Setting Up Node.js Project

This guide will walk you through the process of setting up a Node.js project with a PEAN stack (PostgreSQL, Express, Angular, Node.js) and Docker.

Prerequisites

Before you begin, make sure you have the following installed on your system:

- Docker: Follow the installation instructions for your operating system from the official Docker website (https://www.docker.com/get-started).

Step 1: Create `docker-compose.yml` file

1. Create a file named `docker-compose.yml` in your project folder.
2. Copy and paste the following code into the `docker-compose.yml` file:

```
version: '3'
services:
  parse:
    image: parseplatform/parse-server
    environment:
      PARSE_SERVER_APPLICATION_ID: myAppId
      PARSE_SERVER_MASTER_KEY: myMasterKey
      PARSE_SERVER_DATABASE_URI: postgresql://postgres:myPassword@postgres-
db:5432/myDatabase
      PARSE_SERVER_SERVER_URL: http://localhost:1337/parse
      PARSE_SERVER_MASTER_KEY_IPS: 0.0.0.0/0
    ports:
      - 1337:1337
    depends_on:
      - postgres-db
  postgres-db:
    image: postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: myPassword
      POSTGRES_DB: myDatabase
  parse-dashboard:
    image: parseplatform/parse-dashboard
    environment:
      PARSE_DASHBOARD_ALLOW_INSECURE_HTTP: true
      PARSE_DASHBOARD_SERVER_URL: http://localhost:1337/parse
      PARSE_DASHBOARD_APP_ID: myAppId
      PARSE_DASHBOARD_MASTER_KEY: myMasterKey
      PARSE_DASHBOARD_USER_ID: admin
      PARSE_DASHBOARD_USER_PASSWORD: admin
    ports:
      - 4040:4040
    depends_on:
      - parse
```

1. Replace `myAppId` and `myMasterKey` with your own values. Ensure that the same values are used throughout the file.

Step 2: Start Docker containers

1. Open a command prompt or terminal.
2. Navigate to the project folder containing the `docker-compose.yml` file.
3. Run the following command to start the Docker containers:

```
docker-compose up
```

This command will start the Parse Server, PostgreSQL database, and Parse Dashboard.

Step 3: Create the backend folder

1. Inside your PEAN project folder, create a folder named `backend`.
2. Open a command prompt or terminal and navigate to the project folder.

Step 4: Initialize the Node.js project

1. Run the following command to initialize a new Node.js project:

```
npm init
```

1. Follow the prompts to set up the project. You can press Enter to accept the default values for most fields.

Step 5: Update the `package.json` file

1. Open the `package.json` file generated in the previous step.
2. Replace its contents with the following code:

```
{
  "name": "api",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "nodemailer": "^6.9.0",
    "nodemon": "^2.0.20",
    "parse": "^4.1.0",
    "parse-server": "^3.10.0"
  }
}
```

Step 6: Install project dependencies

1. In the command prompt or terminal, run the following command to install the project dependencies:

```
npm install
```

This command will install the required packages specified in the `package.json` file.

Step 7: Add backend files

1. Inside the `backend` folder, create a file named `.env`.
2. Open the `.env` file and add the following code:

```
databaseURI = 'postgres://postgres:myPassword@postgres-db:5432/myDatabase'
cloudPath = __dirname + '/cloud/main.js'
appId = 'myAppId'
masterKey = 'myMasterKey'
serverURL = 'http://localhost:1337/parse'
port = "1336"
```

1. Create a file named `app.js` inside the `backend` folder and add the following code:

```
require('dotenv').config(); //include .env file(db credentials)
const express = require('express');
const ParseServer = require('parse-server').ParseServer;
const app = express();
const router = express.Router()

const cors = require('cors');

const api = ParseServer({
    databaseURI: process.env.databaseURI,
    cloud: __dirname + '/cloud/main.js', // Absolute path to your Cloud Code
    appId: process.env.appId,
    masterKey: process.env.masterKey,
    serverURL: process.env.serverURL,
    enableAnonymousUsers: true,
    allowClientClassCreation: true
});

app.use(cors());
app.use('/parse', api);

app.get('/', (req, res) => {
    res.send("RUNNING!!!");
});

app.listen(process.env.port, function() {
    console.log('parse-server-example running on port ' + process.env.port);
});
```

Step 8: Create cloud functions

1. Inside the `backend` folder, create a folder named `cloud`.
2. Inside the `cloud` folder, create another folder named `functions`.
3. Inside the `functions` folder, create a file named `MUser.js` (use the same name as the class you created).
4. In the `MUser.js` file, write the following code to define a Parse.Cloud function for inserting data into the `MUser` class:

```
Parse.Cloud.define("addUser", async (request) => {
    const MUser = Parse.Object.extend("MUser");
    const user = new MUser();

    user.set("email", request.params.email);
    user.set("password", request.params.password);
    user.set("firstName", request.params.firstName);
    user.set("lastName", request.params.lastName);

    const result = await user.save();
    return result;
});
```

Step 9: Include cloud functions in `main.js`

1. Inside the `cloud` folder, create a file named `main.js`.
2. In the `main.js` file, require all the files you created in the `functions` folder, like this:

```
require('./functions/MUser.js');
```

Congratulations! You have successfully set up your Node.js project with a PEAN stack using Docker. You can now start building your application using the Parse Server and PostgreSQL database.