

Cryptoverse

Introduction

A Cryptocurrency dashboard displaying five years of historical price data is a valuable tool for investors to track market trends and performance. Interactive charts and customizable timeframes allow for easy comparison of top performers and detailed analysis of specific periods. By examining past price fluctuations, users can identify patterns, assess volatility, and make informed decisions. This feature helps optimize portfolios, manage risk, and gain a deeper understanding of the evolving crypto market, ultimately boosting confidence in navigating its volatility.

Team Members:

The Cryptoverse project was developed by a team of four members:

1. DEVIKA K - kdevika430@gmail.com
2. KAMAKSHI J - kamakshij0503@gmail.com
3. GOPIKA M - gopikamurugesan4@gmail.com
4. GAYATHRI C - gayathric565@gmail.com

Project Overview

Purpose:

Cryptoverse aims to provide a comprehensive and user-friendly platform for cryptocurrency investors to track, analyze, and compare historical price data over the past five years. The primary purpose is to help users make informed investment decisions, optimize their portfolios, and understand market trends and volatility. By offering a detailed overview of past market performance, Cryptoverse supports users in managing risk, predicting future shifts, and navigating the dynamic and often unpredictable world of cryptocurrency trading.

Features:

1. Historical Price Data:

Access to five years of historical price data for a wide range of cryptocurrencies, helping users identify long-term trends and performance patterns.

2. Interactive Charts:

Visual, interactive charts that allow users to compare price fluctuations across different timeframes, making it easy to identify top-performing assets.

3. Customizable Timeframes:

Users can select specific timeframes to analyze detailed market data, facilitating focused assessments of volatility, price cycles, and other trends.

4. Robust Search Functionality:

An intuitive search feature to explore and compare various cryptocurrencies, allowing users to quickly find assets of interest and evaluate their historical performance.

5. Portfolio Optimization:

Tools that help users optimize their cryptocurrency portfolios based on historical data, risk assessments, and performance trends.

6. Educational Resources:

The platform also serves as an educational tool, helping users gain a deeper understanding of the crypto market's behavior and how to make strategic investment choices.

7. User-Friendly Interface:

A simple, intuitive interface designed for both novice and experienced investors, ensuring that all users can easily navigate and access the data and tools they need.

Setup Instructions

To set up a React project using Vite instead of create-react-app, follow these steps: Install Vite and create a new React app:

✓ Create a new Vite project with React:

Run the following command to set up a new Vite project with React:

```
npm create vite@latest my-vite-react-app --template react
```

Replace my-vite-react-app with your preferred project name.

✓ Navigate to the project directory:

```
cd my-vite-react-app
```

Running the React App with Vite:

✓ Install dependencies:

After creating the project, install the necessary dependencies: `npm install`

✓ Start the development server:

Run the following command to start the development server: `npm run dev`

This will launch the Vite development server, and you can access your React app at <http://localhost:5173> in your 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, 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:

- ✓ Choose a code editor or Integrated Development 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/drive/folders/188_vxhyfntQtqv9JV3D_yQc03aU6DYQR?usp=sharing

✓Clone the code from github repository: Follow below steps:

Git repository: <https://github.com/DK2005DK/Cryptoverse-.git>

Git clone command: `git clone https://github.com/DK2005DK/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 <https://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.

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

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. 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.

6. 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.

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.

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

React Router with ``BrowserRouter``:

1. ``<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>``.

- **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`.

2. 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.

3. Chart Data:

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

4. 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.

5. 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.

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.

- **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``.

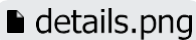
9. Rendering Description and Links:

- 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.

Reference Image 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:

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.`

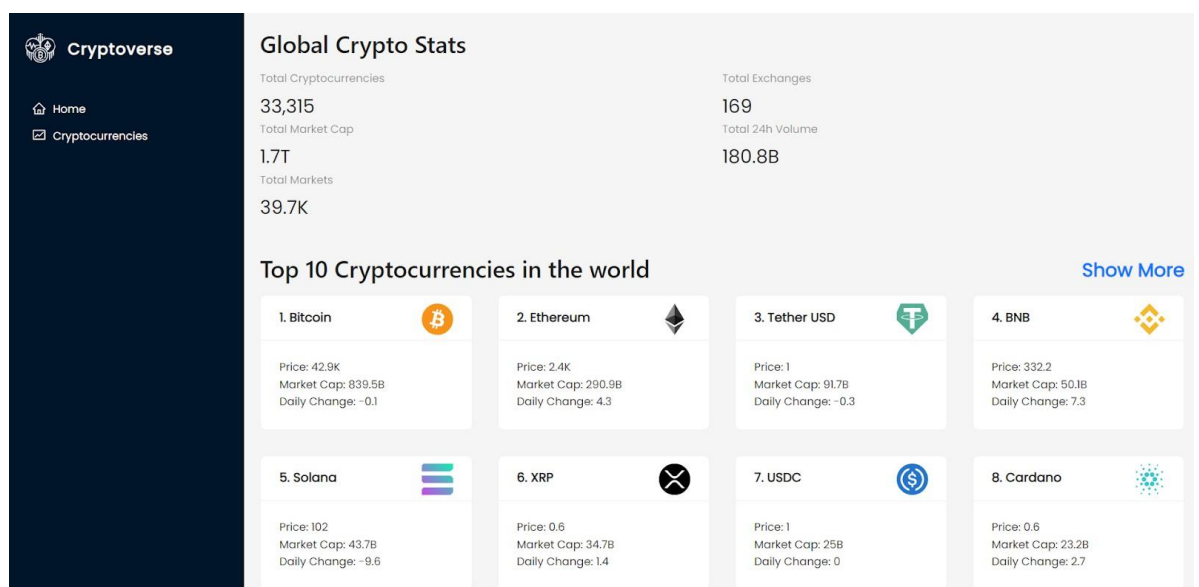
Project Demo:

Demo Link: <https://drive.google.com/file/d/10X0bkqwb5lza4V9zoRdEM6li1qPRljWr/view?usp=sharing>

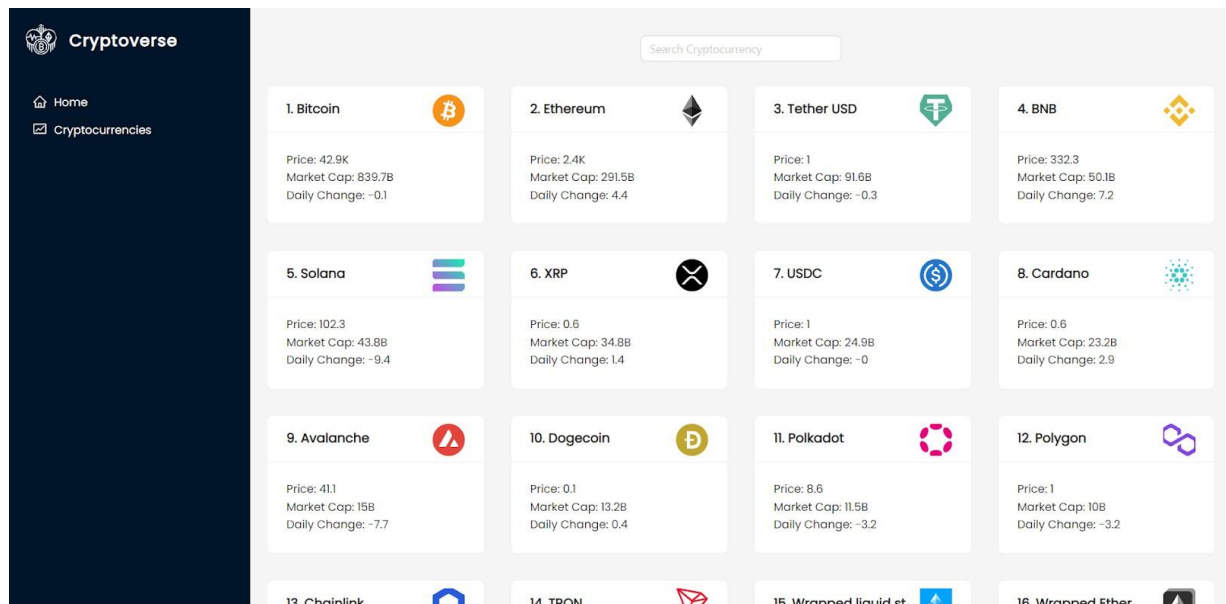
Github link: <https://github.com/DK2005DK/Cryptoverse-.git>

User Interface snips:

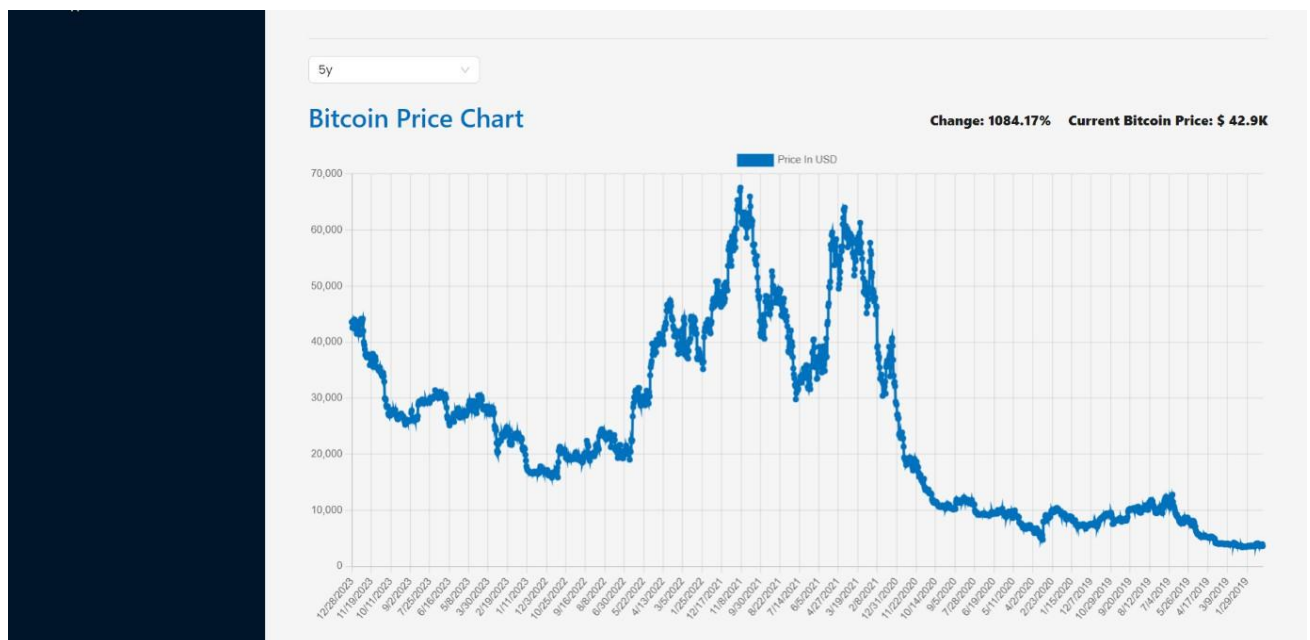
- Home page : This pages consists of stats of global crypto like total cryptocurrencies, total exchanges, market cap etc. Also consist of top 10 cryptocurrencies in the world.



□ 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.



****Happy Coding****

The Project Developed by

- Gayathri.C : Developed the "Cryptocurrencies" component
 - Kamakshi.J : Implemented the User Interface
 - Gopika.M : Implemented services ie. "CryptoAPI"
 - Devika.K : (Team Lead): Documented the project, including remote video, user interface, and Error correction.
-