CRYPTOVERSE

(Cryptocurrency Dashboard)

- **Team Details:**

TEAM LEADER	EMAIL ID
AAISWARYA N V	aaiswaryavenkatesh05@gmail.com

TEAM MEMBERS	EMAIL ID
SHOBANA R	shobana1812r@gmail.com
JOTHIKA S	jothikas791@gmail.com
ABIRAMI U	abirami9342202119@gmail.com

CRYPTOVERSE:

Introduction:

The cryptoverse refers to the expansive digital ecosystem surrounding cryptocurrencies and blockchain technology. It encompasses a various applications, including decentralized finance, virtual worlds, and digital assets, enabling users to engage in innovative financial and creative activities.

Cryptocurrency is a type of digital or virtual currency that uses cryptography for security. It operates on decentralized networks based on blockchain technology, which is a distributed ledger enforced by a network of computers (often referred to as nodes).

Current Trends in the Cryptoverse:

Mass Adoption: Cryptocurrencies are moving beyond rich markets, attracting interest from a broader audience, including everyday consumers and institutional investors.

Environmental Concerns: The energy consumption associated with mining cryptocurrencies has raised concerns, prompting a shift towards more sustainable practices like Proof of Stake.

Educational Initiatives: There is a growing need for educational resources to help the general public understand the complexities of cryptocurrency.

Pre-requisites:

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 application.

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

React.js: It's 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.

- Start the development server:

  npm start

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

✔ 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, that enabling collaboration and tracking changes throughout the development process. We use the Platform GitHub an host your repository.

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

✔ **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits for our preferences, we use Visual Studio Code.

Visual studio code: https://code.visualstudio.com/download.

Install Dependencies:

Navigate into the cloned repository 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.

OUR GITHUB REPOSITORY LINK:

https://github.com/NVAaiswarya/cryptoverse.git

Access the App:

- Open your web browser and navigate tohttp://localhost:5173/
- You should see the our Cryptoverse app's homepage, indicating that the installation and setup were successful.

Now we have successfully installed and set up the application on our local machine. We can now proceed with further customization, development, and testing as needed.

ProjectStructure:

```
PROJECT-1
  crypto
    dist
    node_modules
    public
    src
      app
        store.js
      assets
        cryptocurrency.png
      components
        Cryptocurrencies.jsx
        CryptoDetails.jsx
        Home.jsx
        index.js
        LineChart.jsx
        Loader.jsx
        Navbar.jsx
      services
        cryptoApi.js
      App.css
      App.jsx
      index.css
      main.jsx
    .env
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    README.md
```

Project Flow:

- Project setup and configuration:

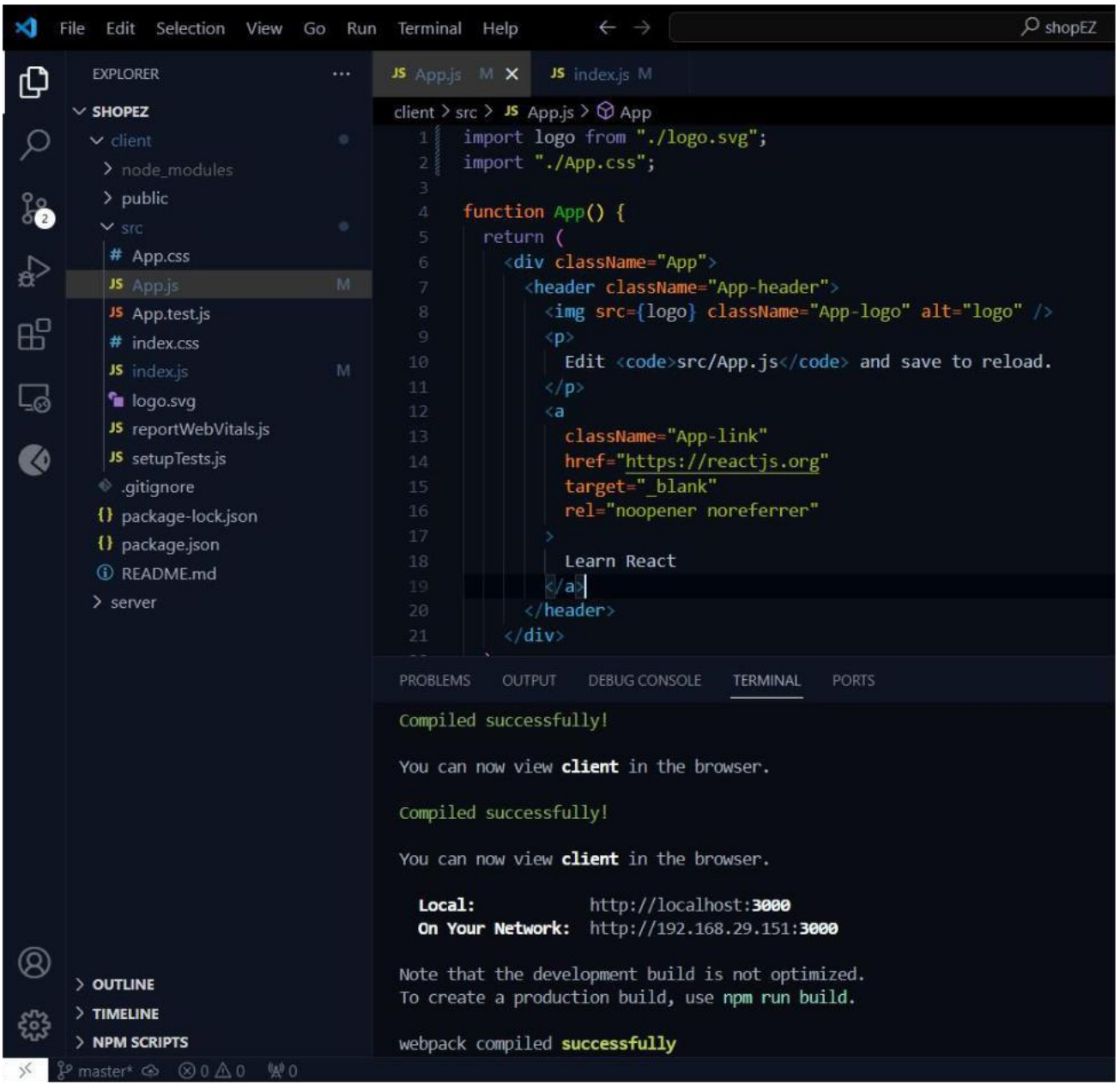
1.SetupReactApplication:
- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

2.DesignUIcomponents:
- Create Components.
- Implement layout and styling.
- Add navigation.

3.Implementfrontendlogic:
- Integration with API endpoints.
- Implement data binding.

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

Reference Image:



ProjectDevelopment:

- **Createareduxstore:**

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.

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

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

, and query functions required for making requests to the cryptocurrency API.

•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 = https://coinranking1.p.rapidapi.com`: 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.

App.jsx | Loader.jsx | cryptoApi.js | Cryptocurrencies.jsx | LineChart.jsx

code > src > services > cryptoApi.js > ...

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

const cryptoApiHeaders = {
  "X-RapidAPI-Key": 'd9ccbe2e00mshe785fadd99ee82cp143ca5jsn2b972854cbd8',
  "X-RapidAPI-Host": 'coinranking1.p.rapidapi.com',
};

const baseUrl = 'https://coinranking1.p.rapidapi.com';

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;
```

•AddingProvidersinthemainfunction:

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:

-

-`<Provider store={store}>`: 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 (`<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>'.

```
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>
);
```

- **CreatingaLinechartcomponent:**

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.

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.

```
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;
```

•Creatingcryptocurrenciescomponent:

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
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;
```

•Createacomponenttoshowthedetailsofcryptocurrency:

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 'usGetCryptoHistoryQuery' 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.

```jsx
import React from "react";
import milify from "millify";
import { Typography, Row, Col, Statistic } from "antd";
import { Link } from "react-router-dom";
const { Title } = Typography;
import { useGetCryptosQuery } from "../services/cryptoApi";
import Cryptocurrencies from "./Cryptocurrencies";
import Loader from "./Loader";

const Home = () => {
  const { data, isFetching } = useGetCryptosQuery(10);
  if (isFetching) return <Loader />;
  const globalStats = data?.data?.stats;

  return (
    <>
      <Title level={2} className="heading">
        Global Crypto Stats
      </Title>
      <Row>
        <Col span={12}>
          <Statistic title="Total Cryptocurrencies" value={globalStats.total} />
        </Col>
        <Col span={12}>
          <Statistic
            title="Total Exchanges"
            value={milify(globalStats.totalExchanges)}
          />
        </Col>
        <Col span={12}>
          <Statistic
            title="Total Market Cap"
            value={milify(globalStats.totalMarketCap)}
          />
        </Col>
        <Col span={12}>
          <Statistic
            title="Total 24h Volume"
            value={milify(globalStats.total24hVolume)}
          />
        </Col>
        <Col span={12}>
          <Statistic
            title="Total Markets"
            value={milify(globalStats.totalMarkets)}
          />
        </Col>
      </Row>
      <div className="home-heading-container">
        <Title level={2} className="home-title">
          Top 10 Cryptocurrencies in the world
        </Title>
        <Title level={3} className="show-more">
          <Link to="/cryptocurrencies">Show More</Link>
        </Title>
      </div>
      <Cryptocurrencies simplified />
    </>
  );
};

export default Home;
```

ProjectExecution:

Here is the video link of react application execution:
https://drive.google.com/file/d/1hbDljCv0R_a_lLrRCh-8yMea-WCvKP1e/view?usp=sharing

Project demo:

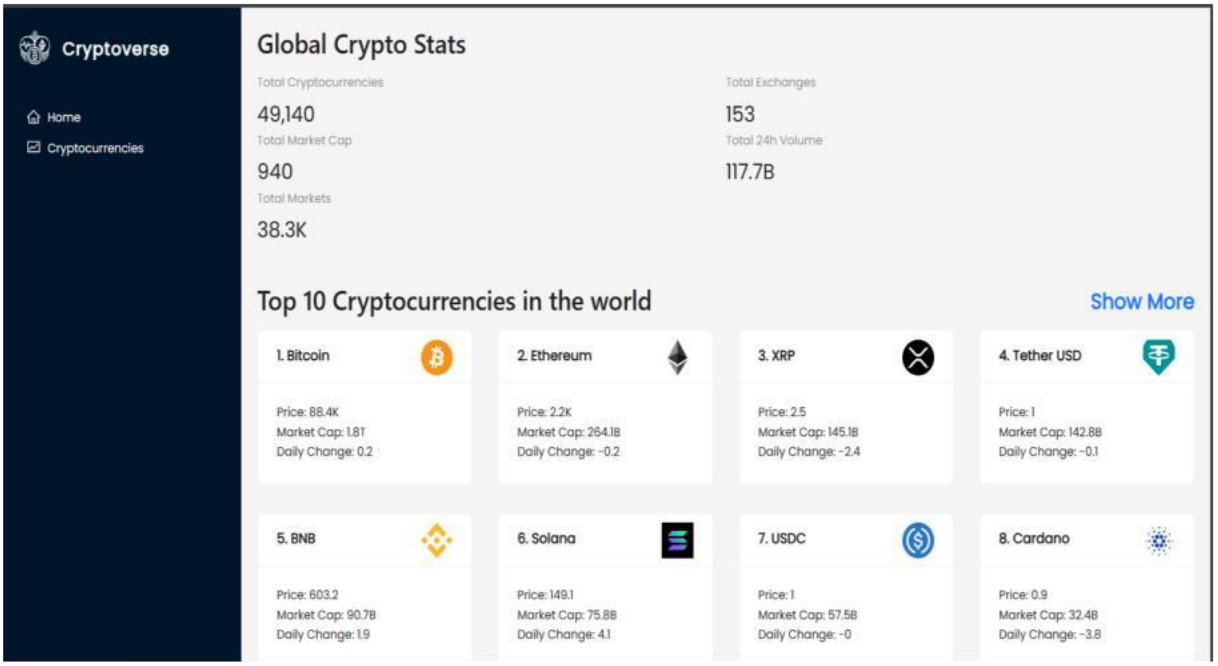
Demo link:

https://drive.google.com/file/d/1absGBe5nv4s6WXi1QpSaICCDWgTbvZ1r/view?usp=drivesdk

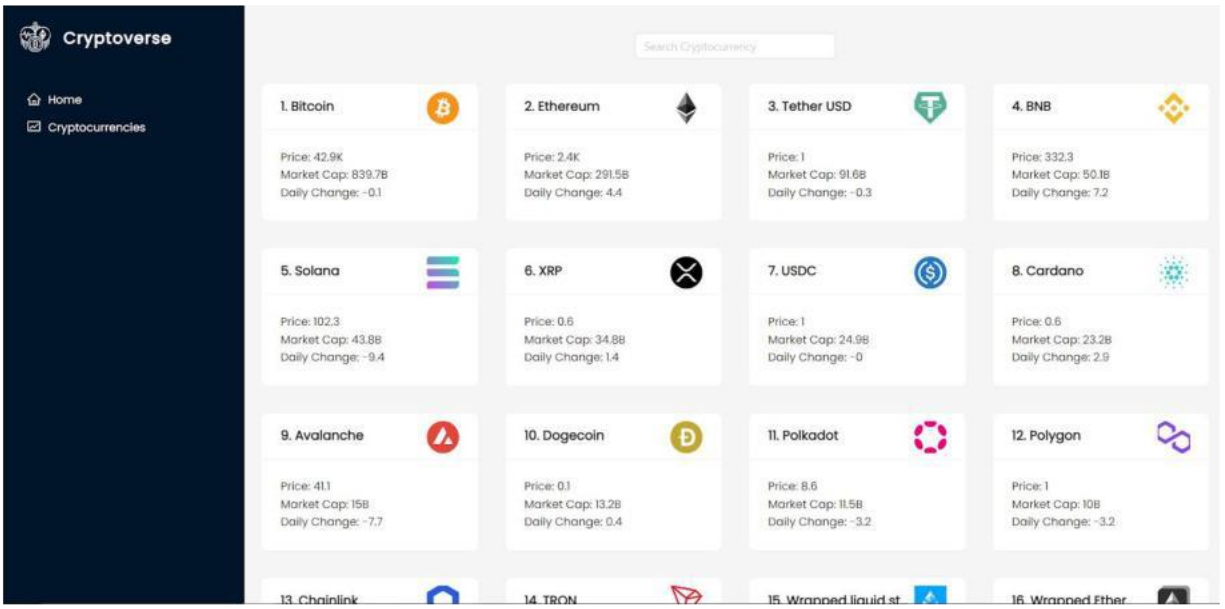
---

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.



This page contains the line chart with data representation of price of cryptocurrencies. Also contains statistics and website links of cryptocurrencies.

Dogecoin value statistics

An overview showing the stats of Dogecoin

Price to USD	$ 0.2
Rank	9
24h Volume	$ 834.8M
Market Cap	$ 28B
All-time-high(daily avg.)	$ 0.8

Others statistics

An overview showing the stats of all cryptocurrencies

Number Of Markets	167
Number Of Exchanges	75
Aprroved Supply	✓
Total Supply	$ 147.2B
Circulating Supply	$ 147.2B

What is Dogecoin ?

Dogecoin is a decentralized, peer-to-peer digital currency created as a joke but now used for charitable causes, founded by Billy Markus, Jackson Palmer, and "Shibetoshi Nakamoto".

Dogecoin Links

Website	dogecoin.com
Bitcointalk	bitcointalk.org
Facebook	Facebook

_____THANK YOU_____