Express.js Academy - REST API

Goal

Build and deploy a RESTful API using **Express.js**, following a step-by-step approach — starting with the basics and gradually adding features like database, routes, roles, and deployment.

You will receive a **class diagram** as the foundation for your models and routes.

What You'll Be Doing (Step-by-Step)

- **Start an Express project**
  - Initialize project with npm init
  - Install express and basic dependencies
  - Set up a basic "Hello World" route (GET /)
- **Create a RESTful structure**
  - Follow the class diagram to define your entities
  - Use in-memory data first, then move to a real database
- **Build API routes**
  - GET, POST, PUT, DELETE endpoints
  - Handle route parameters and request bodies
  - Validate inputs (simple checks)
- **Implement roles/authorization** *(optional/advanced)*
  - Create a basic "admin/user" role system
  - Restrict access to certain routes
- **Test the application**
  - Use Postman to test your endpoints
  - Write example requests & expected responses

Tips

- Keep your code clean: split routes, controllers, and middleware into separate files (NestJS has a good explanation on their docs for this part)
- Always test each route after creating it.
- Focus on functionality first, then refactor later

Challenge

Build a full RESTful backend using Express.js and PostgreSQL (hosted on Railway), with a proper database schema, CRUD routes, and clean architecture. Use Prisma ORM for DB access and simplicity.

Day 1/2 - Setup & Intro

- Initialize a Node.js project
- Install base dependencies
- Create folder structure: `/routes, /controllers, /models`
- Create a repo and push the initial setup
- Create railway account
- Save the `DATABASE_URL` in a .env file
- Initialize Prisma
- Define the models in schema.prisma
- Run the migration on prisma to reflect the changes on the database

Day 3 - Build First Routes

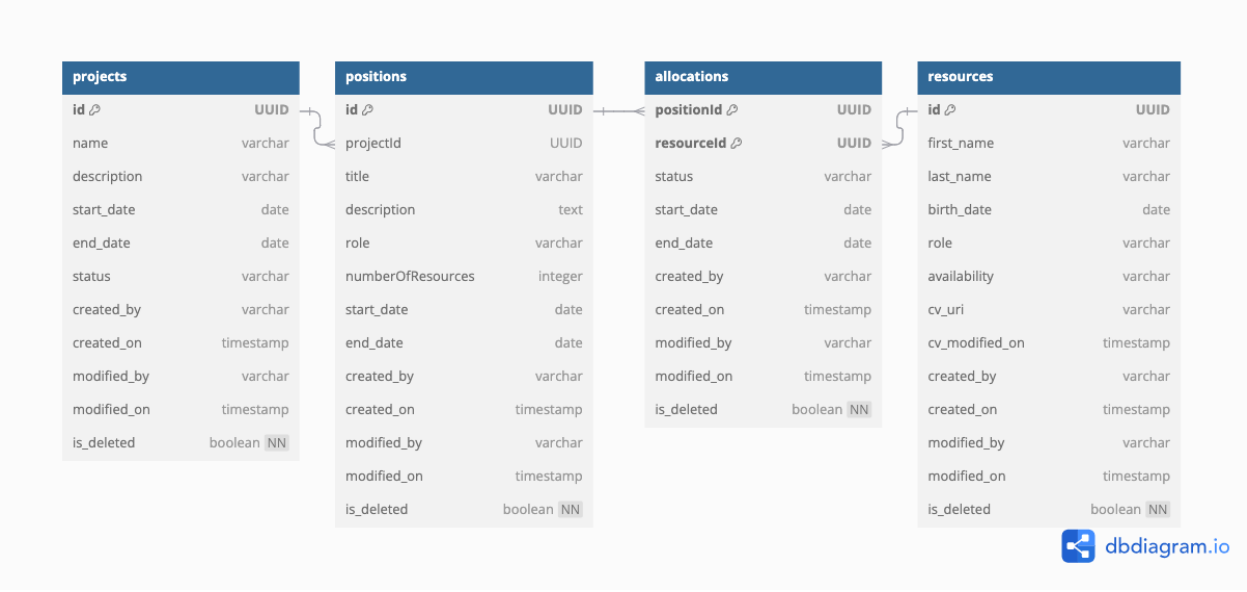
- Create CRUD routes for /projects
- Add corresponding controller logic using Prisma client
- Test with Postman
- Create basic error handling

Day 4 - Expand Schema + Update + Delete + Soft Deletes

- Set up foreign key relations in Prisma
- Add CRUD routes and controllers for new models
- Add PUT and DELETE routes for all models
- Implement soft delete (add is_deleted field)
- Ensure GET routes only return non-deleted entries
- Create utility function to filter or abstract soft deletes

Day 5 – Relations and Nested Queries

- Implement routes like `/projects/:id/positions` or `/positions/:id/allocations`
- Use Prisma's include feature for relational queries
- Test deeply nested queries and connections


projects
id UUID
name varchar
description varchar
start_date date
end_date date
status varchar
created_by varchar
created_on timestamp
modified_by varchar
modified_on timestamp
is_deleted boolean NN
positions
id UUID
projectId UUID
title varchar
description text
role varchar
numberOfResources integer
start_date date
end_date date
created_by varchar
created_on timestamp
modified_by varchar
modified_on timestamp
is_deleted boolean NN
allocations
positionId UUID
resourceId UUID
status varchar
start_date date
end_date date
created_by varchar
created_on timestamp
modified_by varchar
modified_on timestamp
is_deleted boolean NN
resources
id UUID
first_name varchar
last_name varchar
birth_date date
role varchar
availability varchar
cv_uri varchar
cv_modified_on timestamp
created_by varchar
created_on timestamp
modified_by varchar
modified_on timestamp
is_deleted boolean NN
dbdiagram.io