You will need to create the files and folders yourself, copy the code into them, and then follow the setup instructions to run the application.

**Mental Model**

1. **Backend (Server):** We'll create the API first. It will handle data (products, users), authentication (login/register), and cart logic.
2. **Frontend (Client):** The React app will talk to our backend API to display products, manage the cart, and handle user login/logout.
3. **Root:** We'll set up scripts to run both the frontend and backend together for easy development.

**⭐️ Step 1: Project & Folder Setup**

First, create the main project folder and the necessary sub-folders.

Generated bash

mkdir ecommerce-app

cd ecommerce-app

# Create server and client directories

mkdir server

mkdir client

# Create other necessary folders

mkdir seed

touch .env.example .gitignore README.md

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

**⭐️ Step 2: The Backend (Server-Side)**

Let's build the Node.js + Express backend first.

**1. Server package.json**

Create the file server/package.json and add the following content. This defines our backend dependencies.

Generated json

// File: server/package.json

{

"name": "ecommerce-server",

"version": "1.0.0",

"description": "Backend for the e-commerce app",

"main": "app.js",

"type": "module",

"scripts": {

"start": "node app.js",

"dev": "nodemon app.js",

"seed": "node -r dotenv/config --env-file=../.env seed.js"

},

"keywords": [],

"author": "Amjad Manzoor",

"license": "ISC",

"dependencies": {

"bcryptjs": "^2.4.3",

"connect-mongo": "^5.1.0",

"cors": "^2.8.5",

"dotenv": "^16.3.1",

"express": "^4.18.2",

"express-session": "^1.17.3",

"helmet": "^7.1.0",

"mongoose": "^8.0.3",

"passport": "^0.7.0",

"passport-google-oauth20": "^2.0.0",

"passport-local": "^1.0.0"

},

"devDependencies": {

"nodemon": "^3.0.2"

}

}

Use code [with caution](https://support.google.com/legal/answer/13505487).Json

**2. User Model**

Create server/models/User.js. This defines the schema for our users, including a hook to hash passwords before saving.

Generated javascript

// File: server/models/User.js

import mongoose from 'mongoose';

import bcrypt from 'bcryptjs';

const userSchema = new mongoose.Schema({

email: {

type: String,

unique: true,

sparse: true, // Allows multiple null values for email

},

password: {

type: String,

},

googleId: {

type: String,

unique: true,

sparse: true, // Allows multiple null values for googleId

},

displayName: {

type: String,

},

}, { timestamps: true });

// Hash password before saving

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

if (!this.isModified('password') || !this.password) {

return next();

}

try {

const salt = await bcrypt.genSalt(10);

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

next();

} catch (err) {

next(err);

}

});

// Method to compare passwords

userSchema.methods.comparePassword = function(candidatePassword) {

return bcrypt.compare(candidatePassword, this.password);

};

const User = mongoose.model('User', userSchema);

export default User;

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**3. Product Model**

Create server/models/Product.js for the product schema.

Generated javascript

// File: server/models/Product.js

import mongoose from 'mongoose';

const productSchema = new mongoose.Schema({

title: {

type: String,

required: true,

},

description: {

type: String,

required: true,

},

price: {

type: Number,

required: true,

},

image: {

type: String,

required: true,

},

}, { timestamps: true });

const Product = mongoose.model('Product', productSchema);

export default Product;

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**4. Passport Configuration**

Create server/config/passport.js. This file will configure our authentication strategies (local and Google).

Generated javascript

// File: server/config/passport.js

import { Strategy as GoogleStrategy } from 'passport-google-oauth20';

import { Strategy as LocalStrategy } from 'passport-local';

import User from '../models/User.js';

export default function(passport) {

// Google OAuth Strategy

passport.use(new GoogleStrategy({

clientID: process.env.GOOGLE\_CLIENT\_ID,

clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,

callbackURL: process.env.GOOGLE\_CALLBACK

},

async (accessToken, refreshToken, profile, done) => {

const newUser = {

googleId: profile.id,

displayName: profile.displayName,

email: profile.emails[0].value

};

try {

let user = await User.findOne({ googleId: profile.id });

if (user) {

done(null, user);

} else {

user = await User.create(newUser);

done(null, user);

}

} catch (err) {

console.error(err);

done(err, null);

}

}));

// Local Strategy

passport.use(new LocalStrategy({ usernameField: 'email' }, async (email, password, done) => {

try {

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

if (!user) {

return done(null, false, { message: 'Incorrect email.' });

}

if (!user.password) { // User registered with Google

return done(null, false, { message: 'You have previously signed up with Google. Please use Google to log in.' });

}

const isMatch = await user.comparePassword(password);

if (isMatch) {

return done(null, user);

} else {

return done(null, false, { message: 'Incorrect password.' });

}

} catch (err) {

return done(err);

}

}));

// Serialize and Deserialize User

passport.serializeUser((user, done) => {

done(null, user.id);

});

passport.deserializeUser(async (id, done) => {

try {

const user = await User.findById(id);

done(null, user);

} catch (err) {

done(err, null);

}

});

}

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**5. Routes**

Create the folder server/routes.

**Auth Routes: server/routes/auth.js**

Generated javascript

// File: server/routes/auth.js

import express from 'express';

import passport from 'passport';

import User from '../models/User.js';

const router = express.Router();

// Register a new user

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

const { email, password } = req.body;

try {

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

if (existingUser) {

return res.status(400).json({ message: 'User already exists.' });

}

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

await newUser.save();

req.login(newUser, (err) => {

if (err) { return res.status(500).json({ message: 'Session error' }); }

return res.status(201).json({\_id: req.user.\_id, displayName: req.user.displayName, email: req.user.email });

});

} catch (error) {

res.status(500).json({ message: 'Server error', error });

}

});

// Login with email/password

router.post('/login', (req, res, next) => {

passport.authenticate('local', (err, user, info) => {

if (err) return next(err);

if (!user) return res.status(401).json({ message: info.message });

req.logIn(user, (err) => {

if (err) return next(err);

return res.status(200).json({\_id: req.user.\_id, displayName: req.user.displayName, email: req.user.email });

});

})(req, res, next);

});

// Google OAuth routes

router.get('/google', passport.authenticate('google', { scope: ['profile', 'email'] }));

router.get('/google/callback', passport.authenticate('google', { failureRedirect: 'http://localhost:5173/login' }), (req, res) => {

res.redirect('http://localhost:5173/'); // Redirect to frontend home

});

// Logout

router.get('/logout', (req, res, next) => {

req.logout((err) => {

if (err) { return next(err); }

req.session.destroy((err) => {

res.clearCookie('connect.sid');

res.status(200).json({ message: 'Logged out successfully' });

});

});

});

// Check auth status

router.get('/status', (req, res) => {

if (req.isAuthenticated()) {

res.status(200).json({ user: {\_id: req.user.\_id, displayName: req.user.displayName, email: req.user.email } });

} else {

res.status(200).json({ user: null });

}

});

export default router;

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**Product Routes: server/routes/products.js**

Generated javascript

// File: server/routes/products.js

import express from 'express';

import Product from '../models/Product.js';

const router = express.Router();

// Get all products

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

try {

const products = await Product.find({});

res.json(products);

} catch (error) {

res.status(500).json({ message: 'Server Error' });

}

});

export default router;

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**Cart Routes: server/routes/cart.js**

Generated javascript

// File: server/routes/cart.js

import express from 'express';

const router = express.Router();

// Middleware to initialize cart in session

const initializeCart = (req, res, next) => {

if (!req.session.cart) {

req.session.cart = { items: [], total: 0 };

}

next();

};

router.use(initializeCart);

// Get cart

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

res.status(200).json(req.session.cart);

});

// Add item to cart

router.post('/add', (req, res) => {

const { product, quantity } = req.body;

const cart = req.session.cart;

const existingItemIndex = cart.items.findIndex(item => item.product.\_id === product.\_id);

if (existingItemIndex > -1) {

cart.items[existingItemIndex].quantity += quantity;

} else {

cart.items.push({ product, quantity });

}

cart.total = cart.items.reduce((acc, item) => acc + item.product.price \* item.quantity, 0);

req.session.save(() => {

res.status(200).json(cart);

});

});

// Update item quantity

router.put('/update/:productId', (req, res) => {

const { productId } = req.params;

const { quantity } = req.body;

const cart = req.session.cart;

const itemIndex = cart.items.findIndex(item => item.product.\_id === productId);

if (itemIndex > -1) {

cart.items[itemIndex].quantity = quantity;

if (cart.items[itemIndex].quantity <= 0) {

cart.items.splice(itemIndex, 1);

}

}

cart.total = cart.items.reduce((acc, item) => acc + item.product.price \* item.quantity, 0);

req.session.save(() => {

res.status(200).json(cart);

});

});

// Remove item from cart

router.delete('/remove/:productId', (req, res) => {

const { productId } = req.params;

const cart = req.session.cart;

const itemIndex = cart.items.findIndex(item => item.product.\_id === productId);

if (itemIndex > -1) {

cart.items.splice(itemIndex, 1);

}

cart.total = cart.items.reduce((acc, item) => acc + item.product.price \* item.quantity, 0);

req.session.save(() => {

res.status(200).json(cart);

});

});

export default router;

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**6. Main Server File: server/app.js**

This file ties everything together.

Generated javascript

// File: server/app.js

import express from 'express';

import mongoose from 'mongoose';

import dotenv from 'dotenv';

import cors from 'cors';

import helmet from 'helmet';

import session from 'express-session';

import passport from 'passport';

import MongoStore from 'connect-mongo';

import passportConfig from './config/passport.js';

import authRoutes from './routes/auth.js';

import productRoutes from './routes/products.js';

import cartRoutes from './routes/cart.js';

// Load Env Vars

dotenv.config({ path: '../.env' });

// Passport Config

passportConfig(passport);

const app = express();

// DB Connect

mongoose.connect(process.env.MONGODB\_URI)

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

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

// Middleware

app.use(cors({

origin: 'http://localhost:5173', // Your React app's origin

credentials: true

}));

app.use(helmet());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

// Session Middleware

app.use(session({

secret: process.env.SESSION\_SECRET,

resave: false,

saveUninitialized: false,

store: MongoStore.create({ mongoUrl: process.env.MONGODB\_URI }),

cookie: {

maxAge: 1000 \* 60 \* 60 \* 24 // 1 day

}

}));

// Passport Middleware

app.use(passport.initialize());

app.use(passport.session());

// Routes

app.use('/auth', authRoutes);

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

app.use('/api/cart', cartRoutes);

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**⭐️ Step 3: Database Seeder**

Create seed/seed.js to populate your database with initial data.

Generated javascript

// File: seed/seed.js

// File: server/seed.js

// NO LONGER NEEDED: import dotenv from 'dotenv';

// dotenv will be loaded by the npm script

import mongoose from 'mongoose';

import bcrypt from 'bcryptjs';

// Change these paths:

import User from './models/User.js'; // From '../server/models/User.js'

import Product from './models/Product.js'; // From '../server/models/Product.js'

// The rest of the file remains the same...

// Connect to the database using the environment variable

const MONGODB\_URI = process.env.MONGODB\_URI;

const seedDB = async () => {

if (!MONGODB\_URI) {

console.error('Error: MONGODB\_URI is not defined in your .env file.');

process.exit(1);

}

await mongoose.connect(MONGODB\_URI);

console.log('Clearing old data...');

await User.deleteMany({});

await Product.deleteMany({});

console.log('Seeding new data...');

// Seed Users

const users = [

{ email: 'user1@example.com', password: 'password123' },

{ email: 'user2@example.com', password: 'password123' },

{ email: 'user3@example.com', password: 'password123' },

];

for (const userData of users) {

const user = new User(userData);

await user.save(); // The pre-save hook will hash the password

}

console.log('Users seeded.');

// Seed Products

const products = [

{ title: 'Modern React Book', description: 'A comprehensive guide to modern React.', price: 29.99, image: 'https://via.placeholder.com/150/0000FF/808080?Text=ReactBook' },

{ title: 'Node.js for Beginners', description: 'Learn Node.js from scratch.', price: 24.99, image: 'https://via.placeholder.com/150/32CD32/FFFFFF?Text=Node.js' },

{ title: 'Tailwind CSS Pro', description: 'Master utility-first CSS.', price: 39.99, image: 'https://via.placeholder.com/150/87CEEB/FFFFFF?Text=Tailwind' },

{ title: 'Full-Stack JavaScript', description: 'Become a full-stack developer.', price: 49.99, image: 'https://via.placeholder.com/150/FFD700/000000?Text=Full-Stack' },

{ title: 'MongoDB Essentials', description: 'Deep dive into NoSQL databases.', price: 19.99, image: 'https://via.placeholder.com/150/4B8B4B/FFFFFF?Text=MongoDB' },

{ title: 'Express.js in Action', description: 'Build powerful web APIs.', price: 22.50, image: 'https://via.placeholder.com/150/6A0DAD/FFFFFF?Text=Express.js' },

];

await Product.insertMany(products);

console.log('Products seeded.');

mongoose.connection.close();

console.log('Database connection closed.');

};

seedDB().catch(err => {

console.error(err);

mongoose.connection.close();

});

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**⭐️ Step 4: The Frontend (Client-Side)**

Now, let's set up the React client.

**1. Initialize React App & Install Dependencies**

Navigate to the client directory and initialize a new React project using Vite (it's fast and modern).

Generated bash

cd client

npm create vite@latest . -- --template react

# When prompted, select 'React' and then 'JavaScript'.

# Now install the other required packages

npm install axios react-router-dom react-hot-toast @heroicons/react

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

**2. Configure Tailwind CSS**

Replace the content of client/tailwind.config.js with this:

Generated javascript

// File: client/tailwind.config.js

/\*\* @type {import('tailwindcss').Config} \*/

export default {

content: [

"./index.html",

"./src/\*\*/\*.{js,ts,jsx,tsx}",

],

theme: {

extend: {},

},

plugins: [

require('@tailwindcss/forms'),

],

}

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

Replace the content of client/src/index.css with this:

Generated css

/\* File: client/src/index.css \*/

@tailwind base;

@tailwind components;

@tailwind utilities;

Use code [with caution](https://support.google.com/legal/answer/13505487).Css

**3. Frontend Code**

Now, create the following files inside client/src/. You can delete any boilerplate files like App.css that Vite created.

**Main Entry Point: client/src/main.jsx**

Generated jsx

// File: client/src/main.jsx

import React from 'react'

import ReactDOM from 'react-dom/client'

import { BrowserRouter } from 'react-router-dom'

import App from './App.jsx'

import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(

<React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</React.StrictMode>,

)

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**Core App Component: client/src/App.jsx**

(This file will be long as it manages state and routing.)

Generated jsx

// File: client/src/App.jsx

import React, { useState, useEffect, createContext } from 'react';

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

import axios from 'axios';

import { Toaster } from 'react-hot-toast';

import Navbar from './components/Navbar';

import HomePage from './pages/HomePage';

import LoginPage from './pages/LoginPage';

import RegisterPage from './pages/RegisterPage';

import CartPage from './pages/CartPage';

import ProtectedRoute from './components/ProtectedRoute';

// Create a context to share user and cart state

export const AppContext = createContext();

// Configure axios for credentials

axios.defaults.baseURL = 'http://localhost:3000';

axios.defaults.withCredentials = true;

function App() {

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

const [cart, setCart] = useState({ items: [], total: 0 });

const [loading, setLoading] = useState(true);

const fetchUserAndCart = async () => {

try {

const userRes = await axios.get('/auth/status');

if (userRes.data.user) {

setUser(userRes.data.user);

const cartRes = await axios.get('/api/cart');

setCart(cartRes.data);

} else {

setUser(null);

setCart({ items: [], total: 0 }); // Reset cart if not logged in

}

} catch (error) {

console.error('Error fetching user/cart status:', error);

} finally {

setLoading(false);

}

};

useEffect(() => {

fetchUserAndCart();

}, []);

if (loading) {

return <div>Loading...</div>; // Or a spinner component

}

return (

<AppContext.Provider value={{ user, setUser, cart, setCart, fetchUserAndCart }}>

<div className="bg-gray-100 min-h-screen">

<Toaster position="top-center" reverseOrder={false} />

<Navbar />

<main className="container mx-auto p-4">

<Routes>

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

<Route path="/login" element={<LoginPage />} />

<Route path="/register" element={<RegisterPage />} />

<Route

path="/cart"

element={

<ProtectedRoute>

<CartPage />

</ProtectedRoute>

}

/>

</Routes>

</main>

</div>

</AppContext.Provider>

);

}

export default App;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**Components: client/src/components/**

Create the components folder: mkdir client/src/components

**Navbar.jsx**

Generated jsx

// File: client/src/components/Navbar.jsx

import React, { useContext } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import axios from 'axios';

import { AppContext } from '../App';

import toast from 'react-hot-toast';

import { ShoppingCartIcon, UserCircleIcon, ArrowRightOnRectangleIcon } from '@heroicons/react/24/outline';

function Navbar() {

const { user, setUser, cart } = useContext(AppContext);

const navigate = useNavigate();

const handleLogout = async () => {

try {

await axios.get('/auth/logout');

setUser(null);

toast.success('Logged out successfully');

navigate('/login');

} catch (error) {

toast.error('Logout failed');

}

};

const cartItemCount = cart.items.reduce((sum, item) => sum + item.quantity, 0);

return (

<nav className="bg-white shadow-md">

<div className="container mx-auto px-4">

<div className="flex justify-between items-center py-4">

<Link to="/" className="text-2xl font-bold text-gray-800">E-Commerce</Link>

<div className="flex items-center space-x-4">

<Link to="/cart" className="relative text-gray-600 hover:text-gray-800">

<ShoppingCartIcon className="h-6 w-6" />

{cartItemCount > 0 && (

<span className="absolute -top-2 -right-2 bg-red-500 text-white text-xs rounded-full h-5 w-5 flex items-center justify-center">

{cartItemCount}

</span>

)}

</Link>

{user ? (

<>

<span className="text-gray-700 flex items-center">

<UserCircleIcon className="h-6 w-6 mr-1" />

{user.displayName || user.email}

</span>

<button onClick={handleLogout} className="text-gray-600 hover:text-gray-800 flex items-center">

<ArrowRightOnRectangleIcon className="h-6 w-6 mr-1" />

Logout

</button>

</>

) : (

<Link to="/login" className="text-gray-600 hover:text-gray-800">Login</Link>

)}

</div>

</div>

</div>

</nav>

);

}

export default Navbar;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**ProductCard.jsx**

Generated jsx

// File: client/src/components/ProductCard.jsx

import React, { useContext } from 'react';

import axios from 'axios';

import toast from 'react-hot-toast';

import { AppContext } from '../App';

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

function ProductCard({ product }) {

const { user, setCart } = useContext(AppContext);

const navigate = useNavigate();

const handleAddToCart = async () => {

if (!user) {

toast.error('Please log in to add items to the cart.');

navigate('/login');

return;

}

try {

const res = await axios.post('/api/cart/add', { product, quantity: 1 });

setCart(res.data);

toast.success(`${product.title} added to cart!`);

} catch (error) {

toast.error('Failed to add to cart.');

}

};

return (

<div className="bg-white rounded-lg shadow-md overflow-hidden transform hover:scale-105 transition-transform duration-300">

<img src={product.image} alt={product.title} className="w-full h-48 object-cover" />

<div className="p-4">

<h3 className="text-lg font-semibold">{product.title}</h3>

<p className="text-gray-600 mt-2 text-sm">{product.description}</p>

<div className="flex justify-between items-center mt-4">

<span className="text-xl font-bold text-gray-800">${product.price.toFixed(2)}</span>

<button

onClick={handleAddToCart}

className="bg-blue-500 text-white px-4 py-2 rounded-lg hover:bg-blue-600 transition-colors"

>

Add to Cart

</button>

</div>

</div>

</div>

);

}

export default ProductCard;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**ProtectedRoute.jsx**

Generated jsx

// File: client/src/components/ProtectedRoute.jsx

import React, { useContext } from 'react';

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

import { AppContext } from '../App';

const ProtectedRoute = ({ children }) => {

const { user } = useContext(AppContext);

if (!user) {

// Redirect them to the /login page, but save the current location they were

// trying to go to. This is optional but a good user experience.

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

}

return children;

};

export default ProtectedRoute;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**Pages: client/src/pages/**

Create the pages folder: mkdir client/src/pages

**HomePage.jsx**

Generated jsx

// File: client/src/pages/HomePage.jsx

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import ProductCard from '../components/ProductCard';

function HomePage() {

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

const [loading, setLoading] = useState(true);

useEffect(() => {

const fetchProducts = async () => {

try {

const res = await axios.get('/api/products');

setProducts(res.data);

} catch (error) {

console.error("Failed to fetch products", error);

} finally {

setLoading(false);

}

};

fetchProducts();

}, []);

if (loading) {

return <div className="text-center mt-10">Loading products...</div>;

}

return (

<div>

<h1 className="text-3xl font-bold mb-6">Our Products</h1>

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">

{products.map(product => (

<ProductCard key={product.\_id} product={product} />

))}

</div>

</div>

);

}

export default HomePage;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**LoginPage.jsx**

Generated jsx

// File: client/src/pages/LoginPage.jsx

import React, { useState, useContext } from 'react';

import { useNavigate, Link } from 'react-router-dom';

import axios from 'axios';

import toast from 'react-hot-toast';

import { AppContext } from '../App';

function LoginPage() {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const { fetchUserAndCart } = useContext(AppContext);

const navigate = useNavigate();

const handleLogin = async (e) => {

e.preventDefault();

try {

await axios.post('/auth/login', { email, password });

toast.success('Login successful!');

await fetchUserAndCart(); // Refetch user and cart state

navigate('/');

} catch (error) {

toast.error(error.response?.data?.message || 'Login failed');

}

};

const handleGoogleLogin = () => {

window.location.href = 'http://localhost:3000/auth/google';

};

return (

<div className="max-w-md mx-auto mt-10">

<div className="bg-white p-8 rounded-lg shadow-md">

<h2 className="text-2xl font-bold text-center mb-6">Login</h2>

<form onSubmit={handleLogin}>

<div className="mb-4">

<label className="block text-gray-700">Email</label>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} className="w-full px-3 py-2 border rounded-lg" required />

</div>

<div className="mb-6">

<label className="block text-gray-700">Password</label>

<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} className="w-full px-3 py-2 border rounded-lg" required />

</div>

<button type="submit" className="w-full bg-blue-500 text-white py-2 rounded-lg hover:bg-blue-600">Login</button>

</form>

<div className="text-center my-4">OR</div>

<button onClick={handleGoogleLogin} className="w-full bg-red-500 text-white py-2 rounded-lg hover:bg-red-600">

Login with Google

</button>

<p className="text-center mt-4">

Don't have an account? <Link to="/register" className="text-blue-500 hover:underline">Register here</Link>

</p>

</div>

</div>

);

}

export default LoginPage;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**RegisterPage.jsx**

Generated jsx

// File: client/src/pages/RegisterPage.jsx

import React, { useState, useContext } from 'react';

import { useNavigate, Link } from 'react-router-dom';

import axios from 'axios';

import toast from 'react-hot-toast';

import { AppContext } from '../App';

function RegisterPage() {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [confirmPassword, setConfirmPassword] = useState('');

const { fetchUserAndCart } = useContext(AppContext);

const navigate = useNavigate();

const handleRegister = async (e) => {

e.preventDefault();

if (password !== confirmPassword) {

toast.error("Passwords do not match!");

return;

}

try {

await axios.post('/auth/register', { email, password });

toast.success('Registration successful!');

await fetchUserAndCart();

navigate('/');

} catch (error) {

toast.error(error.response?.data?.message || 'Registration failed');

}

};

return (

<div className="max-w-md mx-auto mt-10">

<div className="bg-white p-8 rounded-lg shadow-md">

<h2 className="text-2xl font-bold text-center mb-6">Register</h2>

<form onSubmit={handleRegister}>

<div className="mb-4">

<label className="block text-gray-700">Email</label>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} className="w-full px-3 py-2 border rounded-lg" required />

</div>

<div className="mb-4">

<label className="block text-gray-700">Password</label>

<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} className="w-full px-3 py-2 border rounded-lg" required />

</div>

<div className="mb-6">

<label className="block text-gray-700">Confirm Password</label>

<input type="password" value={confirmPassword} onChange={(e) => setConfirmPassword(e.target.value)} className="w-full px-3 py-2 border rounded-lg" required />

</div>

<button type="submit" className="w-full bg-blue-500 text-white py-2 rounded-lg hover:bg-blue-600">Register</button>

</form>

<p className="text-center mt-4">

Already have an account? <Link to="/login" className="text-blue-500 hover:underline">Login here</Link>

</p>

</div>

</div>

);

}

export default RegisterPage;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**CartPage.jsx**

Generated jsx

// File: client/src/pages/CartPage.jsx

import React, { useContext } from 'react';

import axios from 'axios';

import toast from 'react-hot-toast';

import { AppContext } from '../App';

import { TrashIcon } from '@heroicons/react/24/outline';

function CartPage() {

const { cart, setCart } = useContext(AppContext);

const handleUpdateQuantity = async (productId, quantity) => {

try {

const res = await axios.put(`/api/cart/update/${productId}`, { quantity });

setCart(res.data);

toast.success('Cart updated!');

} catch (error) {

toast.error('Failed to update cart.');

}

};

const handleRemoveItem = async (productId) => {

try {

const res = await axios.delete(`/api/cart/remove/${productId}`);

setCart(res.data);

toast.success('Item removed from cart!');

} catch (error) {

toast.error('Failed to remove item.');

}

};

if (cart.items.length === 0) {

return <div className="text-center text-2xl mt-10">Your cart is empty.</div>;

}

return (

<div className="max-w-4xl mx-auto bg-white p-6 rounded-lg shadow-md">

<h1 className="text-3xl font-bold mb-6">Your Shopping Cart</h1>

<div className="space-y-4">

{cart.items.map(item => (

<div key={item.product.\_id} className="flex items-center justify-between border-b pb-4">

<div className="flex items-center">

<img src={item.product.image} alt={item.product.title} className="w-20 h-20 object-cover rounded-lg mr-4" />

<div>

<h2 className="font-semibold">{item.product.title}</h2>

<p className="text-gray-600">${item.product.price.toFixed(2)}</p>

</div>

</div>

<div className="flex items-center space-x-4">

<input

type="number"

value={item.quantity}

min="1"

onChange={(e) => handleUpdateQuantity(item.product.\_id, parseInt(e.target.value))}

className="w-16 text-center border rounded"

/>

<button onClick={() => handleRemoveItem(item.product.\_id)} className="text-red-500 hover:text-red-700">

<TrashIcon className="h-6 w-6" />

</button>

</div>

</div>

))}

</div>

<div className="mt-6 text-right">

<h2 className="text-2xl font-bold">Total: ${cart.total.toFixed(2)}</h2>

<button className="mt-4 bg-green-500 text-white px-6 py-2 rounded-lg hover:bg-green-600">

Proceed to Checkout

</button>

</div>

</div>

);

}

export default CartPage;

Use code [with caution](https://support.google.com/legal/answer/13505487).Jsx

**⭐️ Step 5: Root Project Files**

Go back to the root ecommerce-app directory.

**1. Root package.json**

This file will contain scripts to run everything.

Generated json

// File: package.json (in the root directory)

{

"name": "ecommerce-app",

"version": "1.0.0",

"description": "Full-stack e-commerce application",

"scripts": {

"install": "npm install --prefix server && npm install --prefix client",

"dev": "concurrently \"npm run dev --prefix server\" \"npm run dev --prefix client\""

},

"keywords": [],

"author": "Amjad Manzoor",

"license": "ISC",

"devDependencies": {

"concurrently": "^8.2.2"

}

}

Use code [with caution](https://support.google.com/legal/answer/13505487).Json

**2. .env.example file**

Create .env.example in the root.

Generated env

# File: .env.example

# MongoDB Connection String

MONGODB\_URI=your\_mongodb\_connection\_string

# Express Session Secret

SESSION\_SECRET=a\_very\_strong\_and\_long\_secret\_key

# Google OAuth 2.0 Credentials

# Get these from the Google Cloud Console: https://console.cloud.google.com/

GOOGLE\_CLIENT\_ID=your\_google\_client\_id

GOOGLE\_CLIENT\_SECRET=your\_google\_client\_secret

GOOGLE\_CALLBACK=http://localhost:3000/auth/google/callback

Use code [with caution](https://support.google.com/legal/answer/13505487).Env

**3. .gitignore file**

Create .gitignore in the root. This is very important.

Generated code

# File: .gitignore

# Dependencies

node\_modules/

client/node\_modules/

client/dist/

# Environment variables

.env

# Logs

npm-debug.log\*

yarn-debug.log\*

yarn-error.log\*

Use code [with caution](https://support.google.com/legal/answer/13505487).

**✅ Final Steps: How to Run Your Application**

1. **Google Credentials**: Go to the [Google Cloud Console](https://www.google.com/url?sa=E&q=https%3A%2F%2Fconsole.cloud.google.com%2F), create a new project, go to "APIs & Services" -> "Credentials". Create an "OAuth client ID" for a "Web application".
   * Add http://localhost:3000 to "Authorized JavaScript origins".
   * Add http://localhost:3000/auth/google/callback to "Authorized redirect URIs".
   * Copy the Client ID and Client Secret.
2. **Create .env File**: In the root directory, create a .env file (copy from .env.example) and fill in your MONGODB\_URI, SESSION\_SECRET, GOOGLE\_CLIENT\_ID, and GOOGLE\_CLIENT\_SECRET.   
   Client ID: 67827108861-6jguvt467i81i9ir05si92k25cabifg7.apps.googleusercontent.com

Client secret: GOCSPX-63Ru58U6jer7T\_OlPvaTAluZzs82  
  
(Steps are mentioned at the bottom how to create and get mongodb\_uri and session\_Secret).

1. **Install Dependencies**: From the root ecommerce-app directory, run:

Generated bash

npm install

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

This script will install dependencies for both the server and the client.

1. **Seed the Database**: Make sure your MongoDB server is running and your .env file is correct. Then run:

Generated bash

npm run seed

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Run in Development Mode**:

Generated bash

npm run dev

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

This will start both the backend server (on port 3000) and the frontend Vite server (on port 5173). Open your browser to http://localhost:5173 to see your app

You're at a key configuration step that is crucial for any full-stack application. Let's break down what these two variables are and how to get the values for them.

**1. MONGODB\_URI (MongoDB Connection String)**

This is like the address and password for your application to connect to its database. It tells your server where the MongoDB database is hosted and how to authenticate with it.

You have two main options for this:

**Option A: Cloud Database (Recommended for Beginners & Deployment)**

We'll use **MongoDB Atlas**, which offers a generous free tier. This is the best way to get started.

**Step-by-Step Guide to get your MONGODB\_URI from Atlas:**

**🔹 Step 1: Create a Free MongoDB Atlas Account**

1. **Go to:** [**https://www.mongodb.com/cloud/atlas**](https://www.mongodb.com/cloud/atlas) **Click "Start Free" and sign up.**

**🔹 Step 2: Create a Cluster**

1. **Click “Build a Database”**
2. **Choose:**
3. **Shared Cluster (Free Tier)**
4. **Cloud provider: AWS or GCP**
5. **Region: Choose closest to your location**
6. **Click Create**

**🔹 Step 3: Create a Database & Collection**

1. **Go to your cluster → Click “Browse Collections”**
2. **Click “Add My Own Data”**
3. **Enter:**
4. **Database Name: ecommerce**
5. **Collection Name: products**

**🔹 Step 4: Create a Database User**

1. **Create a Database User:**
   * Once your cluster is created, you need to create a user for your application to connect as.
   * In the left-hand menu, go to **Database Access**.
   * Click **"Add New Database User"**.
   * Enter a **Username** (e.g., ecommerce\_user).
   * Enter a **Password**. Use the "Autogenerate Secure Password" button and **copy this password somewhere safe immediately**. You will need it for the connection string.
   * Under "Database User Privileges", select **"Read and write to any database"**.
   * Click **"Add User"**.
2. **Configure Network Access:**
   * You need to tell Atlas which IP addresses are allowed to connect to your database.
   * In the left-hand menu, go to **Network Access**.
   * Click **"Add IP Address"**.
   * For development, the easiest option is to click **"ALLOW ACCESS FROM ANYWHERE"**. This will enter 0.0.0.0/0 as the IP address.
   * Add a description (e.g., "Allow all for dev") and click **"Confirm"**.

**Note:** For a real production app, you would restrict this to only your server's IP address for better security.

1. **Get the Connection String:**
   * Go back to **Database** in the left menu.
   * Click the **"Connect"** button on your cluster.
   * In the pop-up window, select **"Drivers"**.
   * You will see a connection string. It will look like this:

Generated code

mongodb+srv://<username>:<password>@cluster0.xxxxx.mongodb.net/?retryWrites=true&w=majority

Use code [with caution](https://support.google.com/legal/answer/13505487).

* + **Copy this string.** This is your MONGODB\_URI!

1. **Finalize the String:**
   * Paste the copied string into your .env file.
   * **Crucially, replace <password> with the actual password you saved in Step 4.** Do not include the < and > brackets.

Your final URI in the .env file will look like this:

Generated env

MONGODB\_URI=mongodb+srv://ecommerce\_user:YourActualSecurePassword@cluster0.xxxxx.mongodb.net/?retryWrites=true&w=majority

Use code [with caution](https://support.google.com/legal/answer/13505487).Env

**Option B: Local Database (For advanced users)**

If you have MongoDB Community Server installed and running on your own machine, your connection string is much simpler:

Generated env

MONGODB\_URI=mongodb://localhost:27017/ecommerce-app

Use code [with caution](https://support.google.com/legal/answer/13505487).Env

Here, ecommerce-app will be the name of the database created on your local machine.

**2. SESSION\_SECRET (Express Session Secret)**

This is a random, secret string of characters that your application uses to "sign" the session ID cookie sent to the user's browser. This signature prevents the cookie from being tampered with.

**Why it must be secret and random:** If someone could guess your secret, they could create fake session cookies and potentially log in as other users.

**How to Generate a Good SESSION\_SECRET:**

* **Do not use** simple, guessable strings like "password", "secret", or "12345".
* It should be long and completely random.

**⭐️ Easiest Method (Using Node.js in your terminal):**

Open your terminal (it doesn't matter which directory you're in) and run this one-line Node.js command:

Generated bash

node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

This will output a long, random, and secure string. For example:  
e4c4e78f9b1f3c3a4d8e9f0a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c

**Copy the output** and paste it as the value for SESSION\_SECRET.

**✅ Your Final .env File**

After following these steps, your .env file should look something like this:

Generated env

# File: .env

# MongoDB Connection String from MongoDB Atlas

MONGODB\_URI=mongodb+srv://ecommerce\_user:YourActualSecurePassword@cluster0.xxxxx.mongodb.net/?retryWrites=true&w=majority

# Express Session Secret generated from the crypto command

SESSION\_SECRET=e4c4e78f9b1f3c3a4d8e9f0a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c

# Google OAuth 2.0 Credentials (you'll get these from Google Cloud Console)

GOOGLE\_CLIENT\_ID=your\_google\_client\_id

GOOGLE\_CLIENT\_SECRET=your\_google\_client\_secret

GOOGLE\_CALLBACK=http://localhost:3000/auth/google/callback

Use code [with caution](https://support.google.com/legal/answer/13505487).Env

Remember to **never** commit your .env file to a public repository like GitHub. The .gitignore file I provided already prevents this.