

Objective:

- Develop and implement RESTful API endpoints in Express to handle the basic CRUD operations (Create, Read, Update, Delete) for a given resource (e.g., users, products, or tasks).
- Use tools like Postman or automated testing libraries to test the API endpoints for correctness, error handling, and validation.

Job Roles : Full Stack Developer, Web Developer, Frontend Developer, Database Administrator (MongoDB), DevOps Engineer (for MERN Stack), API Developer, Web Application Architect (MERN Stack)

Skills: Node JS, Express JS, Restful API (POST, GET, PUT, DELETE), Validation of Restful API, React Hooks such as useState and useEffect, HTML form Submission, BrowserRouter and Toast

Prerequisites:

- **JavaScript:** Basic knowledge of JavaScript, including variables, data types, loops, functions, and how to manipulate the DOM.
- **HTTP (HyperText Transfer Protocol)** is the fundamental protocol used for communication between clients (such as web browsers or React applications) and web servers. It is the protocol that powers the web and facilitates the retrieval of resources (like HTML pages, images, and data) over the internet.

Question

Develop a MERN stack web application as per the theme allotted, to demonstrate the CRUD(Create, Read, Update, Delete) operations using MongoDB and Express JS Restful APIs as given below.

- **Create** – To store the data in MongoDB through HTTP POST Restful API created via Express JS server.
- **Read** – To fetch all the records from database and display using HTTP GET Restful API created via Express JS server.
- **Update** – To modify the existing record by the ID through HTTP PUT Restful API created in Express JS.
- **Delete** - To delete the existing record by the ID through HTTP DELETE Restful API created in Express JS.

Procedure:

Let us assume that, we need to develop a **Product Inventory Management System** using the **MERN (MongoDB, Express.js, React.js, Node.js) stack** to manage products in an inventory. Develop a **RESTful API** to perform **CRUD operations** on products:

- **Create a Product** (POST /api/addproduct)
 - Accept product details like name, price, quantity.
 - Store data in **MongoDB** using **Mongoose**.
- **Retrieve Products**

- **Get all products** (GET /getAllproducts) – Fetch a list of all products.
- **Update a Product** (PUT /editproduct/:id)
 - Modify an existing product's details using its ID.
- **Delete a Product** (DELETE /deleteproduct/:id)
 - Remove a product from the inventory by ID

Build a **React.js** frontend with the following features:

- **Product List Page** – Display all products in a table with options to **Edit** or **Delete**.
- **Add Product Page** – A form to add new products.
- **Edit Product Page** – A form to update an existing product.

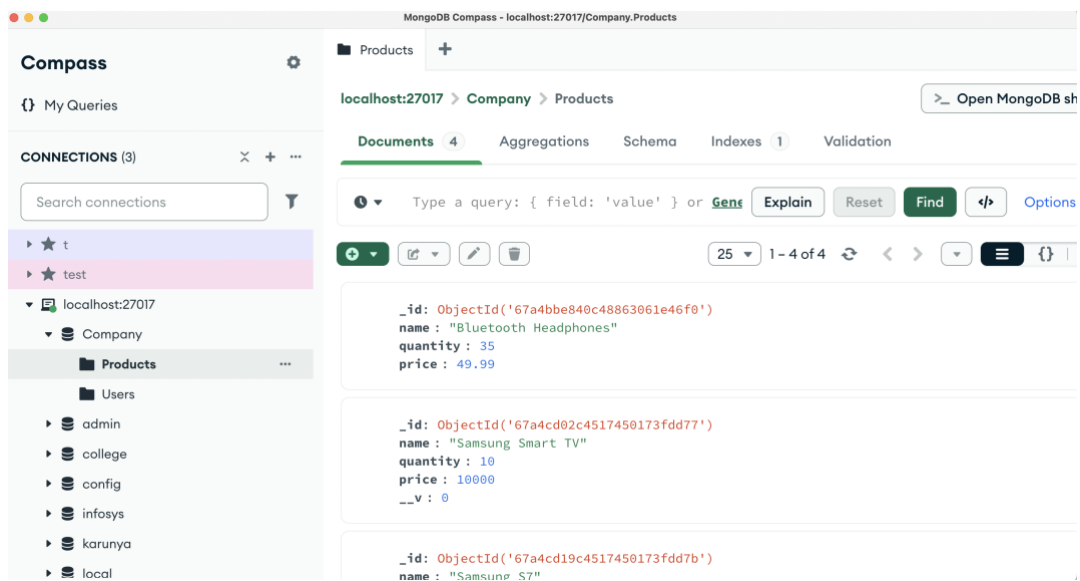
Additional Requirements:

- Use **Axios** for API requests.
- Implement **React BrowserRouter** for navigation between pages

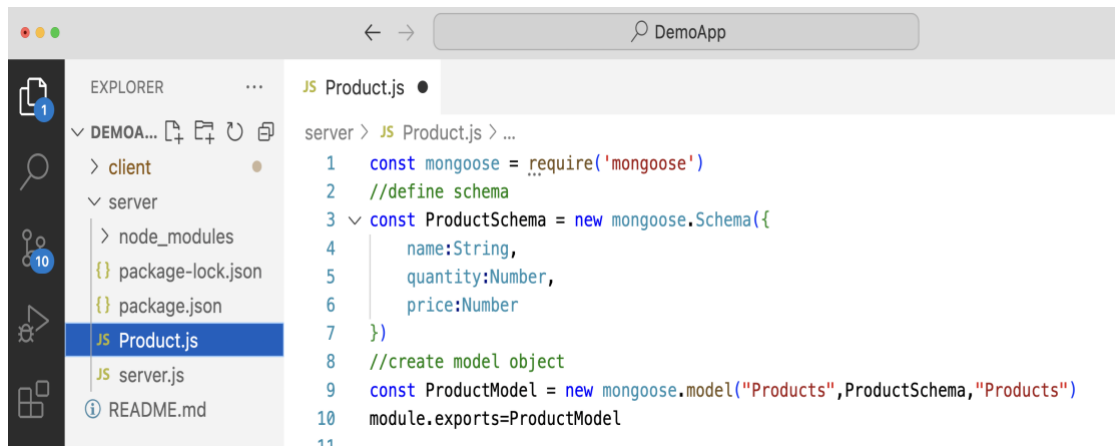
Steps:

Backend Server Application using Express

- Create a **MERN Stack Project** and set up **Node.js**, **Express.js**, **Mongoose**, and **React.js** with **Tailwind CSS**, following the configurations outlined in **Exercises 1, 2, and 3**.
- Create a collection in MongoDB database called **"Products"** with data such as name, quantity, and price.



- Create a **Product schema** and **Product model** in the server application using the **Mongoose module** to define the **"Products"** collection in **MongoDB**.



Here,

```
const ProductModel = new
mongoose.model("Products",ProductSchema,"Products")
```

First argument is the name of the **model**

Second argument is the reference to **ProductSchema**.

Third argument is the **Name of the collection** created already in MongoDB.

- Implement Restful APIs for CRUD operations in server.js application using Express JS and Mongoose APIs with following API functions.

```
const app = express()
```

```
//Create Rest API
```

```
app.post()
```

```
// Read Rest API
```

```
app.get()
```

```
// Update Rest API
```

```
app.put()
```

```
// Delete Rest API
```

```
app.delete()
```

- Complete the basic setup of the server application by including the required modules such as **Express, CORS, and Mongoose** in the server program. Also, import the **ProductModel** created based on the schema. Establish the **MongoDB database connection** using the Mongoose module as shown below.

```
const express = require('express')
const cors = require('cors')
const mongoose = require('mongoose')
const ProductModel = require('./Product')
```

```
const app = express()
app.use(cors())
app.use(express.json())
```

```
mongoose.connect('mongodb://127.0.0.1:27017/Company')
.then(() => console.log('DB connected'))
.catch(err => console.log(err))
```

//Implement all Resful APIs Here

```
const PORT = 8000
app.listen(PORT, ()=>{
  console.log(`Server running on port ${PORT}`)
})
```

- Create Restful APIs for CREATE, READ, UPDATE and DELETE operations.

```
//Create Rest API
app.post('/addProduct', async (req,res)=>{
  try{
    await ProductModel.create(req.body)
    res.json({message:'Product Added Successfully'})
  }
  catch(error){
    res.json(error)
  }
})

//Read All - Rest API
app.get('/viewProducts', async (req,res)=>{
  try{
    const records = await ProductModel.find()
    res.json(records)
  }
  catch(error){
    res.json(error)
  }
})

//Read By ID Rest API - to display before updation (EditProduct)
app.get('/findProduct/:id', async (req,res)=>{
  try{
    const record = await ProductModel.findById(req.params.id)
    res.json(record)
  }
  catch(error){
    res.json(error)
  }
})
```

```

//Update - REST API
app.put('/editProduct/:id', async (req,res)=> {
  try {
    const updatedProduct = await ProductModel.findByIdAndUpdate(
      req.params.id, req.body, {new:true})
    if (!updatedProduct) {
      return res.send('Item not found');
    }
    res.json({message:'Product Updated Successfully'});
  } catch (err) {
    res.json(err);
  }
})

//Delete - REST API
app.delete('/deleteProduct/:id', async (req,res)=>{
  try{
    const deletedItem = await
ProductModel.findByIdAndDelete({ _id:req.params.id})
    res.json({message:'Item Deleted Successfully!'})
  }
  catch(error){
    res.json(error)
  }
})

```

Mongoose APIs Explanation:

`await ProductModel.create(req.body)`

- **req.body** contains the product data (JSON format) sent from the client (via a POST request).
- The `.create()` method:
 - **Validates** the data against the Mongoose schema.
 - **Saves** the new document into the MongoDB database.
 - **Returns** the created document.

`const records = await ProductModel.find()`

- **ProductModel.find()** is a Mongoose query that retrieves all records from the ProductModel collection.
- **await** ensures that the function waits for the query to complete before proceeding.
- `const records` stores the fetched data, which is an array of documents.

`const record = await ProductModel.findById(req.params.id)`

- **ProductModel.findById(id)**: A Mongoose method that retrieves a single document from the ProductModel collection by its `_id` field.
- **req.params.id**: Extracts the id from the request URL parameters (assuming you're using Express.js).

```
const updatedProduct = await ProductModel.findByIdAndUpdate( req.params.id,
                                                            req.body, {new:true})
```

- **ProductModel.findByIdAndUpdate(id, update, options):** A Mongoose method that finds a document by `_id`, updates it, and returns the updated document.
- **req.params.id:** Extracts the id from the request URL parameters.
- **req.body:** Contains the updated data to apply to the document.
- **{ new: true }:** Ensures that the updated document is returned instead of the old one.

```
const deletedItem = await ProductModel.findByIdAndDelete({_id:req.params.id})
```

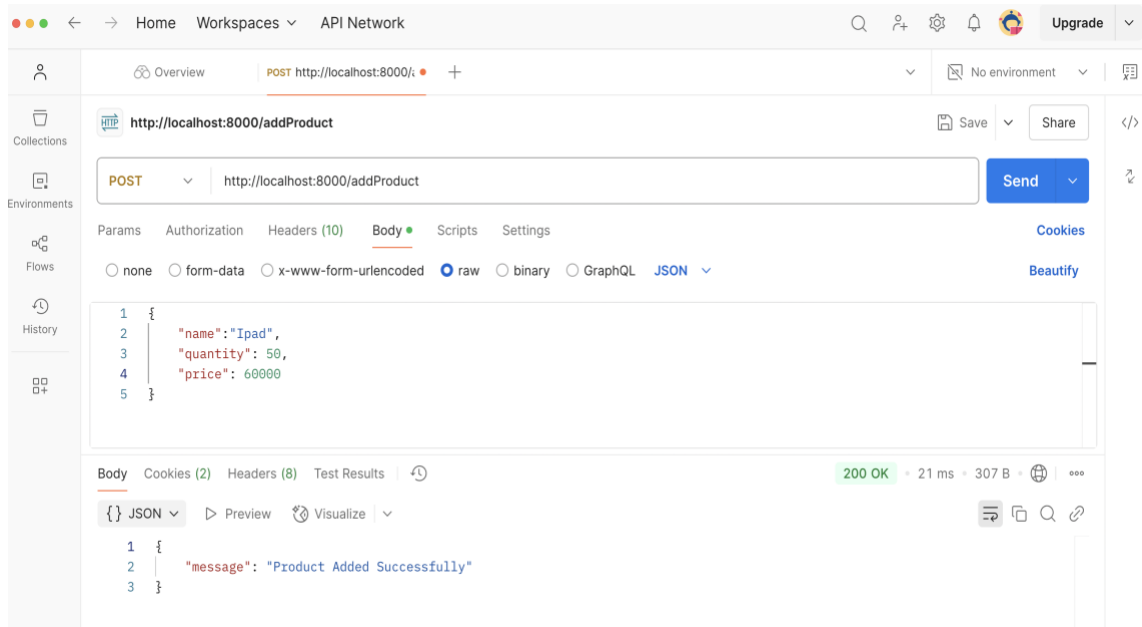
- **ProductModel.findByIdAndDelete({ _id: req.params.id }):** Deletes a document from the ProductModel collection by `_id`.
 - **req.params.id:** Extracts the ID from the request URL parameters.
 - **await:** Ensures the deletion process completes before proceeding.
- Save all the programs and run the server application. Also, validate the API using Postman tool.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    AZURE

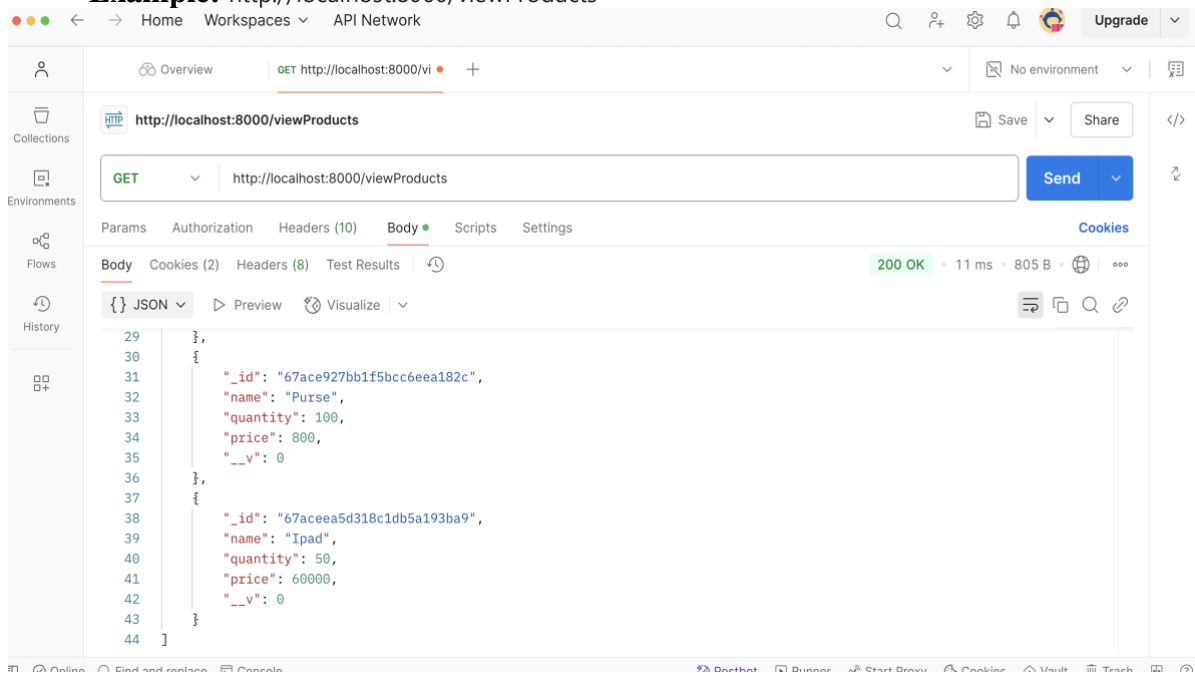
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Server running on port 8000
DB connected
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Server running on port 8000
DB connected
□
```

Validate All the API using Postman

For Example: <http://localhost:8000/addProduct>



Example: `http://localhost:8000/viewProducts`



Implementation of Front End (React JS + Tailwind CSS)

Build a **React.js** frontend with the following features:

- **Product List Page** – Display all products in a table with options to **Edit** or **Delete**.
- **Add Product Page** – A form to add new products.
- **Edit Product Page** – A form to update an existing product.

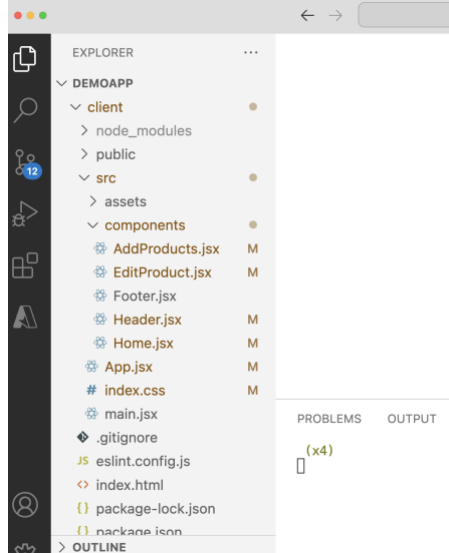
Additional Requirements:

- Use **Axios** for API requests.

- Implement **React BrowserRouter** for navigation between pages

Steps:

- Develop JSX pages or components for Product Listing, Add Product, Edit Product, as well as Header and Footer, and organize them within a separate components directory for better maintainability.



- Develop the App.jsx program with BrowserRouter feature to navigate or route between above mentioned pages as given below. First we need to install the BrowserRouter for the client application.

npm install react-router-dom

After installation, you can import and use BrowserRouter in your app like this:

```
import { BrowserRouter, Routes, Route } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<ComponentName />} />
        <Route path="/ " element={<ComponentName/>} />
        <Route path="/ " element={<ComponentName />} />
      </Routes>
    </BrowserRouter>  );
}

export default App;
```

- For example, In this project, the BrowserRouter module is used to Navigate between Home Page (Product List Page) and Add Products / Edit Products Pages as given below


```

import './index.css';
import {BrowserRouter, Routes, Route} from 'react-router-dom';
import Home from './components/Home';
import Header from './components/Header';
import Footer from './components/Footer';
import AddProducts from './components/AddProducts';
import EditProduct from './components/EditProduct';

function App() {
  return (
    <
      <BrowserRouter>
        <Header/>
        <main className="bg-gray-100 flex flex-col pb-48">
          <Routes>
            <Route path="/" element={<Dashboard />} />
            <Route path="/addproducts" element={<AddProducts />} />
            <Route path="/editproduct/:id" element={<EditProduct />} />
          </Routes>
        </main>
        <Footer/>
      </BrowserRouter>
    </>
  )
}
export default App

```

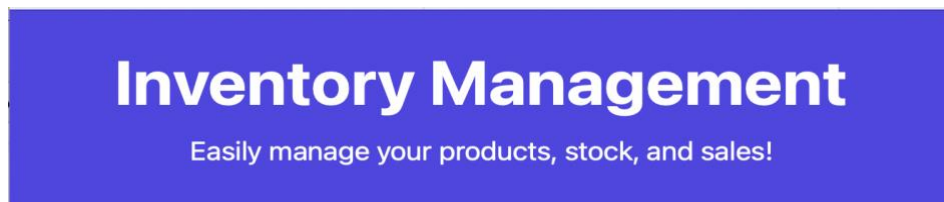
- The Header.jsx page is designed to display Page header information as given below.

```

import React from "react";

function Header() {
  return(
    <header className="text-center bg-indigo-600 text-white py-8">
      <h1 className="text-5xl font-bold">Inventory Management</h1>
      <p className="mt-4 text-xl">Easily manage your products, stock, and
sales!</p>
    </header>
  )
}
export default Header

```



- The Footer.jsx is also designed to display the Footer information such as copyright and contact details.

```
import React from "react";

function Header() {
  return(
    <footer className="text-center py-6 bg-gray-800 text-white">
      <p>&copy; 2025 Inventory Management. All rights reserved.</p>
    </footer>
  )
}
export default Header
```

© 2025 Inventory Management. All rights reserved.

- The **Dashboard.jsx** component is designed to showcase all products available in the MongoDB collection using an HTML table. Each record includes Edit (linking to the Edit Products page) and Delete options. Additionally, a link is provided to navigate to the Add Products page.

```
import React from 'react';
import { useState, useEffect } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';
import { ToastContainer, toast, Bounce } from 'react-toastify';

function Dashboard(){

  const [products,setProducts] = useState([])

  const viewProducts = async ()=>{
    try{
      const response = await axios.get('http://localhost:8000/viewProducts')
      setProducts(response.data)
    }catch(error){
      console.log(error)
    }
  }
  useEffect(()=>{
    viewProducts()
  },[])
}
```

```

const handleDelete = async (id)=>{
  const isConfirmed = confirm('Are you sure you want to delete this product?')
  if(isConfirmed) {
    try {
      const response = await
    axios.delete(`http://localhost:8000/deleteProduct/${id}`)
      toast.success(response.data.message);
      viewProducts()
    }
    catch(error){
      console.log(error)
    }
  }
}

```

```

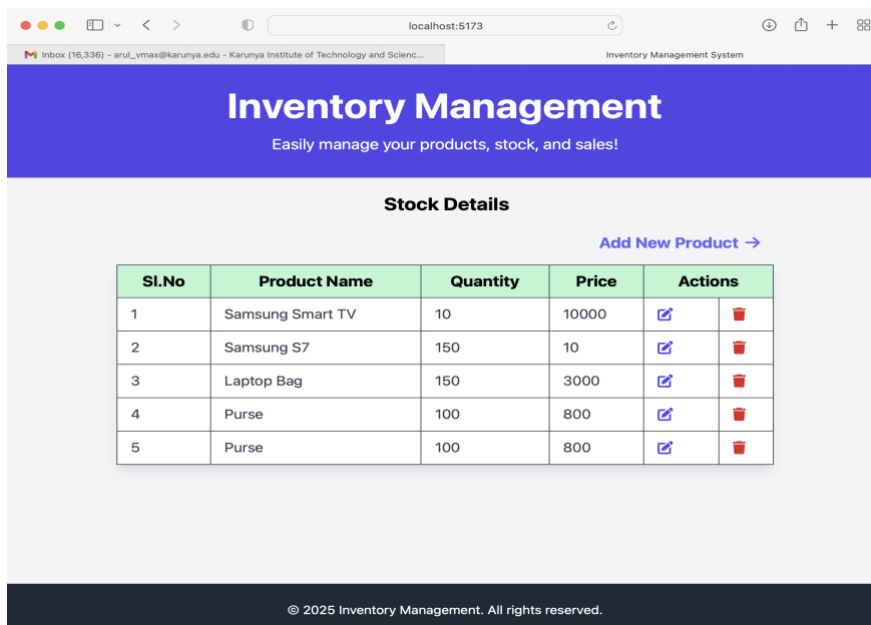
return (
  <div>
    <div>
      <h1>Stock Details </h1>
      <table>
        <caption>
          <Link to="/addproducts">Add New Product </Link>
        </caption>
        <thead>
          <tr>
            <th>Sl.No</th>
            <th>Product Name</th>
            <th>Quantity</th>
            <th>Price</th>
            <th>Actions</th>
          </tr>
        </thead>

```

```

<tbody>
  {products.map((item,index)=>{
    return(
      <tr className="text-lg text-gray-700" key={index} >
        <td>{index+1}</td>
        <td>{item.name}</td>
        <td>{item.quantity}</td>
        <td>{item.price}</td>
        <td>
          <Link to={`/editproduct/${item.id}`} >
            <i className="fas fa-edit text-lg"></i>
          </Link>
        </td>
        <td>
          <button onClick={() => handleDelete(item.id)}>
            <i className="fas fa-trash text-lg"></i>
          </button>
        </td>
      </tr>
    )
  })}
</tbody>
</table>
<ToastContainer position='top-left' autoClose={3000} hideProgressBar={false}
closeOnClick={true} transition={Bounce} theme="colored"/>
</div>
</div>
);
export default Dashboard;

```



[Use Tailwind CSS to style the page according to your preferences.]

Here, the following code makes **HTTP Rest API Request** using **Axios** module and fetch all the records and update the state called “products”. The **useEffect** hook calls the **viewProducts()** function automatically only once when the page gets loaded in Browser.

```
const [products,setProducts] = useState([])

const viewProducts = async ()=>{
  try{
    const response = await axios.get('http://localhost:8000/viewProducts')
    setProducts(response.data)
  } catch(error){
    console.log(error)
  }
}

useEffect(()=>{
  viewProducts()
},[])
```

The **map()** function can be used to iterate the product details from the state called “products” and we can display using Table rows. The expression tag called **{ }** can be used to bind the value of every product detail such as name, quantity and price.

```
<tbody>
{
  products.map((item,index)=>{
    return(
      <tr key={index} >
        <td>{index+1}</td>
        <td>{item.name}</td>
        <td>{item.quantity}</td>
        <td>{item.price}</td>
        <td>
          <Link to={`/editproduct/${item.id}`} >
            <i className="fas fa-edit text-lg"></i>
          </Link>
        </td>
        <td>
          <button onClick={() => handleDelete(item.id)}>
            <i className="fas fa-trash text-lg"></i>
          </button>
        </td>
      </tr>
    )
  })
}
</tbody>
```

The delete functionality is implemented using a "trash icon" button, which triggers the `handleDelete(item._id)` event handler. Within the `handleDelete` function, the `Axios` module is used to make a RESTful API call to the backend server application.

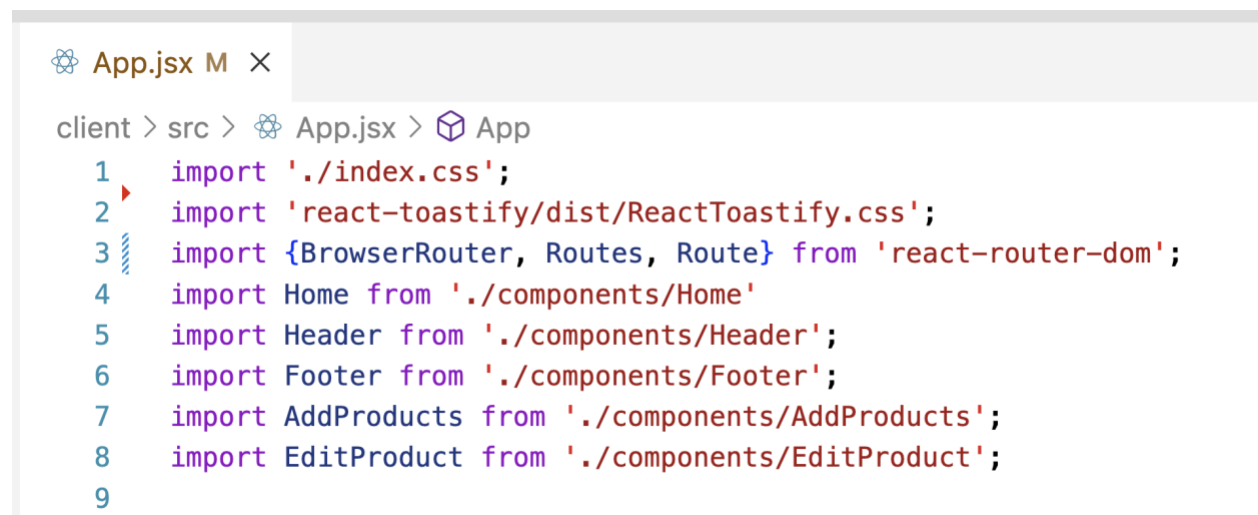
```
const handleDelete = async (id) => {  
  const isConfirmed = confirm('Are you sure you want to delete this product?')  
  if (isConfirmed) {  
    try {  
      const response = await axios.delete(`http://localhost:8000/deleteProduct/${id}`)  
      toast.success(response.data.message);  
      viewProducts()  
    }  
    catch (error) {  
      console.log(error)  
    }  
  }  
}
```

Additionally, toast messages can be used to display alert pop-ups using the `react-toastify` module. To enable this, install the module in the client application folder with the following command.

```
npm install react-toastify
```

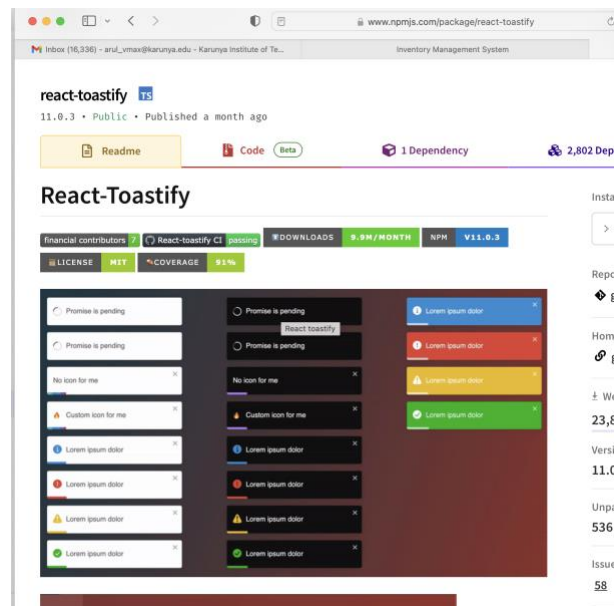
Also, include the following line in `App.jsx`.

```
import 'react-toastify/dist/ReactToastify.css';
```



```
App.jsx M ×  
client > src > App.jsx > App  
1  import './index.css';  
2  import 'react-toastify/dist/ReactToastify.css';  
3  import {BrowserRouter, Routes, Route} from 'react-router-dom';  
4  import Home from './components/Home'  
5  import Header from './components/Header';  
6  import Footer from './components/Footer';  
7  import AddProducts from './components/AddProducts';  
8  import EditProduct from './components/EditProduct';  
9
```

Refer: <https://www.npmjs.com/package/react-toastify>

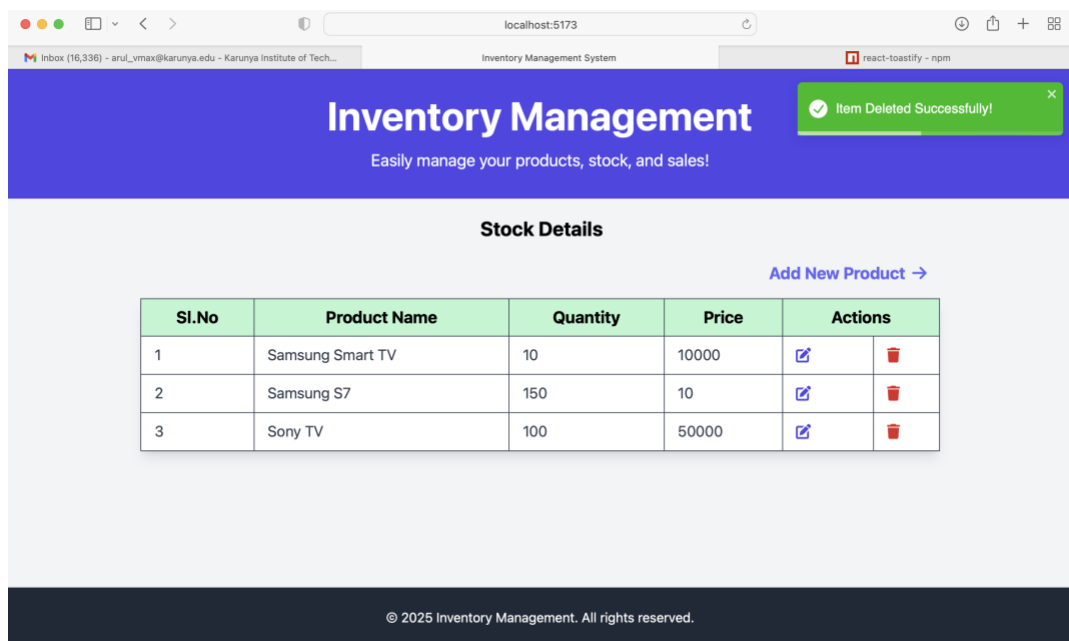


How to use react-toastify ?

```
import { ToastContainer, toast, Bounce } from 'react-toastify';
```

Then, add the following in the HTML part,

```
<ToastContainer position='top-right' autoClose={3000} hideProgressBar={false}
closeOnClick={true} transition={Bounce} theme="colored"/>
```



[Use Tailwind CSS to style the page according to your preferences.]

Add Products

```
import React, { useState } from "react";
import axios from "axios";
import { Link } from 'react-router-dom';
import { ToastContainer, toast, Bounce } from "react-toastify";

function AddProducts(props) {
  const [name, setName] = useState("")
  const [quantity, setQuantity] = useState("")
  const [price, setPrice] = useState("")
  const handleSubmit = async (e) => {
    e.preventDefault()
    try{
      const response = await
    axios.post('http://localhost:8000/addProduct',{name,quantity,price})
      toast.success(response.data.message)
    }
    catch(error){
      console.log(error)
    }
  }

  return (
    <div className="w-full max-w-lg mx-auto mt-5 mb-5">
      <h3 className="text-blue text-center">Add New Product Details</h3>
      <Link to="/">Back</Link>
      <form onSubmit={handleSubmit}>
        <div>
          <label>Product Name</label>
          <input type='text' placeholder="No of Units"
            onChange={(e)=>setName(e.target.value)></input>
        </div>
        <div className="mb-4">
          <label>Quantity</label>
          <input type="number" placeholder="No of Units"
            onChange={(e)=>setQuantity(e.target.value)}></input>
        </div>
        <div className="mb-4">
          <label>Price</label>
          <input type='number' placeholder="Unit price"
            onChange={(e)=>setPrice(e.target.value)} ></input>
        </div>
      </form>
    </div>
```



```

<div className="flex justify-center">
  <button type="submit">Add New Product</button>
</div>
<div>
  <ToastContainer position="top-left" autoClose={2000} hideProgressBar={false}
closeOnClick={true} transition={Bounce} theme="dark"/>
</div>
</form>
</div>
</>
);
}
export default AddProducts

```

localhost:5173/addproducts

Inventory Management

Easily manage your products, stock, and sales!

Product Added Successfully

Add New Product Details

[← Back](#)

Product Name

HP Laptop

Quantity

5

Price

100000

Add New Product

[Use Tailwind CSS to style the page according to your preferences.]

Edit Products

```
import React, { useEffect, useState } from "react";
import axios from "axios"
import { Link } from 'react-router-dom';
import { useParams } from "react-router-dom";
import { ToastContainer, toast, Bounce } from "react-toastify";

function EditProduct(props) {
  const [name, setName] = useState("")
  const [quantity, setQuantity] = useState("")
  const [price, setPrice] = useState("")

  const { id } = useParams();
  const findProduct = async () => {
    try{
      const response = await axios.get(`http://localhost:8000/findProduct/${id}`)
      setName(response.data.name)
      setQuantity(response.data.quantity)
      setPrice(response.data.price)
    }
    catch(error){
      console.log(error)
    }
  }
  useEffect(() => {
    findProduct()
  }, [id])
}
```

```

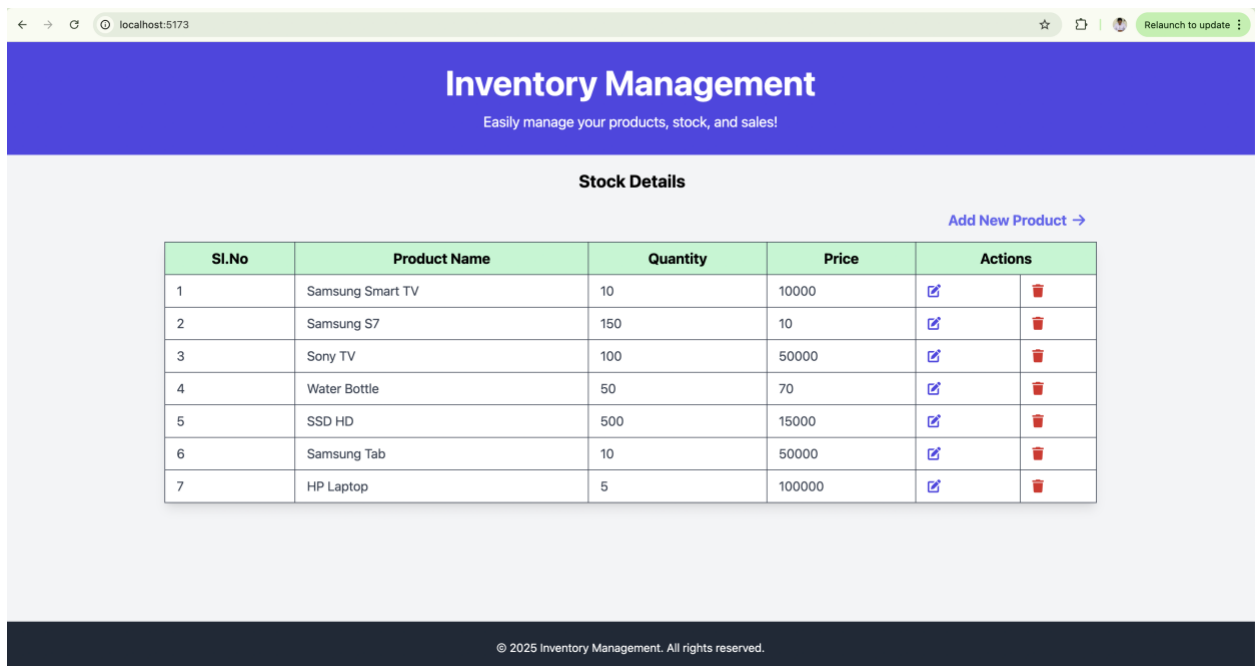
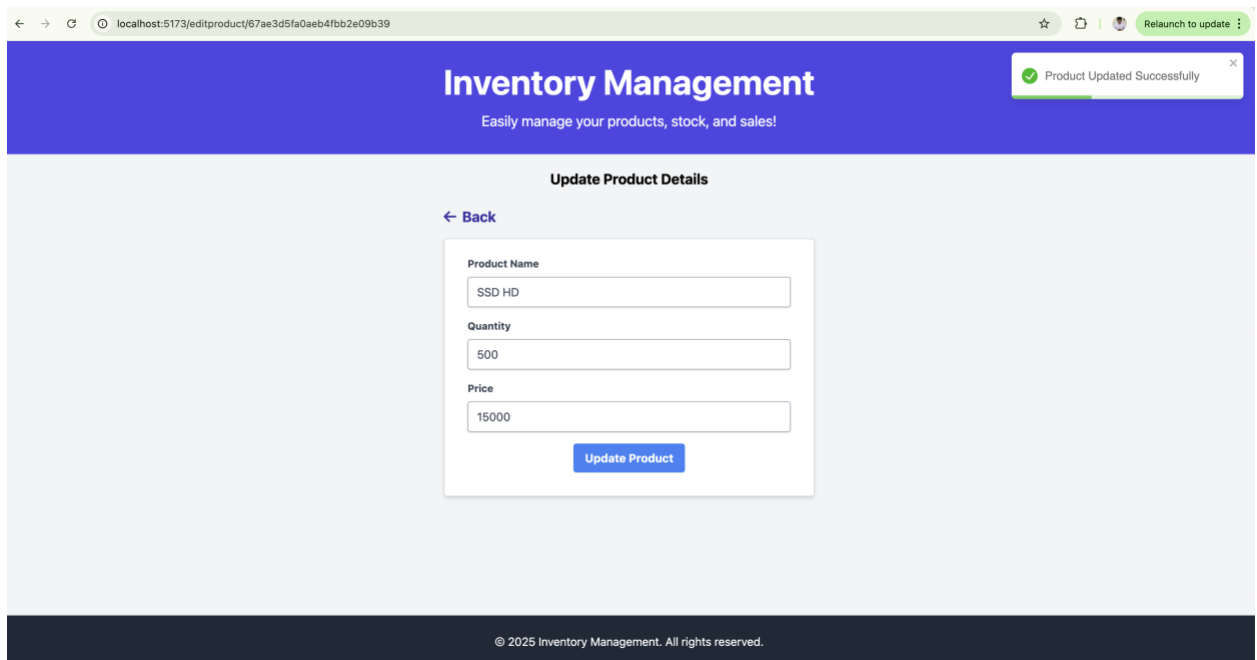
const handleSubmit = async (e) => {
  e.preventDefault()
  try {
    const response = await
    axios.put(`http://localhost:8000/editProduct/${id}`, {name, quantity, price})
    toast.success(response.data.message)
  }
  catch(error){
    console.log(error)
  }
}

return (
  <div className="w-full max-w-lg mx-auto mt-5 mb-5">
    <h3 className="text-blue text-center">Update Product Details</h3>
    <Link to="/" > Back</Link>
    <form onSubmit={handleSubmit}>
      <div className="mb-4">
        <label>Product Name</label>
        <input type='text' value={name} required
          onChange={(e)=>setName(e.target.value)}
        ></input>
      </div>
      <div className="mb-4">
        <label>Quantity</label>
        <input type="number" value={quantity} required
          onChange={(e)=>setQuantity(e.target.value)}
        ></input>
      </div>
      <div className="mb-4">
        <label>Price</label>
        <input type='number' value={price} required
          onChange={(e)=>setPrice(e.target.value)}
        ></input>
      </div>

      <div className="flex justify-center">
        <button type="submit">Update Product</button>
      </div>
    </form>
    <ToastContainer position="top-left" autoClose={2000} hideProgressBar={false}
    closeOnClick={true} transition={Bounce} theme="dark"/>
  </div>
  </>
);
}

export default EditProduct

```



[Use Tailwind CSS to style the page according to your preferences.]

Compass

My Queries

CONNECTIONS (3)

Search connections

localhost:27017

Company

Products

Users

admin

college

config

infosys

karunya

local

test

university

Products

localhost:27017 > Company > Products

Documents 7

Aggregations

Schema

Indexes 1

Validation

Type a query: { field: 'value' } or [Generate query](#)

EXPLAIN

RESET

FIND

Options

ADD DATA

EXPORT DATA

UPDATE

DELETE

251 - 7 of 7

1

2

3

4

5

6

7

8

9

10

ObjectId('67a4cd02c4517450173fdd77')

"Samsung Smart TV"

quantity: 10

price: 10000

--v: 0

ObjectId('67a4cd19c4517450173fdd7b')

"Samsung S7"

quantity: 150

price: 10

--v: 0

ObjectId('67ad084fd318c1db5a193bbd')

"Sony TV"

quantity: 100

price: 50000

--v: 0

ObjectId('67ae3d17a0eb4fbb2e09b35')

"Water Bottle"

quantity: 50

price: 70

--v: 0

ObjectId('67ae3d5fa0eb4fbb2e09b39')

"SSD HD"

quantity: 500

price: 15000