Node.js RESTful API with SQLite

By: Ej Sobrepeña

Date: 2023-01-12

Project Overview

- The project is a Node.js application that implements a RESTful API using SQLite.
- The API allows for searching, adding, modifying, and deleting user information.
- The project follows common API design best practices.
- The project was divided into three parts, each with increasing complexity and additional features.

Used Technologies

Node.js: The project is built on Node.js, which is a JavaScript runtime. It's used for building fast and scalable network applications.

Express.js: This is a minimal and flexible Node.js web application framework that provides robust set of features for web and mobile applications. In this project, it's used for setting up the server and handling HTTP requests.

SQLite: This is a library that provides a lightweight disk-based database. It allows accessing the database using a nonstandard variant of the SQL query language. In this project, it's used for storing and retrieving user data.

Body-parser: This is a Node.js body parsing middleware. It's responsible for parsing the incoming request bodies in a middleware before the handlers. In this project, it's used for parsing incoming request bodies in JSON format.

Challenges

- Implementing the different HTTP methods (GET, POST, PATCH, DELETE).
- Error handling for invalid or unknown parameters or their values.
- Ensuring the API follows common design best practices.
- Managing the SQLite database, including creating tables and inserting data.

Solutions

HTTP Methods: Implemented different HTTP methods (GET, POST, PATCH, DELETE) using Express.js routing.

Error Handling: Added error handling in each API endpoint to handle invalid or unknown parameters or their values.

API Design: Followed common API design best practices for structuring the endpoints.

SQLite Database Management: Used SQLite to create tables and insert data. Managed the database using SQL scripts.

Timetable

1

Project Start: 2024-01-08.

2

Part 1 Completion: 2024-01-11.

3

Part 2 Completion: 2024-01-12.

4

Part 3 Completion: 2024-01-12.

Code Overview

The code is organized into a single server.js file.

The server.js file sets up the server, defines the API endpoints, and handles the database operations.

The sql directory contains SQL scripts for managing the database.

GET /user Endpoint

- Explanation of the code:
  - This endpoint returns all users in JSON format if no query parameters are provided.
  - If the last and dept query parameters are provided, it returns users with the specified last name and department.
  - If any other query parameters are provided, it returns a bad request message.

```
app.get('/user', (req, res) => {
    if (req.query.last && req.query.dept) {
        let sql = `SELECT * FROM Users WHERE LOWER(last) = ? AND dept = ?`;
        let params = [req.query.last.toLowerCase(), req.query.dept];
        db.all(sql, params, (err, rows) => {
            if (err) {
                res.status(HTTP_STATUS_INTERNAL_SERVER_ERROR).json({ "error": err.
                message });
                return;
            }
            if (rows.length === 0) {
                return res.status(HTTP_STATUS_NOT_FOUND).json({ "message": 'No
                users found' });
            }
            res.setHeader('Content-Type', 'application/json');
            res.status(HTTP_STATUS_OK).json(rows);
        });
    } else if (Object.keys(req.query).length === 0) {
        db.all(`SELECT * FROM Users`, (err, rows) => {
            if (err) {
                res.status(HTTP_STATUS_INTERNAL_SERVER_ERROR).json({ "error": err.
                message });
                return;
            }
            res.setHeader('Content-Type', 'application/json');
            res.status(HTTP_STATUS_OK).json(rows);
        });
    } else {
        res.status(HTTP_STATUS_BAD_REQUEST).json({ 'message': 'Bad request. Missing
        required query parameters' });
    }
});
```

GET /user/:id Endpoint

- Explanation of the code:
  - This endpoint returns the user with the specified ID in JSON format.
  - If the ID parameter is missing, it returns a bad request message.
  - If no user with the specified ID exists, it returns a user not found message.

```
app.get('/user/:id', (req, res) => {
    if (!req.params.id) {
        return res.status(HTTP_STATUS_BAD_REQUEST).json({ 'message': 'Bad request.
        Missing ID parameter' });
    }
    let sql = `SELECT * FROM Users WHERE id = ?`;
    let params = [req.params.id];
    db.get(sql, params, (err, row) => {
        if (err) {
            res.status(HTTP_STATUS_INTERNAL_SERVER_ERROR).json({ "error": err.
            message });
            return;
        }
        if (!row) {
            res.status(HTTP_STATUS_NOT_FOUND).json({ "message": "User not found" });
            return;
        }
        res.status(HTTP_STATUS_OK).json(row);
    });
});
```

POST /user Endpoint

- Explanation of the code:
  - This endpoint adds a new user.
  - The user's first name, last name, and department should be sent in the body of the request as JSON.
  - If any of these parameters are missing, it returns a bad request message.

```
app.post('/user', (req, res) => {
    if (!req.body.first || !req.body.last || !req.body.dept) {
        return res.status(HTTP_STATUS_BAD_REQUEST).json({ 'message': 'Bad request.
        Missing required body parameters' });
    }
    let sql = `INSERT INTO Users (first, last, dept) VALUES (?, ?, ?)`;
    let params = [req.body.first, req.body.last, req.body.dept];
    db.run(sql, params, function(err) {
        if (err) {
            res.status(HTTP_STATUS_INTERNAL_SERVER_ERROR).json({ "error": err.
            message });
            return;
        }
        res.status(HTTP_STATUS_OK).json({
            "message": "User added successfully",
            "id": this.lastID
        });
    });
});
```

PATCH /user/:id Endpoint

- Explanation of the code:
  - This endpoint updates the user with the specified ID.
  - The user's new first name, last name, and department should be sent in the body of the request as JSON.
  - If any of these parameters are missing, it returns a bad request message.

```
app.patch('/user/:id', (req, res) => {
    if (!req.body.first || !req.body.last || !req.body.dept) {
        return res.status(HTTP_STATUS_BAD_REQUEST).json({ 'message': 'Bad request.
        Missing required body parameters' });
    }
    let sql = `UPDATE Users SET first = ?, last = ?, dept = ? WHERE id = ?`;
    let params = [req.body.first, req.body.last, req.body.dept, req.params.id];
    db.run(sql, params, function(err) {
        if (err) {
            res.status(HTTP_STATUS_INTERNAL_SERVER_ERROR).json({ "error": err.
            message });
            return;
        }
        res.status(HTTP_STATUS_OK).json({
            "message": "User updated successfully",
            "changes": this.changes
        });
    });
});
```

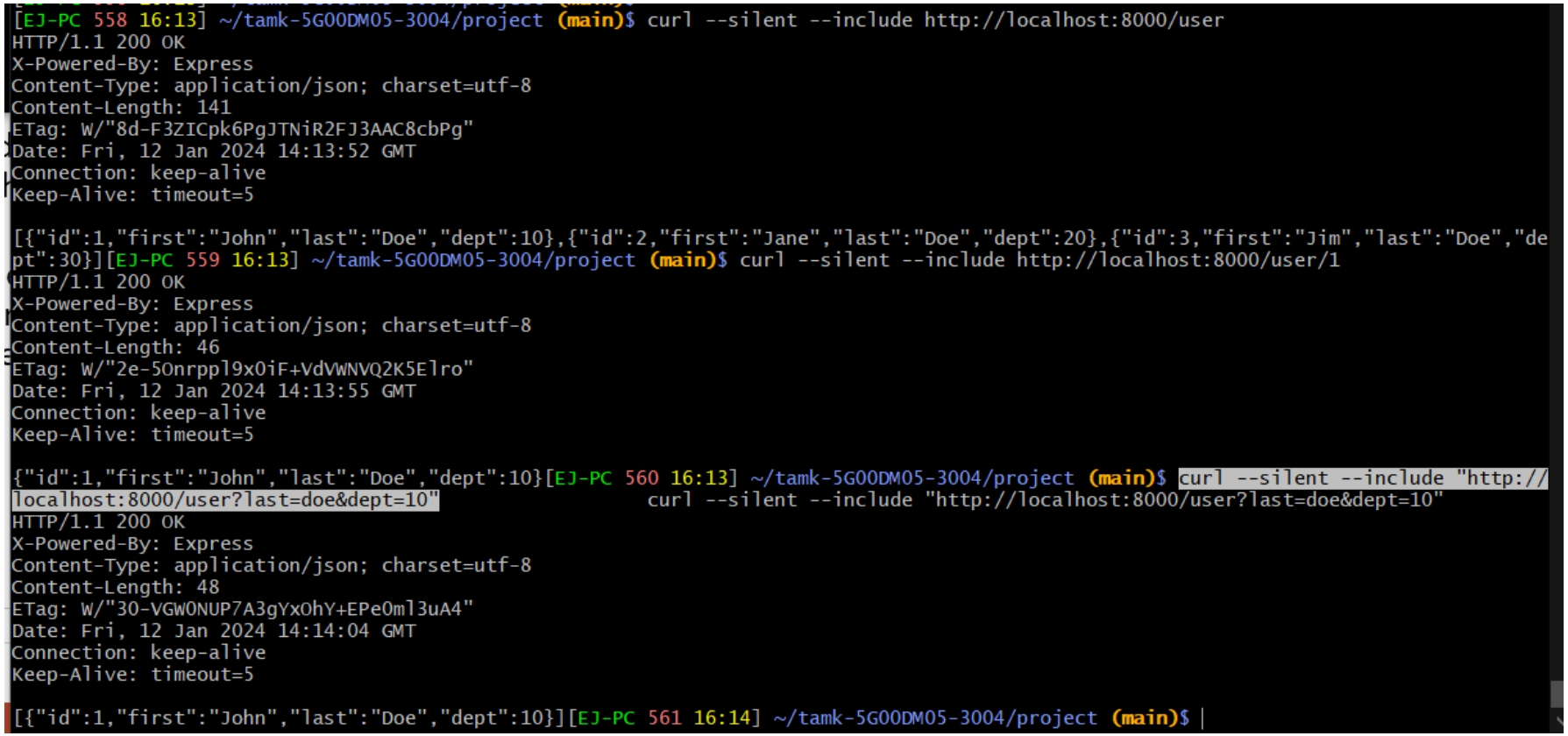
DELETE /user/:id Endpoint

- Explanation of the code:
  - This endpoint deletes the user with the specified ID.
  - If the ID parameter is missing, it returns a bad request message.

```
app.delete('/user/:id', (req, res) => {
    let sql = `DELETE FROM Users WHERE id = ?`;
    let params = [req.params.id];
    db.run(sql, params, function(err) {
        if (err) {
            res.status(HTTP_STATUS_INTERNAL_SERVER_ERROR).json({ "error": err.
            message });
            return;
        }
        res.status(HTTP_STATUS_OK).json({
            "message": "User deleted successfully",
            "changes": this.changes
        });
    });
});
```

Test Calls to API

- Explanation of the requests and responses:
  - curl --silent --include http://localhost:8000/user: This command retrieves all users. The server responds with a JSON array of users.
  - curl --silent --include http://localhost:8000/user/1: This command retrieves the user with ID 1. The server responds with a JSON object of the user's data.
  - curl --silent --include http://localhost:8000/user?last=doe&dept=10: This command retrieves users with the last name "doe" and department number 10. The server responds with a JSON array of matching users.



Testing Other Endpoints

- **POST /user Endpoint Test**
  - Command: curl --header "Content-Type: application/json" --request POST --data '{"first":"Peter","last":"Parker","dept":10}' http://localhost:8000/user
  - Explanation: This command tests the POST /user endpoint. It sends a POST request to add a new user named Peter Parker in department 10.
- **PATCH /user/:id Endpoint Test**
  - Command: curl --header "Content-Type: application/json" --request PATCH --data '{"first":"Peter","last":"DOE","dept":10}' http://localhost:8000/user/4
  - Explanation: This command tests the PATCH /user/:id endpoint. It sends a PATCH request to update the user with ID 4, changing their last name to DOE.
- **DELETE /user/:id Endpoint Test**
  - Command: curl --request DELETE http://localhost:8000/user/4
  - Explanation: This command tests the DELETE /user/:id endpoint. It sends a DELETE request to remove the user with ID 4 from the database.

```
[EJ-PC 567 16:30] ~/tamk-5G00DM05-3004/project (main)$ curl --silent --include http://localhost:8000/user
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 141
ETag: W/"8d-F3ZICpk6PgJTNiR2FJ3AAC8cbPg"
Date: Fri, 12 Jan 2024 14:30:14 GMT
Connection: keep-alive
Keep-Alive: timeout=5

[{"id":1,"first":"John","last":"Doe","dept":10},{"id":2,"first":"Jane","last":"Doe","dept":20},{"id":3,"first":"Jim","last":"Doe","de
pt":30}][EJ-PC 568 16:30] ~/tamk-5G00DM05-3004/project (main)$
[EJ-PC 568 16:30] ~/tamk-5G00DM05-3004/project (main)$
[EJ-PC 568 16:30] ~/tamk-5G00DM05-3004/project (main)$ curl --header "Content-Type: application/json" --request POST --data '{"first"
:"Peter","last":"Parker","dept":10}' http://localhost:8000/user
{"message":"User added successfully","id":4}[EJ-PC 569 16:30] ~/tamk-5G00DM05-3004/project (main)$
[EJ-PC 569 16:30] ~/tamk-5G00DM05-3004/project (main)$ curl --header "Content-Type: application/json" --request PATCH --data '{"first
":"Peter","last":"DOE","dept":10}' http://localhost:8000/user/4
{"message":"User updated successfully","changes":1}[EJ-PC 570 16:31] ~/tamk-5G00DM05-3004/project (main)$
[EJ-PC 570 16:31] ~/tamk-5G00DM05-3004/project (main)$ curl --silent --include http://localhost:8000/user
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 189
ETag: W/"bd-4swwfjncyqrgvqr5kvYF2yMaarA"
Date: Fri, 12 Jan 2024 14:31:21 GMT
Connection: keep-alive
Keep-Alive: timeout=5

[{"id":1,"first":"John","last":"Doe","dept":10},{"id":2,"first":"Jane","last":"Doe","dept":20},{"id":3,"first":"Jim","last":"Doe","de
pt":30},{"id":4,"first":"Peter","last":"DOE","dept":10}][EJ-PC 571 16:31] ~/tamk-5G00DM05-3004/project (main)$ curl --request DELETE
curl --request DELETE http://localhost:8000/user/4
{"message":"User deleted successfully","changes":1}[EJ-PC 572 16:31] ~/tamk-5G00DM05-3004/project (main)$ curl --silent --include htt
curl --silent --include http://localhost:8000/user
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 141
ETag: W/"8d-F3ZICpk6PgJTNiR2FJ3AAC8cbPg"
Date: Fri, 12 Jan 2024 14:31:45 GMT
Connection: keep-alive
Keep-Alive: timeout=5

[{"id":1,"first":"John","last":"Doe","dept":10},{"id":2,"first":"Jane","last":"Doe","dept":20},{"id":3,"first":"Jim","last":"Doe","de
pt":30}][EJ-PC 573 16:31] ~/tamk-5G00DM05-3004/project (main)$
```

Database

```
[EJ-PC 580 16:37] ~/tamk-5G00DM05-3004/project/sql (main)$ cat create.sql

CREATE TABLE IF NOT EXISTS Users (
    id INTEGER PRIMARY KEY,
    first TEXT NOT NULL,
    last TEXT NOT NULL,
    dept INTEGER
);
[EJ-PC 581 16:37] ~/tamk-5G00DM05-3004/project/sql (main)$ cat insert.sql

INSERT INTO Users (first, last, dept) VALUES ('John', 'Doe', 10);
INSERT INTO Users (first, last, dept) VALUES ('Jane', 'Doe', 20);
INSERT INTO Users (first, last, dept) VALUES ('Jim', 'Doe', 30);
[EJ-PC 582 16:37] ~/tamk-5G00DM05-3004/project/sql (main)$ cat delete.sql

DELETE FROM Users;
```

- **Database Setup**: You set up the database and created the Users table using the command sqlite3 database.db < sql/create.sql. This command creates a SQLite database named database.db and a table named Users.
- **Inserting Sample Data**: To populate the Users table with sample data, you used the command sqlite3 database.db < sql/insert.sql. This command inserts sample data into the Users table.
- **Deleting Data**: To delete all data from the Users table, you used the command sqlite3 database.db < sql/delete.sql. This command deletes all records from the Users table.
- **Note**: These operations directly affect the database and cannot be undone. It’s always important to have a backup of your data.

Summary

Recap of the Project: The project involved creating a RESTful API using Node.js and Express.js, with SQLite as the database. The API supports all necessary HTTP methods (GET, POST, PATCH, DELETE), and follows common API design best practices. The project was divided into three parts, each with increasing complexity and additional features.

Challenges and Solutions: The main challenges were implementing the different HTTP methods and handling errors for invalid or unknown parameters. These were overcome by using Express.js routing to handle different HTTP methods and adding error handling in each API endpoint.

Successful Implementation: The project was successfully completed, with a fully functional API that allows for searching, adding, modifying, and deleting user information in an SQLite database. The API follows common design best practices and handles errors effectively.

Future Improvements: There is always room for improvements. Additional API endpoints could be added to provide more functionality. The application could be deployed to a cloud service to make it accessible from anywhere.