

React, Redux, MERN Stack Developer Documentations

Reference: <https://github.com/bradtraversy/proshop-v2>

#1: Tools pre-requisite

- Application required:
 - VSCode,
 - git, gitbash,
 - Nodejs,
 - Postman for API
- Chrome extension:
 - React Developer tools and
 - Redux DevTools
- VSCode Extension:
 - ES7-react/Redux/React-Native snippets,
 - prettier-code formatter,
 - JavaScript (ES6) code snippets,
 - Github Copilot,
 - Github Theme

#2: Starting with Frontend:

React setup and Git-Initialize

```
>> mkdir shopify
>> cd shopify
>> shopify >> npx create-react-app frontend
>> cd frontend
>> npm start // to start the server
```

Delete some files like not required ex. App.css, app.test, logo.svg. is not required.

Then Shopify folder: 

- Frontend: React ui and All frontend dependencies (redux, bootstrap etc)
- Backend: Server-side code modules, models, controller and routes
- Uploads: Images uploads
- Nodemodules: server dependencies (express, mongoose)
- Packages.json: server package.json
- .env: environment variable – mongoDB URI, JWT secret etc.
- .git:

First do remove .git file from folder which is hidden first d stop server

- >> ls -a // to list all folder with hidden folder/files
- >> rm -rf .git // to remove .git folder

- Or even delete manually

- And then .gitignore folder move to root dir and do some change inside file which we need to ignore while git and version control ex: nodemodules and .env files should be ignore

My github Repo: <https://github.com/AniketKatre/shopifyMERN>

Steps to add in github:

- >> git init // inside root folder
- >> git add . // upload all folder except nodemodules folder and env
- >> git commit -m "Initial commit"
- >> git remote add origin <https://github.com/AniketKatre/shopifyMERN.git> // my repo
- >> git branch -M main
- >> git push -u origin main

Then check the repo again do Refresh: <https://github.com/AniketKatre/shopifyMERN>

#3: React bootstrap for header and footer:

>> frontend >> npm i react-bootstrap bootstrap react-icons

Add index.js line of code:

```
>> import 'bootstrap/dist/css/bootstrap.min.css';
```

Then, make folder inside src folder src/components - inside this all screen jsx file will create

Header.jsx file 📁:

Footer.jsx file 📁:

Product.jsx file 📁: - child: HomeScreen.jsx 📁: --get data from product.jsx

#4: React – Router DOM:

>> npm i react-router-dom

In index.js

```
import { BrowserRouter, createRoutesFromElements, Route, RouterProvider } from
'react-router-dom';
```

```
const router = createBrowserRouter(
  createRoutesFromElements(
    <Route path="/" element={<App />}>
      <Route index={true} path="/" element={<HomeScreen />} />
    </Route>
  )
)
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

```
);
```

As Reactjs is single page applicaon so, we don't want to repload pages – here we used react router dom with Link method:

In Product.jsx

```
import { Link } from "react-router-dom";
anchor tag <a href="/.." > here we used as following Link methods and to=/ ..... same changes we will do for all link navbar or footer
```

```
<Link to={`/${product._id}`}>
  <Card.Img src={product.image} variant="top" />
</Link>
```

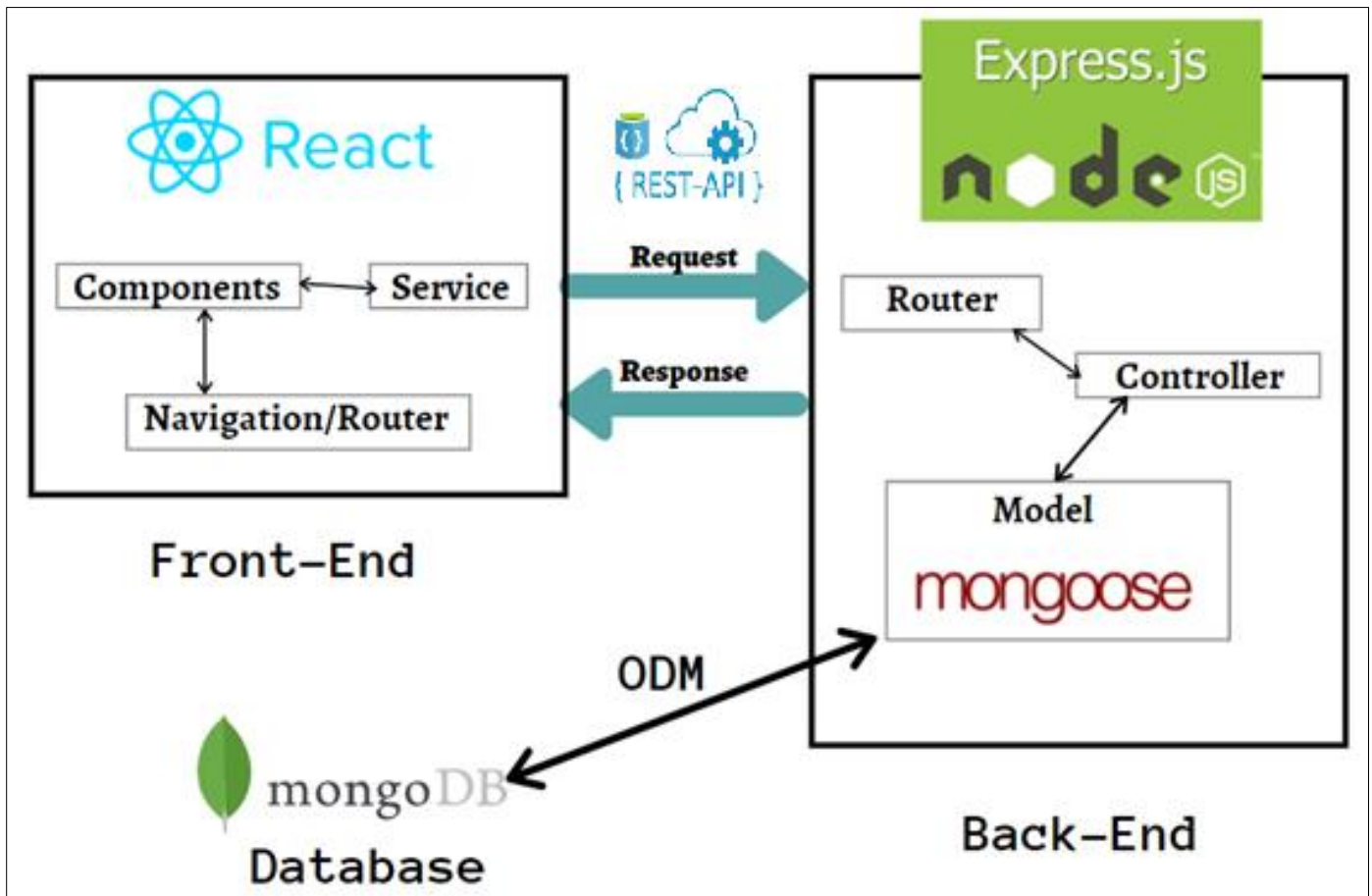
Same we don't want reload on applies for Navbar Header.jsx

>> npm i react-router-bootstrap

```
import { LinkContainer } from "react-router-bootstrap";
LinkContainer wrap to the component to Nav.Link
```

```
<LinkContainer to="/">
  <Navbar.Brand>
    <img src={logo} alt="shopify" />
    Shopify
  </Navbar.Brand>
</LinkContainer>
```

#5: Full Stack Workflow:



#6: Backend with Nodejs:

>> root >> npm init

Package.json file need some modification

```
{
  "name": "shopify",
  "version": "2.0.0",
  "description": "eCommerce application built with the MERN full Stack development",
  "type": "module",
  "main": "server.js",
  "scripts": {
    "start": "node backend/server.js"
  },
  "author": "annie_jb",
  "license": "MIT"
}
```

Then, Create folder Backend 📁: inside server.js file

>> npm i express

Express web framework we'll be using on the backend to create our routes and so on.....

mailto: katreaniket3@gmail.com | [linkedin.com/in/aniket-katre-752465149/](https://www.linkedin.com/in/aniket-katre-752465149/) | IG: [@annie_jb](https://www.instagram.com/annie_jb) | [github/AniketKatre/](https://github.com/AniketKatre/)

```
// default common js type : below method commonly used in js
// const express = require('express')
// type= module ES6 and used also in overall project
import express from 'express';
```

Now make backend server:

```
import express from 'express';

const port = 5000;
const app = express();

app.get('/', (req, res) => {
  res.send('API is running')
})

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

Now, Nodemon and Concurrently

```
>> root >> npm i -D nodemon concurrently
```

Add 4 line of code in package.json in root dir to run both client and server

```
"start": "node backend/server.js",
"server": "nodemon backend/server.js",
"client": "npm start --prefix frontend",
"dev": "concurrently \"npm run server\" \"npm run client\" "
```

#7: Environment Variable:

.env file is used to store credential personalize data for auth, port and pwd tokens

```
>> npm i -D dotenv
```

#8: Fetch Products from backend to frontend:

We gonna use axios to fetch data, even we can use fetch API, We used axios here, bit easier.

```
>> cd frontend // we gonna add frontend dependencies
```

```
>> npm install axios
```

Now, go to HomeScreen.jsx and we gonna fetch data from backend not product.js JSON file. So, since this is a React component, obviously we're not dealing with Next.js or anything like that, so we're using Use Effect hook to fetch out data.

```
import { useEffect, useState } from "react";
import axios from "axios";
```

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

useEffect(() => {
  const fetchProducts = async () => {
    const { data } = await axios.get("/api/products");
    setProducts(data);
  };
  fetchProducts();
}, []);
```

Similarly we can do for all..... productscreen desc....

REDUX: We'll fetch everything from redux and pass it down into the components that needs it.

#8: MongoDB Atlas setup:

Create account on mongoDB atlas:

<https://cloud.mongodb.com/v2/65f09e612967603f954bf61f#/metrics/replicaSet/65f09fc88298c56785b14acd/explorer/shopify>

user: katreaniket3@gmail.com

pass: jb 90 12345@ ---- combine this you know better

Create Project: Shopify

Username

katreaniket3

Password

Create DB inside TAB Collections: DB name=shopify and Collections name: products >>> CREATE



INSTALL Mongo DB Compass Tool GUI:

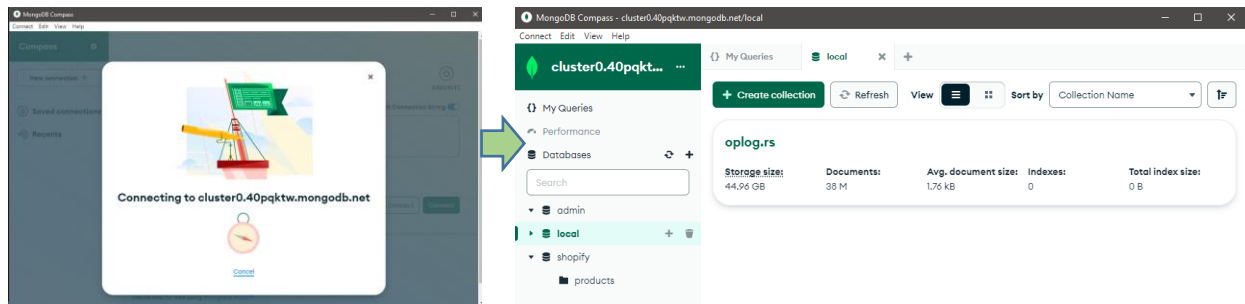
<https://www.mongodb.com/try/download/shell>

Open Compass application mongo DB GUI: and put connection url for connect compass [DOWNLOAD](#)

mongodb+srv://katreaniket3:<password>@cluster0.40pqktw.mongodb.net/shopify

If you received an error while connecting add public IP address:

IP Access List	Description	
0.0.0.0/0	public open for all	<div>  EDIT  REMOVE </div>



#9: Connect with mongoose and VSCode:

VSCODE: npm i mongoose

Backend >> config >> db.js

```
import mongoose from "mongoose";

const connectDB = async() =>{
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB Connected: ${conn.connection.host}`)
  } catch (error) {
    console.log(`Error: ${error.message}`);
    process.exit(1)
  }
}

export default connectDB;
```

and server.js

```
import connectDB from './config/db.js';
connectDB(); // Connect to MongoDB
```

Now, Modeling our Data:

Here we create schema level data and export it into mongoDB for database to implement.

Backend >> models >> productModel.js

EX:

```
import mongoose from "mongoose";
const productSchema = new mongoose.Schema({
  name: {type:String, required: true},
  image:{}.....
},{ timestamps: true });

const Product = mongoose.model("Product", productSchema)
export default Product;
```

Similarly, for UserModel and OrderModel Schema need to create.

NEXT,

Now, we need to add this all schema and few sample data in Database (mongoDB).

Import sample data into mongoDB make some change in product.js file.

Backend >> data >> products.js file and user.js file

For users data we need password should be encrypted so, we used nodejs package called [bcryptjs](#)

```
>> npm i bcryptjs
```

Users.js

```
import bcrypt from 'bcryptjs';
const users = [
  {
    name: 'Admin User',
    email: 'admin@gmail.com',
    password: bcrypt.hashSync('12345', 10),
    isAdmin: true,
  },
  {
    name: 'Annie',
    email: 'annie@gmail.com',
    password: bcrypt.hashSync('12345', 10),
    isAdmin: false,
  },
  {
    name: 'Stark',
    email: 'stark@gmail.com',
    password: bcrypt.hashSync('12345', 10),
    isAdmin: false,
  }
];
export default users;
```

NEXT STEPS is to add this data into DB by using sample script,

Now, Seeding data into mongoDB

```
>> npm install colors // optional : main moto is to add colors in terminal
```

Seeding data to our DB

```
>> backend >> seeder.js // only used for seeding data once then no more used
```

And Then, create script for import and delete data in package.json

```
>> root >> package.json
```

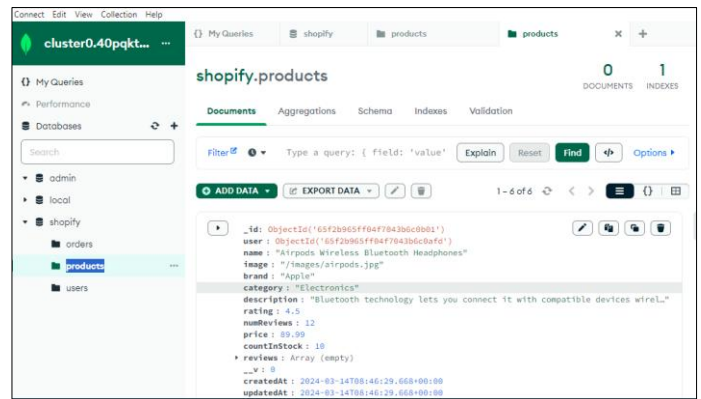
```
"data:import" : "node backend/seeder.js",
"data:destroy" : "node backend/seeder.js -d"
```

And then run above both script. And data imported.


```
PS C:\1_MY DATA\Project\shopify> npm run data:import
> shopify@2.0.0 data:import
> node backend/seeder.js

MongoDB Connected: ac-enzitso-shard-00-00.40pqltw.mongodb.net
Data Imported!
PS C:\1_MY DATA\Project\shopify> npm run data:destroy
> shopify@2.0.0 data:destroy
> node backend/seeder.js -d

MongoDB Connected: ac-enzitso-shard-00-02.40pqltw.mongodb.net
Data Destroyed!
PS C:\1_MY DATA\Project\shopify> npm run data:import
```



Postman API: Build your own api collections.

#10: Get Products from database:

Create routes to get data from mongoDB:

>> backend >> routes >> productRoutes.js

In this above file you need to add express routers to get data from DB. [ref](#)

```
import express from "express";
const router = express.Router();
import asyncHandler from "../middleware/asyncHandler.js";
import Product from "../models/productModel.js";
router.get('/', asyncHandler( async (req, res) =>{
  const products = await Product.find({});
  res.json(products);
}));
router.get('/:id', asyncHandler ( async (req, res) =>{
  // const product = products.find((p) => p._id===req.params.id);
  const product = await Product.findById(req.params.id);
  if (product){
    return res.json(product);
  }
  res.status(404).json({message: 'Product not found'});
}));
export default router;
```

and then server.js previously we fetch data from product.js file hardcoded, now comment out this and put productRouter.js import

server.js

```
import productRoutes from './routes/productRoutes.js'
app.use('/api/products', productRoutes);
```

And to fetch data from mongoose we need wait for fetch here we used [asyncHandler express methods](#) but we have added manual cause only 3 line of below code:

>> backend >> middleware >> asyncHandler.js

```
const asyncHandler = fn => (req, res, next) => {
  Promise.resolve(fn(req, res, next)).catch(next);
}
export default asyncHandler;
```

NEXT Custom error handler Middleware:

[Documentation ref:](#)

To handler **custom** error handler middleware:

>> backend >> middleware >> errorMiddleware.js

```
const notFound = (req, res, next) => {
  const error = new Error(`Not Found - ${req.originalUrl}`);
  res.status(404);
  next(error);
}

const errorHandler = (err, req, res, next) => {
  let statusCode = res.statusCode === 200 ? 500 : res.statusCode;
  let message = err.message;
  // Check for mongoose bad ObjectId
  if(err.name === 'CastError' && err.kind === 'ObjectId'){
    message = `Resource not found`;
    statusCode = 404;
  }
  res.status(statusCode).json({
    message,
    stack: process.env.NODE_ENV === 'production' ? '🍰' : err.stack,
  });
};
export { notFound, errorHandler };
```

And add in server.js file to handle error

```
import { notFound, errorHandler } from './middleware/errorMiddleware.js'
app.use(notFound);
app.use(errorHandler);
```

#11: Product Controller:

Express is a very Unopinionated minimalistic framework.

For Best practice in code: it's better to create a controller for each route and use that controller function. Its help you to keep things organize.

>> backend >> controller >> productController.js

```
import asyncHandler from "../middleware/asyncHandler.js";
import Product from "../models/productModel.js";
// @desc    Fetch all products
// @route    GET /api/products
// @access   Public
const getProducts = asyncHandler (async(req, res) => {
  const products = await Product.find({});
  res.json(products);
});
// @desc    Fetch a product by ids
// @route    GET /api/products/:id
// @access   Public
const getProductById = asyncHandler(async(req, res) =>{
  const product = await Product.findById(req.params.id);
  if (product){
    return res.json(product);
  } else{
    res.status(404);
    throw new Error('Product not found');
  }
});
export {getProducts, getProductById};
```

and in routes also need to change and implement controller to clean our code for best practice:

>> backend >> routes >> productRoutes.js

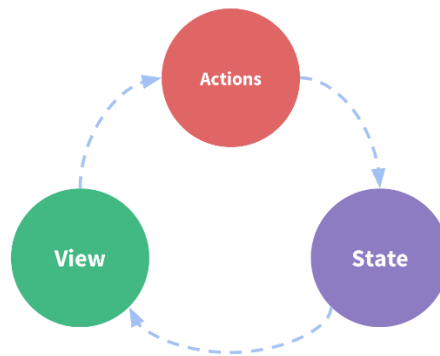
```
import { getProducts, getProductById } from "../controllers/productController.js";

router.route('/').get(getProducts);
router.route('/:id').get(getProductById);
```

#12: Redux Toolkit setup and State Management:



Workflow:



Redux is a popular JavaScript library for managing application state, and it's often used with React, but it can be used with other UI libraries/frameworks, and there's many other state management libraries available, **but** Redux is one of the most popular and well supported.

State: state which is state that is only it only matters in that actual component.

State Management: refers to the way that we manage the data that our applications need to keep track.

Install Redux-toolkit and react-redux:

```
>> frontend >> terminal - - npm i @reduxjs/toolkit react-redux
```

And create store.js = this is entry point redux

```
>> frontend >> src >> store.js
```

```
import { configureStore } from "@reduxjs/toolkit";

const store = configureStore({
  reducer: {},
});

export default store;
```

and change in index.js

```
>> frontend >> src >> index.js
```

```
import { Provider } from "react-redux";
import store from "./store";
```

```
<Provider store={store}>
  <RouterProvider router={router} />
</Provider>
```

Now, create constants file for adding constant URL of api in frontend

```
>> frontend >> src >> constants.js
```

```
export const BASE_URL =
  process.env.NODE_ENV === "development" ? "http://localhost:5000" : "";
```

```
export const PRODUCT_URL = "/api/products";
export const USERS_URL = "/api/users";
export const ORDERS_URL = "/api/orders";
export const PAYPAL_URL = "/api/config/paypal";
```

AND then stores.js:

Slices: Slices is the concepts in Redux toolkit called slices, and its way to organize your state. So it's collection of reducers and actions that are related to each other. We can create multiple slices in or applications and ach slices can have its own state.

>> frontend >> src >> stores.js

```
import { configureStore } from "@reduxjs/toolkit";
import { apiSlice } from "../slices/apiSlice";

const store = configureStore({
  reducer: {
    [apiSlice.reducerPath]: apiSlice.reducer,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(apiSlice.middleware),
  devTools: true,
});

export default store;
```

AND

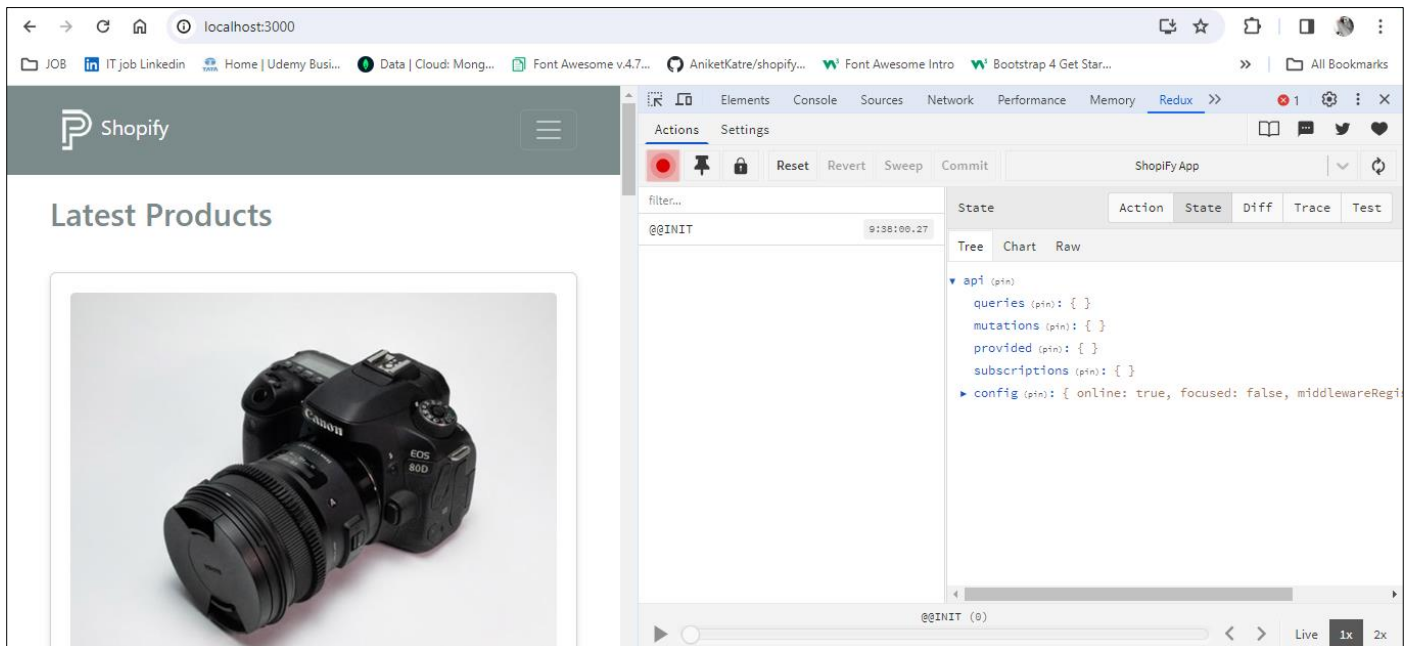
>> frontend >> src >> slices >> apiSlice.js

```
import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";
import { BASE_URL } from "../constants";

const baseQuery = fetchBaseQuery({ baseUrl: BASE_URL });

export const apiSlice = createApi({
  baseQuery,
  tagTypes: ["Product", "Order", "User"],
  endpoints: (builder) => ({}),
});
```

Now un the server and check in redux tools



Now Create ProductApiSlice.js file to get data.....

NOTE: Previously we fetch data from backend using axios and now we fetch data from redux apiSlice.

>>> frontend >> src >> productApiSlice.js

```
import { PRODUCT_URL } from "../constants";
import { apiSlice } from "../apiSlice";

export const productsApiSlice = apiSlice.injectEndpoints({
  endpoints: (builder) => ({
    getProducts: builder.query({
      query: () => ({
        url: PRODUCT_URL,
      }),
      keepUnusedDataFor: 5,
    }),
    getProductDetails: builder.query({
      query: (productId) => ({
        url: `${PRODUCT_URL}/${productId}`,
      }),
      keepUnusedDataFor: 5,
    }),
  }),
});

export const { useGetProductsQuery, useGetProductDetailsQuery } =
  productsApiSlice;
```

Then, remove axios code from HomeScreen and ProductScreen.jsx and fetch data from API slices...

And for next in future in same project will used same.

```
>> frontend >> src >> screens >> HomeScreen.jsx
```

Remove all axios code and useState and useEffect.

```
import { useGetProductsQuery } from "../slices/productsApiSlice";
```

```
const HomeScreen = () => {
  const { data: products, isLoading, error } = useGetProductsQuery();
```

```
  return (
    <>
      {isLoading ? (
        <Loader />
      ) : error ? (
        <Message variant="danger">
          {error?.data?.message || error.error}
        </Message>
      ) : ( <h2> webpage</h2>)} </> );
```

Add few components like Message.jsx, and Loader.jsx

Github commit:

<https://github.com/AniketKatre/shopifyMERN/commit/4b84ba3af0cf407143dad08dbb067a7ad13b42e2>

#13: Shopping Cart Functionality:

Shopping cart functionality implement needs to create new slices i.e., cartSlice.js

```
>> frontend >> src >> slices >> cartSlices.js
```

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = localStorage.getItem("cart")
  ? JSON.parse(localStorage.getItem("cart"))
  : { cartItems: [] };

const cartSlice = createSlice({
  name: "cart",
  initialState,
  reducers: {},
});

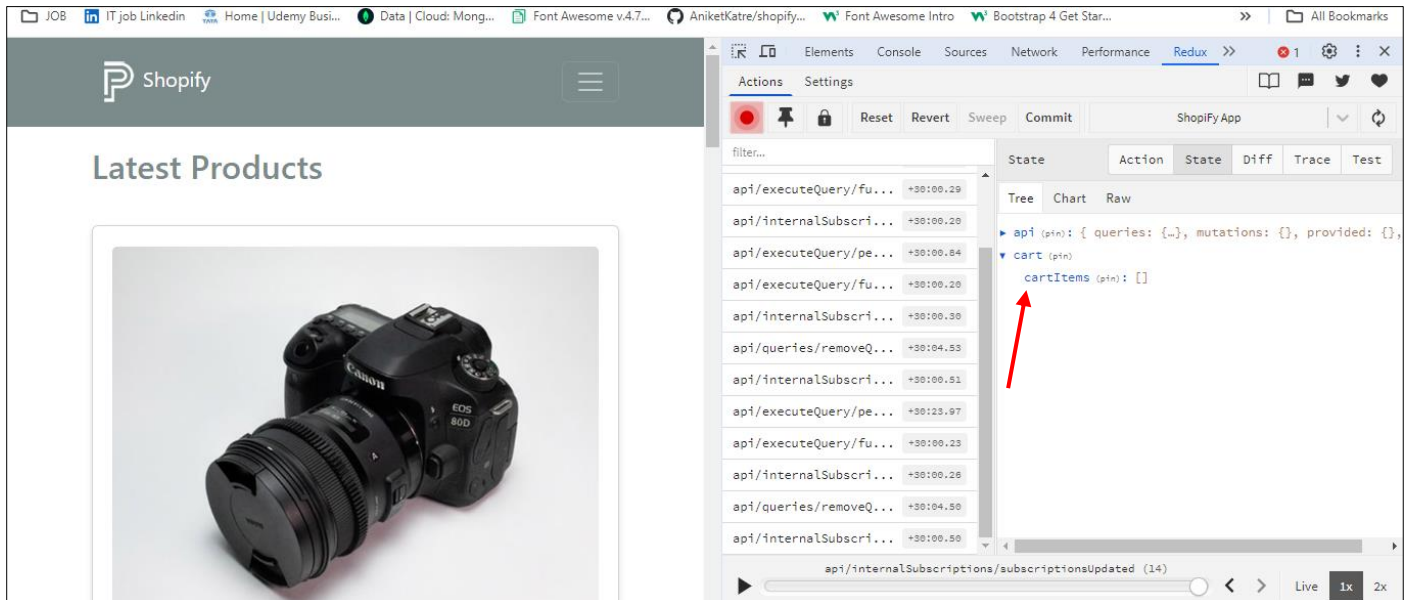
export default cartSlice.reducer;
```

and store.js add cartReducer

```
import cartSliceReducer from "../slices/cartSlice";
```

```
cart: cartSliceReducer,
```

REDUX:



Add to Cart functionality:

We're going to add the reducer function to add an item to the cart.

Github commit:

<https://github.com/AniketKatre/shopifyMERN/commit/a0abecca063a2ed7ab93c21e94a00a16b84bda8a>

#14: Backend Authentications:

User Routes and Controller

Create userController and userRoutes

```
>> backend >> controllers >> userController.js
```

```
>> backend >> routes >> userRoutes.js
```

And implement in sever.js

User Email and Password validations:

```
>> backend >> models >> userModel.js    /// implement bcrypt
```

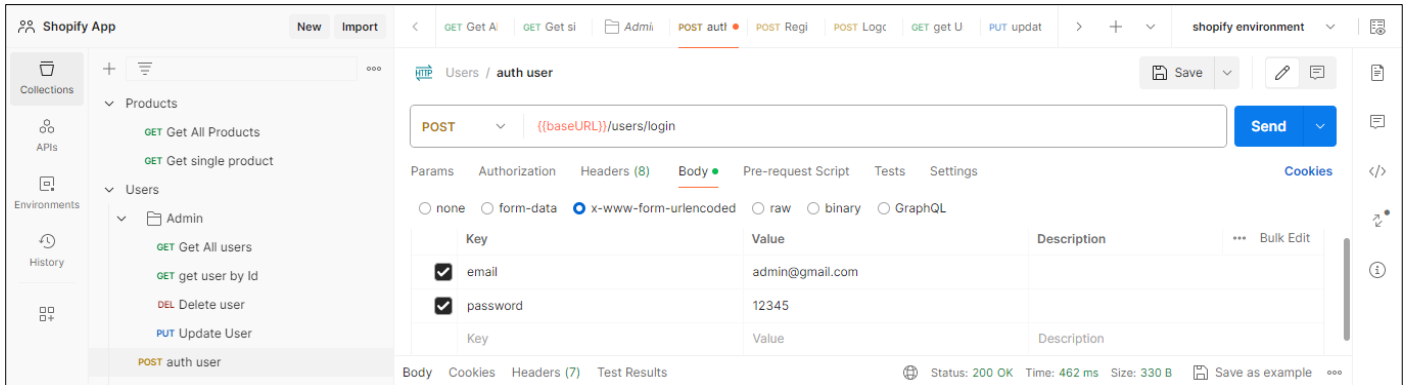
```
//bcrypt password
userSchema.methods.matchPassword = async function (enteredPassword) {
```



```
return await bcrypt.compare(enteredPassword, this.password);
};
```

And now we gonna compare password from db and user authentications

In postman we just passed body as email and password



And authentication code we will do in usercontroller

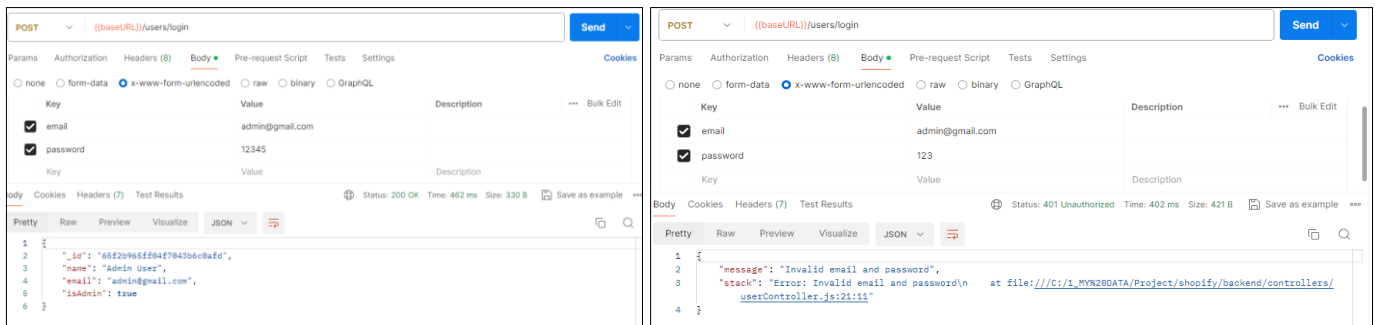
>> backend >> controller >> userController.js

```
// @desc Auth User & get Token
// @route POST /api/users/login
// @access Public
const authUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

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

  if (user && (await user.matchPassword(password))) {
    res.json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
    });
  } else {
    res.status(401);
    throw new Error("Invalid email and password");
  }
});
```

Now hit in **postman** to test with correct and incorrect email and password:



How do JSON web token works ?????:

<https://jwt.io/>

JSON web Token JWT, In web development there's many different ways to authenticate users. You can use cookies, sessions, JSON web Tokens or a combination and there's other service like OAuth and there's third party services like Auth0.

Here we goanna used JSON web tokens because I want to keep this as barebones as possible.

JSON web tokens is a secure way to share information between two parties, such as a web server and a clients and its consists of three parts a header, a payload and a signature.

And Payload contains information like the users ID or the user's role.

Signature is used to verify the information hasn't been tempered with in any way.

JWT https Only Cookies:

Install npm package called jsonwebtoken in roots directory

```
>> npm i jsonwebtoken
```

Now import in controller user in auth..

```
>> userController.js
```

```
import jwt from "jsonwebtoken";
```

```
const authUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

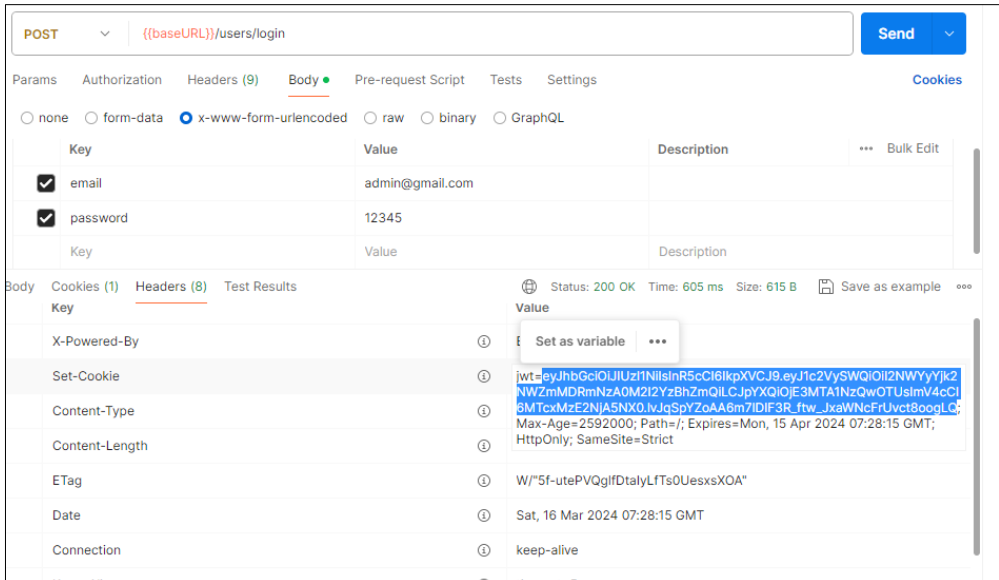
  const user = await User.findOne({ email });

  if (user && (await user.matchPassword(password))) {
    // create JSON web token
    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, {
      expiresIn: "30d",
      // 30 day expire
    });

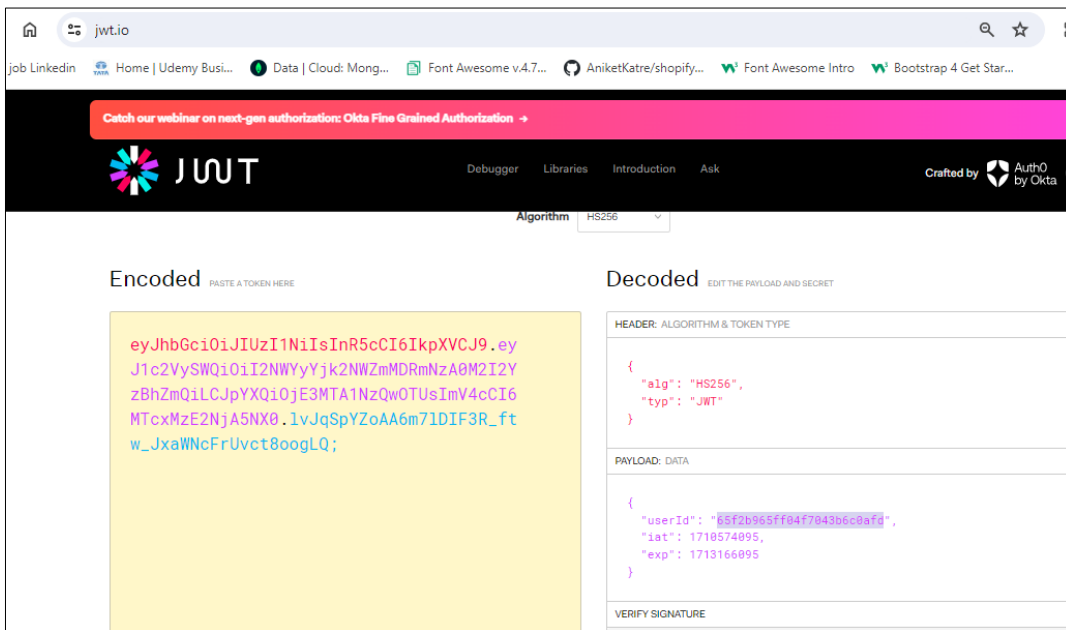
    //set jwt as http-only cookies
    res.cookie("jwt", token, {
      httpOnly: true,
```

```
secure: process.env.NODE_ENV !== "development",
sameSite: "strict",
maxAge: 30 * 24 * 60 * 60 * 1000, // 30 days in milisecond
});
..... continue
```

Now check once in POSTMAN and login with correct credentials Then, we received Cookies and Headers.



And now paste in jwt.io website to cross check with user id along with mongoose user id in DB



Now we will actual use in auth middleware in app:

Now we need a way to take that cookie and use it and we need to get the user ID from it.

```
>> npm i cookie-parser
```

And implement server.js

```
import cookieParser from "cookie-parser";
.....(code more)
// Cookie parser middleware
app.use(cookieParser());
```

Now, we will create authMiddleware to check authentication like normal user don't have permission to see how many order dispatch... n all this. To protect and admin activity we create new middleware

```
>> backend >> middleware >> authMiddleware.js
```

```
import jwt from "jsonwebtoken";
import asyncHandler from "../asyncHandler.js";
import User from "../models/userModel.js";
// Protect routes
const protect = asyncHandler(async (req, res, next) => {
  let token;
  // Read the JWT from the cookie
  token = req.cookies.jwt;

  if (token) {
    try {
      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      req.user = await User.findById(decoded.userId).select("-password");
      next();
    } catch (error) {
      console.log(error);
      res.status(401);
      throw new Error("Not authorized, token failed");
    }
  } else {
    res.status(401);
    throw new Error("Not authorized, no token");
  }
});

// admin middleware
const admin = (req, res, next) => {
  if (req.user && req.user.isAdmin) {
    next();
  } else {
    res.status(401);
    throw new Error("Not authorized as admin");
  }
};

export { protect, admin };
```

and then implement in routes..

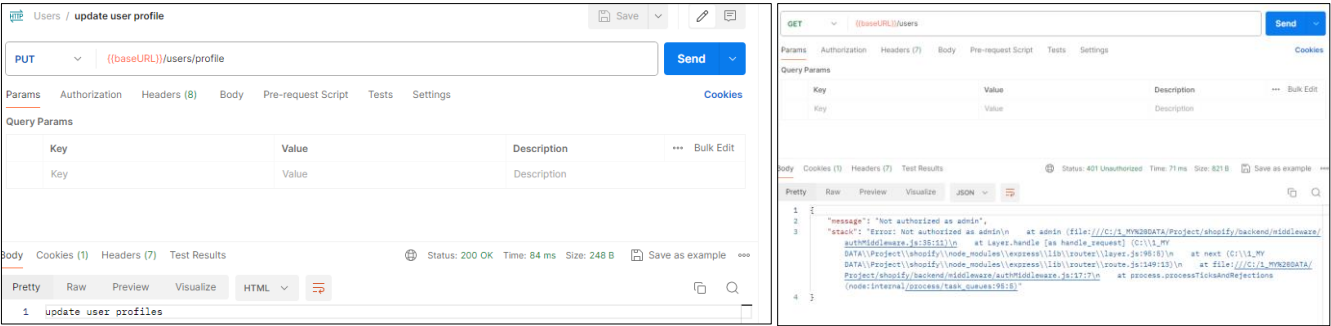
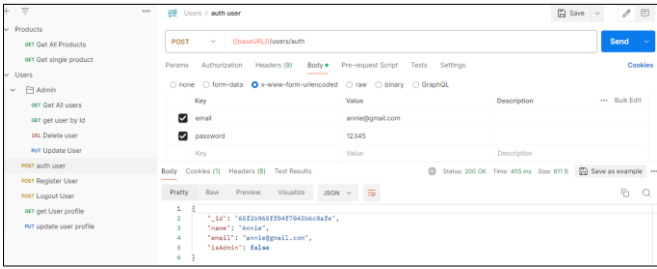
```
>> backend >> routes >> userRoutes.js
```

```
import { protect, admin } from "../middleware/authMiddleware.js";

router.route("/").post(registerUser).get(protect, admin, getUsers);
router.post("/logout", logoutUser);
router.post("/auth", authUser);
router
  .route("/profile")
  .get(protect, getUserProfile)
  .put(protect, updateUserProfile);
router
  .route("/:id")
  .delete(protect, admin, deleteUser)
  .get(protect, admin, getUserById)
  .put(protect, admin, updateUser);
```

Now let check and test in POSTMAN:

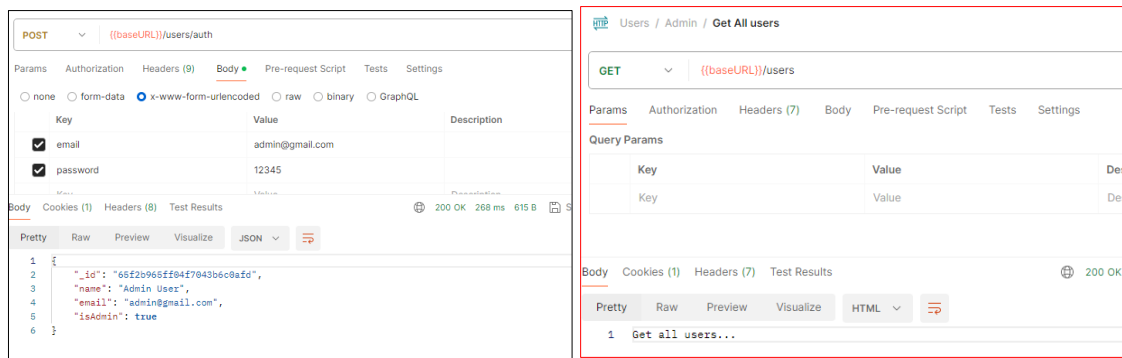
First login with normal User : annie@gmail.com cookies and header generated with token



Get UserProfile can access by annie

And get all user is not authorized as annie is not a admin

#2 Login with Admin:



Now Logged out user:

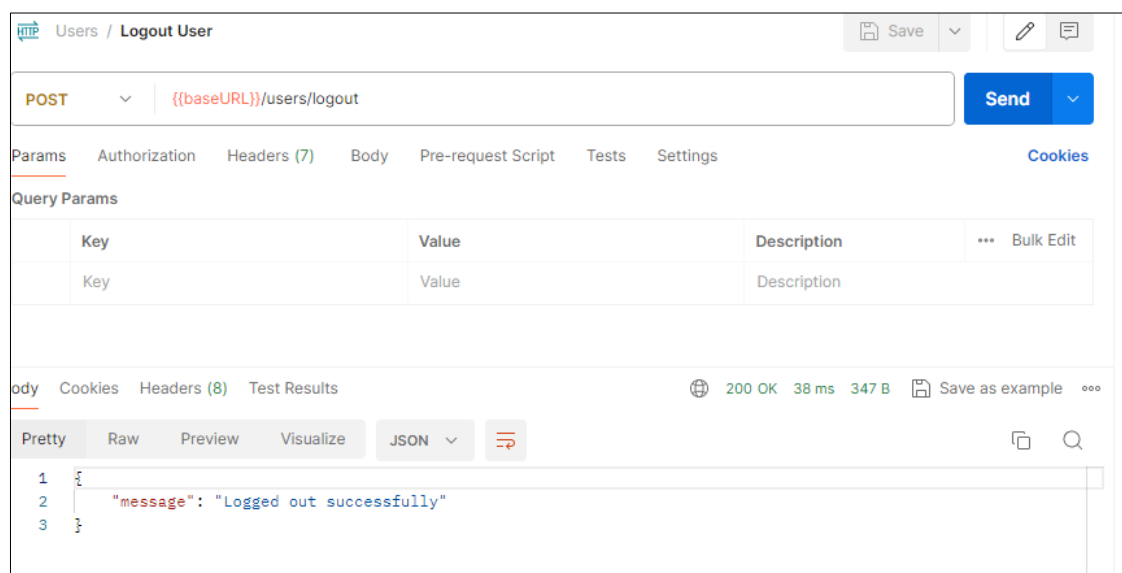
Logged out I just too clear all cookies.

>> backend >> controller >> userController.js

```
const logoutUser = asyncHandler(async (req, res) => {
  res.cookie("jwt", "", {
    httpOnly: true,
    expires: new Date(0),
  });

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

Test in POSTMAN:



Now Register user:

..... check GitHub commit repos

<https://github.com/AniketKatre/shopifyMERN/commit/2258127329bf9d59bf805a08f5d065d537c18370?diff=unified&w=1>

#15: Frontend Authentication:

AuthSlice and UserApiSlice to get data from backend to frontend using slices, Create slice for user authentications.

>> frontend >> slices >> authSlice.js

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
  userInfo: localStorage.getItem("userInfo")
    ? JSON.parse(localStorage.getItem("userInfo"))
    : null,
};

const authSlice = createSlice({
  name: "auth",
  initialState,
  reducers: {
    setCredentials: (state, action) => {
      state.userInfo = action.payload;
      localStorage.setItem("userInfo", JSON.stringify(action.payload));
    },
  },
});

export const { setCredentials } = authSlice.actions;

export default authSlice.reducer;
```

And userApiSlice.jsx in same dir..

```
import { USERS_URL } from "../constants";
import { apiSlice } from "../apiSlice";

export const usersApiSlice = apiSlice.injectEndpoints({
  endpoints: (builder) => ({
    login: builder.mutation({
      query: (data) => ({
        url: USERS_URL / auth,
        method: POST,
        body: data,
      }),
    }),
  }),
});

export const { useLoginMutation } = usersApiSlice;
```

Now, we will store this slices into store.js

```
>> frontend >> src >> store.js
```

```
import authSliceReducer from "../slices/authSlice";
```

```
const store = configureStore({
  reducer: {
    [apiSlice.reducerPath]: apiSlice.reducer,
    cart: cartSliceReducer,
    auth: authSliceReducer,
```

Then Create Login screen using bootstrap and reactrouter-dom

Login Functionality:

If user email address or password get wrong credential, we need to show something in frontend alert message. We use toaster to show message.

```
>> frontend >> terminal- >> npm i react-toastify
```

#16: Checkout Process:

Make cart slices for shipping address.

>> Frontend >> src >> slices >> cartSlice.js

Initial state we have added for shipping address

```
const initialState = localStorage.getItem("cart")
  ? JSON.parse(localStorage.getItem("cart"))
  : { cartItems: [], shippingAddress: {}, paymentMethod: "PayPal" };
```

Inside cartslice function

```
saveShippingAddress: (state, action) => {
  state.shippingAddress = action.payload;
  return updateCart(state);
},
```

And then,

```
export const { addToCart, removeFromCart, saveShippingAddress } =
  cartSlice.actions;
```

Now make ShippingScreen.jsx

Now Next make private routes:

Why need even though if you are not login and you were accessing /shipping routes <http://localhost:3000/shipping>, to make things correct and only if user login then n only able to shipping address routes.

Now we need private routes:

>> frontend >> src >> components >> PrivateRoutes.js

Outlet is basically what we want to return if we're logged in, if there's a user because it just will, it will put out whatever pae or screen we're trying to load.

If we're not logged in, then we're going to use the navigate component to basically just redirect us to login page.

```
import { Outlet, Navigate } from "react-router-dom";
import { useSelector } from "react-redux";

const PrivateRoute = () => {
  const { userInfo } = useSelector((state) => state.auth);

  return userInfo ? <Outlet /> : <Navigate to="/login" replace />;
};

export default PrivateRoute;
```

and import in index.js file 📁

```
{/* private routes */}
<Route path="/" element={<PrivateRoute />}>
  <Route path="/shipping" element={<ShippingScreen />} />
</Route>
```

Checkout steps component:

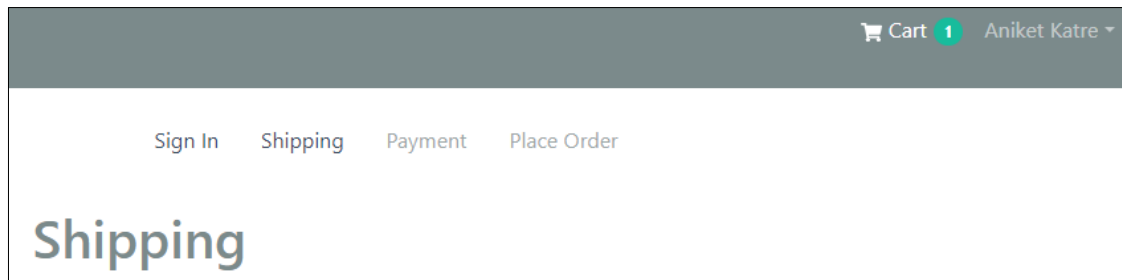
>> frontend >> src >> components >> CheckoutSteps.jsx

```
import React from "react";
import { Nav } from "react-bootstrap";
import { LinkContainer } from "react-router-bootstrap";

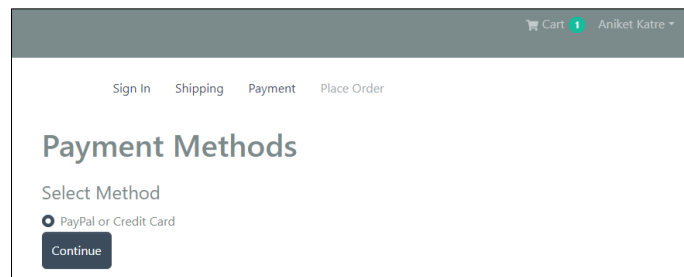
const CheckoutSteps = ({ step1, step2, step3, step4 }) => {
  return (
    <>
      <Nav className="justify-content-center mb-4">
        <Nav.Item>
          {step1 ? (
            <LinkContainer to="/login">
              <Nav.Link>Sign In</Nav.Link>
            </LinkContainer>
          ) : (
            <Nav.Link disabled>Sign In</Nav.Link>
          )}
        </Nav.Item>
      </Nav>
    </>
  );
}
```

And based in screen and steps pass the props on that particular screen.

```
<CheckoutSteps step1 step2 />
```



Payment Methods:



Create payment method on cart slice and export it...

```
>> frontend >> src >> slices >> cartSlices.js
```

```
savePaymentMethod: (state, action) => {
  state.paymentMethod = action.payload;
  return updateCart(state);
},
```

And do export in same file....

```
export const {
  addToCart,
  removeFromCart,
  saveShippingAddress,
  savePaymentMethod,
} = cartSlice.actions;
```

And Make PaymentScreen.jsx UI as show above.

Order routes component:


Now, we going to Backend cause, we need to work with orders and make orders controller in backend. Now, same we need to implement same like we done for user/product controller and its routes.

Let create a new file in the backend route folder and controller.

```
>> backend >> routes >> orderRoutes.js
```

AND

>backend >> controller >> orderController.js

Check the 11th commit GitHub for more info and code... 

Now Get all order and create orders. On same commit.

AFTER this we need to implement in frontend to create or get orders

Now, lets dive to frontend and make new order slices

>> frontend >> src >> slices >> orderApiSlice.js

```
import { apiSlice } from "../apiSlice";
import { ORDERS_URL } from "../constants";

export const ordersApiSlice = apiSlice.injectEndpoints({
  endpoints: (builder) => ({
    createOrder: builder.mutation({
      query: (order) => ({
        url: ORDERS_URL,
        method: "POST",
        body: { ...order },
      }),
    }),
  }),
});

export const { useCreateOrderMutation } = ordersApiSlice;
```

Now once cart items is in placeorder and also need to clear the cart items.

>> frontend >> src >> slices >> cartSlices.js

```
clearCartItems: (state, action) => {
  state.cartItems = [];
  return updateCart(state);
},
```

And create placeOrderScreen.jsx below is the ref. UI

[Sign In](#) [Shipping](#) [Payment](#) [Place Order](#)

Shipping

Address: Gode layout , Nagpur 441108, India

Payment Method

Method: PayPal

Order Items

Your cart is empty!

Order Summary

Items:	\$0.00
Shipping:	\$10.00
Tax:	\$0.00
Total:	\$10.00

[Place Order](#)

Shopify !© 2024 : katreaniket3@gmail.com

For more on above please checkout GitHub 11th commit

<https://github.com/AniketKatre/shopifyMERN/commit/c815f87046e7aea5faa87cc35fde0a041498308b>

#17: Checkout Process and Order screen & PayPal Integration:

Order Page:

Order Sreen65f98fddfa6537e5496c726a

Shipping

Name: Aniket Katre

Email: aniket@gmail.com

Address: Gode layout , Nagpur 441108, India

Not Delivered

Order Summary


Items	\$0
Shipping	\$0
Tax	\$270
Total	\$2069.97

Payment Method

Method: PayPal

Not Paid

Order Items

 iPhone 11 Pro 256GB Memory	3 * \$599.99 = \$1799.97
--	--------------------------

Now PayPal API used for payment:

<https://developer.paypal.com/>

Login in: annie.jb90@ and password

GOTO:

Apps & Credential >> Create App >> fill form and create app

You're in sandbox mode.

[← Back](#)

shopify_github

Viewing sandbox API credentials. [Upgrade your account to PayPal for Business to view live credentials.](#)

API credentials

App name

shopify_github

Client ID

AC

ts9

d5xH3

ccmbK

Secret key 1

.....

[+ Add Second Key](#)

And copy Client ID and pt in .env file as

```
PAYPAL_CLIENT_ID= .....
```

Now this paypal account in backend

>>backend >> server.js

```
app.get("/api/config/paypal", (req, res) =>
  res.send({ clientId: process.env.PAYPAL_CLIENT_ID })
);
```

Now, implement in frontend

>> frontend >> terminal window

>> npm i @paypal/react-paypal-js

And add in index.js

>> frontend >> src >> index.js

```
import { PayPalScriptProvider } from "@paypal/react-paypal-js";
```

```
<PayPalScriptProvider deferLoading={true}>
  <RouterProvider router={router} />
</PayPalScriptProvider>
```

And also add slices in orderApiSlice.js

>> frontend >> src >> slices >> orderApiSlice.js

```
payOrder: builder.mutation({
  query: (orderId, details) => ({
    url: `${ORDERS_URL}/${orderId}/pay`,
    method: "PUT",
    body: { ...details },
  }),
}),
getPayPalClientId: builder.mutation({
  query: () => ({
    url: PAYPAL_URL,
  }),
  keepUnusedDataFor: 5,
}),
```

And then implement in OrderScreen.jsx

Order Sreen 65fabd9c6336f68975644097

Shipping

Name: Aniket Katre
Email: aniket@gmail.com
Address: Gode layout, Nagpur-441108, India

Not Delivered

Payment Method

Method: PayPal

Not Paid

Order Items

1 * iPhone 11 Pro 256GB Memory	\$599.99
--------------------------------	----------

Order Summary

Items	\$599.99
Shipping	\$0
Tax	\$90
Total	\$689.99

Test Pay Order

PayPal

Debit or Credit Card

Powered by PayPal

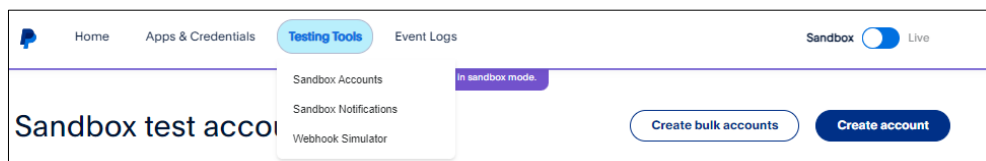
PAY with PayPal:

Click on Paypal and then it will ask for login with mail id and password. This mail id and password we will capture from PayPal developer account sandbox.

Below are the steps to get credential

Step1 : <https://developer.paypal.com> LOGIN

Step2: Click on Testing Tools and SandBox Account



Step3: You will get Personal and Business sandbox account. Choose **Personal** >> click 3dot and **view/edit**

Sandbox test accounts				
<p>You can use sandbox accounts to test your apps and mimic live transactions. Read more about sandbox testing</p> <p>Missing a test account? Link other sandbox accounts to this developer account. Have your sandbox logins ready.</p> <p>Showing 2 sandbox accounts:</p>				
Account name	Type	Country	Date created	
sb-tglcj285...business.example.com	Business	US	12/5/23, 1:33 AM	
sb-6odha28...personal.example.com	Personal	US	12/5/23, 1:33 AM	

[← Back](#)

sb-6od[REDACTED]32@personal.example.com

Login Info

Sandbox URL

https://sandbox.paypal.com

Email

sb-6od[REDACTED]example.com

Password

.....

👁

🔑

[Change password](#)

Sandbox account info

Name	Annie JB ✎	PayPal balance	USD 0.00 ✎
Phone	4088551831	Bank account number	520730366350
Account type	Personal	Bank routing number	325272063

#17: Admin functionalities:

ORDER SCREEN:

fy

🛒 Cart

iamAdmin

Admin

Products

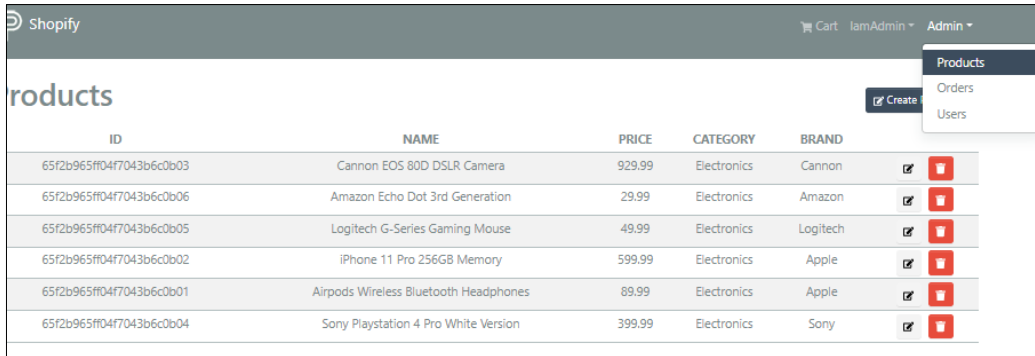
Orders

Users

ID	DATE	TOTAL	PAID	DELIVERED		
65faab59c4a0497afb878bd	Aniket K	2024-03-20	\$689.99	2024-03-20	2024-03-20	Details
65fab706336f68975644081	Aniket K	2024-03-20	\$1069.49	2024-03-20	2024-03-20	Details
65fabd9c6336f68975644097	Aniket K	2024-03-20	\$689.99	✖	✖	Details
65fb17fc44a58a14c78edab8	Annie	2024-03-20	\$689.99	2024-03-20	✖	Details
65fb180f44a58a14c78edaca	Annie	2024-03-20	\$172.47	2024-03-20	✖	Details
65fb187144a58a14c78edae1	Annie	2024-03-20	\$1069.49	✖	✖	Details

PRODUCTs SCREEN:

Create product screen UI:



ID	NAME	PRICE	CATEGORY	BRAND
65f2b965ff04f7043b6c0b03	Cannon EOS 80D DSLR Camera	929.99	Electronics	Cannon
65f2b965ff04f7043b6c0b06	Amazon Echo Dot 3rd Generation	29.99	Electronics	Amazon
65f2b965ff04f7043b6c0b05	Logitech G-Series Gaming Mouse	49.99	Electronics	Logitech
65f2b965ff04f7043b6c0b02	iPhone 11 Pro 256GB Memory	599.99	Electronics	Apple
65f2b965ff04f7043b6c0b01	Airpods Wireless Bluetooth Headphones	89.99	Electronics	Apple
65f2b965ff04f7043b6c0b04	Sony Playstation 4 Pro White Version	399.99	Electronics	Sony

Now, adding create new products below are the steps:

First go to backend and create a new api from create new product in productController.js

>> backend >> controllers >> productController.js

```
// @desc    Create a products
// @route   POST /api/products
// @access  PRIVATE and ADMIN
const createProducts = asyncHandler(async (req, res) => {
  const product = new Product({
    name: "Sample name",
    price: 0,
    user: req.user._id,
    image: "/image/sample.jpg",
    brand: "Sample Brand",
    category: "Sample Category",
    countInStock: 0,
    numReviews: 0,
    description: "Sample description",
  });

  const createProduct = await product.save();
  res.status(201).json(createProduct);
});

export { getProducts, getProductById, createProducts };
```

and create a routes

>> backend >> routes >> productRoutes.js

```
import {
  getProducts,
  getProductById,
  createProducts,
} from "../controllers/productController.js";
import { protect, admin } from "../middleware/authMiddleware.js";
```

```
router.route("/").get(getProducts).post(protect, admin, createProducts);
```

Now, go to frontend

```
>> frontend >> src >> slices >> productApiSlice.js
```

```
createProduct: builder.mutation({
  query: () => ({
    url: PRODUCT_URL,
    method: "POST",
  }),
  invalidatesTags: ["Product"],
}),
```

And export this as useCreateProductMutation

Now, move forward to Admin> ProductLstScreen/jsx

```
>> frontend >> src >> screen >> admin >> ProductListScreen.jsx
```

First do import

```
import {
  useGetProductsQuery,
  useCreateProductMutation,
} from "../../slices/productsApiSlice";
```

Then,

```
const [createProduct, { isLoading: loadingCreate }] =
  useCreateProductMutation();
```

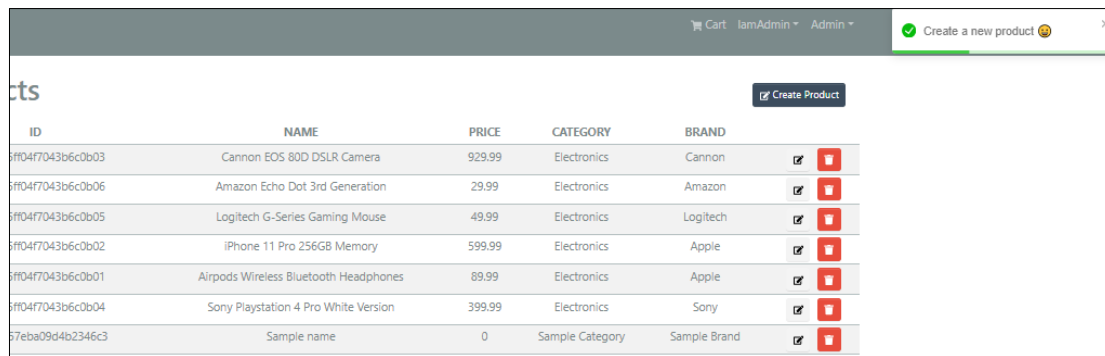
and button handler

```
const createProductHandler = async () => {
  if (window.confirm("Are you sure you want to create a new product?")) {
    try {
      await createProduct();
      refetch();
      toast.success("Create a new product 😊");
    } catch (err) {
      toast.error(err?.data?.message || err.error);
    }
  }
};
```

Below button click above handler call and create new product as shown in below SC

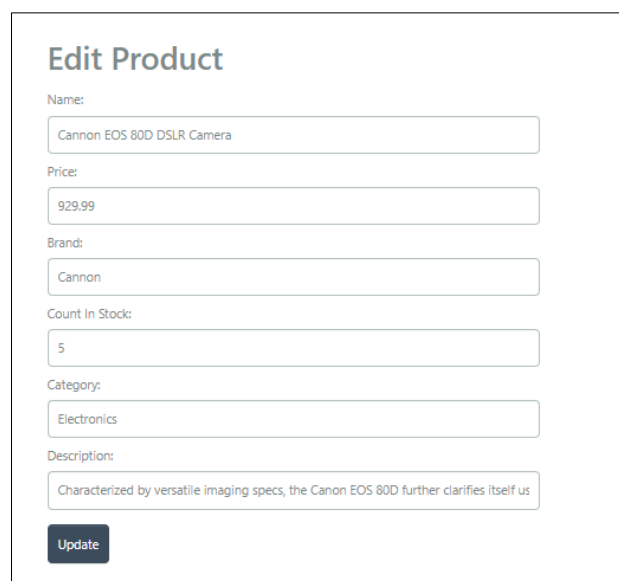
```
<Button className="btn-sm m-3" onClick={createProductHandler}>
  <FaEdit /> Create Product
</Button>
```

SCREEN:



ID	NAME	PRICE	CATEGORY	BRAND		
#f04f7043b6c0b03	Cannon EOS 80D DSLR Camera	929.99	Electronics	Cannon	✓	🔴
#f04f7043b6c0b06	Amazon Echo Dot 3rd Generation	29.99	Electronics	Amazon	✓	🔴
#f04f7043b6c0b05	Logitech G-Series Gaming Mouse	49.99	Electronics	Logitech	✓	🔴
#f04f7043b6c0b02	iPhone 11 Pro 256GB Memory	599.99	Electronics	Apple	✓	🔴
#f04f7043b6c0b01	Airpods Wireless Bluetooth Headphones	89.99	Electronics	Apple	✓	🔴
#f04f7043b6c0b04	Sony Playstation 4 Pro White Version	399.99	Electronics	Sony	✓	🔴
7eba09d4b2346c3	Sample name	0	Sample Category	Sample Brand	✓	🔴

Now Edit button: *check in repos.... 14th commit.*



Edit Product

Name:

Price:

Brand:

Count In Stock:

Category:

Description:

<https://github.com/AniketKatre/shopifyMERN/commit/70d79d3f25ef6f60fd53eb3f6fc668de3ef1cf1a>

Inside form only image upload is remaining we will do using Multer packages.

Multer & Image upload Endpoint:

<https://www.npmjs.com/package/multer>

>> root dir >> npm i multer

#18: Reviews, Search and More.

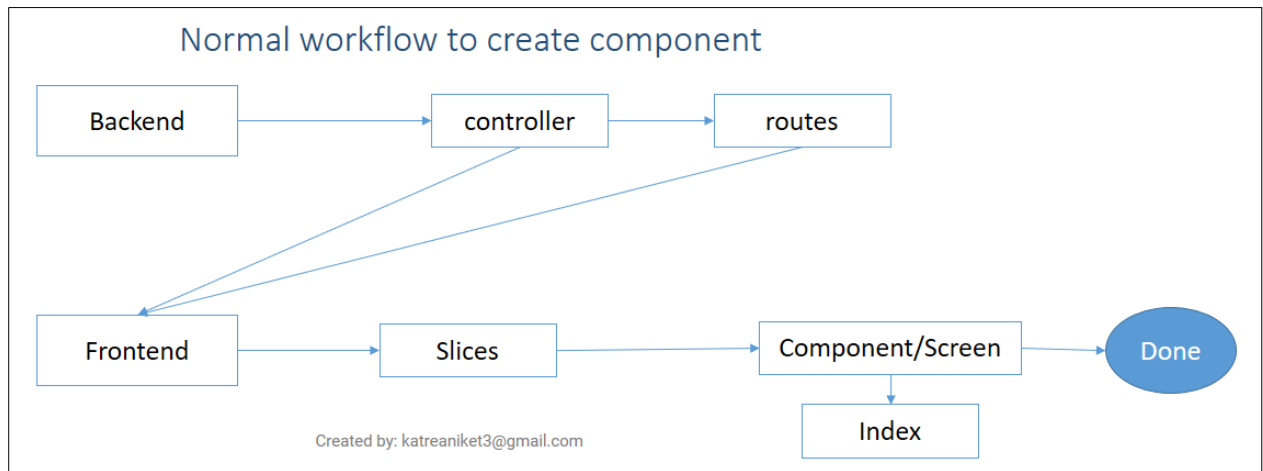
Create Reviews – Backend:

Goto:

>> backend >> controller >> productController.js

Workflows:

mailto: katreaniket3@gmail.com | [linkedin.com/in/aniket-katre-752465149/](https://www.linkedin.com/in/aniket-katre-752465149/) | IG: [@annie_jb_](https://www.instagram.com/annie_jb_) | [github/AniketKatre/](https://github.com/AniketKatre/)



Many more changes for each component please check in github commits

#19: Page title of each page customize.

Install react helmet package

```
>> frontend >> terminal open >> npm i react-helmet-async
```

Then, >> frontend >> src >> index.js

```
import { HelmetProvider } from "react-helmet-async";
```

```

<HelmetProvider>
  <Provider store={store}>
    <PayPalScriptProvider deferLoading={true}>
      <RouterProvider router={router} />
    </PayPalScriptProvider>
  </Provider>
</HelmetProvider>

```

After that

```
>> frontend >> src >> components >> Meta.jsx
```

```

import React from "react";
import { Helmet } from "react-helmet-async";

const Meta = ({ title, description, keywords }) => {
  return (
    <Helmet>
      <title>{title}</title>

```

```

    <meta name="description" content={description} />
    <meta name="keywords" content={keywords} />
  </Helmet>
);
};

Meta.defaultProps = {
  title: "Welcome to Proshop",
  description: "We sell the best experience",
  keywords: "electronic, buy luxury brand, stay classy x sassy",
};

export default Meta;

```

and now implement in each screen like HomeScreen.jsx

```
>> frontend >> src >> screen >> HomeScreen.jsx
```

```
import Meta from "../components/Meta";
```

```
<Meta />
```

Similarly in ProductScreen.jsx

```
<Meta title={product.name} />
```

#20: Deploy to Production.

```
>> backend >> server.js
```

```

if (process.env.NODE_ENV === "production") {
  // uploads folder as static folder
  // Set __dirname to current directory
  const __dirname = path.resolve();
  app.use("/uploads", express.static(path.join(__dirname, "/uploads")));

  // set static folder
  app.use(express.static(path.join(__dirname, "/frontend/build")));
  // any route that is not api will be redirected to index.html
  app.get("*", (req, res) => {
    res.sendFile(path.resolve(__dirname, "frontend", "build", "index.html"));
  });
} else {
  app.get("/", (req, res) => {
    res.send("App is running.....");
  });
}

```

```
});  
}
```

And also change in root package.json

```
"build": "npm install && npm install --prefix frontend && npm run build --prefix  
frontend"
```

It will create build folder in frontend

After that temporarily make changes in env file and change

```
# NODE_ENV=development  
NODE_ENV=production
```

Now, Open terminal window.

Root Dir >> Terminal wnd >> npm run build

Server running in production mode on port [3080](#)

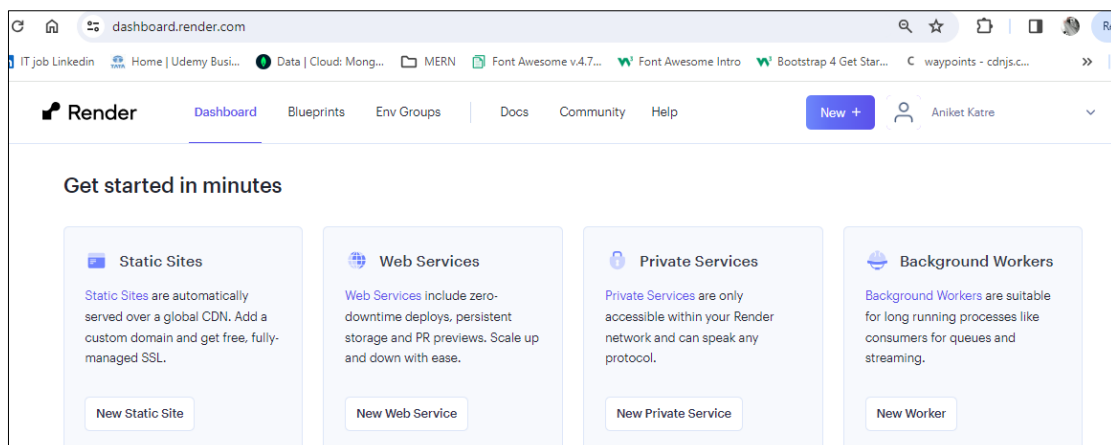
DEPLOY in PROD:

Open <https://www.render.com> and Login with GitHub

Then,

>> <https://dashboard.render.com/>

>> Select New Web Services



>> Build and deploy from git

Create a new Web Service

Connect a Git repository, or use an existing image.

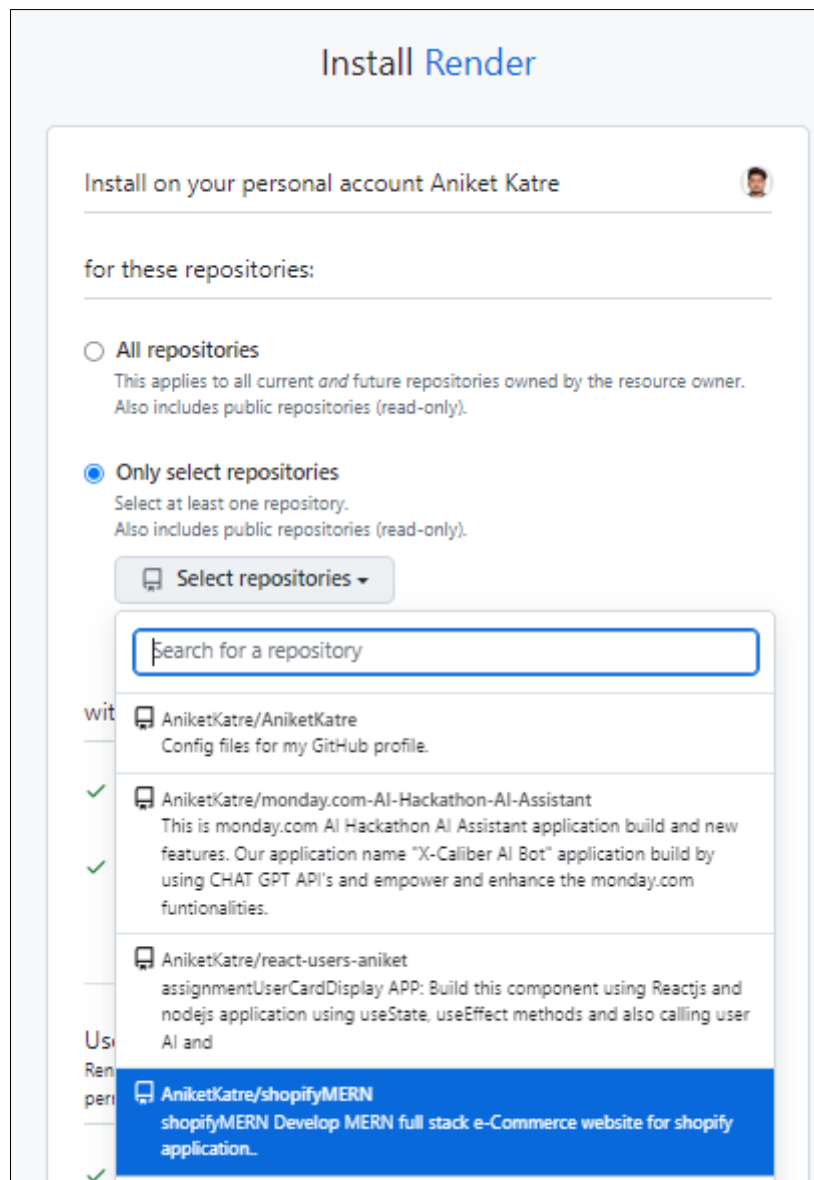
How would you like to deploy your web service?

☒ Build and deploy from a Git repository
Connect a GitHub or GitLab repository.

☐ Deploy an existing image from a registry **ADVANCED**
Pull a public image from any registry or a private image from Docker Hub, GitHub, or GitLab.


Next

>> Connect Repository:



>> Connect to your repo.

>> follow below snapshot

Render [Dashboard](#) [Blueprints](#) [Env Groups](#) | [Docs](#) [Community](#) [Help](#) [New +](#)  Aniket Katre

You are deploying a web service for AniketKatre/shopifyMERN.

You seem to be using Node, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name
A unique name for your web service.

Region
The region where your web service runs.

Branch
The repository branch used for your web service.

Root Directory Optional
Defaults to repository root. When you specify a root directory that is different from your repository root, Render runs all your commands in the specified directory and ignores changes outside the directory.

Runtime
The runtime for your web service.

Build Command
This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs

>> choose free plan and cmd add

Start Command
This command runs in the root directory of your app and is responsible for starting its processes. It is typically used to start a webserver for your app. It can access environment variables defined by you in Render.

Instance Type

For hobby projects

Free \$0 / month	512 MB (RAM) 0.1 CPU	Upgrade to enable more features Free instances spin down after periods of inactivity. They do not support SSH access, scaling, one-off jobs, or persistent disks. Select any paid instance type to enable these features.
----------------------------	-------------------------	---

>> add environment variable in render

OG env from code below paste in render as environment variable field

```

if NODE_ENV=development
NODE_ENV=production
PORT=3080
MONGO_URI=mongodb+srv://katreaniket3:annie123@cluster0.mongodb.net/shopify?retryWrites=true&majorityAppName=Cluster0
JWT_SECRET=annie123
PAYPAL_CLIENT_ID=AXn41_1Fuy45HKNEV3UuQ_2YfP8o51tpYd4jng5FLK-cm6M5v1A7vXUeYd8yPNGeEnr7dhAG8o
PAGINATION_LIMIT=8

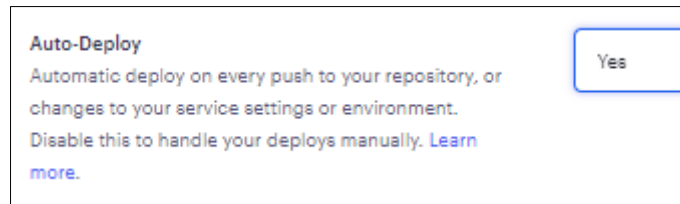
```

Environment Variables Optional
Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more.](#)

MONGO_URI	qktw.mongodb.net/shopify?retryWrites=true&majorityAppName=Cluster0	
JWT_SECRET	annie123	
PAYPAL_CLIENT_ID	AXn41_1Fuy45HKNEV3UuQ_2YfP8o51tpYd4jng5FLK-cm6M5v1A7vXUeYd8yPNGeEnr7dhAG8o	
PAGINATION_LIMIT	8	

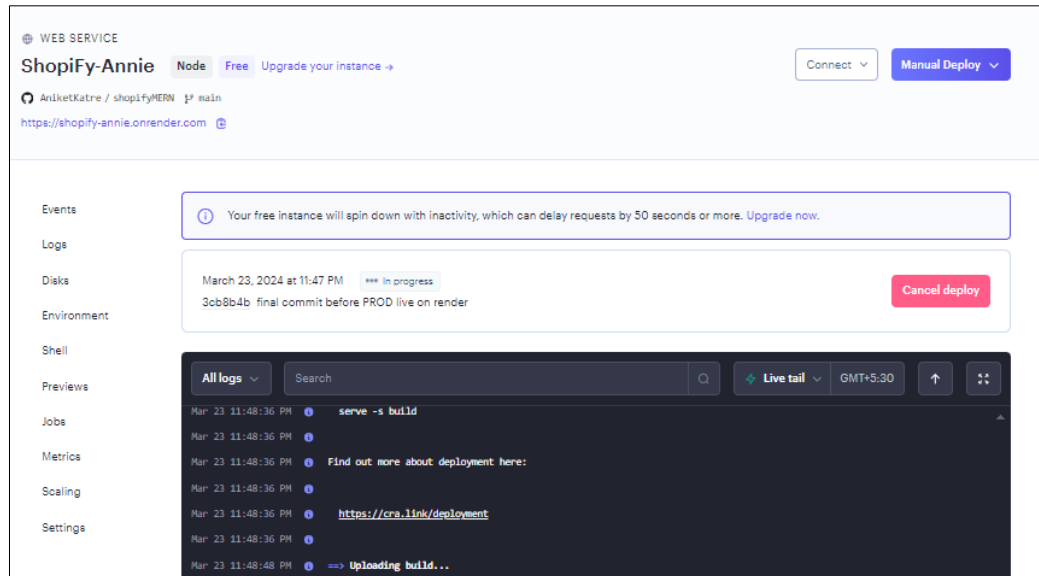
[+ Add Environment Variable](#) [Add from .env](#)

>> AND make auto deploy as yes (your wish)

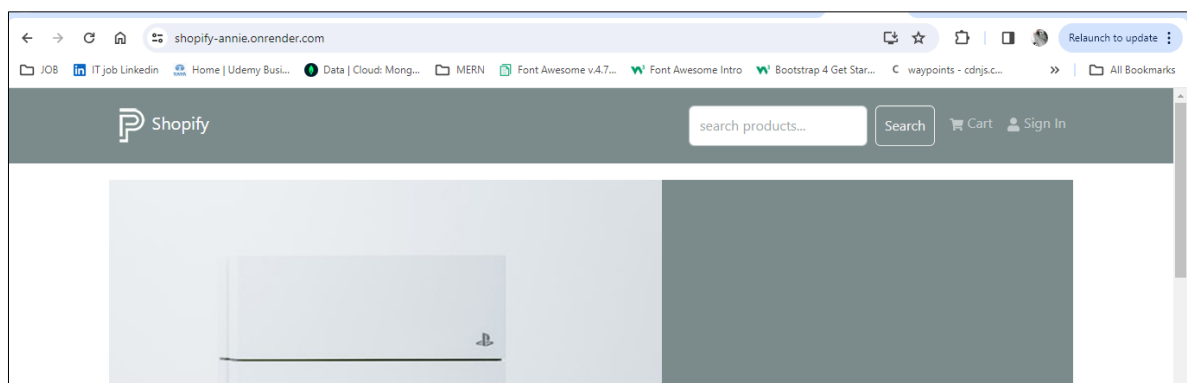


>> Next Click on *Create Web Services*

After this Finally hosted



LIVE:



<https://shopify-annie.onrender.com/>

