crud_operation_node js

Here’s a step-by-step guide on how to create basic CRUD operations using Node.js with Express and a MySQL database:

1. Setup Environment

Install the necessary packages:

```
npm init -y
npm install express mysql2 body-parser
```

2. Create a Simple Project Structure

```
your_project/
├── node_modules/
├── app.js
├── db.js
├── package.json
```

3. Setup Database Connection (`db.js`)

In this file, we will set up a connection to MySQL:

```
const mysql = require('mysql2');

const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'your_password',
  database: 'your_database',
});
```

```
db.connect((err) => {
  if (err) {
    console.error('Database connection error: ', err);
    return;
  }
  console.log('Connected to the MySQL database');
});

module.exports = db;
```

4. Create Express Server (`app.js`)

```
const express = require('express');
const bodyParser = require('body-parser');
const db = require('./db');

const app = express();
app.use(bodyParser.json());

// CRUD routes will go here

const port = 3000;
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

5. Create CRUD Routes

5.1 Create (POST) - Add a new item

```
app.post('/items', (req, res) => {
  const { name, description } = req.body;
  const sql = 'INSERT INTO items (name, description) VALUES
(?, ?)';
```

```
  db.query(sql, [name, description], (err, result) => {
    if (err) {
      return res.status(500).send(err);
    }
    res.status(201).json({ id: result.insertId, name, descrip
tion });
  });
});
```

5.2 Read (GET) - Get all items

```
app.get('/items', (req, res) => {
  const sql = 'SELECT * FROM items';
  db.query(sql, (err, results) => {
    if (err) {
      return res.status(500).send(err);
    }
    res.status(200).json(results);
  });
});
```

5.3 Read (GET) - Get item by ID

```
app.get('/items/:id', (req, res) => {
  const { id } = req.params;
  const sql = 'SELECT * FROM items WHERE id = ?';
  db.query(sql, [id], (err, result) => {
    if (err) {
      return res.status(500).send(err);
    }
    if (result.length === 0) {
      return res.status(404).send('Item not found');
    }
    res.status(200).json(result[0]);
```

```
  });
});
```

5.4 Update (PUT) - Update item by ID

```
app.put('/items/:id', (req, res) => {
  const { id } = req.params;
  const { name, description } = req.body;
  const sql = 'UPDATE items SET name = ?, description = ? WHE
RE id = ?';
  db.query(sql, [name, description, id], (err, result) => {
    if (err) {
      return res.status(500).send(err);
    }
    if (result.affectedRows === 0) {
      return res.status(404).send('Item not found');
    }
    res.status(200).json({ id, name, description });
  });
});
```

5.5 Delete (DELETE) - Delete item by ID

```
app.delete('/items/:id', (req, res) => {
  const { id } = req.params;
  const sql = 'DELETE FROM items WHERE id = ?';
  db.query(sql, [id], (err, result) => {
    if (err) {
      return res.status(500).send(err);
    }
    if (result.affectedRows === 0) {
      return res.status(404).send('Item not found');
    }
    res.status(200).send('Item deleted');
```

```
  });
});
```

6. Run the Application

Start the application:

```
node app.js
```

Now you can test the CRUD operations using Postman or curl to interact with your API.

Sample Requests:

- **Create (POST)**: `POST /items` with JSON body `{"name": "Item1", "description": "Description1"}`
- **Read All (GET)**: `GET /items`
- **Read by ID (GET)**: `GET /items/:id`
- **Update by ID (PUT)**: `PUT /items/:id` with JSON body `{"name": "Updated Item", "description": "Updated Description"}`
- **Delete by ID (DELETE)**: `DELETE /items/:id`

SQL for Table Creatio

```
CREATE TABLE items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Explanation:

- `id`: A primary key that auto-increments with each new row.

- `name`: A string (VARCHAR) with a maximum length of 255 characters, cannot be null.
- `description`: A text field to store item descriptions.
- `created_at`: Automatically records the timestamp of when the row was created, using `CURRENT_TIMESTAMP`.

How to Create the Table:

1. Log in to your MySQL server:

```
mysql -u root -p
```

2. Switch to the appropriate database:

```
USE your_database_name;
```

3. Run the `CREATE TABLE` command:

```
CREATE TABLE items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

4. Verify the table creation:

```
SHOW TABLES;
```

Now, the `items` table is created and ready for CRUD operations from the Node.js backend.