raw


```
{
  "name": "Mediane Ghassan Ozeir"
}
```


Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "message": "Profile updated successfully.",
    "data": {
        "id": 1,
        "name": "Mediane Ghassan Ozeir",
        "email": "mediane@gmail.com",
        "phone": "12345678",
```


I haven't change the phone, it remains the same

Changing only the phone number:

PUT {{base_url}} /profile

Docs Params Authorization Headers 11 Body Scripts

none form-data x-www-form-urlencoded raw


```
{
  "phone": "12345678"
}
```


Body Cookies Headers 10 Test Results

{} JSON Preview Visualize


```
{
    "success": true,
    "message": "Profile updated successfully.",
    "data": {
        "id": 1,
        "name": "Mediane Ghassan Ozeir",
```


We can't leave the name empty:

```
{
  "name": "",
  "phone": "12345678"
}
```

Cookies Headers 10 Test Results 422 Unprocessable

ON Preview Debug with AI

```
{
    "message": "The name field must be a string. (and 1 more error)",
    "errors": {
        "name": [
            "The name field must be a string.",
            "The name field must be at least 1 characters."
        ]
    }
}
```

Filament Admin Panel

This part continue after finishing the RESTful API part of the BabiVoyage project.

So I am adding the bonus admin panel using Filament v3.

The goal of this bonus task is to keep the existing API working exactly as before, and add a browser-based dashboard for admins at: /admin

The API is still the main backend of the project, but the admin panel gives the admin a visual interface to manage flights, airlines, airports, users, and bookings.

Important Rule Before Starting

Before implementing the bonus, I need to make sure I do not break the API work that is already finished.

So I do not modify:

- app/Http/Controllers/API/
- app/Http/Requests/
- routes/api.php
- existing migration files

This is important because the API controllers, validation requests, API routes, and database structure are already working and already tested.

The admin panel is added beside the API, not instead of it.

Existing Project State

The project is a Laravel 12 backend API called BabiVoyage.

It already contains:

- authentication with Laravel Sanctum
- users table with role field
- flight search and CRUD API
- booking API
- passenger API
- saved flights API
- profile API

- Postman collection
- feature tests

The project already has these main models:
- User
- Airline
- Airport
- Flight
- Booking
- Passenger

The relationships are already defined in the models, so Filament can use them directly.

Bonus Feature Overview

For the bonus task, I added a Filament v3 admin panel.

The admin panel contains:
1- A login page for admins
2- Dashboard statistics
3- Flight management
4- Booking monitoring
5- Booking cancellation action
6- Passenger list inside each booking
7- Airline management
8- Airport management
9- User management

Only users with: role = admin can access the Filament panel.

Normal users can still use the API, but they cannot open the admin dashboard.

Installing Filament

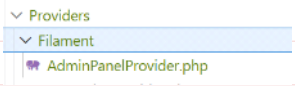
First, I installed Filament v3 with Composer.

The package added to composer.json is: "filament/filament": "^3.0"

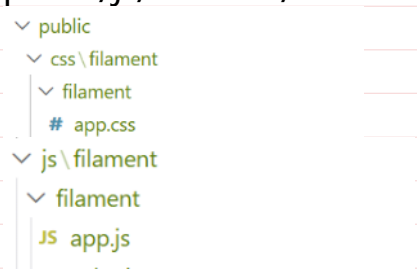
The installed version resolved to: filament/filament v3.3.52

Then I ran the Filament panel installer:
C:\wamp64\bin\php\php8.4.0\php.exe artisan filament:install --panels --no-interaction

This created the Filament admin provider:
app/Providers/Filament/AdminPanelProvider.php



It also published Filament frontend assets into:
public/css/filament/
public/js/filament/



Small Problem During Installation

At first, the normal php command was using PHP 8.2.

The project can run tests with PHP 8.2, but the installed Laravel vendor configuration was expecting PHP 8.4's newer PDO MySQL class in one place.

Because WAMP already has PHP 8.4 installed, I used this PHP binary for Filament and Artisan commands.

Admin Panel Provider

I adjust AdminPanelProvider to match the project requirements.

```php
class AdminPanelProvider extends PanelProvider
{
    public function panel(Panel $panel): Panel
    {
        return $panel
            ->default()
            ->id('admin')
            ->path('admin')
            ->authGuard('web')
            ->login()
            ->colors([
                'primary' => Color::Blue,
            ])
```

This means:

- the panel ID is admin
- the panel URL is /admin

- Filament uses the web guard
- Filament provides its own login screen

I also changed the primary color to blue

Then I registered all admin resources manually:

```
->resources([
    FlightResource::class,
    BookingResource::class,
    AirlineResource::class,
    AirportResource::class,
    UserResource::class,
])
```

And I registered the dashboard statistics widget

```
->discoverPages(in: app_path('Filament/Pages'), for: 'App\\Filament\\Pages')
->pages([
    Pages\Dashboard::class,
])
->discoverWidgets(in: app_path('Filament/Widgets'), for: 'App\\Filament\\Widgets')
->widgets([
    StatsOverviewWidget::class,
])
```

Registering the Provider

The provider is registered in: bootstrap/providers.php

```
bootstrap
  > cache
    app.php
    providers.php
```

The file now includes:

```
use App\Providers\AppServiceProvider;
use App\Providers\Filament\AdminPanelProvider;

return [
    AppServiceProvider::class,
    AdminPanelProvider::class,
];
```

Without this provider registration, Laravel would not load the Filament panel.

Restricting Admin Access

Filament needs to know which users are allowed to access the panel.
So I updated: app/Models/User.php

The User model now implements: FilamentUser

And it contains:

```php
use Database\Factories\UserFactory;
use Filament\Models\Contracts\FilamentUser;
use Filament\Panel;
```

```php
public function canAccessPanel(Panel $panel): bool
{
    return $this->role === 'admin';
}
```

This means:

- admin users can login to /admin
- normal users cannot access the panel

This is better than just hiding links, because the access control is checked by Filament itself.

Filament Resources Created

I created the resources inside: app/Filament/Resources

The resources are:

```
app
  Filament
    Resources
      AirlineResource\Pages
      AirportResource
      BookingResource
      FlightResource
      UserResource
```

Each resource controls one section of the admin panel.

Airline Resource

This resource manages airlines.

The table shows:

```php
public static function table(Table $table): Table
{
    return $table
        ->columns([
            Tables\Columns\TextColumn::make('id')
                ->sortable(),
            Tables\Columns\TextColumn::make('name')
                ->searchable()
                ->sortable(),
            Tables\Columns\TextColumn::make('code')
                ->searchable()
                ->sortable(),
            Tables\Columns\TextColumn::make('created_at')
                ->date()
                ->sortable(),
        ])
        ->defaultSort('name')
        ->actions([
            Tables\Actions\EditAction::make(),
        ])
        ->bulkActions([
            Tables\Actions\BulkActionGroup::make([
                Tables\Actions\DeleteBulkAction::make(),
            ]),
        ]);
}
```

The form contains:

```php
public static function form(Form $form): Form
{
    return $form
        ->schema([
            Forms\Components\TextInput::make('name')
                ->required()
                ->unique(ignoreRecord: true)
                ->maxLength(255),
            Forms\Components\TextInput::make('code')
                ->required()
                ->unique(ignoreRecord: true)
                ->maxLength(5)
                ->formatStateUsing(fn (?string $state): ?string => filled($state) ? strtoupper($state) : null)
                ->dehydrateStateUsing(fn (?string $state): ?string => filled($state) ? strtoupper($state) : null),
            Forms\Components\TextInput::make('logo_url')
                ->label('Logo URL')
                ->url()
                ->nullable()
                ->maxLength(255),
        ]);
}
```

The code field is automatically converted to uppercase before saving
The airline code is also unique, so two airlines cannot have the same code.

Pages Created:

```php
public static function getPages(): array
{
    return [
        'index' => Pages\ListAirlines::route('/'),
        'create' => Pages\CreateAirline::route('/create'),
        'edit' => Pages\EditAirline::route('/{record}/edit'),
    ];
}
```

One of the pages:

```php
namespace App\Filament\Resources\AirlineResource\Pages;

use App\Filament\Resources\AirlineResource;
use Filament\Actions;
use Filament\Resources\Pages\ListRecords;

class ListAirlines extends ListRecords
{
    protected static string $resource = AirlineResource::class;

    protected function getHeaderActions(): array
    {
        return [
            Actions\CreateAction::make(),
        ];
    }
}
```

Airport Resource

This resource manages airports.

```php
public static function form(Form $form): Form
{
    return $form
        ->schema([
            Forms\Components\TextInput::make('name')
                ->required()
                ->maxLength(255),
            Forms\Components\TextInput::make('code')
                ->label('IATA Code')
                ->required()
                ->unique(ignoreRecord: true)
                ->maxLength(3)
                ->formatStateUsing(fn (?string $state): ?string => filled($state) ? strtoupper($state) : null)
                ->dehydrateStateUsing(fn (?string $state): ?string => filled($state) ? strtoupper($state) : null),
            Forms\Components\TextInput::make('city')
                ->required()
                ->maxLength(255),
            Forms\Components\TextInput::make('country')
                ->required()
                ->maxLength(255),
        ]);
}
```

```php
public static function table(Table $table): Table
{
    return $table
        ->modifyQueryUsing(fn (Builder $query): Builder => $query->orderBy('country')->orderBy('city'))
        ->columns([
            Tables\Columns\TextColumn::make('id')
                ->sortable(),
            Tables\Columns\TextColumn::make('name')
                ->searchable()
                ->sortable(),
            Tables\Columns\TextColumn::make('code')
                ->label('IATA')
                ->searchable()
                ->sortable(),
            Tables\Columns\TextColumn::make('city')
                ->searchable()
                ->sortable(),
            Tables\Columns\TextColumn::make('country')
                ->searchable()
                ->sortable(),
        ])
        ->actions([
            Tables\Actions\EditAction::make(),
        ])
        ->bulkActions([
            Tables\Actions\BulkActionGroup::make([
                Tables\Actions\DeleteBulkAction::make(),
            ]),
        ]);
}
```

The IATA code is converted to uppercase before saving.

The default ordering is by country, then by city.

```php
public static function getPages(): array
{
    return [
        'index' => Pages\ListAirports::route('/'),
        'create' => Pages\CreateAirport::route('/create'),
        'edit' => Pages\EditAirport::route('/{record}/edit'),
    ];
}
```

Flight Resource

This is one of the most important admin resources because admins need to manage flights.

The table shows:

- ID
- flight number
- airline name
- origin airport code
- destination airport code
- departure time
- arrival time
- price
- available seats
- status

```php
public static function table(Table $table): Table
{
    return $table
        ->columns([
            Tables\Columns\TextColumn::make('id')
                ->sortable(),
            Tables\Columns\TextColumn::make('flight_number')
                ->searchable()
                ->sortable(),
            Tables\Columns\TextColumn::make('airline.name')
                ->label('Airline')
                ->searchable()
                ->sortable(),
            Tables\Columns\TextColumn::make('originAirport.code')
                ->label('From')
                ->searchable(),
            Tables\Columns\TextColumn::make('destinationAirport.code')
                ->label('To')
                ->searchable(),
            Tables\Columns\TextColumn::make('departure_time')
                ->dateTime('d M Y H:i')
                ->sortable(),
            Tables\Columns\TextColumn::make('arrival_time')
                ->dateTime('d M Y H:i')
                ->sortable(),
            Tables\Columns\TextColumn::make('price')
                ->prefix('$')
                ->sortable(),
            Tables\Columns\TextColumn::make('available_seats')
                ->sortable(),
            Tables\Columns\TextColumn::make('status')
                ->badge()
                ->color(fn (string $state): string => match ($state) {
```

```php
        Tables\Columns\TextColumn::make('status')
            ->badge()
            ->color(fn (string $state): string => match ($state) {
                'scheduled' => 'primary',
                'delayed' => 'warning',
                'cancelled' => 'danger',
                'completed' => 'success',
            })
            ->sortable(),
    ])
    ->filters([
        Tables\Filters\SelectFilter::make('status')
            ->options([
                'scheduled' => 'Scheduled',
                'delayed' => 'Delayed',
                'cancelled' => 'Cancelled',
                'completed' => 'Completed',
            ]),
        Tables\Filters\SelectFilter::make('airline_id')
            ->label('Airline')
            ->relationship('airline', 'name'),
    ])
    ->defaultSort('departure_time', 'desc')
    ->actions([
        Tables\Actions\EditAction::make(),
    ])
    ->bulkActions([
        Tables\Actions\BulkActionGroup::make([
            Tables\Actions\DeleteBulkAction::make(),
        ]),
    ]);
```

The status appears as a badge:
scheduled -> primary
delayed -> warning
cancelled -> danger
completed -> success

The form:

```php
return $form
    ->schema([
        Forms\Components\TextInput::make('flight_number')
            ->required()
            ->unique(ignoreRecord: true)
            ->maxLength(20),
        Forms\Components\Select::make('airline_id')
            ->required()
            ->relationship('airline', 'name')
            ->searchable()
            ->preload(),
        Forms\Components\Select::make('origin_airport_id')
            ->label('Origin Airport')
            ->required()
            ->relationship('originAirport', 'name')
            ->searchable()
            ->preload(),
        Forms\Components\Select::make('destination_airport_id')
            ->label('Destination Airport')
            ->required()
            ->relationship('destinationAirport', 'name')
            ->searchable()
            ->preload()
            ->different('origin_airport_id'),
        Forms\Components\DateTimePicker::make('departure_time')
            ->required(),
        Forms\Components\DateTimePicker::make('arrival_time')
            ->required()
            ->rule('after_or_equal:departure_time'),
        Forms\Components\TextInput::make('price')
            ->numeric()
            ->required()
            ->minValue(0)
            ->prefix('$'),
```

```php
        Forms\Components\TextInput::make('total_seats')
            ->numeric()
            ->required()
            ->minValue(1)
            ->rules(['integer']),
        Forms\Components\TextInput::make('available_seats')
            ->numeric()
            ->required()
            ->minValue(0)
            ->rules(['integer', 'lte:total_seats']),
        Forms\Components\Select::make('status')
            ->required()
            ->options([
                'scheduled' => 'Scheduled',
                'delayed' => 'Delayed',
                'cancelled' => 'Cancelled',
                'completed' => 'Completed',
            ]),
    ]);
```

The airline and airport fields use relationships from the existing Eloquent models.

For example:
->relationship('airline', 'name')
->relationship('originAirport', 'name')
->relationship('destinationAirport', 'name')

This means Filament can display readable names instead of only IDs.

Validation added in the Filament form:

- flight number must be unique
- destination airport must be different from origin airport
- arrival time must be after or equal to departure time
- price must be at least 0
- total seats must be at least 1
- available seats must be at least 0
- available seats cannot exceed total seats

Filters added:

- filter by status
- filter by airline

Default sorting:

- departure_time DESC

```php
public static function getPages(): array
{
    return [
        'index' => Pages\ListFlights::route('/'),
        'create' => Pages\CreateFlight::route('/create'),
        'edit' => Pages\EditFlight::route('/{record}/edit'),
    ];
}
```

Booking Resource

This resource is read-only because bookings are created through the API.

So I did not add create or edit pages for bookings.

The booking pages are only:

- List
- View

The booking table shows:

```php
return $table
    ->columns([
        Tables\Columns\TextColumn::make('id')
            ->sortable(),
        Tables\Columns\TextColumn::make('booking_reference')
            ->searchable()
            ->sortable(),
        Tables\Columns\TextColumn::make('user.name')
            ->label('Passenger Name')
            ->searchable()
            ->sortable(),
        Tables\Columns\TextColumn::make('user.email')
            ->label('Email')
            ->searchable(),
        Tables\Columns\TextColumn::make('flight.flight_number')
            ->label('Flight')
            ->searchable()
            ->sortable(),
        Tables\Columns\TextColumn::make('seat_count')
            ->sortable(),
        Tables\Columns\TextColumn::make('total_price')
            ->prefix('$')
            ->sortable(),
        Tables\Columns\TextColumn::make('status')
            ->badge()
            ->color(fn (string $state): string => match ($state) {
                'pending' => 'warning',
                'confirmed' => 'success',
                'cancelled' => 'danger',
            })
            ->sortable(),
        Tables\Columns\TextColumn::make('created_at')
            ->label('Booked At')
            ->date()
            ->sortable(),
    ])
    ->filters([
        Tables\Filters\SelectFilter::make('status')
            ->options([
                'pending' => 'Pending',
                'confirmed' => 'Confirmed',
                'cancelled' => 'Cancelled',
            ]),
        Tables\Filters\SelectFilter::make('flight_id')
            ->label('Flight')
            ->relationship('flight', 'flight_number'),
    ])
    ->defaultSort('created_at', 'desc')
    ->actions([
        Tables\Actions\ViewAction::make(),
        Tables\Actions\Action::make('cancel')
            ->label('Cancel')
            ->color('danger')
            ->requiresConfirmation()
            ->modalHeading('Are you sure you want to cancel this booking?')
            ->visible(fn (Booking $record): bool => $record->status !== 'cancelled')
            ->action(function (Booking $record): void {
                DB::transaction(function () use ($record): void {
                    $record->load('flight');
                    $record->flight?->increment('available_seats', $record->seat_count);
                    $record->update(['status' => 'cancelled']);
                });

                Notification::make()
                    ->title('Booking cancelled successfully.')
                    ->success()
                    ->send();
            }),
    ]);
```

The status appears as a badge:

pending -> warning
confirmed -> success
cancelled -> danger

Filters added:

- filter by status
- filter by flight

Default sorting: created_at DESC

In BookingResource, I added a custom row action called: Cancel
This action is visible only when: status != cancelled

When the admin clicks Cancel, Filament shows a confirmation modal:
Are you sure you want to cancel this booking?

If confirmed, the action does two things inside a database transaction:
1- Updates booking status to cancelled
2- Releases the seats back to the flight

Example:
If a booking has: seat_count = 3
and the flight has: available_seats = 20
after cancellation: available_seats = 23

This is important because cancelling a booking should make the seats available again.

After cancellation, Filament shows a success notification:
Booking cancelled successfully

```php
public static function getPages(): array
{
    return [
        'index' => Pages\ListBookings::route('/'),
        'view' => Pages\ViewBooking::route('/{record}'),
    ];
}
```

The booking view page shows all important booking details.

```php
public static function infolist(Infolist $infolist): Infolist
{
    return $infolist
        ->schema([
            Infolists\Components\TextEntry::make('id'),
            Infolists\Components\TextEntry::make('user.name')
                ->label('Passenger Name'),
            Infolists\Components\TextEntry::make('user.email')
                ->label('Email'),
            Infolists\Components\TextEntry::make('flight.flight_number')
                ->label('Flight'),
            Infolists\Components\TextEntry::make('booking_reference'),
            Infolists\Components\TextEntry::make('seat_count'),
            Infolists\Components\TextEntry::make('total_price')
                ->prefix('$'),
            Infolists\Components\TextEntry::make('status')
                ->badge()
                ->color(fn (string $state): string => match ($state) {
                    'pending' => 'warning',
                    'confirmed' => 'success',
                    'cancelled' => 'danger',
                }),
            Infolists\Components\TextEntry::make('created_at')
                ->dateTime('d M Y H:i'),
            Infolists\Components\TextEntry::make('updated_at')
                ->dateTime('d M Y H:i'),
        ])
        ->columns(2);
}
```

This page is useful for the admin because the admin can inspect a booking without editing it.

Passenger Relation Manager

Inside a booking view, I added a passengers table

The passengers table shows:

```php
class PassengersRelationManager extends RelationManager
{
    protected static string $relationship = 'passengers';

    public function table(Table $table): Table
    {
        return $table
            ->columns([
                Tables\Columns\TextColumn::make('first_name')
                    ->searchable(),
                Tables\Columns\TextColumn::make('last_name')
                    ->searchable(),
                Tables\Columns\TextColumn::make('passport_number')
                    ->searchable(),
                Tables\Columns\TextColumn::make('nationality')
                    ->searchable(),
                Tables\Columns\TextColumn::make('date_of_birth')
                    ->date()
                    ->sortable(),
            ]);
    }
```

This relation manager is read-only.

That means the admin can see the passengers for a booking, but cannot create, edit, or delete passengers from this panel.

This matches the API logic because passenger data belongs to the booking flow.

User Resource

This resource allows the admin to manage users.

```php
return $table
    ->columns([
        Tables\Columns\TextColumn::make('id')
        Tables\Columns\TextColumn::make('name')
            ->searchable()
            ->sortable(),
        Tables\Columns\TextColumn::make('email')
            ->searchable()
            ->sortable(),
        Tables\Columns\TextColumn::make('phone')
            ->searchable(),
        Tables\Columns\TextColumn::make('role')
            ->badge()
            ->color(fn (string $state): string => match ($state) {
                'admin' => 'danger',
                'user' => 'primary',
            })
            ->sortable(),
        Tables\Columns\TextColumn::make('created_at')
            ->date()
            ->sortable(),
    ])
    ->filters([
        Tables\Filters\SelectFilter::make('role')
            ->options([
                'admin' => 'Admin',
                'user' => 'User',
            ]),
    ])
    ->defaultSort('created_at', 'desc')
    ->actions([
        Tables\Actions\EditAction::make(),
    ])
    ->bulkActions([
        Tables\Actions\BulkActionGroup::make([
```

The role appears as a badge:
admin -> danger
user -> primary

The form:

```php
public static function form(Form $form): Form
{
    return $form
        ->schema([
            Forms\Components\TextInput::make('name')
                ->required()
                ->maxLength(255),
            Forms\Components\TextInput::make('email')
                ->required()
                ->email()
                ->unique(ignoreRecord: true)
                ->maxLength(255),
            Forms\Components\TextInput::make('phone')
                ->nullable()
                ->maxLength(255),
            Forms\Components\Select::make('role')
                ->required()
                ->options([
                    'admin' => 'Admin',
                    'user' => 'User',
                ])
                ->default('user'),
            Forms\Components\TextInput::make('password')
                ->password()
                ->revealable()
                ->required(fn (string $operation): bool => $operation === 'create')
                ->dehydrated(fn (?string $state): bool => filled($state))
                ->dehydrateStateUsing(fn (string $state): string => bcrypt($state))
                ->maxLength(255),
        ]);
}
```

The password field is important.
On create: password is required
On edit: password is optional

This means if I edit a user and leave the password empty, the old password stays unchanged.
The password is only saved when a value is filled.

This keeps passwords secure and avoids saving plain text passwords.

Filter added:

- filter by role

```php
public static function getPages(): array
{
    return [
        'index' => Pages\ListUsers::route('/'),
        'create' => Pages\CreateUser::route('/create'),
        'edit' => Pages\EditUser::route('/{record}/edit'),
    ];
}
```

Dashboard Stats Widget

This widget appears on the Filament dashboard.

It shows these statistics:

1- Total Flights
2- Total Bookings
3- Confirmed Bookings
4- Cancelled Bookings
5- Registered Users
6- Total Revenue

The stats are calculated directly from the database models.

```php
protected function getStats(): array
{
    return [
        Stat::make('Total Flights', Flight::count())
            ->color('blue'),
        Stat::make('Total Bookings', Booking::count())
            ->color('primary'),
        Stat::make('Confirmed Bookings', Booking::where('status', 'confirmed')->count())
            ->color('success'),
        Stat::make('Cancelled Bookings', Booking::where('status', 'cancelled')->count())
            ->color('danger'),
        Stat::make('Registered Users', User::where('role', 'user')->count())
            ->color('warning'),
        Stat::make('Total Revenue', '$'.number_format(Booking::where('status', 'confirmed')->sum('total_price'), 2))
            ->color('success'),
    ];
}
```

The total revenue only counts confirmed bookings.

This makes sense because pending or cancelled bookings should not be included in revenue.

Admin Seeder

I added a seeder to create the first admin user.

The admin credentials are:
Email: admin@babivoyage.com
Password: admin123456

```php
class AdminSeeder extends Seeder
{
    public function run(): void
    {
        User::updateOrCreate(
            ['email' => 'admin@babivoyage.com'],
            [
                'name' => 'BabiVoyage Admin',
                'email' => 'admin@babivoyage.com',
                'password' => bcrypt('admin123456'),
                'role' => 'admin',
                'phone' => null,
            ]
        );
    }
}
```

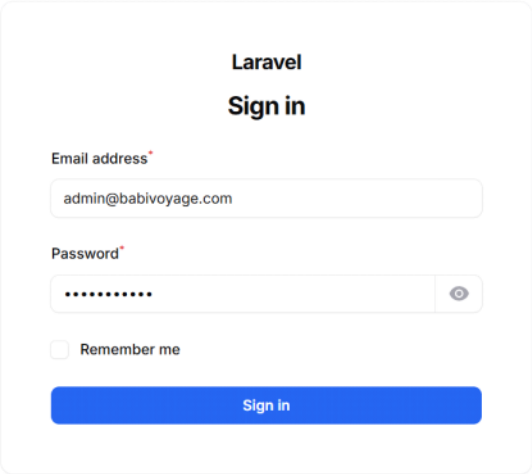
This is useful because if I run the seeder more than once, it will update the same admin user instead of creating duplicates.

The admin user has: role = admin

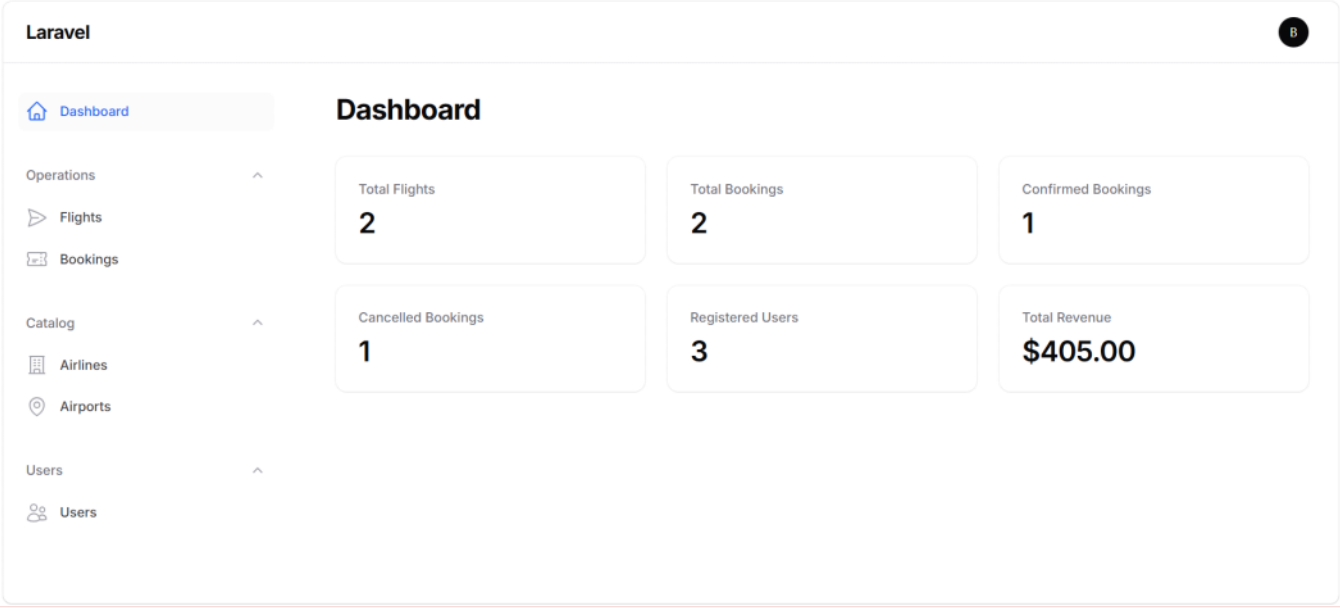
Then I registered the seeder in: database/seeders/DatabaseSeeder.php

```php
$this->call([
    AdminSeeder::class,
    AirlineSeeder::class,
    AirportSeeder::class,
]);
```

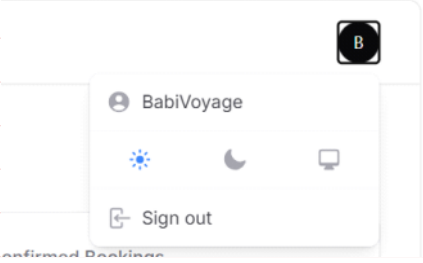
Screenshots of the Filament Project



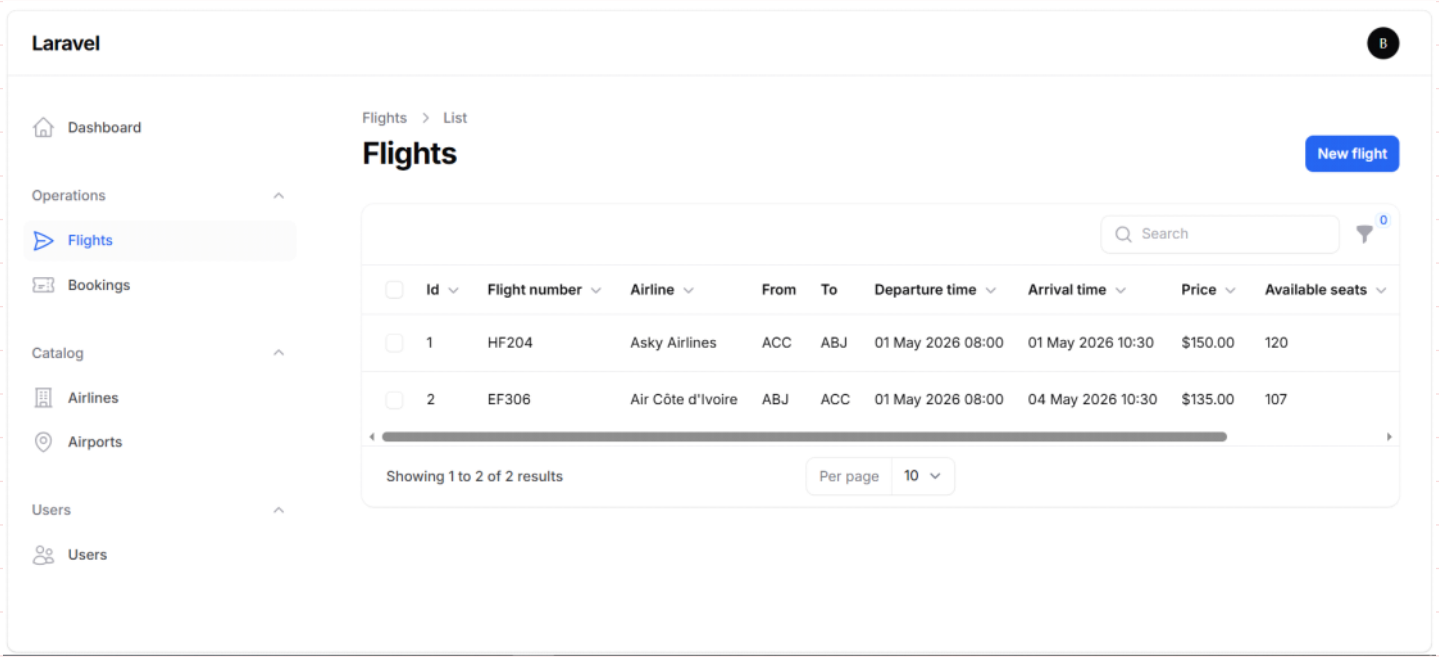
Dashboard Page:

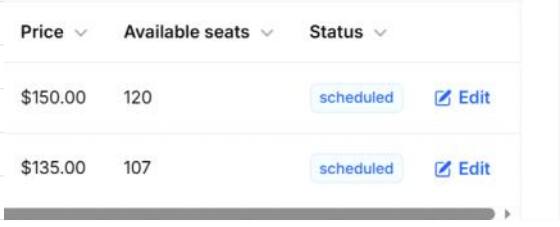


Profile

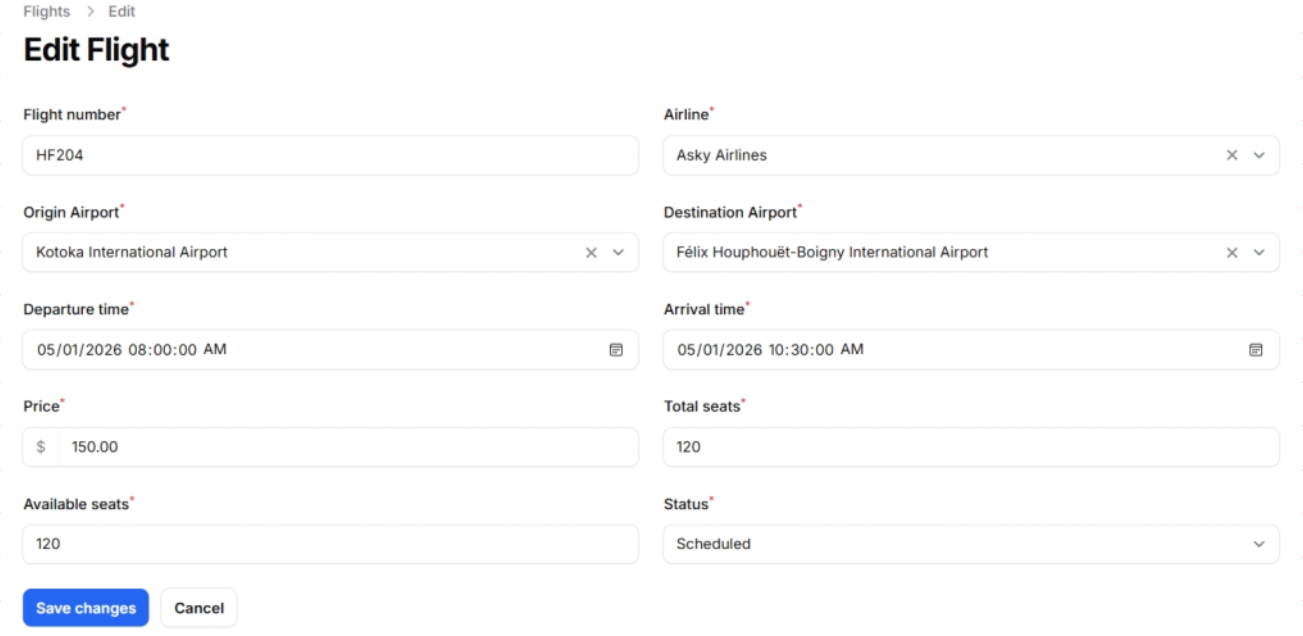


Flights Page:





Edit Flight Page:



Flights > Edit

Edit Flight

Flight number*
HF204

Airline*
Asky Airlines

Origin Airport*
Kotoka International Airport

Destination Airport*
Félix Houphouët-Boigny International Airport

Departure time*
05/01/2026 08:00:00 AM

Arrival time*
05/01/2026 10:30:00 AM

Price*
$ 150.00

Available seats*
120

Select an option
Scheduled
Delayed
Cancelled
Completed

Scheduled

Save changes | Cancel

Saved:

Saved

Flights > Edit

Edit Flight

Flight number*
HF204

Airline*
Asky Airlines

Origin Airport*
Kotoka International Airport

Destination Airport*
Félix Houphouët-Boigny International Airport

Departure time*
05/01/2026 08:00:00 AM

Arrival time*
05/01/2026 10:30:00 AM

Price*
$ 150.00

Total seats*
125

Available seats*
120

Status*
Scheduled

Save changes | Cancel

Create new Flight

Flights > Create

Create Flight

Flight number*

Airline*
Select an option

Origin Airport*
Select an option

Destination Airport*
Select an option

Departure time*
mm/dd/yyyy --:--:-- --

Arrival time*
mm/dd/yyyy --:--:-- --

Price*
$

Total seats*

Available seats*

Status*
Select an option

Create | Create & create another | Cancel

Created:

Created

Flights > Edit

Edit Flight

Flight number*
HK025

Airline*
Air Côte d'Ivoire

Origin Airport*
Maya-Maya Airport

Destination Airport*
Murtala Muhammed International Airport

Departure time*
06/10/2026 04:16:13 PM

Arrival time*
06/10/2026 08:20:00 PM

Price*
$ 125.00

Total seats*
130

Available seats*
124

Status*
Scheduled

Save changes | Cancel

Flights > List

Flights

New flight

Search

Id	Flight number	Airline	From	To	Departure time	Arrival time	Price	Available seats	Status	
5	HK025	Air Côte d'Ivoire	BZV	LOS	10 Jun 2026 16:16	10 Jun 2026 20:20	$125.00	124	scheduled	Edit
1	HF204	Asky Airlines	ACC	ABJ	01 May 2026 08:00	01 May 2026 10:30	$150.00	120	scheduled	Edit
2	EF306	Air Côte d'Ivoire	ABJ	ACC	01 May 2026 08:00	04 May 2026 10:30	$135.00	107	scheduled	Edit

Showing 1 to 3 of 3 results

Per page 10

Booking List:

Bookings > List

Bookings

Search

Id	Booking reference	Passenger Name	Email	Flight	Seat count	Total price	Status	Booked At	
2	BV-TUZP8CZ3	Mediane Ghassan Ozeir	mediane@gmail.com		3	$450.00	cancelled	Apr 29, 2026	
1	BV-SVIP93UP	Mediane Ghassan Ozeir	mediane@gmail.com		3	$405.00	confirmed	Apr 29, 2026	View

Showing 1 to 2 of 2 results

Per page 10

We can cancel the confirmed booking:

Status	Booked At	
cancelled	Apr 29, 2026	View
confirmed	Apr 29, 2026	View Cancel

View specific booking:

Id 1	Passenger Name Mediane Ghassan Ozeir
Email mediane@gmail.com	Flight
Booking reference BV-SVIP93UP	Seat count 3
Total price $405.00	Status confirmed
Created at 29 Apr 2026 11:59	Updated at 29 Apr 2026 11:59

Passengers

Search

First name	Last name	Passport number	Nationality	Date of birth
Mediane	Ozeir	A123456	Lebanese	Oct 2, 2002
Jad	Ozeir	A123123	Lebanese	Mar 27, 1998

Airline List:

Airlines > List

Airlines

New airline

Search

Id	Name	Code	Created at	
5	Air Burkina	2J		Edit
2	Air Côte d'Ivoire	HF		Edit
3	Air Senegal	HC		Edit
4	Arik Air	W3		Edit
1	Asky Airlines	KP		Edit
6	Ceiba Intercontinental	C2		Edit
7	Cronos Airlines	C8		Edit

Showing 1 to 7 of 7 results

Per page 10

Edit Airline:

Airlines > Edit

Edit Airline

Name*

Air Burkina

Code*

2J

Logo URL

Save changes

Cancel

Airport:

Airports > List

Airports

New airport

Search

Id	Name	IATA	City	Country	
6	Maya-Maya Airport	BZV	Brazzaville	Congo	Edit
2	Félix Houphouët-Boigny International Airport	ABJ	Abidjan	Côte d'Ivoire	Edit
1	Kotoka International Airport	ACC	Accra	Ghana	Edit
5	Nnamdi Azikiwe International Airport	ABV	Abuja	Nigeria	Edit
3	Murtala Muhammed International Airport	LOS	Lagos	Nigeria	Edit
4	Blaise Diagne International Airport	DSS	Dakar	Senegal	Edit
7	Lungi International Airport	FNA	Freetown	Sierra Leone	Edit

Showing 1 to 7 of 7 results

Per page 10

Airport edit:

Airports > Edit

Edit Airport

Name*

Maya-Maya Airport

IATA Code*

BZV

City*

Brazzaville

Country*

Congo

Save changes | Cancel

Users List:

Users > List

Users

New user

Search 0

Id	Name	Email	Phone	Role	Created at	
6	BabiVoyage	admin@babivoyage.com	81634440	admin	Jun 10, 2026	Edit
5	Test User	test@example.com		user	Apr 29, 2026	Edit
3	Jad Ozeir	jadozeir@gmail.com	12345678	user	Apr 26, 2026	Edit
2	Mediane Ozeir	medianeOz@gmail.com	123	user	Apr 25, 2026	Edit
1	Mediane Ghassan Ozeir	mediane@gmail.com	12345678	admin	Apr 25, 2026	Edit

Showing 1 to 5 of 5 results

Per page 10

Specific User Edit:

Users > Edit

Edit User

Name*

Test User

Email*

test@example.com

Phone

Role*

User

Password

Save changes | Cancel