MERN

The MERN stack consists of MongoDB, Express.js ,React.js and Node.js.

Setting up MERN

When it comes to setting up MERN, there are a couple steps.

1. PostgreSQL Setup (read this to set it up)

Download and install PostgreSQL and ensure `psql` is recognized on the terminal.

Connect to Postgres

```
psql -d postgres -U postgres
```

> -d | specifies the name of database
> -U | specifies the username you want to connect as
>
> NOTE : Replace `postgres` with your database name and username if necessary.

Create a New User

```
CREATE ROLE my_user WITH LOGIN PASSWORD 'your_password';
ALTER ROLE my_user CREATEDB;
```

> This creates a user named `my_user` with the specified password and grants it database creation privileges. If you're comfortable using the default `postgres` user, you can skip this step.

Create a New Database

```
CREATE DATABASE my_database;
```

Grant Privileges (optional, but recommended)

```
GRANT ALL PRIVILEGES ON DATABASE my_database TO username;
```

Connect to the Database

```
psql -d my_database -U username
```

Switch to the New Database

```
\c my_database
```

Create a table

```
CREATE TABLE merchants( id SERIAL PRIMARY KEY, name VARCHAR(30), email
VARCHAR(30) );
```

Basic SQL queries

```
SELECT id, name, email from merchants;
/* retrieve data by key */


SELECT * from merchants;
/* retrieve all columns in the table */


INSERT INTO merchants (name, email) VALUES
/* inserts new data into the table */


DELETE from merchants WHERE id IN(1,2);
```

```
/* delete multiple rows at the same time */
```

```
UPDATE merchants SET name = 'jake', email = 'jake@mail.com' WHERE id = 1;
/* update a certain row */
```

Additional Notes:

- Replace `my_user`, `your_password`, and `my_database` with your desired values.
- Ensure you have the necessary permissions to perform these actions.
- For more advanced scenarios or production environments, consider additional security measures like role-based access control (RBAC).

Creating an API server with Node.js and Express

The backend can only communicate with the frontend via the API, thus it is crucial that we set one up. In this stack, `Node.js` and `Express.js` are used to create a simple server to connect to the PostgreSQL database we've created.

In order to imitate the behavior of a typical full-stack application, we'll build a React frontend and communicate with the CRUD API we build on the server.

However, in order to connect your React app with a PostgreSQL database, you must first create an API server that can process HTTP requests.

Create a Project Directory

```
mkdir node-postgres
cd node-postgres
```

Initialize a Node.js Project

```
npm init -y
```

> This creates a `package.json` file with default settings.

Create a backend folder

```
mkdir backend
cd backend
```

> - `express` is a popular Node.js web framework.
> - `pg` is a PostgreSQL client library for Node.js.

Install Required Packages

```
npm install express pg
```

> - `express` is a popular Node.js web framework.
> - `pg` is a PostgreSQL client library for Node.js.

Create a Server File

```
touch index.js
```

Setting up a basic server

```
/* index.js */

const express = require('express')
const app = express()
const port = 3001

app.get('/', (req, res) => {
  res.status(200).send('Hello World!');
})

app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

add `npm start` command

Within `package.json`, edit the “scripts” section to contain the start command like the following:

```
/* package.json */
  ..
  "scripts": {
    "start": "node index.js"
  },
  ..
```

> `node index.js` can also be run manually too via the terminal. However, it is much more convenient to simply attach an `npm start` command instead.

Open your terminal in the same directory and run `npm start`. Your Node application will run on port 3001, so open your browser and navigate to http://localhost:3001. You’ll see “Hello World!” text displayed in your browser.

You now have everything you need to write your API.

Making Node.js talk with Postgres

The `pg` library allows your Node application to talk with Postgres, so you’ll want to import it first.

Create a new file named `merchantModel.js` and input the following code:

```
/* merchantModel.js */

const Pool = require('pg').pool
const pool = new Pool({
        user: "my_user",
        host: "localhost",
        database: "my_database",
        password: 'root',
        port: 5432
})
```

Please note that putting credentials such as user, host, database, password, and port like in the example above is not recommended in a production environment. We'll keep it in this file to simplify the tutorial.

The right way to do this, however, would be to install a package called `dotenv` (remember to do this in your backend folder still) and use environment variables to reference these values from a `.env` file. Before pushing your project to a public platform such as GitHub, be sure to include the `.env` file in your `.gitignore`.

Back in our `merchantModel.js` code, the pool object you created above will allow you to query the database that it's connected to. Let's create three queries to make use of this pool. These queries will be placed inside a function, which you can call from your `index.js`:

```
/* merchantModel.js */
const Pool = require("pg").Pool;
const pool = new Pool({
  user: "my_user",
  host: "localhost",
  database: "my_database",
  password: "root",
  port: 5432,
});
//get all merchants our database
const getMerchants = async () => {
  try {
    return await new Promise(function (resolve, reject) {
      pool.query("SELECT * FROM merchants", (error, results) => {
        if (error) {
          reject(error);
        }
        if (results && results.rows) {
          resolve(results.rows);
        } else {
          reject(new Error("No results found"));
        }
      });
    });
  } catch (error_1) {
    console.error(error_1);
    throw new Error("Internal server error");
  }
};
//create a new merchant record in the databsse
const createMerchant = (body) => {
  return new Promise(function (resolve, reject) {
```

```
    const { name, email } = body;
    pool.query(
      "INSERT INTO merchants (name, email) VALUES ($1, $2) RETURNING *",
      [name, email],
      (error, results) => {
        if (error) {
          reject(error);
        }
        if (results && results.rows) {
          resolve(
            `A new merchant has been added:
${JSON.stringify(results.rows[0])}`
          );
        } else {
          reject(new Error("No results found"));
        }
      }
    );
  });
};
//delete a merchant
const deleteMerchant = (id) => {
  return new Promise(function (resolve, reject) {
    pool.query(
      "DELETE FROM merchants WHERE id = $1",
      [id],
      (error, results) => {
        if (error) {
          reject(error);
        }
        resolve(`Merchant deleted with ID: ${id}`);
      }
    );
  });
};
//update a merchant record
const updateMerchant = (id, body) => {
  return new Promise(function (resolve, reject) {
    const { name, email } = body;
    pool.query(
      "UPDATE merchants SET name = $1, email = $2 WHERE id = $3 RETURNING
*",
      [name, email, id],
      (error, results) => {
        if (error) {
          reject(error);
        }
        if (results && results.rows) {
          resolve(`Merchant updated: ${JSON.stringify(results.rows[0])}`);
        } else {
```

```
          reject(new Error("No results found"));
        }
      }
    );
  });
};
module.exports = {
  getMerchants,
  createMerchant,
  deleteMerchant,
  updateMerchant
};
```

The three functions above don't do anything extraordinary:

- The `getMerchants` function gets all the merchants from our database
- `createMerchant` creates a new merchant, with the name and email received from the body. These values will be passed by our frontend — we'll see how this happens as we code
- The `deleteMerchant` function deletes a particular merchant from the database. This merchant is specified by `id`, which is passed in as a parameter. On the frontend, `id` is a prompt
- `updateMerchant` updates a specific merchant, determined by that merchant's `id`

The code above will process and export the `getMerchants`, `createMerchant`, `updateMerchants`, and `deleteMerchant` functions. Now it's time to update your `index.js` file and use these functions:

```
/* index.js */

const express = require('express')
const app = express()
const port = 3001

const merchant_model = require('./merchantModel')

app.use(express.json())
app.use(function (req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', 'http://localhost:5173');
  res.setHeader('Access-Control-Allow-Methods',
'GET,POST,PUT,DELETE,OPTIONS');
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Access-
Control-Allow-Headers');
  next();
```

```
});

app.get('/', (req, res) => {
  merchant_model.getMerchants()
  .then(response => {
    res.status(200).send(response);
  })
  .catch(error => {
    res.status(500).send(error);
  })
})

app.post('/merchants', (req, res) => {
  merchant_model.createMerchant(req.body)
  .then(response => {
    res.status(200).send(response);
  })
  .catch(error => {
    res.status(500).send(error);
  })
})

app.delete('/merchants/:id', (req, res) => {
  merchant_model.deleteMerchant(req.params.id)
  .then(response => {
    res.status(200).send(response);
  })
  .catch(error => {
    res.status(500).send(error);
  })
})
app.put("/merchants/:id", (req, res) => {
  const id = req.params.id;
  const body = req.body;
  merchant_model
    .updateMerchant(id, body)
    .then((response) => {
      res.status(200).send(response);
    })
    .catch((error) => {
      res.status(500).send(error);
    });
});

app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

Now the app has three HTTP routes that can accept requests. The app uses middleware to ensure that requests are being processed properly and that certain requests are valid. For example, `app.use(express.json())` is written so that Express can accept incoming requests with JSON payloads.

Creating your React application

Your API is ready to serve and process requests, so now it's time to create a React application to send requests into it. Before moving on, navigate out of the `backend` folder and back into the root `node-postgres` folder.

Navigate into your frontend folder,

```
cd frontend
```

then run,

```
npm create vite@latest ./
```

When prompted with the selections, be sure to select React and JavaScript.

Now write a simple React app from scratch by replacing the contents of `App.jsx` with,

```
import {useState, useEffect} from 'react';

function App() {
  const [merchants, setMerchants] = useState(false);

  function getMerchant() {
    fetch('http://localhost:3001')
      .then(response => {
        return response.text();
      })
      .then(data => {
        setMerchants(data);
      });
  }

  function createMerchant() {
    let name = prompt('Enter merchant name');
```

```
  let email = prompt('Enter merchant email');
  fetch('http://localhost:3001/merchants', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({name, email}),
  })
    .then(response => {
      return response.text();
    })
    .then(data => {
      alert(data);
      getMerchant();
    });
}

function deleteMerchant() {
  let id = prompt('Enter merchant id');
  fetch(`http://localhost:3001/merchants/${id}`, {
    method: 'DELETE',
  })
    .then(response => {
      return response.text();
    })
    .then(data => {
      alert(data);
      getMerchant();
    });
}

function updateMerchant() {
  let id = prompt('Enter merchant id');
  let name = prompt('Enter new merchant name');
  let email = prompt('Enter new merchant email');
  fetch(`http://localhost:3001/merchants/${id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({name, email}),
  })
    .then(response => {
      return response.text();
    })
    .then(data => {
      alert(data);
      getMerchant();
    });
}
```

```
  useEffect(() => {
    getMerchant();
  }, []);
  return (
    <div>
      {merchants ? merchants : 'There is no merchant data available'}
      <br />
      <button onClick={createMerchant}>Add merchant</button>
      <br />
      <button onClick={deleteMerchant}>Delete merchant</button>
      <br />
      <button onClick={updateMerchant}>Update merchant</button>
    </div>
  );
}
export default App;
```

Now run your React app with `npm run dev`. You can test and see how the data collected from your React application is recorded in PostgreSQL

Sources and related content

Repo Recriacao App