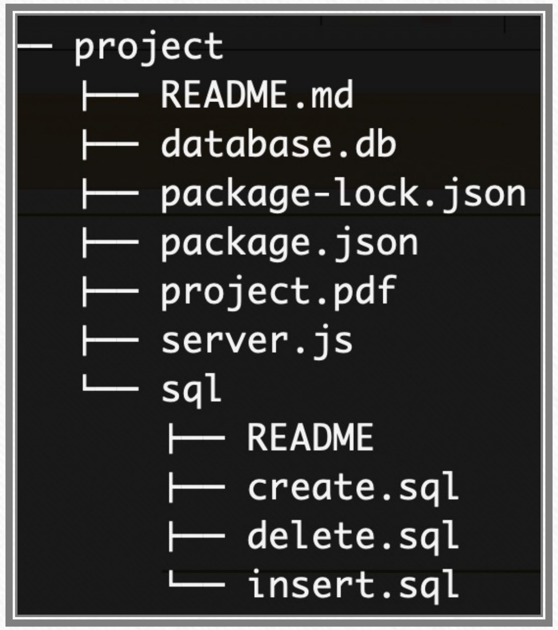
NodeJS Backend Project

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

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(3000, () => {
  console.log("Server is running on <http://localhost:3000> ...");
});
```

Retrieve existing records (HTTP GET method)

```javascript
// 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 });
  });
});

app.listen(3000, () => {
  console.log("Server is running on <http://localhost:3000> ...");
});
```

Test calls to backend

GET all clothing items

```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:3000
Found. Redirecting to /clothes%

jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:3000/clothes
[{"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

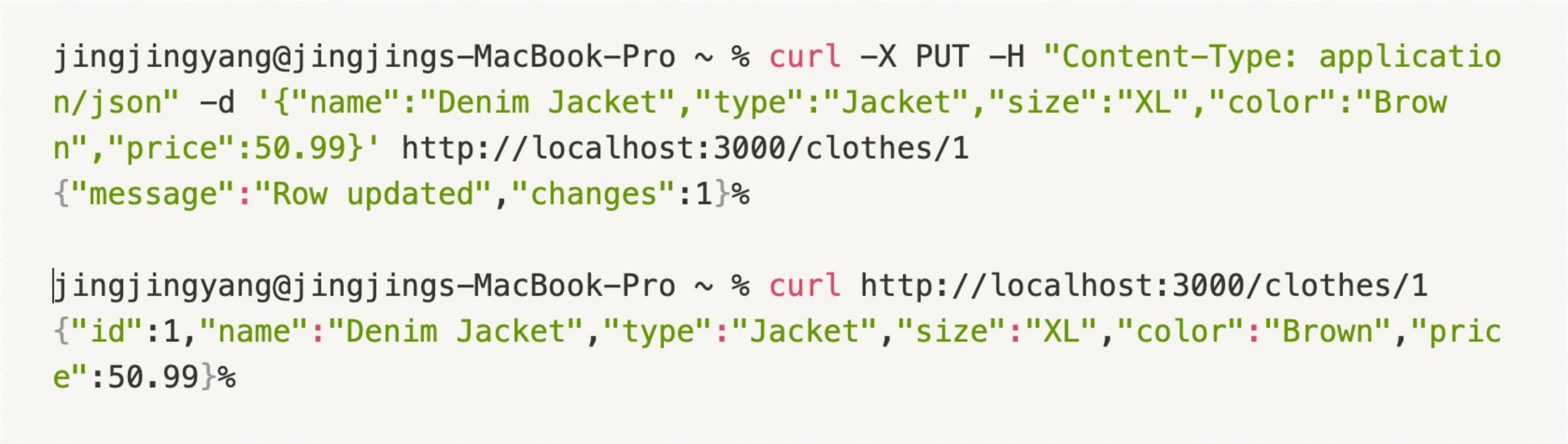
```
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:3000/clothes/1
{"id":1,"name":"Denim Jacket","type":"Jacket","size":"M","color":"Blue","price":
59.99}%
```

POST a new item

```
jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: applicatio
n/json" -d '{"name":"Wool Sweater","type":"Sweater","size":"L","color":"Red","pr
ice":60.99}' http://localhost:3000/clothes
{"id":5}%

jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:3000/clothes
[{"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},{"id":5,"name":"Wool Sweater","type":"Sw
eater","size":"L","color":"Red","price":60.99}]%
```

PUT to update an item



```
jingjingyang@jingjings-MacBook-Pro ~ % curl -X PUT -H "Content-Type: applicatio
n/json" -d '{"name":"Denim Jacket","type":"Jacket","size":"XL","color":"Brow
n","price":50.99}' http://localhost:3000/clothes/1
{"message":"Row updated","changes":1}%

jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:3000/clothes/1
{"id":1,"name":"Denim Jacket","type":"Jacket","size":"XL","color":"Brown","pric
e":50.99}%
```

DELETE an item

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

jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:3000/clothes/1
jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:3000/clothes
[{"id":2,"name":"Leather Boots","type":"Shoes","size":"8","color":"Black","pric
e":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","siz
e":"L","color":"White","price":15.99},{"id":5,"name":"Wool Sweater","type":"Swea
ter","size":"L","color":"Red","price":60.99}]%
```

Summary

- 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.