**🔰 Project Goal:**

**Seller-Frontend Website** for an e-commerce platform  
👉 Jisme seller login karke apna product add, orders manage, profile update kar sake

**✅ Step 1: Project Setup with Vite + React**

**📌 Command:**

npm create vite@latest seller-frontend -- --template react

**📌 Steps:**

cd seller-frontend

npm install

npm run dev

**🔍 Explanation:**

| **Item** | **Purpose** |
| --- | --- |
| vite | Fastest dev tool for React apps. Faster than create-react-app. |
| --template react | React ke liye Vite ka template use kar rahe hain. |
| npm run dev | Local server chalu hota hai, app browser me dikhega (default: localhost:5173) |

**✅ Step 2: Folder Structure Setup**

Create this basic structure inside src/:

src/

├── components/ 👉 Reusable UI parts (e.g., Sidebar)

├── pages/ 👉 All the main page components (Dashboard, Orders etc.)

├── layout/ 👉 Layout component for sidebar + content

├── App.jsx 👉 Main component where routing defined

├── main.jsx 👉 App root file

**✅ Step 3: Install react-router-dom for Routing**

npm install react-router-dom

**🔍 Kyu chahiye?**

**react-router-dom** humein page-to-page navigation deta hai bina page reload kiye.

**✅ Step 4: Create Pages**

Inside src/pages/, create files:

Login.jsx, Dashboard.jsx, AddProduct.jsx, MyProducts.jsx, Orders.jsx, Profile.jsx

**Example: Login.jsx**

function Login() {

return <h2>Seller Login Page</h2>;

}

export default Login;

Baaki pages bhi aise hi temporarily bana lo with different text.

**✅ Step 5: Setup Routing in App.jsx**

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

import Login from "./pages/Login";

import Dashboard from "./pages/Dashboard";

import AddProduct from "./pages/AddProduct";

import MyProducts from "./pages/MyProducts";

import Orders from "./pages/Orders";

import Profile from "./pages/Profile";

function App() {

return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Login />} />

<Route path="/dashboard" element={<Dashboard />} />

<Route path="/add-product" element={<AddProduct />} />

<Route path="/my-products" element={<MyProducts />} />

<Route path="/orders" element={<Orders />} />

<Route path="/profile" element={<Profile />} />

</Routes>

</BrowserRouter>

);

}

export default App;

**✅ Step 6: Sidebar Component (components/Sidebar.jsx)**

import { Link } from "react-router-dom";

function Sidebar() {

return (

<div style={{ width: "200px", height: "100vh", background: "#f2f2f2", padding: "20px" }}>

<h3>Seller Panel</h3>

<ul style={{ listStyle: "none", padding: 0 }}>

<li><Link to="/dashboard">Dashboard</Link></li>

<li><Link to="/add-product">Add Product</Link></li>

<li><Link to="/my-products">My Products</Link></li>

<li><Link to="/orders">Orders</Link></li>

<li><Link to="/profile">Profile</Link></li>

</ul>

</div>

);

}

export default Sidebar;

**✅ Step 7: SellerLayout (layout/SellerLayout.jsx)**

import Sidebar from "../components/Sidebar";

import { Outlet } from "react-router-dom";

function SellerLayout() {

return (

<div style={{ display: "flex" }}>

<Sidebar />

<div style={{ flex: 1, padding: "20px" }}>

<Outlet />

</div>

</div>

);

}

export default SellerLayout;

**🔍 Explanation:**

| **Part** | **Explanation** |
| --- | --- |
| <Sidebar /> | Sidebar fix dikhe har page pe |
| <Outlet /> | Nested pages (Dashboard, Orders...) yahin render honge |

**✅ Step 8: Update Routes with SellerLayout**

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

import Login from "./pages/Login";

import Dashboard from "./pages/Dashboard";

import AddProduct from "./pages/AddProduct";

import MyProducts from "./pages/MyProducts";

import Orders from "./pages/Orders";

import Profile from "./pages/Profile";

import SellerLayout from "./layout/SellerLayout";

function App() {

return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Login />} />

<Route path="/" element={<SellerLayout />}>

<Route path="dashboard" element={<Dashboard />} />

<Route path="add-product" element={<AddProduct />} />

<Route path="my-products" element={<MyProducts />} />

<Route path="orders" element={<Orders />} />

<Route path="profile" element={<Profile />} />

</Route>

</Routes>

</BrowserRouter>

);

}

export default App;

**✅ Ab kya kaam karta hai?**

* / → Login Page
* /dashboard → Dashboard page with sidebar
* /add-product → Add Product form with sidebar
* All pages sidebar ke saath hi dikhenge (thanks to SellerLayout)

Ab chalte hain **Add Product Form** banane ki taraf, jisme seller ek naya product upload karega — **image ke saath**.  
Yeh step **form handling + image preview + state management** ke liye important hai.

✅ Step 9: Add Product Form (with Image Upload)

🔧 File: pages/AddProduct.jsx

import { useState } from "react";

function AddProduct() {

const [product, setProduct] = useState({

name: "",

price: "",

description: "",

image: null,

});

const [preview, setPreview] = useState(null);

const handleChange = (e) => {

const { name, value } = e.target;

setProduct((prev) => ({

...prev,

[name]: value,

}));

};

const handleImageChange = (e) => {

const file = e.target.files[0];

setProduct((prev) => ({

...prev,

image: file,

}));

// Preview ke liye

const reader = new FileReader();

reader.onloadend = () => {

setPreview(reader.result);

};

reader.readAsDataURL(file);

};

const handleSubmit = (e) => {

e.preventDefault();

console.log("Form Data:", product);

alert("Product added (console me check karo)");

// future: send to backend using FormData + axios

};

return (

<div>

<h2>Add New Product</h2>

<form onSubmit={handleSubmit} style={{ maxWidth: "400px" }}>

<input

type="text"

name="name"

placeholder="Product Name"

value={product.name}

onChange={handleChange}

required

/><br /><br />

<input

type="number"

name="price"

placeholder="Price"

value={product.price}

onChange={handleChange}

required

/><br /><br />

<textarea

name="description"

placeholder="Description"

value={product.description}

onChange={handleChange}

required

/><br /><br />

<input type="file" accept="image/\*" onChange={handleImageChange} /><br /><br />

{preview && <img src={preview} alt="preview" width="100" />}

<br /><br />

<button type="submit">Add Product</button>

</form>

</div>

);

}

export default AddProduct;

**🔍 Full Explanation:**

| **Line** | **Kaam kya karta hai** |
| --- | --- |
| useState | Product form data store karta hai |
| handleChange | Input box se data update karta hai |
| handleImageChange | File (image) select karta hai + preview banata hai |
| FileReader | Image ko browser me dikhaata hai |
| handleSubmit | Form submit hone par product object print karta hai |

**🧪 Test karne ke liye:**

1. npm run dev
2. Go to /add-product
3. Form bharo + image choose karo
4. Submit → Console me object dikhega

Chal ab **Formik + Yup** se form validation sikhta hai — step-by-step, easy language mein.

✅ Step-by-Step: Formik + Yup Validation in Add Product Form

**📦 Step 1: Install Formik and Yup**

npm install formik yup ( **first type validation of Formik + Yup**)

🧠 Step 2: Update AddProduct.jsx with Formik + Yup

import { useState } from "react";

import { useFormik } from "formik";

import \* as Yup from "yup";

function AddProduct() {

const [preview, setPreview] = useState(null);

// 🔐 Yup validation schema

const validationSchema = Yup.object({

name: Yup.string().required("Product name is required"),

price: Yup.number()

.typeError("Price must be a number")

.positive("Price must be positive")

.required("Price is required"),

description: Yup.string()

.min(10, "Description must be at least 10 characters")

.required("Description is required"),

image: Yup.mixed().required("Image is required"),

});

// 🧰 Formik form setup

const formik = useFormik({

initialValues: {

name: "",

price: "",

description: "",

image: null,

},

validationSchema,

onSubmit: (values) => {

console.log("Form values:", values);

alert("Product added (console check karo)");

},

});

const handleImageChange = (e) => {

const file = e.target.files[0];

formik.setFieldValue("image", file);

// Image preview

const reader = new FileReader();

reader.onloadend = () => {

setPreview(reader.result);

};

reader.readAsDataURL(file);

};

return (

<div>

<h2>Add New Product</h2>

<form onSubmit={formik.handleSubmit} style={{ maxWidth: "400px" }}>

{/\* Name \*/}

<input

type="text"

name="name"

placeholder="Product Name"

value={formik.values.name}

onChange={formik.handleChange}

onBlur={formik.handleBlur}

/>

{formik.touched.name && formik.errors.name && (

<div style={{ color: "red" }}>{formik.errors.name}</div>

)}

<br /><br />

{/\* Price \*/}

<input

type="text"

name="price"

placeholder="Price"

value={formik.values.price}

onChange={formik.handleChange}

onBlur={formik.handleBlur}

/>

{formik.touched.price && formik.errors.price && (

<div style={{ color: "red" }}>{formik.errors.price}</div>

)}

<br /><br />

{/\* Description \*/}

<textarea

name="description"

placeholder="Description"

value={formik.values.description}

onChange={formik.handleChange}

onBlur={formik.handleBlur}

/>

{formik.touched.description && formik.errors.description && (

<div style={{ color: "red" }}>{formik.errors.description}</div>

)}

<br /><br />

{/\* Image \*/}

<input type="file" accept="image/\*" onChange={handleImageChange} />

{formik.touched.image && formik.errors.image && (

<div style={{ color: "red" }}>{formik.errors.image}</div>

)}

<br /><br />

{preview && <img src={preview} alt="preview" width="100" />}

<br /><br />

<button type="submit">Add Product</button>

</form>

</div>

);

}

export default AddProduct;

**🔍 Samjhauta (Explanation):**

| **Concept** | **Kya karta hai?** |
| --- | --- |
| useFormik() | Form handle karta hai (values, submit, errors) |
| Yup.object() | Rules banata hai har field ke liye |
| setFieldValue() | Manually image set karta hai Formik ke state me |
| touched + errors | Error messages tab dikhte hain jab user ne input touch kiya ho |

**useFormik ka full structure**

const formik = useFormik({

initialValues: { ... },

validationSchema: ...,

onSubmit: (values, { resetForm }) => {

// use resetForm here

}

});

**✅ Goal:**

**Form submit hone ke baad:**

* Sab fields blank ho jayein
* Image preview bhi hata diya jaye

**✅ Correct Way to Use resetForm**

Formik ka onSubmit function ka **second parameter** hota hai formikHelpers, jisme resetForm hota hai.

**✅ Fix your code like this:**

onSubmit: (values, { resetForm }) => {

console.log("Form values:", values);

alert("Product added (console check karo)");

resetForm(); // ✅ Form clear

setPreview(null); // ✅ Image preview clear

},

**🔍 Explanation:**

Formik internally onSubmit(values, formikHelpers) deta hai  
Aur formikHelpers ke andar hota hai:

* resetForm()
* setSubmitting()
* setErrors() etc.
* **🔍 Explanation:**

| **Line** | **Kya karta hai?** |
| --- | --- |
| resetForm() | Formik ke saare input clear kar deta hai |
| setPreview(null) | Image preview hata deta hai browser se |

✅ **What is Formik and Yup?**

Ye dono tools React projects ke forms ke liye **bohot important** hain.

**✅ Formik — Full Definition & Explanation**

**📘 Definition:**

**Formik** ek React library hai jo form banane, input handle karne, aur validation manage karne ko asaan banaata hai.

**🔧 Problem without Formik:**

Form banana React mein manually:

* Har input ke liye useState banana padta hai
* onChange, onBlur, value sab manually likhna padta hai
* Validation bhi manually karna padta hai (bohot boring & error-prone)
* **✅ Formik kya karta hai?**

| **Feature** | **Kya kaam karta hai** |
| --- | --- |
| initialValues | Form ke initial values set karta hai |
| handleChange | Input field me typing detect karta hai |
| handleBlur | Jab user field chhodega tab run hota hai |
| values | Current values of all inputs |
| errors | Validation errors show karta hai |
| touched | Field touch hua ya nahi yeh track karta hai |
| handleSubmit | Form submit hone pe kya karna hai, yeh control karta hai |

**✅ Basic Example:**

const formik = useFormik({

initialValues: {

name: '',

email: ''

},

onSubmit: (values) => {

console.log(values);

}

});

**✅ Yup — Full Definition & Explanation**

**📘 Definition:**

**Yup** ek JavaScript library hai jo validation schema banata hai.  
Yeh Formik ke sath milke form ke har field ko validate karta hai.

**🔧 Without Yup:**

* Tu manually if/else se check karega:
  + Kya name blank hai?
  + Kya price number hai?
  + Kya description lamba hai?

Bohot boring + lengthy code

**✅ Yup kya karta hai?**

| **Field Type** | **Validation** |
| --- | --- |
| string() | Text input ke liye |
| number() | Numbers ke liye |
| boolean() | Checkbox, switch etc. |
| mixed() | File/image jaise non-standard data |
| required() | Required banata hai |
| min() / max() | Length ya value ki limit |

**✅ Basic Yup Example:**

const validationSchema = Yup.object({

name: Yup.string().required("Name is required"),

price: Yup.number().required("Price is required").positive("Must be positive"),

});

**🧠 Formik + Yup Together:**

const formik = useFormik({

initialValues: { name: '', price: '' },

validationSchema: Yup.object({

name: Yup.string().required("Name chahiye"),

price: Yup.number().positive().required("Price chahiye"),

}),

onSubmit: (values) => {

console.log(values);

}

});

**✅ Final Benefits of Formik + Yup:**

| **Feature** | **Benefit** |
| --- | --- |
| Less Code | useState har field ke liye nahi banana padta |
| Easy Validation | Yup se simple rules likh lo |
| Error Handling | errors.name, touched.name se easy error display |
| Clean Structure | Formik pura structure manage karta hai |
| Scalable | Bada form ho ya chhota — dono me useful |

✅ **Backend Setup Overview:**

**🧱 Tech Stack:**

* Node.js + Express → Server
* Multer → File (image) upload
* MongoDB → Data save karne ke liye
* Mongoose → MongoDB ke liye schema/model banane ke liye

**✅ Step-by-Step Backend Plan**

**✅ Step 1: Create backend folder**

mkdir backend

cd backend

npm init -y

**✅ Step 2: Install required packages**

npm install express mongoose multer cors

**📌 Explanation:**

| **Package** | **Use** |
| --- | --- |
| express | Server create karne ke liye |
| mongoose | MongoDB se connect aur model banane ke liye |
| multer | Image/file upload ke liye |
| cors | Frontend-backend connection ke liye |

**✅ Step 3: Folder structure**

backend/

├── uploads/ 👉 Images yahan save hongi

├── models/Product.js 👉 MongoDB schema

├── routes/productRoute.js👉 All product APIs

├── server.js 👉 Entry point

**✅ Step 4: server.js (backend ka entry point)**

const express = require("express");

const mongoose = require("mongoose");

const cors = require("cors");

const productRoutes = require("./routes/productRoute");

const app = express();

// Middleware

app.use(cors());

app.use(express.json());

app.use("/uploads", express.static("uploads")); // Static image access

// Routes

app.use("/api/products", productRoutes);

// Connect MongoDB

mongoose

.connect("mongodb://127.0.0.1:27017/seller-ecom", {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log("MongoDB Connected"))

.catch((err) => console.log(err));

app.listen(5000, () => {

console.log("Server running on http://localhost:5000");

});

**✅ Step 5: Create models/Product.js**

const mongoose = require("mongoose");

const productSchema = new mongoose.Schema({

name: String,

price: Number,

description: String,

imageUrl: String,

});

module.exports = mongoose.model("Product", productSchema);

**✅ Step 6: Create routes/productRoute.js**

const express = require("express");

const router = express.Router();

const multer = require("multer");

const Product = require("../models/Product");

// 🔧 Multer config

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, "uploads/");

},

filename: function (req, file, cb) {

cb(null, Date.now() + "-" + file.originalname);

},

});

const upload = multer({ storage: storage });

// ✅ POST route: Add product

router.post("/add", upload.single("image"), async (req, res) => {

try {

const { name, price, description } = req.body;

const imageUrl = req.file ? req.file.path : "";

const product = new Product({ name, price, description, imageUrl });

await product.save();

res.status(201).json({ message: "Product added successfully", product });

} catch (err) {

res.status(500).json({ message: "Something went wrong", error: err });

}

});

module.exports = router;

**✅ Step 7: Start the backen**

node server.js

If working correctly, you’ll see:

Server running on http://localhost:5000

MongoDB Connected

**✅ Step 8: Update React form to send to backend**

Next step: Main tujhko sikhaunga **React form se image + data backend me kaise bhejna hai using Axios and FormData**

**✅ Goal:**

* Form submit karte hi image aur data backend (Multer + MongoDB) me save ho jaye

✅ Step-by-Step: React to Backend Connection

**✅ Step 1: Install Axios in frontend**

Go to your React app:

npm install axios

**✅ Step 2: Update AddProduct.jsx**

Full working version:

import { useState } from "react";

import { useFormik } from "formik";

import \* as Yup from "yup";

import axios from "axios";

function AddProduct() {

const [preview, setPreview] = useState(null);

const [success, setSuccess] = useState(false);

const validationSchema = Yup.object({

name: Yup.string().required("Name is required"),

price: Yup.number()

.typeError("Price must be a number")

.positive("Must be positive")

.required("Price is required"),

description: Yup.string()

.min(10, "Description must be at least 10 characters")

.required("Description is required"),

image: Yup.mixed().required("Image is required"),

});

const formik = useFormik({

initialValues: {

name: "",

price: "",

description: "",

image: null,

},

validationSchema,

onSubmit: async (values, { resetForm }) => {

try {

const formData = new FormData();

formData.append("name", values.name);

formData.append("price", values.price);

formData.append("description", values.description);

formData.append("image", values.image);

const res = await axios.post(

"http://localhost:5000/api/products/add",

formData,

{

headers: {

"Content-Type": "multipart/form-data",

},

}

);

console.log("Server Response:", res.data);

setSuccess(true);

resetForm();

setPreview(null);

setTimeout(() => setSuccess(false), 3000);

} catch (error) {

console.error("Upload error:", error);

alert("Failed to upload product.");

}

},

});

const handleImageChange = (e) => {

const file = e.target.files[0];

formik.setFieldValue("image", file);

const reader = new FileReader();

reader.onloadend = () => {

setPreview(reader.result);

};

reader.readAsDataURL(file);

};

return (

<div>

<h2>Add Product</h2>

{success && <p style={{ color: "green" }}>✅ Product uploaded!</p>}

<form onSubmit={formik.handleSubmit} style={{ maxWidth: "400px" }}>

<input

type="text"

name="name"

placeholder="Product Name"

value={formik.values.name}

onChange={formik.handleChange}

onBlur={formik.handleBlur}

/>

{formik.touched.name && formik.errors.name && (

<div style={{ color: "red" }}>{formik.errors.name}</div>

)}

<br />

<input

type="text"

name="price"

placeholder="Price"

value={formik.values.price}

onChange={formik.handleChange}

onBlur={formik.handleBlur}

/>

{formik.touched.price && formik.errors.price && (

<div style={{ color: "red" }}>{formik.errors.price}</div>

)}

<br />

<textarea

name="description"

placeholder="Description"

value={formik.values.description}

onChange={formik.handleChange}

onBlur={formik.handleBlur}

/>

{formik.touched.description && formik.errors.description && (

<div style={{ color: "red" }}>{formik.errors.description}</div>

)}

<br />

<input type="file" accept="image/\*" onChange={handleImageChange} />

{formik.touched.image && formik.errors.image && (

<div style={{ color: "red" }}>{formik.errors.image}</div>

)}

<br />

{preview && <img src={preview} alt="preview" width="100" />}

<br /><br />

<button type="submit">Add Product</button>

</form>

</div>

);

}

export default AddProduct;

**✅ Summary of Important Parts:**

| **Feature** | **Explanation** |
| --- | --- |
| FormData() | Image + text data ko multipart format me backend bhejne ke liye |
| axios.post() | Backend ko request bhejta hai |
| Content-Type | "multipart/form-data" — required for file uploads |
| setPreview(null) | Image preview clear after upload |
| resetForm() | Form inputs clear after submit |

**✅ Test Now:**

1. Start backend: node server.js
2. Start frontend: npm run dev
3. Go to /add-product
4. Fill form + choose image
5. Submit → Check terminal & MongoDB

**Tu chahe to:**

* Uploaded image dekh sakta hai at: [http://localhost:5000/uploads/<filename](http://localhost:5000/uploads/%3cfilename)>
  + http://localhost:5000/uploads/1752201331444-smartWatch1.jpg
* MongoDB me seller-ecom.products collection me document create ho jaayega.

**✅ My Products Page**

Jahan seller apne saare products dekh sakta hai — jo usne add kiye hain.

**📦 Goal:**

* Backend se saare products fetch karo
* Frontend me **list + image** show karo
* Future me: Edit / Delete buttons bhi add karenge
* **🔧 Backend: Add GET Route**
* File: routes/productRoute.js  
  Add this route at bottom:

// ✅ GET route: Get all products

router.get("/", async (req, res) => {

try {

const products = await Product.find().sort({ \_id: -1 }); // latest first

res.json(products);

} catch (err) {

res.status(500).json({ message: "Error fetching products" });

}

});

**🧠 Frontend: Create MyProducts.jsx Page**

File: pages/MyProducts.jsx

import { useEffect, useState } from "react";

import axios from "axios";

function MyProducts() {

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

useEffect(() => {

// ✅ Get data from backend

axios

.get("http://localhost:5000/api/products")

.then((res) => setProducts(res.data))

.catch((err) => console.error("Error fetching:", err));

}, []);

return (

<div>

<h2>My Products</h2>

{products.length === 0 ? (

<p>No products added yet.</p>

) : (

<div style={{ display: "flex", flexWrap: "wrap", gap: "20px" }}>

{products.map((product) => (

<div

key={product.\_id}

style={{

border: "1px solid #ccc",

padding: "10px",

width: "200px",

}}

>

<img

src={`http://localhost:5000/${product.imageUrl}`}

alt={product.name}

style={{ width: "100%", height: "150px", objectFit: "cover" }}

/>

<h4>{product.name}</h4>

<p>₹{product.price}</p>

<small>{product.description}</small>

</div>

))}

</div>

)}

</div>

);

}

export default MyProducts;

**✅ Explanation:**

| **Part** | **Kya karta hai** |
| --- | --- |
| axios.get() | Backend se product list laata hai |
| useEffect() | Page load pe hi call karta hai |
| setProducts() | Data ko state me store karta hai |
| map() | Har product ka card banata hai |
| img src={...} | Image ko backend ke static URL se dikhata hai |

**🧪 Test kar:**

1. Add few products using /add-product
2. Go to /my-products
3. Dekh tere products list ho rahe hain ya nahi

**✅ Next Options:**

1. **🗑 Delete product**
2. **✏️ Edit product**
3. **🔍 Search / filter**
4. **🧾 View product detail page**
5. **🌐 Deploy frontend + backend online**

**🗑 Delete product**

**✅ Goal:**

* Seller 🗑️ button pe click kare
* Product backend + MongoDB se delete ho
* Frontend me turant list update ho

🔧 Step 1: Backend DELETE route (routes/productRoute.js)

// DELETE route: delete a product by ID

router.delete("/:id", async (req, res) => {

try {

const { id } = req.params;

await Product.findByIdAndDelete(id);

res.json({ message: "Product deleted successfully" });

} catch (err) {

res.status(500).json({ message: "Failed to delete product" });

}

});

**Step 2: Frontend UI me Delete Button**

Update pages/MyProducts.jsx:

1️⃣ Import icon & axios:

import { Trash } from "lucide-react"; // optional, ya simple button use kar

2️⃣ Add delete handler function:

const handleDelete = async (id) => {

if (!window.confirm("Are you sure to delete this product?")) return;

try {

await axios.delete(`http://localhost:5000/api/products/${id}`);

setProducts((prev) => prev.filter((item) => item.\_id !== id)); // remove from UI

} catch (err) {

console.error("Delete failed", err);

alert("Failed to delete product");

}

};

3️⃣ Add delete button inside .map():

<button

onClick={() => handleDelete(product.\_id)}

style={{

background: "red",

color: "white",

border: "none",

padding: "5px 10px",

cursor: "pointer",

marginTop: "10px",

}}

>

🗑 Delete

</button>

**✅ Summary:**

| **Kaam** | **Tool/Code** |
| --- | --- |
| Backend delete | Product.findByIdAndDelete(id) |
| Frontend delete | axios.delete(...) |
| UI update | setProducts() se item hata diya |

**✅ Test It:**

1. Go to /my-products
2. Kisi product ke "🗑 Delete" button pe click karo
3. Confirm → Product turant hat jayega
4. Check MongoDB — document gaya ✅

**✏️ Edit product**

**✅ Goal:**

* MyProducts me har product ke saath ✏️ **Edit button**
* Click karne pe ek **EditProduct** page open ho
* Wahan pe form me old values already filled ho (pre-filled)
* User edit kare → Save kare → Data backend + MongoDB me update ho

🔧 Step-by-Step Implementation

**✅ Step 1: Backend PUT Route (routes/productRoute.js)**

// ✅ Update product by ID

router.put("/:id", async (req, res) => {

try {

const { id } = req.params;

const { name, price, description } = req.body;

const updated = await Product.findByIdAndUpdate(

id,

{ name, price, description },

{ new: true }

);

res.json({ message: "Product updated", product: updated });

} catch (err) {

res.status(500).json({ message: "Update failed", error: err });

}

});

⚠️ Ye route PUT /api/products/:id hai — frontend isse call karega.

**✅ Step 2: Add Edit Button in MyProducts.jsx**

<button

onClick={() => navigate(`/edit-product/${product.\_id}`)}

style={{

background: "orange",

color: "white",

border: "none",

padding: "5px 10px",

cursor: "pointer",

marginTop: "10px",

}}

>

✏️ Edit

</button>

Don’t forget:

import { useNavigate } from "react-router-dom";

const navigate = useNavigate();

**✅ Step 3: Add Route in App.jsx or wherever routes defined**

<Route path="/edit-product/:id" element={<EditProduct />} />

**✅ Step 4: Create EditProduct.jsx Page with Formik + Yup validation (Second type validation of Formik + Yup)**

import { useEffect, useState } from "react";

import { useParams, useNavigate } from "react-router-dom";

import axios from "axios";

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as Yup from "yup";

function EditProduct() {

  const { id } = useParams();

  const navigate = useNavigate();

  const [initialValues, setInitialValues] = useState({

    name: "",

    price: "",

    description: "",

  });

  const [image, setImage] = useState(null);

  const [preview, setPreview] = useState(null);

  useEffect(() => {

    axios

      .get(`http://localhost:5000/api/products/${id}`)

      .then((res) => {

        const data = res.data;

        setInitialValues({

          name: data.name || "",

          price: data.price || "",

          description: data.description || "",

        });

        if (data.imageUrl) {

          const fullUrl = `http://localhost:5000/${data.imageUrl.replace(

            /\\/g,

            "/"

          )}`;

          setPreview(fullUrl);

        }

      })

      .catch((err) => console.log("Error loading product", err));

  }, [id]);

  const validationSchema = Yup.object({

    name: Yup.string().required("Name is required"),

    price: Yup.number()

      .typeError("Price must be a number")

      .positive("Price must be positive")

      .required("Price is required"),

    description: Yup.string().required("Description is required"),

  });

  const handleImageChange = (e) => {

    const file = e.target.files[0];

    if (file) {

      setImage(file);

      setPreview(URL.createObjectURL(file));

    }

  };

  const handleSubmit = async (values) => {

    try {

      const filledValues = {

        name: values.name.trim() || initialValues.name,

        price: values.price || initialValues.price,

        description: values.description.trim() || initialValues.description,

      };

      const formData = new FormData();

      formData.append("name", filledValues.name);

      formData.append("price", filledValues.price);

      formData.append("description", filledValues.description);

      if (image) {

        formData.append("image", image);

      }

      await axios.put(`http://localhost:5000/api/products/${id}`, formData, {

        headers: { "Content-Type": "multipart/form-data" },

      });

      alert("✅ Product updated");

      navigate("/my-products");

    } catch (err) {

      alert("❌ Update failed");

    }

  };

  return (

    <div>

      <h2>Edit Product</h2>

      {/\* ✅ Image preview \*/}

      {preview && <img src={preview} width="100" alt="Preview" />}

      <br />

      <br />

      <Formik

        enableReinitialize

        initialValues={initialValues}

        validationSchema={validationSchema}

        onSubmit={handleSubmit}

      >

        <Form style={{ maxWidth: "400px" }}>

          <Field type="text" name="name" placeholder="Product Name" />

          <ErrorMessage name="name" component="div" style={{ color: "red" }} />

          <br />

          <br />

          <Field type="number" name="price" placeholder="Price" />

          <ErrorMessage name="price" component="div" style={{ color: "red" }} />

          <br />

          <br />

          <Field as="textarea" name="description" placeholder="Description" />

          <ErrorMessage

            name="description"

            component="div"

            style={{ color: "red" }}

          />

          <br />

          <br />

          <input type="file" accept="image/\*" onChange={handleImageChange} />

          <br />

          <br />

          <button type="submit">Save Changes</button>

        </Form>

      </Formik>

    </div>

  );

}

export default EditProduct;

**✅ Bonus:**

name, price, description and image edit kar rahe hain.  
Image update ke liye Multer handling ka use hua hai.

**✅ Test It:**

1. Go to /my-products
2. Kisi product ke ✏️ Edit pe click karo
3. Edit form khulega (old values pre-filled)
4. Change karo → Save karo → Wapas product list me changes dikhenge

**🔧 Backend + Multer Image Update Flow (Already Covered But Confirm)**

**✅ 1. Multer Setup in productRoute.js**

const multer = require("multer");

const storage = multer.diskStorage({

destination: (req, file, cb) => {

cb(null, "uploads/");

},

filename: (req, file, cb) => {

cb(null, Date.now() + "-" + file.originalname);

},

});

const upload = multer({ storage });

**✅ 2. PUT Route to Update Product (With Optional Image)**

router.put("/:id", upload.single("image"), async (req, res) => {

try {

const { name, price, description } = req.body;

const id = req.params.id;

const updateData = { name, price, description };

if (req.file) {

updateData.imageUrl = req.file.path; // 👈 update image path too

}

const updated = await Product.findByIdAndUpdate(id, updateData, { new: true });

res.json({ message: "Product updated", product: updated });

} catch (err) {

res.status(500).json({ message: "Update failed", error: err.message });

}

});

**🧪 Final Test Flow:**

1. Go to /edit-product/:id
2. Purana data + image visible ho
3. Choose new image → preview change ho
4. Submit → Image backend me update ho
5. MyProducts page pe new image dikhe ✅

**✅ Basic Structure of Formik + Yup Form**

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as Yup from "yup";

function MyForm() {

const initialValues = {

name: "",

email: "",

age: "",

};

const validationSchema = Yup.object({

name: Yup.string().required("Name is required"),

email: Yup.string().email("Invalid email").required("Email is required"),

age: Yup.number().positive("Must be positive").integer("Must be an integer"),

});

const handleSubmit = (values) => {

console.log("Form data:", values);

alert("Form submitted!");

};

return (

<Formik

initialValues={initialValues}

validationSchema={validationSchema}

onSubmit={handleSubmit}

>

<Form>

<label>Name:</label>

<Field type="text" name="name" />

<ErrorMessage name="name" component="div" style={{ color: "red" }} />

<label>Email:</label>

<Field type="email" name="email" />

<ErrorMessage name="email" component="div" style={{ color: "red" }} />

<label>Age:</label>

<Field type="number" name="age" />

<ErrorMessage name="age" component="div" style={{ color: "red" }} />

<button type="submit">Submit</button>

</Form>

</Formik>

);

}

export default MyForm;

**🔍 Structure Breakdown:**

| Part | Explanation |
| --- | --- |
| initialValues | Form ke default values |
| validationSchema | Yup se field validation rules |
| <Formik> | Wrapper that controls the whole form |
| <Form> | React form (Formik's version) |
| <Field> | Input field controlled by Formik |
| <ErrorMessage> | Error show karega if validation fail ho |

**📦 Example Yup Rules:**

Yup.string().required("Required")

Yup.number().positive("Must be positive").integer()

Yup.string().email("Invalid email")

Yup.string().min(5, "Too short")

Yup.string().max(20, "Too long")

**✅ Step 2: Update AddProduct.jsx with Formik + Yup + Image Upload**

Ab start karte hain **“Add Product” Form** using **Formik + Yup** step-by-step ✅  
Yeh form image upload bhi karega (with preview), aur har field ka validation karega.

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as Yup from "yup";

import { useState } from "react";

import axios from "axios";

function AddProduct() {

const [image, setImage] = useState(null);

const [preview, setPreview] = useState(null);

const initialValues = {

name: "",

price: "",

description: "",

};

const validationSchema = Yup.object({

name: Yup.string().required("Name is required"),

price: Yup.number()

.typeError("Price must be a number")

.positive("Price must be positive")

.required("Price is required"),

description: Yup.string().required("Description is required"),

});

const handleImageChange = (e) => {

const file = e.target.files[0];

if (file) {

setImage(file);

setPreview(URL.createObjectURL(file));

}

};

const handleSubmit = async (values, { resetForm }) => {

try {

const formData = new FormData();

formData.append("name", values.name);

formData.append("price", values.price);

formData.append("description", values.description);

if (image) {

formData.append("image", image);

}

await axios.post("http://localhost:5000/api/products/add", formData, {

headers: { "Content-Type": "multipart/form-data" },

});

alert("✅ Product added successfully");

resetForm();

setImage(null);

setPreview(null);

} catch (err) {

console.error("Error:", err);

alert("❌ Failed to add product");

}

};

return (

<div>

<h2>Add Product</h2>

{/\* ✅ Image Preview \*/}

{preview && <img src={preview} width="100" alt="Preview" />}

<br /><br />

<Formik

initialValues={initialValues}

validationSchema={validationSchema}

onSubmit={handleSubmit}

>

<Form style={{ maxWidth: "400px" }}>

<label>Product Name:</label>

<Field type="text" name="name" />

<ErrorMessage name="name" component="div" style={{ color: "red" }} />

<br /><br />

<label>Price:</label>

<Field type="number" name="price" />

<ErrorMessage name="price" component="div" style={{ color: "red" }} />

<br /><br />

<label>Description:</label>

<Field as="textarea" name="description" />

<ErrorMessage name="description" component="div" style={{ color: "red" }} />

<br /><br />

<label>Image:</label>

<input type="file" accept="image/\*" onChange={handleImageChange} />

<br /><br />

<button type="submit">Add Product</button>

</Form>

</Formik>

</div>

);

}

export default AddProduct;

**✅ Features:**

| **Feature** | **Description** |
| --- | --- |
| Formik | Handles form data |
| Yup | Validates inputs |
| FormData | Sends text + file to backend |
| Image Preview | Shows selected image |
| resetForm | Clears form after success |

**🔧 Backend Route (for reference):**

Ensure your backend has this POST route:

router.post("/add", upload.single("image"), async (req, res) => {

const { name, price, description } = req.body;

const imageUrl = req.file ? req.file.path : "";

const product = new Product({ name, price, description, imageUrl });

await product.save();

res.status(201).json({ message: "Product added successfully", product });

});

**✅ Ab Tu Kya Kar Chuka Hai:**

| **Feature** | **Status** |
| --- | --- |
| Add Product (with image + validation) | ✅ Done |
| Show All Products (MyProducts.jsx) | ✅ Done |
| Delete Product button | ✅ Done |
| Edit Product (image + Formik) | ✅ Done |

**✅ Filter Search Structure – Name + Price**

const [search, setSearch] = useState("");

const filteredProducts = products.filter((product) => {

const nameMatch = product.name

.toLowerCase()

.includes(search.toLowerCase());

const priceMatch = product.price

.toString()

.includes(search.toLowerCase());

return nameMatch || priceMatch;

});

**✅ Use It Like:**

<input

type="text"

placeholder="Search by name or price"

value={search}

onChange={(e) => setSearch(e.target.value)}

/>

{filteredProducts.map((product) => (

<div key={product.\_id}>

<h4>{product.name}</h4>

<p>₹{product.price}</p>

</div>

))}

**✅ Updated & Friendly MyProducts.jsx Search with Name + Price**

import { useEffect, useState } from "react";

import { useNavigate } from "react-router-dom";

import axios from "axios";

function MyProducts() {

  const navigate = useNavigate();

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

  const [search, setSearch] = useState("");

  useEffect(() => {

    axios

      .get("http://localhost:5000/api/products")

      .then((res) => setProducts(res.data))

      .catch((err) => console.error("Error fetching:", err));

  }, []);

  const handleDelete = async (id) => {

    if (!window.confirm("Are you sure to delete this product?")) return;

    try {

      await axios.delete(`http://localhost:5000/api/products/${id}`);

      setProducts((prev) => prev.filter((item) => item.\_id !== id));

    } catch (err) {

      console.error("Delete failed", err);

      alert("Failed to delete product");

    }

  };

  const filtered = products.filter((p) => {

    const nameMatch = p.name.toLowerCase().includes(search.toLowerCase());

    const priceMatch = p.price.toString().includes(search); // convert price to string

    return nameMatch || priceMatch;

  });

  return (

    <div style={{ padding: "20px" }}>

      <h2>My Products</h2>

      {/\*  Search Bar \*/}

      <input

        type="text"

        placeholder="🔍 Search by name. or price..."

        value={search}

        onChange={(e) => setSearch(e.target.value)}

        style={{

          padding: "10px",

          width: "300px",

          marginBottom: "20px",

          borderRadius: "5px",

          border: "1px solid #ccc",

          fontSize: "16px",

        }}

      />

      {/\*  No products at all \*/}

      {products.length === 0 ? (

        <p>No products added yet.</p>

      ) : (

        <>

          {/\*  No matching search result \*/}

          {filtered.length === 0 ? (

            <p>No products match your search.</p>

          ) : (

            <div style={{ display: "flex", flexWrap: "wrap", gap: "20px" }}>

              {filtered.map((product) => (

                <div

                  key={product.\_id}

                  style={{

                    border: "1px solid #ccc",

                    padding: "10px",

                    width: "200px",

                    borderRadius: "8px",

                    boxShadow: "2px 2px 10px rgba(0,0,0,0.1)",

                  }}

                >

                  <img

                    src={`http://localhost:5000/${product.imageUrl}`}

                    alt={product.name}

                    style={{

                      width: "100%",

                      height: "150px",

                      objectFit: "cover",

                      borderRadius: "6px",

                    }}

                  />

                  <h4>{product.name}</h4>

                  <p>₹{product.price}</p>

                  <small>{product.description}</small>

                  <br />

                  <button

                    onClick={() => handleDelete(product.\_id)}

                    style={{

                      background: "red",

                      color: "white",

                      border: "none",

                      padding: "5px 10px",

                      cursor: "pointer",

                      marginTop: "10px",

                      borderRadius: "4px",

                    }}

                  >

                    Delete

                  </button>

                  <button

                    onClick={() => navigate(`/edit-product/${product.\_id}`)}

                    style={{

                      background: "orange",

                      color: "white",

                      border: "none",

                      padding: "5px 10px",

                      cursor: "pointer",

                      marginTop: "10px",

                      marginLeft: "10px",

                      borderRadius: "4px",

                    }}

                  >

                    Edit

                  </button>

                </div>

              ))}

            </div>

          )}

        </>

      )}

    </div>

  );

}

export default MyProducts;

**✅ Output:**

* ✅ Live search bar
* ✅ Real-time filtering without reload
* ✅ Case-insensitive match
* ✅ Product name ke base pe filter ho raha hai

**✅ Authentication Flow:**

Chalo ab start karte hain **Authentication (Login/Signup system)** using **React (frontend)** + **Node/Express + MongoDB (backend)**

| **Step** | **Feature** | | |
| --- | --- | --- | --- |
| 1️⃣ | Signup (register seller) | |
| 2️⃣ | Login (with JWT token) | |
| 3️⃣ | Protect Routes (dashboard, product add/edit) | |
| 4️⃣ | Logout (clear token) | |

**🔧 Let's Start With Step 1 — Signup API + Frontend**

**📁 Backend Folder Structure:**

backend/

├── models/User.js

├── routes/authRoute.js

├── server.js

**✅ 1. User Model – models/User.js**

const mongoose = require("mongoose");

const bcrypt = require("bcrypt");

const userSchema = new mongoose.Schema({

name: String,

email: { type: String, required: true, unique: true },

password: { type: String, required: true },

});

// password hash before save

userSchema.pre("save", async function (next) {

if (!this.isModified("password")) return next();

this.password = await bcrypt.hash(this.password, 10);

next();

});

module.exports = mongoose.model("User", userSchema);

**✅ 2. Auth Routes – routes/authRoute.js**

const express = require("express");

const router = express.Router();

const jwt = require("jsonwebtoken");

const bcrypt = require("bcrypt");

const User = require("../models/User");

const SECRET = "jwt\_secret"; // Replace with dotenv later

// Signup route

router.post("/signup", async (req, res) => {

try {

const { name, email, password } = req.body;

const exist = await User.findOne({ email });

if (exist) return res.status(400).json({ message: "Email already used" });

const user = new User({ name, email, password });

await user.save();

res.status(201).json({ message: "Signup successful" });

} catch (err) {

res.status(500).json({ message: "Signup failed", error: err.message });

}

});

// Login route

router.post("/login", async (req, res) => {

try {

const { email, password } = req.body;

const user = await User.findOne({ email });

if (!user) return res.status(404).json({ message: "User not found" });

const match = await bcrypt.compare(password, user.password);

if (!match) return res.status(400).json({ message: "Invalid credentials" });

const token = jwt.sign({ id: user.\_id }, SECRET, { expiresIn: "7d" });

res.json({ message: "Login successful", token, user: { name: user.name, email: user.email } });

} catch (err) {

res.status(500).json({ message: "Login failed", error: err.message });

}

});

module.exports = router;

**✅ 3. Update server.js (backend root)**

const authRoutes = require("./routes/authRoute");

// Already added:

// app.use(cors());

// app.use(express.json());

app.use("/api/auth", authRoutes);

**✅ Now Frontend Signup Form (React)**

Chalo start karte hain **Signup form (React)** using **Formik + Yup + Axios**

**🏗️ Signup Form Structure:**

| **Tool** | **Use Case** |
| --- | --- |
| Formik | Form handling (values, submit) |
| Yup | Validation (email, password check) |
| Axios | API call to backend (/signup) |

**✅ Step-by-Step Signup Form Code:**

**🔹 Signup.jsx**

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as Yup from "yup";

import axios from "axios";

import { useNavigate } from "react-router-dom";

function Signup() {

const navigate = useNavigate();

// ✅ Validation Schema

const validationSchema = Yup.object().shape({

name: Yup.string().required("Name is required"),

email: Yup.string().email("Invalid email").required("Email is required"),

password: Yup.string()

.min(6, "Minimum 6 characters")

.required("Password is required"),

});

// ✅ Handle submit

const handleSubmit = async (values, { resetForm }) => {

try {

const res = await axios.post("http://localhost:5000/api/auth/signup", values);

alert(res.data.message || "Signup successful");

resetForm();

navigate("/login");

} catch (err) {

alert(err.response?.data?.message || "Signup failed");

}

};

return (

<div style={{ maxWidth: "400px", margin: "auto" }}>

<h2>Signup</h2>

<Formik

initialValues={{ name: "", email: "", password: "" }}

validationSchema={validationSchema}

onSubmit={handleSubmit}

>

<Form>

<label>Name</label>

<Field type="text" name="name" />

<ErrorMessage name="name" component="div" style={{ color: "red" }} />

<label>Email</label>

<Field type="email" name="email" />

<ErrorMessage name="email" component="div" style={{ color: "red" }} />

<label>Password</label>

<Field type="password" name="password" />

<ErrorMessage name="password" component="div" style={{ color: "red" }} />

<br />

<button type="submit" style={{ marginTop: "10px" }}>

Signup

</button>

</Form>

</Formik>

</div>

);

}

export default Signup;

**Chalo ab banate hain: Login Form using JWT (React + Formik + Yup + Axios)**

**🎯 Login Form Ka Flow:**

1. User enters email & password
2. API request to backend (/api/auth/login)
3. Backend returns:  
   → token (JWT)  
   → user info
4. ✅ Token store (localStorage)
5. ✅ Navigate to dashboard / home

**✅ Step-by-Step Code – Login.jsx**

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as Yup from "yup";

import axios from "axios";

import { useNavigate } from "react-router-dom";

function Login() {

const navigate = useNavigate();

// ✅ Validation

const validationSchema = Yup.object({

email: Yup.string().email("Invalid email").required("Required"),

password: Yup.string().required("Required"),

});

// ✅ Submit

const handleSubmit = async (values, { resetForm }) => {

try {

const res = await axios.post("http://localhost:5000/api/auth/login", values);

const { token, user } = res.data;

// 🔐 Save token in localStorage

localStorage.setItem("token", token);

localStorage.setItem("user", JSON.stringify(user));

alert("Login successful");

resetForm();

navigate("/my-products"); // ya /dashboard

} catch (err) {

alert(err.response?.data?.message || "Login failed");

}

};

return (

<div style={{ maxWidth: "400px", margin: "auto" }}>

<h2>Login</h2>

<Formik

initialValues={{ email: "", password: "" }}

validationSchema={validationSchema}

onSubmit={handleSubmit}

>

<Form>

<label>Email</label>

<Field type="email" name="email" />

<ErrorMessage name="email" component="div" style={{ color: "red" }} />

<label>Password</label>

<Field type="password" name="password" />

<ErrorMessage name="password" component="div" style={{ color: "red" }} />

<br />

<button type="submit" style={{ marginTop: "10px" }}>

Login

</button>

</Form>

</Formik>

</div>

);

}

export default Login;

**Ab chalu karte hain Protected Routes using JWT (React side).**

**🔐 What is Protected Route?**

Protected Route = **Aisa page jo sirf login hone ke baad hi dikhai de**  
Jaise:

* /my-products
* /add-product
* /edit-product/:id

**✅ Step-by-Step Implementation**

**✅ 1. Create PrivateRoute.jsx**

import { Navigate } from "react-router-dom";

function PrivateRoute({ children }) {

const token = localStorage.getItem("token");

if (!token) {

return <Navigate to="/login" />;

}

return children;

}

export default PrivateRoute;

**✅ 2. Corrected App.jsx with PrivateRoute Integration:**

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

import Login from "./pages/Login";

import Signup from "./pages/Signup";

import Dashboard from "./pages/Dashboard";

import AddProduct from "./pages/AddProduct";

import MyProducts from "./pages/MyProducts";

import Orders from "./pages/Orders";

import Profile from "./pages/Profile";

import EditProduct from "./pages/EditProduct";

import SellerLayout from "./layout/SellerLayout";

import PrivateRoute from "./components/PrivateRoute"; // ✅

function App() {

return (

<BrowserRouter>

<Routes>

{/\* ✅ Public Routes \*/}

<Route path="/" element={<Login />} />

<Route path="/signup" element={<Signup />} />

{/\* ✅ Protected Routes inside SellerLayout \*/}

<Route

path="/"

element={

<PrivateRoute>

<SellerLayout />

</PrivateRoute>

}

>

<Route path="dashboard" element={<Dashboard />} />

<Route path="add-product" element={<AddProduct />} />

<Route path="my-products" element={<MyProducts />} />

<Route path="orders" element={<Orders />} />

<Route path="profile" element={<Profile />} />

<Route path="edit-product/:id" element={<EditProduct />} />

</Route>

</Routes>

</BrowserRouter>

);

}

export default App;

**🔒 Summary:**

| **Route** | **Access** | **Wrapper** |
| --- | --- | --- |
| / (login) | ✅ Public | Direct |
| /signup | ✅ Public | Direct |
| /dashboard | 🔒 Protected | via PrivateRoute + SellerLayout |
| /add-product etc. | 🔒 Protected | same as above |

**🧠 <PrivateRoute><SellerLayout /></PrivateRoute> ka matlab kya hai?**

Iska **matlab** hai:

👉 Jab user /dashboard, /add-product, /my-products jaata hai  
toh pehle hum check karte hain:

✅ Kya user login hai? (localStorage me token hai?)

**🔒 Agar login hai:**

✅ Toh SellerLayout show karo  
➡️ aur uske andar ke pages bhi show ho jaayenge (like Dashboard, MyProducts, etc.)

**❌ Agar login nahi hai:**

🚫 User ko **/login** pe redirect kar do

**🔁 Ye kaam PrivateRoute.jsx file karta hai:**

function PrivateRoute({ children }) {

const token = localStorage.getItem("token");

if (!token) {

return <Navigate to="/login" />;

}

return children; // 👈 SellerLayout yahan pass hota hai

}

**🔁 Yeh line:**

<PrivateRoute>

<SellerLayout />

</PrivateRoute>

**Matlab:**  
"Sirf login user ko SellerLayout + uske routes dikhana."

Ab mai tujhe batata hoon **kya-kya aur add kar sakte ho** sellerLayout me, taaki UI aur user experience aur improve ho jaaye 🔥

**🔧 Tera Current Layout:**

function SellerLayout() {

return (

<div style={{ display: "flex" }}>

<Sidebar />

<div style={{ flex: 1, padding: "20px" }}>

<Outlet />

</div>

</div>

);

}

**✅ Add Karne Layak Cheezein:**

| **Feature** | **Purpose** | **How to Add** |
| --- | --- | --- |
| 🔝 **Top Navbar** | Show username, logout button, title | Add above <Outlet /> |
| 💬 **Toast/Alert** | Show success/error messages | Add react-toastify or your own |
| 📱 **Responsive Sidebar** | Collapse on mobile, toggle button | Add sidebar toggle logic |
| 🎨 **Theme switcher** | Light/Dark toggle for seller UI | Add context/global state |
| 🚪 **Logout Button** | Auth control directly in layout | Inside topbar |
| 🧭 **Breadcrumb** | Show current page path | Optional UI above <Outlet /> |
| ⏳ **Loader** | Show spinner when navigating | React Router useNavigation or custom logic |

**🔁 Example Improved SellerLayout.jsx (with Topbar + Logout):**

import Sidebar from "../components/Sidebar";

import { Outlet, useNavigate } from "react-router-dom";

function SellerLayout() {

const navigate = useNavigate();

const handleLogout = () => {

localStorage.removeItem("token");

localStorage.removeItem("user");

navigate("/login");

};

return (

<div style={{ display: "flex", minHeight: "100vh" }}>

<Sidebar />

<div style={{ flex: 1, padding: "20px" }}>

{/\* 🔝 Topbar \*/}

<div

style={{

display: "flex",

justifyContent: "space-between",

alignItems: "center",

marginBottom: "20px",

borderBottom: "1px solid #ddd",

paddingBottom: "10px",

}}

>

<h3>Seller Panel</h3>

<button onClick={handleLogout} style={{ padding: "5px 10px" }}>

Logout

</button>

</div>

{/\* Page Content \*/}

<Outlet />

</div>

</div>

);

}

export default SellerLayout;

**✅ Output:**

* Sidebar left side
* Top navbar with "Seller Panel" & Logout button
* All page content shown using <Outlet />

Bhai **mitr**, ab banate hain ek simple aur clean **Dashboard.jsx** page seller panel ke liye 💼  
Ye dashboard show karega kuch **important stats** jaise:

* 👕 Total Products
* 📦 Total Orders
* 💸 Total Sales (dummy for now)
* 👤 Seller Name (optional)

**✅ 1. Dummy Dashboard.jsx Code (basic working UI)**

import { useEffect, useState } from "react";

import axios from "axios";

function Dashboard() {

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

const [orders, setOrders] = useState([]); // use later if orders implemented

useEffect(() => {

axios

.get("http://localhost:5000/api/products")

.then((res) => setProducts(res.data))

.catch((err) => console.log("Error loading products", err));

}, []);

return (

<div>

<h2>📊 Dashboard</h2>

<div

style={{

display: "flex",

gap: "20px",

flexWrap: "wrap",

marginTop: "20px",

}}

>

<div style={cardStyle}>

<h3>Total Products</h3>

<p style={numberStyle}>{products.length}</p>

</div>

<div style={cardStyle}>

<h3>Total Orders</h3>

<p style={numberStyle}>{orders.length}</p> {/\* optional \*/}

</div>

<div style={cardStyle}>

<h3>Total Sales</h3>

<p style={numberStyle}>₹0</p> {/\* dummy for now \*/}

</div>

</div>

</div>

);

}

const cardStyle = {

width: "200px",

background: "#fff",

padding: "20px",

border: "1px solid #ddd",

borderRadius: "8px",

boxShadow: "0 2px 8px rgba(0,0,0,0.1)",

};

const numberStyle = {

fontSize: "24px",

fontWeight: "bold",

marginTop: "10px",

};

export default Dashboard;

**🔥 Bonus Idea (Later):**

| Feature | Add Later When Ready |
| --- | --- |
| 🎯 Graph chart | Use Recharts or Chart.js |
| 📦 Real Orders Count | Add order model/backend |
| 💰 Earnings calculation | Use total sales formula |
| 🕐 Last login info | From user/token data |

**Ab chalte hain Profile Page par — jahan seller apna naam, email, aur "Logout" button dekh sakta hai.**

**🧱 Profile Page Basic Idea:**

* 👤 Seller Info (dummy or from token/localStorage)
* ✏️ Future option: edit profile
* 🚪 Logout button

**✅ Step-by-Step Profile.jsx**

import { useNavigate } from "react-router-dom";

import { useEffect, useState } from "react";

function Profile() {

const navigate = useNavigate();

const [user, setUser] = useState({});

useEffect(() => {

// Dummy user info from localStorage

const storedUser = localStorage.getItem("user");

if (storedUser) {

setUser(JSON.parse(storedUser));

}

}, []);

const handleLogout = () => {

localStorage.removeItem("token");

localStorage.removeItem("user");

navigate("/login");

};

return (

<div style={containerStyle}>

<h2>👤 Seller Profile</h2>

<div style={cardStyle}>

<p><strong>Name:</strong> {user.name || "Demo Seller"}</p>

<p><strong>Email:</strong> {user.email || "demo@example.com"}</p>

<button onClick={handleLogout} style={buttonStyle}>

🚪 Logout

</button>

</div>

</div>

);

}

const containerStyle = { padding: "20px" };

const cardStyle = {

padding: "20px",

border: "1px solid #ddd",

borderRadius: "8px",

width: "300px",

marginTop: "20px",

background: "#fff",

boxShadow: "0 2px 8px rgba(0,0,0,0.1)",

};

const buttonStyle = {

marginTop: "20px",

background: "#ff4d4f",

color: "white",

border: "none",

padding: "10px 15px",

cursor: "pointer",

borderRadius: "5px",

};

export default Profile;

**✅ Optional:**

* 🔒 Get user data from token (JWT decode)
* 📝 Add Edit Profile form
* 🖼️ Add profile picture

**Ab main tujhe step-by-step samjhaata hoon ki JWT ko decode kaise karte hain React me, taaki user info token se directly mil jaaye — bina backend call ke.**

**🧠 JWT Token kya hota hai?**

JWT (JSON Web Token) = 3 parts ka hota hai:

eyJhbGciOi... <-- Header

eyJ1c2VySWQiOiIxMjM... <-- Payload (👇 yahi part humein chahiye)

SflKxwRJSMe... <-- Signature

**✅ Step-by-Step JWT Decode in React**

**🔹 1. 📦 Install jwt-decode package:**

npm install jwt-decode

**🔹 2. Update Profile.jsx:**

import jwt\_decode from "jwt-decode";

import { useEffect, useState } from "react";

import { useNavigate } from "react-router-dom";

function Profile() {

const [user, setUser] = useState({});

const navigate = useNavigate();

useEffect(() => {

const token = localStorage.getItem("token");

if (token) {

try {

const decoded = jwt\_decode(token);

setUser(decoded.user); // 👈 if you saved user inside token as `user`

} catch (err) {

console.error("Invalid token", err);

}

}

}, []);

const handleLogout = () => {

localStorage.removeItem("token");

navigate("/login");

};

return (

<div>

<h2>👤 Profile</h2>

<div style={{ padding: "20px", border: "1px solid #ccc", width: "300px" }}>

<p><strong>Name:</strong> {user.name || "N/A"}</p>

<p><strong>Email:</strong> {user.email || "N/A"}</p>

<p><strong>ID:</strong> {user.\_id || "N/A"}</p>

<button onClick={handleLogout}>Logout</button>

</div>

</div>

);

}

export default Profile;

**✅ Example JWT Payload (backend me aisa bana):**

// Node.js backend

const token = jwt.sign(

{ user: { \_id: user.\_id, name: user.name, email: user.email } },

"SECRET\_KEY",

{ expiresIn: "1d" }

);

**🧠 Summary:**

| **Step** | **Result** |
| --- | --- |
| 🔐 Store token | In localStorage after login |
| 📖 Decode token | Using jwt-decode |
| 👤 Access user info | Without hitting backend |
|  |  |

**🔧 Backend → authRoute.js (ya jahan login POST route hai)**

🔹 1. Pehle jsonwebtoken install karo:

🔹 2. Backend login route me apply karo:

const express = require("express");

const router = express.Router();

const jwt = require("jsonwebtoken");

const User = require("../models/User"); // your user model

const SECRET\_KEY = "SECRET\_KEY"; // strong secret env me rakhna

router.post("/login", async (req, res) => {

const { email, password } = req.body;

try {

const user = await User.findOne({ email });

if (!user || user.password !== password) {

return res.status(401).json({ message: "Invalid credentials" });

}

// ✅ JWT Token create yahan

const token = jwt.sign(

{

user: {

\_id: user.\_id,

name: user.name,

email: user.email,

},

},

SECRET\_KEY,

{ expiresIn: "1d" }

);

res.json({ token }); // 👈 send to frontend

} catch (err) {

res.status(500).json({ message: "Login failed", error: err.message });

}

});

module.exports = router;

**🔑 Frontend me kya kare:**

* Login hone ke baad token ko localStorage me save karo:
  + localStorage.setItem("token", res.data.token);
* Profile page ya protected route me jwt-decode se info extract karo (jaise pehle bataya).

**📦 Summary:**

| **Kya** | **Kahaan** | **Kaam** |
| --- | --- | --- |
| jwt.sign(...) | Backend Login route | Token generate |
| res.json({ token }) | Frontend bhejna |  |
| localStorage.setItem("token", token) | React me store karna |  |
| jwt-decode(token) | User info lena |  |

**✅ 1. Quick Fix → Optional Chaining Use Kar:**

**🔧 Change this:**

<p><strong>Name:</strong> {user.name}</p>

**✅ To this:**

<p><strong>Name:</strong> {user?.name || "N/A"}</p>

<p><strong>Email:</strong> {user?.email || "N/A"}</p>

**✅ 2. OR: Conditional Rendering**

{user ? (

<div>

<p><strong>Name:</strong> {user.name}</p>

<p><strong>Email:</strong> {user.email}</p>

</div>

) : (

<p>Loading...</p>

)}

**✅ 3. Make sure user exist in decoded token**

const decoded = jwtDecode(token);

console.log("Decoded:", decoded);

setUser(decoded.user); // 👈 check if "user" exist here

If your token payload looks like this:

{

"name": "prakash",

"email": "prakash@example.com",

"iat": ...,

"exp": ...

}

**Then use:**

setUser(decoded); // not decoded.user

**✅ Final Tip:**

Make sure your JWT was created like this (in backend):

const token = jwt.sign(

{ name: user.name, email: user.email }, // 👈 no "user" key

SECRET\_KEY,

{ expiresIn: "1d" }

);

✅ Final Profile.jsx Example:

import { jwtDecode } from "jwt-decode";

import { useEffect, useState } from "react";

import { useNavigate } from "react-router-dom";

function Profile() {

const [user, setUser] = useState(null);

const navigate = useNavigate();

useEffect(() => {

const token = localStorage.getItem("token");

if (token) {

try {

const decoded = jwtDecode(token);

setUser(decoded); // or decoded.user based on token structure

} catch (err) {

console.error("Invalid token", err);

}

}

}, []);

const handleLogout = () => {

localStorage.removeItem("token");

navigate("/login");

};

if (!user) return <p>Loading...</p>;

return (

<div>

<h2>👤 Profile</h2>

<p><strong>Name:</strong> {user.name || "N/A"}</p>

<p><strong>Email:</strong> {user.email || "N/A"}</p>

<button onClick={handleLogout}>Logout</button>

</div>

);

}

export default Profile;

🔹 **Option 1**: Image validation manually inside handleSubmit

const handleSubmit = async (values, { resetForm }) => {

if (!image) {

alert("❌ Please upload an image.");

return;

}

try {

const formData = new FormData();

formData.append("name", values.name);

formData.append("price", values.price);

formData.append("description", values.description);

formData.append("image", image);

await axios.post("http://localhost:5000/api/products/add", formData, {

headers: { "Content-Type": "multipart/form-data" },

});

alert("✅ Product added successfully");

resetForm();

setImage(null);

setPreview(null);

} catch (err) {

console.error("Error:", err);

alert("❌ Failed to add product");

}

};

🔹 validationSchema:

const validationSchema = Yup.object({

name: Yup.string().required("Name is required"),

price: Yup.number()

.typeError("Price must be a number")

.positive("Price must be positive")

.required("Price is required"),

description: Yup.string().required("Description is required"),

});

🔹 handleSubmit:

const handleSubmit = async (values, { resetForm }) => {

if (!image) {

alert("❌ Please upload an image.");

return;

}

try {

const formData = new FormData();

formData.append("name", values.name);

formData.append("price", values.price);

formData.append("description", values.description);

formData.append("image", image);

await axios.post("http://localhost:5000/api/products/add", formData, {

headers: { "Content-Type": "multipart/form-data" },

});

alert("✅ Product added successfully");

resetForm();

setImage(null);

setPreview(null);

} catch (err) {

console.error("Error:", err);

alert("❌ Failed to add product");

}

};

**🔚 Final Tip:**

File input ka validation hamesha manually handle karo jab Formik use kar rahe ho — kyunki Formik usko track nahi karta.

**✅ Goal:**

* Starting me sirf **100 characters** dikhaye.
* Niche ek "More" button ho.
* Button click pe **poora description** dikhe + "Less" button aa jaye.

**🧠 Example Code Snippet:**

Yeh code tu MyProducts.jsx ya jahan product card hai wahan laga:

import { useState } from "react";

function ProductCard({ product }) {

const [showFull, setShowFull] = useState(false);

const toggleText = () => setShowFull(!showFull);

const shortDesc =

product.description.length > 100

? product.description.slice(0, 100) + "..."

: product.description;

return (

<div

style={{

border: "1px solid #ccc",

padding: "10px",

width: "200px",

borderRadius: "10px",

}}

>

<img

src={`http://localhost:5000/${product.imageUrl}`}

alt={product.name}

style={{ width: "100%", height: "150px", objectFit: "cover" }}

/>

<h4>{product.name}</h4>

<p>₹{product.price}</p>

<p style={{ fontSize: "14px" }}>

{showFull ? product.description : shortDesc}

<br />

{product.description.length > 100 && (

<button

onClick={toggleText}

style={{

border: "none",

background: "none",

color: "blue",

cursor: "pointer",

padding: 0,

}}

>

{showFull ? "Less ▲" : "More ▼"}

</button>

)}

</p>

{/\* Buttons \*/}

<button style={{ background: "red", color: "white" }}>Delete</button>

<button style={{ background: "orange", color: "white" }}>Edit</button>

</div>

);

}

export default ProductCard;

🔹 1. useState se toggle state banate hain

const [showFull, setShowFull] = useState(false);

🔹 2. Button click pe state toggle karne ke liye function:

const toggleText = () => setShowFull(!showFull);

Jab user "More" ya "Less" pe click kare, tab showFull ka state change ho jaye.

🔹 3. Description ko short ya full dikhane ka logic:

const shortDesc =

product.description.length > 100

? product.description.slice(0, 100) + "..."

: product.description;

 Agar description 100 characters se lamba hai, toh uska **first 100 chars + ...** dikhate hain.

 Warna full description hi dikhate hain.

🔹 4. JSX me conditional rendering:

<p style={{ fontSize: "14px" }}>

{showFull ? product.description : shortDesc}

<br />

{product.description.length > 100 && (

<button

onClick={toggleText}

style={{

border: "none",

background: "none",

color: "blue",

cursor: "pointer",

padding: 0,

}}

>

{showFull ? "Less ▲" : "More ▼"}

</button>

)}

</p>

 showFull true ho toh full description.

 Warna short description + "More ▼".

 Button sirf tab dikhate hain jab text 100 se lamba ho.

**🧩 Final Code Block (Component Form)**

import { useState } from "react";

function ProductCard({ product }) {

const [showFull, setShowFull] = useState(false);

const toggleText = () => setShowFull(!showFull);

const shortDesc =

product.description.length > 100

? product.description.slice(0, 100) + "..."

: product.description;

return (

<div

style={{

border: "1px solid #ccc",

padding: "10px",

width: "200px",

borderRadius: "10px",

}}

>

<img

src={`http://localhost:5000/${product.imageUrl}`}

alt={product.name}

style={{ width: "100%", height: "150px", objectFit: "cover" }}

/>

<h4>{product.name}</h4>

<p>₹{product.price}</p>

<p style={{ fontSize: "14px" }}>

{showFull ? product.description : shortDesc}

<br />

{product.description.length > 100 && (

<button

onClick={toggleText}

style={{

border: "none",

background: "none",

color: "blue",

cursor: "pointer",

padding: 0,

fontSize: "13px",

}}

>

{showFull ? "Less ▲" : "More ▼"}

</button>

)}

</p>

<button style={{ background: "red", color: "white" }}>Delete</button>

<button style={{ background: "orange", color: "white" }}>Edit</button>

</div>

);

}

export default ProductCard;

**✅ Final Output:**

| **State** | **Description** | **Button** |
| --- | --- | --- |
| showFull = false | First 100 characters + ... | "More ▼" |
| showFull = true | Full description | "Less ▲" |

**showFull ko har product ke liye local banaao.**

**✅ Option 1: Move each product card to a separate ProductCard component.**

**✅ 1. Replace this line:**

const [showFull, setShowFull] = useState(false);

**with this:**

const [expandedDesc, setExpandedDesc] = useState({});

**✅ 2. Update toggleText function:**

const toggleText = (id) => {

setExpandedDesc((prev) => ({

...prev,

[id]: !prev[id],

}));

};

**✅ 3. Inside your .map() loop, replace:**

{showFull ? product.description : shortDesc}

**with:**

js

CopyEdit

{expandedDesc[product.\_id]

? product.description

: product.description.length > 100

? product.description.slice(0, 100) + "..."

: product.description}

**✅ 4. Replace onClick={toggleText} with:**

<button

onClick={() => toggleText(product.\_id)}

style={{

border: "none",

background: "none",

color: "blue",

cursor: "pointer",

padding: 0,

}}

>

{expandedDesc[product.\_id] ? "Less ▲" : "More ▼"}

</button>

**🔚 Final Result:**

* Har product ki description **alag se toggle** hogi.
* UI bilkul user-friendly lagega.
* Search/filter functionality already working ✅

**📦 2. Order Management Module**

👉 Ek fake "Orders" list dikhao seller ko:

* Order ID
* Product
* Customer name
* Status: Pending / Shipped
* Button: "Mark as Shipped"

✅ Bas dummy data se start karein (backend later)

Ab hum **Order Management Module** start karte hain — seller ke liye ek "Orders" page jisme wo apne product ke orders dekh sake.

**✅ Step 1: Basic Dummy Order Data**

Hum pehle fake order data se page banayenge (API baad me banayenge).

**✅ Step 2: UI Design**

Har order card me dikhe:

* Product Image & Name
* Customer Name
* Quantity
* Status (Pending / Shipped)
* Button: Mark as Shipped

**✅ Step 3: Status Change Logic**

Click karne pe "Pending" → "Shipped" ho jaye (UI level pe)

**🛠️ Step 1: Create File pages/Orders.jsx**

import { useState } from "react";

function Orders() {

// Step 1: Dummy Order Data

const [orders, setOrders] = useState([

{

\_id: "1",

productName: "Apple iPhone 15",

imageUrl: "uploads/iphone.jpg",

customer: "Ravi Sharma",

quantity: 1,

status: "Pending",

},

{

\_id: "2",

productName: "Boat Headphones",

imageUrl: "uploads/headphone.jpg",

customer: "Anjali Verma",

quantity: 2,

status: "Shipped",

},

]);

// Step 2: Status Update Handler

const markShipped = (id) => {

setOrders((prev) =>

prev.map((order) =>

order.\_id === id ? { ...order, status: "Shipped" } : order

)

);

};

return (

<div style={{ padding: "20px" }}>

<h2>🛒 Orders</h2>

{orders.length === 0 ? (

<p>No orders yet.</p>

) : (

<div style={{ display: "flex", flexDirection: "column", gap: "20px" }}>

{orders.map((order) => (

<div

key={order.\_id}

style={{

border: "1px solid #ccc",

borderRadius: "8px",

padding: "15px",

display: "flex",

gap: "20px",

alignItems: "center",

}}

>

<img

src={`http://localhost:5000/${order.imageUrl}`}

alt={order.productName}

width="100"

style={{ borderRadius: "5px" }}

/>

<div style={{ flex: 1 }}>

<h4>{order.productName}</h4>

<p>

👤 <strong>Customer:</strong> {order.customer}

</p>

<p>

🔢 <strong>Qty:</strong> {order.quantity}

</p>

<p>

📦 <strong>Status:</strong>{" "}

<span

style={{

color: order.status === "Pending" ? "orange" : "green",

fontWeight: "bold",

}}

>

{order.status}

</span>

</p>

</div>

{order.status === "Pending" && (

<button

onClick={() => markShipped(order.\_id)}

style={{

background: "green",

color: "white",

border: "none",

padding: "10px",

borderRadius: "5px",

cursor: "pointer",

}}

>

Mark as Shipped

</button>

)}

</div>

))}

</div>

)}

</div>

);

}

export default Orders;

**📌 Is code me:**

* orders: dummy array state me hai
* markShipped(id): status update karta hai
* UI clean aur mobile-friendly banega
* imageUrl ke liye uploads/ folder ka image path use ho raha hai

**🔮 Jab Tu Buyer-Frontend Banayega Tab:**

Tabhi **Order Create (POST)** karna backend se start hoga.

**🛒 Buyer Panel me kya hoga:**

* Buyer koi product kharidega
* POST /api/orders request jayegi backend pe
* Order save hoga DB me (MongoDB)
* Seller-panel ka GET /api/orders usi DB se orders fetch karega

**🧾 Toh abhi kya kare?**

Abhi seller-panel me hum **dummy data** se UI bana rahe hain  
Jab tu buyer-panel banayega tab:

* ✅ Order schema banega
* ✅ POST /orders route banega
* ✅ Seller-panel me GET /orders se live data aayega

**🔚 Conclusion:**

Tu seller-panel ka order UI ready kar raha hai  
Baad me jab buyer frontend banayega, tab real order backend se connect karenge

**📊 Dashboard Summary Page – Overview Cards**

**🎯 Target:**

Seller jab login kare to Dashboard.jsx me dikhe:

| **Card** | **Detail** |
| --- | --- |
| 📦 Total Products | MongoDB se count |
| 🛒 Total Orders | (abhi dummy se) |
| 💰 Total Revenue | (orders × price — dummy logic) |

**✅ Step 1: Create File – Dashboard.jsx**

✅ Let’s Improve It:

**1. Add dummy orders array temporarily (until real backend added)**

const [orders, setOrders] = useState([

{ quantity: 1, price: 1000 },

{ quantity: 2, price: 500 },

]);

**2. Calculate Total Revenue:**

const totalSales = orders.reduce(

(total, order) => total + order.quantity \* order.price,

0

);

**3. Final updated Dashboard code:**

import { useEffect, useState } from "react";

import axios from "axios";

function Dashboard() {

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

const [orders, setOrders] = useState([

{ quantity: 1, price: 1000 },

{ quantity: 2, price: 500 },

]);

const totalSales = orders.reduce(

(total, order) => total + order.quantity \* order.price,

0

);

useEffect(() => {

axios

.get("http://localhost:5000/api/products")

.then((res) => setProducts(res.data))

.catch((err) => console.log("Error loading products", err));

}, []);

return (

<div>

<h2>📊 Dashboard</h2>

<div

style={{

display: "flex",

gap: "20px",

flexWrap: "wrap",

marginTop: "20px",

}}

>

<div style={cardStyle}>

<h3>Total Products</h3>

<p style={numberStyle}>{products.length}</p>

</div>

<div style={cardStyle}>

<h3>Total Orders</h3>

<p style={numberStyle}>{orders.length}</p>

</div>

<div style={cardStyle}>

<h3>Total Sales</h3>

<p style={numberStyle}>₹{totalSales}</p>

</div>

</div>

</div>

);

}

const cardStyle = {

width: "200px",

background: "#fff",

padding: "20px",

border: "1px solid #ddd",

borderRadius: "8px",

boxShadow: "0 2px 8px rgba(0,0,0,0.1)",

};

const numberStyle = {

fontSize: "24px",

fontWeight: "bold",

marginTop: "10px",

};

export default Dashboard;

**📦 Future Upgrade Plan:**

* Replace dummy orders array with:  
  axios.get("http://localhost:5000/api/orders")

reduce() JavaScript ka **built-in array method** hai jo array ke sare items ko **ek hi value** me **reduce** kar deta hai.

🔁 **reduce() Syntax:**

array.reduce((accumulator, currentValue) => {

return newValue;

}, initialValue);

**📖 Explanation:**

| **Term** | **Meaning** |
| --- | --- |
| accumulator | pehle ka result |
| currentValue | abhi ka element |
| initialValue | starting value |

**🔧 Example 1: Total of numbers**

const nums = [10, 20, 30];

const total = nums.reduce((acc, curr) => acc + curr, 0);

console.log(total); // 👉 60

**Breakdown:**

| **Step** | **acc** | **curr** | **acc + curr** |
| --- | --- | --- | --- |
| 1 | 0 | 10 | 10 |
| 2 | 10 | 20 | 30 |
| 3 | 30 | 30 | 60 |

**🔧 Example 2: Total Sales Calculation**

const orders = [

{ quantity: 2, price: 100 },

{ quantity: 1, price: 200 },

];

const totalSales = orders.reduce(

(acc, order) => acc + order.quantity \* order.price,

0

);

console.log(totalSales); // 👉 400

**Kaise kaam karta hai:**

* 1st item: 0 + 2×100 → 200
* 2nd item: 200 + 1×200 → 400

**📌 Use reduce() when:**

* Total banana hai (sum, average, etc.)
* Single object/value me array ka result chahiye
* Custom calculation karni ho har item ke sath

**2️⃣ Object Grouping (group by category, etc.)**

const items = [

{ name: "Apple", category: "Fruit" },

{ name: "Carrot", category: "Vegetable" },

{ name: "Banana", category: "Fruit" },

];

const grouped = items.reduce((acc, item) => {

if (!acc[item.category]) {

acc[item.category] = [];

}

acc[item.category].push(item.name);

return acc;

}, {});

console.log(grouped);

/\*

{

Fruit: ["Apple", "Banana"],

Vegetable: ["Carrot"]

}

\*/

**📦 Order Management Module (continue karna chahoge?)**

* Add New Order
* View All Orders
* Filter Orders (by status, date, etc.)
* Update Order Status (Pending → Shipped → Delivered)

**🧾 Order Management Flow (Seller Panel Side):**

**🔹 Step 1: Backend Setup**

* Create Mongoose Order Model (models/Order.js)
* Add Order Routes:
  + POST /api/orders/create ➤ Create new order
  + GET /api/orders ➤ Get all orders (for seller)
  + PUT /api/orders/:id ➤ Update status (optional)

**🔹 Step 2: Frontend - Orders.jsx**

* Fetch all orders
* Display them in table/cards
* Show: Product name, price, quantity, buyer name, status
* Optional: Change order status from dropdown (e.g. Pending → Shipped)

**🔹 Step 3: Connect Buyer Frontend Later**

* Right now hum dummy order data se kaam chalayenge (ya manually postman se create karenge)

**💡 First Step Now:**

➡️ models/Order.js banana chahoge?

**➡️Chalo ab hum models/Order.js file banate hain.  
Ye schema har order ke important details ko store karega.**

**🧾 models/Order.js (Mongoose Model)**

const mongoose = require("mongoose");

const orderSchema = new mongoose.Schema(

{

product: {

type: mongoose.Schema.Types.ObjectId,

ref: "Product",

required: true,

},

buyerName: {

type: String,

required: true,

},

buyerEmail: {

type: String,

required: true,

},

quantity: {

type: Number,

required: true,

default: 1,

},

totalPrice: {

type: Number,

required: true,

},

status: {

type: String,

enum: ["Pending", "Shipped", "Delivered"],

default: "Pending",

},

orderedAt: {

type: Date,

default: Date.now,

},

},

{ timestamps: true }

);

module.exports = mongoose.model("Order", orderSchema);

**✅ Explanation:**

| **Field** | **Description** |
| --- | --- |
| product | Reference to product ID (Product model) |
| buyerName | Buyer's name |
| buyerEmail | Email of the buyer |
| quantity | How many items bought |
| totalPrice | quantity × product price |
| status | Pending / Shipped / Delivered |
| orderedAt | Date when order was placed |

Ab hum banayenge:

**📦 routes/orderRoute.js**

✅ Create New Order  
✅ Get All Orders

**🔹 Step 1: Create file routes/orderRoute.js**

const express = require("express");

const router = express.Router();

const Order = require("../models/Order");

// 🔸 Create a new order (POST /api/orders/create)

router.post("/create", async (req, res) => {

try {

const { product, buyerName, buyerEmail, quantity, totalPrice } = req.body;

const newOrder = new Order({

product,

buyerName,

buyerEmail,

quantity,

totalPrice,

});

await newOrder.save();

res.status(201).json({ message: "Order placed", order: newOrder });

} catch (err) {

res.status(500).json({ message: "Failed to create order", error: err });

}

});

// 🔸 Get all orders (GET /api/orders)

router.get("/", async (req, res) => {

try {

const orders = await Order.find()

.populate("product", "name price")

.sort({ createdAt: -1 });

res.json(orders);

} catch (err) {

res.status(500).json({ message: "Failed to fetch orders", error: err });

}

});

module.exports = router;

**🔹 Step 2: Add in server.js**

const orderRoutes = require("./routes/orderRoute");

app.use("/api/orders", orderRoutes);

**✅ Test Now**

* 🧪 Postman se POST /api/orders/create karo order create ke liye
* 🧪 GET /api/orders se sare orders fetch karo

**Ready for Frontend**

Ab hum frontend me Orders.jsx banate hain — jisme seller apne sare orders dekh sakta hai.

✅ Orders.jsx (React Frontend)

import { useEffect, useState } from "react";

import axios from "axios";

function Orders() {

const [orders, setOrders] = useState([]);

// ✅ Step 1: Fetch orders from backend

useEffect(() => {

axios

.get("http://localhost:5000/api/orders")

.then((res) => setOrders(res.data))

.catch((err) => console.error("Failed to fetch orders:", err));

}, []);

// ✅ Step 2: Mark as shipped (update status in DB)

const markShipped = async (id) => {

try {

const res = await axios.put(`http://localhost:5000/api/orders/${id}`, {

status: "Shipped",

});

setOrders((prev) =>

prev.map((order) =>

order.\_id === id ? { ...order, status: res.data.status } : order

)

);

} catch (err) {

console.error("Status update failed", err);

alert("❌ Failed to update status");

}

};

return (

<div style={{ padding: "20px" }}>

<h2>📦 Orders</h2>

{orders.length === 0 ? (

<p>No orders found.</p>

) : (

<div style={{ display: "flex", flexDirection: "column", gap: "20px" }}>

{orders.map((order) => (

<div

key={order.\_id}

style={{

border: "1px solid #ccc",

borderRadius: "8px",

padding: "15px",

display: "flex",

gap: "20px",

alignItems: "center",

}}

>

<img

src={`http://localhost:5000/${order.product?.imageUrl}`}

alt={order.product?.name}

width="100"

style={{ borderRadius: "5px" }}

/>

<div style={{ flex: 1 }}>

<h4>{order.product?.name}</h4>

<p>👤 <strong>Customer:</strong> {order.buyerName}</p>

<p>🔢 <strong>Qty:</strong> {order.quantity}</p>

<p>

📦 <strong>Status:</strong>{" "}

<span

style={{

color: order.status === "Pending" ? "orange" : "green",

fontWeight: "bold",

}}

>

{order.status}

</span>

</p>

</div>

{order.status === "Pending" && (

<button

onClick={() => markShipped(order.\_id)}

style={{

background: "green",

color: "white",

border: "none",

padding: "10px",

borderRadius: "5px",

cursor: "pointer",

}}

>

Mark as Shipped

</button>

)}

</div>

))}

</div>

)}

</div>

);

}

export default Orders;

**🔍 Features:**

* ✅ Lists all orders
* ✅ Shows product name, quantity, total, buyer info
* ✅ Order date and status shown

**✅ Backend Required Routes:**

Make sure you already have these in your backend (orderRoutes.js):

// GET all orders

router.get("/", async (req, res) => {

const orders = await Order.find().populate("product"); // for product name/image

res.json(orders);

});

// UPDATE status

router.put("/:id", async (req, res) => {

const order = await Order.findByIdAndUpdate(

req.params.id,

{ status: req.body.status },

{ new: true }

);

res.json(order);

});

**🔚 Result:**

* Real orders fetched from MongoDB
* Seller can update status from **Pending → Shipped**
* No need for dummy orders now

**✅ Next Steps:**

We'll build the following backend functionality step-by-step:

**🔹 1. Add Order Route (POST)**

When a buyer places an order (you'll connect this later in the buyer-frontend), it will send:

{

"productId": "123",

"customer": "Ravi Sharma",

"quantity": 2

}

**🛠 Sample Route:**

// POST /api/orders

router.post("/", async (req, res) => {

try {

const { productId, customer, quantity } = req.body;

// Get product details from DB

const product = await Product.findById(productId);

if (!product) return res.status(404).json({ message: "Product not found" });

const order = new Order({

productId,

productName: product.name,

imageUrl: product.imageUrl,

customer,

quantity,

status: "Pending",

});

await order.save();

res.status(201).json({ message: "Order placed", order });

} catch (err) {

res.status(500).json({ message: "Error placing order", error: err.message });

}

});

**🔹 2. Get All Orders (GET)**

// GET /api/orders

router.get("/", async (req, res) => {

try {

const orders = await Order.find().sort({ \_id: -1 });

res.json(orders);

} catch (err) {

res.status(500).json({ message: "Error fetching orders" });

}

});

**🔹 3. Update Order Status (PUT)**

// PUT /api/orders/:id/status

router.put("/:id/status", async (req, res) => {

try {

const { status } = req.body;

const updated = await Order.findByIdAndUpdate(

req.params.id,

{ status },

{ new: true }

);

res.json({ message: "Order updated", order: updated });

} catch (err) {

res.status(500).json({ message: "Error updating order", error: err.message });

}

});

**🛒 Model: Order.js (in models folder)**

const mongoose = require("mongoose");

const orderSchema = new mongoose.Schema({

productId: { type: mongoose.Schema.Types.ObjectId, ref: "Product" },

productName: String,

imageUrl: String,

customer: String,

quantity: Number,

status: { type: String, default: "Pending" },

}, { timestamps: true });

module.exports = mongoose.model("Order", orderSchema);

**✅ Once you're done:**

* Test all APIs on Postman.
* Then we’ll update your Orders.jsx to use real data.

**✅ Step-by-Step in Postman:**

**🔹 1. Choose PUT Method**

**🔹 2. Enter the URL**

http://localhost:5000/api/orders/ORDER\_ID/status

👉 Replace ORDER\_ID with the actual \_id of the order you want to update. Example:

http://localhost:5000/api/orders/64fb1f88a823b73b7dd28d45/status

**🔹 3. Go to Body tab → Select raw → Choose JSON**

Paste the status update JSON:

{

"status": "Shipped"

}

// You can change status to "Delivered" or any other value you use.

**🔹 4. Click Send**

If everything works fine, you’ll get a success response like:

{

"message": "Order updated",

"order": {

"\_id": "64fb1f88a823b73b7dd28d45",

"productName": "iPhone 15",

"status": "Shipped",

...

}

}

**⚠️ Common Issues:**

| **Problem** | **Solution** |
| --- | --- |
| 404 Not Found | Check if the order ID is correct |
| 500 Internal Server Error | Make sure your backend PUT route is implemented correctly and server restarted |
| Empty body response | Ensure you're sending Content-Type: application/json |

ab hum **Order Management Module** ka agla step continue karte hain:

**📦 Step-by-Step: Connect Orders API with Real Data (Frontend + Backend)**

**✅ Goal:**

* Seller Orders.jsx page pe real order data show ho.
* Seller "Mark as Shipped" kare to database me status update ho.

**🧱 Step 1: Backend - GET /api/orders Route**

Aapka routes/orderRoutes.js me ye hona chahiye:

const express = require("express");

const router = express.Router();

const Order = require("../models/Order");

// GET all orders (for seller)

router.get("/", async (req, res) => {

try {

const orders = await Order.find().sort({ createdAt: -1 });

res.json(orders);

} catch (err) {

res.status(500).json({ message: "Error fetching orders" });

}

});

**🧱 Step 2: Frontend - Load Real Orders in Orders.jsx**

Update useEffect like this:

useEffect(() => {

axios

.get("http://localhost:5000/api/orders")

.then((res) => setOrders(res.data))

.catch((err) => console.error("Error loading orders", err));

}, []);

**🧱 Step 3: Handle "Mark as Shipped"**

Replace markShipped() function:

const markShipped = async (id) => {

try {

const res = await axios.put(`http://localhost:5000/api/orders/${id}/status`, {

status: "Shipped",

});

const updated = res.data.order;

setOrders((prev) =>

prev.map((order) => (order.\_id === updated.\_id ? updated : order))

);

} catch (err) {

console.error("Failed to update status", err);

alert("Failed to mark as shipped");

}

};

**✅ Sample Final Preview on Page**

* Seller will see real orders from DB.
* Seller can click “Mark as Shipped”.
* Status changes to green ✅.

For fine : dono routes **functionally same**