Build a REST API with Node.js SQLite and Express JS

To-do list App:

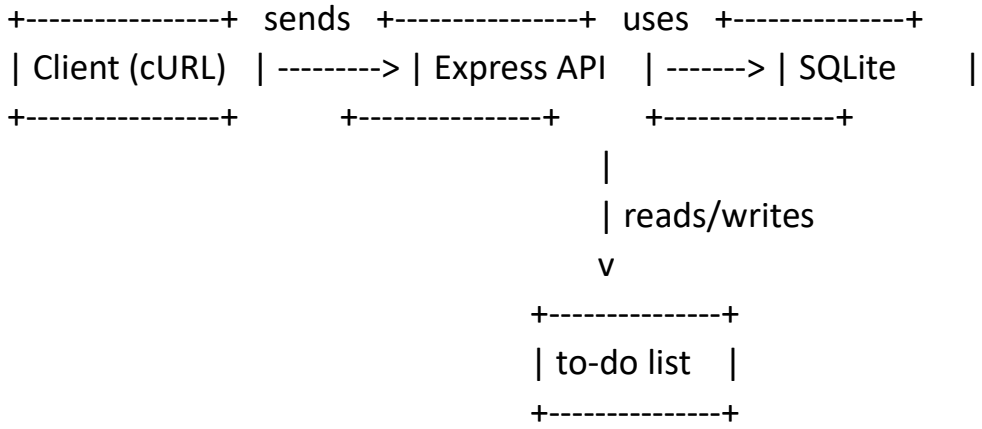
This code is for building a REST API using Node.js, SQLite, and Express JS. An API is like a menu at a restaurant - it tells you what you can order and how to order it.

The API we're building here is for a to-do list app. You can use it to add things to your to-do list, see what's on your list, change something on your list, and remove something from your list.

We use a package called "express" to create the API. It makes it easy for us to handle incoming requests and send responses.

We use another package called "sqlite3" to store our to-do list items. SQLite is a database that can store lots of information in a structured way.

We also use a package called "body-parser" to help us read the data sent in the request. This data is used to add, update, or remove items from our to-do list.



In the code, we start by setting up the "express" package and connecting to the SQLite database. Then, we define four endpoints for our API:

- `GET /todos`: Returns a list of all to-do items
- `GET /todos/:id`: Returns a single to-do item with a specific ID
- `POST /todos`: Adds a new to-do item to the list
- `PUT /todos/:id`: Updates a to-do item with a specific ID

Each endpoint has a specific purpose and uses different SQLite commands to interact with the database.

And that's it! With these endpoints, we have a simple REST API for our to-do list app.

Here's what you need to do:

1. Start by installing the necessary dependencies:

```
npm install express sqlite3 body-parser
```

2. Next, create a new file called `app.js` and import the dependencies:

3. Next, define the endpoints for your API. For this example, let's create a simple API for managing a to-do list:

```
// 1st Installing dependencies

//2nd create express app

import express from 'express';
import  sqlite3  from 'sqlite3';
import bodyParser from 'body-parser';
const app = express();

//Pacakge body-parser
// read the data send in the request

//3rd Task Connecting to database
// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json
app.use(bodyParser.json())

// create a new database
```

```
const db = new sqlite3.Database(':memory:', (err) => {
  if (err) {
    return console.error(err.message);
  }
  console.log('Connected to the in-memory SQlite database.');
});

// create the todos table
db.run(`
CREATE TABLE IF NOT EXISTS todos (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  task TEXT NOT NULL,
  time NUMBER
)
`);

//4th Part Define endpoints

// •    GET /todos: Returns a list of all to-do items
// •    GET /todos/:id: Returns a single to-do item with a specific ID
// •    POST /todos: Adds a new to-do item to the list
// •    PUT /todos/:id: Updates a to-do item with a specific ID


//Root endpoint
//Callback function -It has two parameteres- dependent to each other
//Arrow functions-It does not have any difference in usage- we dont have to
use this and arguments variables.
// app.get('/', (req,res)=> {
//     res.send('Hello World from Express!');
// });

// GET /todos: Returns a list of all to-do items


app.get('/todos',(req,res)=>{
    db.all('SELECT *  FROM todos',(err,rows)=>{
        if(err){
            res.status(500).json({error:err.message});
            return;
        }
        res.json({
            todos:rows
        });
    });
});
//POST /todos: Adds a new to-do item to the list

app.post('/todos', (req, res) => {
```

```
    const task = req.body.task;
    if (!task) {
      res.status(400).json({ error: 'Task is required' });
      return;
    }
    db.run('INSERT INTO todos (task) VALUES (?)', [task], function(err) {
      if (err) {
        res.status(500).json({ error: err.message });
        return;
      }
      res.json({
        id: this.lastID,
        task: task
      });
    });
  });


  //In order to post method we have to add JSON in payload in the new request
// Select "POST" as the request method in thunder client and enter the URL
http://localhost:3000/todos.

// In the "Body" section, select "raw" and change the format to "JSON
(application/json)".

// Enter the following JSON payload in the text area:
// {
//     "task": "Take out the trash"
//   }
// Click on the "Send" button to send the request.




// DELETE/todos/:id: DELETE item with a specific ID
app.delete('/todos/:id', (req, res) => {
    const id = req.params.id;
    db.run('DELETE FROM todos WHERE id = ?', [id], function(err) {
      if (err) {
        res.status(500).json({ error: err.message });
        return;
      }
      res.json({
        message: 'To-do item deleted successfully'
      });
    });
  });
```

```
// PUT /todos/:id: Updates a to-do item with a specific ID

app.put('/todos/:id', (req, res) => {
    const id = req.params.id;
    const task = req.body.task;
    if (!task) {
      res.status(400).json({ error: 'Task is required' });
      return;
    }
    db.run('UPDATE todos SET task = ? WHERE id = ?', [task, id], function(err)
{
      if (err) {
        res.status(500).json({ error: err.message });
        return;
      }
      res.json({
        message: 'To-do item updated successfully'
      });
    });
  });


app.listen(3000);
```