## Phase 1: Planning & Design

**1. Define Requirements & Sketch UI/UX:**

- **User Flows & Wireframes:**
  - Draw rough sketches (or use a tool like Figma) for the dashboard layout.
  - Define three main sections:
    - **Central AI Prompt:** Chat-like interface for queries.
    - **Portfolio Analysis – News Room:** Display aggregated trade analytics and performance charts.
    - **Wallet Management – Fort Knox:** List and manage wallets, show balances, and history.
- **API Contract:**
  - List endpoints you'll need (e.g., `/api/portfolio`, `/api/ask`, `/api/wallet`).
  - Define the data format (what data is sent to and returned from each endpoint).

**2. Project Scope Documentation:**

- Write a brief document summarizing your project's scope, MVP features, and tech stack choices (FastAPI for backend, React for frontend, Covalent/The Graph for data aggregation, OpenAI for AI analysis).
- Decide on development milestones and priorities.

---

## Phase 2: Environment Setup

**1. Set Up Your Coding Environment (Cursor AI):**

- **Install and Configure Cursor AI:**
  - Use Cursor as your primary editor; it's similar to VS Code.
  - Ensure the Python extension is installed for language support.
- **Create Python Virtual Environment:**
  - Run `python -m venv env` and activate it.
  - Install FastAPI, Uvicorn, and other dependencies (e.g., `pip install fastapi uvicorn openai requests`).

**2. Initialize Your Frontend Project:**

- **Using Create React App:**
  - Run `npx create-react-app my-crypto-portfolio` in your terminal.
  - Open this project in Cursor for development.

- **Install UI Libraries:**
  - Consider using a component library like Material-UI or Chakra UI to speed up UI design.

---

## Phase 3: UI Prototype with Mock Data

**1. Build the React UI Components:**

- **Central AI Prompt Panel:**
  - Create a chat interface component with a scrollable message area and an input field.
  - Use sample/mock responses to simulate AI output.
- **Portfolio Analysis Dashboard:**
  - Design components for charts (e.g., pie chart for asset allocation, line chart for performance over time).
  - Use placeholder data to lay out tables, charts, and summary stats.
- **Wallet Management Panel:**
  - Build a simple form for entering wallet addresses.
  - List mock wallets and display static information (creation date, last trade, etc.).

**2. Wire up Basic Navigation:**

- Create a basic layout (e.g., using CSS grid or Flexbox) that positions these panels similarly to pro.x.com's column design.

---

## Phase 4: Backend Development with FastAPI

**1. Develop Your API Endpoints:**

- **Create Endpoints:**
  - /api/portfolio – returns aggregated portfolio data.
  - /api/ask – accepts user questions and returns AI responses.
  - /api/wallet – manages wallet connections and returns wallet-specific analytics.
- **Dummy Data Implementation:**
  - Start with static responses to allow the frontend to integrate and test data flows.
- **API Documentation:**

- Leverage FastAPI's auto-generated docs (available at /docs) to review your endpoints and test them.

**2. Set Up Local Testing:**

- Use Uvicorn to run your FastAPI server (uvicorn main:app --reload).
- Test endpoints with tools like Postman or directly through the browser.

---

## Phase 5: Integration of Data Aggregation & AI

**1. Integrate Data Aggregation (Covalent/The Graph):**

- **Choose Your Data Source:**
  - For a broad wallet view, start with the Covalent API for fetching balances and transaction history.
  - Later, you can incorporate The Graph for specialized protocol data if needed.
- **Implement API Calls:**
  - Write functions in FastAPI that call Covalent's endpoints using your API key (stored securely in an environment variable).
  - Parse and normalize the data to match your API contract.

**2. Add OpenAI API Integration for AI Queries:**

- **Set Up API Key:**
  - Store your OpenAI API key securely (e.g., in a .env file).
- **Develop AI Service Module:**
  - Write a function (in Python) that takes a user's question (and any additional context), calls OpenAI's Chat Completion API, and returns the generated response.
- **Connect AI Endpoint:**
  - Hook this service into your /api/ask endpoint so that when the frontend sends a query, the backend fetches and returns an AI-generated response.

---

## Phase 6: Frontend & Backend Integration

**1. Connect the React UI to FastAPI:**

- **API Calls in React:**
  - Use Axios or the Fetch API to call your FastAPI endpoints.
  - Replace static/mock data with live data from your backend.

- **Error Handling & Loading States:**
  - Add loading indicators for AI queries and data fetches.
  - Handle errors gracefully in the UI.

**2. End-to-End Testing:**

- Test user flows (from entering a query to receiving AI insights) to ensure smooth integration between frontend and backend.

---

## Phase 7: Security and Final Touches

**1. Secure Your Environment:**

- **API Keys & Secrets:**
  - Ensure all API keys (OpenAI, Covalent) are stored in environment variables and not hard-coded.
- **Endpoint Security:**
  - For your backend endpoints, implement simple authentication (like a token check) if you ever expose the app beyond local development.
- **Data Encryption:**
  - If you store any sensitive information (wallet details, credentials), consider encrypting those fields.

**2. UI/UX Polishing:**

- Refine your design based on your initial sketches.
- Ensure the UI is responsive and that components have consistent styling.
- Fine-tune any interactive elements for a smooth user experience.

**3. Final Testing & Debugging:**

- Perform comprehensive testing, both automated (if possible) and manual, to ensure that all components work as expected.
- Iterate based on your own usage feedback.

---

## Phase 8: Deployment and Future Enhancements

**1. Deployment Options:**

- For local use, you can run both the FastAPI backend and the React frontend on your computer.

- For remote access or backups, consider deploying the backend on a cloud service (Heroku, AWS, or DigitalOcean) and serving the frontend using a static site host (Netlify, Vercel).

**2. Future Roadmap:**

- **Extend Data Sources:**
  - Integrate additional APIs for CEX data (e.g., Binance, Coinbase) using libraries like CCXT.
- **Enhance AI Capabilities:**
  - Expand AI insights, refine prompts, and possibly enable more interactive commands (like simulated trade execution).
- **Advanced Security:**
  - If your app evolves, consider adding more robust authentication and encryption measures.

---

# Project Plan: Phase 1 – Planning & Design

## 1. Project Overview

### Project Name:

AI-Powered Crypto Portfolio Manager

### Objective:

Develop an AI-driven application enabling automated aggregation, management, and analysis of cryptocurrency transactions from centralized (CEX) and decentralized exchanges (DEX), integrated with OpenAI for intelligent portfolio queries and insights.

### End User:

Single-user retail trader aiming for improved decision-making and competitive advantage.

---

## 2. Development Approach

### Lifecycle Methodology:

**Agile / Iterative Development**

- Utilize short, iterative sprints.
- Continuous feedback loop for rapid iterations.
- Flexibility to adapt requirements based on evolving insights.

### Development Phases:

- **Phase 1: Planning & Design** *(Current Phase)*
- **Phase 2: Environment & Initial UI Setup**
- **Phase 3: Backend Development & API Integration**
- **Phase 4: AI Integration & Testing**
- **Phase 5: Security, Optimization, and Deployment**
- **Phase 6: Future Enhancements & Scaling**

---

# 3. Defining Requirements

### Core Functional Requirements:

#### Main Feature Sets:

- **Central AI Prompt (Main Area):**
  - ChatGPT-powered conversational interface.
  - User can query portfolio balances, transactions, and performance.
- **Portfolio Analysis – News Room:**
  - Historical trade analytics (win/loss, profitability, trade size).
  - Trend recognition (market cycles analysis: bullish, bearish, accumulation).
  - Visual analytics (interactive charts).
- **Wallet Management – Fort Knox:**
  - Wallet aggregation (balances, transaction histories from CEX & DEX).
  - Interactive visualizations (bubble charts).
  - Detailed wallet analytics (creation date, interactions, successful trades).
  - Security features (dust detection, revoke approvals, scam alerts).

### Non-Functional Requirements:

- Responsive, intuitive, and performant user interface.
- Secure handling of sensitive data (wallet addresses, API keys).
- Efficient backend processing and scalable architecture.
- Clear documentation (both technical/API and end-user guides).

---

# 4. Technology Stack & Tools

### Frontend:

- **Framework:** ReactJS (with Create React App)
- **Component Libraries:** Mantine (for rapid prototyping)
- **Charting Library:** Recharts or Chart.js

### Backend:

- **Language:** Python 3.x
- **Web Framework:** FastAPI (async processing, auto-documentation, validation)
- **Server:** Uvicorn (ASGI)

### AI Integration:

- **OpenAI API:** GPT-4 or GPT-3.5 Turbo
- **API Client:** openai Python package

### Data Aggregation APIs:

- **On-chain data:**
  - **Primary Choice:** Covalent (general wallet balances, historical transactions)
  - **Secondary Choice:** The Graph (for detailed DeFi-specific queries)
- **CEX Integration:**
  - Binance and Coinbase official APIs
  - CCXT (Unified Crypto Exchange API)

### Development Environment:

- **Editor/IDE:** Cursor AI
- **Virtual Environment Management:** Python's built-in venv
- **Version Control:** Git/GitHub
- **Documentation & Collaboration:** Notion, GitHub Projects, Markdown docs
- **Testing Tools:** Postman (API testing), Jest (Frontend), Pytest (Backend)

---

## 5. APIs & Data Sources

### Primary APIs:

| Service | Purpose | Integration Method |
|---|---|---|
| **Covalent API** | Aggregate on-chain data (wallet balances, transactions) | RESTful API via requests |
| **OpenAI API** | AI-driven insights and conversational analytics | RESTful API via Python SDK |
| **CCXT Library** | Centralized exchange account integration | Python SDK |

### Secondary APIs:

| Service | Purpose | Integration Method |
|---|---|---|
| **The Graph** | DEX data and specific DeFi insights | GraphQL via requests |
| **Web3 RPC** (Infura, Alchemy) | Direct blockchain queries | Python web3 library |

---

# 6. Project Structure & Codebase

### Recommended Project Folder Structure (Industry Standard):

```
ai-crypto-portfolio-manager/
├── backend/                # FastAPI backend
│   ├── app/
│   │   ├── routers/        # API route handlers
│   │   ├── services/       # Business logic (AI integration, data aggregation)
│   │   ├── models/         # Pydantic schemas and validation
│   │   ├── core/           # Security, settings, and configuration
│   │   └── main.py         # Entry point for FastAPI
│   └── tests/              # Unit and integration tests
│
├── frontend/               # React frontend
│   ├── public/
│   ├── src/
│   │   ├── components/     # UI components
│   │   ├── pages/          # Individual pages and views
│   │   ├── services/       # API integrations
│   │   ├── utils/          # Utility functions (data formatting, API helpers)
│   │   └── App.js          # Main React app component
│
├── docs/                   # Technical documentation and project notes
├── .env                    # Environment variables
└── README.md               # Project overview and setup instructions
```

---

# 7. Security Considerations (Initial Plan)

- **API Keys:** Stored securely in environment variables (.env file, not committed to GitHub).
- **Endpoints:** FastAPI endpoints protected with simple token-based authentication (for single-user scenario).
- **Wallet Security:** Do not store private keys directly; consider hardware wallet integrations.
- **Communication:** HTTPS for all external API interactions.
- **Data Storage:** Local storage encryption (if necessary).

---

## 8. Timeline & Priorities

| Task | Duration | Priority |
|---|---|---|
| **Phase 1 (Planning & Design)** | 1 Week | High |
| Requirements & Wireframes | 2-3 days | High |
| Define API Contract | 1-2 days | High |
| Technology Stack Decision | 1 day | Medium |
| Initial Security Plan | 1 day | Medium |
| Project Documentation & Setup | Concurrent | Medium |

| Next Phases (for reference) | Duration | Priority |
|---|---|---|
| Phase 2 (Setup & UI) | 1-2 Weeks | High |
| Phase 3 (Backend & APIs) | 2 Weeks | High |
| Phase 4 (AI Integration & Testing) | 1-2 Weeks | High |
| Phase 5 (Security & Optimization) | 1 Week | Medium |
| Phase 6 (Deployment & Scaling) | Ongoing | Medium |

**Recommended Total MVP Timeline:**
Approximately **6-8 weeks** to achieve MVP completion and usability.

---

## 9. Key Success Factors & Industry Best Practices

- **Iterative Development:** Rapid prototyping, frequent testing.
- **Clear API Contracts:** Establish clear data schemas early.
- **Modularity:** Keep frontend/backend logic and code modular and organized.
- **Continuous Integration & Delivery (CI/CD):** Automate deployment/testing if feasible.
- **Secure by Default:** Always assume sensitive data requires highest security standards.

---

## 10. Tools for Task Management & Progress Tracking

- **GitHub Issues / Projects** for task management, bug tracking.
- **Notion or Jira (light)** for overall project tracking and notes.
- **Weekly personal retrospectives:** to evaluate progress and learning milestones.

---

**Next Steps:**

- **Immediately:** Start sketching wireframes and defining your API schema clearly.
- **Short-term:** Set up your project repository, initialize your dev environments (React/FastAPI).
- **Continued:** Begin early implementation of your frontend prototype with mocked data.

---