**PROJECT REPORT**

**ON**

**E-COMMERCE PLATFORM: SHOPCART**

Submitted by:

WEB WARP

Group: G19

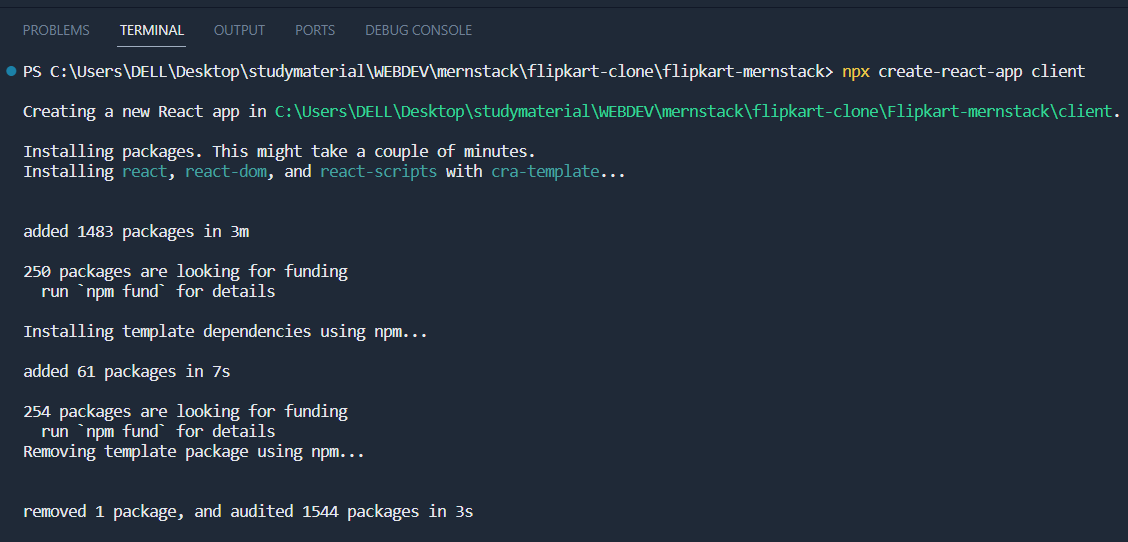
# Submitted to: Mr. Ashwani Dubey



**Department of Computer Science Engineering**

**Chitkara University, Punjab**

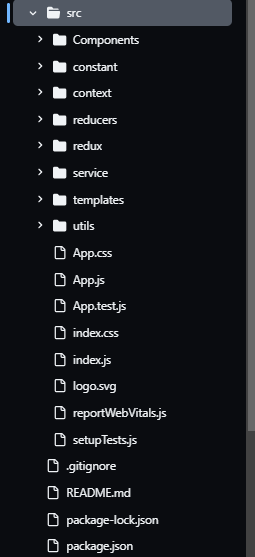
**DOCUMENTATION FOR WEB WARP**



**Frontend Part :**

For front-end part we’ve here used is react javascript framework and created client side application using command **npx create-react-app client .** This command creates a client side app using react.

**The components used here are :**

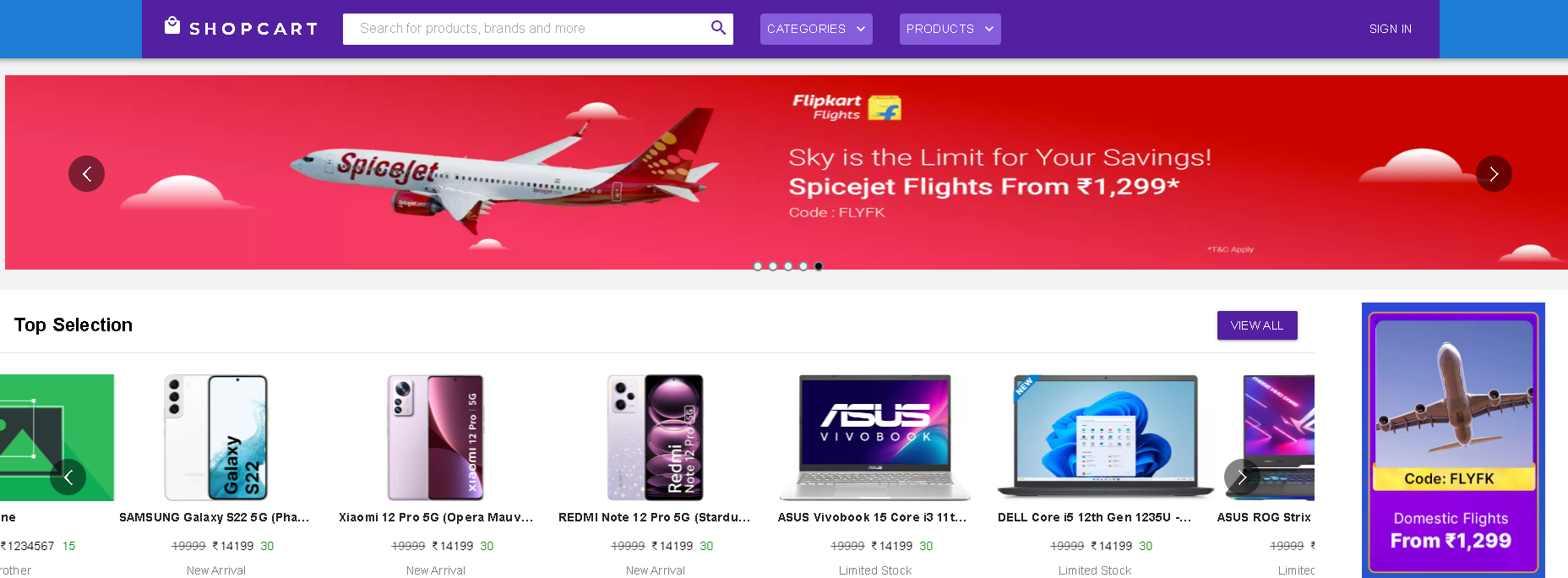


Front-end side components uses :

* + Components : Inside components we’ve subcomponents that are :
    - **Header** consisting of CustomButtons.jsx , Header.jsx , Profile.jsx and Search.jsx
    - **Home** consisting of jsx files of banner , midsection , midslide , navbar and slide.jsx
    - **Login** consisting of Home.jsx, Notdound.jsx,Product,jsx and default.jsx
  + Constant : Inside components data.js file
  + Context : Inside context file we’ve ContextProvider.jsx file
  + Reducers : Inside this folder we’ve reducer.js file
  + Redux : Inside redux we’ve subcomponents such as action , constants and reducers along with store.js file
    - Action contains cartActions.js and productActions.js
    - Constants consists of cartConstants.js and productConstant.js
    - Inside reducer we’ve store.js
  + Service : Inside service file we’ve api.js
  + Templates : Inside templates we’ve TemplateProvider.js file
  + Utils : Inside this we’ve 2 files i.e. payment.js and util.js

**EXPLANATION OF COMPONENTS :**

* + - **HEADER**



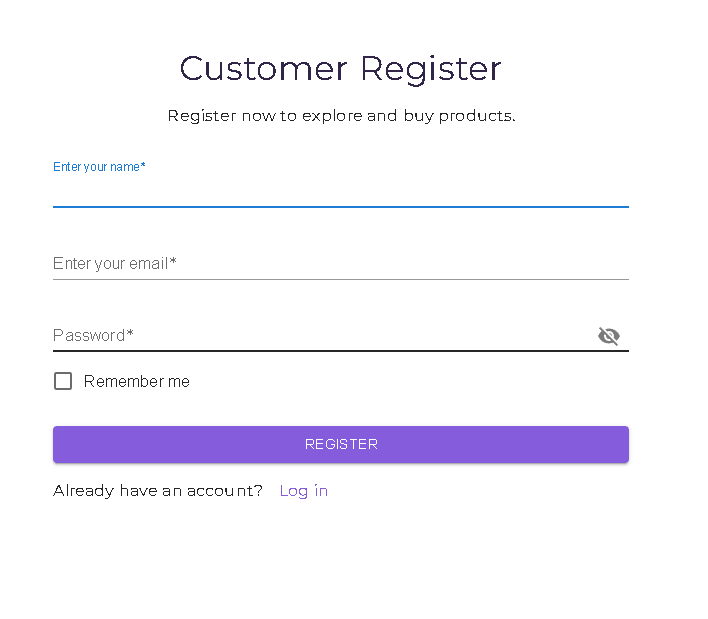
This is a React component called CustomButton, which functions as a custom navigation bar with buttons for logging in, becoming a seller, and seeing the cart.

* + - * + It imports React hooks (useState and useContext).
        + It uses components and icons from the Material-UI package (Box, Button, Typography, and Shopping Cart).
        + It includes the styled function from '@emotion/styled'.
        + It obtains the DataContext from '../../context/DataProvider'.
        + It includes the LoginDialog component from '../login/LoginDialog'.
        + It leverages Emotion's styled tool to generate styled components (Wrapper, container, and LoginButton) with particular styling.
        + The primary component is a functional component called CustomButton.
        + It use the useState hook to control the status of the open variable, which decides whether the login dialogue is open or closed.
        + It utilises the useContext hook to retrieve the account value from the DataContext. The DataContext most likely contains global state information about the user account.
        + The openDialog function is designed to set
        + The component presents various components depending on whether there is an account in the context.
        + If an account exists, it displays the account details using a Typography component.
        + If no account exists, it displays a LoginButton with the words "Login". This button launches the login dialogue.
        + It also generates typographic components for "Become a Seller" and "More".
        + A container is used to show the ShoppingCart symbol and the "Cart" typography.
        + The LoginDialog component is presented in an open state and has a method (setOpen) to adjust its visibility
    - **HOME**
      * + This is a React component called Home that represents the primary material of a home page. It has a navigation bar and a banner. Here's an overview of the important features :

It imports two custom components: NavBar and Banner.

* + - * + It imports the essential components from the Material-UI library, such as Box and Styled.
        + It makes advantage of Material-UI's styled utility to construct a stylized component called Component. This component represents a container with specified padding and background colour as a container for the primary content.
        + Inside the Component, the Banner component is rendered. The Banner component might contain promotional content or vital information for the home page.
        + The Home component exports as the module's default export.
        + Overall, this Home component offers a framework for the home page by incorporating a navigation bar (NavBar) and a banner (Banner). The container's styling is done with Material-UI's styled utility.
    - **LOGIN**

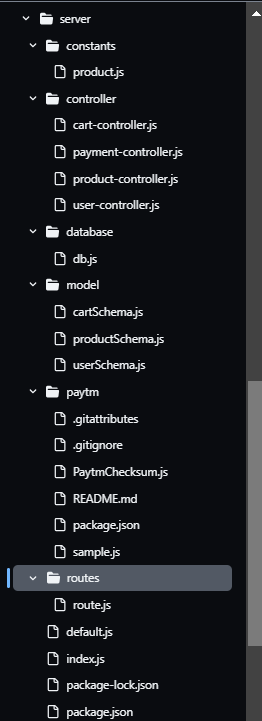


* + - * + It employs Material-UI's styled utility to generate a number of styled components (Component, Image, Wrapper, LoginButton, RequestOTP, Text, and CreateAccount) that are tailored to the supplied theme.
        + It specifies the starting settings for the registration form (signupInitialValues) and the account view (accountInitialValues), which include login and signup information.
        + It manages the dialog's state using the useState hook. The account state controls the current view (login or signup), whereas the signup state stores the signup form data.
        + It accesses the DataContext using the useContext hook. The setAccount function in the context is used to change the account information.
        + It defines toggleSignup, which switches the display between login and signup.
        + It defines handleClose, which closes the dialogue and resets the view to login.
        + It defines onInputChange, which updates the signup state as the user types into the form.
        + It defines signupUser, which sends a signup request using the authenticateSignup method from the service/api file.
        + After a successful signup, it closes the dialogue and updates the account details with the setAccount method.
        + The component renders a Dialogue from Material-UI with appropriate open and onClose properties.
        + The main container for the dialogue is a Component styled component.
        + It creates a Box with two children.
        + The picture themed component shows a backdrop picture with some text dependent on the account state.
        + When the account.view is 'login', the Wrapper displays a login form with email/mobile and password fields, a login button, an OTP request button, and a link to establish a new account.
        + When the account.view is'signup', the Wrapper displays a signup form with fields for firstname, lastname, username, email, password, and phone, as well as a "Continue" button.
        + The LoginDialog component exports as the module's default export.
    - **CONSTANT**
      * + An array of objects that represent navigation elements.
        + Each object includes a url property that links to an image and a text property that contains the written description of the navigation item.
        + An array of objects that represent banner pictures.
        + Each object contains a unique id and a url attribute that points to the banner picture.
        + An array of objects representing deal items.
        + Each object contains characteristics such as id, url for the product picture, detailUrl for a detailed look, title (both short and long), price (which includes MRP, cost, and discount), description, discount, and tagline.
        + An array of objects that represent furniture pieces.
        + Each object has characteristics such as the URL for the product picture, the title (which includes a short title), the discount, and the tagline.
        + navData stores data for navigation elements that are often shown at the top of a webpage.
        + BannerData provides graphics that might be used as banners on a website.
        + dealData represents discounted products.
        + FurnitureData includes information on furniture goods that may be shown on a furniture-related area of the website.
        + These data sets appear to be appropriate for an e-commerce website that may include sections like navigation, advertisements, specials of the day, and furniture goods.
        + These data sets may be used in React components to dynamically generate information based on the input data. For example, you might iterate through these arrays and render components for each item in your user interface.
    - **CONTEXT**
      * + Certainly! This code creates a React context and a context provider using the createContext and useState functions from React.
        + The createContext method is used to establish a new context. A context in React allows several components to communicate variables such as state without explicitly providing props through each level of the component tree.
        + The useState hook manages state in a functional component. It produces an array containing two elements: the current state value and a method to update it.
        + DataContext is a context object produced using createContext. This context is exported, allowing other components to utilise it.
        + This is a component that provides context. It encapsulates its children in the DataContext.Provider component,
        + . This is a component that provides context. It encapsulates its children in the DataContext.Provider component, making the context value accessible to all of its successors.
        + const { account, setAccount} = useState('');:
        + This line used the useState hook to establish a piece of state called account with an initial value of an empty string, and setAccount as the function to change the state.
        + The component returns the DataContext.Provider with its value set to an object that includes account and setAccount. This makes these values available to any component that is a descendant of the DataProvider component.
        + This DataProvider will cover the components named {children}. This is a typical approach in React for constructing a wrapper component that gives specific context.
        + This is a typical React approach for constructing a wrapper component that gives a specific context or state to its children.
        + When you use DataProvider in your application, every component that is wrapped in DataProvider gains access to the DataContext. Components may then consume the context and access the shared state using the useContext hook (in this example, account and setAccount).
        + This arrangement is excellent for maintaining global state that has to be shared across several areas of your application without having to manually send props through each level.
    - **REDUCERS**
      * + Import statements include createContext and useState from the'react' package.
        + Data Context Creation:
        + Using the createContext method, we create a context named DataContext. This context will be used to pass state across components.
        + DataProvider Component:
        + Define a functional component called DataProvider that accepts children as a prop. This component will wrap its children in the DataContext.Provider.
        + State initialization
        + Using the useState hook to set a state variable account to an empty string ('').
        + Context Provider Usage:
        + Rendering the DataContext.Provider of a value prop set to an object containing account and setAccount. This offers the state as well as the method to update the state for all components in the provider.
        + Rendering Children:
        + Rendering the DataContext.Provider's child components. This enables the wrapped components to access the shared state using the useContext hook.
        + Exporting Data Provider:
        + Exporting the DataProvider component as the module's default export.
        + In this example, DataProvider acts as a context provider, making the account state and setAccount method available to any components that inherit from it. This is helpful for handling the global state of a React application.
    - **SERVICE**
      * + This file offers a series of utility methods that use Axios to send asynchronous HTTP requests to a server at http://localhost:8000. Let us break down each function:
        + To authenticate a user, submit a POST request to the /login endpoint.
        + The user object contains the user credentials.
        + On success, a promise is returned, which resolves to an Axios response object. If an error occurs, it records it and returns undefined.
        + A POST request to the /signup endpoint creates a new user.
        + Parameters: A user object holding user information.
        + On success, a promise is returned, which resolves to an Axios response object. If an error occurs, it records it and returns undefined.
        + To retrieve product details, submit a GET request to the /product/${id} endpoint.
        + id of the product to be retrieved.
        + On success, a promise is returned, which resolves to an Axios response object. If an error occurs, it records it and returns undefined.
        + Pay Using Paytm Function
        + The purpose is to initiate a Paytm payment by submitting a POST request to the /payment endpoint.
        + Parameters: a data object with payment-related information.
        + Returns: A promise that resolves to the data received from the server upon success. If an error occurs, it records it and returns undefined.
        + These functions are intended to be used in an asynchronous situation, and it is anticipated that the caller code will handle the promises provided by these functions correctly. Furthermore, error handling is included in each function to report failures and prevent unhandled promise rejections.
    - **TEMPLATES**
      * + This React component creates a template context with the Material-UI (MUI) theming framework. Let's go out the main elements of this component. Creates a React context with React.createContext(null). Context allows you to transfer data across the component tree without having to manually feed props down at each level.
        + This is the major component that contains the application's MUI theming and style configuration.
        + Props: It accepts a single parameter, children, which denotes the child components nested within this provider.
        + Const theme = createTheme();:
        + It uses @mui/material/styles' createTheme method to generate a MUI theme. The theme object is then supplied to the ThemeProvider, which applies the theme to the whole component tree.
        + return (...) Block.
        + The component generates a JSX block containing: Provides template context to its descendants. Wraps the application with the MUI theme. Provides a uniform basic style across several browsers.
    - **UTILS**
      * + This JavaScript module includes functions names isDate, isObj, stringifyValue, and buildForm. isDate determines whether the provided value val is a JavaScript Date object.
        + Returns true if val is a Date object, and false otherwise.
        + isObj determines whether the specified value val is an object.
        + Returns true if val is an object; otherwise, false.
        + stringify takes a value (val) and changes it into a string. If val is an object (except Date objects), the conversion is performed using JSON.stringify.
        + returns the string representation of val.
        + buildForm It generates a new HTML form dynamically.
        + Sets the form's method to 'post' and the action to the value specified in the action argument.
        + The params object generates hidden input fields for each key-value combination.
        + It turns the values to strings using stringifyValue.
        + Returns the built form.
        + It accepts an object's information, including attributes, actions, and parameters.
        + The buildForm function generates a form with the supplied action and parameters.
        + Adds the form to the document body.
        + Submit the form.
        + Removes the form from the document body once it has been submitted.
    - **APP.JS**
      * + This is the React application starting point (App.js). Let us break down the essential components:
        + BrowserRouter:
        + This component is part of the react-router-dom package, which provides your application's routing architecture.
        + paths are containers for specifying individual paths inside an application.
        + Route: Identifies a single route.
        + The path property specifies the route path.
        + The element prop specifies which React element should be rendered when the route is matched.
        + Box is a utility component from the @mui/material library used for style. In this example, it adds a top margin to the content to prevent it from overlapping with the header.
        + Header:
        + A custom component (perhaps with navigation links) that appears at the top of the programme.
        + Routes inside the BrowserRouter:
        + There are three defined routes.
        + The default route ('/') displays the Home component.
        + The route 'cart' displays the Cart component.
        + The route 'product/:id' displays the DetailView component, where:id is a dynamic argument that represents the product ID.
        + TemplateProvider, ContextProvider:
        + Custom providers for managing themes and context in your app.
        + Home, Cart, Detailview:
        + These are components that are rendered according to the matched route.
        + Overall, this design creates a straightforward navigation framework for your React application. The BrowserRouter lets you specify distinct views for different routes, while the Header component acts as the navigation header. The TemplateProvider and ContextProvider most likely offer global themes and context for your application.

**Back-end Part :**

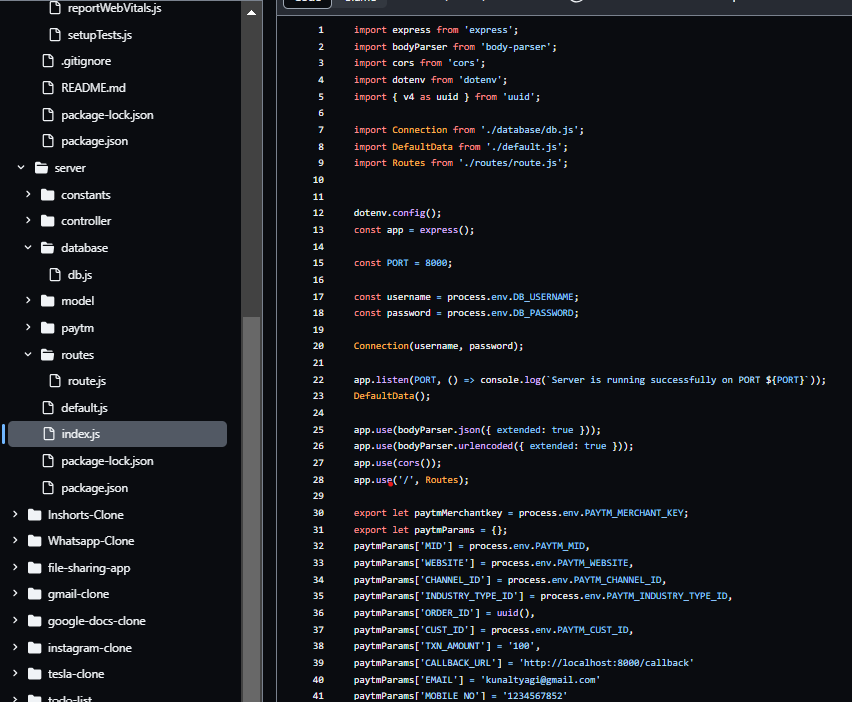
For back-end part we’ve here created a server folder inside the main web warp-clone folder where we’ve created **index.js** file. Also installed nodemon using **npm install nodemon.**

**The components used here are :**

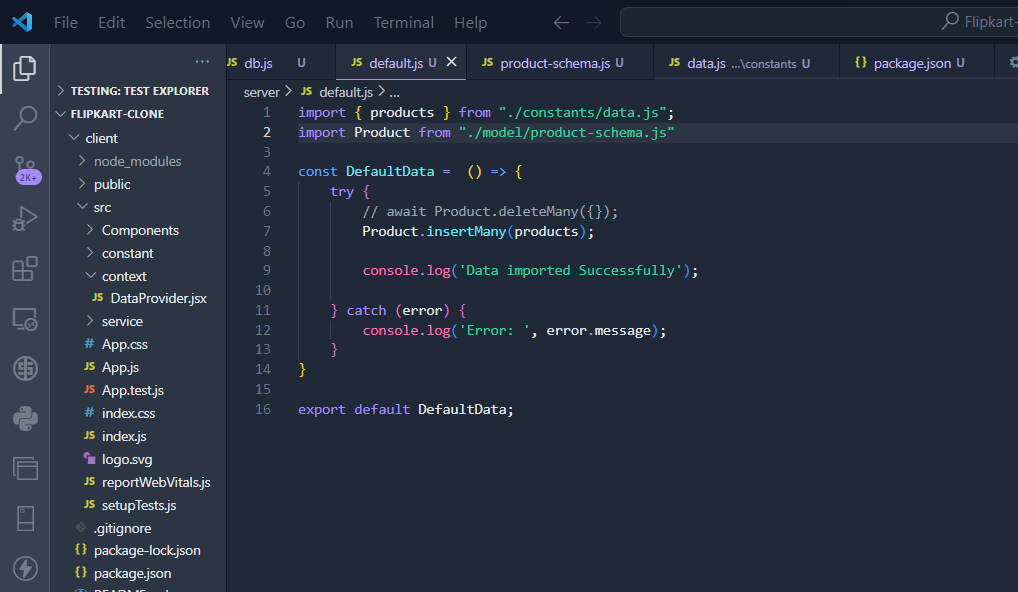


Back-end side components uses :

* + Constants : Inside components we’ve product.js file
  + Controllers : Inside components there are 4 files named cart-controller.js , payment.js , product-controller.js and user-controller.js
  + database : Inside context file we’ve db.js file
  + model : Inside this folder we’ve cartSchema.js , productSchema.js and userSchema.js
  + paytm : Inside redux we’ve paytmchecksum.js and sample.js
  + routes : Inside service file we’ve route.js
  + And we’ve files i.e. default.js and index.js
    - **Index.js**



* + - * + To begin, the code uses the ES6 import syntax to import required modules. These modules include express, body-parser, cors, dotenv, and the uuid library's v4 function. Each of these modules has a distinct role to play in the development of an Express.js server.
        + To configure environment variables, use the dotenv module to load them from a.env file into the process.env. This allows you to keep sensitive data, such as database connection information or API keys, away from the code.
        + To set up an Express application, the code generates an instance.
        + Database Connection:
        + It appears that the Connection method is used to make a database connection, providing in the DB\_USERNAME and DB\_PASSWORD values from environment variables.
        + To start the Express server, set it to listen on a certain port (PORT) and use a callback function to record a message upon successful startup.
        + Default Data Initialization: The DefaultData function is invoked to either initialise default data or execute setup chores. The code sample does not describe the intricacies of this method.
        + Middleware Setup: The Express app uses many middleware methods. Middleware functions have access to the request object (req), the response object (res), and the following middleware function in the application's request-response cycle.
        + Routing Setup: The code creates routes using the Routes module. The specific implementation of these routes is not shown in the supplied sample.
        + Overall, this code is a basic setup for an Express.js server with middleware configuration, environment variable usage, and potential database connection. The specific behavior of routes and the implementation of the **Connection** and **DefaultData** functions would require further inspection of their respective code files.
    - **DATABASE**
      * + Import mongoose from'mongoose';
        + const connection = async(username, password) => { // const URL ='mongodb://${username}:${password}@ecommerceweb-shard-00-00.fdvft.mongodb.net:27017,ecommerceweb-shard-00-01.fdvft.mongodb.net:27017,ecommerceweb-shard-00-02.fdvft.mongodb.net:27017/ECOMMERCE?ssl=true&replicaSet=atlas-8a6bhp-shard-0&authSource=admin&retryWriteSet=atlas-sjmqa0-shard-0&authSource=admin&retryWrites=true&w=majority'
        + try { await mongoose.connect.(URL, { useUnifiedTopology: true, useNewUrlParser: true, useFindAndModify: false }); console.log('Database connected successfully'); } catch(error) { console.log('Error: ', error.message); }};
        + Export the default Connection
    - **ROUTES**
      * + This code creates an Express.js router to handle multiple routes such as user login, product retrieval, and payment processing. Let us break down the important components:
        + Importing required modules:
        + To establish an Express router, the code first imports the express module.
        + It imports certain functions (getProductById, getProducts, userSignUp, userLogIn, addPaymentGateway, paymentResponse) from the respective controller files in '../controller/'. These functions manage the logic for certain routes.
        + The code uses express to construct an Express router instance.Router().
        + Routes are defined using HTTP methods (e.g., POST, GET) and assigned to certain controller functions.
        + Route handlers are defined for user authentication (/signup and /login), product retrieval (/products and /product/:id), and payment processing (/payment and /callback).
        + Exporting the Router: The router may be utilised in other areas of the programme.
        + This router is designed to be embedded into an Express application and specifies the routes that the application will use. Each route is coupled with a single controller function, which separates routing and business logic concerns. Uncommenting cart-related routes may require importing the necessary controller function and adding the route accordingly.
    - **Default.js**



* + - * + This code exports a method DefaultData, which is in charge of populating a MongoDB collection (assumed to be represented by the Product model) with default values. Let us break down the important components:
        + The code uses the Product model from './model/productSchema.js'. This model is intended to depict the structure of documents in a MongoDB collection connected to products.
        + It imports the constant products from './constants/product.js', which appears to contain an array of default product data.
        + Default Data Function:
        + The DefaultData function is asynchronous, which means that it may do asynchronous actions such as database queries.
        + Inside the function:
        + To destroy all documents in the Product collection, execute Product.deleteMany({}).
        + The default product data is then inserted into the collection using Product.insertMany(products).
        + If the operations are completed successfully, a success message is reported; otherwise, an error message is recorded.
        + Exporting a Function:
        + The DefaultData function is exported as the module's default export.
        + This module is most likely designed to be used as a script or as part of the initialization process for a Node.js application. When this script is executed, it clears the current data in the Product collection and replaces it with the default product data supplied in the products constant.

**OUR WEBSITE**