

NodeJS Backend Project

Jingjing Yang

16th Nov 2023

1

Project structure

- README.md: Contains information about the project and instructions.
- database.db: The SQLite database file.
- package-lock.json & package.json: Manage project dependencies.
- project.pdf: The project presentation.
- server.js: The main server file.
- sql/: Contains SQL scripts for creating, deleting, and inserting data into the database.

```
— project
  ├── README.md
  ├── database.db
  ├── package-lock.json
  ├── package.json
  ├── project.pdf
  ├── server.js
  └── sql
      ├── README
      ├── create.sql
      ├── delete.sql
      └── insert.sql
```

Main functionalities

- Create, read, update, and delete data in a SQLite database via a Node.js server.
- The server processes HTTP GET requests to retrieve data, POST requests to add data, UPDATE requests to modify data, and DELETE requests to remove data.
- It also handles errors and sends responses in JSON format.

Technologies Used

- Node.js
- SQLite
- Express.js

Challenges

1. Setting up and managing the SQLite database.
2. Implementing the HTTP methods with proper error handling.
3. Ensuring the server responses are in the correct JSON format.

Solutions

1. Express.js middleware for error handling

Use 'app.use()' to define a middleware function that checks for errors and responds with the appropriate HTTP status code and message.

2. Express.js 'res.json()' method

It automatically sends responses as JSON.

Timeline

1. Create necessary scripts to setup SQLite database
2. Connect to the SQLite database from the Node.js application
3. Implement GET HTTP request to retrieve data from the database
4. Implement POST HTTP request to add data to the database
5. Implement PUT HTTP request to update data in the database
6. Implement DELETE HTTP request to remove data from the database
7. Implement error handling for all HTTP methods
8. Test all endpoints and ensure they work as expected
9. Write documentation for the project, including setup and usage instructions

Code

Jingjing Yang

16th Nov 2023

8

Create a basic Express server connected to a SQLite database

```
const express = require("express");
const sqlite3 = require("sqlite3").verbose();

// Connect to SQLite database
let db = new sqlite3.Database("./database.db");

// Initialize Express app
const app = express();

// Enable to parse JSON body in POST requests
app.use(express.json());

app.listen(8080, () => {
  console.log("Server is running on http://localhost:8080 ...");
});
```

Retrieve existing records (HTTP GET method)

```
// Redirect from '/' to '/clothes'  
app.get("/", (req, res) => {  
  res.redirect("/clothes");  
});  
  
// HTTP GET endpoint  
app.get("/clothes", (req, res) => {  
  db.all("SELECT * FROM Clothes", [], (err, rows) => {  
    if (err) {  
      res.status(500).json({ error: err.message });  
      return;  
    }  
    res.json(rows);  
  });  
});  
  
// HTTP GET endpoint for a specific record  
app.get("/clothes/:id", (req, res) => {  
  const query = "SELECT * FROM Clothes WHERE id = ?";  
  const values = [req.params.id];  
  
  db.get(query, values, (err, row) => {  
    if (err) {  
      res.status(500).json({ error: err.message });  
      return;  
    }  
    res.json(row);  
  });  
});
```

Create a new record (HTTP POST method)

```
// HTTP POST endpoint
app.post("/clothes", (req, res) => {
  const query =
    "INSERT INTO Clothes (name, type, size, color, price) VALUES (?, ?, ?, ?, ?,
?)";
  const values = [
    req.body.name,
    req.body.type,
    req.body.size,
    req.body.color,
    req.body.price,
  ];

  db.run(query, values, function (err) {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json({ id: this.lastID });
  });
});
```

Update an existing record (HTTP PUT method)

```
// HTTP UPDATE endpoint
app.put("/clothes/:id", (req, res) => {
  const query =
    "UPDATE Clothes SET name = ?, type = ?, size = ?, color = ?, price = ? WHERE
  id = ?";
  const values = [
    req.body.name,
    req.body.type,
    req.body.size,
    req.body.color,
    req.body.price,
    req.params.id,
  ];

  db.run(query, values, function (err) {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json({ message: "Row updated", changes: this.changes });
  });
});
```

Delete an existing record (HTTP DELETE method)

```
// HTTP DELETE endpoint
app.delete("/clothes/:id", (req, res) => {
  const query = "DELETE FROM Clothes WHERE id = ?";
  const values = [req.params.id];

  db.run(query, values, function (err) {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json({ message: "Row deleted", changes: this.changes });
  });
});
```

Test calls to backend

Jingjing Yang

16th Nov 2023

14

GET all clothing items

```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8080  
Found. Redirecting to /clothes%
```

```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8080/clothes  
[{"id":1,"name":"Denim Jacket","type":"Jacket","size":"M","color":"Blue","price":59.99}, {"id":2,"name":"Leather Boots","type":"Shoes","size":"8","color":"Black","price":89.99}, {"id":3,"name":"Floral Dress","type":"Dress","size":"S","color":"Multicolor","price":45.99}, {"id":4,"name":"Cotton T-Shirt","type":"Shirt","size":"L","color":"White","price":15.99}]%
```

GET a specific item

```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8080/clothes/1
{"id":1,"name":"Denim Jacket","type":"Jacket","size":"M","color":"Blue","price":59.99}%
```

POST a new item

```
jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: application/json" -d '{"name":"Wool Sweater","type":"Sweater","size":"L","color":"Red","price":60.99}' http://localhost:8080/clothes  
{"id":5}%
```

```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8080/clothes  
[{"id":1,"name":"Denim Jacket","type":"Jacket","size":"M","color":"Blue","price":59.99}, {"id":2,"name":"Leather Boots","type":"Shoes","size":"8","color":"Black","price":89.99}, {"id":3,"name":"Floral Dress","type":"Dress","size":"S","color": "Multicolor","price":45.99}, {"id":4,"name":"Cotton T-Shirt","type":"Shirt","size":"L","color":"White","price":15.99}, {"id":5,"name":"Wool Sweater","type":"Sweater","size":"L","color":"Red","price":60.99}]%
```

PUT to update(replace) an item

```
jingjingyang@jingjings-MacBook-Pro ~ % curl -X PUT -H "Content-Type: application/json" -d '{"name":"Denim Jacket","type":"Jacket","size":"XL","color":"Brown","price":50.99}' http://localhost:8080/clothes/1
{"message":"Row updated","changes":1}%
```

```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8080/clothes/1
{"id":1,"name":"Denim Jacket","type":"Jacket","size":"XL","color":"Brown","price":50.99}%
```

DELETE an item

```
jingjingyang@jingjings-MacBook-Pro ~ % curl -X DELETE http://localhost:3000/clothes/1
{"message":"Row deleted","changes":1}%
```

```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8080/clothes/1
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8080/clothes
[{"id":2,"name":"Leather Boots","type":"Shoes","size":"8","color":"Black","price":89.99}, {"id":3,"name":"Floral Dress","type":"Dress","size":"S","color":"Multi color","price":45.99}, {"id":4,"name":"Cotton T-Shirt","type":"Shirt","size":"L","color":"White","price":15.99}, {"id":5,"name":"Wool Sweater","type":"Sweater","size":"L","color":"Red","price":60.99}]%
```

Test via Postman

Jingjing Yang

16th Nov 2023

20

GET all clothing items

Sample Footer Text

The screenshot shows a UI tool with two parallel API requests. Both requests are GET methods to `http://localhost:8080`. The left request has a status of 200 OK, time 4 ms, size 594 B, and the right request has a status of 200 OK, time 8 ms, size 594 B. Both requests have an empty body. The left request's response is displayed in a JSON viewer, and the right request's response is also displayed in a JSON viewer.

Left Request (Status: 200 OK, Time: 4 ms, Size: 594 B)

```
[{"id": 1, "name": "Denim Jacket", "type": "Jacket", "size": "M", "color": "Blue", "price": 59.99}, {"id": 2, "name": "Leather Boots", "type": "Shoes", "size": "8", "color": "Black", "price": 89.99}, {"id": 3, "name": "Floral Dress", "type": "Dress", "size": "S", "color": "Multicolor", "price": 45.99}, {"id": 4, "name": "Cotton T-Shirt", "type": "Shirt", "size": "L", "color": "White", "price": 15.99}]
```

Right Request (Status: 200 OK, Time: 8 ms, Size: 594 B)

```
[{"id": 1, "name": "Denim Jacket", "type": "Jacket", "size": "M", "color": "Blue", "price": 59.99}, {"id": 2, "name": "Leather Boots", "type": "Shoes", "size": "8", "color": "Black", "price": 89.99}, {"id": 3, "name": "Floral Dress", "type": "Dress", "size": "S", "color": "Multicolor", "price": 45.99}, {"id": 4, "name": "Cotton T-Shirt", "type": "Shirt", "size": "L", "color": "White", "price": 15.99}]
```

GET a specific item

Sample Footer Text

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/clothes/1
- Params:** None
- Query Params:** None
- Body:** Status: 200 OK Time: 7 ms Size: 321 B
- Response Preview:** JSON (Pretty)

```
1 {  
2   "id": 1,  
3   "name": "Denim Jacket",  
4   "type": "Jacket",  
5   "size": "M",  
6   "color": "Blue",  
7   "price": 59.99  
8 }
```

POST a new item

HTTP <http://localhost:8080/clothes> Save No Environment

POST http://localhost:8080/clothes Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Code Cookies

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

```
1 {"name": "Wool Sweater", "type": "Sweater", "size": "L", "color": "Red", "price": 60.99}
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 25 ms Size: 241 B

Pretty Raw Preview JSON

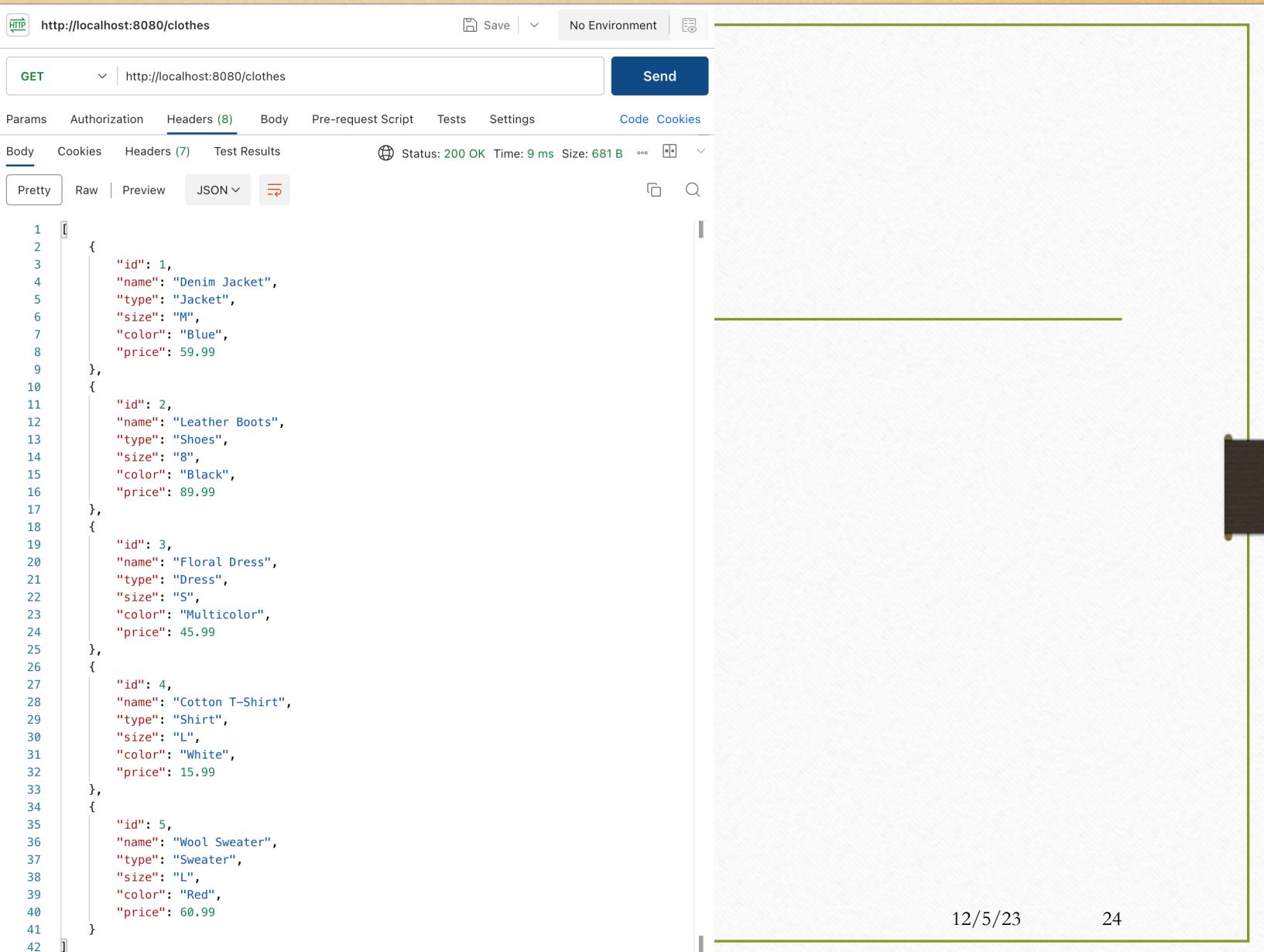
```
1 {
2   "id": 5
3 }
```

Sample Footer Text

Code Cookies

POST a new item

Sample Footer Text



The screenshot shows the Postman application interface. At the top, it displays the URL `http://localhost:8080/clothes`. Below the URL, there are tabs for `GET`, `Params`, `Authorization`, `Headers (8)`, `Body`, `Pre-request Script`, `Tests`, `Settings`, `Code`, and `Cookies`. The `Headers (8)` tab is currently selected. Under the `Body` tab, there are options for `Pretty`, `Raw`, `Preview`, and `JSON`. The `JSON` option is selected, and the response is displayed as a JSON array:

```
[{"id": 1, "name": "Denim Jacket", "type": "Jacket", "size": "M", "color": "Blue", "price": 59.99}, {"id": 2, "name": "Leather Boots", "type": "Shoes", "size": "8", "color": "Black", "price": 89.99}, {"id": 3, "name": "Floral Dress", "type": "Dress", "size": "S", "color": "Multicolor", "price": 45.99}, {"id": 4, "name": "Cotton T-Shirt", "type": "Shirt", "size": "L", "color": "White", "price": 15.99}, {"id": 5, "name": "Wool Sweater", "type": "Sweater", "size": "L", "color": "Red", "price": 60.99}]
```

At the bottom right of the interface, there is a status bar showing the date `12/5/23` and page number `24`.

PUT to update an item

The screenshot shows the Postman application interface for making a PUT request to update a clothes item.

Request Headers:

Key	Value
Content-Type	application/json
Key	Value

Request URL: http://localhost:8080/clothes/1

Method: PUT

Body (raw JSON):

```
1  {"name": "Denim Jacket", "type": "Jacket", "size": "XL", "color": "Brown", "price": 50.99}
```

Response Status: 200 OK

Response Body (Pretty JSON):

```
1  {
2      "message": "Row updated",
3      "changes": 1
4  }
```

Sample Footer Text

PUT to update an item

Sample Footer Text

The screenshot shows the Postman application interface. At the top, the URL `http://localhost:8080/clothes/1` is entered into the address bar, along with the method `GET`. To the right are buttons for `Save`, `No Environment`, and a gear icon. Below the address bar is a search bar containing the same URL and method. A large blue `Send` button is positioned to the right of the search bar.

Below the search bar, there are several tabs: `Params`, `Authorization`, `Headers (8)` (which is currently selected), `Body`, `Pre-request Script`, `Tests`, and `Settings`. To the right of these are links for `Code` and `Cookies`.

The `Headers` section displays the following table:

	Key	Value
<input type="checkbox"/>	Content-Type	application/json
	Key	Value

Below the headers, there are tabs for `Body`, `Cookies`, `Headers (7)`, and `Test Results`. The `Body` tab is selected. It contains three tabs: `Pretty` (selected), `Raw`, and `Preview`. To the right of the body tabs, status information is displayed: `Status: 200 OK Time: 4 ms Size: 323 B`. There are also icons for copy, search, and refresh.

The `Pretty` tab shows the JSON response with line numbers:

```
1 {  
2   "id": 1,  
3   "name": "Denim Jacket",  
4   "type": "Jacket",  
5   "size": "XL",  
6   "color": "Brown",  
7   "price": 50.99  
8 }
```

DELETE an item

HTTP <http://localhost:8080/clothes/1> Save | No Environment |

DELETE ▾ http://localhost:8080/clothes/1 **Send**

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

Headers

	Key	Value
<input type="checkbox"/>	Content-Type	application/json
	Key	Value

Body Cookies Headers (7) Test Results Status: 200 OK Time: 5 ms Size: 272 B ...

Pretty Raw Preview JSON

```
1 {  
2   "message": "Row deleted",  
3   "changes": 1  
4 }
```

Sample Footer Text

DELETE an item

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:8080/clothes/1`. The method is selected as `DELETE`. Below the URL, there are tabs for `Params`, `Authorization`, `Headers (8)`, `Body`, `Pre-request Script`, `Tests`, and `Settings`. The `Headers (8)` tab is currently active. It displays a table with two rows. The first row has a checkbox next to "Content-Type" and its value is "application/json". The second row has a column labeled "Key" and another labeled "Value". In the "Body" section, there are tabs for `Body`, `Cookies`, `Headers (6)`, and `Test Results`. The `Body` tab is active. Below this, there are buttons for `Pretty`, `Raw`, `Preview`, and `JSON`. The `Test Results` section shows a status of `200 OK` with a time of `5 ms` and a size of `192 B`. At the bottom left, there is a footer note: `Sample Footer Text`.

Summary

Jingjing Yang

16th Nov 2023

29

-
- In conclusion, I have successfully implemented a backend server using Node.js and SQLite in this project. I created a basic HTTP server, handled different types of HTTP requests, and connected to a SQLite database. I ensured to incorporate error handling and provide responses in JSON format, which are crucial for effective communication with clients. This project has been a foundational step in my journey towards mastering backend development.