**✈️ Flight Management Backend – Request Flow Documentation**

**1. Application Startup**

* **Entry Point:** src/index.js
* Creates Express app → Adds middlewares (json, urlencoded) → Mounts /api routes → Starts server at PORT (from .env or default 3000).

**2. Routing Structure**

1. /api → handled in routes/index.js
2. /api/V1 → handled in routes/V1/index.js
3. /api/V1/airoplane → handled in routes/V1/airoplane-route.js
4. POST / → calls AiroplaneController.createAiroplane

**3. Controller Layer**

* File: controllers/airoplane-controller.js
* Job: Accepts request, extracts input (modelNumber, capacity), calls Service.
* Returns:
  + ✅ **201** + created airplane (on success)
  + ❌ **500** + error details (on failure)

**4. Service Layer**

* File: services/airoplane-service.js
* Job: Handles business logic (currently just delegates to repository).
* Calls: airoplaneRepository.create(data)
* Returns: Created airplane record (or error).

**5. Repository Layer**

* File: repositories/airoplane-repository.js
* Inherits from CrudRepository.
* Points to Airoplane model.

**CrudRepository (common operations)**

* create(data) → uses Sequelize’s .create() to save record.
* Logs error if DB operation fails.

**6. Model Layer**

* File: models/airoplane.js
* Sequelize definition:
  + modelNumber: STRING, not null
  + capacity: INTEGER, not null
* Sequelize handles DB interaction.

**7. Full Request → Response Flow**

Client (POST /api/V1/airoplane/)

↓

Express Routers

↓

Controller (createAiroplane)

↓

Service (createAiroplane)

↓

Repository (create)

↓

CRUD Repository → Sequelize Model → Database

↑

Response (success/error) back through layers

**8. Responses**

**✅ Success (201 Created)**

{

"success": true,

"message": "Successfully create an Airoplane",

"data": {

"id": 1,

"modelNumber": "Boeing-737",

"capacity": 180,

"createdAt": "...",

"updatedAt": "..."

},

"error": {}

}

**❌ Error (500 Internal Server Error)**

{

"success": false,

"message": "Something went wrong while creating a airplane",

"data": {},

"error": "Error details here"

}

👉 This is a **layered architecture (MVC + Repository + Service)** that keeps each responsibility separate:

* **Controller** = API handling
* **Service** = Business logic
* **Repository** = Database operations
* **Model** = Data schema

**Model Creation**

* Create New Model

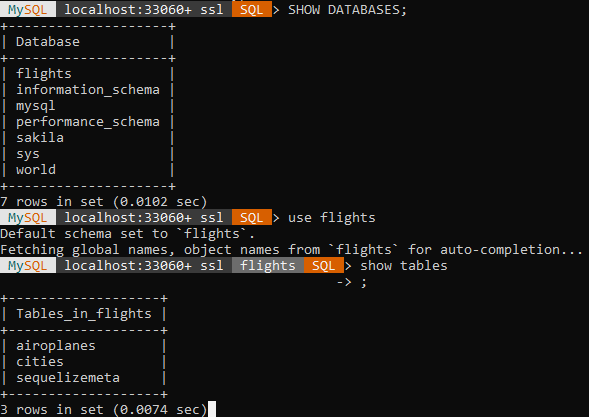
npx sequelize-cli model:generate --name City --attributes names:string

When we use npx sequelize-cli model:generate, Sequelize creates both a **model file** (under models/) and a **migration file** (under migrations/). The generated migration file contains extra default fields like id, createdAt, and updatedAt along with our defined attributes. If we only need the migration file (without the model), we can directly run the migration:generate command instead.

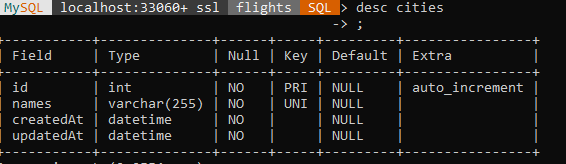
* Create Model based an migrate

npx sequelize db:migrate

it migrates all code in migrate file and create model table in Database.



Cities tables are created.



With these attributes.