**Steps in Setting Up:**

1. **Npm init** - initialize the folder containing ur workspace
2. **Npm i mongoose** - install mongoose libs
3. **Npm i express** - install express libs
4. **Npm i nodemon** - install nodemon package
5. **npm install** -g nodemon - to install globally
6. **Nodemon** - to start and check if api is ready

**Setting up Frontend:**

1. Npx create-react-app frontend
2. npm install react@18 react-dom@18

**NOTES:**

1. **MERN** - Mongo Express React Node
2. **Nav -**
3. **Router -**
4. **Route -**
5. **Sample of functional component in javascript**

| **const PrivateComponent = () => {**  **return <Outlet />**  **}** |
| --- |

**Error Encountered:**



### **Components of MERN:**

1. **MongoDB** (Database) - **Backend**
   * A NoSQL database that stores data in a flexible, JSON-like format.
   * Used to manage and store application data.
   * Ideal for modern applications that require scalability and fast development cycles.
2. **Express.js** (Web Framework) - **Backend**
   * A web application framework for Node.js.
   * Simplifies the process of building web applications and APIs.
   * Manages HTTP requests, routing, and middleware integration.
3. **React.js** (Frontend Library) - **Frontend**
   * A JavaScript library for building user interfaces.
   * Focuses on creating reusable UI components.
   * Used to build dynamic and interactive client-side web applications.
4. **Node.js** (JavaScript Runtime) - **Backend**
   * A runtime environment that allows you to run JavaScript on the server side.
   * Provides the ability to handle multiple requests concurrently.
   * Serves as the execution environment for Express.js and connects with MongoDB.

### 

### 

### 

### 

### **Responsibilities of Each in MERN Stack:**

| **Component** | **Frontend/Backend** | **Responsibility** |
| --- | --- | --- |
| **MongoDB** | Backend | Storing and retrieving data for the application. |
| **Express.js** | Backend | Managing HTTP requests, routes, and APIs. |
| **React.js** | Frontend | Creating the user interface and managing the user experience. |
| **Node.js** | Backend | Running the backend logic and acting as a server. |

### **How They Work Together:**

1. **Frontend (React.js):**
   * The user interacts with the UI, built using React.
   * React makes API calls (via HTTP) to the backend.
2. **Backend (Express.js & Node.js):**
   * The backend (Node.js + Express.js) processes these requests.
   * Express handles routing and communicates with the database.
3. **Database (MongoDB):**
   * MongoDB stores and retrieves data as needed by the backend.

**Steps in coding:**

1. Install Router - npm i react-router-dom
   1. In this step, this is necessary in order to create links or like the buttons to be used to navigate, that’s why the name of the file is Nav which means navigate.

| import React from 'react';  import { Link } from 'react-router-dom';  const Nav = () => {  return (  <div>  <ul>  <li><Link to="/">Products</Link></li>  <li><Link to="/add"> Add Products</Link></li>  <li><Link to="/update">Update Products</Link></li>  <li><Link to="/logout">Logout</Link></li>  <li><Link to="/profile">Profile</Link></li>  </ul>  </div>  )  }  // we need to export it  export default Nav; |
| --- |

* 1. Next is to show content in each nav, in order to ready how it will work and function:

| import './App.css';  import Nav from './Nav';  import { BrowserRouter, Routes, Route } from 'react-router-dom';  function App() {  return (  <div className="App">  <BrowserRouter>  <Nav />  <Routes>  <Route path="/" element={<h1>Product Component</h1>} />  <Route path="/add" element={<h1>Add Product Component</h1>} />  <Route path="/update" element={<h1>Update Product Component</h1>} />  <Route path="/logout" element={<h1>Logout Component</h1>} />  <Route path="/profile" element={<h1>Profile Component</h1>} />  </Routes>  </BrowserRouter>  </div>  );  }  export default App; |
| --- |

1. Header and Footer -
   1. Separate file for header and footer
   2. U can code using the terminal F12 and set the css file then just copy it in the app.css

| import React from "react";  const Footer = () => {  return (  <div>  <h3 className="footer">  E-Dashboard  </h3>  </div>  )  }  export default Footer; |
| --- |

* 1. Separate the components to its respective components.

1. Make Sign-up component
   1. Create another functional component named SignUp.js
   2. Change the Nav.js to show the nav of signup
   3. Input should be like this, it is not closed immediately

| <input type="password" placeholder="Enter Password" /> |
| --- |

* 1. Keep in mind that this is how u will be able to access each div or element in css, keep their classname clean and neat, e.g.

| className="inputBox" |
| --- |

* 1. To make the button functionable, utilize the usestate from react:

| import React, { useState } from "react";  const [name, setName] = useState("");  const [email, setEmail] = useState("");  const [password, setPassword] = useState(""); |
| --- |

* 1. To make use of the created useState, after declaring it, use it in the inputBox u created like this:

| <input className="inputBox" type="text" placeholder="Enter Name"  value={name} onChange={(event) => setName(event.target.value)}  />  <input className="inputBox" type="text" placeholder="Enter Email Address"  value={email} onChange={(event) => setEmail(event.target.value)}  />  <input className="inputBox" type="password" placeholder="Enter Password"  value={password} onChange={(event) => setPassword(event.target.value)}  /> |
| --- |

* 1. Now, in creating the button, u are now introduced with onClick method, declare it together with the usestate, then put onClick on the button.

| const SignUp = () => {  const [name, setName] = useState("");  const [email, setEmail] = useState("");  const [password, setPassword] = useState("");  const collectData = () => {  console.warn(name, email, password);  }  <button onClick={collectData} className="SignButton" type="button">SIGN UP</button> |
| --- |

* 1. Finally, this is the whole snippet of code for SignUp.js

| import React, { useState } from "react";  const SignUp = () => {  const [name, setName] = useState("");  const [email, setEmail] = useState("");  const [password, setPassword] = useState("");  const collectData = () => {  console.warn(name, email, password);  }  return (  <div className="register">  <h1>  Register  <input className="inputBox" type="text" placeholder="Enter Name"  value={name} onChange={(event) => setName(event.target.value)}  />  <input className="inputBox" type="text" placeholder="Enter Email Address"  value={email} onChange={(event) => setEmail(event.target.value)}  />  <input className="inputBox" type="password" placeholder="Enter Password"  value={password} onChange={(event) => setPassword(event.target.value)}  />  <button onClick={collectData} className="SignButton" type="button">SIGN UP</button>  </h1>  </div>  )  }  export default SignUp; |
| --- |

1. Database Setup
   1. MongoDB and mongosh fixed here: <https://stackoverflow.com/questions/15053893/mongo-command-not-recognized-when-trying-to-connect-to-a-mongodb-server/41507803#41507803>
   2. If mongo does not work, then proceed with downloading mongosh.
2. React and NodeJS signup set up
   1. Nodemon can start the backend because nodemon always start in the package
   2. Use this:   
      **mongoose.connect("mongodb://localhost:27017/e-commerce")**

If this doesn’t work: **mongoose.connect("mongodb://127.0.0.1:27017/e-commerce")**

* 1. Check the api using postman: <http://localhost:5000/register>
  2. This is how the signup will change in order to make it work to collect data: some things like body, method, or headers are not necessary but sometimes it is to declare the content type and such.

| const collectData = async () => {  console.warn(name, email, password);  let result = await fetch("http://localhost:5000/register", {  method: 'post',  body: JSON.stringify({ name, email, password }), // we need to change and use json.stringify when usiong mongodb  headers: {  'Content-Type': 'application/json'  }  });  result = await result.json();  console.warn(result);  } |
| --- |

1. Complete SignUp Flow
   1. Store data in a local storage, this snippet can store the user (specifically for mongodb as it needs to be JSON format). So each time it will be store in the local storage.

| localStorage.setItem("user", JSON.stringify(result)) // 2 parameters again |
| --- |

* 1. Make private component, there are components that should only be visible for those who logged in, snippet below is the one responsible for it, also to make every route private manage the app.js wherein the signup nav should be outside that private route.

| import React from "react";  import { Navigate, Outlet } from 'react-router-dom';  // to navigate and authneticate if the user logged in  const PrivateComponent = () => {  const auth = localStorage.getItem("user");  return auth ? <Outlet /> : <Navigate to="signup" />  }  export default PrivateComponent |
| --- |

* 1. Vice versa, we need to ensure that once logged in, the user will not be able to access sign up button.

| useEffect(() => {  const auth = localStorage.getItem("user");  if (auth) {  navigate('/')  }  }, []) |
| --- |

* 1. This is read as, if auth is true “ ? “, then proceed with the link, else “ : “ proceed with this.

| <li>{auth ?<Link to="/logout">Logout</Link>:<Link to="/signup">Sign Up</Link>}</li> |
| --- |

1. Logout Tutorial
   1. To fix the error of showing logout and signup by refreshing, we use navigate to automatically clear the data, useNavigate is vv powerful.

| const navigate = useNavigate();  const logout = () => {  localStorage.clear();  navigate('/signup')  }  <li>{auth ? <Link onClick={logout} to="/signup">Logout</Link> : <Link to="/signup">Sign Up</Link>}</li> |
| --- |