**Steps in Setting Up:**

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

**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 />**  **}** |
| --- |

1. **It’s a good practice to test first the api or backend, then separate the creation of components.**
2. **<ul> </ul> means unordered list**
3. **Usually in react:**
   1. **React frontend: Running on localhost:3000.**
   2. **Backend API: Running on localhost:5000.**
4. **Async, await =** A promise in JavaScript is an object that represents the eventual completion or failure of an asynchronous operation. It is used for handling asynchronous operations, such as making API calls or reading files, in a more organized and readable way.
5. **When excluding a variable, just simply add it with a space like this:**

| **const products = await Product.find().select("-name -\_id");** |
| --- |

**Not this:**

| **const products = await Product.find().select("-name" && "-\_id");** |
| --- |

1. **Cntrl+C will result to exit of terminal**
2. **There are lots of methods in making api route, such as get, delete, post, put, and etc. Explore them.**
3. **$set is the standard when updating data**
4. **Auth token is necessary for specifying the components to be shown specifically to that user.**

**Error Encountered:**

1. **Promise {<pending>} [[Prototype]]** = this is because u forgot the await function somewhere
2. **react-dom.development.js:4266 Uncaught TypeError: setPassword is not a functio**n = check if you declared the variables correctly, such us the brackets
3. **React Hook useEffect has a missing dependency: 'getProductDetails'. Either include it or remove the dependency array react-hooks/exhaustive-deps,** solution:

| The warning arises because the getProductDetails function is defined outside the useEffect hook and is not listed as a dependency in the dependency array ([]). This can lead to potential issues if getProductDetails relies on variables or closures that might change over time. React wants you to include getProductDetails in the dependency array or move the function inside the useEffect.  Here's how to address it:  Solution 1: Move getProductDetails Inside useEffect  Since getProductDetails is only used within the useEffect, you can move its definition inside the hook to avoid the warning: |
| --- |

**To try on your own:**

1. Test it by using online mongodb instead of localhost
2. Explore the css and create your own designs

### **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> |
| --- |

1. **Login API in NodeJS**
   1. Make route for API -
      1. User.find (to find the data from mongodb, this is the method, this will be used for login

| //api route for login  app.post("/login", async (req, resp) => {  try {  let user = await User.findOne(req.body);  if (user) {  resp.send(user); // Send the user details if found  } else {  resp.status(404).send({ message: "User not found" }); // Send error message if not found  }  } catch (error) {  console.error(error);  resp.status(500).send({ error: "An error occurred during login" });  }  }); |
| --- |

* + 1. U should implement .select (“-password”) for security measures so that api will not send password

| let user = await User.findOne(req.body).select("-password"); |
| --- |

* + 1. (“-password”) will only work with findOne method
    2. To make the same safety precautions with signup, this is how it should be done:

| result = result.toObject();  delete result.password |
| --- |

1. **Make login components:**
   1. Similar with how SignUp button created, we must create a component Login.js, then export it to be imported on app.js, then modify the nav.js to make it functional.
   2. For creating the Login component this is the code:

| import React, { useState } from 'react';  const Login = () => {  const [email, setEmail] = useState('');  const [password, setPassword] = useState('');  const handleLogin = () => {  console.warn(email, password)  }  return (  <div className="login">  <h1>  Login  <input className="inputBox" type="text" placeholder="Enter email"  onChange={(e) => setEmail(e.target.value)} value={email} />  <input className="inputBox" type="password" placeholder="Enter password"  onChange={(e) => setPassword(e.target.value)} value={password} />  <button onClick={handleLogin} classname="LoginButton" type="button">Login</button>  </h1>  </div>  )  };  export default Login; |
| --- |

* 1. This line of code is kind of confusing: **onChange={(e) => setEmail(e.target.value)},** but all it does is to get the email and password we will input from the input box, the “e” indicates the event, it’s like actionlistener of java.
  2. Creating the const handleLogin and using it on the button will make it functional, for now it does create a warning but we need to integrate api in order to get the emails and password if they align.

1. **Login API Integration**

| **import React, { useEffect, useState } from 'react';**  **import { useNavigate } from "react-router-dom"**  **const Login = () => {**  **const [email, setEmail] = useState('');**  **const [password, setPassword] = useState('');**  **const navigate = useNavigate();**  **useEffect(() => {**  **const auth = localStorage.getItem('user')**  **if (auth) {**  **navigate('/')**  **}**  **})**  **const handleLogin = async () => {**  **let result = await fetch("http://localhost:5000/login", {**  **method: 'post',**  **body: JSON.stringify({ email, password }),**  **headers: {**  **'Content-Type': 'application/json'**  **}**  **});**  **result = await result.json();**  **console.warn(result)**  **if (result.name) {**  **localStorage.setItem("user", JSON.stringify(result))**  **navigate('/')**  **} else {**  **alert('Please enter valid details.')**  **}**  **}**  **return (**  **<div className="login">**  **<h1>**  **Login**  **<input className="inputBox" type="text" placeholder="Enter email"**  **onChange={(e) => setEmail(e.target.value)} value={email} />**  **<input className="inputBox" type="password" placeholder="Enter password"**  **onChange={(e) => setPassword(e.target.value)} value={password} />**  **<button onClick={handleLogin} className="LoginButton" type="button">Login</button>**  **</h1>**  **</div>**  **)**  **};**  **export default Login;** |
| --- |

* 1. The whole source code above is similar to signup.js useEffect method is the one we used to auth that once we logged in we can’t go to signup button
  2. Next fulfilling the handleLogin function similarly to signUp.js we first fetch if the api works just fine
  3. Create an alert if there’s an error
  4. Create a navigator using react-dom library to limit the navigation once logged in similar to signup.js
  5. Allow saving in local storage using localstorage.setItem

1. **Update Navbar**
   1. Hide some nav bar that is not yet necessary, for example:

| const Nav = () => {  const auth = localStorage.getItem("user");  const navigate = useNavigate();  const logout = () => {  localStorage.clear();  navigate('/signup')  }  return (  <div>  {  auth ?  <ul className="nav-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="/profile">Profile</Link></li>  <li> <Link onClick={logout} to="/signup">Logout</Link></li>  </ul>  :  <ul className="nav-ul nav-right" >  <li><Link to="/signup">Sign Up</Link> </li>  <li><Link to="/login">Login</Link></li>  </ul>  }  </div >  )  } |
| --- |

* 1. The one responsible is the if else statement of auth, that if the user is not yet logged in then it will only show some components.
  2. Next, if you want to make the nav into right, simply add its class name:

| Nav.js:  <ul className="nav-ul nav-right" >  App.css:  .nav-ul li a {  color: #fff;  text-decoration: none;  padding: 5px;  }  .nav-right {  text-align: right;  } |
| --- |

* 1. This line of code will retrieve and show the name of the user through localstorage, take note it needs to be parse not stringify:

| <li> <Link onClick={logout} to="/signup">Logout({JSON.parse(auth).name})</Link></li> |
| --- |

* 1. To add an image or logo, click open image in new tab (if online), any picture then paste it to this code:

| <img  alt='logo'  src="https://img.freepik.com/free-vector/hand-drawn-flat-design-japan-food-illustration\_23-2149297535.jpg?semt=ais\_hybrid"></img> |
| --- |

* 1. Add its css into inspect element first, then paste it into app.css with its designated classname:

| .logo {  margin-top: 3px;  margin-left: 10px;  width: 45px;  height: 45px;  border-radius: 50%;  float: left; /\* responsible for making it above the texts \*/  } |
| --- |

1. **Add Product API**
   1. First create a collection named products in the mongodb compass
   2. Similarly to User.js, create a schema and its designated variables to be sent or stored, then check it using postman:

| const mongoose = require("mongoose");  const productSchema = new mongoose.Schema({  name: String,  price: String,  category: String,  userId: String,  company: String  });  module.exports = mongoose.model("products", productSchema); |
| --- |

* 1. At the index.js, test it on postman using this snippet :

| // api route for products creating its link, second parameter callback function with 2 parameters  app.post("/add-product", async (req, resp) => {  let product = new Product(req.body);  let result = await product.save();  resp.send(result)  }) |
| --- |

1. **Add Product Components**
   1. Make 4 input fields for the needed inputs

| import React from "react";  const AddProduct = () => {  return (  <div className="product">  <h1>Add Product</h1>  <input type="text" placeholder="Enter product name" />  <input type="text" placeholder="Enter product price" />  <input type="text" placeholder="Enter product category" />  <input type="text" placeholder="Enter product company" />  </div>  )  };  export default AddProduct; |
| --- |

* 1. You can use two class names if they have the same elements in css just separate it using a comma. for example:

| .register,  .product {  margin-left: 30%;  } |
| --- |

* 1. Create the necessary usestates and button to handle the input from each inputbox:

| import React, { useState } from "react";  const AddProduct = () => {  const [name, setName] = useState('');  const [price, setPrice] = useState('');  const [category, setCategory] = useState('');  const [company, setCompany] = useState('');  // button to handle the passed inputs from useState  const AddProduct = () => {  console.warn(name)  console.warn(price)  console.warn(category)  console.warn(company)  }  return (  <div className="product">  <h1>Add Product</h1>  <input className="inputBox" type="text" placeholder="Enter product name"  onChange={(e) => { setName(e.target.value) }} value={name}  />  <input className="inputBox" type="text" placeholder="Enter product price"  onChange={(e) => { setPrice(e.target.value) }} value={price}  />  <input className="inputBox" type="text" placeholder="Enter product category"  onChange={(e) => { setCategory(e.target.value) }} value={category}  />  <input className="inputBox" type="text" placeholder="Enter product company"  onChange={(e) => { setCompany(e.target.value) }} value={company}  />  <button onClick={AddProduct} className="ProductButton">Add Product</button>  </div>  )  };  export default AddProduct; |
| --- |

* 1. After that, check on the terminal whetter u are getting the needed output from it.

1. **Call AddProduct API**
   1. We need to know which product is added by which user, with the code below, it is the very snippet that needs to be used to access the local storage and get which is which.

| const AddProduct = () => {  console.warn(name)  const userId = localStorage.getItem('user');  console.warn(userId)  } |
| --- |

* 1. This snippet below, “.\_id” will handle only the id to be passed:

| const userId = JSON.parse(localStorage.getItem('user')).\_id; |
| --- |

* 1. Snippet below is the one responsible for routing the api and passing the inputs to mongodb. First get the userID to get which is which, and parse it. Create a let result wherein we will access the api we created for add product, passing the parameters needed such as method, body, etc. then call the result making it await.

| const AddProduct = async () => {  console.warn(name)  const userId = JSON.parse(localStorage.getItem('user')).\_id;  console.warn(userId)  let result = await fetch("http://localhost:5000/add-product", {  method: "post",  body: JSON.stringify({ name, price, category, company, userId }),  headers: {  "Content-type": "application/json"  }  });  result = await result.json();  console.warn(result)  } |
| --- |

1. **Form validation**
   1. By default we need to set the error as false, since there’s no error yet, then we need to pass and if statement that if there’s an empty or invalid input then it will turn true.

| const [error, setError] = useState(false);  // button to handle the passed inputs from useState  const addProduct = async () => {  console.warn(name)  const userId = JSON.parse(localStorage.getItem('user')).\_id;  console.warn(userId)  if (!name || !price || !category || !company) {  setError(true);  return false  } |
| --- |

* 1. In simple terms, we created a const and usestate to handle errors, first we need to initialize it as false since every time we start a homepage there’s no error ofcourse. Now create an if statement within the addproduct button that will handle errors if one of the inputs are not fulfilled. Then create a span accessing each variables based on its inputbox.

| import React, { useState } from "react";  const AddProduct = () => {  const [name, setName] = useState('');  const [price, setPrice] = useState('');  const [category, setCategory] = useState('');  const [company, setCompany] = useState('');  const [error, setError] = useState(false);  // button to handle the passed inputs from useState  const addProduct = async () => {  console.warn(name)  const userId = JSON.parse(localStorage.getItem('user')).\_id;  console.warn(userId)  if (!name || !price || !category || !company) {  setError(true);  return false  }  let result = await fetch("http://localhost:5000/add-product", {  method: "post",  body: JSON.stringify({ name, price, category, company, userId }),  headers: {  "Content-type": "application/json"  }  });  result = await result.json();  console.warn(result)  }  return (  <div className="product">  <h1>Add Product</h1>  <input className="inputBox" type="text" placeholder="Enter product name"  onChange={(e) => { setName(e.target.value) }} value={name}  />  {error && !name && <span className="invalidInput">Enter valid name</span>}  <input className="inputBox" type="text" placeholder="Enter product price"  onChange={(e) => { setPrice(e.target.value) }} value={price}  />  {error && !price && <span className="invalidInput">Enter valid price</span>}  <input className="inputBox" type="text" placeholder="Enter product category"  onChange={(e) => { setCategory(e.target.value) }} value={category}  />  {error && !category && <span className="invalidInput">Enter valid category</span>}  <input className="inputBox" type="text" placeholder="Enter product company"  onChange={(e) => { setCompany(e.target.value) }} value={company}  />  {error && !company && <span className="invalidInput">Enter valid company</span>}  <button onClick={addProduct} className="ProductButton">Add Product</button>  </div>  )  };  export default AddProduct; |
| --- |

1. **Product List API**
   1. Create GET method route, create in nodejs file which is the index.j, read the comments that’s all there is to it.

| app.get("/products", async (req, resp) => {  const products = await Product.find(); // find method will prodive all data from that table  // simply means we have some data in it  if (products.length > 0) {  resp.send(products) // return the data from it  } else {  resp.send({ result: "No product found" }) // prompt there's none in that  }  }) //this function will return a promise so we need to use async and await |
| --- |

* 1. Finally, check on postman if you can get through the api and u will receive the necessary data: **http://localhost:5000/products**

1. **Integrate Product List API**
   1. Similar to much earlier we need to create its respective component named; ProductList.js

| import React from "react";  const ProductList = () => {  return (  <div className="prodList">  <h1>  Product List  </h1>  </div>  )  };  export default ProductList; |
| --- |

* 1. useEffect and useState is necessary to show if we can see the data from the products table, this is necessary ,[], so that there will be no loop and unlimited sending of data

| const [product, setProducts] = useState([]);  useEffect(() => {  getProducts();  },[])  const getProducts = async () => {  let result = await fetch('http://localhost:5000/products');  result = await result.json();  setProducts(result)  } |
| --- |

* 1. Create as static list or header list, the .map will also create a bunch and many rows:

| <div className="prodList">  <h3>  Product List  <ul>  <li>S. No</li>  <li>Name</li>  <li>Price</li>  <li>Category</li>  <li>Company</li>  </ul>  {  product.map(() =>  <ul>  <li>S. No</li>  <li>Name</li>  <li>Price</li>  <li>Category</li>  <li>Company</li>  </ul>)  }  </h3>  </div> |
| --- |

* 1. The .map simply means to map all the table that acn be created from the retrieved or get method from postman, creating its index and item then its variable name to be map:

| {  product.map((item, index) =>  <ul key={item}>  <li>{index + 1}</li>  <li>{item.name}</li>  <li>{item.price}</li>  <li>{item.category}</li>  <li>{item.company}</li>  </ul>)  } |
| --- |

1. **Delete Product API**
   1. Create the route API for it at index.js, the deleteOne will handle the deletion of the specified id using req.params.id then passing it to resp.send to fulfill the request.

| app.delete("/product/:id", async (req,resp)=>{  let result = await Product.deleteOne({\_id:req.params.id}) // delete the id specified  resp.send(result)  })  app.listen(5000); |
| --- |

* 1. Test the created api using postman with this localhost using the DELETE method:

| http://localhost:5000/product/6777ced0a4280a210dc639d3 |
| --- |

1. **Integrate Delete Product API**
   1. Create a designated button column for each product that will handle the delete function.
   2. These are template strings: **`string text`.**
   3. We used dollar signs to access the ids specified.

| const deleteProduct= async (id)=>{  console.warn(id)  let result = await fetch(`http://localhost:5000/product/${id}`,{  method: "Delete"  });  result = await result.json();  if(result){  alert("Product Deleted")  }  }  return (  <div className="prodList">  <h3>  Product List  <ul>  <li>S. No</li>  <li>Name</li>  <li>Price</li>  <li>Category</li>  <li>Company</li>  <li>Operation</li>  </ul>  {  product.map((items, index) =>  <ul key={items.\_id}>  <li>{index + 1}</li>  <li>{items.name}</li>  <li>{items.price}</li>  <li>{items.category}</li>  <li>{items.company}</li>  <li><button onClick={()=>deleteProduct(items.\_id)}>Delete</button></li>  </ul>)  }  </h3>  </div >  )  }; |
| --- |

1. **Update Product Component UI**
   1. Add link for update product
   2. Create a second parameter for update so that it will access the specific id

| <Route path="/update/:id" element={<UpdateProduct />} />  on app.js |
| --- |

1. **API to get specific data**
   1. Create a API route for Update function, test it with postman also if it will return the specific data.

| // API for update single product  app.get("/product/:id", async (req, resp) => {  let result = await Product.findOne({ \_id: req.params.id }) //find only one  if (result) {  resp.send(result)  } else {  resp.send({ "result": "No result found" })  }  }) |
| --- |

1. **Prefill update form**
   1. Get the necessary id from each URL params, and we will use paramHook with that. “import {useParams} from 'react-router-dom';”
   2. After importing, we need to declare the const params that will be accessed using the getProductDetails function, after declaring it as a useEffect.
   3. Now fetching the details we need to use the params.id in order to use that id.
   4. Since we are already using the states of name, price, product, we just need to put its states so that it will be on that input field.

| import React, { useEffect ,useState } from "react";  import {useParams} from 'react-router-dom';  const UpdateProduct = () => {  const [name, updateName] = useState();  const [price, updatePrice] = useState();  const [category, updateCategory] = useState();  const [company, updateCompany] = useState();  const params = useParams();  const UpdateProduct = () => {  console.warn(name, price, category, company)  }  useEffect(()=>{    getProductDetails();  },[])  const getProductDetails= async ()=>{  console.warn(params);  let result = await fetch(`http://localhost:5000/product/${params.id}`);  result = await result.json();  console.warn(result)  updateName(result.name);  updatePrice(result.price);  updateCategory(result.category);  updateCompany(result.company);  }  return (  <div className="UpdateProduct">  <h1>Update Product</h1>  <input type="text" className="inputBox" placeholder="Enter name to update"  onChange={(e) => { updateName(e.target.value) }} value={name}  />  <input type="text" className="inputBox" placeholder="Enter price to update"  onChange={(e) => { updatePrice(e.target.value) }} value={price}  />  <input type="text" className="inputBox" placeholder="Enter category to update"  onChange={(e) => { updateCategory(e.target.value) }} value={category}  />  <input type="text" className="inputBox" placeholder="Enter company to update"  onChange={(e) => { updateCompany(e.target.value) }} value={company}  />  <button className="UpdateButton" onClick={UpdateProduct}> Update</button>  </div>  )  }  export default UpdateProduct; |
| --- |

1. **API for Update Product**
   1. Put is the method needed for it
   2. $set is the standard when updating data

| // new route for api to update the product  app.put("/product/:id", async (req, resp) => {  let result = await Product.updateOne(  { \_id: req.params.id },  { $set: req.body }  ) // when we are updating, we use this updateOne method and having its two parameters, first is the data then what we want to update the data is the 2nd object  resp.send(result)  }) |
| --- |

1. **Integrate Update Product API with UI**
   1. We need to create useEffect, create the necessary method, updateProduct function, and insert the necessary parameters without typo.
   2. We got an error here, we needed to use the function inside useEffect to prevent the error.

| const [name, updateName] = useState();  const [price, updatePrice] = useState();  const [category, updateCategory] = useState();  const [company, updateCompany] = useState();  const params = useParams();  useEffect(() => {  const getProductDetails = async () => {  console.warn(params);  let result = await fetch(`http://localhost:5000/product/${params.id}`);  result = await result.json();  console.warn(result);  updateName(result.name);  updatePrice(result.price);  updateCategory(result.category);  updateCompany(result.company);  };  getProductDetails();  }, [params]);  const updateProduct = async () => {  console.warn(name, price, category, company)  let result = await fetch(`http://localhost:5000/product/${params.id}`, {  method: "Put",  body: JSON.stringify({ name, price, category, company }),  headers: {  'Content-Type': 'Application/json'  }  });  result = await result.json();  console.warn(result)  } |
| --- |

1. **Search API for Product**
   1. Make route for API for backend so that search engine will work
   2. get method is also what we need to handle search
   3. Test it with postman
   4. You only need strings in searching
   5. Comments are provided in the snippet below

| // Endpoint for searching products by a key provided in the URL parameter  app.get("/search/:key", async (req, resp) => {  // Use the 'Product' model to search the database  // The query uses the '$or' operator to match the 'name' field  // The '$regex' operator performs a case-sensitive pattern match with the key  // 'req.params.key' contains the dynamic search term provided in the URL  let result = await Product.find({  $or: [  {  name: { $regex: req.params.key },  },  {  company: { $regex: req.params.key }  },  {  category: { $regex: req.params.key }  }  ],  });  // Send the search results back to the client as the response  resp.send(result);  }); |
| --- |

1. **Integrate Search API**
   1. Make input box for Search

| <div className="prodList">  <h3>  Product List  <input type="" className='search-product-box' placeholder='Search Product'  onChange={searchHandle}  /> |
| --- |

* 1. Create the function to handle the event. Event.target.value is the one we use to test the searchhandle

| const searchHandle = async (event) => {  let key = event.target.value;  if(key){    let result = await fetch(`http://localhost:5000/search/${key}`);  result = await result.json()  if (result){  setProducts(result)  }  }  else{  getProducts();  }  } |
| --- |

* 1. Create a function within mapping of products, an if else statement that will show a No Result Found pop up when there’s no product

| {  product.length > 0 ? product.map((items, index) =>  <ul key={items.\_id}>  <li>{index + 1}</li>  <li>{items.name}</li>  <li>{items.price}</li>  <li>{items.category}</li>  <li>{items.company}</li>  <li><button onClick={() => deleteProduct(items.\_id)}>Delete</button>  <Link to={"/update/" + items.\_id} > Update</Link>  </li>  </ul>  )  :  <h1>No Result Found</h1>  } |
| --- |

1. **JWT token Authentication = https://www.npmjs.com/package/jsonwebtoken**
   1. npm i jsonwebtoken
   2. Search for this:

| const Jwt = require('jsonwebtoken');  const jwtKey = 'e-comm';  const app = express();  //api route for login  app.post("/login", async (req, resp) => {  if (req.body.password && req.body.email) {  let user = await User.findOne(req.body).select("-password");  if (user) {  Jwt.sign({user},jwtKey,{expiresIn:"2h"},(err,token)=>{  if(err){  resp.send({result:"Something went wrong, please try after some time"})  }  resp.send({user,auth:token}); // Send the user details if found  })    } else {  resp.send({ result: "User not found" }); // Send error message if not found  }  } else {  resp.send({ result: "No user found" });  }  }); |
| --- |

1. **Handle JWT Token in React js**
   1. To create auth for productlist that will be used later:

| useEffect(() => {  const getProducts = async () => {  let result = await fetch('http://localhost:5000/products',{  headers:{  authorization:JSON.parse(localStorage.getItem('token'))  }  });  result = await result.json();  setProducts(result)  };  getProducts();  }, []) |
| --- |