Thought Note for E-Commerse(ECom) Backend Project:-

# Front-end Part:-

1. Components:-
   1. App.js:-
      1. Imports: The component imports various necessary modules, components, and pages required for the application, including stylesheets, routes, and individual page components.
      2. Routing: It sets up routing using BrowserRouter and Routes from react-router-dom. Each Route component defines a specific URL path and the corresponding component to render when that path is matched.
      3. Routes Configuration: Each Route specifies a path and the component to render when that path is matched. For example:
      4. The / path renders the Login component.
      5. The /AllProductPage path renders the AllProductPage component.
      6. Similarly, there are routes for different pages such as Contact, Paypal, About, ShippingScreen, CartPage, ProductPage, SignIn, HomePage, Loading, OrderHistory, Profile, Adminmenu, Searchscreen, PlaceOrderScreen, and a catch-all route for any unmatched paths, which renders the ErrorPage component.
      7. Default Element: The BrowserRouter wraps the Routes component, which contains all the individual Route components. This structure defines the routing hierarchy for the entire application.
   2. Header.js:-

The header comprises various sections:

User Authentication: It checks for user authentication status stored in the local storage. If authenticated, it displays a welcome message along with the user's name and profile picture. If not, it provides a Sign In link.

Navigation Links: Links to different sections of the application like Home, About, Products, Contact, and Cart are provided.

Conditional Rendering: Certain links, like the Admin Page link, are conditionally rendered based on the user's role.

Logout Functionality: Logging out clears user authentication data from local storage and dispatches a logout action.

Responsive Design: The component is designed to be responsive, with a modal dialog appearing on smaller screens for easier navigation.

* 1. Footer.js:-
     1. This Footer component provides a structured and visually appealing bottom section for the application. It's divided into several sections, including product pages, about section, sign-up option, and useful links. Each section is organized with clear headings and links. The component's design is responsive, ensuring usability across different screen sizes. Additionally, it includes copyright information for proper attribution. Overall, it enhances the user experience by providing easy access to important sections of the application.
  2. Checkoutwizard.js:-
     1. The CheckoutWizard component is designed to provide a visual representation of the steps in the checkout process. It takes the current step as a prop and dynamically renders each step along with its corresponding number and name. The steps are defined as an array within the component, allowing for easy customization. Additionally, a divider is displayed between steps, except for the last step. Overall, this component enhances the user experience by offering clear guidance through the checkout process.
  3. LoadingComponent.js:-
     1. The Loading component is a simple yet effective way to provide feedback to users that content is being loaded. It displays a spinner animation centered within a container, indicating that something is happening behind the scenes. This component enhances user experience by managing expectations and reducing frustration during loading periods, contributing to a smoother interaction with the application.

1. Redux:-
   1. Combine\_reducer.js
   2. Store.js
   3. userReducer.js
2. Pages:
   1. About.js:-
      1. The About component is designed to provide information about the developer. It includes a header, a section describing the developer's background and interests, and an image. The component fetches user data from local storage, presumably for personalized content. The layout is well-structured, with clear divisions between sections. Additionally, it includes a footer for consistent navigation. Overall, the component effectively communicates information about the developer while maintaining a visually appealing design.
   2. AllProduct.js:-
      1. This component efficiently handles the display of all available products. It effectively integrates with the backend API to fetch product data and utilizes React's state management to handle loading states. The navigation functionality allows for smooth transitions between different pages, enhancing user experience. The use of Bootstrap CSS ensures a clean and responsive layout, while the inclusion of a loading indicator provides visual feedback during data fetching. Overall, the code demonstrates a good understanding of React principles and best practices, resulting in a well-structured and functional component.
   3. Adminpage.js:-
      1. This component appears to be an admin panel for managing products and users. It allows for CRUD operations on products and users, utilizing Axios to interact with the backend API. The state management with useState hooks efficiently handles form inputs and data fetching/loading states. The use of modals enhances the user interface for adding/editing products and users exclusively. Overall, it seems well-structured and functional, providing essential functionalities required for an admin dashboard.
   4. cartPage.js:-
      1. In the CartPage component, the code manages the display of items in the shopping cart. It retrieves cart items from the server, calculates subtotal and total amounts, and provides options to delete individual items or clear the entire cart. The layout is designed to be user-friendly, with clear navigation for continuing shopping or proceeding to payment confirmation. The code structure is straightforward and well-organized, making it easy to understand and maintain.
   5. Contact.js:-
      1. Contact form component for a website. It imports necessary components like Header, Footer, and Loading for layout and functionality. The useEffect hook is used to simulate an API call, setting isLoading state to false after 2 seconds. The form is structured with inputs for email, username, and a textarea for the message. When submitted, it posts data to a specified endpoint. Overall, it provides a simple interface for users to send messages using the “formspree” website to send email.
   6. ErrorPage.js:-
      1. This component represents an error page (404) for when a user navigates to a non-existent route. It includes a header, a message indicating the page is not found, and a button to return to the homepage. The Loading component is rendered while waiting for the isLoading state to change.
   7. HomePage.js:
      1. This component represents the homepage of an e-commerce website. It fetches the top five products from the backend and displays them along with product categories and brands. The Loading component is rendered while waiting for the isLoading state to change.
   8. Login.js:-
      1. This Login component handles user authentication by allowing users to input their email and password. Upon submission, it sends a POST request to the server for authentication. The component displays a loading spinner while fetching data and shows error messages if login fails. It also provides a link to the sign-up page for new users. Overall, the code structure is clear, and the component follows best practices for handling user authentication in a React application.
   9. OrderHistory.js:-
      1. The OrderHistory component efficiently handles the retrieval and display of order history data. It uses state variables to manage loading state and stores the fetched order data. The component also includes error handling to notify users in case of any errors during data fetching. Overall, it follows a clean structure and effectively presents the order history information in a tabular format.
   10. PlaceOrderScreen.js:-
       1. This PlaceOrderScreen component handles the process of placing orders efficiently. It fetches cart items, calculates order summary, and displays shipping address, cart items, and order summary. Users can place orders by clicking the "Place Order" button, which triggers the placeOrder function and redirect to the PayPal payment Screen. Overall, the component provides a smooth user experience for placing orders.
   11. ProductPage.js:-
       1. Display Product details allow function to increase or decrease the quantity of the product

Redirect the page to shipping screen transfer data along. Review and comment option available by clicking on the review and comment text.

* 1. ProfilePage.js:-
     1. The Profile component appears to handle displaying user profile information, including their name, email, and profile image. Users can also edit their details through a modal, update their information, and navigate to their cart or order history. The component uses local state and useEffect hook for managing state and side effects.
  2. SearchPage.js:-
     1. The Searchscreen component is responsible for rendering a search interface where users can search for products based on their name and brand. It fetches the search results from an API endpoint and displays them as cards. Users can click on a product card to navigate to the product page for more details.
     2. The component utilizes local state to manage input fields for product name and brand, as well as to store the search results. It also uses useEffect to simulate loading while fetching data from the API. Overall, it provides a user-friendly search experience with a responsive design and clear navigation.
     3. Additionally, the component includes the Header and Footer components to maintain consistency across the application layout.
  3. ShippingPage.js:-
     1. The ShippingScreen component is responsible for rendering a form where users can input their shipping address details. It includes input fields for the full name, address, city, postal code, and country. Users can submit the form to save their shipping address and proceed to the next step of the checkout process.
     2. The component utilizes React's useState hook to manage state for each input field. It also uses useNavigate from React Router DOM to navigate to the next screen (PlaceOrderScreen) upon form submission, passing the shipping address data as state.
     3. Additionally, the component includes the Header and Footer components to maintain consistency across the application layout. It also displays a checkout wizard bar (CheckoutWizard) to indicate the current step in the checkout process.
  4. Signin.js:-
     1. This component is responsible for rendering a sign-up form where users can register by providing their full name, email, password, and optionally, an image URL. Upon successful registration, users are redirected to the login page. The component also handles form submission and displays loading spinners during processing.

# Back-End Thought Note:-

1. Payment\_controller:-
   1. Import Statements: The code imports necessary modules such as paypal-rest-sdk, path, express, and mongoose.
   2. Router Configuration: It configures an Express router for handling different routes related to payment processing.
   3. PayPal Configuration: The PayPal SDK is configured with mode (sandbox or live) and client credentials obtained from environment variables.
   4. Render Buy Page: Defines a function renderBuyPage to render the buy page. It sends the index.ejs file to the client.
   5. Pay Product: Defines a function payProduct to initiate the PayPal payment process. It constructs a payment object with details like intent, payer information, transaction details, and redirect URLs. Then, it calls the PayPal SDK's payment.create method to create the payment. If successful, it sends back the approval URL to the client.
   6. Success Page: Defines a function successPage to handle the redirect from PayPal after successful payment. It extracts the payer ID and payment ID from the query parameters, constructs an execute payment JSON object, and calls the PayPal SDK's payment.execute method to execute the payment. If successful, it saves the payment details to the database and redirects the user to the order history page.
   7. Cancel Page: Defines a function cancelPage to handle the redirect from PayPal after payment cancellation. It renders the cancel.ejs page.
   8. Export: Exports the functions renderBuyPage, payProduct, successPage, and cancelPage for use in other modules.
2. Protected\_resource middleware:-
   1. Import Statements: The code imports necessary modules such as jsonwebtoken and mongoose.
   2. JWT Secret: It imports the JWT secret from the application's configuration file.
   3. User Model: It imports the UserModel from Mongoose to interact with the user database.
   4. Authentication Middleware: The middleware function checks for the presence of the JWT token in the request headers. If the token is missing, it responds with a 401 status and an error message indicating that the user is not logged in.
   5. Extract Token: It extracts the token from the authorization header, removing the "Bearer " prefix.
   6. Verify Token: It verifies the JWT token using the JWT secret. If the verification fails, it responds with a 401 status and an error message indicating that the user is not logged in.
   7. Retrieve User: If the token is successfully verified, it extracts the user ID from the token's payload and queries the database to find the corresponding user.
   8. Attach User to Request: It attaches the user object retrieved from the database to the request object (req.user).
   9. Next Middleware: It calls the next() function to proceed to the next middleware or route handler in the request processing pipeline.
3. Models:
   1. Cart Model
   2. Product Model
   3. User Model
   4. Order Model
4. Routes
   1. Cart\_route.js:-
      1. GET /myallproducts: Retrieves all products in the cart associated with the logged-in user. It uses the CartModel to find all cart items and returns them as a JSON response.
      2. POST /createmyproduct: Creates a new product in the cart for the logged-in user. It expects parameters such as description, Quantity, Name, Brand, Price, and image in the request body. If any mandatory field is missing, it returns a 400 error. Otherwise, it creates a new CartModel object with the provided data and the user information obtained from the authentication middleware. After saving the new cart item, it returns the created product as a JSON response.
      3. DELETE /deleteMyproduct/:postId: Deletes a specific product from the cart. It first finds the cart item by its ID (postId) and ensures that the logged-in user is the author of the cart item. If the item is found and the user is authorized, it removes the item from the database and returns a success message.
      4. DELETE /deletewholeCart: Deletes all products from the cart. It removes all cart items from the database using CartModel.deleteMany() and returns a success message upon completion.
   2. Order\_route.js:-
      1. POST /myorder: This route is responsible for creating a new order. It expects the following data in the request body: cartItems (items in the cart), shippingAddress (address where the order will be shipped), and orderSummary (summary of the order, including total price). If any of these mandatory fields are missing, it returns a 400 error. Otherwise, it creates a new OrderModel object with the provided data and the user information obtained from the authentication middleware. After saving the new order, it returns the created order as a JSON response with status code 201.
      2. GET /orderHistory: This route retrieves all order history data. It uses the OrderModel to find all orders in the database and returns them as a JSON response with status code 200.
   3. Payment\_route:-
      1. Express Setup: It initializes the payment route using express().
      2. Body Parser Setup: It configures body parser middleware to parse JSON and URL-encoded request bodies.
      3. View Engine Configuration: It sets up the view engine as EJS and specifies the views directory.
      4. Importing Controller: It imports the payment controller module, which contains the logic for handling payment-related requests.
      5. Route Definitions:
      6. GET /render: This route renders the payment page.
      7. POST /pay: This route handles the payment process.
      8. GET /pay/success: This route handles successful payments.
      9. GET /pay/cancel: This route handles canceled payments.
      10. GET /config: This route sends the PayPal client key as a response.
      11. Exporting Route: It exports the configured payment route for use in other parts of the application.
   4. Product\_route.js:-
      1. GET /allproducts: Retrieves all products stored in the database.
      2. PUT /ProductEdit/:ProductId: Updates a product with the specified ID.
      3. POST /createproduct: Creates a new product. Accessible only to administrators.
      4. PUT /comment: Allows users to add comments and ratings to products.
      5. GET /commented/:postId: Retrieves comments for a specific product.
      6. GET /topfour: Retrieves the top four products.
      7. DELETE /deleteadminproduct/:postId: Deletes a product. Accessible only to administrators.
      8. GET /searchedproducts/:productName/:brand: Searches for products based on name and brand.
   5. User\_route.js:-
      1. POST /signup: Allows users to sign up by providing their full name, email, password, and optional profile image. Passwords are hashed before being stored in the database to enhance security.
      2. POST /login: Enables users to log in with their email and password. Upon successful authentication, a JSON Web Token (JWT) is generated and returned, along with user information, including ID, email, full name, and profile image.
      3. GET /allusers: Retrieves all registered users from the database. Accessible only to administrators.
      4. DELETE /deleteadminuser/:postId: Deletes a user from the database. Accessible only to administrators.
      5. PUT /userEdit/:userId: Allows users to update their profile information, including full name, email, and profile image. Accessible to both administrators and logged-in users.
5. Server.js:-
   1. Environment Configuration: The application loads environment variables from a .env file using the dotenv package.
   2. Database Connection: It connects to a MongoDB database using Mongoose. The connection URL is retrieved from the environment variables.
   3. Middleware Setup:
   4. cors: Cross-Origin Resource Sharing middleware is used to enable CORS for all routes.
   5. express.json(): Middleware to parse incoming JSON requests.
   6. Route Handling:
   7. Routes for user-related operations are handled by user\_routes.js.
   8. Routes for managing the shopping cart are handled by cart\_route.js.
   9. Routes for handling product-related operations are managed by product\_route.js.
   10. Routes for managing orders are handled by order\_route.js.
   11. Routes for handling payments are managed by payment\_route.js.
   12. Server Initialization: The Express application listens on port 4000 for incoming requests.
   13. Database Connection Event Handlers:
   14. connected: Logs a message when the database connection is established successfully.
   15. error: Logs an error message if there's a problem connecting to the database.

Inserting Some Pictures of the project





