

A CRYPTOCURRENCY DASHBOARD

Team Members:

S. Hema Prava Sethi - Team Leader- shemapravasethi@gmail.com

Amrah Iffath - Team Member- amrahiffath3@gmail.com

Fizrah Ismath - Team Member- fizrahismath786@gmail.com

E. Dharani - Team Member- dprofile70007@gmail.com

Introduction:

A crypto currency dashboard that displays historical price data over the past five years is a powerful tool for investors seeking a comprehensive understanding of market dynamics.

This feature-rich interface offers users a detailed historical perspective on the performance

of various crypto currencies, enabling insightful analysis and informed decision-making.

Through visually intuitive charts and graphs, the dashboard allows for effective

comparisons of multiple crypto currencies, aiding in the identification of top performers and

overall market trends. Users can customize timeframes for a more granular examination of

price movements, facilitating in-depth volatility analysis and risk assessment. This historical

data not only supports investors in making data-driven decisions but also assists in

recognizing recurring patterns and cycles. Beyond its role in optimizing cryptocurrency

portfolios, the dashboard serves as an educational resource, empowering users to grasp the

evolving nature of crypto currency markets and the nuanced factors shaping price

movements over an extended period.

Project Overview

Purpose:

A cryptocurrency dashboard project serves as a centralized platform for tracking, analysing, and managing digital assets in real-time. It provides users with live market data, portfolio management tools, price alerts, and interactive charts to visualize trends and make informed investment decisions. Additionally, it can integrate news updates, technical indicators, and DeFi insights to help traders, investors, and enthusiasts stay ahead in the fast-paced crypto market. By offering a user-friendly interface with customization options, the dashboard enhances accessibility and simplifies complex data, making it an essential tool for anyone engaged in the cryptocurrency ecosystem.

Features:

1. Real-Time Market Data

- Live updates on cryptocurrency prices, market cap, and trading volume.

- Fetch data from APIs like Coin Gecko, CoinMarketCap, or Binance.

2. Portfolio Management

- Track users' crypto holdings with real-time profit/loss calculations.
- Display asset allocation and total portfolio value.

3. Data Visualization & Charts

- Interactive candlestick charts, line graphs, and historical price trends.
- Technical indicators like Moving Averages, RSI, and MACD.

4. Alerts & Notifications

- Set price alerts for specific cryptocurrencies.
- Receive notifications when a coin reaches a target price.

5. News & Market Sentiment Analysis

- Fetch latest crypto news from sources like Coin Telegraph or Twitter.
- Display market sentiment using AI-based analysis.

Architecture:

App

| -- Layout

| | — Header

| | — Sidebar

| | — Main Content

| -- Pages

| | — Dashboard

```
|   |—— Portfolio
|   |—— Market
|   |—— Settings
|-- Components
|   |—— Crypto Chart
|   |—— Price List
|   |—— Portfolio Overview
|   |—— Market Trends
|   |—— Trade Form
|-- Services
|   |—— api.js (Handles API calls)
|-- State Management
|   |—— store.js (Redux/Zust and context)
|-- Utilities
|   |—— format.js (Helpers for currency, date)
```

State Management:

1. Context API (For Light State Management)

Best for: Small-scale dashboards with minimal global state needs.

◆ Use Case in a Crypto Dashboard:

- Theme toggling (dark/light mode).
- User authentication state.
- Simple UI preferences (e.g., selected currency: USD, EUR).

◆ How it Works:

- Uses React built-in Context API with use Context and use State.
- Wraps the app with a provider to share global state.

2. Redux Toolkit (For Complex State Management)

Best for: Medium to large-scale dashboards requiring global state, API integration, and WebSocket data handling.

◆ Use Case in a Crypto Dashboard:

- Managing real-time crypto price updates across multiple components.
- Handling user portfolio state (total holdings, transactions).
- Storing market trends data for various coins.

◆ How it Works:

- Uses Redux Toolkit (@reduxjs/toolkit) for efficient state management.
- Stores state globally and updates components via Redux store.
- RTK Query can be used for API calls instead of use Effect.

Routing:

In a cryptocurrency dashboard project, React Router is used to manage navigation between different sections such as the Dashboard (/), Market (/market), Portfolio (/portfolio), and Settings (/settings). The Layout component wraps all pages, providing a consistent structure with a Sidebar and Header, while nested routes allow dynamic paths like /market/:coin Id to display individual coin details. A Protected Route mechanism ensures secure access to the Portfolio page, redirecting unauthenticated users. Additionally, lazy loading improves performance by loading pages only when needed. The

wildcard route (*) handles 404 errors, ensuring a seamless user experience.

Setup Instructions

Pre- requisites:

- ❖ Here are the key prerequisites for developing a frontend application using React.js:
- ❖ Node.js and npm: Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications.
- ❖ React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.
- ❖ HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of the app, CSS for styling, and JavaScript for client-side interactivity.
- ❖ Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

Installation:

Clone the code from github repository:

Follow below steps: Git repository:

<https://github.com/SSC369/cryptoverse> Git clone command: `git clone https://github.com/SSC369/cryptoverse` Use this command to clone code into your project folder. Install Dependencies: • Navigate into the cloned repository directory and install libraries: `cd crypto npm install` Start the Development Server: • To start the development

server, execute the following command: `npm run dev (vite)` or `npm start`

Access the App: • Open your web browser and navigate to `http://localhost:3000`. • You should see the Cryptoverse app's homepage, indicating that the installation and setup were successful.

Folder Structure

Client:

1. `public/` – Static Assets
 - Stores static files like `index.html`, logos, and favicons.
 - Example: `/public/logo.png`
2. `src/assets/` – Images, Icons, Styles
 - Stores logos, icons, and CSS files.
3. `src/components/` – Reusable UI Components
 - Reusable, independent components used across pages.
4. `src/pages/` – Main Page Views
 - Stores page components (Dashboard, Market, Portfolio).
5. `src/features/` – State Management (Redux, Zustand)
 - Manages global state using Redux Toolkit or Zustand.
6. `src/hooks/` – Custom Hooks
 - Stores reusable hooks like API fetching or authentication.
7. `src/context/` – Context API for Global State
 - Stores Context API providers for theming or authentication.
8. `src/services/` – API Calls & WebSockets
 - Handles API requests and real-time WebSocket connections.
9. `src/routes/` – Route Configuration
 - Centralized routing logic.

Running the Application:

Using Create React App (CRA):

`npm install` # Install dependencies

`npm start` # Start the development server

The app will run at `http://localhost:3000/` by default.

Component Documentation

Key Components:

Navbar Component (Navbar.js)

 Purpose:

- Displays the site logo, navigation links, and a theme toggle button.
- Provides navigation between pages (Dashboard, Market, Portfolio).

Props:

The `onThemeToggle` function toggles between the light and dark mode.

Sidebar Component (Sidebar.js)

 Purpose:

- Provides a collapsible menu with quick access to different pages.
- Enhances navigation UX by highlighting the active page.

Props:

The `isOpen` Boolean function controls the sidebar visibility

Crypto Card Component (CryptoCard.js)

Purpose:

- Displays individual cryptocurrency details (price, market cap, change %).
- Used inside the Market Page to list multiple cryptos.

Market List Component (MarketList.js)

Purpose:

- Displays a list of cryptocurrencies using multiple CryptoCard components.
- Fetches real-time data from an API (e.g., CoinGecko).

State Management

Global State:

In a cryptocurrency dashboard, global state management ensures that data such as market prices, user portfolio, theme settings, and authentication status are accessible across multiple components. There are several ways to handle global state in React, including Context API, Redux Toolkit, or Zustand.

Local State:

Local state refers to the data that is managed within a specific component and is not shared globally. In a cryptocurrency dashboard, local states can control:

- User interactions (e.g., toggling dark mode)
- Loading states (e.g., showing a spinner while fetching data)

- Search and filter inputs (e.g., filtering cryptocurrencies by name or price range)
- Chart display options (e.g., selecting time intervals for price charts)

User Interface:



Styling:

CSS Frameworks

These frameworks provide pre-styled components, grid systems, and utilities to speed up development.

- Bootstrap – Offers a flexible grid system, responsive components, and utility classes for quick UI development.
- Tailwind CSS – A utility-first framework that allows for rapid customization while keeping styles modular.
- Bulma – A lightweight, modern framework with a clean and minimal design approach.

CSS Preprocessors

These preprocessors extend CSS functionality with variables, nesting, and mixins, making styles more manageable.

- Sass (SCSS) – Adds features like variables, loops, and mixins, which help in managing styles efficiently, especially for a large project.

Testing:

Unit Testing

Purpose:

- Tests individual components or functions in isolation.
- Ensures correctness of UI components, utility functions, and state management.

Tools:

- Jest – A powerful JavaScript testing framework.
- React Testing Library (RTL) – Focuses on testing React components as a user would interact with them.

Approach:

- Test React components by checking if they render correctly and handle props.

- Test utility functions (e.g., price calculation, data formatting).
- Mock API calls and context/state to isolate the test.

End-to-End (E2E) Testing

Purpose:

- Tests the entire user flow (e.g., login, view dashboard, buy crypto).
- Ensures UI works correctly in a real browser environment.

Tools:

- Cypress – Best for front-end user testing.
- Playwright – Fast, supports multiple browsers.

Approach:

- Simulate user actions (clicks, form submissions).
- Test navigation between pages.
- Validate real-time price updates.

Tools for Measuring Test Coverage

These tools analyse how much of your code is covered by tests, identifying untested areas.

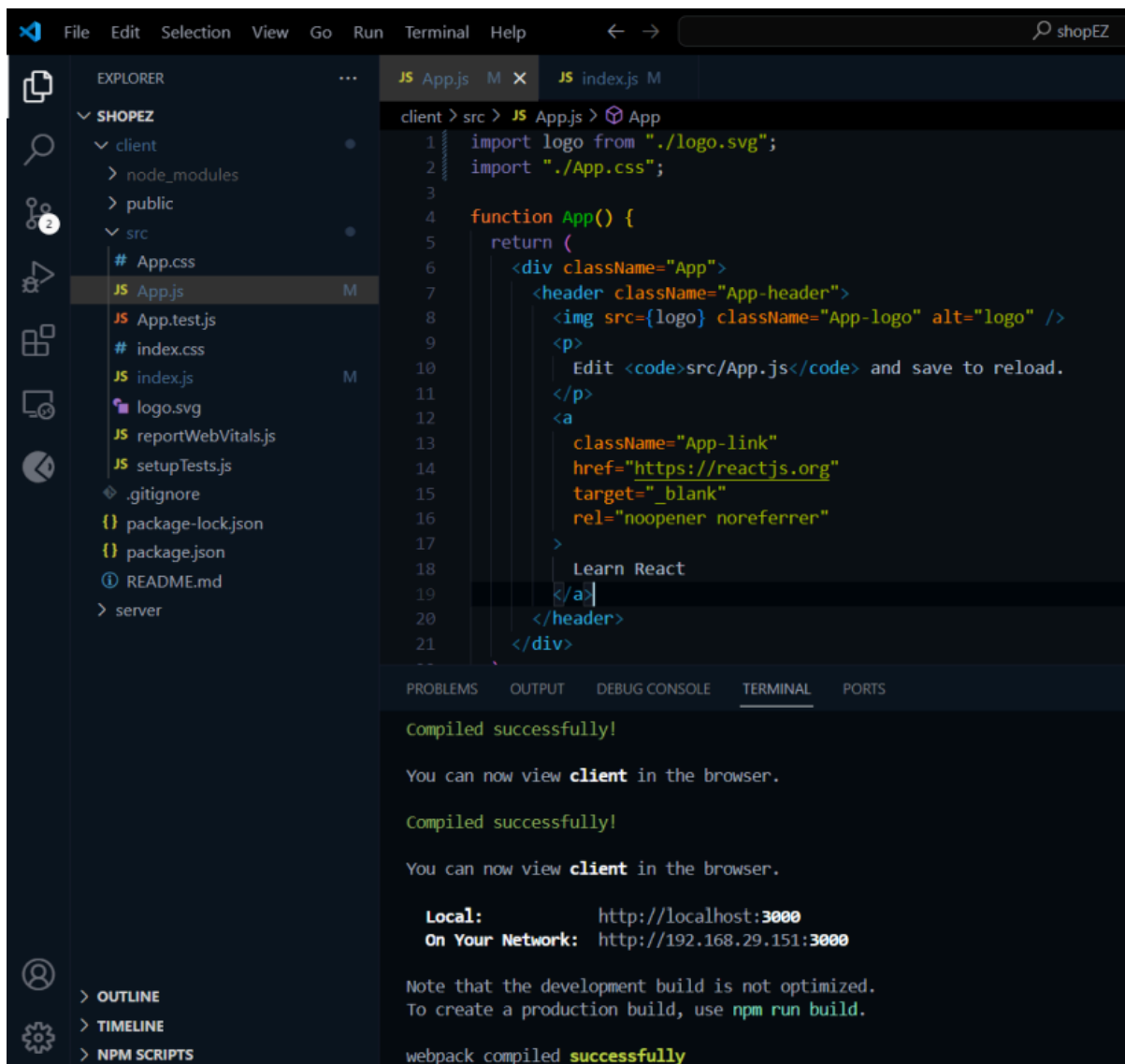
◆ Jest with Istanbul (Default Coverage Tool)

- Jest, the popular JavaScript testing framework, integrates Istanbul to generate test coverage reports.
- It highlights which lines, statements, functions, and branches are covered.

Screenshots:

PROJECT-1

- ▼ crypto
 - > dist
 - > node_modules
 - > public
 - ▼ src
 - ▼ app
 - JS store.js
 - ▼ assets
 - cryptocurrency.png
 - ▼ components
 - Cryptocurrencies.jsx
 - CryptoDetails.jsx
 - Home.jsx
 - JS index.js
 - LineChart.jsx
 - Loader.jsx
 - Navbar.jsx
 - ▼ services
 - JS cryptoApi.js
 - # App.css
 - App.jsx
 - # index.css
 - main.jsx
 - .env
 - .eslintrc.cjs
 - .gitignore
 - index.html
 - package-lock.json
 - package.json
 - README.md



Project Structure (Left Sidebar - Explorer)

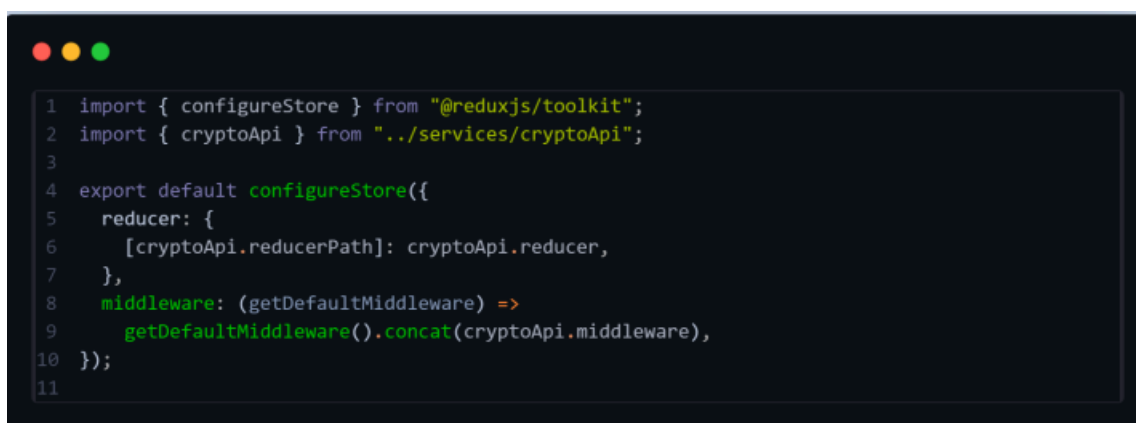
- The project seems to be named "SHOPEZ" and contains a client directory (likely a frontend project).
- Inside src/, key files include:
 - App.js: The main React component.
 - index.js: The entry point for rendering the app.
 - App.css: Styles for the application.
 - App.test.js: Likely for testing with Jest.
 - Other supporting files like setupTests.js, reportWebVitals.js, and logo.svg.

Code Editor (Center)

- App.js is open and contains a functional React component named App().
- It imports:
 - A logo (logo.svg).
 - A CSS file (App.css).
- The JSX structure includes:
 - A <header> with an image and a link to React documentation.

Terminal (Bottom Section)

- The React development server has successfully compiled and is running at:
 - Localhost: http://localhost:3000
 - On Network: http://192.168.29.151:3000
- A message confirms that Webpack compiled successfully, meaning no critical errors.

A screenshot of a code editor with a dark background and light-colored text. The code is written in a syntax-highlighted style. It shows the configuration of a Redux store using Redux Toolkit. The code includes imports for `configureStore` from `@reduxjs/toolkit` and `cryptoApi` from a local service. The `configureStore` function is called with a `reducer` object and a `middleware` function that concatenates the default middleware with `cryptoApi.middleware`. The code is as follows:

```
1 import { configureStore } from "@reduxjs/toolkit";
2 import { cryptoApi } from "../services/cryptoApi";
3
4 export default configureStore({
5   reducer: {
6     [cryptoApi.reducerPath]: cryptoApi.reducer,
7   },
8   middleware: (getDefaultMiddleware) =>
9     getDefaultMiddleware().concat(cryptoApi.middleware),
10 });
11
```

- Create a API slice using Redux toolkit's:
 1. Import Statements: - `import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";`: This line imports the necessary functions from Redux Toolkit's query-related module. `createApi` is

used to create an API slice, while `fetchBaseQuery` is a utility function provided by Redux Toolkit for making network requests using `fetch`.

2. Header and Base URL Conguration: - `const cryptoApiHeaders = { ... }`: This object contains headers required for making requests to the cryptocurrency API. The values for `"X-RapidAPI-Key"` and `"X-RapidAPI-Host"` are retrieved from environment variables using `import.meta.env`. - `const baseUrl = import.meta.env.VITE_BASE_URL;`: This variable holds the base URL for the cryptocurrency API, which is also retrieved from environment variables.

3. Request Creation Function: - `const createRequest = (url) => ({ url, headers: cryptoApiHeaders })`: This function `createRequest` takes a URL and returns an object with the URL and headers required for making a request. It utilizes `cryptoApiHeaders` to include necessary headers in the request.

4. Create API Slice: - `export const cryptoApi = createApi({ ... })`: This part uses the `createApi` function to create an API slice named `cryptoApi`. It takes an object with several properties: - `reducerPath`: Species the path under which the slice's reducer will be mounted in the Redux store. - `baseQuery`: Congures the base query function used by the API slice. In this case, it uses `fetchBaseQuery` with the base URL specied. - `endpoints`: Denes the API endpoints available in the slice. It's an object with keys corresponding to endpoint names and values being endpoint denitions.

5. API Endpoints: - `getCryptos`, `getCryptoDetails`, `getCryptoHistory`: These are endpoints dened using the `builder.query` method. Each endpoint is congured with a `query` function that returns the request conguration object created by `createRequest`.

6. Exporting Hooks: - `export const { ... }`: This line exports hooks generated by the `createApi` function, allowing components to easily

fetch data from the API slice. Each hook corresponds to an endpoint dened in the `endpoints` object.


```
1 import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";
2
3 const cryptoApiHeaders = {
4   "X-RapidAPI-Key": import.meta.env.VITE_RAPID_API_KEY,
5   "X-RapidAPI-Host": import.meta.env.VITE_RAPID_API_HOST,
6 };
7
8 const baseUrl = import.meta.env.VITE_BASE_URL;
9
10 const createRequest = (url) => ({ url, headers: cryptoApiHeaders });
11
12 export const cryptoApi = createApi({
13   reducerPath: "cryptoApi",
14   baseQuery: fetchBaseQuery({ baseUrl }),
15   endpoints: (builder) => ({
16     getCryptos: builder.query({
17       query: (count) => createRequest(`/coins?limit=${count}`),
18     }),
19
20     getCryptoDetails: builder.query({
21       query: (coinId) => createRequest(`/coin/${coinId}`),
22     }),
23     getCryptoHistory: builder.query({
24       query: ({ coinId, timePeriod }) =>
25         createRequest(`/coin/${coinId}/history?timePeriod=${timePeriod}`),
26     }),
27   }),
28 });
29
30 export const {
31   useGetCryptosQuery,
32   useGetCryptoDetailsQuery,
33   useGetCryptoHistoryQuery,
34 } = cryptoApi;
```

Adding Providers in the main function: React Router with `BrowserRouter`: - ``: This component is provided by `react-router-dom` and enables client-side routing using the HTML5 history API. It wraps the application, allowing it to use routing features.

Redux Provider:

``: This component is provided by `react-redux` and is used to provide the Redux store to the entire application. It wraps the application, allowing all components to access the Redux store. Overall, this code initializes the React application by rendering the

root component (``) into the DOM, while also providing routing capabilities through `BrowserRouter` and state management with Redux through `Provider`. Additionally, it ensures stricter development mode checks with ``.



```
1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import App from "./App.jsx";
4 import "./index.css";
5 import { BrowserRouter } from "react-router-dom";
6 import { Provider } from "react-redux";
7 import store from "./app/store.js";
8
9 ReactDOM.createRoot(document.getElementById(
10   "root")).render(
11   <React.StrictMode>
12     <BrowserRouter>
13       <Provider store={store}>
14         <App />
15       </Provider>
16     </BrowserRouter>
17   </React.StrictMode>
18 );
```

Creating a Line chart component: This code defines a React component called `LineChart` which renders a line chart using the `react-chartjs-2` library. 1. Imports: - `import React from "react";`: Imports the `React` module. - `import { Line } from "react-chartjs-2";`: Imports the `Line` component from the `react-chartjs-2` library, which is used to render line charts.

`import { Col, Row, Typography } from "antd";`: Imports specific components from the Ant Design library, including `Col`, `Row`, and `Typography`. - `const { Title } = Typography;`: Destructures the `Title` component from the `Typography` module.

2. Component Definition: - `const LineChart = ({ coinHistory, currentPrice, coinName }) => { ... }`: Defines a functional component

called `'LineChart'`. It takes three props: `'coinHistory'`, `'currentPrice'`, and `'coinName'`.

3. Data Preparation: - Inside the component, it loops through the `'coinHistory'` data to extract `'coinPrice'` and `'coinTimestamp'` arrays. These arrays will be used as data points for the line chart.

4. Chart Data: - `'const data = { ... }'`: Defines the data object for the line chart. It includes labels (timestamps) and datasets (coin prices).

5. Rendering: - Inside the return statement, it renders the chart header, including the coin name, price change, and current price. - `'Row'` and `'Col'` from Ant Design are used to structure the layout. - The `'Line'` component renders the actual line chart using the data object defined earlier.

6. Export: - `'export default LineChart;'`: Exports the `'LineChart'` component as the default export.

```

1 import React from "react";
2 import { Line } from "react-chartjs-2";
3 import { Col, Row, Typography } from "antd";
4 const { Title } = Typography;
5
6 const LineChart = ({ coinHistory, currentPrice, coinName }) => {
7   const coinPrice = [];
8   const coinTimestamp = [];
9
10  for (let i = 0; i < coinHistory?.data?.history?.length; i += 1) {
11    coinPrice.push(coinHistory?.data?.history[i].price);
12    coinTimestamp.push(
13      new Date(
14        coinHistory?.data?.history[i].timestamp * 1000
15      ).toLocaleDateString()
16    );
17  }
18
19  const data = {
20    labels: coinTimestamp,
21    datasets: [
22      {
23        label: "Price In USD",
24        data: coinPrice,
25        backgroundColor: "#0071bd",
26        borderColor: "#0071bd",
27      },
28    ],
29  };
30
31  return (
32    <>
33      <Row className="chart-header">
34        <Title level={2} className="chart-title">
35          {coinName} Price Chart
36        </Title>
37        <Col className="price-container">
38          <Title level={5} className="price-change">
39            Change: {coinHistory?.data?.change}%
40          </Title>
41          <Title level={5} className="current-price">
42            Current {coinName} Price: $ {currentPrice}
43          </Title>
44        </Col>
45      </Row>
46      <Line className="chart" data={data} />
47    </>
48  );
49 };
50
51 export default LineChart;
52

```

Creating cryptocurrencies component:

1. Component Denition: - `const Cryptocurrencies = ({ simplified }) => { ... }`: Denes a functional component named `Cryptocurrencies`. It accepts a prop named `simplified`, which determines whether to display a simplified version of the list.
2. Initialization: - `const count = simplified ? 10 : 100;`: Initializes the `count` variable based on the value of the `simplified` prop. If `simplified` is true, `count` is set to 10; otherwise, it's set to 100.

3. Fetching Cryptocurrency Data: - ``const { data: cryptosList, isFetching } = useGetCryptosQuery(count);``: Uses the ``useGetCryptosQuery`` hook provided by the ``cryptoApi`` service to fetch cryptocurrency data. It retrieves the list of cryptocurrencies (``cryptosList``) and a boolean flag (``isFetching``) indicating whether the data is being fetched.

4. Filtering Cryptocurrency Data: - The ``useEffect`` hook is used to filter the cryptocurrency data based on the ``searchTerm`` state variable. It updates the ``cryptos`` state with filtered data whenever ``cryptosList`` or ``searchTerm`` changes.

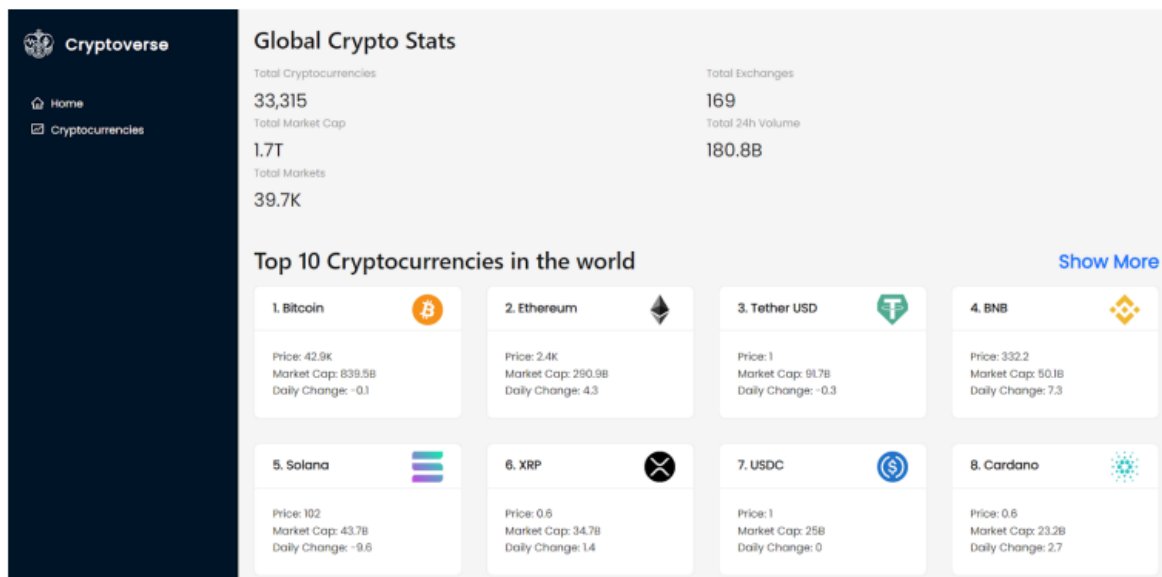
5. Rendering Loader: - ``if (isFetching) return ;``: If data is still being fetched (``isFetching`` is true), it returns a ``Loader`` component to indicate that the data is loading.

6. Rendering Search Input: - ``!simplified && (...)``: If ``simplified`` is false, it renders a search input field allowing users to search for specific cryptocurrencies by name.

7. Rendering Cryptocurrency Cards:

- The ``Row`` and ``Col`` components from Ant Design are used to create a grid layout for displaying cryptocurrency cards. - For each cryptocurrency in the ``cryptos`` array, it renders a ``Card`` component containing details such as name, price, market cap, and daily change. Each card is wrapped in a ``Link`` component, allowing users to navigate to the details page of a specific cryptocurrency.

8. Return Statement: - ``return (...)``: Returns JSX representing the component's structure and content.



1. Sidebar Navigation (Left Panel)

- The sidebar has a **dark theme** and contains navigation options:
 - **Home** – Likely redirects to the main dashboard.
 - **Cryptocurrencies** – Possibly shows detailed market data or additional coins.

2. Global Crypto Stats (Top Section)

- Displays key statistics about the cryptocurrency market:
 - **Total Cryptocurrencies:** 33,315
 - **Total Exchanges:** 169
 - **Total Market Cap:** 1.7T (Trillion)
 - **Total 24h Volume:** 180.8B (Billion)
 - **Total Markets:** 39.7K

3. Top 10 Cryptocurrencies (Main Section)

- Lists the **top cryptocurrencies** with details like:
 - **Price**
 - **Market Cap**
 - **Daily Change (%)**

- Each cryptocurrency has its respective logo and rank.
- The "**Show More**" link suggests there's additional data beyond the top 10.

4. Design & UI Elements

- **Modern & Minimalist Design** – Uses a clean white background with a contrasting dark sidebar.
- **Card-based Layout** – Each cryptocurrency is displayed in a card, making it visually appealing and easy to read.
- **Color-coded Changes** – Daily price changes appear in different colors (e.g., red for negative changes, green for positive).

5. Possible Functionality

- **Live Market Data:** The stats and cryptocurrency prices suggest real-time or periodically updated data.
- **Expandable List:** The "Show More" link indicates pagination or a detailed market page.
- **API Integration:** The data is likely fetched from an external API like **CoinGecko** or **CoinMarketCap**.

Conclusion

This dashboard provides an **overview of the global crypto market** in an **organized, user-friendly interface**. It's likely built using **React.js** with a **charting library** for data visualization.

Future Enhancements:

1. Real-Time Data Updates

- Use WebSockets (e.g., Binance API) to provide live price updates instead of periodic API calls.
- Implement polling (e.g., refresh every 5 seconds) for APIs that don't support WebSockets.

Tech Stack:

- Socket.io (for real-time communication)
- Binance WebSocket API

2. Interactive Charts & Data Visualization

- Use candlestick charts to show price trends.
- Implement historical price trends over different timeframes.
- Show market dominance & trading volume insights.

Libraries:

- Recharts, Chart.js, D3.js

3. User Authentication & Portfolio Management

- Allow users to sign up, log in, and track their crypto holdings.
- Enable a portfolio tracker where users can add their assets and see profit/loss.

Tech Stack:

- Firebase (for authentication)
- MongoDB / PostgreSQL (for storing portfolio data)

