Introduction to Cryptoverse Project

The Cryptoverse project is a React (TypeScript) + Vite web application designed to provide real-time cryptocurrency data, news, and detailed insights into various digital assets. The project follows a modular architecture with API integration, state management, and a clean UI built with Tailwind CSS.

The team members of the project are:

* Hema Priya dhrashini M
* Indhumathi G
* Velan V
* Nethaji I

Key Features

1. Cryptocurrency Market Data – Displays live data on different cryptocurrencies using API integration.

2. Coin Details & Charts – Shows price trends, charts, and historical data.

3. Latest Crypto News – Fetches and presents the latest crypto-related news.

4. User-friendly Interface – Optimized UI using Tailwind CSS.

5. Modular & Scalable Codebase – Well-structured React components with reusable UI elements.

6. Dark Mode Support – Ensures a smooth viewing experience.

7. Optimized Performance – Built with Vite for fast development and production builds.

Tech Stack

Frontend: React (TypeScript)

Styling: Tailwind CSS

Build Tool: Vite

State Management: Context API

API Integration: Fetching live cryptocurrency data

Routing: React Router

Project Description: Cryptoverse

The Cryptoverse project is a cryptocurrency tracking web application built with React (TypeScript) and Vite. It provides users with real-time cryptocurrency data, market insights, and the latest news in the crypto world. The project is designed to be fast, responsive, and user-friendly, offering a seamless experience for crypto enthusiasts and investors.

Core Functionalities

Live Cryptocurrency Prices & Market Trends

Users can view real-time price updates, market capitalization, trading volume, and historical trends of different cryptocurrencies.

Detailed Coin Information

Each cryptocurrency has a dedicated page with detailed information, including interactive charts and price fluctuations.

Latest Crypto News

The app integrates with crypto news sources to display the most recent news articles related to cryptocurrencies and blockchain technology.

Search & Filtering

Users can search for specific cryptocurrencies and filter the data based on their preferences.

Modern UI with Dark Mode Support

The interface is designed using Tailwind CSS, providing a clean, responsive, and visually appealing experience, including dark mode support.

Technical Overview

Frontend: React (TypeScript)

UI Framework: Tailwind CSS

Build Tool: Vite

State Management: Context API

API Integration: Fetching real-time data from cryptocurrency APIs

Routing: React Router

This project is structured in a way that ensures scalability and maintainability, with modular components, reusable UI elements, and efficient API handling.

Scenario-Based Introduction to Cryptoverse

In today's rapidly evolving digital financial landscape, cryptocurrency trading and investment have become mainstream. With thousands of cryptocurrencies available and market prices fluctuating every second, investors and enthusiasts need a reliable platform to stay updated. This is where Cryptoverse comes in—a real-time cryptocurrency tracking and information hub designed to provide live market data, detailed coin insights, and the latest crypto news in a user-friendly interface.

Key Objectives of Cryptoverse

1. Real-Time Market Tracking

Users can monitor the latest price updates, market trends, and key statistics of various cryptocurrencies in real time.

Interactive charts and historical data help users make informed decisions.

2. Comprehensive Coin Analysis

Each cryptocurrency has a dedicated page with in-depth details, including price fluctuations, historical performance, and essential financial metrics.

3. Crypto News Aggregation

The app fetches the latest news articles related to blockchain, DeFi, and cryptocurrencies, helping users stay ahead in the industry.

4. User-Friendly & Accessible Interface

Built with React (TypeScript) and Tailwind CSS, the application provides a fast, responsive, and visually appealing user experience.

Dark mode support ensures a comfortable viewing experience.

5. Scalable & Efficient Performance

Using Vite for fast builds and optimized performance, Cryptoverse ensures seamless navigation and quick data retrieval.

Modular code structure allows for easy scalability and future enhancements.

Use Case Scenario

Imagine a crypto investor looking to analyze Bitcoin’s market trend before making a trading decision. Instead of visiting multiple sources, they can simply open Cryptoverse, search for Bitcoin, and instantly access live price updates, historical charts, and recent news—all in one place. This allows them to make informed investment choices with confidence.

Features of Cryptoverse

Cryptoverse is a modern cryptocurrency tracking application designed to provide real-time market insights, historical data, and the latest news. Below are the key features of the platform:

1. Live Cryptocurrency Market Data

Displays real-time price updates, market capitalization, trading volume, and percentage changes.

Supports multiple cryptocurrencies, including Bitcoin (BTC), Ethereum (ETH), and other altcoins.

2. Detailed Coin Insights & Analysis

Each cryptocurrency has a dedicated page displaying:

Live price charts with historical performance.

Market statistics like supply, market cap, and trading volume.

Price fluctuation trends over different time periods (24h, 7d, 30d).

3. Latest Crypto News Aggregation

Fetches the latest news articles from trusted crypto news sources.

News updates include market trends, regulatory changes, and blockchain innovations.

4. Cryptocurrency Search & Filtering

Users can search for specific cryptocurrencies using a search bar.

Advanced filtering options allow users to sort by market cap, volume, or price change.

5. Modern UI with Dark Mode Support

Built using Tailwind CSS, providing a sleek and intuitive interface.

Dark mode option for a comfortable user experience.

6. Interactive Charts & Visualizations

Users can view interactive line charts that show cryptocurrency price trends.

Provides a clear visualization of market movements.

7. Mobile-Responsive Design

The website is fully responsive, making it accessible on mobile, tablet, and desktop devices.

8. Fast Performance & Optimized Build

Uses Vite for faster loading times and improved performance.

Ensures smooth navigation and quick data retrieval.

9. Scalable & Modular Architecture

Built with React (TypeScript), making the codebase easy to maintain and expand.

Uses a Context API for state management, ensuring seamless data flow.

10. Secure & Reliable

Fetches real-time data from cryptocurrency APIs, ensuring accuracy and reliability.

Follows best practices for data security and user privacy.

---

Future Enhancements (Planned Features)

User Authentication & Portfolio Tracking – Allow users to log in and track their crypto holdings.

Price Alerts & Notifications – Set alerts for significant price movements.

AI-Based Market Predictions – Use AI algorithms to suggest possible market trends.

Technical Architecture of Cryptoverse

Cryptoverse is built using a modern frontend tech stack, ensuring scalability, performance, and maintainability. Below is the detailed breakdown of its technical architecture:

---

1. High-Level Architecture

Cryptoverse follows a three-layer architecture:

a) Presentation Layer (Frontend)

Built with: React (TypeScript) + Tailwind CSS

Handles:

UI rendering

User interactions

Data visualization (charts, tables, and filters)

Components:

Navbar (Search & Filtering)

CoinDetail (Individual Crypto Data)

News (Latest Crypto News)

b) Data Layer (State Management & API Handling)

Context API for global state management

RESTful API calls to fetch cryptocurrency data

Axios / Fetch API for network requests

c) Backend & Data Sources

Cryptocurrency API (e.g., CoinGecko, CoinMarketCap)

Provides real-time market data, historical trends, and financial metrics

News API

Aggregates the latest cryptocurrency news from different sources

---

2. Technology Stack

---

3. Detailed System Architecture

a) Component-Based UI Design

The UI is built using modular, reusable React components.

Each feature (crypto list, news, charts) is separated into dedicated components under src/components/.

b) API Integration Layer

API.tsx handles all external API requests.

Uses Axios or Fetch API for efficient data fetching.

Data is structured and passed through Context API for state management.

c) State Management Flow

Context API stores fetched crypto and news data.

Components subscribe to state changes and update automatically when new data arrives.

d) Routing & Navigation

React Router handles navigation between different pages (MainPage.tsx, CoinDetail.tsx, News.tsx).

---

4. Folder Structure Overview

/Cryptoverse-main

├── public/ # Static assets (logos, images)

├── src/

│ ├── API/ # API handling (RESTful requests)

│ ├── Context/ # Global state management (Context API)

│ ├── Utils/ # Helper functions (formatting, currency conversion)

│ ├── assets/ # Additional media files

│ ├── components/ # Reusable UI components

│ ├── pages/ # Page-specific components (Cryptocurrency, News, CoinDetail)

│ ├── App.tsx # Main application entry point

│ ├── main.tsx # React root rendering file

│ ├── index.css # Global styles

├── package.json # Project dependencies

├── vite.config.ts # Vite build configuration

├── tailwind.config.js # Tailwind CSS configuration

└── tsconfig.json # TypeScript configuration

---

5. Security & Optimization

✅ Optimized Performance:

Uses Vite for fast builds and lazy loading to improve loading times.

API requests are cached to reduce redundant network calls.

✅ Security Measures:

API keys are stored securely in .env files (if applicable).

Implements CORS policies to protect data exchanges.

---

6. Scalability & Future Enhancements

Scalability:

Well-structured modular components ensure easy expansion.

Context API can be replaced with Redux or Zustand for larger applications.

Future Features:

User Authentication: Allow users to log in and track their crypto portfolios.

AI-Powered Insights: Implement AI models to predict price movements.

WebSockets for Real-Time Data: Fetch live price updates without refreshing the page.

Cryptoverse is a modern, scalable, and high-performance cryptocurrency tracking web app, designed with React (TypeScript), Vite, Tailwind CSS, and API integrations. Its modular architecture, fast build times, and optimized UI make it a powerful tool for crypto enthusiasts and traders.

Pre-requisites for Running Cryptoverse

Before setting up and running the Cryptoverse project, ensure that your system meets the following requirements.

---

1. System Requirements

✅ Operating System: Windows, macOS, or Linux

✅ Node.js: Version 16+ (Recommended: Latest LTS version)

✅ Package Manager: npm (default with Node.js) or yarn

---

2. Software & Tools Required

---

3. API Keys (If Required)

If the project fetches real-time cryptocurrency data, you may need an API key from providers like:

CoinGecko API → Register Here

CoinMarketCap API → Register Here

API keys should be stored in a .env file inside the project root (if applicable).

---

4. Installation & Setup Guide

Step 1: Clone the Repository

git clone https://github.com/your-repo/Cryptoverse.git

cd Cryptoverse

Step 2: Install Dependencies

npm install

or

yarn install

Step 3: Set Up Environment Variables

If the project uses an API, create a .env file in the root directory and add:

VITE\_API\_KEY=your\_api\_key\_here

Step 4: Start the Development Server

npm run dev

or

yarn dev

This will launch the application at http://localhost:3000/ (or another available port).

---

5. Optional Dependencies (For Advanced Development)

Redux or Zustand (For state management)

Postman (For API testing)

Docker (For containerized deployment)

---

6. Deployment Requirements

Vercel or Netlify for easy hosting

Node.js and npm installed on the server

Build the project for production:

npm run build

If you want to get the Cryptoverse project from Google Drive, follow these steps:

1. Download the Project from Google Drive

Open Google Drive in your browser.

Locate the Cryptoverse-main.zip file.

Right-click → Download the ZIP file to your local system.

---

2. Extract the Project Files

Once downloaded, extract the ZIP file:

Windows: Right-click the ZIP file → Extract All

Mac/Linux: Use the unzip command in the terminal:

unzip Cryptoverse-main.zip

---

3. Open the Project in VS Code

Open VS Code

Click File → Open Folder

Select the extracted Cryptoverse-main folder

---

4. Install Dependencies

Navigate to the project directory in the terminal:

cd Cryptoverse-main

npm install

---

5. Run the Application

npm run dev

The application should now be accessible in your browser at http://localhost:3000/.

Project Flow of Cryptoverse

The Cryptoverse application follows a structured frontend flow that ensures smooth navigation, real-time data fetching, and user-friendly interaction. Below is the step-by-step project flow from startup to user interaction.

---

1. Application Startup

Step 1: User Opens the Application

The user visits Cryptoverse in their browser.

If running locally → http://localhost:3000/

If deployed → https://your-deployed-site.com/

Step 2: Application Initialization

The main.tsx file mounts the React App and renders the App.tsx component.

The global styles (index.css) and Tailwind configurations are applied.

The Context API is initialized to manage global state.

---

2. Home Page (MainPage.tsx)

Step 3: UI Renders the Dashboard

The homepage displays:

✅ Hero Banner – Welcoming UI with an introduction to Cryptoverse.

✅ Top Cryptocurrencies – Fetches live market data and shows trending coins.

✅ Latest Crypto News – Fetches and displays news articles.

Step 4: API Requests & Data Fetching

The API.tsx file makes REST API calls to fetch:

✅ Live cryptocurrency prices & trends

✅ Historical price data for charts

✅ Latest crypto news

---

3. Cryptocurrency Market (Cryptocurrency.tsx)

Step 5: User Browses the Market

Displays a list of cryptocurrencies with market stats.

Users can:

✅ Search for a specific cryptocurrency.

✅ Filter by price, market cap, or volume.

✅ Click on a coin to view its detailed page.

---

4. Detailed Coin View (CoinDetail.tsx)

Step 6: User Views a Specific Coin

When a user clicks on a coin, they are taken to:

/coin/:id

The page fetches:

✅ Live price updates

✅ Interactive price chart (via Chart.js)

✅ Market statistics (market cap, supply, volume, etc.)

---

5. Crypto News Section (News.tsx)

Step 7: User Reads Latest News

The News API fetches the latest articles.

Displays news cards with:

✅ Headlines

✅ Source & timestamp

✅ "Read More" button (redirects to full article)

---

6. User Interaction & Navigation

Step 8: Navbar & Navigation

The Header/Navbar allows users to switch between:

✅ Home (/)

✅ Cryptocurrency Market (/cryptocurrency)

✅ News (/news)

Step 9: Mobile & Dark Mode Support

The UI adapts for mobile devices using Tailwind CSS.

Users can toggle dark mode for a better viewing experience.

---

7. Performance & Optimization

Step 10: Data Caching & API Optimization

API responses are cached to reduce redundant calls.

Lazy loading ensures only required components are loaded when needed.

---

8. Deployment & Hosting

Step 11: Deployment to Production

The project is built using:

npm run build

Hosted on Vercel or Netlify for easy access.

---

🔹 Summary of the Cryptoverse Project Flow

1️⃣ App Initialization → Load UI components & styles

2️⃣ Fetch & Display Market Data → API calls for crypto prices & news

3️⃣ User Navigates the App → Home, Cryptos, Coin Details, News

4️⃣ Interactive Features → Search, filter, charts, and news

5️⃣ Performance Optimizations → Data caching & lazy loading

6️⃣ Deployment & Hosting → Running on Vercel/Netlify

If you're looking for a live demo of the Cryptoverse project, here are some options based on your needs:

1. Running a Local Demo

If you have the project files, follow these steps to run the project on your local machine:

Step 1: Navigate to the Project Folder

cd Cryptoverse-main

Step 2: Install Dependencies

npm install

Step 3: Start the Development Server

npm run dev

Open your browser and visit: http://localhost:3000/

---

2. Hosting the Project for an Online Demo

If you want to make the project accessible online, you can deploy it to Vercel or Netlify:

Deploy on Vercel (Recommended)

1. Install Vercel CLI:

npm install -g vercel

2. Deploy the project:

vercel

3. Vercel will provide a live demo link (e.g., https://cryptoverse.vercel.app/).

Deploy on Netlify

1. Install Netlify CLI:

npm install -g netlify-cli

2. Build and deploy:

npm run build

netlify deploy --prod

3. Netlify will generate a live URL for your project.

---

3. Video Demo & Walkthrough

If you want to record a project demo, you can use:

OBS Studio (for screen recording)

Loom (for quick video sharing)

Zoom/Google Meet (for live demonstration)

Here are some important code snippets from the Cryptoverse project, categorized by functionality:

---

1️⃣ API Handling (API.tsx)

This file is responsible for fetching cryptocurrency data from an external API (e.g., CoinGecko).

import axios from "axios";

const API\_URL = "https://api.coingecko.com/api/v3";

export const fetchCryptoData = async () => {

try {

const response = await axios.get(`${API\_URL}/coins/markets`, {

params: {

vs\_currency: "usd",

order: "market\_cap\_desc",

per\_page: 10,

page: 1,

},

});

return response.data;

} catch (error) {

console.error("Error fetching crypto data:", error);

return [];

}

};

✅ What It Does?

Calls the CoinGecko API to fetch real-time cryptocurrency data.

Returns a list of top 10 cryptocurrencies ranked by market cap.

---

2️⃣ Global State Management (AppContext.tsx)

This Context API handles global state for cryptocurrencies.

import { createContext, useState, useEffect } from "react";

import { fetchCryptoData } from "../API/API";

export const AppContext = createContext(null);

export const AppProvider = ({ children }) => {

const [cryptoList, setCryptoList] = useState([]);

useEffect(() => {

const getData = async () => {

const data = await fetchCryptoData();

setCryptoList(data);

};

getData();

}, []);

return (

<AppContext.Provider value={{ cryptoList }}>

{children}

</AppContext.Provider>

);

};

✅ What It Does?

Fetches cryptocurrency data once on app load.

Stores it in cryptoList, making it accessible globally.

---

3️⃣ Display Cryptocurrency List (Cards.tsx)

This component displays the list of top cryptocurrencies.

import { useContext } from "react";

import { AppContext } from "../../Context/AppContext";

const Cards = () => {

const { cryptoList } = useContext(AppContext);

return (

<div className="grid grid-cols-3 gap-4">

{cryptoList.map((coin) => (

<div key={coin.id} className="p-4 border rounded-lg shadow-md">

<img src={coin.image} alt={coin.name} className="w-10 h-10" />

<h2 className="text-lg font-bold">{coin.name} ({coin.symbol.toUpperCase()})</h2>

<p>Price: ${coin.current\_price.toFixed(2)}</p>

<p>Market Cap: ${coin.market\_cap.toLocaleString()}</p>

</div>

))}

</div>

);

};

export default Cards;

✅ What It Does?

Loops through cryptoList and displays crypto cards with price & market cap.

---

4️⃣ Coin Detail Page (CoinDetail.tsx)

This page displays detailed information about a selected coin.

import { useState, useEffect } from "react";

import { useParams } from "react-router-dom";

import axios from "axios";

const CoinDetail = () => {

const { id } = useParams();

const [coin, setCoin] = useState(null);

useEffect(() => {

const fetchCoinDetail = async () => {

const response = await axios.get(`https://api.coingecko.com/api/v3/coins/${id}`);

setCoin(response.data);

};

fetchCoinDetail();

}, [id]);

if (!coin) return <p>Loading...</p>;

return (

<div className="p-6">

<h1 className="text-2xl font-bold">{coin.name} ({coin.symbol.toUpperCase()})</h1>

<img src={coin.image.large} alt={coin.name} className="w-20 h-20" />

<p>Current Price: ${coin.market\_data.current\_price.usd.toFixed(2)}</p>

<p>Market Cap: ${coin.market\_data.market\_cap.usd.toLocaleString()}</p>

</div>

);

};

export default CoinDetail;

✅ What It Does?

Fetches details for a single coin based on id.

Displays name, image, price, and market cap.

---

5️⃣ Navigation (Header.tsx)

This is the navigation bar for switching between pages.

import { Link } from "react-router-dom";

const Header = () => {

return (

<nav className="flex justify-between p-4 bg-gray-800 text-white">

<h1 className="text-2xl font-bold">Cryptoverse</h1>

<div>

<Link to="/" className="mx-2">Home</Link>

<Link to="/cryptocurrency" className="mx-2">Market</Link>

<Link to="/news" className="mx-2">News</Link>

</div>

</nav>

);

};

export default Header;

✅ What It Does?

Uses React Router for easy navigation.

Links to Home, Market, and News pages.

---

6️⃣ Dark Mode Toggle (button.tsx)

This allows users to switch between Light & Dark Mode.

import { useState } from "react";

const DarkModeToggle = () => {

const [darkMode, setDarkMode] = useState(false);

const toggleMode = () => {

setDarkMode(!darkMode);

document.documentElement.classList.toggle("dark");

};

return (

<button onClick={toggleMode} className="p-2 bg-gray-700 text-white rounded">

{darkMode ? "Light Mode" : "Dark Mode"}

</button>

);

};

export default DarkModeToggle;

✅ What It Does?

Toggles dark mode using Tailwind’s dark mode feature.

---

7️⃣ News Section (News.tsx)

This fetches & displays latest crypto news.

import { useState, useEffect } from "react";

import axios from "axios";

const News = () => {

const [news, setNews] = useState([]);

useEffect(() => {

const fetchNews = async () => {

const response = await axios.get("https://cryptonews-api-url.com/news");

setNews(response.data.articles);

};

fetchNews();

}, []);

return (

<div className="p-6">

<h2 className="text-2xl font-bold">Latest Crypto News</h2>

{news.map((article, index) => (

<div key={index} className="border-b p-4">

<h3 className="text-lg font-semibold">{article.title}</h3>

<p>{article.description}</p>

<a href={article.url} target="\_blank" rel="noopener noreferrer" className="text-blue-500">Read More</a>

</div>

))}

</div>

);

};

export default News;

✅ What It Does?

Fetches crypto news articles and displays them.

Each article has a title, description, and "Read More" link.

These are some of the important code snippets that power Cryptoverse.

Would you like more details on any specific feature or a step-by-step implementation of a particular part

Breakdown of Cryptoverse Code – Dependencies

The dependencies in the Cryptoverse project are listed in the package.json file. Below is a breakdown of each dependency and its purpose.

---

1️⃣ Main Dependencies (Project Core)

These are required to build and run the application.

✅ Why these dependencies?

React + React-DOM → Required for building UI components.

React Router → Handles routing between pages (/, /coin/:id, /news).

Axios → Simplifies API calls to fetch real-time crypto data.

Tailwind CSS → Provides responsive and fast styling.

Vite → Optimizes the build process for better performance.

---

2️⃣ Development Dependencies

These are used only during development to improve code quality and performance.

✅ Why these dependencies?

TypeScript → Ensures type safety to catch errors before runtime.

ESLint + Prettier → Improves code quality and maintains a consistent style.

Vite Plugin for ESLint → Runs linting automatically when using Vite.

---

3️⃣ UI & Utility Dependencies

Additional packages that enhance user experience.

✅ Why these dependencies?

Chart.js → Displays historical price trends with interactive charts.

Heroicons → Provides SVG icons for buttons and navigation.

Headless UI → Helps create accessible dropdowns, modals, and toggles.

---

4️⃣ Optional Dependencies (For Future Enhancements)

These may not be in the current package.json, but they could be useful for future features:

---

🔹 How to Install Dependencies?

After cloning the project, install dependencies using:

npm install

or

yarn install

Libraries Used in Cryptoverse

The Cryptoverse project utilizes several libraries to ensure a smooth, efficient, and scalable application. Below is a breakdown of the main libraries used in the project and their purpose.

---

1️⃣ Core Libraries (React & Routing)

These are essential for building the React-based frontend and handling navigation.

✅ How It’s Used?

React: Used to create UI components (e.g., <App />, <Header />, <Cards />).

React Router: Enables navigation between pages (/, /cryptocurrency, /news, /coin/:id).

---

2️⃣ API Handling & Data Fetching

These libraries are used to interact with external APIs to fetch cryptocurrency data and news.

✅ How It’s Used?

axios.get() is used to fetch live crypto prices from APIs like CoinGecko.

Example:

import axios from "axios";

const fetchCryptoData = async () => {

const response = await axios.get("https://api.coingecko.com/api/v3/coins/markets", {

params: { vs\_currency: "usd", order: "market\_cap\_desc", per\_page: 10, page: 1 },

});

return response.data;

};

---

3️⃣ UI & Styling (Tailwind CSS)

Used to create a modern, responsive design with minimal CSS writing.

✅ How It’s Used?

Defines styles using utility classes:

<div className="p-4 bg-gray-800 text-white rounded-lg shadow-md">

Hello, Cryptoverse!

</div>

---

4️⃣ Charts & Data Visualization

These libraries handle interactive charts for price trends.

✅ How It’s Used?

Displays historical price trends for cryptocurrencies.

Example:

import { Line } from "react-chartjs-2";

const data = {

labels: ["Jan", "Feb", "Mar"],

datasets: [{ label: "Bitcoin", data: [40000, 45000, 42000], borderColor: "blue" }],

};

return <Line data={data} />;

---

5️⃣ Icons & UI Enhancements

Used to add icons and improve UI accessibility.

✅ How It’s Used?

Heroicons Example:

import { SearchIcon } from "@heroicons/react/solid";

<SearchIcon className="w-6 h-6 text-gray-500" />;

---

6️⃣ State Management (Context API)

Handles global state across the app.

✅ How It’s Used?

Stores crypto market data globally and shares it across components.

import { createContext, useContext, useState } from "react";

const AppContext = createContext();

export const AppProvider = ({ children }) => {

const [cryptoList, setCryptoList] = useState([]);

return <AppContext.Provider value={{ cryptoList }}>{children}</AppContext.Provider>;

};

export const useCrypto = () => useContext(AppContext);

---

7️⃣ Development Tools

These libraries help with TypeScript, linting, and performance optimization.

✅ How It’s Used?

ESLint + Prettier automatically formats and enforces best coding practices.

Vite is used instead of Webpack for faster builds.

---

🔹 Summary of Libraries Used in Cryptoverse

1️⃣ Core Libraries → react, react-dom, react-router-dom

2️⃣ API Handling → axios

3️⃣ UI & Styling → tailwindcss

4️⃣ Charts & Data Visualization → chart.js, react-chartjs-2

5️⃣ Icons & UI Enhancements → @heroicons/react, @headlessui/react

6️⃣ State Management → Context API (built-in React feature)

7️⃣ Development Tools → vite, typescript, eslint, prettier

Fetching Data Functions in Cryptoverse

In the Cryptoverse project, data is primarily fetched from external APIs to display real-time cryptocurrency prices, market trends, and news. Below are the main data fetching functions, categorized by their usage.

---

1️⃣ Fetching Cryptocurrency Market Data

This function retrieves live price updates of cryptocurrencies from the CoinGecko API.

📌 fetchCryptoData() - Get Market Prices

import axios from "axios";

const API\_URL = "https://api.coingecko.com/api/v3";

export const fetchCryptoData = async () => {

try {

const response = await axios.get(`${API\_URL}/coins/markets`, {

params: {

vs\_currency: "usd", // Currency in USD

order: "market\_cap\_desc", // Sort by market cap

per\_page: 10, // Get top 10 cryptocurrencies

page: 1,

},

});

return response.data; // Return fetched crypto data

} catch (error) {

console.error("Error fetching crypto data:", error);

return []; // Return empty array if error occurs

}

};

✅ What It Does?

Calls CoinGecko API to fetch real-time market prices.

Retrieves top 10 cryptocurrencies sorted by market cap.

Returns an array of crypto data.

---

2️⃣ Fetching Single Coin Details

This function gets detailed information about a specific cryptocurrency, such as historical price trends, supply, and market cap.

📌 fetchCoinDetail(id) - Get Coin Info

import axios from "axios";

export const fetchCoinDetail = async (coinId) => {

try {

const response = await axios.get(`https://api.coingecko.com/api/v3/coins/${coinId}`);

return response.data; // Return coin details

} catch (error) {

console.error(`Error fetching ${coinId} details:`, error);

return null; // Return null if an error occurs

}

};

✅ What It Does?

Takes a coin ID (e.g., bitcoin, ethereum).

Fetches detailed market data for that specific cryptocurrency.

Returns the coin details object or null if an error occurs.

---

3️⃣ Fetching Historical Price Data for Charts

This function retrieves historical price trends for the selected cryptocurrency over different time periods.

📌 fetchHistoricalData(id, days) - Get Price History

export const fetchHistoricalData = async (coinId, days = 30) => {

try {

const response = await axios.get(

`https://api.coingecko.com/api/v3/coins/${coinId}/market\_chart`,

{

params: {

vs\_currency: "usd", // Prices in USD

days: days, // Timeframe (e.g., 7 days, 30 days, 1 year)

interval: "daily", // Daily price points

},

}

);

return response.data.prices; // Return price history data

} catch (error) {

console.error(`Error fetching historical data for ${coinId}:`, error);

return [];

}

};

✅ What It Does?

Fetches historical price data for a coin over the last 7, 30, or 365 days.

Returns an array of date-wise price points for chart visualization.

---

4️⃣ Fetching Latest Crypto News

This function retrieves the latest cryptocurrency news from an external news API.

📌 fetchCryptoNews() - Get Latest News

export const fetchCryptoNews = async () => {

try {

const response = await axios.get("https://cryptonews-api-url.com/latest");

return response.data.articles; // Return news articles

} catch (error) {

console.error("Error fetching crypto news:", error);

return []; // Return empty array on error

}

};

✅ What It Does?

Fetches the latest crypto news from a news API.

Returns an array of news articles with headlines, descriptions, and links.

---

5️⃣ Using These Functions in a React Component

Here’s how you can use these functions inside a React component to fetch and display data.

📌 Example: Fetching and Displaying Cryptocurrency List

import { useEffect, useState } from "react";

import { fetchCryptoData } from "../API/API";

const CryptoList = () => {

const [cryptoList, setCryptoList] = useState([]);

useEffect(() => {

const getData = async () => {

const data = await fetchCryptoData();

setCryptoList(data);

};

getData();

}, []);

return (

<div>

<h1>Top Cryptocurrencies</h1>

{cryptoList.map((coin) => (

<div key={coin.id}>

<img src={coin.image} alt={coin.name} width="30" />

<h2>{coin.name} ({coin.symbol.toUpperCase()})</h2>

<p>Price: ${coin.current\_price.toFixed(2)}</p>

</div>

))}

</div>

);

};

export default CryptoList;

✅ How This Works?

Calls fetchCryptoData() once on component mount.

Stores fetched data in state (cryptoList).

Displays name, symbol, price, and image for each coin.

---

🔹 Summary of Data Fetching Functions in Cryptoverse

Breakdown of Cryptoverse Code – API Endpoints & Key Usage

Cryptoverse fetches real-time cryptocurrency data and news from external APIs. Below is a breakdown of the API endpoints used, their purpose, and how API keys are managed.

---

1️⃣ API Base URL

The project primarily uses the CoinGecko API for fetching cryptocurrency data.

const API\_URL = "https://api.coingecko.com/api/v3";

✅ Why CoinGecko?

Free to use (no API key required for public endpoints).

Provides real-time price updates, market trends, and historical data.

---

2️⃣ API Endpoints & Their Purpose

---

3️⃣ API Key Usage (If Required)

CoinGecko does not require an API key for basic usage.

If using an API that requires authentication, the API key should be stored securely in a .env file.

📌 Example: Using an API Key Securely

1️⃣ Create a .env file in the root directory

VITE\_API\_KEY=your\_api\_key\_here

2️⃣ Modify API Call to Include the API Key

const API\_KEY = import.meta.env.VITE\_API\_KEY;

const fetchSecureData = async () => {

const response = await axios.get("https://secure-api.com/data", {

headers: { Authorization: `Bearer ${API\_KEY}` },

});

return response.data;

};

✅ Why use .env?

Keeps the API key secure and hidden from the frontend.

Prevents leaking sensitive information in public repositories.

---

4️⃣ Example: Full API Call with Endpoint

export const fetchCryptoData = async () => {

try {

const response = await axios.get(`${API\_URL}/coins/markets`, {

params: { vs\_currency: "usd", order: "market\_cap\_desc", per\_page: 10, page: 1 },

});

return response.data;

} catch (error) {

console.error("Error fetching crypto data:", error);

return [];

}

};

✅ How This Works?

Calls https://api.coingecko.com/api/v3/coins/markets

Returns top 10 cryptocurrencies with their market prices.

🔹 Summary of API Management in Cryptoverse

Would you like help with adding more APIs, or do you need assistance with environment variables in deployment

Error Handling in Cryptoverse

Handling errors properly ensures that the Cryptoverse application remains stable, user-friendly, and secure even when API requests fail or unexpected issues occur. Below is a breakdown of error handling techniques used in the project.

---

1️⃣ Handling API Errors (Axios)

Since Cryptoverse fetches data from external APIs, network failures, invalid requests, or API downtime can cause errors.

✅ Solution: Try-Catch & Logging Errors

import axios from "axios";

export const fetchCryptoData = async () => {

try {

const response = await axios.get("https://api.coingecko.com/api/v3/coins/markets", {

params: { vs\_currency: "usd", order: "market\_cap\_desc", per\_page: 10, page: 1 },

});

return response.data;

} catch (error) {

console.error("Error fetching crypto data:", error.message);

return []; // Return an empty array to prevent crashes

}

};

🛠 How This Works?

✅ Wraps the API request in a try-catch block.

✅ Logs errors to the console for debugging.

✅ Returns an empty array ([]) instead of breaking the app.

---

2️⃣ Showing User-Friendly Error Messages

Instead of showing a blank screen, it's better to display an error message when data fetching fails.

📌 Example: Handling API Errors in a Component

import { useState, useEffect } from "react";

import { fetchCryptoData } from "../API/API";

const CryptoList = () => {

const [cryptoList, setCryptoList] = useState([]);

const [error, setError] = useState(null);

useEffect(() => {

const getData = async () => {

try {

const data = await fetchCryptoData();

if (data.length === 0) throw new Error("No data available.");

setCryptoList(data);

} catch (err) {

setError(err.message); // Store error message

}

};

getData();

}, []);

if (error) return <p className="text-red-500">Error: {error}</p>; // Show error message

if (cryptoList.length === 0) return <p>Loading...</p>; // Show loading state

return (

<div>

{cryptoList.map((coin) => (

<p key={coin.id}>{coin.name} - ${coin.current\_price.toFixed(2)}</p>

))}

</div>

);

};

export default CryptoList;

🛠 How This Works?

✅ Uses useState to store an error message.

✅ If fetching fails, displays an error message instead of a blank page.

✅ Prevents showing empty data by checking cryptoList.length.

---

3️⃣ Handling Slow API Responses (Timeout Handling)

Sometimes, APIs take too long to respond, causing the UI to freeze. We can set a timeout using Axios.

📌 Example: Setting an API Timeout (5 Seconds)

const fetchCryptoData = async () => {

try {

const response = await axios.get("https://api.coingecko.com/api/v3/coins/markets", {

params: { vs\_currency: "usd", order: "market\_cap\_desc", per\_page: 10, page: 1 },

timeout: 5000, // Set timeout to 5 seconds

});

return response.data;

} catch (error) {

if (error.code === "ECONNABORTED") {

console.error("API request timed out.");

} else {

console.error("Error fetching crypto data:", error.message);

}

return [];

}

};

🛠 How This Works?

✅ Prevents the app from freezing by limiting API response time to 5 seconds.

✅ If the API takes too long, it cancels the request and logs an error.

---

4️⃣ Handling Route Errors (Invalid URLs)

If a user tries to access a non-existent page, they should see a custom 404 page instead of a blank screen.

📌 Example: Creating a 404 Page

import { Link } from "react-router-dom";

const NotFound = () => {

return (

<div className="text-center p-6">

<h1 className="text-3xl font-bold text-red-600">404 - Page Not Found</h1>

<p>The page you are looking for does not exist.</p>

<Link to="/" className="text-blue-500">Go Back to Home</Link>

</div>

);

};

export default NotFound;

📌 Example: Adding 404 Page to Routes

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Home from "./pages/MainPage";

import Cryptocurrency from "./pages/Cryptocurrency";

import NotFound from "./pages/NotFound"; // Import 404 Page

const App = () => {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/cryptocurrency" element={<Cryptocurrency />} />

<Route path="\*" element={<NotFound />} /> {/\* Catch-all for invalid routes \*/}

</Routes>

</Router>

);

};

export default App;

🛠 How This Works?

✅ If a user enters an invalid URL, they are redirected to a custom 404 page.

✅ Prevents blank pages and improves user experience.

---

5️⃣ Preventing Crashes from Undefined Data

Sometimes, the API response might be missing data, leading to errors like undefined values.

📌 Example: Handling Undefined Data Gracefully

const CryptoCard = ({ coin }) => {

if (!coin || !coin.name || !coin.current\_price) {

return <p className="text-red-500">Invalid data.</p>; // Display error message

}

return (

<div className="p-4 border rounded-lg shadow-md">

<h2 className="text-lg font-bold">{coin.name}</h2>

<p>Price: ${coin.current\_price.toFixed(2)}</p>

</div>

);

};

🛠 How This Works?

✅ Checks if coin exists before displaying data.

✅ Prevents "undefined" errors from breaking the UI.

---

🔹 Summary of Error Handling in Cryptoverse

Fetching Related Videos in Cryptoverse

If the Cryptoverse project includes a feature for fetching related cryptocurrency videos, it likely integrates with YouTube API or another video platform API. Below is a breakdown of how to fetch related videos based on a cryptocurrency name or topic.

---

1️⃣ Setting Up the YouTube API

To fetch related videos, you'll need to use the YouTube Data API v3.

📌 Steps to Get an API Key:

1. Go to the Google Cloud Console.

2. Create a new project and enable YouTube Data API v3.

3. Go to Credentials and generate an API Key.

4. Store the key in a .env file for security.

---

2️⃣ API Endpoint for Fetching Videos

YouTube API provides an endpoint to search for videos based on a keyword.

Endpoint:

https://www.googleapis.com/youtube/v3/search

Required Parameters:

| Parameter | Purpose | |-----------|---------| | part=snippet | Includes title, description, thumbnails, etc. | | q=bitcoin | Search query (e.g., cryptocurrency name) | | maxResults=5 | Limits the number of videos | | type=video | Ensures only videos are returned | | key=YOUR\_API\_KEY | Your API key |

---

3️⃣ Fetching Related Videos Function

✅ fetchRelatedVideos() - Get Related Videos from YouTube

import axios from "axios";

const YOUTUBE\_API\_KEY = import.meta.env.VITE\_YOUTUBE\_API\_KEY;

const YOUTUBE\_API\_URL = "https://www.googleapis.com/youtube/v3/search";

export const fetchRelatedVideos = async (query) => {

try {

const response = await axios.get(YOUTUBE\_API\_URL, {

params: {

part: "snippet",

q: query, // Search for related videos (e.g., "Bitcoin news")

maxResults: 5, // Fetch 5 related videos

type: "video",

key: YOUTUBE\_API\_KEY, // Use secure API key

},

});

return response.data.items; // Return video list

} catch (error) {

console.error("Error fetching related videos:", error);

return []; // Return empty array on error

}

};

✅ What It Does?

Calls YouTube API to fetch 5 related videos for a given cryptocurrency.

Uses an API key stored in .env for security.

Returns an array of video details (title, thumbnail, link, etc.).

---

4️⃣ Displaying Related Videos in a React Component

Once we fetch the videos, we need to display them in the UI.

📌 Example: RelatedVideos.tsx - Display Videos

import { useState, useEffect } from "react";

import { fetchRelatedVideos } from "../API/YoutubeAPI";

const RelatedVideos = ({ searchQuery }) => {

const [videos, setVideos] = useState([]);

useEffect(() => {

const getVideos = async () => {

const videoList = await fetchRelatedVideos(searchQuery);

setVideos(videoList);

};

getVideos();

}, [searchQuery]); // Re-fetch when query changes

if (videos.length === 0) return <p>Loading related videos...</p>;

return (

<div>

<h2 className="text-xl font-bold">Related Videos</h2>

<div className="grid grid-cols-2 gap-4">

{videos.map((video) => (

<a

key={video.id.videoId}

href={`https://www.youtube.com/watch?v=${video.id.videoId}`}

target="\_blank"

rel="noopener noreferrer"

className="block border p-2 rounded-md shadow-md hover:bg-gray-200"

>

<img src={video.snippet.thumbnails.medium.url} alt={video.snippet.title} />

<p className="text-sm font-semibold">{video.snippet.title}</p>

</a>

))}

</div>

</div>

);

};

export default RelatedVideos;

✅ How This Works?

Calls fetchRelatedVideos(searchQuery) to get videos.

Displays video thumbnails, titles, and clickable links.

Updates whenever the searchQuery (e.g., "Bitcoin") changes.

---

5️⃣ Adding Videos to the Coin Detail Page

To show related videos for a specific cryptocurrency, we can integrate RelatedVideos.tsx into CoinDetail.tsx.

📌 Example: CoinDetail.tsx with Videos

import { useParams } from "react-router-dom";

import RelatedVideos from "../components/RelatedVideos";

const CoinDetail = () => {

const { id } = useParams(); // Get coin ID (e.g., "bitcoin")

return (

<div className="p-6">

<h1 className="text-2xl font-bold">{id.toUpperCase()} Details</h1>

<RelatedVideos searchQuery={id} /> {/\* Show videos for this coin \*/}

</div>

);

};

export default CoinDetail;

✅ What This Does?

Extracts id from the URL (/coin/bitcoin).

Passes id (e.g., "bitcoin") to RelatedVideos.tsx to fetch relevant videos.

---

6️⃣ Storing the API Key Securely

To avoid exposing the API key, store it in a .env file:

📌 .env File:

VITE\_YOUTUBE\_API\_KEY=your\_api\_key\_here

✅ Why Use .env?

Prevents exposing API keys in public repositories.

Environment variables allow easy configuration.

---

🔹 Summary of Video Fetching in Cryptoverse

API Configuration in Cryptoverse

To fetch cryptocurrency data, news, and related videos, the Cryptoverse project requires proper API configuration. Below is a breakdown of how APIs are set up and secured.

---

1️⃣ Setting Up API Configuration (API.tsx)

📌 Create a Central API Configuration File

Instead of defining API URLs in multiple places, we centralize them in a single file.

✅ API.tsx – Centralized API Configuration

import axios from "axios";

// API Base URLs

const CRYPTO\_API\_URL = "https://api.coingecko.com/api/v3";

const NEWS\_API\_URL = "https://cryptonews-api-url.com/latest";

const YOUTUBE\_API\_URL = "https://www.googleapis.com/youtube/v3/search";

// Secure API Keys from `.env`

const YOUTUBE\_API\_KEY = import.meta.env.VITE\_YOUTUBE\_API\_KEY;

const NEWS\_API\_KEY = import.meta.env.VITE\_NEWS\_API\_KEY; // If needed

// Axios instance with global settings

const apiClient = axios.create({

timeout: 5000, // Set a timeout for requests

headers: { "Content-Type": "application/json" },

});

// Function to fetch cryptocurrency market data

export const fetchCryptoData = async () => {

try {

const response = await apiClient.get(`${CRYPTO\_API\_URL}/coins/markets`, {

params: { vs\_currency: "usd", order: "market\_cap\_desc", per\_page: 10, page: 1 },

});

return response.data;

} catch (error) {

console.error("Error fetching crypto data:", error.message);

return [];

}

};

// Function to fetch cryptocurrency news

export const fetchCryptoNews = async () => {

try {

const response = await apiClient.get(NEWS\_API\_URL, {

headers: { Authorization: `Bearer ${NEWS\_API\_KEY}` }, // If API key required

});

return response.data.articles;

} catch (error) {

console.error("Error fetching crypto news:", error.message);

return [];

}

};

// Function to fetch related videos from YouTube

export const fetchRelatedVideos = async (query) => {

try {

const response = await apiClient.get(YOUTUBE\_API\_URL, {

params: {

part: "snippet",

q: query, // Search for related videos (e.g., "Bitcoin news")

maxResults: 5,

type: "video",

key: YOUTUBE\_API\_KEY,

},

});

return response.data.items;

} catch (error) {

console.error("Error fetching related videos:", error.message);

return [];

}

};

✅ What This Does?

Centralizes all API requests in one file (API.tsx).

Uses Axios with a timeout to handle slow API responses.

Stores API keys in environment variables (.env) for security.

---

2️⃣ Storing API Keys Securely in .env

To prevent exposing sensitive API keys, store them in a .env file.

📌 .env File:

VITE\_YOUTUBE\_API\_KEY=your\_youtube\_api\_key\_here

VITE\_NEWS\_API\_KEY=your\_news\_api\_key\_here

✅ Why Use .env?

Protects API keys from being exposed in the code.

Keeps credentials secure and easy to change.

---

3️⃣ Using API Functions in React Components

Now that APIs are configured, we can call them in React components.

📌 Example: Fetching & Displaying Cryptocurrency Data

import { useEffect, useState } from "react";

import { fetchCryptoData } from "../API/API";

const CryptoList = () => {

const [cryptoList, setCryptoList] = useState([]);

useEffect(() => {

const getData = async () => {

const data = await fetchCryptoData();

setCryptoList(data);

};

getData();

}, []);

return (

<div>

<h1>Top Cryptocurrencies</h1>

{cryptoList.map((coin) => (

<p key={coin.id}>{coin.name} - ${coin.current\_price.toFixed(2)}</p>

))}

</div>

);

};

export default CryptoList;

✅ How This Works?

Calls fetchCryptoData() when the component loads.

Stores fetched crypto market data in state.

Displays name and price of each cryptocurrency.

---

4️⃣ Handling API Errors

Since API requests can fail, we add error handling.

📌 Example: Handling API Errors with a Fallback Message

const [error, setError] = useState(null);

useEffect(() => {

const getData = async () => {

try {

const data = await fetchCryptoData();

if (data.length === 0) throw new Error("No data available.");

setCryptoList(data);

} catch (err) {

setError(err.message); // Store error message

}

};

getData();

}, []);

if (error) return <p className="text-red-500">Error: {error}</p>;

✅ Why This is Important?

Prevents the app from crashing if the API request fails.

Displays a user-friendly error message.

---

5️⃣ Optimizing API Calls

To reduce unnecessary API requests, we can:

1. Cache API responses in state or local storage.

2. Debounce search requests to avoid too many calls.

📌 Example: Debouncing API Calls for Search

import { useState, useEffect } from "react";

import { fetchRelatedVideos } from "../API/API";

const SearchVideos = ({ query }) => {

const [videos, setVideos] = useState([]);

const [debouncedQuery, setDebouncedQuery] = useState(query);

useEffect(() => {

const timer = setTimeout(() => {

setDebouncedQuery(query);

}, 500); // Wait 500ms before updating query

return () => clearTimeout(timer);

}, [query]);

useEffect(() => {

if (debouncedQuery) {

fetchRelatedVideos(debouncedQuery).then(setVideos);

}

}, [debouncedQuery]);

return (

<div>

<h2>Videos for {debouncedQuery}</h2>

{videos.map((video) => (

<p key={video.id.videoId}>{video.snippet.title}</p>

))}

</div>

);

};

✅ Why This Works?

Delays API requests when the user is typing (prevents too many calls).

Only fetches videos after 500ms if the query remains the same.

---

🔹 Summary of API Configuration in Cryptoverse

Project Execution – Running Cryptoverse

To successfully execute and run the Cryptoverse project, follow these steps:

---

1️⃣ Prerequisites

Before executing the project, ensure your system meets the following requirements:

✅ Operating System: Windows, macOS, or Linux

✅ Node.js: Version 16+ (Check with node -v)

✅ Package Manager: npm or yarn (Check with npm -v)

✅ Code Editor: VS Code (Recommended)

---

2️⃣ Step-by-Step Execution

📌 Step 1: Clone the Project

If you haven’t already downloaded the project, clone it from GitHub:

git clone https://github.com/your-repo/Cryptoverse.git

cd Cryptoverse

---

📌 Step 2: Install Dependencies

Install all required packages using npm or yarn:

npm install

or

yarn install

✅ This installs all required dependencies from package.json.

---

📌 Step 3: Set Up API Keys (If Required)

If the project uses API keys for YouTube, CoinGecko, or News API, create a .env file:

Create .env File in the Project Root

VITE\_YOUTUBE\_API\_KEY=your\_youtube\_api\_key

VITE\_NEWS\_API\_KEY=your\_news\_api\_key

✅ Why? Keeps API keys secure and prevents exposing sensitive information.

---

📌 Step 4: Start the Development Server

Run the following command to start the project locally:

npm run dev

or

yarn dev

✅ This launches the project at http://localhost:3000/.

---

📌 Step 5: Running the Project in a Web Browser

Once the server is running, open a browser and visit:

http://localhost:3000/

✅ You should see the Cryptoverse homepage with cryptocurrency data and news.

---

📌 Step 6: Building for Production

If you want to prepare the project for deployment, run:

npm run build

✅ This generates an optimized version in the /dist folder.

---

📌 Step 7: Deploying Cryptoverse (Optional)

You can deploy Cryptoverse on platforms like Vercel or Netlify.

Deploy on Vercel

npm install -g vercel

vercel

Deploy on Netlify

npm install -g netlify-cli

netlify deploy --prod

✅ This generates a live URL where your project is accessible online.

---

🔹 Summary of Project Execution

Hero Component in Cryptoverse

The Hero Component is typically the main banner of the website, providing an engaging introduction to the platform. It usually includes:

✅ A headline and a short description.

✅ A call-to-action (CTA) button.

✅ A background image or animation.

✅ A responsive design for mobile users.

---

1️⃣ Hero Component Code (Hero.tsx)

const Hero = () => {

return (

<div className="relative w-full h-[400px] flex flex-col items-center justify-center text-center bg-gradient-to-r from-blue-500 to-purple-600 text-white p-6">

<h1 className="text-4xl md:text-6xl font-bold">Welcome to Cryptoverse</h1>

<p className="text-lg md:text-xl mt-3">Track live cryptocurrency prices and market trends in real-time.</p>

<a

href="/cryptocurrency"

className="mt-5 px-6 py-3 bg-white text-blue-600 font-semibold rounded-lg shadow-md hover:bg-gray-200 transition"

>

Explore Market

</a>

</div>

);

};

export default Hero;

✅ How It Works?

Hero section with a gradient background.

Large heading and description to introduce the platform.

"Explore Market" button links to the cryptocurrency market page.

---

2️⃣ Using the Hero Component in MainPage.tsx

To display the Hero Component, import it inside the homepage file:

import Hero from "../components/Hero";

const MainPage = () => {

return (

<div>

<Hero />

{/\* Other homepage sections go here \*/}

</div>

);

};

export default MainPage;

---

3️⃣ Adding an Image to the Hero Component

If you want a background image instead of a gradient:

📌 Updated Hero.tsx with Background Image

const Hero = () => {

return (

<div

className="relative w-full h-[400px] flex flex-col items-center justify-center text-center text-white p-6"

style={{ backgroundImage: "url('/assets/crypto.jpg')", backgroundSize: "cover", backgroundPosition: "center" }}

>

<div className="absolute inset-0 bg-black bg-opacity-50"></div> {/\* Overlay for better readability \*/}

<h1 className="text-4xl md:text-6xl font-bold relative">Welcome to Cryptoverse</h1>

<p className="text-lg md:text-xl mt-3 relative">Track live cryptocurrency prices and market trends in real-time.</p>

<a

href="/cryptocurrency"

className="mt-5 px-6 py-3 bg-blue-600 font-semibold rounded-lg shadow-md hover:bg-blue-700 transition relative"

>

Explore Market

</a>

</div>

);

};

export default Hero;

✅ New Features Added:

Background image (crypto.jpg) instead of a gradient.

Dark overlay (bg-opacity-50) for better text readability.

---

4️⃣ Making the Hero Section Responsive

To ensure it looks good on all devices, use Tailwind CSS breakpoints.

📌 Responsive Updates

<h1 className="text-3xl sm:text-5xl md:text-6xl font-bold relative">

✅ Now scales text size:

text-3xl on small screens

text-5xl on tablets

text-6xl on desktops

---

🔹 Summary of the Hero Component