API Service Project

Author: Duc Thai

Tampere University of Applied Sciences

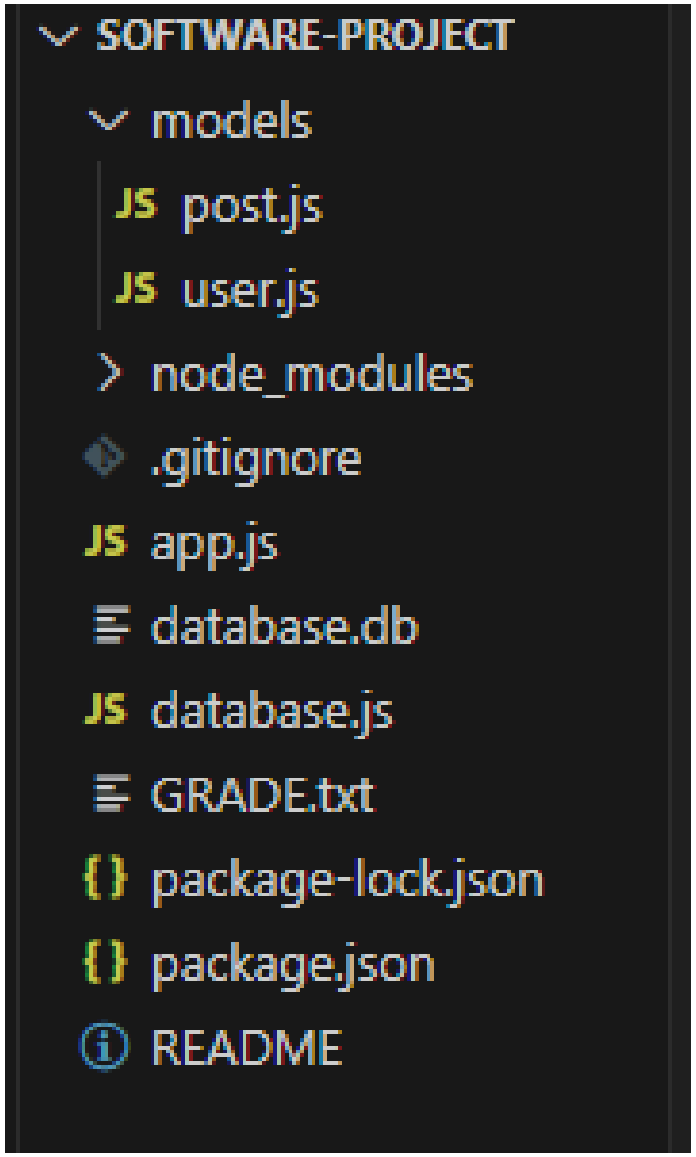
Introduction

- This project was developed as a part of API services course, aiming to demonstrate practical implementation of web development technologies.

Overview

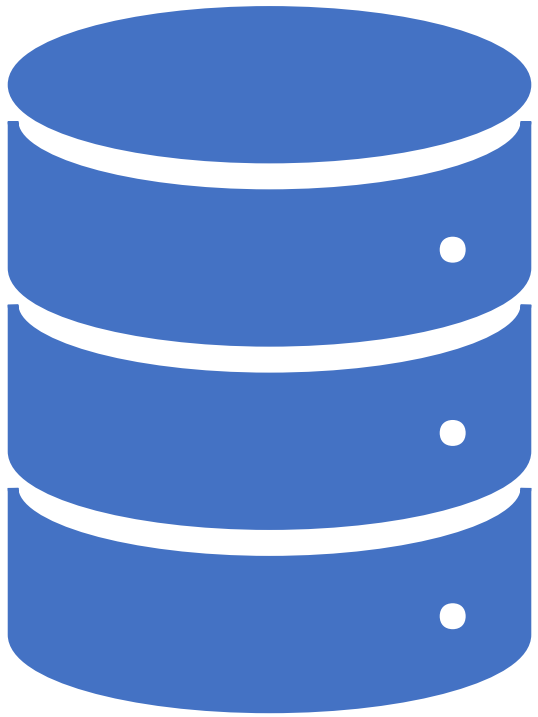
- This project is an APi developed using Node.js with Express and Sequelize, interfacing with a SQLite database.
- Key features:
  - HTTP Server: Utilizes the Express framework to handle HTTP requests and responses efficiently.
  - Relational Database Integration: Incorporates a SQLite database, managed through the Sequelize ORM, ensuring structured and efficient data handling.
  - API Functionality:
    - GET Endpoints: Allows searching and retrieving information using multiple criteria.
    - POST Endpoints: Enables adding new data to the database.
    - PATCH Endpoints: Facilitates the modification of existing data.
    - Data Relationships: Manages complex data relationships, exemplified by two related tables with JOIN operations.
  - Error Handling: Implements comprehensive error handling to ensure reliability and ease of debugging.
  - JSON Responses: Ensures all responses are in JSON format, providing a standardized API response structure.

Project Architechture



Database Design

- User Table:
  - Columns:
    - id: Integer, Primary Key, Auto Increment.
    - name: String, Not Null.
    - email: String, Not Null, Unique.
- Post Table:
  - Columns:
    - id: Integer, Primary Key, Auto Increment.
    - title: String, Not Null.
    - content: Text, Not Null.
    - userId: Integer, Foreign Key referencing id in User table, Not Null.
- Relationships:
    - One-to-Many Relationship between User and Post:
    - One User can have multiple Posts.
    - Represent this by a line connecting User.id to Post.userId.



Key Features

```javascript
const express = require("express");
const app = express();

const sequelize = require("./database");
const User = require("./models/user");
const Post = require("./models/post");
const router = express.Router();

app.use(express.json());

sequelize.sync({ force: true }).then(() => {
  console.log("Database & tables created!");
});

router.get("/users", async (req, res) => {
  try {
    const users = await User.findAll({
      include: [Post],
    });
    res.json(users);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

```javascript
Post.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    title: {
        type: DataTypes.STRING,
        allowNull: false
    },
    content: {
        type: DataTypes.TEXT,
        allowNull: false
    }
}, {
    sequelize,
    modelName: 'Post',
    timestamps: true
});

Post.belongsTo(User, {
    foreignKey: {
        name: 'userId',
        allowNull: false
    },
    onDelete: 'CASCADE'
});
User.hasMany(Post, {
    foreignKey: 'userId'
});
```

```javascript
class User extends Model {}

User.init({

    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    name: {
        type: DataTypes.STRING,
        allowNull: false
    },
    email: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true,
    }
}, {
    sequelize,
    modelName: 'User',
    timestamps: true
});

module.exports = User;
```

Key feature (part 2)

```
router.patch("/posts/:id", async (req, res) => {
  try {
    const { id } = req.params;
    const updateData = req.body;

    const post = await Post.findByPk(id);
    if (!post) {
      return res.status(404).json({ message: "Post not found" });
    }

    await post.update(updateData);

    res.status(200).json(post);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
app.use(router);
const port = process.env.PORT || 5000;

app.listen(port, () => console.log(`server running on port ${port}`));
```

```
router.post("/users", async (req, res) => {
  try {
    const { name, email } = req.body;
    const newUser = await User.create({ name, email });
    res.status(201).json(newUser);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

router.post("/posts", async (req, res) => {
  try {
    const { title, content, userId } = req.body;
    const newPost = await Post.create({ title, content, userId });
    res.status(201).json(newPost);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

```
const {Sequelize   } = require('sequelize')

const sequelize = new Sequelize({
    dialect: 'sqlite',
    storage: './database.db'
})

module.exports = sequelize;
```

Demo

```
    res.status(200).json(post);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
app.use(router);
const port = process.env.PORT || 5000;

app.listen(port, () => console.log(`server running on port ${port}`));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash

```
Executing (default): DROP TABLE IF EXISTS `Posts`;
Executing (default): PRAGMA foreign_keys = ON
Executing (default): DROP TABLE IF EXISTS `Users`;
Executing (default): CREATE TABLE IF NOT EXISTS `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `na
me` VARCHAR(255) NOT NULL, `email` VARCHAR(255) NOT NULL UNIQUE, `createdAt` DATETIME NOT NULL, `upd
atedAt` DATETIME NOT NULL);
Executing (default): PRAGMA INDEX_LIST(`Users`)
Executing (default): PRAGMA INDEX_INFO(`sqlite_autoindex_Users_1`)
Executing (default): DROP TABLE IF EXISTS `Posts`;
Executing (default): CREATE TABLE IF NOT EXISTS `Posts` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `ti
tle` VARCHAR(255) NOT NULL, `content` TEXT NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATE
TIME NOT NULL, `userId` INTEGER NOT NULL REFERENCES `Users` (`id`) ON DELETE CASCADE ON UPDATE CASCA
DE);
Executing (default): PRAGMA INDEX_LIST(`Posts`)
Database & tables created!
```

GET http://localhost:5000/users

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (7) Test Results 200 OK 7 ms 659

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "name": "Test2",
        "email": "test2@email.com",
        "createdAt": "2023-11-29T12:43:32.537Z",
        "updatedAt": "2023-11-29T12:43:32.537Z",
        "Posts": [
            {
                "id": 1,
                "title": "Test post",
                "content": "Fixed test post",
                "createdAt": "2023-11-29T12:43:45.296Z",
                "updatedAt": "2023-11-29T12:45:53.518Z",
                "userId": 1
            }
        ]
```

Demo (part 2)

POST http://localhost:5000/users

Params Authorization Headers (8) Body Pre-request Script Tests Sett

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

```
1 {
2     "name": "Test",
3     "email": "test@email.com"
4 }
```

ody Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 1,
3     "name": "Test",
4     "email": "test@email.com",
5     "updatedAt": "2023-11-29T13:18:59.219Z",
6     "createdAt": "2023-11-29T13:18:59.219Z"
7 }
```

POST http://localhost:5000/posts

Params Authorization Headers (8) Body Pre-request Script Tests Settings

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

```
1 {
2     "title": "Test post",
3     "userId": 1,
4     "content": "This is test"
5 
6 }
```

Body Cookies Headers (7) Test Results 201 Created 17 m

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 1,
3     "title": "Test post",
4     "content": "This is test",
5     "userId": 1,
6     "updatedAt": "2023-11-29T13:19:19.699Z",
7     "createdAt": "2023-11-29T13:19:19.699Z"
8 }
```

Demo (part 3)

PATCH http://localhost:5000/posts/1

Params Authorization Headers (8) Body Pre-request Script Tests Settings

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

```
1 {
2     "content": "Fixed test post"
3 }
```

Body Cookies Headers (7) Test Results 200 OK 2

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 1,
3     "title": "Test post",
4     "content": "Fixed test post",
5     "createdAt": "2023-11-29T13:19:19.699Z",
6     "updatedAt": "2023-11-29T13:19:52.402Z",
7     "userId": 1
8 }
```

PATCH http://localhost:5000/posts/3

Params Authorization Headers (8) Body Pre-request Script Tests Settings

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

```
1 {
2     "content": "Fixed test post"
3 }
```

Body Cookies Headers (7) Test Results 404 Not Found 6 ms 270 B

Pretty Raw Preview Visualize JSON

```
1 {
2     "message": "Post not found"
3 }
```