

# NodeJS Backend Project

---

Jingjing Yang

16<sup>th</sup> 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

16<sup>th</sup> Nov 2023

8

# Create a basic Express server connected to a SQLite database

---

```
1 const express = require("express"); // Import Express
2 const app = express(); // Initialize Express app
3 app.use(express.json()); // Enable to parse JSON body in POST requests
4
5 const sqlite3 = require("sqlite3").verbose(); // Import SQLite
6 let db = new sqlite3.Database("./database.db"); // Connect to SQLite database
7 const table = "Clothes"; // Define table name
8
139 app.listen(8080, () => {
140   // Start server
141   console.log(`Server is running on http://localhost:8080 ...`);
142 });


```

# Define variables in the beginning

---

```
9 // Define HTTP status codes
10 const HTTP_STATUS_OK = 200;
11 const HTTP_STATUS_CREATED = 201;
12 const HTTP_STATUS_NO_CONTENT = 204;
13 const HTTP_STATUS_INTERNAL_SERVER_ERROR = 500;
14
15 // Define SQL queries
16 const SELECT_ALL = `SELECT * FROM ${table}`;
17 const SELECT_ID = `SELECT * FROM ${table} WHERE id = ?`;
18 const INSERT_INTO = `INSERT INTO ${table} (name, type, size, color, price) VALUES (?, ?, ?, ?, ?)`;
19 const UPDATE = `UPDATE ${table} SET name = ?, type = ?, size = ?, color = ?, price = ? WHERE id = ?`;
20 const DELETE = `DELETE FROM ${table} WHERE id = ?`;
21
```

# error response

---

```
22 // Function to create error response
23 function createErrorResponse(err) {
24   return {
25     error: err.message, // Error message
26     code: err.code, // Error code
27     details: {
28       /* any relevant details */
29     },
30     stack: process.env.NODE_ENV === "development" ? err.stack : "hidden", // Stack trace (only in development)
31   };
32 }
33
```

# Retrieve existing records (HTTP GET method)

---

```
34 app.get("/", (req, res) => {
35   // Root endpoint
36   res.redirect("/clothes"); // Redirect to '/clothes'
37 });
38
39 app.get('/clothes', (req, res) => {
40   // GET endpoint for all records
41   db.all(SELECT_ALL, [], (err, rows) => {
42     // Execute SQL query
43     if (err) {
44       // If error occurs
45       res
46         .status(HTTP_STATUS_INTERNAL_SERVER_ERROR)
47         .json(createErrorResponse(err)); // Send error response
48       return;
49     }
50     res.json(rows); // Send response with all rows
51   });
52 });
53
```

# Retrieve a specific record (HTTP GET method)

---

```
54 app.get(`/clothes/:id`, (req, res) => {
55   // GET endpoint for a specific record
56   const values = [req.params.id]; // ID parameter from URL
57
58   db.get(SELECT_ID, values, (err, row) => {
59     // Execute SQL query
60     if (err) {
61       // If error occurs
62       res
63         .status(HTTP_STATUS_INTERNAL_SERVER_ERROR)
64         .json(createErrorResponse(err)); // Send error response
65       return;
66     }
67     res.json(row); // Send response with the row
68   });
69 });
70
```

# Create a new record (HTTP POST method)

---

```
71 app.post(`/clothes`, (req, res) => {
72   // POST endpoint
73   const values = [
74     req.body.name, // Values from request body
75     req.body.type,
76     req.body.size,
77     req.body.color,
78     req.body.price,
79   ];
80
81   db.run(INSERT_INTO, values, function (err) {
82     // Execute SQL query
83     if (err) {
84       // If error occurs
85       res
86         .status(HTTP_STATUS_INTERNAL_SERVER_ERROR)
87         .json(createErrorResponse(err)); // Send error response
88       return;
89     }
90     res.status(HTTP_STATUS_CREATED).json({ id: this.lastID }); // Send response with created ID
91   });
92 });
93
```

# Update an existing record (HTTP PUT method)

---

```
94 app.put('/clothes/:id', (req, res) => {
95   // PUT endpoint
96   const values = [
97     req.body.name, // Values from request body and ID from URL
98     req.body.type,
99     req.body.size,
100    req.body.color,
101    req.body.price,
102    req.params.id,
103  ];
104
105  db.run(UPDATE, values, function (err) {
106    // Execute SQL query
107    if (err) {
108      // If error occurs
109      res
110        .status(HTTP_STATUS_INTERNAL_SERVER_ERROR)
111        .json(createErrorResponse(err)); // Send error response
112      return;
113    }
114    res
115      .status(HTTP_STATUS_OK)
116      .json({ message: "Row updated", changes: this.changes }); // Send response with update message
117  });
118});
119
```

# Create a new record (HTTP POST method)

---

```
120 app.delete(`/clothes/:id`, (req, res) => {
121   // DELETE endpoint
122   const values = [req.params.id]; // ID parameter from URL
123
124   db.run(DELETE, values, function (err) {
125     // Execute SQL query
126     if (err) {
127       // If error occurs
128       res
129         .status(HTTP_STATUS_INTERNAL_SERVER_ERROR)
130         .json(createErrorResponse(err)); // Send error response
131       return;
132     }
133     res
134       .status(HTTP_STATUS_NO_CONTENT)
135       .json({ message: "Row deleted", changes: this.changes }); // Send response with delete message
136     });
137   });
138 }
```

# Delete an existing record (HTTP DELETE method)

---

## 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

# 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}]%
```

# Include the HTTP response headers in curl

# GET all clothing items

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include http://loc  
alhost:8080  
HTTP/1.1 302 Found  
X-Powered-By: Express  
Location: /clothes  
Vary: Accept  
Content-Type: text/plain; charset=utf-8  
Content-Length: 30  
Date: Tue, 12 Dec 2023 10:40:59 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5  
  
Found. Redirecting to /clothes%
```

# GET all clothing items

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include http://loc
alhost:8080/clothes
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 357
ETag: W/"165-20EAdHwAIDZ0E8V902NHHPCAnwE"
Date: Tue, 12 Dec 2023 10:29:28 GMT
Connection: keep-alive
Keep-Alive: timeout=5

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

# GET a specific item

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include http://loc  
alhost:8080/clothes/1  
HTTP/1.1 200 OK  
X-Powered-By: Express  
Content-Type: application/json; charset=utf-8  
Content-Length: 86  
ETag: W/"56-aGu82n/Q/4/aqfNMoVuUmqfb98w"  
Date: Tue, 12 Dec 2023 10:30:44 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5  
  
{"id":1,"name":"Denim Jacket","type":"Jacket","size":"M","color":"Blue","price":  
59.99}%
```

# GET a non-existing resource

(Content-Length: 0)

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include http://  
localhost:8080/clothes/88  
HTTP/1.1 200 OK  
X-Powered-By: Express  
Content-Type: application/json; charset=utf-8  
Date: Tue, 12 Dec 2023 10:54:35 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5  
Content-Length: 0
```

# POST a new item

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include -X POST -H "Content-Type: application/json" -d '{"name":"Wool Sweater","type":"Sweater","size":"L","color":"Red","price":60.99}' http://localhost:8080/clothes
HTTP/1.1 201 Created
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 8
ETag: W/"8-4U72Xq+t/P6Xscj7oQblSpGEbo"
Date: Tue, 12 Dec 2023 10:49:24 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"id":5}%
```

# PUT to update(replace) an item

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include -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
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 37
ETag: W/"25-JB1Tin60AHjLd750N4t0/Q1kVD8"
Date: Tue, 12 Dec 2023 10:31:07 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Row updated","changes":1}%
```

# PUT with invalid data

("changes":0)

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include -X PUT  
-H "Content-Type: application/json" -d '{"name":"Denim Jacket","type":"Jacke  
t","size":"XL","color":"Brown","price":50.99}' http://localhost:8080/clothes/  
88  
HTTP/1.1 200 OK  
X-Powered-By: Express  
Content-Type: application/json; charset=utf-8  
Content-Length: 37  
ETag: W/"25-H9XUMJUwB1GDqSwx5WsIscmVTlk"  
Date: Tue, 12 Dec 2023 10:59:55 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5  
  
{"message":"Row updated","changes":0}%
```

# DELETE an item

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include -X DELETE  
http://localhost:8080/clothes/1  
HTTP/1.1 204 No Content  
X-Powered-By: Express  
ETag: W/"25-F8iueZV101Skkf6zApRwls6i7iU"  
Date: Tue, 12 Dec 2023 10:32:35 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5
```

# Test via Postman

# GET all clothing items

Sample Footer Text

The image shows a screenshot of the Postman application interface. It displays two separate API requests side-by-side.

**Request 1 (Left):** A GET request to `http://localhost:8080`. The response status is 200 OK, time taken is 4 ms, and size is 594 B. The response body is a JSON array containing four clothing items:

```
[{"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}]
```

**Request 2 (Right):** A GET request to `http://localhost:8080/clothes`. The response status is 200 OK, time taken is 8 ms, and size is 594 B. The response body is identical to the first request, also a JSON array of four clothing items:

```
[{"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 a Postman interface with a green header bar. The URL in the address bar is `http://localhost:8080/clothes`. The method dropdown shows `GET`. Below the address bar, there are tabs for `Params`, `Authorization`, `Headers (8)`, `Body`, `Pre-request Script`, `Tests`, `Settings`, `Code`, and `Cookies`. The `Headers (8)` tab is selected. Under the `Body` tab, there are buttons for `Pretty`, `Raw`, `Preview`, and `JSON`. The `JSON` button is selected. The response body is displayed as a JSON array of five objects:

```
[{"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}]
```

The status bar at the bottom right of the Postman window shows `Status: 200 OK Time: 9 ms Size: 681 B`.

# PUT to update an item

The screenshot shows the Postman application interface with the following details:

- Request URL:** http://localhost:8080/clothes/1
- Method:** PUT
- Headers:** Content-Type: application/json
- 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  }
```

# 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:

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

Below the headers, the `Body` tab is active. It shows a status message: `Status: 200 OK Time: 5 ms Size: 192 B`. There are buttons for `Pretty`, `Raw`, `Preview`, and `JSON`. A small number `1` is visible at the bottom left of the body section.

Sample Footer Text

# Test the various errors

---

# POST with invalid data

## (HTTP\_STATUS\_INTERNAL\_SERVER\_ERROR = 500)

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include -X POST  
-H "Content-Type: application/json" -d '{"Name":"Wool Sweater","type":"Sweate  
r","size":"L","color":"Red","price":60.99}' http://localhost:8080/clothes  
HTTP/1.1 500 Internal Server Error  
X-Powered-By: Express  
Content-Type: application/json; charset=utf-8  
Content-Length: 128  
ETag: W/"80-B/PeHB4EmU5cnsDI1dkBPK43RQ0"  
Date: Tue, 12 Dec 2023 11:01:55 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5  
  
{"error":"SQLITE_CONSTRAINT: NOT NULL constraint failed: Clothes.name","cod  
e":"SQLITE_CONSTRAINT","details":{},"stack":"hidden"}%
```

# DELETE a non-existing resource

(HTTP\_STATUS\_NO\_CONTENT = 204)

---

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl --silent --include -X DELETE http://localhost:8080/clothes/88
HTTP/1.1 204 No Content
X-Powered-By: Express
ETag: W/"25-EMgarNurQ4RT+HU/18cr5LeBED4"
Date: Tue, 12 Dec 2023 11:02:23 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

# Summary

Jingjing Yang

16<sup>th</sup> Nov 2023

49

- 
- 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.