MERN Stack: Building a Full-Stack E-Commerce App

Objective

This hands-on lab will guide you through building a mini e-commerce application using the MERN stack, which includes MongoDB, Express, React, and Node.js. The main objective is to develop a full-stack application where you create a backend RESTful API to manage product data and store it securely in MongoDB. On the frontend, you will build a React interface that lists all products and allows users to add new products through a form.

Throughout this project, you will gain practical experience setting up a Node.js server with Express, defining data models with Mongoose, and connecting the backend with a responsive React frontend using Axios for API communication. By the end of this lab, you will understand the core architecture and workflow of a MERN stack application, preparing you for more advanced full-stack JavaScript development.

Backend Setup (Node.js + Express + MongoDB)

In this section, we will set up the backend server that provides a RESTful API to manage product data for our mini e-commerce application. We will use **Node.js** with **Express** as the web framework, connect to a **MongoDB** database via **Mongoose**, and handle essential middleware including **CORS** and environment variables with **dotenv**. Let's dive into the step-by-step process.

1. Open the Integrated Terminal

1. Launch VS Code

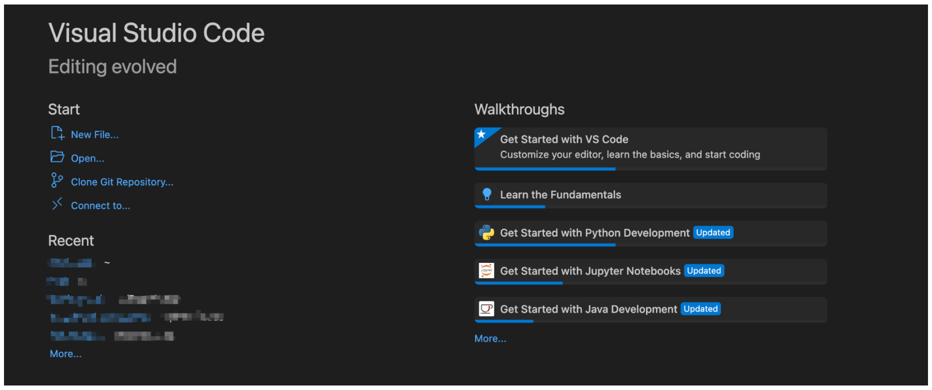

Visual Studio Code
Editing evolved
Start
New File...
Open...
Clone Git Repository...
Connect to...
Recent
More...
Walkthroughs
Get Started with VS Code
Customize your editor, learn the basics, and start coding
Learn the Fundamentals
Get Started with Python Development Updated
Get Started with Jupyter Notebooks Updated
Get Started with Java Development Updated
More...

Figure 1*: VS Code Default Window*

2. Open your existing project folder or create a new one:

    a. Click on “Open”.

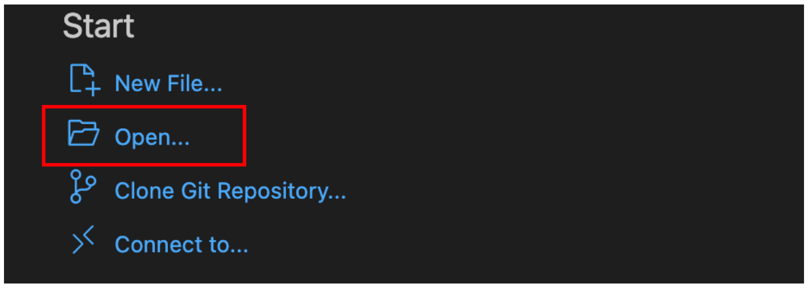


***Figure 2**: Opening the project directory in VS Code*

    b. Select the folder and click on “Open”.

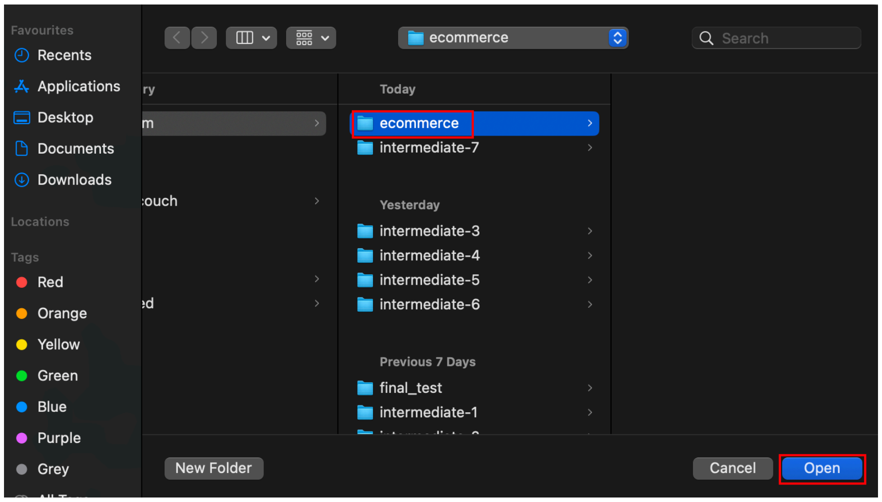
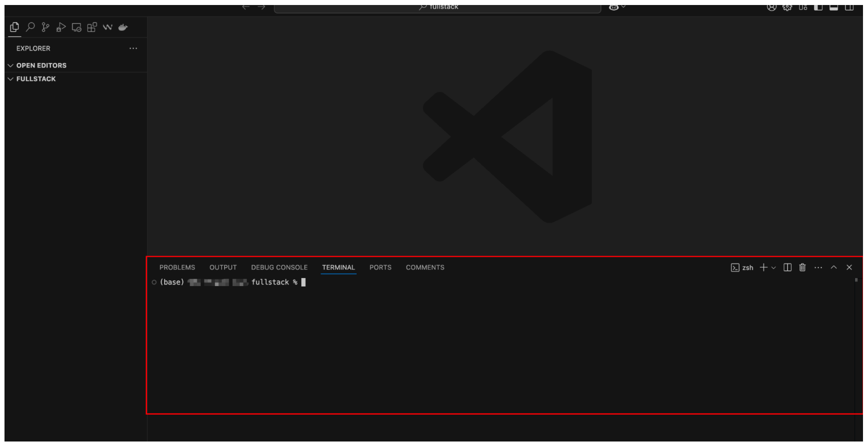


***Figure 3**: Selecting the project directory*

3. Open the terminal with Ctrl+` (backtick)



```
(base) [redacted] fullstack %
```

***Figure 4**: VS Code’s Terminal*

2. Initialize the Node.js Project

Start by creating a new directory for your backend project, navigate into it via VS Code's terminal, and initialize a fresh Node.js project using `npm`:

```
mkdir backend
cd backend
npm init -y
```

The `-y` flag will accept default settings and generate a `package.json` file.

Step 2: Install Dependencies

Next, install the required packages: `express` for the web framework, `mongoose` for MongoDB object modeling, `cors` to handle cross-origin requests from the frontend, and `dotenv` to manage environment variables.

```
npm install express mongoose cors dotenv
```

Step 3: Create `.env` File

Create a file named `.env` in your `backend` directory. This file will store your MongoDB connection string. **Replace `<your_mongodb_connection_string>` with your actual MongoDB URI.** If you don't have one, you can get a free cluster from MongoDB Atlas.

```
MONGO_URI=mongodb+srv://<username>:<password>@cluster0.your_mongodb_server.
net/ecommerce?retryWrites=true&w=majority
PORT=5000
```

Step 4: Define the Product Model

Create a folder named `models` inside `backend`, and then create a file named `Product.js` inside the `models` folder. This file will define the Mongoose schema for our product.

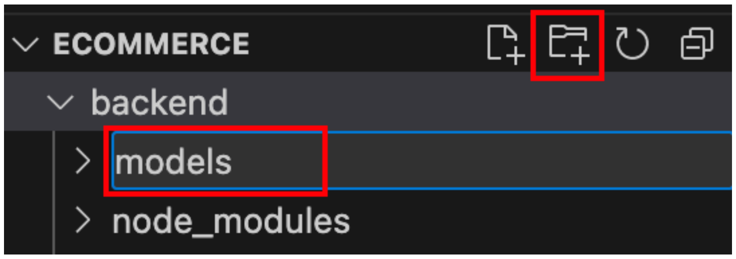


***Figure 5**: models directory creation*

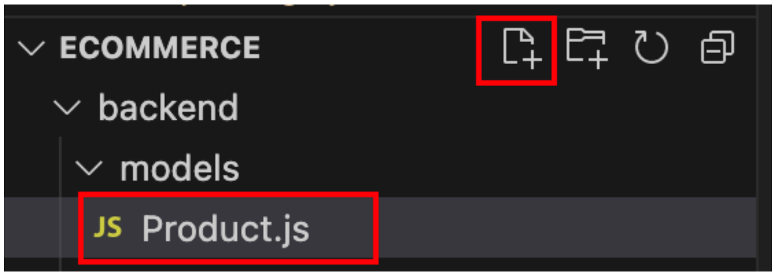


***Figure 6**: Product.js creation*

```
// backend/models/Product.js

const mongoose = require('mongoose');

// Define the schema for a Product
const productSchema = new mongoose.Schema({
 name: {
   type: String,
   required: [true, 'Product name is required'], // Name is a
required string
   trim: true, // Remove whitespace from both ends of a string
   unique: true // Ensure product names are unique
 },
 description: {
   type: String,
   required: [true, 'Product description is required'] // Description
is a required string
 },
 price: {
   type: Number,
   required: [true, 'Product price is required'], // Price is a
required number
```

```
    min: [0, 'Price cannot be negative'] // Price must be non-negative
  },
  imageUrl: {
    type: String,
    default: 'https://placehold.co/600x400/FFF/000?text=No+Image' //
Default image URL if none provided
  },
  createdAt: {
    type: Date,
    default: Date.now // Automatically set creation date
  }
});

// Create and export the Product model based on the schema
module.exports = mongoose.model('Product', productSchema);
```

Step 5: Create Product Controllers

Create a folder named `controllers` inside `backend`, and then create a file named `productController.js` inside the `controllers` folder. This file will contain the logic for handling product-related operations (CRUD: Create, Read, Update, Delete).

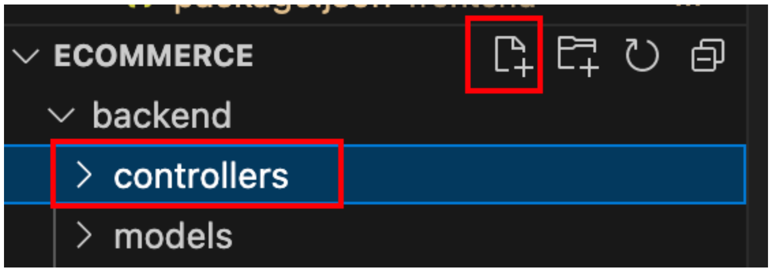


***Figure 7**: controller directory creation*

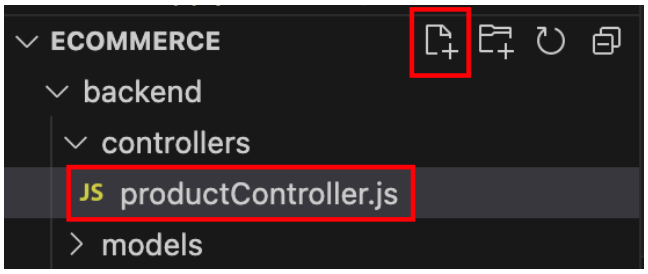


***Figure 8**: productController.js creation*

```
// backend/controllers/productController.js
```

```
const Product = require('../models/Product'); // Import the Product model

// @desc    Get all products
// @route   GET /api/products
// @access  Public
exports.getAllProducts = async (req, res) => {
 try {
   // Find all products in the database
   const products = await Product.find({});
   // Send the products as a JSON response
   res.status(200).json(products);
 } catch (error) {
   // If an error occurs, send a 500 status with an error message
   console.error('Error fetching products:', error);
   res.status(500).json({ message: 'Server Error: Could not retrieve
products.' });
 }
};

// @desc    Add a new product
// @route   POST /api/products
// @access  Public
exports.addProduct = async (req, res) => {
 try {
   // Extract product details from the request body
   const { name, description, price, imageUrl } = req.body;

   // Validate if required fields are present
   if (!name || !description || !price) {
     return res.status(400).json({ message: 'Please enter all required
fields: name, description, and price.' });
   }

   // Create a new product instance
   const newProduct = new Product({
     name,
     description,
     price,
     imageUrl
   });

   // Save the new product to the database
   const product = await newProduct.save();
   // Send the newly created product as a JSON response with 201 status
(Created)
   res.status(201).json(product);
 } catch (error) {
   // Handle potential errors, e.g., duplicate product name (due to unique:
true in schema)
   if (error.code === 11000) { // MongoDB duplicate key error code
     return res.status(409).json({ message: 'A product with this name
already exists.' });
   }
   console.error('Error adding product:', error);
   res.status(500).json({ message: 'Server Error: Could not add product.'
});
 }
};

// @desc    Update a product by ID
// @route   PUT /api/products/:id
// @access  Public
exports.updateProduct = async (req, res) => {
 try {
   // Find the product by ID and update it with the request body data
   // { new: true } returns the updated document
   // { runValidators: true } runs schema validators on update
   const product = await Product.findByIdAndUpdate(req.params.id, req.body,
{
     new: true,
     runValidators: true
   });

   // If no product is found with the given ID, send a 404 response
   if (!product) {
     return res.status(404).json({ message: 'Product not found.' });
   }

   // Send the updated product as a JSON response
   res.status(200).json(product);
 } catch (error) {
   console.error('Error updating product:', error);
   res.status(500).json({ message: 'Server Error: Could not update
product.' });
 }
};
```

```
// @desc    Delete a product by ID
// @route   DELETE /api/products/:id
// @access  Public
exports.deleteProduct = async (req, res) => {
 try {
   // Find the product by ID and delete it
   const product = await Product.findByIdAndDelete(req.params.id);

   // If no product is found, send a 404 response
   if (!product) {
     return res.status(404).json({ message: 'Product not found.' });
   }

   // Send a success message
   res.status(200).json({ message: 'Product successfully deleted.' });
 } catch (error) {
   console.error('Error deleting product:', error);
   res.status(500).json({ message: 'Server Error: Could not delete
product.' });
 }
};
```

Step 6: Define Product Routes

Create a folder named `routes` inside `backend`, and then create a file named `productRoutes.js` inside the `routes` folder. This file will define the API endpoints that will be exposed by our backend.

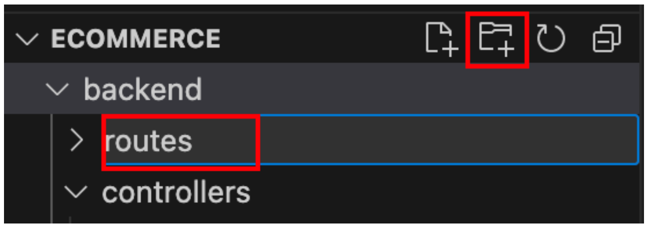


Figure 9:*routes directory creation*

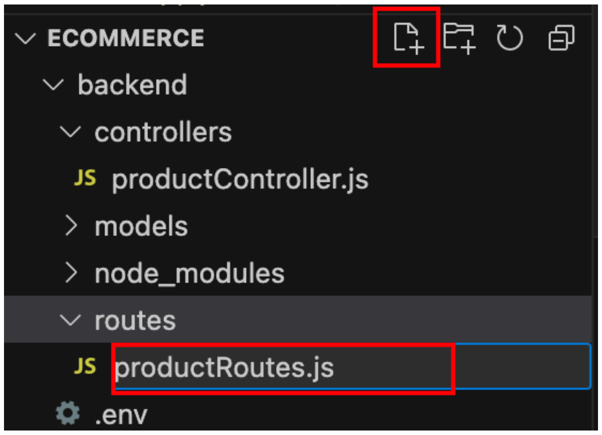

ECOMMERCE
backend
controllers
productController.js
models
node_modules
routes
productRoutes.js
.env

***Figure 10**: productRoutes.js creation*

```
// backend/routes/productRoutes.js

const express = require('express');
const router = express.Router(); // Create an Express router
const {
 getAllProducts,
 addProduct,
 updateProduct,
 deleteProduct
} = require('../controllers/productController'); // Import controller
functions

// Define routes and associate them with controller functions
router.route('/')
 .get(getAllProducts) // GET /api/products - Get all products
 .post(addProduct);    // POST /api/products - Add a new product

router.route('/:id')
 .put(updateProduct)    // PUT /api/products/:id - Update a product by ID
 .delete(deleteProduct); // DELETE /api/products/:id - Delete a product by
ID

module.exports = router; // Export the router
```

Step 7: Create the Main Server File

Create a file named `server.js` in the `backend` directory. This file will be the entry point for your backend application, setting up the Express server, connecting to the database, and integrating the routes.

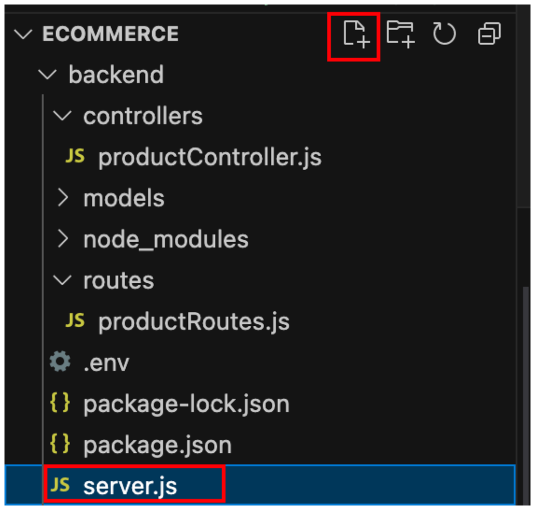


***Figure 11**: server.js creation*

```
// backend/server.js

const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');
const productRoutes = require('./routes/productRoutes'); // Import product
routes

// Load environment variables from .env file
dotenv.config();

const app = express(); // Initialize Express app

// Middleware
app.use(express.json()); // Enable parsing of JSON request bodies
app.use(cors()); // Enable CORS for all origins

// Connect to MongoDB
const connectDB = async () => {
 try {
   // Connect to MongoDB using the URI from environment variables
   await mongoose.connect(process.env.MONGO_URI);
   console.log('MongoDB Connected...');
 } catch (err) {
   // Log any connection errors and exit the process
   console.error('MongoDB connection error:', err.message);
   process.exit(1); // Exit process with failure
 }
};

// Call the connectDB function to establish database connection
connectDB();

// API Routes
// All requests to /api/products will be handled by productRoutes
app.use('/api/products', productRoutes);

// Basic route for testing server
app.get('/', (req, res) => {
 res.send('API is running...');
});

// Define the port to listen on, default to 5000 if not specified in .env
const PORT = process.env.PORT || 5000;

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

Step 8: Run the Backend Server

Now that all backend files are set up, you can start the server.

1. Start the server:

```
node server.js
```

You should see output similar to:

```
Server running on port 5000
MongoDB Connected...
```

This indicates your backend is running and connected to your MongoDB database. You can now test the API endpoints using tools like Postman or Insomnia, or proceed to build the frontend.

Now, let's set up the React frontend. This will involve creating a new React application, installing necessary packages, and building components to interact with our backend API. Open another terminal or close the process with **Ctrl + C**.

Frontend Creation

Step 1: Create React Application

1. Open your terminal or command prompt. Navigate back to the root `ecommerce` directory (assuming you are inside backend directory) :

```
cd..
```

2. Create a new React application named `frontend`:

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

3. This might take a few minutes.

Step 2: Navigate to Frontend Directory and Install Dependencies

Once the React app is created, navigate into the `frontend` directory:

```
cd frontend
```

Install `axios` for making HTTP requests to your backend

```
npm install axios
```

Step 3: Create ProductList Component

Create a folder named `components` inside `frontend/src`, and then create a file named `ProductList.js` inside the `components` folder. This component will be responsible for displaying the list of products fetched from the backend.

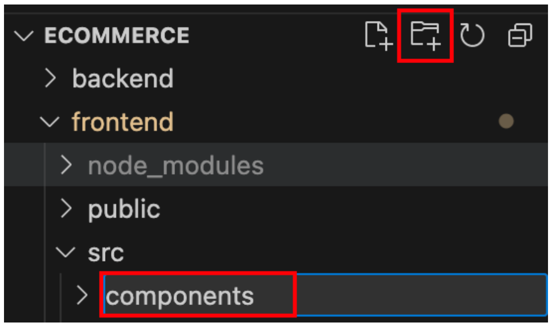


***Figure 12**: components directory creation*

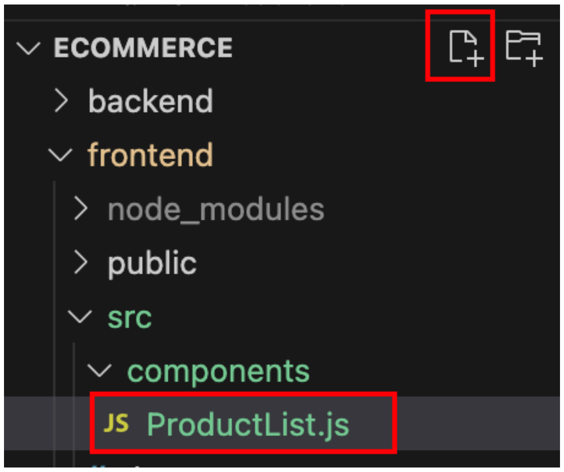


***Figure 13**: ProductList.js creation*

ProductList.js

```
// frontend/src/components/ProductList.js

import React from 'react';
import './ProductList.css'; // Import the ProductList CSS

const ProductList = ({ products, onDeleteProduct }) => {
 return (
   <div className="product-list-container">
     <h2 className="product-list-title">Available Products</h2>
     {products.length === 0 ? (
       <p className="no-products-message">No products available. Add
some!</p>
     ) : (
       <div className="product-grid">
         {products.map((product) => (
           <div key={product._id} className="product-card">
             <img
```

```
                src={product.imageUrl}
                alt={product.name}
                className="product-image"
                onError={(e) => {
                  e.target.onerror = null;
                  e.target.src =
'https://placehold.co/600x400/FFF/000?text=Image+Not+Found';
                }}
              />
              <h3 className="product-name">{product.name}</h3>
              <p className="product-description">{product.description}</p>
              <p className="product-price">${product.price.toFixed(2)}</p>
              <button
                onClick={() => onDeleteProduct(product._id)}
                className="delete-button"
              >
                Delete
              </button>
            </div>
          ))}
        </div>
      )}
    </div>
  );
};

export default ProductList;
```

Create ProductList.css in the same directory

ProductList.css

```
/* frontend/src/components/ProductList.css */

.product-list-container {
 background-color: white;
 padding: 24px;
 border-radius: 8px;
 box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0,
0.05); /* shadow-lg */
 max-width: 1200px; /* max-w-4xl, adjusted for larger screen */
 margin: 32px auto; /* mx-auto my-8 */
}
```

```css
.product-list-title {
 font-size: 2rem; /* 3xl */
 font-weight: bold;
 color: #1f2937; /* gray-800 */
 margin-bottom: 24px;
 text-align: center;
}

.no-products-message {
 text-align: center;
 color: #4b5563; /* gray-600 */
 font-size: 1.125rem; /* lg */
}

.product-grid {
 display: grid;
 grid-template-columns: 1fr; /* Default for mobile */
 gap: 24px; /* gap-6 */
}

@media (min-width: 640px) { /* sm breakpoint */
 .product-grid {
   grid-template-columns: repeat(2, 1fr); /* sm:grid-cols-2 */
 }
}

@media (min-width: 768px) { /* md breakpoint */
 .product-grid {
   grid-template-columns: repeat(3, 1fr); /* md:grid-cols-3 */
 }
}

@media (min-width: 1024px) { /* lg breakpoint */
 .product-grid {
   grid-template-columns: repeat(4, 1fr); /* lg:grid-cols-4 */
 }
}

.product-card {
 border: 1px solid #e5e7eb; /* border-gray-200 */
 border-radius: 8px;
 padding: 16px;
 display: flex;
 flex-direction: column;
 align-items: center;
 text-align: center;
 box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0,
0.06); /* shadow-md */
 transition: all 0.3s ease-in-out; /* transition-shadow duration-300 */
}

.product-card:hover {
 box-shadow: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(0, 0,
0, 0.04); /* hover:shadow-xl */
}

.product-image {
 width: 100%;
 height: 160px; /* h-40 */
 object-fit: cover;
 border-radius: 6px; /* rounded-md */
 margin-bottom: 16px;
}

.product-name {
 font-size: 1.25rem; /* xl */
 font-weight: 600; /* semibold */
 color: #111827; /* gray-900 */
 margin-bottom: 8px;
}

.product-description {
 color: #4b5563; /* gray-600 */
 font-size: 0.875rem; /* sm */
 margin-bottom: 12px;
 flex-grow: 1; /* flex-grow */
 overflow: hidden;
 text-overflow: ellipsis;
 display: -webkit-box;
 -webkit-line-clamp: 3; /* Limit to 3 lines */
 -webkit-box-orient: vertical;
}

.product-price {
 font-size: 1.5rem; /* 2xl */
 font-weight: bold;
 color: #4f46e5; /* indigo-600 */
 margin-bottom: 16px;
}
```

```
.delete-button {
 background-color: #ef4444; /* red-500 */
 color: white;
 font-weight: bold;
 padding: 8px 16px; /* py-2 px-4 */
 border-radius: 9999px; /* rounded-full */
 box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0,
0.06); /* shadow-md */
 transition: all 0.3s ease-in-out; /* transition-all duration-300 ease-in-out
*/
 transform: scale(1); /* Initial scale */
 border: none;
 cursor: pointer;
 outline: none; /* focus:outline-none */
}

.delete-button:hover {
 background-color: #dc2626; /* hover:bg-red-600 */
 box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0,
0.05); /* hover:shadow-lg */
 transform: scale(1.05); /* hover:scale-105 */
}

.delete-button:focus {
 box-shadow: 0 0 0 3px rgba(239, 68, 68, 0.5); /* focus:ring-2
focus:ring-red-500 focus:ring-opacity-75 */
}
```

Step 6: Create AddProductForm Component

Create a file named `AddProductForm.js` inside the `frontend/src/components` folder. This component will provide a form for users to input details for a new product and add it to the database.

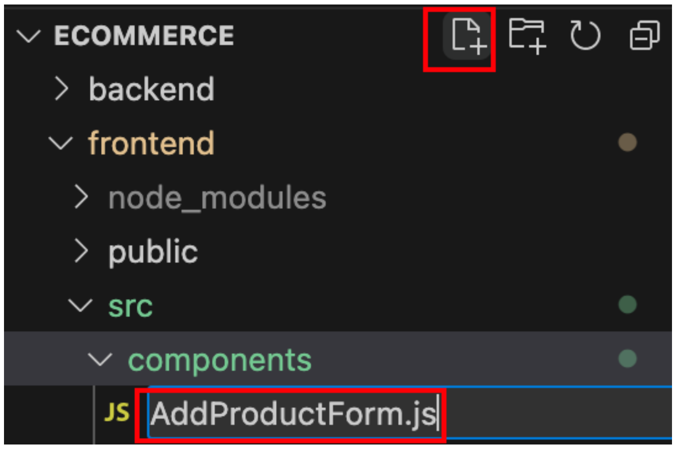


Figure 14: *AddProductForm.js creation*

AddProductForm.js

```
// frontend/src/components/AddProductForm.js

import React, { useState } from 'react';
import './AddProductForm.css'; // Import the AddProductForm CSS

const AddProductForm = ({ onAddProduct }) => {
 const [name, setName] = useState('');
 const [description, setDescription] = useState('');
 const [price, setPrice] = useState('');
 const [imageUrl, setImageUrl] = useState('');
 const [message, setMessage] = useState('');

 const handleSubmit = async (e) => {
   e.preventDefault();

   if (!name || !description || !price) {
     setMessage('Please fill in all required fields (Name, Description,
Price).');
     return;
   }
   if (isNaN(price) || parseFloat(price) < 0) {
     setMessage('Price must be a non-negative number.');
     return;
   }

   const newProduct = {
     name,
     description,
     price: parseFloat(price),
     imageUrl: imageUrl ||
'https://placehold.co/600x400/FFF/000?text=No+Image'
   };

   try {
     await onAddProduct(newProduct);
     setMessage('Product added successfully!');
     setName('');
     setDescription('');
     setPrice('');
     setImageUrl('');
   } catch (error) {
     setMessage(error.message || 'Failed to add product. Please try
again.');
   }
 };

 return (
   <div className="add-product-form-container">
     <h2 className="form-title">Add New Product</h2>
     <form onSubmit={handleSubmit} className="product-form">
       <div className="form-group">
         <label htmlFor="name" className="form-label">
           Product Name:
         </label>
         <input
           type="text"
           id="name"
           value={name}
           onChange={(e) => setName(e.target.value)}
           className="form-input"
           required
         />
       </div>

       <div className="form-group">
         <label htmlFor="description" className="form-label">
           Description:
         </label>
         <textarea
           id="description"
           value={description}
           onChange={(e) => setDescription(e.target.value)}
           rows="3"
           className="form-textarea"
           required
         ></textarea>
       </div>

       <div className="form-group">
         <label htmlFor="price" className="form-label">
           Price ($):
         </label>
         <input
           type="number"
           id="price"
           value={price}
           onChange={(e) => setPrice(e.target.value)}
           className="form-input"
           step="0.01"
           required
         />
       </div>

       <div className="form-group">
         <label htmlFor="imageUrl" className="form-label">
           Image URL (Optional):
         </label>
         <input
           type="text"
           id="imageUrl"
           value={imageUrl}
           onChange={(e) => setImageUrl(e.target.value)}
           className="form-input"
           placeholder="e.g., https://example.com/image.jpg"
         />
       </div>

       <button
         type="submit"
         className="submit-button"
       >
         Add Product
       </button>
```

```
       {message && (
         <p className={`form-message ${message.includes('successfully') ?
'success' : 'error'}`}>
           {message}
         </p>
       )}
     </form>
   </div>
 );
};

export default AddProductForm;
```

Create AddProductForm.css in the same directory

AddProductForm.css

```css
/* frontend/src/components/AddProductForm.css */

.add-product-form-container {
 background-color: white;
 padding: 24px;
 border-radius: 8px;
 box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0,
0, 0.05); /* shadow-lg */
 max-width: 448px; /* max-w-md */
 margin: 32px auto; /* mx-auto my-8 */
}

.form-title {
 font-size: 2rem; /* 3xl */
 font-weight: bold;
 color: #1f2937; /* gray-800 */
 margin-bottom: 24px;
 text-align: center;
}

.product-form {
 display: flex;
 flex-direction: column;
 gap: 16px; /* space-y-4 */
```

```css
}

.form-group {
 margin-bottom: 16px;
}

.form-label {
 display: block;
 color: #374151; /* gray-700 */
 font-size: 0.875rem; /* sm */
 font-weight: bold;
 margin-bottom: 8px;
}

.form-input,
.form-textarea {
 box-shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.1), 0 1px 2px 0 rgba(0, 0, 0,
0.06); /* shadow */
 appearance: none;
 border: 1px solid #d1d5db; /* border */
 border-radius: 4px;
 width: 100%;
 padding: 8px 12px; /* py-2 px-3 */
 color: #374151; /* text-gray-700 */
 line-height: 1.25; /* leading-tight */
 outline: none; /* focus:outline-none */
 transition: all 0.2s ease-in-out;
}

.form-input:focus,
.form-textarea:focus {
 outline: none;
 box-shadow: 0 0 0 3px rgba(99, 102, 241, 0.5); /* focus:ring-2
focus:ring-indigo-500 focus:ring-opacity-75 */
 border-color: transparent; /* focus:border-transparent */
}

.form-textarea {
 resize: vertical; /* Allow vertical resizing */
}

.submit-button {
 background-color: #4f46e5; /* indigo-600 */
 color: white;
 font-weight: bold;
 padding: 8px 16px; /* py-2 px-4 */
 border-radius: 9999px; /* rounded-full */
 width: 100%;
 box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0,
0, 0.06); /* shadow-md */
 transition: all 0.3s ease-in-out; /* transition-all duration-300
ease-in-out */
 transform: scale(1); /* Initial scale */
 border: none;
 cursor: pointer;
 outline: none; /* focus:outline-none */
}

.submit-button:hover {
 background-color: #4338ca; /* hover:bg-indigo-700 */
 box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0,
0, 0.05); /* hover:shadow-lg */
 transform: scale(1.05); /* hover:scale-105 */
}

.submit-button:focus {
 box-shadow: 0 0 0 3px rgba(99, 102, 241, 0.5); /* focus:ring-2
focus:ring-indigo-500 focus:ring-opacity-75 */
}

.form-message {
 text-align: center;
 font-size: 0.875rem; /* sm */
 margin-top: 16px;
}

.form-message.success {
 color: #065f46; /* green-600 */
}

.form-message.error {
 color: #b91c1c; /* red-600 */
}
```

Step 7: Update `App.js`

Open `src/App.js`. This is the main component that will fetch products from the backend and manage the state for adding and deleting products. It will integrate `ProductList` and `AddProductForm`.

```
// frontend/src/App.js

import React, { useState, useEffect, useCallback } from 'react';
import axios from 'axios';
import ProductList from './components/ProductList';
import AddProductForm from './components/AddProductForm';
import './App.css'; // Import the main App CSS

// Define the base URL for the backend API
const API_URL = 'http://localhost:5000/api/products';

function App() {
 const [products, setProducts] = useState([]);
 const [error, setError] = useState(null);
 const [loading, setLoading] = useState(true);

 // Function to fetch products from the backend
 const fetchProducts = useCallback(async () => {
   setLoading(true);
   setError(null);
   try {
     const response = await axios.get(API_URL);
     setProducts(response.data);
   } catch (err) {
     console.error('Error fetching products:', err);
     setError('Failed to fetch products. Please ensure the backend is
running.');
   } finally {
     setLoading(false);
   }
 }, []);

 // useEffect hook to fetch products when the component mounts
 useEffect(() => {
   fetchProducts();
 }, [fetchProducts]);
 // Function to handle adding a new product
 const handleAddProduct = async (newProduct) => {
   setError(null);
   try {
     const response = await axios.post(API_URL, newProduct);
     setProducts((prevProducts) => [...prevProducts, response.data]);
     return { success: true };
   } catch (err) {
     console.error('Error adding product:', err);
     const errorMessage = err.response?.data?.message || 'Failed to add
product.';
     setError(errorMessage);
     throw new Error(errorMessage);
   }
 };

 // Function to handle deleting a product
 const handleDeleteProduct = async (id) => {
   setError(null);
   try {
     await axios.delete(`${API_URL}/${id}`);
     setProducts((prevProducts) => prevProducts.filter((product) =>
product._id !== id));
   } catch (err) {
     console.error('Error deleting product:', err);
     setError('Failed to delete product. Please try again.');
   }
 };

 return (
   <div className="app-container">
     <header className="app-header">
       <h1 className="app-title">Mini E-commerce Store</h1>
     </header>

     <main className="app-main">
       {/* Error Message Display */}
       {error && (
         <div className={`message-box ${error.includes('successfully') ?
'success' : 'error'}`}>
           <strong>Error!</strong>
           <span> {error}</span>
```

```
          </div>
        )}

        {/* Loading Indicator */}
        {loading && (
          <div className="loading-indicator">
            Loading products...
          </div>
        )}

        {/* Add Product Form */}
        <AddProductForm onAddProduct={handleAddProduct} />

        {/* Product List */}
        {!loading && <ProductList products={products}
onDeleteProduct={handleDeleteProduct} />}
      </main>

      <footer className="app-footer">
        <p>&copy; {new Date().getFullYear()} Mini E-commerce. All rights
reserved.</p>
      </footer>
    </div>
  );
}

export default App;
```

Updated app.css

```
/* frontend/src/App.css */

/* General body styles */
body {
 margin: 0;
 font-family: 'Inter', sans-serif;
 -webkit-font-smoothing: antialiased;
 -moz-osx-font-smoothing: grayscale;
 background-color: #f0f2f5; /* Light grey background */
}

/* App container */
.app-container {
 min-height: 100vh;
```

```css
 display: flex;
 flex-direction: column;
 align-items: center;
 padding-top: 40px;
 padding-bottom: 40px;
}

/* Header styles */
.app-header {
 width: 100%;
 background-color: #4f46e5; /* Indigo 700 */
 color: white;
 padding: 24px 0;
 box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
 margin-bottom: 32px;
}

.app-title {
 font-size: 2.25rem; /* 4xl */
 font-weight: 800; /* Extrabold */
 text-align: center;
 letter-spacing: -0.025em; /* Tracking-tight */
}

/* Main content area */
.app-main {
 width: 100%;
 max-width: 1200px; /* Equivalent to container mx-auto */
 padding-left: 16px; /* px-4 */
 padding-right: 16px; /* px-4 */
}

/* Message Box (for errors/success) */
.message-box {
 background-color: #fee2e2; /* Red 100 */
 border: 1px solid #ef4444; /* Red 400 */
 color: #b91c1c; /* Red 700 */
 padding: 12px 16px;
 border-radius: 8px;
 position: relative;
 margin-bottom: 24px;
 text-align: center;
}
```

```css
.message-box strong {
 font-weight: bold;
}

.message-box span {
 margin-left: 8px;
 display: block; /* For small screens */
}

@media (min-width: 640px) { /* sm breakpoint */
 .message-box span {
   display: inline;
 }
}

.message-box.success {
 background-color: #d1fae5; /* Green 100 */
 border-color: #34d399; /* Green 400 */
 color: #065f46; /* Green 700 */
}

.message-box.error {
 background-color: #fee2e2; /* Red 100 */
 border-color: #ef4444; /* Red 400 */
 color: #b91c1c; /* Red 700 */
}

/* Loading Indicator */
.loading-indicator {
 text-align: center;
 color: #4f46e5; /* Indigo 700 */
 font-size: 1.25rem; /* xl */
 font-weight: 600; /* Semibold */
 margin-top: 32px;
 margin-bottom: 32px;
}

/* Footer styles */
.app-footer {
 width: 100%;
 background-color: #1f2937; /* Gray 800 */
 color: white;
 padding: 16px 0;
 margin-top: auto; /* Pushes footer to the bottom */
 text-align: center;
}
```

Output:

Go to your parent directory “ecommerce” first.

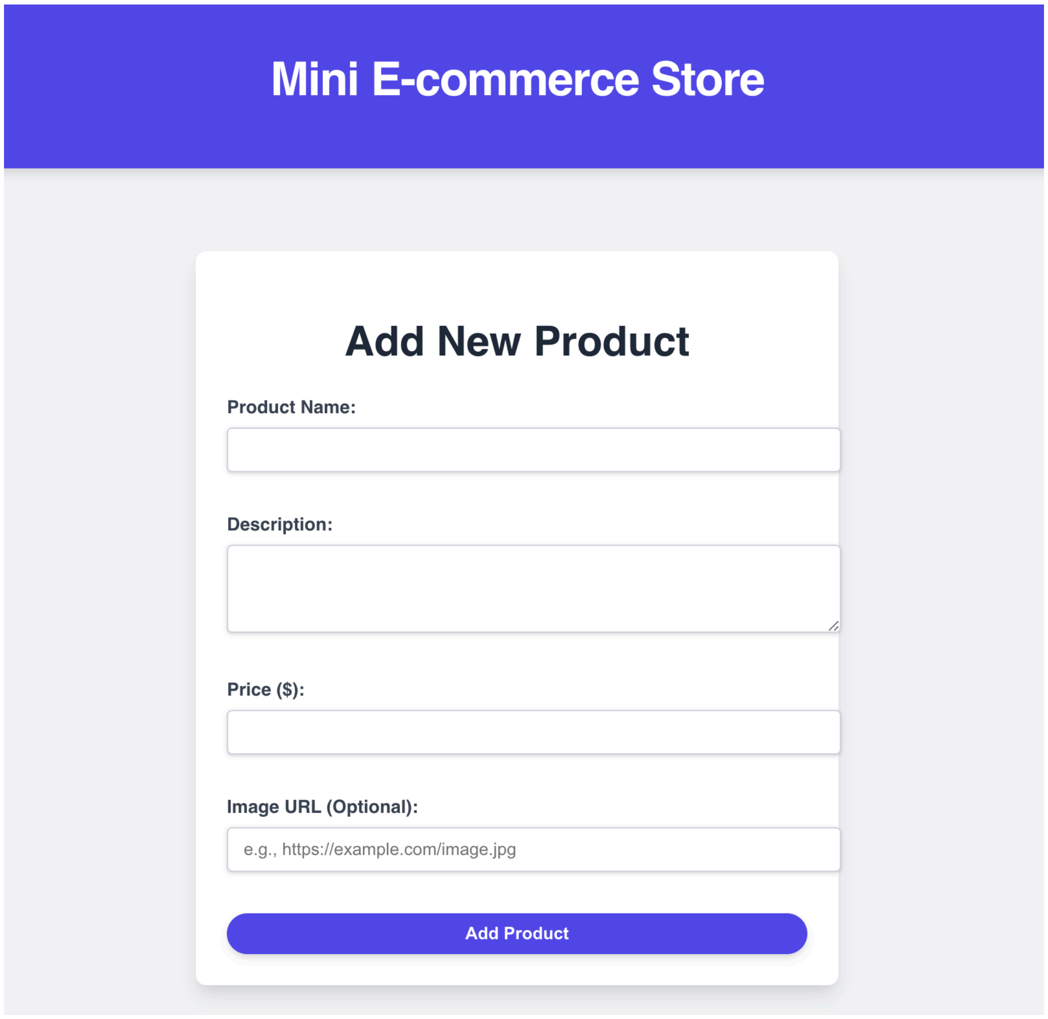
Run the frontend

```
cd frontend
npm start
```

Run the backend is separate terminal

```
cd backend
node server.js
```

Note*: For Image ,* ***copy image link / copy image address*** *and paste the url.*


Mini E-commerce Store

Add New Product

Product Name:

Description:

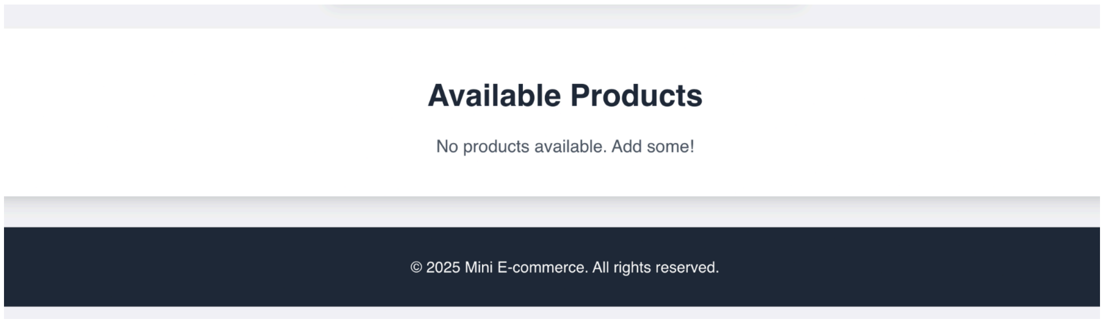
Price ($):

Image URL (Optional):

e.g., https://example.com/image.jpg

Add Product

Figure 15*: Add Product Form UI*


Available Products

No products available. Add some!

© 2025 Mini E-commerce. All rights reserved.

Figure 16*: Initial Stage as Zero Products added*

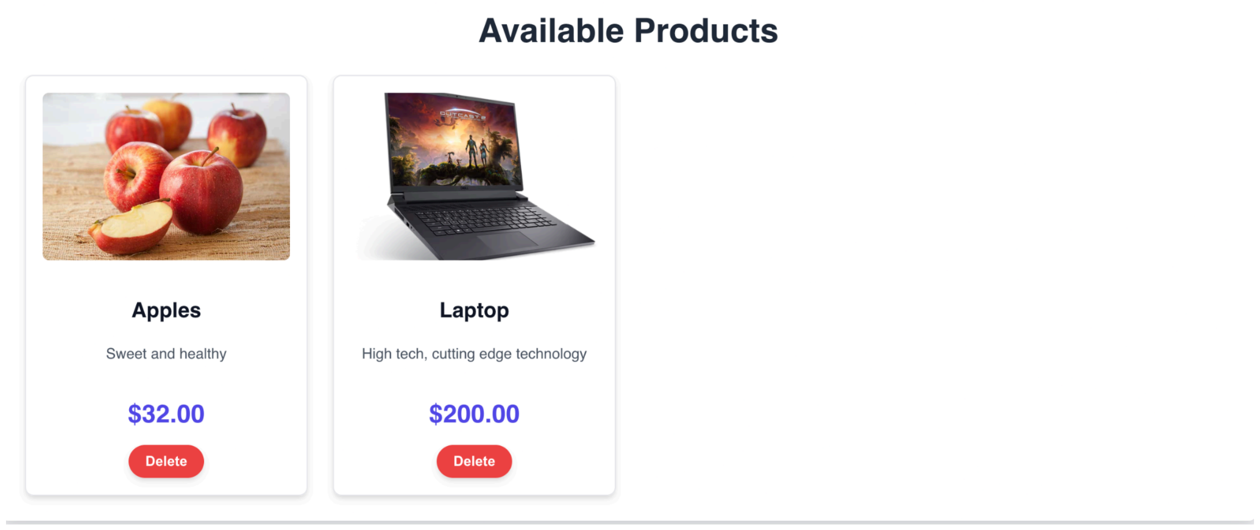


***Figure 17**: Products added*

Future Enhancements

This mini e-commerce project provides a solid foundation. Here are some small yet impactful future enhancements that could be added to expand its functionality and improve the user experience:

1. **Product Details Page:**
    - **Enhancement:** Allow users to click on a product in the listing to navigate to a dedicated page displaying more detailed information about that specific product, perhaps with a larger image, extended description, and a "Buy Now" or "Add to Cart" button.
    - **Implementation Idea:** Create a new React component (e.g., `ProductDetail.js`), add a route in React Router to handle `/products/:id`, and fetch individual product data from a new backend API endpoint (`/api/products/:id`).
2. **Basic Shopping Cart Functionality (Frontend Focus):**
    - **Enhancement:** Enable users to "add to cart" products without requiring a full backend cart system initially. This would involve managing a temporary cart state in the frontend (e.g., using React Context or local storage).
    - **Implementation Idea:** Add an "Add to Cart" button on each product card. When clicked, it adds the product (and its quantity) to a global state or local storage. A small cart icon in the header could display the number of items.

3. **Product Search & Filtering:**

    - **Enhancement:** Implement a search bar to allow users to find products by name or description. You could also add basic filters (e.g., by price range, or if you add categories, by category).
    - **Implementation Idea:** On the frontend, add an input field for search. When the user types, send a request to the backend (e.g., `/api/products?search=apple`) and update the product list based on the results. The backend's `getAllProducts` controller would need to be modified to accept and process search query parameters.

4. **Direct Image Upload:**

    - **Enhancement:** Instead of just pasting an image URL, allow users (e.g., administrators) to directly upload an image file when adding a new product.
    - **Implementation Idea:** On the frontend, replace the `imageUrl` text input with a file input (`<input type="file">`). On the backend, integrate a library like `multer` to handle file uploads, store the image on the server (or upload to a cloud storage like Cloudinary/AWS S3), and then save the resulting image URL in MongoDB.

5. **User Authentication (Basic):**

    - **Enhancement:** Introduce simple user registration and login. Initially, you could just have one "admin" user who can add/delete products.
    - **Implementation Idea:** Add user models and routes in the backend (e.g., `/api/auth/register`, `/api/auth/login`). Use JSON Web Tokens (JWT) for authentication. On the frontend, add login/logout forms and conditionally render the "Add Product" form and "Delete" buttons only if the user is authenticated as an admin.

Conclusion

In this hands-on lab, you successfully built a complete mini e-commerce application using the MERN stack. You developed a robust backend RESTful API with Node.js, Express, and MongoDB to manage products, and crafted a React frontend to display product listings and add new items via a form. This project emphasized the importance of clear separation of concerns between backend and frontend, as well as seamless full-stack integration through API communication. Equipped with these foundational skills, you are now well-prepared to extend your app with advanced features such as user authentication, shopping cart functionality, or cloud deployment to further enhance your MERN development expertise.