Hướng Dẫn Tạo Dự Án Crypto Tracker - Từng Bước Chi Tiết

Mục Lục

Giới Thiệu Dự Án

Crypto Tracker là một ứng dụng web theo dõi giá tiền điện tử theo thời gian thực, được xây dựng bằng:

- **React 19** - Thư viện UI hiện đại
- **TypeScript 5.9** - JavaScript với kiểu dữ liệu
- **Vite 7** - Công cụ build nhanh
- **React Router DOM 7** - Điều hướng trang
- **Recharts 3** - Vẽ biểu đồ
- **CoinGecko API** - Dữ liệu tiền điện tử

Cài Đặt Môi Trường

Bước 1: Cài Đặt Node.js

1. Truy cập nodejs.org
2. Tải và cài đặt phiên bản **LTS** (18.x trở lên)
3. Kiểm tra cài đặt:

```
node --version
npm --version
```

Bước 2: Cài Đặt Code Editor

Khuyến nghị sử dụng:

- **VS Code** hoặc **WebStorm**
- Cài extension: ESLint, Prettier, TypeScript

Tạo Dự Án Từ Đầu

Bước 1: Tạo Dự Án Vite + React + TypeScript

```
# Tạo dự án mới
npm create vite@latest crypto -- --template react-ts

# Di chuyển vào thư mục
cd crypto

# Cài đặt dependencies
npm install
```

Bước 2: Cài Đặt Các Thư Viện Cần Thiết

```
# React Router để điều hướng
npm install react-router-dom

# Recharts để vẽ biểu đồ
npm install recharts

# CoinGecko API client (tùy chọn, có thể dùng fetch)
npm install @coingecko/coingecko-typescript
```

Bước 3: Cài Đặt Type Definitions

```
npm install --save-dev @types/node
```

Bước 4: Chạy Dự Án

```
# Chạy development server
npm run dev

# Mở trình duyệt tại http://localhost:5173
```

Cấu Trúc Dự Án

```
crypto/
├── public/                    # File tĩnh (logo, favicon)
│   └── vite.svg
├── src/
│   ├── components/           # Các component tái sử dụng
│   │   ├── ChartSection.tsx
│   │   ├── Controls.tsx
```

```
│   │   ├── CryptoCard.tsx
│   │   ├── CryptoList.tsx
│   │   ├── Footer.tsx
│   │   ├── Header.tsx
│   │   ├── Loading.tsx
│   │   ├── NewsCard.tsx
│   │   ├── NewsList.tsx
│   │   ├── NoResults.tsx
│   │   ├── PriceSection.tsx
│   │   └── StatsGrid.tsx
│   ├── layout/               # Layout components
│   │   └── MainLayout.tsx
│   ├── pages/                # Các trang chính
│   │   ├── CoinDetail.tsx
│   │   ├── CryptoNews.tsx
│   │   └── Home.tsx
│   ├── services/             # API services
│   │   ├── coinGecko.ts
│   │   └── newsApi.ts
│   ├── utils/                # Utility functions
│   │   └── formatter.ts
│   ├── styles/               # CSS files
│   │   └── news.css
│   ├── App.tsx               # Component chính
│   ├── main.tsx              # Entry point
│   └── index.css             # Global styles
├── index.html                # HTML template
├── package.json              # Dependencies
├── tsconfig.json             # TypeScript config
├── vite.config.ts            # Vite config
└── README.md
```

Giải Thích Chi Tiết Từng File

1. `package.json` - Quản Lý Dependencies

```json
{
  "name": "crypto",
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",              // Chạy dev server
    "build": "tsc -b && vite build",  // Build production
    "lint": "eslint .",         // Kiểm tra lỗi code
    "preview": "vite preview"   // Xem preview build
  },
  "dependencies": {
    // Các thư viện cần thiết khi chạy ứng dụng
    "react": "^19.2.0",
    "react-dom": "^19.2.0",
```

```
    "react-router-dom": "^7.10.1",
    "recharts": "^3.5.1"
  },
  "devDependencies": {
    // Các thư viện chỉ cần khi phát triển
    "typescript": "~5.9.3",
    "vite": "^7.2.4",
    "@vitejs/plugin-react": "^5.1.1"
  }
}
```

Giải thích:

- `dependencies`: Các package cần thiết khi ứng dụng chạy
- `devDependencies`: Các package chỉ dùng khi phát triển (TypeScript, Vite, ESLint)
- `scripts`: Các lệnh npm có thể chạy

---

2. `vite.config.ts` - Cấu Hình Vite

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
})
```

Giải thích từng phần:

Phần 1: Import

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
```

- `defineConfig`: Hàm helper để định nghĩa cấu hình Vite với TypeScript autocomplete
- `react`: Plugin để Vite hiểu và xử lý file React/JSX

Phần 2: Export Config

```
export default defineConfig({
  plugins: [react()],
})
```

- `plugins`: Mảng các plugin Vite sử dụng
- `react()`: Plugin cho phép Vite compile JSX và hỗ trợ Fast Refresh

Vite là gì?

- Build tool nhanh hơn Webpack
- HMR (Hot Module Replacement) - tự động reload khi code thay đổi
- ES modules native - không cần bundle trong development

---

3. `tsconfig.json` - Cấu Hình TypeScript

```
{
  "files": [],
  "references": [
    { "path": "./tsconfig.app.json" },
    { "path": "./tsconfig.node.json" }
  ]
}
```

Giải thích:

- `files: []`: Không include file trực tiếp
- `references`: Sử dụng Project References để chia config thành nhiều file
  - `tsconfig.app.json`: Config cho code ứng dụng
  - `tsconfig.node.json`: Config cho build scripts

Lợi ích:

- Tách biệt config cho app và build tools
- Compile nhanh hơn
- Quản lý dễ hơn

---

4. `index.html` - HTML Template

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Crypto Tracker</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Giải thích từng phần:

Phần 1: Meta Tags

```
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

- `charset`: Định dạng ký tự UTF-8
- `viewport`: Responsive design cho mobile

Phần 2: Root Element

```
<div id="root"></div>
```

- Đây là nơi React sẽ render toàn bộ ứng dụng

Phần 3: Entry Script

```
<script type="module" src="/src/main.tsx"></script>
```

- `type="module"`: Sử dụng ES modules
- `src="/src/main.tsx"`: File bắt đầu của ứng dụng

---

5. `src/main.tsx` - Entry Point

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

Giải thích từng phần:

Phần 1: Import Dependencies

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'
```

- `StrictMode`: Component React giúp phát hiện lỗi tiềm ẩn
- `createRoot`: API mới của React 18+ để tạo root
- `'./index.css'`: Import global styles
- `App`: Component chính của ứng dụng

Phần 2: Render Application

```
createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

- `document.getElementById('root')!`: Lấy element root từ HTML (dấu `!` là TypeScript assertion)
- `createRoot()`: Tạo React root mới
- `.render()`: Render component vào DOM
- `StrictMode`: Bọc App để kiểm tra lỗi

Luồng hoạt động:

1. Tìm element `#root` trong HTML
2. Tạo React root
3. Render component `App` vào root
4. React bắt đầu quá trình render

---

6. `src/App.tsx` - Component Chính

```
import { BrowserRouter, Route, Routes } from 'react-router-dom'
import './App.css'
import {Home} from "./pages/Home.tsx";
import {CoinDetail} from "./pages/CoinDetail.tsx";
import {CryptoNews} from "./pages/CryptoNews.tsx";

function App() {
  return (
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/coin/:id" element={<CoinDetail />} />
          <Route path="/news" element={<CryptoNews />} />
        </Routes>
      </BrowserRouter>
  )
}

export default App
```

Giải thích từng phần:

Phần 1: Import

```
import { BrowserRouter, Route, Routes } from 'react-router-dom'
```

- `BrowserRouter`: Component cung cấp routing cho ứng dụng (sử dụng HTML5 History API)
- `Route`: Component định nghĩa một route
- `Routes`: Component wrapper cho các Route

Phần 2: Import Pages

```
import {Home} from "./pages/Home.tsx";
import {CoinDetail} from "./pages/CoinDetail.tsx";
import {CryptoNews} from "./pages/CryptoNews.tsx";
```

- Import các component trang

Phần 3: Routing Setup

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/coin/:id" element={<CoinDetail />} />
    <Route path="/news" element={<CryptoNews />} />
  </Routes>
</BrowserRouter>
```

- `path="/"`: Route cho trang chủ
- `path="/coin/:id"`: Route động, `:id` là parameter (ví dụ: `/coin/bitcoin`)
- `element={<Home />}`: Component sẽ render khi match route

Cách hoạt động:

1. User truy cập URL
2. React Router kiểm tra route nào match
3. Render component tương ứng
4. Nếu có `:id`, lấy giá trị từ URL

---

7. `src/services/coinGecko.ts` - API Service

```
const BASE_URL = "https://api.coingecko.com/api/v3";

export interface Crypto {
```

```
  id: string;
  symbol: string;
  name: string;
  image: string;
  current_price: number;
  market_cap: number;
  market_cap_rank: number;
  price_change_percentage_24h: number;
  total_volume: number;
}
```

Giải thích phần 1: Interface Definitions

```
export interface Crypto {
  id: string;                              // ID duy nhất (ví dụ: "bitcoin")
  symbol: string;                          // Ký hiệu (ví dụ: "btc")
  name: string;                            // Tên đầy đủ (ví dụ: "Bitcoin")
  image: string;                           // URL hình ảnh
  current_price: number;                   // Giá hiện tại (USD)
  market_cap: number;                      // Vốn hóa thị trường
  market_cap_rank: number;                 // Xếp hạng theo vốn hóa
  price_change_percentage_24h: number;     // % thay đổi 24h
  total_volume: number;                    // Khối lượng giao dịch 24h
}
```

- `interface`: Định nghĩa cấu trúc dữ liệu TypeScript
- Giúp TypeScript kiểm tra type và autocomplete

```
export interface CoinData {
  id: string;
  symbol: string;
  name: string;
  description: {
    en: string;                            // Mô tả bằng tiếng Anh
  };
  market_data: {
    current_price: { usd: number; };       // Giá hiện tại
    market_cap: { usd: number; };          // Vốn hóa
    price_change_percentage_24h: number;   // % thay đổi
    market_cap_rank: number;               // Xếp hạng
    high_24h: { usd: number; };            // Giá cao nhất 24h
    low_24h: { usd: number; };             // Giá thấp nhất 24h
    total_volume: { usd: number; };        // Khối lượng
    circulating_supply: number;            // Lượng lưu hành
    total_supply: number;                  // Tổng cung
  };
  image: {
    thumb: string;                         // Ảnh nhỏ
    small: string;                         // Ảnh vừa
```

```
    large: string;                          // Ảnh lớn
  };
}
```

```
export interface ChartData {
  prices: [number, number][];               // Mảng [timestamp, price]
}
```

Phần 2: Fetch Functions

```
export const fetchCryptos = async (): Promise<Crypto[]> => {
  const response = await fetch(
      `${BASE_URL}/coins/markets?
vs_currency=usd&order=market_cap_desc&per_page=100&page=1&sparkline=false`
  );

  if (!response.ok) {
    throw new Error("Failed to fetch cryptos");
  }

  return response.json();
};
```

Giải thích từng dòng:

1. **Function Declaration**

```
export const fetchCryptos = async (): Promise<Crypto[]> => {
```

- `export`: Export để dùng ở file khác
- `async`: Hàm bất đồng bộ (trả về Promise)
- `Promise<Crypto[]>`: Trả về Promise với mảng Crypto

2. **API Call**

```
const response = await fetch(
    `${BASE_URL}/coins/markets?
vs_currency=usd&order=market_cap_desc&per_page=100&page=1&sparkline=false`
);
```

- `fetch()`: API browser để gọi HTTP request
- `await`: Đợi response trả về
- Query params:

- `vs_currency=usd`: Đơn vị tiền tệ
- `order=market_cap_desc`: Sắp xếp theo vốn hóa giảm dần
- `per_page=100`: Lấy 100 coin
- `page=1`: Trang đầu tiên
- `sparkline=false`: Không lấy dữ liệu biểu đồ nhỏ

3. **Error Handling**

```
if (!response.ok) {
  throw new Error("Failed to fetch cryptos");
}
```

- Kiểm tra response có thành công không
- Nếu lỗi, throw error

4. **Return Data**

```
return response.json();
```

- Parse JSON response thành object JavaScript

```
export const fetchCoinData = async (id: string): Promise<CoinData> => {
  const response = await fetch(
      `${BASE_URL}/coins/${id}?
localization=false&tickers=false&market_data=true&community_data=false&developer_d
ata=false&sparkline=false`
  );
  if (!response.ok) {
    throw new Error("Failed to fetch coin data");
  }
  return response.json();
};
```

Giải thích:

- Nhận `id` của coin (ví dụ: "bitcoin")
- Gọi API để lấy thông tin chi tiết
- Query params để tối ưu response (chỉ lấy data cần thiết)

```
export const fetchChartData = async (id: string): Promise<ChartData> => {
  const response = await fetch(
      `${BASE_URL}/coins/${id}/market_chart?vs_currency=usd&days=7`
  );
  if (!response.ok) {
    throw new Error("Failed to fetch chart data");
```

```
  }
  return response.json();
};
```

Giải thích:

- Lấy dữ liệu biểu đồ 7 ngày
- Trả về mảng `[timestamp, price]`

---

8. `src/utils/formatter.ts` - Utility Functions

```
export const formatPrice = (price: number): string => {
  if (price < 0.01) return price.toFixed(8);

  return new Intl.NumberFormat("en-US", {
    style: "currency",
    currency: "USD",
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }).format(price);
};
```

Giải thích từng phần:

Phần 1: Kiểm Tra Giá Nhỏ

```
if (price < 0.01) return price.toFixed(8);
```

- Nếu giá < $0.01 (ví dụ: Shiba Inu), hiển thị 8 chữ số thập phân
- `toFixed(8)`: Làm tròn đến 8 chữ số

Phần 2: Format Currency

```
return new Intl.NumberFormat("en-US", {
  style: "currency",
  currency: "USD",
  minimumFractionDigits: 2,
  maximumFractionDigits: 2,
}).format(price);
```

- `Intl.NumberFormat`: API browser để format số theo locale
- `"en-US"`: Locale (Mỹ)
- `style: "currency"`: Format thành tiền tệ
- `currency: "USD"`: Đơn vị USD

- `minimumFractionDigits: 2`: Tối thiểu 2 chữ số thập phân
- `maximumFractionDigits: 2`: Tối đa 2 chữ số thập phân

Ví dụ:

- `formatPrice(50000)` → `"$50,000.00"`
- `formatPrice(0.0001)` → `"0.00010000"`

```
export const formatMarketCap = (marketCap: number): string => {
  if (marketCap >= 1e12) return `${(marketCap / 1e12).toFixed(2)}T`;
  if (marketCap >= 1e9) return `${(marketCap / 1e9).toFixed(2)}B`;
  if (marketCap >= 1e6) return `${(marketCap / 1e6).toFixed(2)}M`;
  return marketCap.toLocaleString();
};
```

Giải thích:

- `1e12` = 1,000,000,000,000 (1 nghìn tỷ)
- `1e9` = 1,000,000,000 (1 tỷ)
- `1e6` = 1,000,000 (1 triệu)
- Format số lớn thành dạng ngắn gọn

Ví dụ:

- `formatMarketCap(1500000000000)` → `"1.50T"`
- `formatMarketCap(500000000)` → `"500.00M"`

---

9. `src/pages/Home.tsx` - Trang Chủ

```
import { useEffect, useState } from "react";
import { fetchCryptos } from "../services/coinGecko";
import type {Crypto} from "../services/coinGecko";
import { Controls } from "../components/Controls";
import { CryptoList } from "../components/CryptoList";
import { Loading } from "../components/Loading";
import {Layout} from "../layout/MainLayout.tsx";
```

Giải thích phần Import:

- `useEffect, useState`: React Hooks
- `fetchCryptos`: Function lấy dữ liệu từ API
- `Crypto`: Type definition
- Các component con

```
export const Home: React.FC = () => {
  const [cryptoList, setCryptoList] = useState<Crypto[]>([]);
```

```
  const [filteredList, setFilteredList] = useState<Crypto[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [viewMode, setViewMode] = useState<"grid" | "list">("grid");
  const [sortBy, setSortBy] = useState<
      "market_cap_rank" | "name" | "price" | "price_desc" | "change" |
"market_cap"
  >("market_cap_rank");
  const [searchQuery, setSearchQuery] = useState("");
```

Giải thích phần State:

1. **cryptoList**: Danh sách đầy đủ từ API
2. **filteredList**: Danh sách sau khi filter và sort
3. **isLoading**: Trạng thái loading
4. **viewMode**: Chế độ hiển thị (grid/list)
5. **sortBy**: Tiêu chí sắp xếp
6. **searchQuery**: Từ khóa tìm kiếm

```
useEffect(() => {
  const interval = setInterval(fetchCryptoData, 2000);
  return () => clearInterval(interval);
}, []);
```

Giải thích:

- `useEffect`: Chạy khi component mount
- `setInterval`: Gọi `fetchCryptoData` mỗi 2 giây
- `return () => clearInterval(interval)`: Cleanup khi unmount

```
useEffect(() => {
  filterAndSort();
}, [sortBy, cryptoList, searchQuery]);
```

Giải thích:

- Chạy lại `filterAndSort` khi `sortBy`, `cryptoList`, hoặc `searchQuery` thay đổi

```
const fetchCryptoData = async () => {
  try {
    const data: Crypto[] = await fetchCryptos();
    setCryptoList(data);
  } catch (err) {
    console.error(err);
  } finally {
    setIsLoading(false);
```

```
  }
};
```

Giải thích:

- `async/await`: Xử lý bất đồng bộ
- `try/catch`: Bắt lỗi
- `finally`: Luôn chạy (tắt loading)

```
const filterAndSort = () => {
  const filtered = cryptoList.filter(
      (crypto) =>
          crypto.name.toLowerCase().includes(searchQuery.toLowerCase()) ||
          crypto.symbol.toLowerCase().includes(searchQuery.toLowerCase())
  );

  filtered.sort((a, b) => {
    switch (sortBy) {
      case "name":
        return a.name.localeCompare(b.name);
      case "price":
        return a.current_price - b.current_price;
      case "price_desc":
        return b.current_price - a.current_price;
      case "change":
        return a.price_change_percentage_24h - b.price_change_percentage_24h;
      case "market_cap":
        return a.market_cap - b.market_cap;
      default:
        return a.market_cap_rank - b.market_cap_rank;
    }
  });

  setFilteredList(filtered);
};
```

Giải thích từng phần:

1. **Filter**

```
const filtered = cryptoList.filter(
    (crypto) =>
        crypto.name.toLowerCase().includes(searchQuery.toLowerCase()) ||
        crypto.symbol.toLowerCase().includes(searchQuery.toLowerCase())
);
```

- Lọc theo tên hoặc symbol
- `toLowerCase()`: Không phân biệt hoa thường

2. **Sort**

```
filtered.sort((a, b) => {
  switch (sortBy) {
    case "name":
      return a.name.localeCompare(b.name);  // So sánh chuỗi
    case "price":
      return a.current_price - b.current_price;  // Tăng dần
    case "price_desc":
      return b.current_price - a.current_price;  // Giảm dần
    // ...
  }
});
```

- `sort()`: Sắp xếp mảng
- Return < 0: a đứng trước b
- Return > 0: b đứng trước a

```
return (
    <Layout searchQuery={searchQuery} setSearchQuery={setSearchQuery}>
      <Controls
          sortBy={sortBy}
          setSortBy={setSortBy}
          viewMode={viewMode}
          setViewMode={setViewMode}
      />
      {isLoading ? (
          <Loading />
      ) : (
          <CryptoList cryptos={filteredList} viewMode={viewMode} />
      )}
    </Layout>
);
```

Giải thích JSX:

- `Layout`: Component wrapper
- `Controls`: Component điều khiển
- Conditional rendering: Hiển thị Loading hoặc CryptoList

---

10. `src/components/CryptoCard.tsx` - Card Hiển Thị Coin

```
import {Link} from "react-router-dom";
import {formatPrice, formatMarketCap} from "../utils/formatter";
import type {Crypto} from "../services/coinGecko";

interface CryptoCardProps {
```

```
  crypto: Crypto & {
    total_volume: number;
  };
}
```

Giải thích:

- `Link`: Component React Router để điều hướng
- `formatPrice, formatMarketCap`: Utility functions
- `CryptoCardProps`: Props interface

```
export const CryptoCard: React.FC<CryptoCardProps> = ({crypto}) => {
  return (
      <Link to={`/coin/${crypto.id}`} style={{textDecoration: "none"}}>
        <div className="crypto-card">
          {/* ... */}
        </div>
      </Link>
  );
};
```

Giải thích:

- `Link to={/coin/${crypto.id}}`: Link đến trang chi tiết
- `textDecoration: "none"`: Bỏ gạch chân

```
<div className="crypto-header">
  <div className="crypto-info">
    <img src={crypto.image} alt={crypto.name}/>
    <div>
      <h3>{crypto.name}</h3>
      <p className="symbol">{crypto.symbol.toUpperCase()}</p>
      <span className="rank">#{crypto.market_cap_rank}</span>
    </div>
  </div>
</div>
```

Giải thích:

- Hiển thị hình ảnh, tên, symbol, rank

```
<div className="crypto-price">
  <p className="price">{formatPrice(crypto.current_price)}</p>
  <p
      className={`change ${
          crypto.price_change_percentage_24h >= 0 ? "positive" : "negative"
      }`}
```

```
    >
      {crypto.price_change_percentage_24h >= 0 ? "↑" : "↓"}{" "}
      {Math.abs(crypto.price_change_percentage_24h).toFixed(2)}%
    </p>
  </div>
```

Giải thích:

- Hiển thị giá và % thay đổi
- Conditional className: "positive" hoặc "negative"
- `Math.abs()`: Lấy giá trị tuyệt đối

```
<div className="crypto-stats">
  <div className="stat">
    <span className="stat-label">Market Cap</span>
    <span className="stat-value">
      ${formatMarketCap(crypto.market_cap)}
    </span>
  </div>
  <div className="stat">
    <span className="stat-label">Volume</span>
    <span className="stat-value">
      ${formatMarketCap(crypto.total_volume)}
    </span>
  </div>
</div>
```

Giải thích:

- Hiển thị Market Cap và Volume

---

11. `src/components/CryptoList.tsx` - Danh Sách Cards

```
import React from "react";
import { CryptoCard } from "../components/CryptoCard";
import type {Crypto} from "../pages/Home";

interface CryptoListProps {
  cryptos: Crypto[];
  viewMode: "grid" | "list";
}

export const CryptoList: React.FC<CryptoListProps> = ({ cryptos, viewMode }) => (
    <div className={`crypto-container ${viewMode}`}>
      {cryptos.map((crypto) => (
          <CryptoCard crypto={crypto} key={crypto.id} />
      ))}
```

```
    </div>
);
```

Giải thích từng phần:

1. **Props Interface**

```
interface CryptoListProps {
  cryptos: Crypto[];              // Mảng các coin
  viewMode: "grid" | "list";      // Chế độ hiển thị
}
```

2. **Component**

```
export const CryptoList: React.FC<CryptoListProps> = ({ cryptos, viewMode }) => (
    <div className={`crypto-container ${viewMode}`}>
      {cryptos.map((crypto) => (
          <CryptoCard crypto={crypto} key={crypto.id} />
      ))}
    </div>
);
```

- `className={`crypto-container ${viewMode}`}`: Dynamic class (grid hoặc list)
- `cryptos.map()`: Render mỗi crypto thành CryptoCard
- `key={crypto.id}`: Key unique cho React

Cách hoạt động:

- Nhận mảng `cryptos` và `viewMode`
- Render mỗi crypto thành `CryptoCard`
- CSS sẽ style khác nhau cho grid/list

---

12. `src/components/Controls.tsx` - Điều Khiển

```
interface ControlsProps {
  sortBy: "market_cap_rank" | "name" | "price" | "price_desc" | "change" |
"market_cap";
  setSortBy: React.Dispatch<React.SetStateAction<...>>;
  viewMode: "grid" | "list";
  setViewMode: React.Dispatch<React.SetStateAction<"grid" | "list">>;
}
```

Giải thích:

- Props nhận state và setter từ parent

```
export const Controls: React.FC<ControlsProps> = ({ sortBy, setSortBy, viewMode,
setViewMode }) => (
    <div className="controls">
      <div className="filter-group">
        <label>Sort by:</label>
        <select
            value={sortBy}
            onChange={(e) =>
                setSortBy(e.target.value as
                    "market_cap_rank" | "name" | "price" | "price_desc" | "change"
| "market_cap"
                )
            }
        >
          <option value="market_cap_rank">Rank</option>
          <option value="name">Name</option>
          <option value="price">Price (Low to High)</option>
          <option value="price_desc">Price (High to Low)</option>
          <option value="change">24h Change</option>
          <option value="market_cap">Market Cap</option>
        </select>
      </div>
```

Giải thích:

- `select`: Dropdown để chọn sort option
- `value={sortBy}`: Controlled component
- `onChange`: Cập nhật state khi thay đổi
- `as`: Type assertion cho TypeScript

```
<div className="view-toggle">
  <button
      className={viewMode === "grid" ? "active" : ""}
      onClick={() => setViewMode("grid")}
  >
    Grid
  </button>
  <button
      className={viewMode === "list" ? "active" : ""}
      onClick={() => setViewMode("list")}
  >
    List
  </button>
</div>
```

Giải thích:

- 2 button để toggle view mode
- Conditional className: "active" khi được chọn

13. `src/layout/MainLayout.tsx` - Layout Chính

```
import React from "react";
import { Header } from "../components/Header";
import { Footer } from "../components/Footer";

interface LayoutProps {
  children: React.ReactNode;
  searchQuery?: string;
  setSearchQuery?: (value: string) => void;
  coinName?: string;
  onBack?: () => void;
}
```

Giải thích:

- `children`: Nội dung bên trong layout
- Optional props cho search và back button

```
export const Layout: React.FC<LayoutProps> = ({
  children,
  searchQuery,
  setSearchQuery,
  coinName,
  onBack,
}) => {
  return (
      <div className="app">
        <Header
            searchQuery={searchQuery}
            setSearchQuery={setSearchQuery}
            coinName={coinName}
            onBack={onBack}
        />
        <main>{children}</main>
        <Footer />
      </div>
  );
};
```

Giải thích:

- Wrapper component
- `Header`: Component header
- `main`: Nội dung chính (children)
- `Footer`: Component footer

Cách sử dụng:

```
<Layout searchQuery={searchQuery} setSearchQuery={setSearchQuery}>
  <HomeContent />
</Layout>
```

---

14. src/components/Header.tsx - Header Component

```
interface HeaderProps {
  searchQuery?: string;
  setSearchQuery?: (value: string) => void;
  coinName?: string;
  onBack?: () => void;
}
```

Giải thích:

- Optional props để linh hoạt

```
export const Header: React.FC<HeaderProps> = ({
  searchQuery,
  setSearchQuery,
  coinName,
  onBack,
}) => {
  return (
      <header className="header">
        <div className="header-content">
          <div className="logo-section">
            <h1>🚀 Crypto Tracker</h1>
            <p>Real-time cryptocurrency prices and market data</p>
          </div>
```

Giải thích:

- Logo và mô tả

```
{coinName ? (
    <div className="coin-header-controls">
      <button className="back-button" onClick={onBack}>
        ← Back
      </button>
    </div>
) : (
    <div className="search-section">
      <input
          type="text"
```

```
            placeholder="Search cryptos..."
            className="search-input"
            value={searchQuery}
            onChange={(e) => setSearchQuery?.(e.target.value)}
        />
      </div>
)}
```

Giải thích:

- Conditional rendering:
  - Nếu có `coinName`: Hiển thị back button (trang chi tiết)
  - Nếu không: Hiển thị search (trang chủ)
- `setSearchQuery?.(e.target.value)`: Optional chaining

---

15. `src/pages/CoinDetail.tsx` - Trang Chi Tiết Coin

```
import React, { useEffect, useState } from "react";
import {useNavigate, useParams} from "react-router-dom";
import {fetchCoinData, fetchChartData, type ChartData} from
"../services/coinGecko";
import type { CoinData } from "../services/coinGecko";
```

Giải thích:

- `useParams`: Lấy params từ URL
- `useNavigate`: Điều hướng programmatically

```
export const CoinDetail: React.FC = () => {
  const navigate = useNavigate();
  const { id } = useParams<{ id: string }>();
  const [coin, setCoin] = useState<CoinData | null>(null);
  const [chartData, setChartData] = useState<ChartPoint[]>([]);
  const [isLoading, setIsLoading] = useState(true);
```

Giải thích:

- `useParams<{ id: string }>()`: Lấy `id` từ URL `/coin/:id`
- State để lưu coin data, chart data, và loading

```
useEffect(() => {
  if (!id) return;

  const loadData = async () => {
    try {
      const coinData = await fetchCoinData(id);
```

```
      const chart: ChartData = await fetchChartData(id);

      const formattedChart: ChartPoint[] = chart.prices.map(
          (p: [number, number]) => ({
            time: new Date(p[0]).toLocaleDateString("en-US", {
              month: "short",
              day: "numeric",
            }),
            price: Number(p[1].toFixed(2)),
          })
      );

      setCoin(coinData);
      setChartData(formattedChart);
    } catch (err) {
      console.error(err);
    } finally {
      setIsLoading(false);
    }
  };

  loadData();
}, [id]);
```

Giải thích từng phần:

1. **Check ID**

```
if (!id) return;
```

- Nếu không có id, return sớm

2. **Fetch Data**

```
const coinData = await fetchCoinData(id);
const chart: ChartData = await fetchChartData(id);
```

- Lấy dữ liệu coin và chart song song

3. **Format Chart Data**

```
const formattedChart: ChartPoint[] = chart.prices.map(
    (p: [number, number]) => ({
      time: new Date(p[0]).toLocaleDateString("en-US", {
        month: "short",
        day: "numeric",
      }),
      price: Number(p[1].toFixed(2)),
```

```
    })
);
```

- `p[0]`: Timestamp
- `p[1]`: Price
- Format timestamp thành date string
- Format price thành 2 chữ số thập phân

```
if (isLoading) return <Loading />;
if (!coin) return <NoResults />;
```

Giải thích:

- Early return nếu loading hoặc không có data

```
return (
    <Layout coinName={coin.name} onBack={() => navigate("/")}>
      <div className="coin-detail">
        <div className="coin-header">
          <div className="coin-title">
            <img src={coin.image?.large} alt={coin.name} />
            <div>
              <h1>{coin.name}</h1>
              <p className="symbol">{coin.symbol.toUpperCase()}</p>
            </div>
          </div>
          <span className="rank">Rank #{coin.market_data?.market_cap_rank}</span>
        </div>

        <PriceSection coin={coin} />
        <ChartSection chartData={chartData} />
        <StatsGrid coin={coin} />
      </div>
    </Layout>
);
```

Giải thích:

- Render các component con
- `coin.image?.large`: Optional chaining
- `navigate("/")`: Quay về trang chủ

---

16. `src/components/PriceSection.tsx` - Phần Giá

```
export const PriceSection: React.FC<PriceSectionProps> = ({ coin }) => {
  const priceChange = coin.market_data?.price_change_percentage_24h || 0;
```

```
  const isPositive = priceChange >= 0;

  return (
      <div className="coin-price-section">
        <div className="current-price">
          <h2>{formatPrice(coin.market_data?.current_price.usd || 0)}</h2>
          <span className={`change-badge ${isPositive ? "positive" :
"negative"}`}>
            {isPositive ? "↑" : "↓"} {Math.abs(priceChange).toFixed(2)}%
          </span>
        </div>

        <div className="price-ranges">
          <div className="price-range">
            <span className="range-label">24h High</span>
            <span className="range-value">
{formatPrice(coin.market_data?.high_24h.usd || 0)}</span>
          </div>
          <div className="price-range">
            <span className="range-label">24h Low</span>
            <span className="range-value">
{formatPrice(coin.market_data?.low_24h.usd || 0)}</span>
          </div>
        </div>
      </div>
  );
};
```

Giải thích:

- Hiển thị giá hiện tại, % thay đổi, và giá cao/thấp 24h
- Conditional styling cho positive/negative

---

17. src/components/ChartSection.tsx - Biểu Đồ

```
import React from "react";
import { LineChart, ResponsiveContainer, CartesianGrid, XAxis, YAxis, Line,
Tooltip } from "recharts";
```

Giải thích:

- Import các component từ Recharts

```
export const ChartSection: React.FC<ChartSectionProps> = ({ chartData }) => (
    <div className="chart-section">
      <h3>Price Chart (7 Days)</h3>
      <ResponsiveContainer width="100%" height={400}>
        <LineChart data={chartData}>
```

```
          <CartesianGrid strokeDasharray="3 3" stroke="rgba(255, 255, 255, 0.1)"
/>
          <XAxis dataKey="time" stroke="#9ca3af" style={{ fontSize: "12px" }} />
          <YAxis stroke="#9ca3af" style={{ fontSize: "12px" }} domain={["auto",
"auto"]} />
          <Tooltip
              contentStyle={{
                backgroundColor: "rgba(20, 20, 40, 0.95)",
                border: "1px solid rgba(255, 255, 255, 0.1)",
                borderRadius: "8px",
                color: "#e0e0e0",
              }}
          />
          <Line type="monotone" dataKey="price" stroke="#ADD8E6" strokeWidth={2}
dot={false} />
        </LineChart>
      </ResponsiveContainer>
    </div>
);
```

Giải thích từng phần:

1. **ResponsiveContainer**

```
<ResponsiveContainer width="100%" height={400}>
```

- Container tự động resize theo parent

2. **LineChart**

```
<LineChart data={chartData}>
```

- Component biểu đồ đường

3. **CartesianGrid**

```
<CartesianGrid strokeDasharray="3 3" stroke="rgba(255, 255, 255, 0.1)" />
```

- Lưới nền (đường đứt nét)

4. **XAxis, YAxis**

```
<XAxis dataKey="time" stroke="#9ca3af" />
<YAxis stroke="#9ca3af" domain={["auto", "auto"]} />
```

- Trục X: Hiển thị time
- Trục Y: Tự động scale

5. **Tooltip**

```
<Tooltip contentStyle={{...}} />
```

- Tooltip khi hover

6. **Line**

```
<Line type="monotone" dataKey="price" stroke="#ADD8E6" strokeWidth={2} dot={false}
/>
```

- Đường biểu đồ
- `type="monotone"`: Làm mượt đường
- `dataKey="price"`: Lấy giá trị từ field "price"
- `dot={false}`: Ẩn chấm trên đường

---

18. `src/components/StatsGrid.tsx` - Lưới Thống Kê

```
export const StatsGrid: React.FC<StatsGridProps> = ({ coin }) => (
    <div className="stats-grid">
      <div className="stat-card">
        <span className="stat-label">Market Cap</span>
        <span className="stat-
value">${formatMarketCap(coin.market_data?.market_cap.usd || 0)}</span>
      </div>
      <div className="stat-card">
        <span className="stat-label">Volume (24h)</span>
        <span className="stat-
value">${formatMarketCap(coin.market_data?.total_volume.usd || 0)}</span>
      </div>
      <div className="stat-card">
        <span className="stat-label">Circulating Supply</span>
        <span className="stat-value">
{coin.market_data?.circulating_supply?.toLocaleString() || "N/A"}</span>
      </div>
      <div className="stat-card">
        <span className="stat-label">Total Supply</span>
        <span className="stat-value">
{coin.market_data?.total_supply?.toLocaleString() || "N/A"}</span>
      </div>
    </div>
);
```

Giải thích:

- Grid 4 cột hiển thị thống kê
- `toLocaleString()`: Format số với dấu phẩy (ví dụ: 1,000,000)

Cách Các Component Hoạt Động Với Nhau

Sơ Đồ Component Tree

```
App
└── BrowserRouter
    └── Routes
        ├── Route "/" → Home
        │   └── Layout
        │       ├── Header (search)
        │       ├── Controls
        │       ├── CryptoList
        │       │   └── CryptoCard (x100)
        │       └── Footer
        │
        ├── Route "/coin/:id" → CoinDetail
        │   └── Layout
        │       ├── Header (back button)
        │       ├── PriceSection
        │       ├── ChartSection
        │       ├── StatsGrid
        │       └── Footer
        │
        └── Route "/news" → CryptoNews
            └── NewsList
                └── NewsCard (xN)
```

Luồng Dữ Liệu

1. Trang Chủ (Home)

```
Home Component
  ↓ (state)
  ├── cryptoList: Lưu data từ API
  ├── filteredList: Data sau khi filter/sort
  ├── searchQuery: Từ khóa tìm kiếm
  ├── sortBy: Tiêu chí sắp xếp
  └── viewMode: Chế độ hiển thị
  ↓ (props)
  ├── Layout
  │   └── Header (searchQuery, setSearchQuery)
  ├── Controls (sortBy, setSortBy, viewMode, setViewMode)
  └── CryptoList (filteredList, viewMode)
      └── CryptoCard (crypto) x N
```

Luồng hoạt động:

1. `Home` mount → `useEffect` gọi `fetchCryptoData()`
2. `fetchCryptoData()` → Gọi API → Cập nhật `cryptoList`
3. `useEffect` (khi `sortBy`, `cryptoList`, `searchQuery` thay đổi) → Gọi `filterAndSort()`
4. `filterAndSort()` → Filter và sort → Cập nhật `filteredList`
5. `CryptoList` nhận `filteredList` → Render `CryptoCard` cho mỗi item
6. User click `CryptoCard` → `Link` điều hướng đến `/coin/:id`

2. Trang Chi Tiết (CoinDetail)

```
CoinDetail Component
  ↓ (useParams)
  └── id: Lấy từ URL
  ↓ (useEffect)
  ├── fetchCoinData(id) → coin state
  └── fetchChartData(id) → chartData state
  ↓ (props)
  ├── Layout
  │   └── Header (coinName, onBack)
  ├── PriceSection (coin)
  ├── ChartSection (chartData)
  └── StatsGrid (coin)
```

Luồng hoạt động:

1. User vào `/coin/bitcoin`
2. `CoinDetail` mount → `useParams` lấy `id = "bitcoin"`
3. `useEffect` chạy → Gọi `fetchCoinData("bitcoin")` và `fetchChartData("bitcoin")`
4. Data trả về → Cập nhật `coin` và `chartData`
5. Render các component con với data

3. Component Reusable

Layout Component:

- Được dùng bởi: `Home`, `CoinDetail`
- Props khác nhau:
  - Home: `searchQuery`, `setSearchQuery`
  - CoinDetail: `coinName`, `onBack`

Header Component:

- Conditional rendering dựa trên props
- Nếu có `coinName`: Hiển thị back button
- Nếu không: Hiển thị search

CryptoCard Component:

- Được dùng bởi: `CryptoList`
- Nhận `crypto` object
- Render thông tin và link đến detail

Luồng Dữ Liệu Trong Ứng Dụng

1. Fetch Data Flow

```
User Action
  ↓
Component (Home/CoinDetail)
  ↓
Service (coinGecko.ts)
  ↓
API (CoinGecko)
  ↓
Response (JSON)
  ↓
Service (parse data)
  ↓
Component (setState)
  ↓
Re-render UI
```

2. State Management Flow

```
Home Component State:
  cryptoList (source of truth)
    ↓
  filterAndSort()
    ↓
  filteredList (derived state)
    ↓
  CryptoList Component
    ↓
  CryptoCard Components
```

3. User Interaction Flow

```
User types in search
  ↓
setSearchQuery("bitcoin")
  ↓
useEffect triggers
```

```
  ↓
filterAndSort() runs
  ↓
filteredList updates
  ↓
CryptoList re-renders
  ↓
Only matching cards shown
```

Tổng Kết

Các Khái Niệm Quan Trọng

1. **React Hooks:**
   - `useState`: Quản lý state
   - `useEffect`: Side effects (API calls, intervals)
   - `useParams`: Lấy params từ URL
   - `useNavigate`: Điều hướng programmatically
2. **Component Patterns:**
   - **Presentational Components**: Chỉ hiển thị UI (CryptoCard, Loading)
   - **Container Components**: Quản lý logic và state (Home, CoinDetail)
   - **Layout Components**: Wrapper cho layout (MainLayout)
3. **TypeScript:**
   - Interfaces: Định nghĩa cấu trúc dữ liệu
   - Type safety: Kiểm tra type tại compile time
   - Autocomplete: IDE gợi ý code
4. **React Router:**
   - Client-side routing
   - Dynamic routes với params
   - Navigation với Link và navigate()
5. **API Integration:**
   - Fetch API để gọi HTTP requests
   - Async/await để xử lý bất đồng bộ
   - Error handling với try/catch

Best Practices

1. **Component Structure:**
   - Tách component nhỏ, tái sử dụng được
   - Props interface rõ ràng

- Single responsibility

2. **State Management:**
   - State ở component cần thiết nhất
   - Derived state từ source state
   - Cleanup effects
3. **Code Organization:**
   - Services cho API calls
   - Utils cho helper functions
   - Components tách biệt logic và UI
4. **TypeScript:**
   - Type mọi thứ
   - Sử dụng interfaces
   - Avoid `any` type

Tài Liệu Tham Khảo

- React Documentation
- TypeScript Handbook
- Vite Guide
- React Router
- Recharts Documentation
- CoinGecko API

Chúc bạn thành công với dự án! 🚀