**Frontend Development with React.js - Project Documentation**

**1. Introduction:**

**Project Title: CryptoVerse**

**Project Description:**

CryptoVerse is a user-friendly cryptocurrency dashboard built with React.js. It allows users to search for different cryptocurrencies and view their real-time prices, as well as historical price graphs. The data is sourced from the CoinRanking API (via RapidAPI), offering accurate and up-to-date cryptocurrency information. With an intuitive interface and smooth navigation, Crypto Verse is designed to help users easily track the crypto market and make informed decisions.

**Team Members:**

* **Team Leader:** MUKESH S
* **Team Member:** JOEL JEBASINGH J
* **Team Member:** GOWTHAM S
* **Team Member:** DEEPAK R
* **Team Member:** TANUJ MYTREAN S

**Project Overview**

**Purpose**

The purpose of the **CryptoVerse** project is to provide a comprehensive, user-friendly cryptocurrency dashboard that allows users to easily track real-time cryptocurrency prices and historical trends. With an intuitive interface and powerful search and graphing features, CryptoVerse aims to help users make informed decisions in the volatile cryptocurrency market. The project utilizes React.js for dynamic, responsive frontend development and integrates the CoinRanking API to provide accurate and up-to-date data.

**Features**

* **Search Cryptocurrencies:** Users can search for any cryptocurrency by name or symbol and get instant results.
* **Real-Time Cryptocurrency Prices:** Displays live, real-time price updates for a wide range of cryptocurrencies.
* **Historical Price Graphs:** Allows users to view historical price data in graph form for any selected cryptocurrency, helping them to track market trends over time.
* **Responsive Design:** Optimized for both mobile and desktop screens, providing a seamless experience across devices.
* **User-Friendly Interface:** A clean and intuitive design that makes it easy for users to navigate and find the information they need.

**Architecture**

**Component Structure**

The frontend of CryptoVerse is built with React.js, with key components structured to ensure reusability and maintainability. Below is an outline of the main components:

1. **App Component:**  
   The root component that houses all other components and sets up global routing.
2. **SearchBar Component:**  
   Handles the user input for searching cryptocurrencies by name or symbol. It communicates with the parent component to trigger data fetching.
3. **CryptocurrencyList Component:**  
   Displays the list of cryptocurrencies with key data such as name, symbol, and current price. It uses data passed from the parent component and dynamically renders the list based on the search query.
4. **CryptoDetails Component:**  
   Displays detailed information about a selected cryptocurrency, including historical price graphs. It handles the visualization of the cryptocurrency data in the form of graphs using a graphing library like Recharts or Chart.js.
5. **GraphComponent (or PriceGraph):**  
   Displays the historical price graph of a selected cryptocurrency, allowing users to track price fluctuations over a defined time period (e.g., daily, weekly, monthly).
6. **Loader Component:**  
   Shows a loading spinner or message while data is being fetched from the CoinRanking API.
7. **ErrorBoundary Component (Optional):**  
   A higher-order component to catch JavaScript errors in any part of the UI and display an error message.

**State Management**

CryptoVerse uses **React's Context API** for state management to ensure a clean and efficient flow of data between components. Here's how it's used:

* **Global State Management:**  
  The **App Component** serves as the central hub, passing down state via the Context API to child components. For example, the state containing the list of cryptocurrencies, their details, and selected graph data is stored in a **CryptocurrencyContext** and accessed by components like **SearchBar**, **CryptocurrencyList**, and **CryptoDetails**.
* **State Updates:**  
  When a user searches for a cryptocurrency, the data is fetched from the CoinRanking API, and the state is updated with the response. This triggers re-rendering of the relevant components (e.g., the cryptocurrency list and graph).

**Routing**

CryptoVerse uses **React Router** for client-side routing, allowing users to navigate between different sections of the application without reloading the page. Here's an overview of the routing structure:

* **React Router Setup:**  
  The **App Component** contains the main routing logic, using the component to define different routes.
* **Home Route (/):**  
  The homepage displays a list of cryptocurrencies with real-time price data and a search bar.
* **Crypto Details Route (/crypto/:id):**  
  When a user clicks on a cryptocurrency from the list, they are directed to a detailed page showing the cryptocurrency's historical price graph and additional information.
* **Dynamic Routing:**  
  The **CryptoDetails** component uses the cryptocurrency ID from the URL (/crypto/:id) to fetch the specific data of the selected cryptocurrency and display its details.

**Setup Instructions**

**Prerequisites**

Before starting with the project setup, ensure you have the following software installed:

* **Node.js** (Version 14 or later)
  + Download and install from: [Node.js official website](https://nodejs.org/)
* **npm** (Node Package Manager) comes bundled with Node.js, so no need to install it separately.
* **Git** (Optional, if you want to clone the repository using Git)
  + Download and install from: [Git official website](https://git-scm.com/)

**Installation**

Follow these steps to get the project up and running locally:

1. **Clone the Repository** First, clone the project repository to your local machine
2. **Navigate to the Project Directory** Change your directory to the cloned project folder
3. **Install Dependencies** Run the following command to install all the necessary dependencies listed in package.json
4. **Configure Environment Variables** Create a .env file in the root of the project (if it doesn’t already exist). Add the following environment variables to access the CoinRanking API
5. **Start the Development Server** After the dependencies are installed and environment variables are configured, start the development server:
6. **Access the Application** Open your browser and go to http://localhost:3000 to view the application running locally.

**Folder Structure**

The folder structure of the **CryptoVerse** React application is organized to ensure scalability and maintainability:

**Client**

* **src**  
  This is the main source folder containing all the front-end code for the project.
  + **components**  
    Contains reusable UI components such as buttons, forms, loaders, and individual cryptocurrency card components.
    - Example:
      * SearchBar.js – Handles cryptocurrency search input and submission.
      * CryptoCard.js – Displays individual cryptocurrency information (name, symbol, price).
      * Loader.js – Displays a loading spinner while waiting for data from the API.
  + **pages**  
    Contains the different pages (views) of the app. These could be routed views such as the homepage or a specific cryptocurrency details page.
    - Example:
      * HomePage.js – Displays the list of cryptocurrencies and search functionality.
      * CryptoDetailsPage.js – Displays detailed information and price graph for a selected cryptocurrency.
  + **assets**  
    Stores static assets like images, icons, or fonts used in the project.
    - Example:
      * logo.png – The application logo.
      * cryptocurrency-icons – Icons for different cryptocurrencies.
  + **context**  
    Contains React Context Providers for managing global state in the app.
    - Example:
      * CryptocurrencyContext.js – Provides the cryptocurrency data and functions to manage the state (search results, selected cryptocurrency).
  + **styles**  
    Stores global CSS/SCSS files or styled components for styling the app.
    - Example:
      * App.css – General styling for the app layout and components.
      * Responsive.css – Specific styles for making the app responsive on mobile and desktop.
  + **utils**  
    Contains utility functions and custom hooks to help with tasks like API calls and data formatting.
    - Example:
      * api.js – Contains functions to fetch data from the CoinRanking API using Axios or fetch.
      * useFetch.js – A custom hook for fetching API data with state management for loading and error handling.
      * formatCurrency.js – A utility function to format cryptocurrency values into a more readable format (e.g., adding commas for thousands).

**Utilities**

* **Custom Hooks:**
  + **useFetch.js**  
    A custom hook for fetching data from the CoinRanking API and handling loading states, errors, and the API response:
  + **useCurrencySearch.js**  
    A custom hook to handle search queries and update the results dynamically based on user input.
* **Helper Functions:**
  + **formatCurrency.js**  
    A utility function to format numerical values into a more user-friendly format for currency display:
* **API Call Functions:**
  + **api.js**  
    Contains functions to make API calls to the CoinRanking API and return data. For example, a function to get cryptocurrency prices

**Running the Application**

**Frontend**

To run the frontend server locally, follow these steps:

1. **Navigate to the Client Directory:** Open your terminal and go to the **client** directory of the project:
2. **Install Dependencies:** Install all the necessary dependencies by running:
3. **Start the Development Server:** After the dependencies are installed, start the development server using:
4. **Access the Application:** Once the server starts, open your browser and navigate to http://localhost:3000 to view the application.

**Component Documentation**

**Key Components**

1. **SearchBar.js**
   * **Purpose:** This component is used to allow users to search for cryptocurrencies by name or symbol. It includes an input field for text entry and a button to trigger the search.
   * **Props:**
     + searchQuery (string): The current text value entered in the search input field.
     + setSearchQuery (function): A function passed to update the searchQuery state when the user types into the input.
     + onSubmit (function): A function passed to handle the form submission (usually triggers an API call to fetch cryptocurrencies based on the search term).
2. **CryptocurrencyList.js**
   * **Purpose:** Displays a list of cryptocurrencies with their basic information like name, symbol, and price. It dynamically updates as the search query changes.
   * **Props:**
     + coins (array): Array of cryptocurrency objects that contain data such as name, symbol, price, and other relevant information.
     + loading (boolean): A flag indicating if data is still being fetched from the API.
     + error (string): An error message to display if the API request fails.
3. **CryptoDetails.js**
   * **Purpose:** Displays detailed information about a selected cryptocurrency, including its name, symbol, price, and historical price graph.
   * **Props:**
     + cryptoData (object): Detailed information about the selected cryptocurrency, including current price, market cap, etc.
     + graphData (array): Historical data used to generate the cryptocurrency price graph, typically an array of data points over time.
4. **Loader.js**
   * **Purpose:** A reusable component that displays a loading spinner or message when the app is waiting for data from the API.
   * **Props:**
     + None (just a UI element to show the loading state).

**Reusable Components**

1. **CryptoCard.js**
   * **Purpose:** A small, reusable card component that displays brief information about a cryptocurrency, such as its name, symbol, and current price.
   * **Props:**
     + coin (object): Contains data for a single cryptocurrency (e.g., name, symbol, current price).
     + onClick (function): A function to handle the click event when a user clicks the card. This typically navigates to the details page for that specific cryptocurrency.
2. **Button.js**
   * **Purpose:** A reusable button component used throughout the application for different actions like submitting forms or navigating between pages.
   * **Props:**
     + text (string): The text to display inside the button.
     + onClick (function): The function to execute when the button is clicked.
     + className (string): A CSS class for styling the button (optional).

**State Management**

**Global State**

In this project, **global state** is managed using **React Context API**. The global state allows components to share data across the application without passing props down manually to each child component. Here's how state flows in the app:

1. **CryptocurrencyContext.js**:  
   This file contains the context provider that encapsulates the global state. It includes variables like the list of cryptocurrencies (coins), selected cryptocurrency data (selectedCrypto), loading status (loading), and error messages (error).
2. **State Flow:**
   * **App Component**: The root component of the application that wraps the entire app inside the context provider. This enables all child components to access the global state.
   * **SearchBar & CryptocurrencyList**: These components use the coins state to filter and display the list of cryptocurrencies. The search query triggers an update to the coins state when the user enters a cryptocurrency name or symbol.
   * **CryptoDetails**: The selected cryptocurrency data is passed to this component through the global state. This ensures that, when a user clicks on a cryptocurrency, it displays detailed information about that coin, such as its name, symbol, price, and historical price graph.
3. **Flow Example:**
   * When a user submits a search query in the SearchBar component, it triggers an API call to fetch cryptocurrency data. The coins state is updated globally, and this updated data flows to the CryptocurrencyList component for rendering.
   * If a user selects a cryptocurrency from the list, the selectedCrypto state is updated, and this data flows to the CryptoDetails component to display the cryptocurrency’s details.

**Local State**

Local state is used within components for managing temporary data that is specific to that component. It is not shared across the app. Here are examples of how local state is used:

1. **SearchBar.js**:
   * **State**: The searchQuery state holds the current search input. This state is used to update the input field dynamically and trigger the search function when the form is submitted.
   * **Handling**: The searchQuery state is updated whenever the user types in the input field. Once the user submits the form, the local state is used to trigger a function that updates the global state with the filtered cryptocurrency data.

Example:

js

Copy

const [searchQuery, setSearchQuery] = useState('');

const handleSearchChange = (e) => {

setSearchQuery(e.target.value);

};

1. **CryptoDetails.js**:
   * **State**: The selectedTimeFrame state is used to store the selected time period for the cryptocurrency price graph (e.g., 24h, 7d, 30d). It is local to this component because it doesn’t affect other parts of the application.
   * **Handling**: The local state allows users to change the time frame and update the displayed price graph dynamically.

Example:

js

Copy

const [selectedTimeFrame, setSelectedTimeFrame] = useState('24h');

const handleTimeFrameChange = (timeFrame) => {

setSelectedTimeFrame(timeFrame);

};

**User Interface**

**Screenshots / GIFs**

While I can’t generate actual screenshots or GIFs, here’s a description of the user interface features and where you can add your own visuals:

1. **Homepage (Cryptocurrency List Page)**:
   * **Search Bar**: A large input field at the top allows users to search for cryptocurrencies by name or symbol.
   * **Cryptocurrency List**: A grid or list of cryptocurrency cards displaying the name, symbol, and current price of each coin. Each card is clickable, which redirects the user to the cryptocurrency details page.
   * **Loading State**: When data is being fetched, a loading spinner or message is displayed.

**Screenshot Example**:

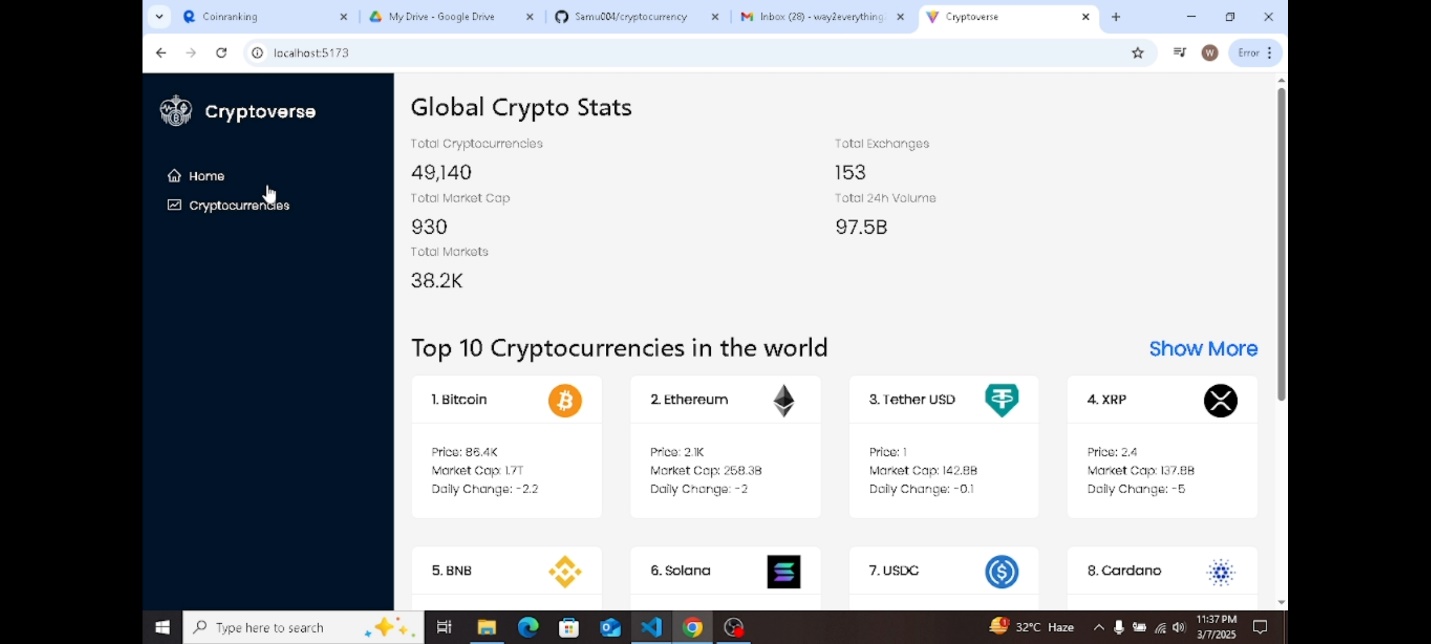
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot showing the search bar at the top and a list of cryptocurrency cards below, with a loading spinner on the side if the data is being fetched.

1. **Cryptocurrency Details Page**:
   * **Graph**: A graph displays the historical price data for the selected cryptocurrency, allowing users to select different time frames (e.g., 24h, 7d, 30d).
   * **Detailed Info**: Information about the selected cryptocurrency, such as its market cap, current price, and volume, is displayed in a clean, readable format.
   * **Back Button**: A button that allows users to navigate back to the main page.

**Screenshot Example**:



A screenshot showing the detailed view of a selected cryptocurrency, including the price graph, price data, and a back button.

**Styling**

**CSS Frameworks/Libraries**

In this project, **custom CSS** is primarily used for styling the application. We did not rely on any pre-built CSS frameworks like Bootstrap or Material UI. However, we organized the styles in a modular way to ensure better maintainability and readability.

* **CSS:** All the styles are written using plain CSS files. For example:
  + **App.css**: Contains global styles for the app (e.g., layout, typography, colors).
  + **Component-specific CSS files**: Each component has its own CSS file (e.g., SearchBar.css, CryptoCard.css) for component-specific styling.

Custom classes and selectors are used to ensure styles are applied correctly to specific components.

**Color Scheme**

The primary color combination for the app is **dark blue and white**, which provides a clean and professional look, with good contrast for readability.

* **Primary Color**:
  + **Dark Blue** (#0D3B66): This color is used for header backgrounds, buttons, and links to ensure a strong presence and maintain readability.
* **Secondary Color**:
  + **White** (#FFFFFF): The background color and text are primarily white, creating a high-contrast look that makes the content stand out.
* **Accent Colors**:
  + **Light Blue** (#1F77B4): Used for hover effects and accent elements like buttons or selected items to add vibrancy and interactivity.
  + **Gray** (#F5F5F5): A light gray color used for secondary background areas to help the dark blue elements stand out.

**Theming**

Currently, there is no comprehensive **theming system** or dynamic theming (e.g., dark mode) implemented in the project. The styling is primarily based on a **fixed color scheme** of dark blue and white to provide a consistent and clean look.

However, if future updates are required, a theming system could be introduced using CSS variables, or a library like **Styled Components** or **Sass** could be incorporated for more flexibility in theming.

**Typography**

The typography of the app follows a simple design:

* **Primary Font:** Arial, Helvetica, sans-serif for clean, readable text.
* **Headings and Titles:** Text elements like headings are bold and slightly larger for clarity, typically in dark blue (#0D3B66) to stand out.

**Buttons & Cards**

* **Buttons:** Styled with a **light blue** background (#1F77B4) and **white text**. They also have hover effects that transition to a **dark blue** color (#0D3B66) to show interactivity.
* **Cards:** The cryptocurrency cards have a **white background** and a subtle **shadow effect**, with a **hover effect** that changes the background to light gray to highlight them when users interact with them.