3.NodeJS Assignment + Notes - Sequelize and Axios

CENTRALOGIC

12/5/24

> 💡 **Github code reference using pool**
>
> https://github.com/gauravwani127/NODEJS_training/tree/main/postgresWithSequelize

Assignment

1. Setup a NodeJS application that will execute following requirements in form of different apis

   A. **A weather fetching and storing api using sequelize with**

   - POST API with route (**/api/SaveWeatherMapping**)
   - Request Body

```
{[
    {
        "city":"Pune",
        "country":"India"
    },
    {
        "city":"Mumbai",
        "country":"India"
    },
    {
        "city":"London",
        "country":"England"
    }
    ]
}
```

   - The api will take list of objects (Cities from request body) and will find its coordinates

   - Api will make axios request to GeoCoding api for finding the coordinates https://api-ninjas.com/api/geocoding
   - Once coordinates are fetched , the data will be used to find weather of all the cities in the list. https://rapidapi.com/weatherapi/api/weatherapi-com/
   - Save all the received information for all the cities in the postgres database using sequelize

     The data that will be saved in the database will look like -

     id | city | country | weather | time(Time at which data is saved) | Longitude | Latitude

   B. **A dashboard API that will be fetching the weather information from the database (Postgres)**

   - GET request with route (’/api/weatherDashboard’)
   - Request Parameters - City (Optional)
   - Response → If city parameter then response should be all data related to city
   - Response →If no parameter , all the cities available **(with only latest weather conditions)**

```
{[
    {
        "id": "",
        "city":"Pune",
        "country":"India",
        "date": "<DATE>",
        "weather": "<Weather>"
    },
    {
        "id": "",
        "city":"Mumbai",
        "country":"India",
        "date": "<DATE>",
        "weather": "<Weather>"
    },
    {
        "id": "",
        "city":"London",
        "country":"England",
        "date": "<DATE>",
```

```
        "weather": "<Weather>"
    }
    ]
}
```

C. **Mailing api that will have request body same like above api**

It will mail the data in form of table (same columns like dashboard)

D. **Debug the response that will come from https://api-ninjas.com/api/geocoding in the api (/api/SaveWeatherMapping).**

Take a screenshot to show the contents fetched from the api. Put the ss in the folder

---

Below questions are not to be documented in assignment , but requested to search for answers to enhance knowledge and develop a sense of curiosity in the subject.

1. Why headers are required for any api? Are there any API that can excecute without headers ? What are default headers?
2. Are there any industry standard alternative to nodemailer? Have you heard use of SMTP?
3. Alternatives to axios ?
4. What are DTOs ? What is its purpose?
5. Why sequelize is perferred over other ORMs ?
6. What problems can arise due to use of raw SQL queries in application?
7. All the functions are imported and exported seperately, Can there be use of Class to wrap these all funcitions and only classes can be exported and imported then?

Notes

To start with postgres connection, firstly create a web server, as it was promted in first assignment notes ...

Note: Asuming that the postgres setup is already done , database and table schema is already created

To connect your database into NodeJS application -

1. Configuring database
2. Adding a service layer and models (if any).
3. Using the service function for your requirements

1. **Configuring the database**

```
npm install sequelize
```

```
npm i --save-dev @types/sequelize
```

Create pgConfig.ts file to configure your database with the credentials and add the below code

```
// import { Pool } from 'pg';
import { Sequelize } from 'sequelize';


// const pool = new Pool({
 const sequelize = new Sequelize({

    // user: 'postgres',
    username: 'postgres',

    host: 'localhost',
    database: "testdatabas",
    password: "Password1#",
    port: 5432,

    //dialect
    dialect: "postgres",
});


//This are the functions provided by sequeize to test database connection and synchronize database schema to that of
models
sequelize.authenticate()
  .then(() => {
    console.log('Database connection established successfully.');
  })
  .catch((err) => {
    console.error('Unable to connect to the database:', err);
  });


//Synchronize
sequelize.sync()
```

```
  .then(() => {
    console.log('Models synchronized with the database.');
  })
  .catch((err) => {
    console.error('Unable to synchronize models with the database:', err);
  });

export default sequelize;
```

2. **Adding models fot that -**

```
//Importing functionalities from sequelize
import { DataTypes, Model } from 'sequelize';
import  sequelize  from './pgConfig';

interface UserAttributes {
  id?: number;
  name: string;
  email: string;
}

class User extends Model<UserAttributes> implements UserAttributes {
  public id!: number;
  public name!: string;
  public email!: string;

  // You can define additional methods and associations here

}

User.init(
  {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
    },
  },
  {
    sequelize,
    tableName: 'users',
    timestamps: false,
  }
);

export { User };
```

2. **Adding a service layer and models (if any).**

Create a service.ts , so pool can be used for manipulation of the database according to our requirement

```
import pool from './pgConfig';
import {User} from './userModel';

async function createUser(user: User): Promise<any> {
    try {
        // const query = 'INSERT INTO users (id, name, email) VALUES ($1, $2, $3)';
        // let result = await pool.query(query, [id,name, email])

        const newUser =await  User.create(user);
        if(newUser){
            return newUser;
        }

}catch(err: any){
    throw err
}

}

// Read
```

```
async function getUsers(): Promise<any[]> {
    // const query = 'SELECT * FROM users';
    // const result = await pool.query(query);
    // return result.rows;
    const users = await User.findAll();
    return users;
}

// Update
async function updateUser(user:User): Promise<any> {
    // const query = 'UPDATE users SET name = $2, email = $3 WHERE id = $1';
    // await pool.query(query, [id, name, email]);

     try {
    const userEntity = await User.findByPk(user.id);
    if (!userEntity) {
      return "User not found !"
    }
    await userEntity.update(user);


     return "User updated successfully";
  } catch (err:any) {
    return `Error updating user due to ${err.message}`;
  }
}

// Delete
async function deleteUser(id: number): Promise<any> {
    // const query = 'DELETE FROM users WHERE id = $1';
    // await pool.query(query, [id]);
     try {
    const userEntity = await User.findByPk(id);
    if (!userEntity) {
      return "User not found !"
    }
    await userEntity.destroy();
     return "User updated successfully";
  } catch (err:any) {
    return `Error updating user due to ${err.message}`;
  }

}

export { createUser, getUsers, updateUser, deleteUser };
```

3. **Using the MAIN function for your requirements**

Add Below code in app.ts for database manipulation

```
import express, {Request , Response} from 'express';

import { createUser, getUsers, updateUser, deleteUser } from './service';

//2.-------Initiate dependency (Instance of Express -- app)
const app = express();
const port = 8000; //

app.use(express.json());

app.post('/users', async (req, res) => {
    // const { id,name, email } = req.body;
     await createUser(req.body); // Create one api for calling
    res.send("User created successfully")

});

app.get('/users', async (req, res) => {
    const users = await getUsers();
    res.json(users);
});

app.put('/users', async (req, res) => {
    // const id = parseInt(req.params.id);
    // const { name, email } = req.body;
    await updateUser(req.body);
    res.send('User updated successfully');
});
```

```
app.delete('/users/:id', async (req, res) => {
    const id = parseInt(req.params.id);
    await deleteUser(id);
    res.status(200).send('User deleted successfully');
});


app.listen(port, ()=> {
    console.log(` Hii we are comfortable in NodeJS `);

})
```

Axios in NodeJS

```
npm install axios
npm install --save-dev @types/axios
```

Important string functions often used during projects

```
import axios, { AxiosResponse } from 'axios';

const BASE_URL = 'https://yourapi.com';

interface RequestData {
  [key: string]: any;
}

interface ResponseData {
  [key: string]: any;
}

// GET request
async function getRequest(endpoint: string): Promise<AxiosResponse<ResponseData>> {
  try {
    const response = await axios.get<ResponseData>(`${BASE_URL}${endpoint}`);
    return response;
  } catch (error) {
    throw new Error(`GET request failed: ${error}`);
  }
}

// POST request
async function postRequest(endpoint: string, data: RequestData): Promise<AxiosResponse<ResponseData>> {
  try {
    const response = await axios.post<ResponseData>(`${BASE_URL}${endpoint}`, data);
    return response;
  } catch (error) {
    throw new Error(`POST request failed: ${error}`);
  }
}

// PUT request
async function putRequest(endpoint: string, data: RequestData): Promise<AxiosResponse<ResponseData>> {
  try {
    const response = await axios.put<ResponseData>(`${BASE_URL}${endpoint}`, data);
    return response;
  } catch (error) {
    throw new Error(`PUT request failed: ${error}`);
  }
}

// PATCH request
async function patchRequest(endpoint: string, data: RequestData): Promise<AxiosResponse<ResponseData>> {
  try {
    const response = await axios.patch<ResponseData>(`${BASE_URL}${endpoint}`, data);
    return response;
  } catch (error) {
    throw new Error(`PATCH request failed: ${error}`);
  }
}

// DELETE request
async function deleteRequest(endpoint: string): Promise<AxiosResponse<ResponseData>> {
  try {
    const response = await axios.delete<ResponseData>(`${BASE_URL}${endpoint}`);
    return response;
```

```
  } catch (error) {
    throw new Error(`DELETE request failed: ${error}`);
  }
}
```

JavaScript basics

I suggest solving atleast one question every day-

https://exercism.org/tracks/javascript

https://javascript.info/

https://learning.postman.com/docs/sending-requests/requests/

How to submit assignment?

Theoretical questions are not to be submitted, Check documentations for finding answers to them

For 1. Coding question to be sent by pushing it into a public git respository . Exclude Node_Modules using gitignore

Maintain a diffeernt folder for Outputs , put here ss of the resposses of the apis given in the requirement

+

Documentation is the key, Happy Learning---

CentraLogic

Gaurav Wani

Team Lead | CentraLogic Consultancy