Express with Sequelize

Learning Objectives

- Understand how to create RESTful endpoints
- Understand how to integrate Express with Sequelize to perform CRUD operations

Material

Using Sequelize with Express

Let's say we want to expose our database to the outside world so that users on the internet can create, read, update and delete from it. To achieve this, we need to create Express endpoints for each of these operations, and have them modify the database using Sequelize.

Below is an example endpoint for getting all the cities in the database:

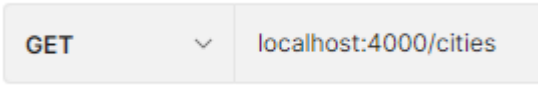
```
// app.js
const express = require("express");
const City = require("./city");
const setupDb = require("./setupDb");
const app = express();

setupDb();

app.get("/cities", async (req, res) => {
    const cities = await City.findAll();
    res.json(cities);
});

app.listen(4000, () => {
    console.log("listening on port 4000");
});
```

`findAll` is a static method Sequelize provides us which will return all the cities in the database. `res.json` can be used to send JSON data in the HTTP response.

If we run:

```
node app.js
```

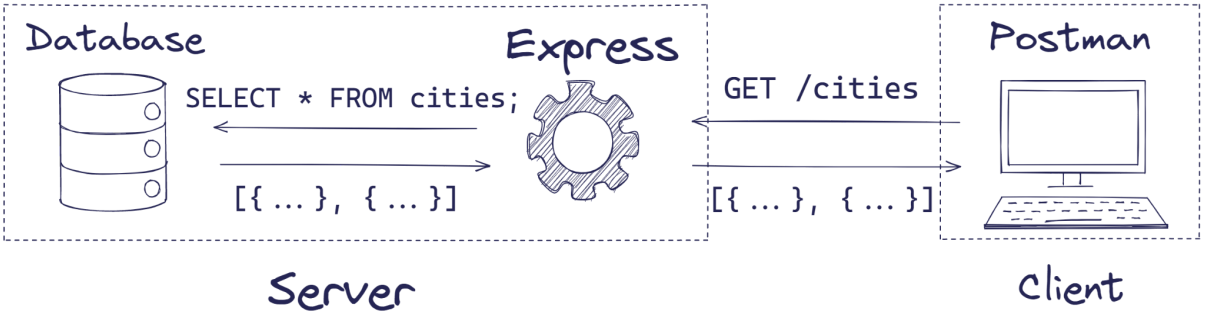
Then make a request to this endpoint using Postman:



We get the following data back:

```
[
    {
        "id": 1,
        "name": "London",
        "population": 9000000
    },
    {
        "id": 2,
        "name": "Madrid",
        "population": 3000000
    }
]
```

Here is a diagram illustrating what happens when Postman makes the GET request:



1. The client (Postman, in this case) makes an HTTP GET request to the /cities endpoint.
2. We've configured this endpoint in our app.js file by including `app.get("/cities"...)` so Express fires our callback.
3. Our callback tells Sequelize to find all the cities in our database.
4. Sequelize converts this into an SQL query to SELECT all the cities in our database.
5. When we receive these city rows, we send an HTTP response to the client with the cities data in JSON format.

REST

REST is short for "**RE**presentative **S**tate **T**ransfer". It's a popular architectural style for designing endpoints, making it easier for systems to communicate with each other. In order for our API to be considered RESTful, we need to follow certain rules.

First, we need to make sure we're using the proper HTTP methods for the proper actions:

HTTP Method	**RESTful action**
GET	Read
POST	Create
DELETE	Delete
PATCH	Partially modify
PUT	Replace

Second, we need to clearly define the resources our server works with. A resource will often (but not always) correspond to a table in our database. In our cities example, the resources are cities and landmarks. The **path of our endpoint determines which resource the endpoint is acting upon**:

Endpoint path	**Resource**
`/cities`	All the cities
`/cities/:id`	The city with the corresponding id
`/cities/:id/landmarks`	All the landmarks belonging to the city with the corresponding id

`/cities/:id1/landmarks/:id2`
or
`/landmarks/:id`
	The landmark with the corresponding id

Notice how the endpoints are pluralised ("`/cities`" not "`/city`"). Also, notice that we explicitly represent the fact that landmarks are nested within cities.

So the **path determines which resource we're acting on** and the **HTTP method determines what action we are performing**.

Endpoint	**Role**
`GET /cities`	Retrieve all the cities
`POST /cities`	Create a new city
`DELETE /cities/:id`	Delete a specific city
`POST /cities/:id/landmarks`	Add a landmark to the specific city

Sequelize Methods

Here are some Sequelize methods that are helpful for performing CRUD operations. Remember: Sequelize methods return promises so these all need to be `awaited`.

Method	Description	Example
`Class.findAll()`	Get all the rows in the database for that particular model	`City.findAll()`
`Class.findByPk(id)`	Finds the row in the table with the corresponding primary key	`City.findByPk(2)`
`instance.destroy()`	Removes a row from the database.	`london.destroy()`
`instance.update(data)`	Updates the data for a particular row	`madrid.update( { population: 3000100 } );`

As an example, here's an endpoint for replacing a city's data:

```
app.put("/cities/:id", async (req, res) => {
    const city= await City.findByPk(req.params.id);
    if (!city) {
        return res.sendStatus(404);
    }
    await city.update(req.body);
    res.sendStatus(200);
});
```

Since this endpoint isn't retrieving data, we can just send a status code with the Express `sendStatus` method. If we can't find the city in the database (the client has sent us a bad id) we send them a 404 "not found". If we find and update it, we send back a 200 "OK".

Core Assignments

RESTaurant API

You're going to create an API which handles restaurant chain data. You should have tables for **companies**, **locations** and **menus**. Locations and menus both belong to a particular company. For example, *McDonalds* is a company with a *Drinks* menu and locations in *Cheltenham* and *Cirencester*. You will need to represent models using Sequelize:

Companies should have:

- name: string
- logoUrl: string

Menus should have:

- title: string

Locations should have:

- name: string
- capacity: integer
- manager: string

Create an Express API to allow clients to make HTTP requests to read and modify this data. At minimum, your API should have the following endpoints

- Get all the companies
- Get a specific company by its id
- Get all a company's menus
- Create a new company
- Delete a company

For example, getting all the companies would be an endpoint like:

```
app.get("/companies", async (req, res) => {
    const companies = await Company.findAll();
    res.json(companies);
});
```

Make sure your endpoints function as expected by testing them manually in Postman and inspecting the database.

Extension Assignments

RESTaurant API++

Add as many of the following endpoints to your API:

- Get a specific menu by its id
- Replace a specific company
- Create a new menu
- Delete a menu
- Create a new location
- Delete a location