CRYPTOVERSE

(A CRYPTOCURRENCY DASHBOARD)

- **PROJECT TITLE :**

Cryptoverse : A Cryptocurrency Dashboard

- **TEAM MEMBERS :**

- **THAMIZHMOZHI K**
  (cs2201111058050@lngovernmentcollege.com)

- **BAKKIYAMMAL L**
  (cs2201111058005@lngovernmentcollege.com)

- **VIGNESH R**
  (cs2201111058055@lngovernmentcollege.com)

- **PAVITHRA S**
  (cs2201111058037@lngovernmentcollege.com)

- **DILLI GANESH S**
  (cs2201111058012@lngovernmentcollege.com)

PROJECT OVERVIEW

• PURPOSE & FEATURES :

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.

ARCHITECTURE

Component Structure

The major React components and their interactions in the **Cryptoverse** cryptocurrency dashboard are structured as follows:

- **Home Component (Home.js)**
  - Displays global cryptocurrency statistics.
  - Lists the top 10 cryptocurrencies.
  - Provides navigation to the full list of cryptocurrencies.

- **Cryptocurrencies Component (Cryptocurrencies.js)**
  - Displays a list of cryptocurrencies.
  - Provides search functionality.
  - Each cryptocurrency links to a details page.

- **CryptoDetails Component (CryptoDetails.js)**
  - Shows detailed information about a selected cryptocurrency.
  - Includes historical price data and charts.
  - Provides statistics, a description, and related links.
- **LineChart Component (LineChart.js)**
  - Renders a line chart for historical price trends.
  - Uses react-chartjs-2 and Chart.js.

Routing and Navigation

- The application uses react-router-dom for seamless navigation.
- Pages are structured with <Routes> and <Route> components.

State Management

The application uses **Redux Toolkit** for state management. Key aspects:

- **Redux Store (store.js)**
  - Configures and exports the Redux store.
  - Uses middleware to handle API requests.
- **API Slice (cryptoApi.js)**
  - Manages cryptocurrency data fetching.
  - Uses createApi from Redux Toolkit to define endpoints.
- **Data Fetching**
  - API calls fetch cryptocurrency details, market stats, and historical data.
  - The state is updated in Redux and accessed through custom hooks like `useGetCryptosQuery`.

Routing The app uses **React Router (react-router-dom)** for client-side navigation. Key routes:

- / → **Home page** (Global stats & Top 10 cryptos)
- /cryptocurrencies → **Cryptocurrency list**
- /crypto/:coinId → **Cryptocurrency details**
- Other pages include navigation and structured layouts.

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.

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

• Download: https://nodejs.org/en/download/

• Installation instructions: https://nodejs.org/en/download/package-manager/

✓ 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. Install React.js, a JavaScript library for building user interfaces.

• Create a new React app: **npx create-react-app my-react-app** Replace my-react-app with your preferred project name.

• Navigate to the project directory: **cd my-react-app**

• Running the React App: With the React app created, you can now start the development server and see your React application in action.

• Start the development server: **npm start**

This command launches the development server, and you can access your React app at http://localhost:3000 in your web browser

BASIC WEB DEVELOPMENT KNOWLEDGE

Understand **HTML**, **CSS**, and **JavaScript** for structuring, styling, and interactivity.

INSTALLATION :

✓ 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

✓ Choose a code editor or Integrated Developmet Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

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

• **Sublime Text**: Download from https://www.sublimetext.com/download

• **WebStorm**: Download from https://www.jetbrains.com/webstorm/download

✓ Get the code from google drive:

• Download the code from the drive link given below:

Drive_link:
https://drive.google.com/file/d/1fhCsitsdlwAMMxu9hjd1PM2rxdkQx6MN/view?usp=sharing

✓ Clone the code from github repository:

Follow below steps:

Git repository: https://github.com/Thamizhmozhi/cryptoverse.git

Git clone command: git clone https://github.com/Thamizhmozhi/cryptoverse.git

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

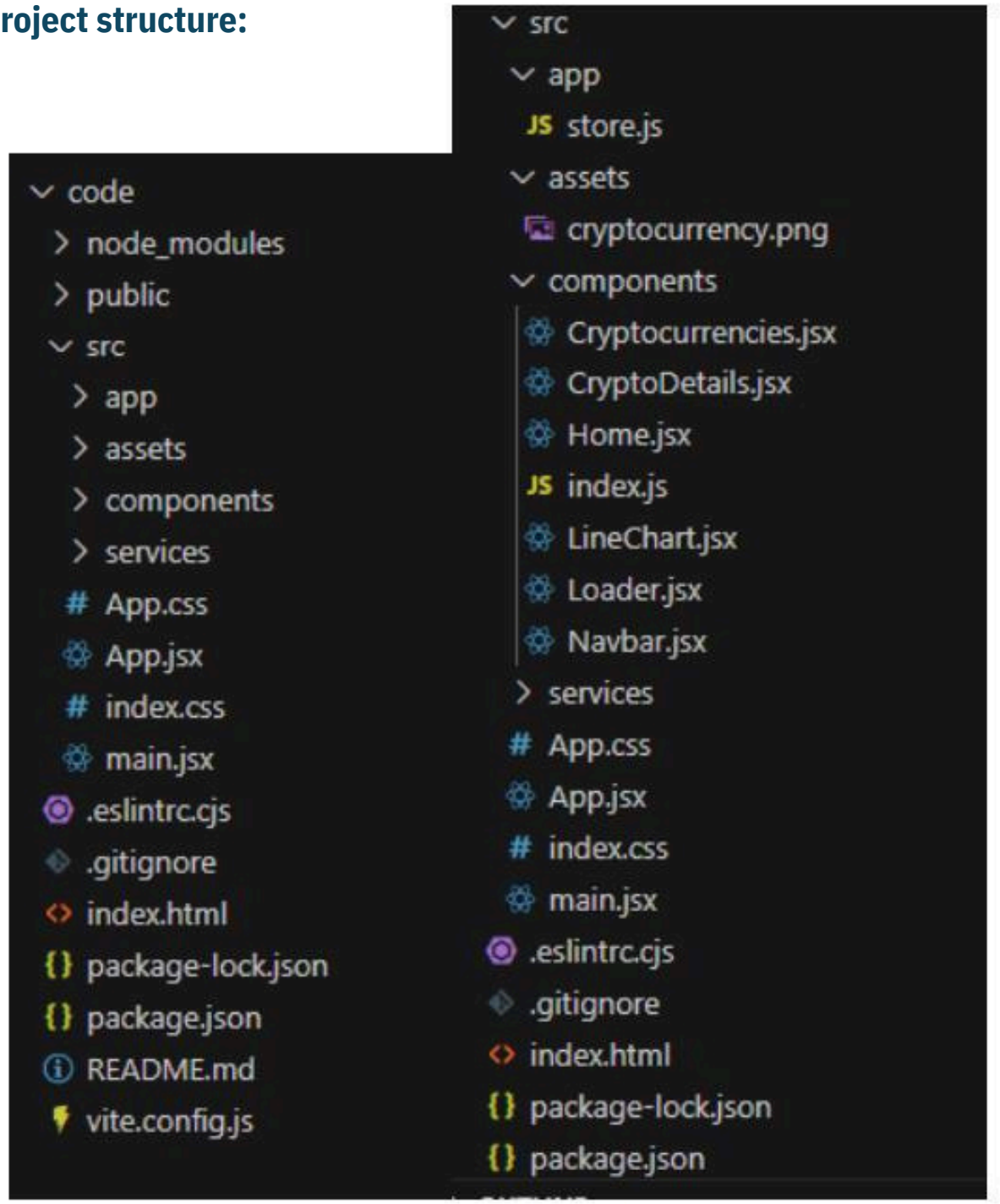
Access the App:

• 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.

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:



In this project, we've split the files into 3 major folders, Components, app and services. In the folder, we store the files that acts as pages at different url's in the application. The components folder stores all the files, that returns the small components in the application

Running the Application :

Provide commands to start the frontend server locally.

- **cd code**
- **npm install**
- **npm run dev**

Component Documentation

1. Key Components

These are the major components that form the core functionality of the application.

1.1 Home Component (Home.js)

- **Purpose**: Displays global cryptocurrency statistics and a list of the top 10 cryptocurrencies.
- **Key Features**:
  - Fetches and displays market statistics (total cryptocurrencies, exchanges, market cap, etc.).
  - Renders a simplified version of the Cryptocurrencies component.
  - Provides a link to view more cryptocurrencies.

1.2 Cryptocurrencies Component (Cryptocurrencies.js)

- **Purpose**: Displays a list of cryptocurrencies, allows searching, and provides links to detailed pages.
- **Props**:
  - simplified *(Boolean)* – If true, limits the displayed cryptocurrencies to the top 10.
- **Key Features**:
  - Fetches a list of cryptocurrencies from the API.
  - Filters and searches cryptocurrencies dynamically.
  - Displays cryptocurrency cards with price, market cap, and daily change.

1.3 CryptoDetails Component (CryptoDetails.js)

- **Purpose**: Displays detailed information about a specific cryptocurrency.
- **Props**: None (fetches coinId from URL parameters)
- **Key Features**:
  - Fetches cryptocurrency details and historical price data.
  - Displays a line chart showing price trends.
  - Lists key statistics, description, and useful links.

1.4 LineChart Component (LineChart.js)

- **Purpose**: Displays a historical price trend of a cryptocurrency using a line chart.
- **Props**:
  - o coinHistory *(Array)* – Historical price data.
  - o currentPrice *(String)* – Current price of the cryptocurrency.
  - o coinName *(String)* – Name of the cryptocurrency.
- **Key Features**:
  - o Renders a line chart using react-chartjs-2 and Chart.js.
  - o Dynamically updates based on the selected timeframe.

2. Reusable Components

These components are used across multiple sections of the application.

2.1 Loader Component (Loader.js)

- **Purpose**: Displays a loading animation when data is being fetched.

2.2 Navbar Component (Navbar.js)

- **Purpose**: Provides site-wide navigation.

Project Flow:

• Project setup and configuration:

1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

2.Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

3.Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Reference Video Link:

https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?usp=sharing

Reference Article Link: https://www.w3schools.com/react/react_getstarted.asp

State Management

Type	Managed By	Usage
Global State	Redux Toolkit	Fetching/storing API data, shared across components.
Local State	useState/useEffect	UI-specific interactions like search inputs, dropdown selections.

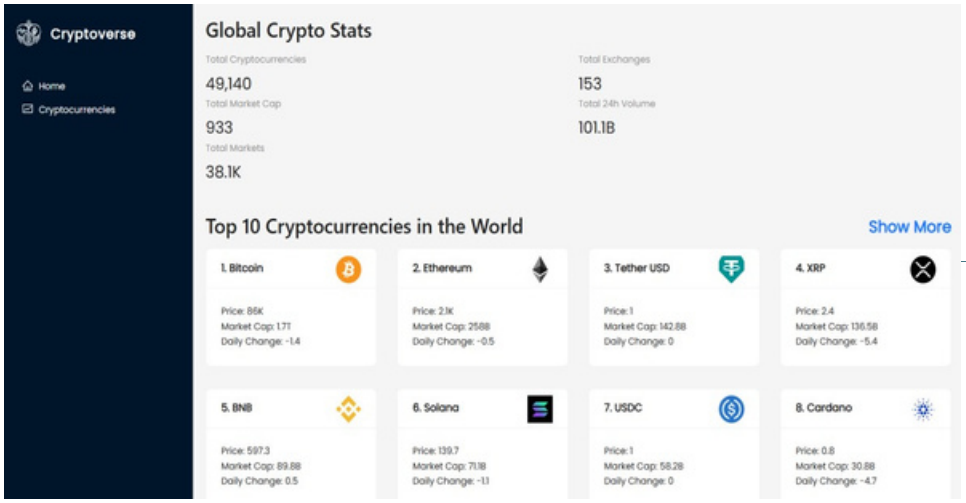
Project demo:

Demo link:

https://drive.google.com/file/d/1bvUCN_4RCD_6VGJPxEqo_mnWlVWWatMB/view?usp=drive_link

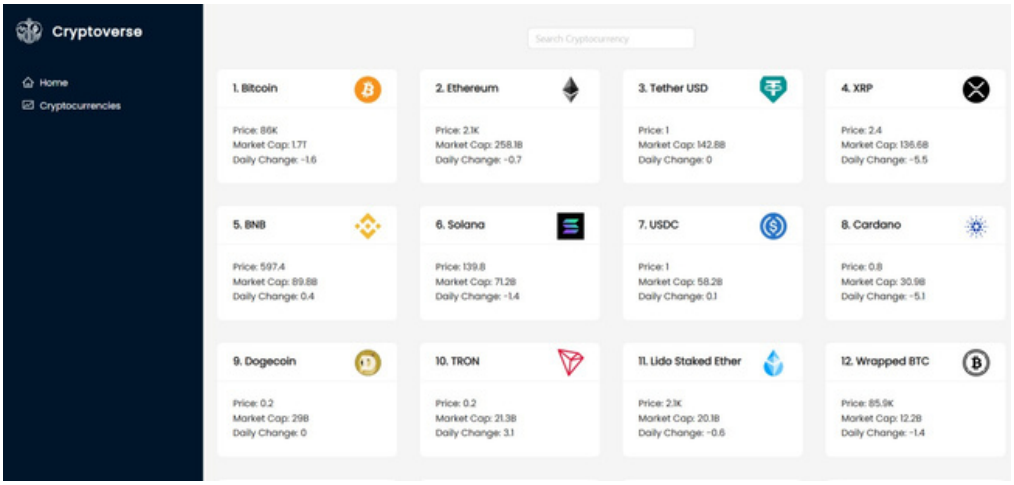
User Interface snips:

 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.



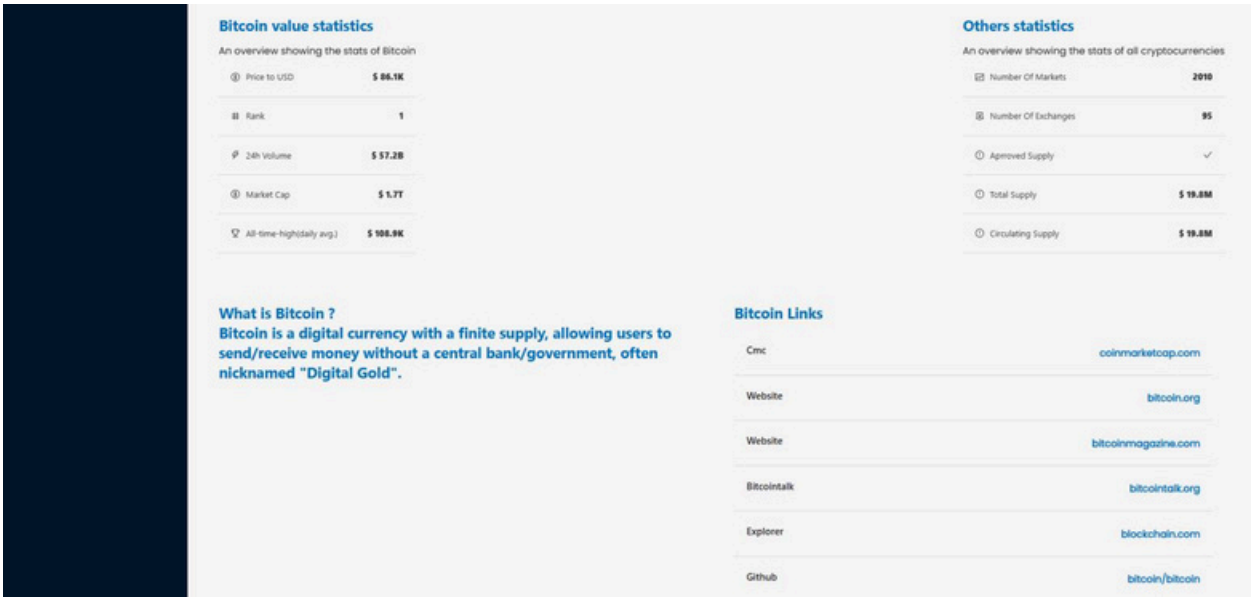
 Crypto currencies page :

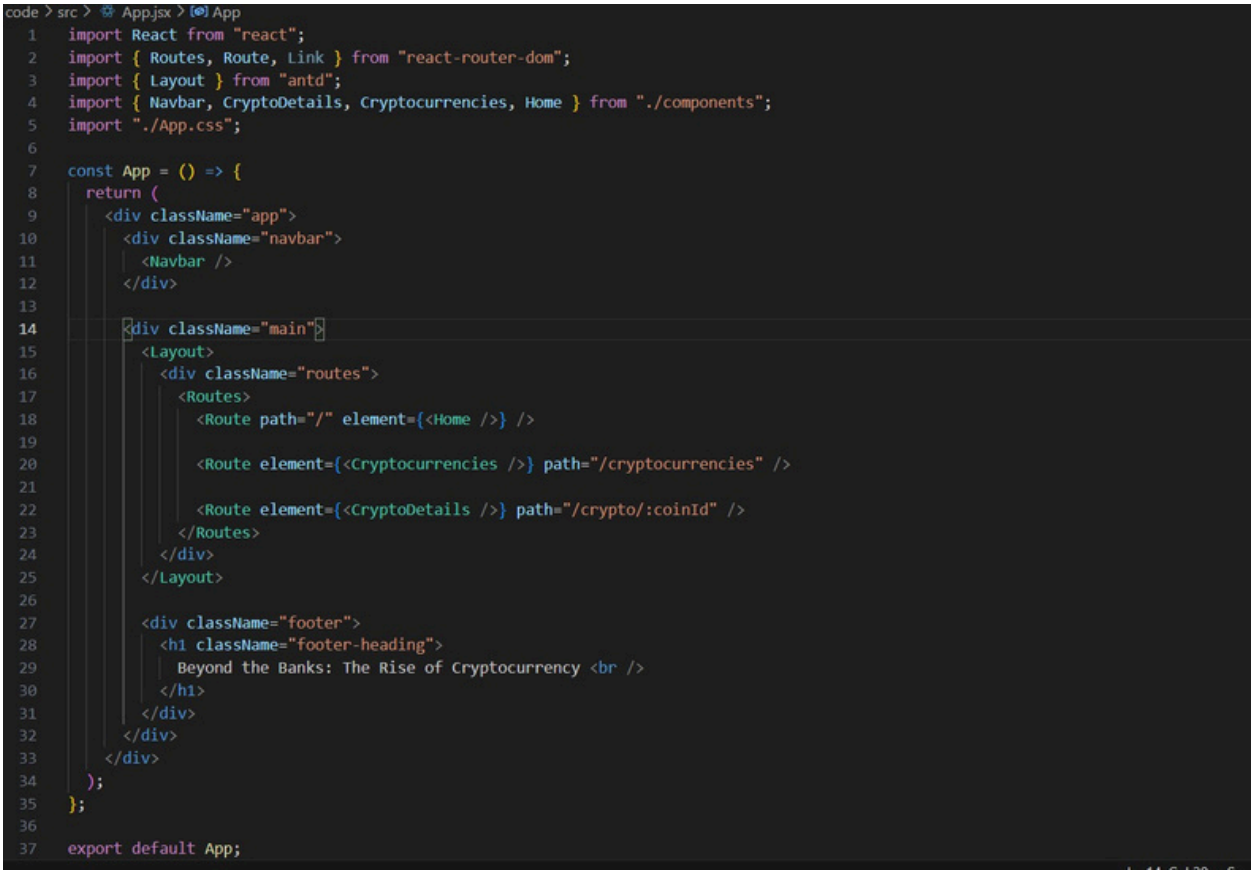
This pages contains all cryptocurrencies which are currently in flow in the world. There is also a search feature where users can search and find out about their desired cryptocurrency.



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.





```
code > src > App.jsx > App
1   import React from "react";
2   import { Routes, Route, Link } from "react-router-dom";
3   import { Layout } from "antd";
4   import { Navbar, CryptoDetails, Cryptocurrencies, Home } from "./components";
5   import "./App.css";
6
7   const App = () => {
8     return (
9       <div className="app">
10        <div className="navbar">
11          <Navbar />
12        </div>
13
14        <div className="main">
15          <Layout>
16            <div className="routes">
17              <Routes>
18                <Route path="/" element={<Home />} />
19
20                <Route element={<Cryptocurrencies />} path="/cryptocurrencies" />
21
22                <Route element={<CryptoDetails />} path="/crypto/:coinId" />
23              </Routes>
24            </div>
25          </Layout>
26
27          <div className="footer">
28            <h1 className="footer-heading">
29              Beyond the Banks: The Rise of Cryptocurrency <br />
30            </h1>
31          </div>
32        </div>
33      </div>
34    );
35  };
36
37  export default App;
```

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.
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.

```
import { configureStore } from "@reduxjs/toolkit";
import { cryptoApi } from "../services/cryptoApi";

export default configureStore({
  reducer: {
    [cryptoApi.reducerPath]: cryptoApi.reducer,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(cryptoApi.middleware),
});

```

• **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 `f**etchBaseQuery**` 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 `i**mport.meta.env**`. - `const baseUrl **= import.meta.env.VITE_BASE_URL;**`: This variabl 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**` 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. - `baseQueryConfigures 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.

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

const cryptoApiHeaders = {
  "X-RapidAPI-Key": 'ceb1a4f0a7msh4536fafc5647bcep123e58jsnc8fb6eec0272',
  "X-RapidAPI-Host": "coinranking1.p.rapidapi.com",
};

const baseUrl = "https://api.coinranking.com/v2";

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**`: - ``: 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.

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "./index.css";
import { BrowserRouter } from "react-router-dom";
import { Provider } from "react-redux";
import store from "./app/store.js";

ReactDOM.createRoot(document.getElementById(
"root")).render(
  <React.StrictMode>
    <BrowserRouter>
      <Provider store={store}>
        <App />
      </Provider>
    </BrowserRouter>
  </React.StrictMode>
);
```

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.

```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;
```

```jsx
import React, { useEffect, useState } from "react";
import millify from "millify";
import { Link } from "react-router-dom";
import { Card, Row, Col, Input } from "antd";
import { useGetCryptosQuery } from "../services/cryptoApi";
import Loader from "./Loader";

const Cryptocurrencies = ({ simplified }) => {
  const count = simplified ? 10 : 100;
  const { data: cryptosList, isFetching } = useGetCryptosQuery(count);
  const [cryptos, setcryptos] = useState([]);
  const [searchTerm, setsearchTerm] = useState("");

  useEffect(() => {
    const filteredData = cryptosList?.data?.coins.filter((item) =>
      item.name.toLowerCase().includes(searchTerm.toLowerCase())
    );
    setcryptos(filteredData);
  }, [cryptosList, searchTerm]);

  if (isFetching) return <Loader />;

  return (
    <>
      {!simplified && (
        <div className="search-crypto">
          <Input
            placeholder="Search Cryptocurrency"
            onChange={(e) => setsearchTerm(e.target.value)}
          />
        </div>
      )}
      <Row gutter={[32, 32]} className="crypto-card-container">
        {cryptos?.map((currency) => {
          return (
            <Col
              xs={24}
              sm={12}
              lg={6}
              className="crypto-card"
              key={currency.uuid}
            >
              <Link key={currency.uuid} to={`/crypto/${currency.uuid}`}>
                <Card
                  extra={
                    <img classNamе="crypto-image" src={currency.iconUrl} />
                  }
                  title={`${currency.rank}. ${currency.name}`}
                >
                  <p>Price: {millify(currency.price)}</p>
                  <p>Market Cap: {millify(currency.marketCap)}</p>
                  <p>Daily Change: {millify(currency.change)}</p>
                </Card>
              </Link>
            </Col>
          );
        })}
      </Row>
    </>
  );
};

export default Cryptocurrencies;
```

• **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 (` `) 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`.

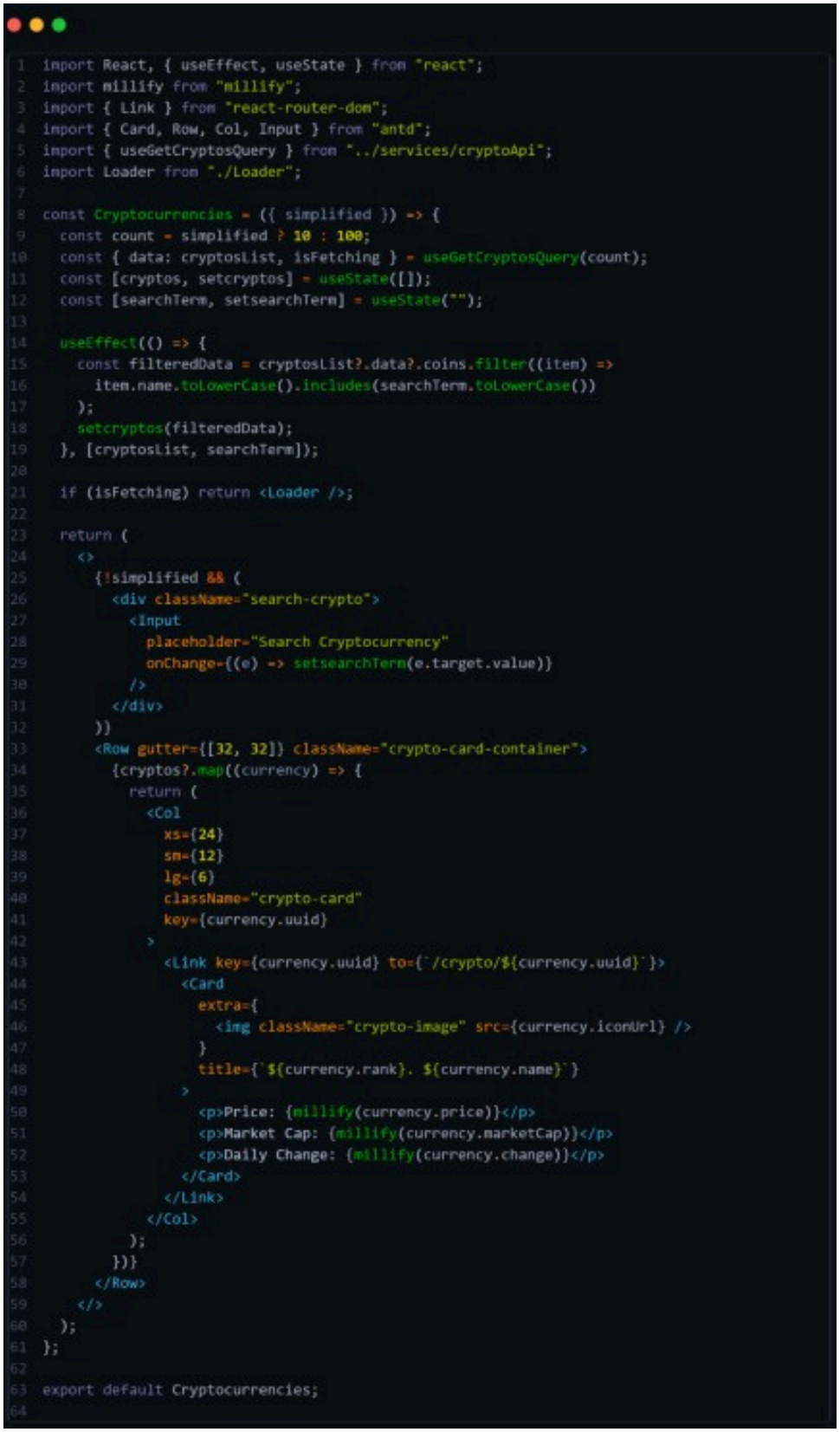
9. Rendering Description and Links: - Parses and displays the description of the cryptocurrency using `HTMLReactParser`. - Renders links related to the cryptocurrency, such as official websites, social media, etc.

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.

Reference Image Link: code link

• **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:



```jsx
import React, { useEffect, useState } from "react";
import millify from "millify";
import { Link } from "react-router-dom";
import { Card, Row, Col, Input } from "antd";
import { useGetCryptosQuery } from "../services/cryptoApi";
import Loader from "./Loader";

const Cryptocurrencies = ({ simplified }) => {
  const count = simplified ? 10 : 100;
  const { data: cryptosList, isFetching } = useGetCryptosQuery(count);
  const [cryptos, setcryptos] = useState([]);
  const [searchTerm, setsearchTerm] = useState("");

  useEffect(() => {
    const filteredData = cryptosList?.data?.coins.filter((item) =>
      item.name.toLowerCase().includes(searchTerm.toLowerCase())
    );
    setcryptos(filteredData);
  }, [cryptosList, searchTerm]);

  if (isFetching) return <Loader />;

  return (
    <>
      {!simplified && (
        <div className="search-crypto">
          <Input
            placeholder="Search Cryptocurrency"
            onChange={(e) => setsearchTerm(e.target.value)}
          />
        </div>
      )}
      <Row gutter={[32, 32]} className="crypto-card-container">
        {cryptos?.map((currency) => {
          return (
            <Col
              xs={24}
              sm={12}
              lg={6}
              className="crypto-card"
              key={currency.uuid}
            >
              <Link key={currency.uuid} to={`/crypto/${currency.uuid}`}>
                <Card
                  extra={
                    <img classname="crypto-image" src={currency.iconUrl} />
                  }
                  title={`${currency.rank}. ${currency.name}`}
                >
                  <p>Price: {millify(currency.price)}</p>
                  <p>Market Cap: {millify(currency.marketCap)}</p>
                  <p>Daily Change: {millify(currency.change)}</p>
                </Card>
              </Link>
            </Col>
          );
        })}
      </Row>
    </>
  );
};

export default Cryptocurrencies;
```

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 (` `) 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.