RAJESWARI VEDHACHALAM GOVERNMENT ARTS COLLEGE CHENGALPATTU

CRYPTOVERSE
(CRYPTOCURRENCY DASHBOARD)

TEAM ID   : SWTID1741156497

TEAM LEADER   : KALAIYARASI P
EMAIL ID   : kalaiyarasipkalai@gmail.com
ROLE   : Managed Project & API integration

TEAM MEMBER 1 : PRAVEEN S
EMAIL ID   : praveenshezhian@gmail.com
ROLE   : developed backend

TEAM MEMBER 2 : HEMALATHA R
EMAIL ID   : hhema908947@gmail.com
ROLE   : Demo link creator

TEAM MEMBER 3 : KAVIYA A
EMAIL ID   : vinithkumar4022@gmail.com
ROLE   : Documentation

CLASS :III-BSC COMPUTER SCIENCE

Cryptoverse

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.

Description

Cryptoverse is a sophisticated cryptocurrency dashboard designed to provide investors with comprehensive insights into market dynamics through detailed historical price data analysis spanning five years. Featuring visually intuitive charts, interactive tools, and seamless navigation, the platform empowers users to identify top-performing assets and make informed investment decisions. With its robust search functionality, users can easily explore a wide range of cryptocurrencies and compare their performance over time. Cryptoverse not only serves as a powerful tool for optimizing investment portfolios but also acts as an educational resource, helping users understand the evolving nature of cryptocurrency markets.

Required things to run the program:

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.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

- Download Link: https://nodejs.org/en/download/

✔ HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

✔ 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.

  • Git: Download and installation instructions can be found at: https://git-scm.com/downloads

✔ Development Environment: IChoose a code editor or Integrated Development Environment (IDE) that suits my preferences as Visual Studio Code.

  • Visual Studio Code: Download from https://code.visualstudio.com/download

Install Dependencies:

  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.

You have successfully installed and set up the application on your local machine. You can now proceed with further customization, development, and testing as needed.

Project Structure:

```
∨ 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
```

Reference Image:
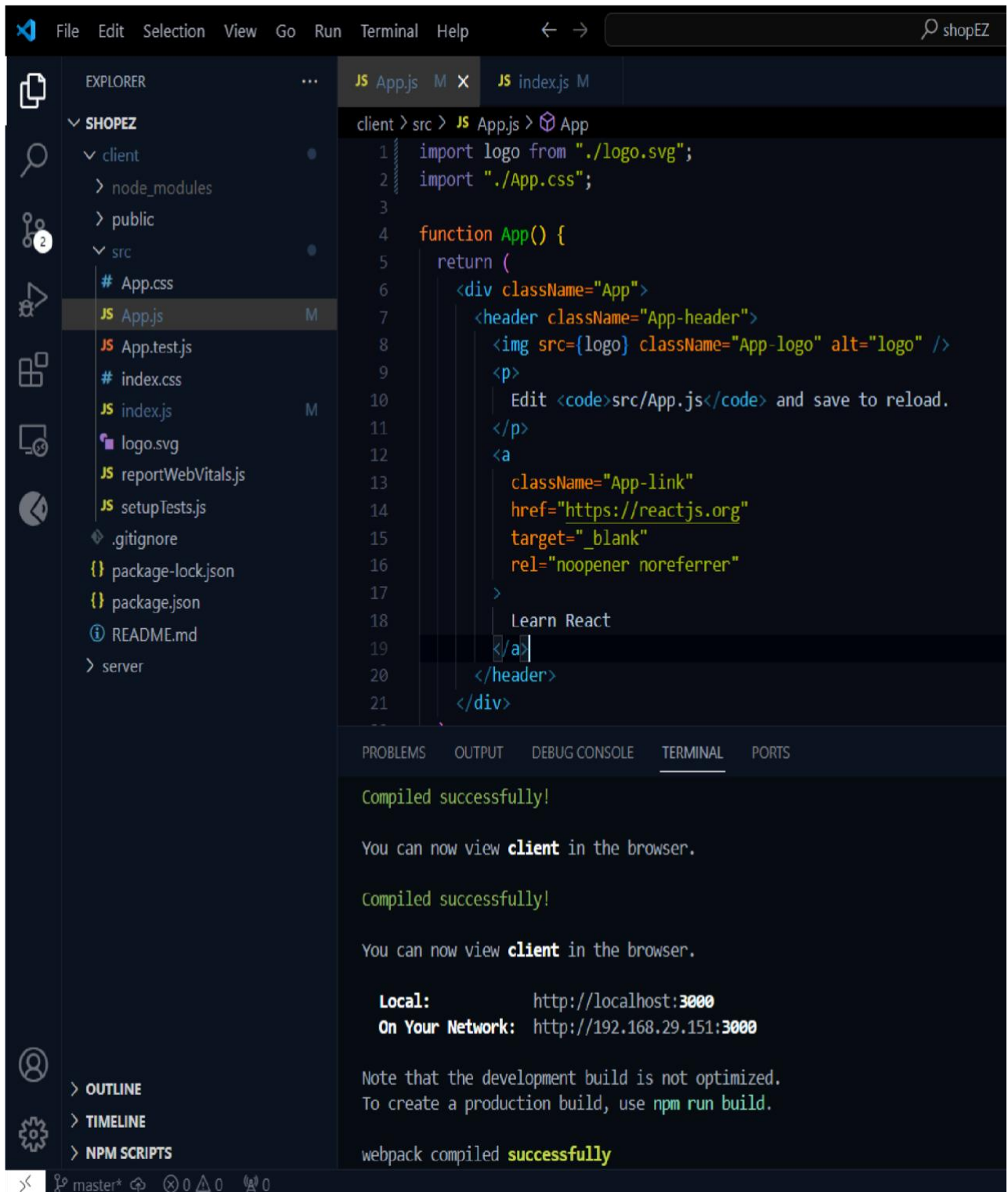
Project Development:

- **Create a redux store:**

    1. `import { configureStore } from "@reduxjs/toolkit";`: This line imports the `configureStore` function from Redux Toolkit. Redux Toolkit is a package that provides utilities to simplify Redux development, making it easier to write Redux logic with less boilerplate code.

    2. `import { cryptoApi } from "../services/cryptoApi";`: This line imports the `cryptoApi` object from the `cryptoApi.js` file located in the `../services` directory. This object likely contains configurations and functions related to making API requests for cryptocurrency data.

    3. `export default configureStore({ ... });`: This line exports the Redux store configuration created by the `configureStore` function as the default export of this module.

    4. `reducer: { [cryptoApi.reducerPath]: cryptoApi.reducer }`: This part of the configuration specifies the root reducer for the Redux store. In this case, it sets the `cryptoApi.reducer` as the reducer for the slice of state managed by the `cryptoApi` API slice. The `cryptoApi.reducerPath` likely refers to the slice's unique identifier, which is used internally by Redux Toolkit.

    5. `middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(cryptoApi.middleware),`: This part of the configuration specifies middleware for the Redux store. Middleware intercepts actions before they reach the reducers and can be used for various purposes, such as logging, asynchronous actions, or handling API requests. Here, it uses the `getDefaultMiddleware` function provided by Redux Toolkit to get the default middleware stack and appends the `cryptoApi.middleware`. This middleware likely handles asynchronous API requests and dispatches corresponding actions based on the API response.

    this configuration sets up a Redux store with a specific reducer and middleware provided by the `cryptoApi` object, which presumably manages state related to cryptocurrency data fetched from an external API. This setup allows you to manage and interact with this data using Redux within your React application.

```
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 Configuration:

  - `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`: Specifies the path under which the slice's reducer will be mounted in the Redux store.

   -    `baseQuery`: Configures the base query function used by the API slice. In this case, it uses `fetchBaseQuery` with the base URL specified.

   -    `endpoints`: Defines the API endpoints available in the slice. It's an object with keys corresponding to endpoint names and values being endpoint definitions.

5. API Endpoints:

   - `getCryptos`, `getCryptoDetails`, `getCryptoHistory`: These are endpoints defined using the `builder.query` method. Each endpoint is configured with a `query` function that returns the request configuration 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 defined in the `endpoints` object.

Overall, this code sets up an API slice named `cryptoApi` using Redux Toolkit's query functionality. It defines endpoints for fetching cryptocurrencies, cryptocurrency details, and cryptocurrency history. The slice is configured with base URL, headers, and query functions required for making requests to the cryptocurrency API.

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

const cryptoApiHeaders = {
  "X-RapidAPI-Key": import.meta.env.VITE_RAPID_API_KEY,
  "X-RapidAPI-Host": import.meta.env.VITE_RAPID_API_HOST,
};

const baseUrl = import.meta.env.VITE_BASE_URL;

const createRequest = (url) => ({ url, headers: cryptoApiHeaders });

export const cryptoApi = createApi({
  reducerPath: "cryptoApi",
  baseQuery: fetchBaseQuery({ baseUrl }),
  endpoints: (builder) => ({
    getCryptos: builder.query({
      query: (count) => createRequest(`/coins?limit=${count}`),
    }),

    getCryptoDetails: builder.query({
      query: (coinId) => createRequest(`/coin/${coinId}`),
    }),
    getCryptoHistory: builder.query({
      query: ({ coinId, timePeriod }) =>
        createRequest(`coin/${coinId}/history?timePeriod=${timePeriod}`),
    }),
  }),
});

export const {
  useGetCryptosQuery,
  useGetCryptoDetailsQuery,
  useGetCryptoHistoryQuery,
} = cryptoApi;
```

- **Adding Providers in the main function:**
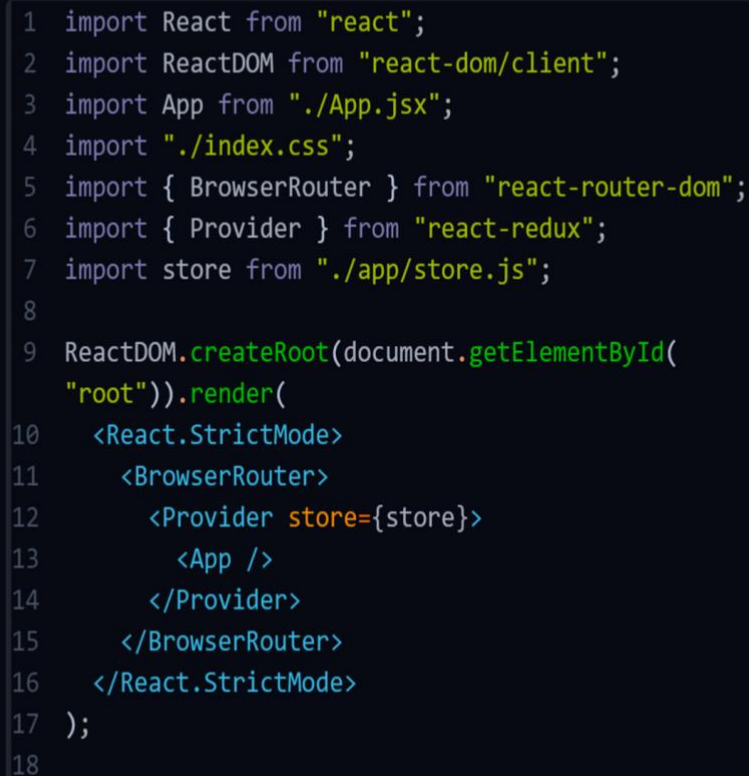

    React Router with `BrowserRouter`:

  - `<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:

o

- `<Provider store={store}>`: This c mponent 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.

```
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(
   "root")).render(
10   <React.StrictMode>
11     <BrowserRouter>
12       <Provider store={store}>
13         <App />
14       </Provider>
15     </BrowserRouter>
16   </React.StrictMode>
17 );
18
```

Overall, this code initializes the React application by rendering the root component (`<App />`) 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 `<React.StrictMode>`.

o

● **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 } fr m "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).

o

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.
   Overall, this component receives historical data (`coinHistory`), current price (`currentPrice`), and the name of the cryptocurrency (`coinName`) as props, and renders a line chart displaying the historical price data. It also includes additional information such as the price change and current price displayed above the chart.

o

```jsx
import React from "react";
import { Line } from "react-chartjs-2";
import { Col, Row, Typography } from "antd";
const { Title } = Typography;

const LineChart = ({ coinHistory, currentPrice, coinName }) => {
  const coinPrice = [];
  const coinTimestamp = [];

  for (let i = 0; i < coinHistory?.data?.history?.length; i += 1) {
    coinPrice.push(coinHistory?.data?.history[i].price);
    coinTimestamp.push(
      new Date(
        coinHistory?.data?.history[i].timestamp * 1000
      ).toLocaleDateString()
    );
  }

  const data = {
    labels: coinTimestamp,
    datasets: [
      {
        label: "Price In USD",
        data: coinPrice,
        backgroundColor: "#0071bd",
        borderColor: "#0071bd",
      },
    ],
  };

  return (
    <>
      <Row className="chart-header">
        <Title level={2} className="chart-title">
          {coinName} Price Chart
        </Title>
        <Col className="price-container">
          <Title level={5} className="price-change">
            Change: {coinHistory?.data?.change}%
          </Title>
          <Title level={5} className="current-price">
            Current {coinName} Price: $ {currentPrice}
          </Title>
        </Col>
      </Row>
      <Line className="chart" data={data} />
    </>
  );
};

export default LineChart;
```

● **Creating cryptocurrencies component:**

1. Component Definition:

   - `const Cryptocurrencies = ({ simplified }) => { ... }`: Defines 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 <Loader />;`: 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.
Overall, this component fetches cryptocurrency data, filters it based on a search term, and renders the data in a visually appealing format with card-based UI. It also provides a search functionality for users to find specific cryptocurrencies
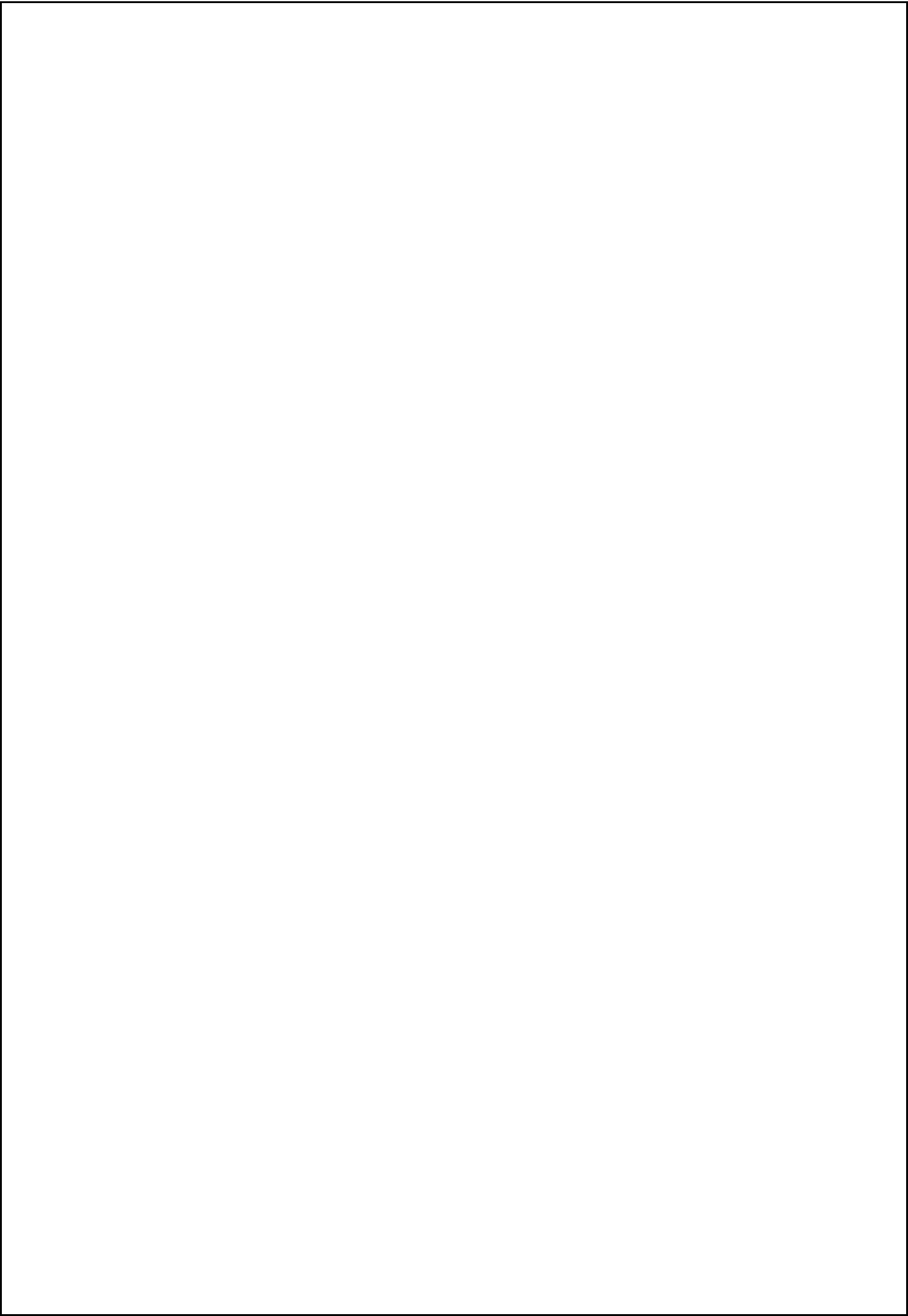
```jsx
1  import React, { useEffect, useState } from "react";
2  import millify from "millify";
3  import { Link } from "react-router-dom";
4  import { Card, Row, Col, Input } from "antd";
5  import { useGetCryptosQuery } from "../services/cryptoApi";
6  import Loader from "./Loader";
7
8  const Cryptocurrencies = ({ simplified }) => {
9    const count = simplified ? 10 : 100;
10   const { data: cryptosList, isFetching } = useGetCryptosQuery(count);
11   const [cryptos, setcryptos] = useState([]);
12   const [searchTerm, setsearchTerm] = useState("");
13
14   useEffect(() => {
15     const filteredData = cryptosList?.data?.coins.filter((item) =>
16       item.name.toLowerCase().includes(searchTerm.toLowerCase())
17     );
18     setcryptos(filteredData);
19   }, [cryptosList, searchTerm]);
20
21   if (isFetching) return <Loader />;
22
23   return (
24     <>
25       {!simplified && (
26         <div className="search-crypto">
27           <Input
28             placeholder="Search Cryptocurrency"
29             onChange={(e) => setsearchTerm(e.target.value)}
30           />
31         </div>
32       )}
33       <Row gutter={[32, 32]} className="crypto-card-container">
34         {cryptos?.map((currency) => {
35           return (
36             <Col
37               xs={24}
38               sm={12}
39               lg={6}
40               className="crypto-card"
41               key={currency.uuid}
42             >
43               <Link key={currency.uuid} to={`/crypto/${currency.uuid}`}>
44                 <Card
45                   extra={
46                     <img className="crypto-image" src={currency.iconUrl} />
47                   }
48                   title={`${currency.rank}. ${currency.name}`}
49                 >
50                   <p>Price: {millify(currency.price)}</p>
51                   <p>Market Cap: {millify(currency.marketCap)}</p>
52                   <p>Daily Change: {millify(currency.change)}</p>
53                 </Card>
54               </Link>
55             </Col>
56           );
57         })}
58       </Row>
59     </>
60   );
61 };
62
63 export default Cryptocurrencies;
64
```

● **Create a component to show the details of cryptocurrency:**

This code defines a React functional component called `CryptoDetails` responsible for displaying detailed information about a specific cryptocurrency. Let's break down the code:

1. Component Definition:

   - `const CryptoDetails = () => { ... }`: Defines a functional component named `CryptoDetails`. It doesn't accept any props directly but utilizes React Router's `useParams` hook to get the `coinId` parameter from the URL.

2. State Initialization:

   - Initializes state variables `timePeriod` and `coinHistory`. `timePeriod` represents the selected time period for displaying cryptocurrency history, and `coinHistory` stores historical data of the selected cryptocurrency.

3. Fetching Data:

   - Utilizes `useGetCryptoDetailsQuery` and `useGetCryptoHistoryQuery` hooks provided by the `cryptoApi` service to fetch details and historical data of the cryptocurrency specified by `coinId`. It uses `coinId` obtained from `useParams` to fetch data for the specific cryptocurrency.

4. Setting Coin History:

   - Utilizes `useEffect` hook to update the `coinHistory` state when `coinHistoryData` changes. This ensures that the component re-renders with updated historical data.

5. Rendering Loader:

   - Displays a loading indicator (`<Loader />`) while data is being fetched (`isFetching` is true).
6. Time Period Selection:

   - Renders a `Select` component allowing users to choose the time period for displaying historical data. It triggers the `setTimePeriod` function when the selection changes.

7. Rendering Line Chart:

   - Utilizes the `LineChart` component to display the historical price trend of the cryptocurrency over the selected time period.

8. Rendering Statistics:

   - Displays various statistics related to the cryptocurrency, such as price, rank, volume, market cap, etc. These statistics are displayed in two sections: `stats` and `genericStats`.

9. Rendering Description and Links:

   - Parses and displays the description of the cryptocurrency using `HTMLReactParser`.

   - Renders links related to the cryptocurrency, such as official websites, social media, etc.

10. Return Statement:

   - Returns JSX representing the structure and content of the component.

Overall, this component fetches and displays detailed information about a specific cryptocurrency, including historical price data, key statistics, description, and related links.

● **Create a Homepage:**
   This component, named `Home`, is a React functional component responsible for rendering the home page of the cryptocurrency dashboard. Let's break down the code:
   1. Component Definition:
     - `const Home = () => { ... }`: Defines a functional component named `Home`.
   2. `Data Fetching:
     - Uses the `useGetCryptosQuery` hook provided by the `cryptoApi` service to fetch data for the top 10 cryptocurrencies. It retrieves data and a boolean flag indicating whether data is being fetched.

   3. Rendering Loader:

     - Displays a loading indicator (`<Loader />`) while data is being fetched (`isFetching` is true).

4. Global Crypto Stats:

   - Renders statistics about the global cryptocurrency market, including total cryptocurrencies, total exchanges, total market cap, total 24-hour volume, and total markets. These statistics are displayed using the `Statistic` component from Ant Design.

5. Top 10 Cryptocurrencies:

   -      Renders a section displaying the top 10 cryptocurrencies in the world.

   -      Utilizes the `Cryptocurrencies` component with the `simplified` prop set to true to display a simplified version of the list.

   -      Provides a link to view more cryptocurrencies using the `Link` component from React Router.

6. Return Statement:
   - Returns JSX representing the structure and content of the component.

Overall, this component fetches and displays global cryptocurrency statistics and the top 10 cryptocurrencies on the homepage of the dashboard. It provides links for users to navigate to the full list of cryptocurrencies.

Github link : https://github.com/Praveenshezhian/Cryptocurrency
Project Demo :
https://drive.google.com/file/d/17wNaRwoeT1qwV9nH6bBItAzcioaD4y
h_/view?usp=drivesdk

.env file :
VITE_RAPID_API_KEY =
407caa43a9msh3cbe4c56d310e44p1a2e9bjsnc3c9d3bf7640
VITE_RAPID_API_HOST= https://coinranking1.p.rapidapi.com
**VITE_BASE_URL=** https://coinranking1.p.rapidapi.com

```jsx
1   import React from "react";
2   import milify from "millify";
3   import { Typography, Row, Col, Statistic } from "antd";
4   import { Link } from "react-router-dom";
5   const { Title } = Typography;
6   import { useGetCryptosQuery } from "../services/cryptoApi";
7   import Cryptocurrencies from "./Cryptocurrencies";
8   import Loader from "./Loader";
9
10  const Home = () => {
11    const { data, isFetching } = useGetCryptosQuery(10);
12    if (isFetching) return <Loader />;
13    const globalStats = data?.data?.stats;
14
15    return (
16      <>
17        <Title level={2} className="heading">
18          Global Crypto Stats
19        </Title>
20        <Row>
21          <Col span={12}>
22            <Statistic title="Total Cryptocurrencies" value={globalStats.total} />
23          </Col>
24          <Col span={12}>
25            <Statistic
26              title="Total Exchanges"
27              value={milify(globalStats.totalExchanges)}
28            />
29          </Col>
30          <Col span={12}>
31            <Statistic
32              title="Total Market Cap"
33              value={milify(globalStats.totalMarketCap)}
34            />
35          </Col>
36          <Col span={12}>
37            <Statistic
38              title="Total 24h Volume"
39              value={milify(globalStats.total24hVolume)}
40            />
41          </Col>
42          <Col span={12}>
43            <Statistic
44              title="Total Markets"
45              value={milify(globalStats.totalMarkets)}
46            />
47          </Col>
48        </Row>
49        <div className="home-heading-container">
50          <Title level={2} className="home-title">
51            Top 10 Cryptocurrencies in the world
52          </Title>
53          <Title level={3} className="show-more">
54            <Link to="/cryptocurrencies">Show More</Link>
55          </Title>
56        </div>
57        <Cryptocurrencies simplified />
58      </>
59    );
60  };
61
62  export default Home;
63
```

User Interface snips:

Home page : This pages consists of stats of global crypto like total cryptocurrencies, total exchanges, market cap etc. Also consist of top 10 cryptocurrencies in the world.
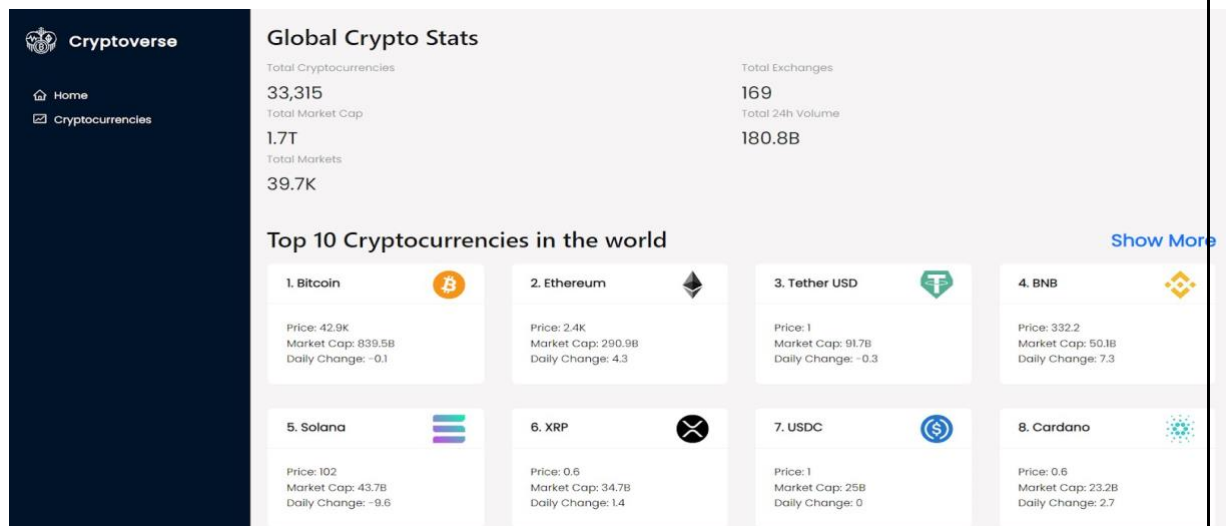
Crypto currencies page : This pages contains all cryptocurrencies which are currently in flow in the world. There is also a search feature where users can search and find out about their desired cryptocurrency.
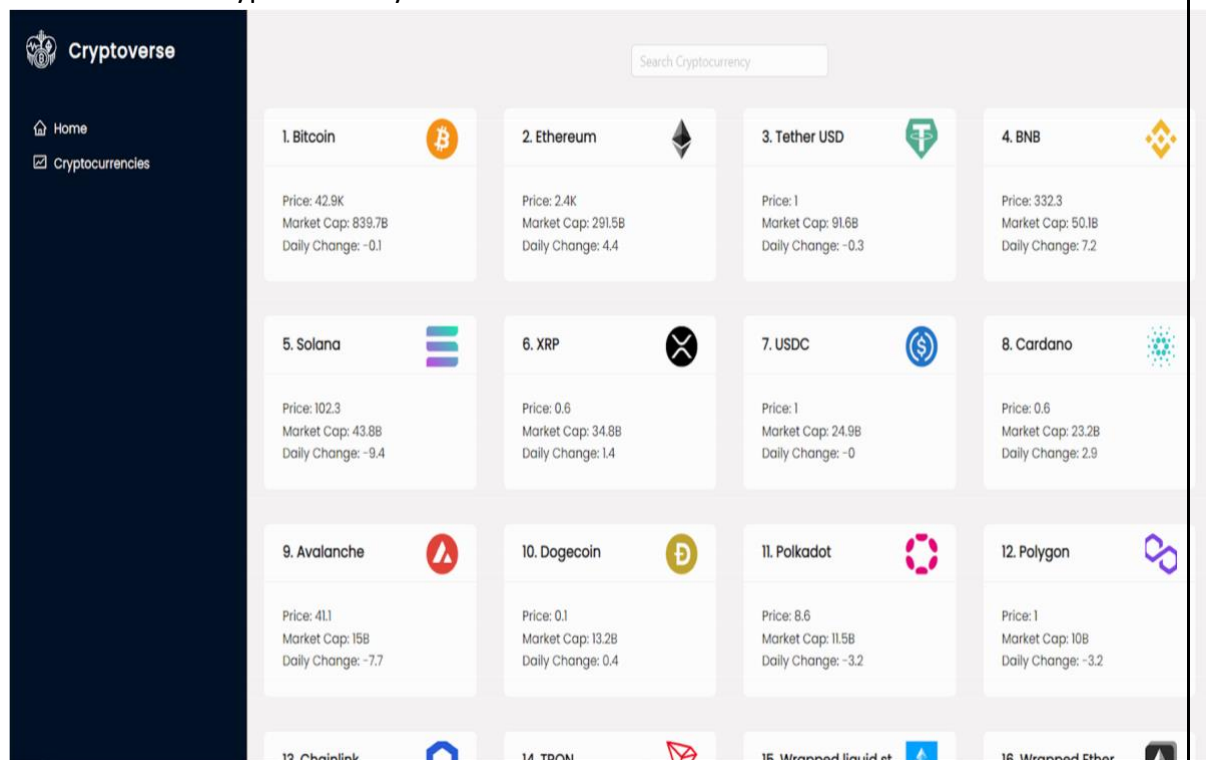
Crypto currency details page : This page contains the line chart with data representation of price of cryptocurrencies. Also contains statistics and website links of cryptocurrencies.

Bitcoin Price Chart

Change: 1084.17%   **Current Bitcoin Price: $ 42.9K**

Bitcoin value statistics

An overview showing the stats of Bitcoin

Price to USD		$ 42.9K
# Rank		1
24h Volume		$ 22.8B
Market Cap		$ 840.4B
All-time-high(daily avg.)		$ 68.8K

Others statistics

An overview showing the stats of all cryptocurrencies

Number Of Markets		3212
Number Of Exchanges		121
Aprroved Supply		✓
Total Supply		$ 19.6M
Circulating Supply		$ 19.6M

What is Bitcoin ?

Bitcoin is a digital currency with a finite supply, allowing users to send/receive money without a central bank/government, often nicknamed "Digital Gold".

Bitcoin Links

Website	bitcoin.org
Website	bitcoinmagazine.com
Bitcointalk	bitcointalk.org

CONCLUSION :

- In this article, we describe an API-based method for Establishing a cryptocurrency dashboard. Our app collects realtime data on several cryptocurrencies via APIs and displays it in An accessible and user-friendly dashboard. The dashboard includes Real-time price updates, historical price charts, market Capitalisation statistics, and news feeds, among other things.We examined the crypto currency dashboard in this article. The Research paper is an essential component of academic activity. A Cryptocurrency dashboard is an essential component of the Cryptocurrency market. It gives cryptocurrency market data, such As price and volume, that investors can use to make smarter Investing decisions.

- The crypto currency dashboard is a web application that displays Real-time information on cryptocurrencies. The application makes Use of data from exchanges such as CoinGecko and Coinbase, which Provide real-time updates on the values and volumes of various Coins.