Hyperiondev

TASK

React - Back-end Integration

Visit our website

Introduction

WELCOME TO THE REACT - BACK-END INTEGRATION TASK!

Having explored the essentials of back-end development, from creating web applications with Express to handling HTTP requests and responses and interfacing with MongoDB using Mongoose. We now turn our attention to a crucial aspect of modern web development: integrating the front-end with the back-end. In this task, you will learn how to establish a communication channel between the front-end and the back-end.

REACT - BACK-END INTEGRATION

Connecting the React front-end with the back-end requires communication between the two using HTTP requests. You will need to set up the front- and back-end using their required dependencies.

Create an Express.js back-end

Open up your terminal from your project's root directory and run the following commands:

Create the back-end directory:

```
mkdir backend
```

Change directory to the new directory:

```
cd backend
```

Run the following command to initialise the `package.json` file so that dependencies can be installed:

```
npm init -y
```

You will need '`express`' for creating the server and '`cors`' for Cross-Origin Resource Sharing. It is a process that allows the user to access restricted resources on the web. You can install both `express` and `cors` using the command given below:

```
npm install express cors
```

Lastly, create a file called `server.js` inside your back-end directory.

Take note:

CORS is an abbreviation for **Cross-Origin Resource Sharing**. It is a security measure for web applications to restrict unwanted domain access. It is, therefore, a set of rules that dictate how the web should handle requests for resources that originate from different domains. When a request is made to the server, a header known as **`origin`** is sent to the server, which contains information regarding the origin of the request.

Then, you can write the following code inside your **`server.js`** file in your back-end folder. This will create a simple server that you can adjust according to your requirements. The port number used in this server is '**`5000`**', which can be changed according to your browser. However, it is not advised to use a port number on which your React app will be running, such as port **`3000,`** as it can cause your back-end to crash.

```
const express = require('express');
const cors = require('cors');

const app = express();
// Define the port number for the server
const PORT = process.env.PORT || 5000;

// Enable Cross-Origin Resource Sharing
app.use(cors());

// Define the route to retrieve the message
app.get('/api/data', (req, res) => {
  const data = { message: 'Hello from the back end!' };
  res.json(data); // Send data as a response
});

// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

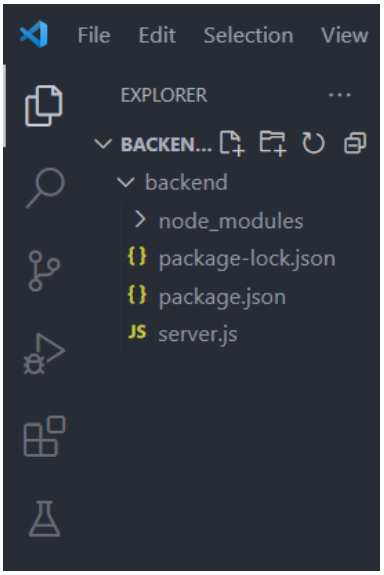
To check if your server is working, you can type in the following command in your terminal:

```
node server.js
```

If it shows a message saying, "`server is running on port 5000`", then that means your server's connection was established successfully.

Now that the server side has been set up, you can move on to creating the front-end of your project.

Your VS Code directory will look something like this at the end:



Create a React.js front-end

Open up a `new` terminal from your project's root directory and run the following command to initialise a React App:

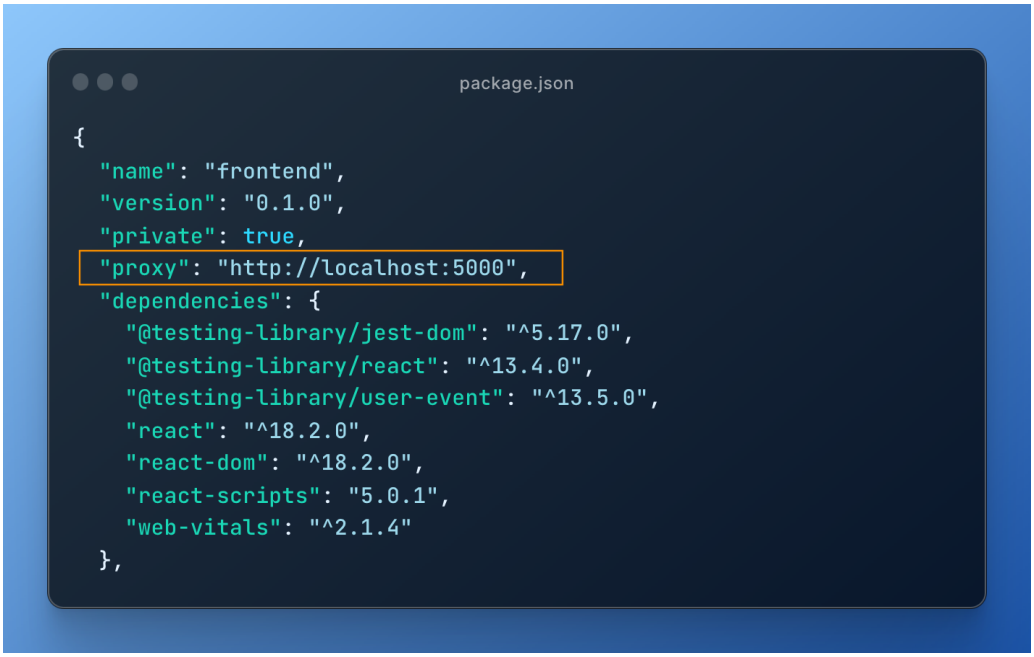
```
npx create-react-app frontend
```

The above command will create a folder named `frontend` in which all your React App files, such as `index.html, App.js,` and `package.json`, will be present.

Once your React App installation is complete, open the `frontend/package.json` file and add the following line:

```
"proxy": "http://localhost:5000",
```

Your `package.json` should now look like the following:



This ensures that the default localhost is set on port **`5000.`**

Now install the **`axios`** library for making HTTP requests from your React app. Axios helps with API fetching and facilitates communication with the back-end.

Open your terminal in the **`frontend`** directory and run the following command:

```
npm install axios
```

Now open your **`src/App.js`** file in your project directory and update the following code as starter code which you can alter according to your project requirements.

Notice that the following code is written by removing the pre-written code in the **`App.js`** file:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './App.css';

function App() {
  const [data, setData] = useState({}); // State to store fetched data

  useEffect(() => {
    fetchData(); // Fetch data each time the component loads
  }, []);
```

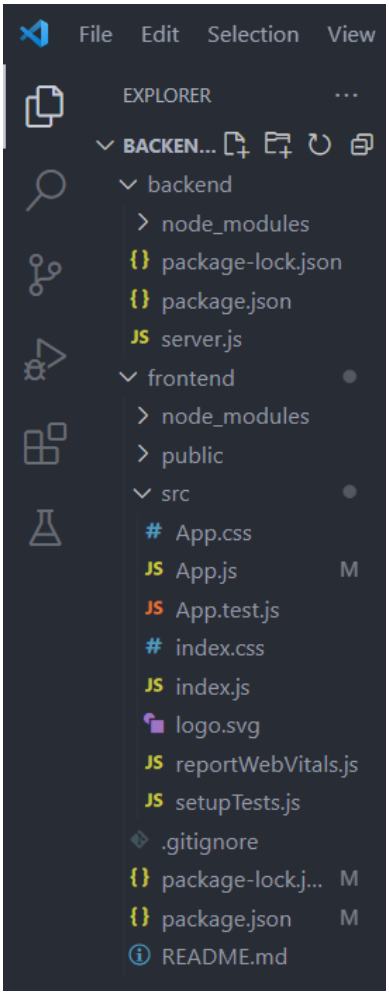
```
  // Function to fetch data from the server
  const fetchData = async () => {
    try {
      /* Sends a GET request to
      'http://localhost:5000//api/data' (backend server) */
      const response = await axios.get('/api/data');
      setData(response.data); // Update state with fetched data
    } catch (error) {
      console.error('Error fetching data:', error);
    }
  };

  return (
    <div className="App">
      <header className="App-header">
        {/* Display the message, or 'Loading...' if data is not yet fetched*/}
        <h1>{data.message || 'Loading...'}</h1>
      </header>
    </div>
  );
}
export default App;
```

You can start working on your front-end by running the command in the front-end directory of your terminal:

```
npm start
```

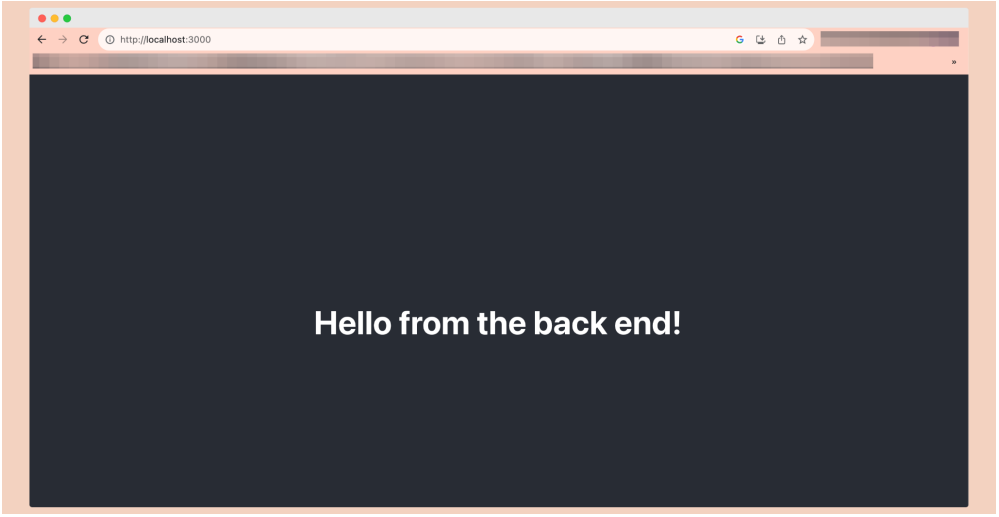
Your react app will be accessible at a local port site like **http://localhost:3000**.

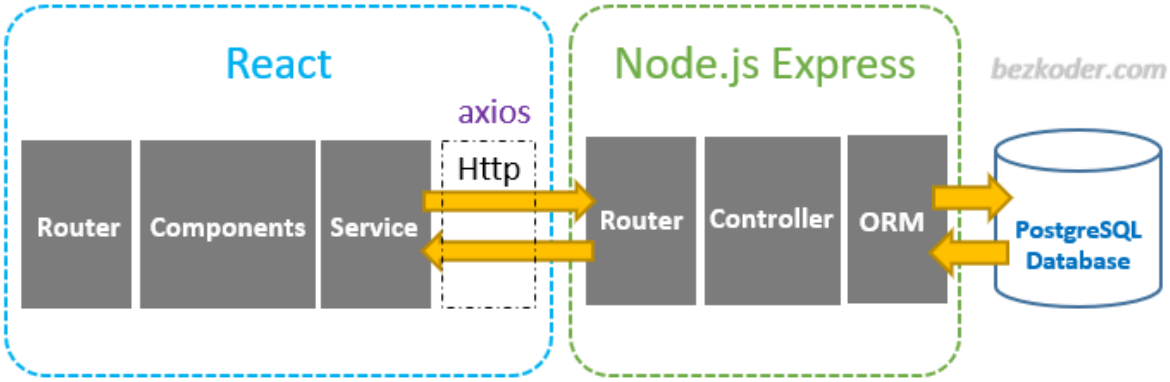
Your directory will look something like this at the end:



Connecting the front-end with the back-end

After setting up your front- and back-ends, run both servers in separate terminals. When you run your React app and access it through your browser, it will request the express.js back-end's '**`/api/data`**' (this is a generic example) endpoint and display the response. You will see the following output:





React + Node.js + Express + PostgreSQL example: Build a CRUD App - BezKoder
https://www.bezkoder.com/react-node-express-postgresql/

In conclusion, full-stack web development requires seamless integration of the front-end in React.js and the back-end in Express.js. This task illustrates the process of integrating a React front-end with a Node.js Express back-end, providing step-by-step instructions for creating both components and establishing communication between them.

Compulsory Task

Create a new API endpoint in the back-end that will be responsible for displaying a custom message.

Step 1: Add a back-end endpoint

1. Go to the back-end folder.
2. Edit `server.js`.
3. Add a new endpoint: `` `app.get('/api/message', ...)` ``.
4. Define a custom message in the endpoint's response.
5. Save and start the back-end server.

Step 2: Fetch the message from the front-end

1. Go to the front-end folder.
2. Open `src/App.js`.
3. Import the necessary libraries.
4. Add a new state: `` `const [customMessage, setCustomMessage] = useState('');` ``.
5. Use `` `useEffect()` `` to fetch a message from "`/api/message`".
6. Update `` `customMessage` `` with the fetched message.
7. Display `` `customMessage` `` in your app.

Step 3: Run servers and check

1. Run both back-end and front-end servers.
2. Open your app in a web browser.
3. You should see the original and custom messages displayed.

To submit the task for review:

1. Create a .pdf document named "`CustomMessageTask`".
2. Include a screenshot of your app showing both messages.
3. Compress your application folder and add it to the relevant task folder in Dropbox.
    - Remember to **delete** the **node_modules** folder **before submitting**. This folder typically contains thousands of files which, if you're working directly from Dropbox, has the potential to slow down Dropbox sync and possibly your computer.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.