TÀI LIỆU: TỔNG QUAN KIẾN TRÚC VÀ CÔNG NGHỆ

Dự án: Hệ thống AI Hỗ trợ Đầu tư Cá nhân Thông minh và Minh bạch (ITAPIA)
Phiên bản: 1.0
Ngày: 19/06/2025

1. TỔNG QUAN

Tài liệu này trình bày chi tiết về kiến trúc hệ thống, các thành phần công nghệ được lựa chọn, và luồng dữ liệu chính của dự án ITAPIA. Mục tiêu của tài liệu là cung cấp một bản thiết kế kỹ thuật (technical blueprint) rõ ràng, làm cơ sở cho việc triển khai và phát triển hệ thống.

2. CÁC NGUYÊN TẮC KIẾN TRÚC CỐT LÕI

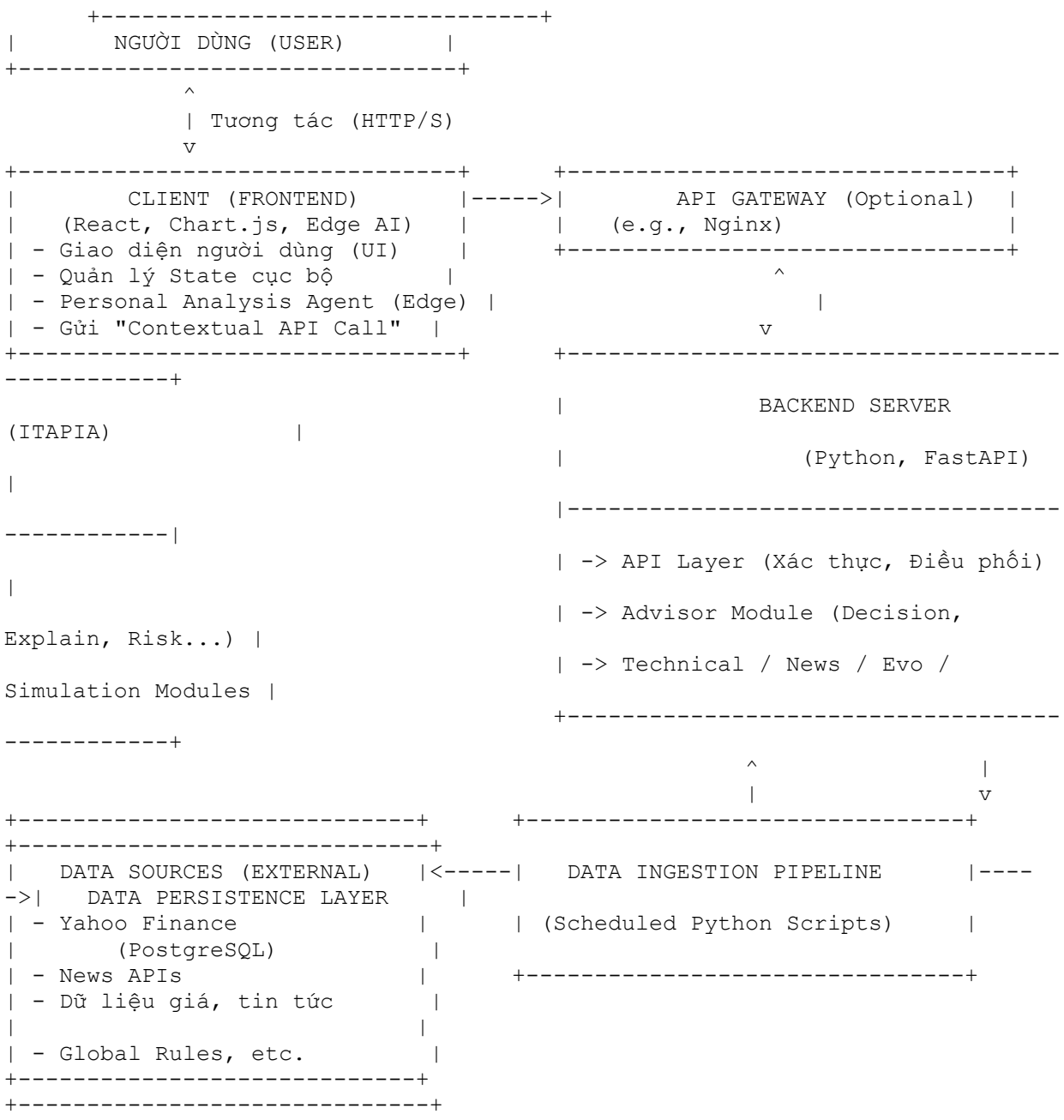
Kiến trúc của ITAPIA được xây dựng dựa trên các nguyên tắc nền tảng sau để đảm bảo đáp ứng được các mục tiêu của dự án:

- **AI Engine trên Server, Context từ Client:** Toàn bộ các tác vụ tính toán nặng (AI, LLM, Thuật toán Tiến hóa, Backtesting) được thực hiện trên server để đảm bảo hiệu năng và không làm quá tải thiết bị người dùng. Client chịu trách nhiệm cung cấp "bối cảnh" (context) cho mỗi yêu cầu.
- **API Phi trạng thái (Stateless API Design):** Server không lưu trữ trạng thái hay dữ liệu phiên làm việc của người dùng giữa các yêu cầu. Mỗi yêu cầu API từ client đều chứa tất cả thông tin cần thiết để server xử lý. Điều này giúp hệ thống dễ dàng mở rộng, bảo trì và tăng tính bảo mật.
- **Module hóa và Tách biệt Mối quan tâm (Modularity & Separation of Concerns):** Hệ thống được chia thành các module chức năng rõ ràng (Data, Technical, News, Advisor, Evo, Simulation), giúp việc phát triển, kiểm thử và nâng cấp từng phần trở nên độc lập và dễ dàng hơn.
- **Bảo mật và Người dùng làm Trung tâm (Privacy-First & User-in-Control):** Dữ liệu và logic cá nhân hóa (như bộ quy tắc) được ưu tiên lưu trữ tại client. Dữ liệu chỉ được gửi lên server khi cần thiết cho một tác vụ cụ thể và không được lưu trữ lâu dài mà không có sự cho phép rõ ràng.
- **Mã nguồn mở và Chi phí thấp (Open-Source & Cost-Effective):** Ưu tiên tối đa việc sử dụng các công nghệ mã nguồn mở, đã được kiểm chứng để giảm chi phí phát triển và vận hành, giúp sản phẩm dễ tiếp cận.

3. KIẾN TRÚC HỆ THỐNG TỔNG THỂ (HIGH-LEVEL ARCHITECTURE)

Hệ thống được thiết kế theo mô hình Client-Server với các thành phần chính như sau:

Generated code



4. CHI TIẾT CÁC THÀNH PHẦN CÔNG NGHỆ

4.1. Client (Frontend)

- **Framework: React.js**.
  - *Lý do:* Hệ sinh thái mạnh mẽ, cộng đồng lớn, quản lý state hiệu quả (với Context API hoặc Redux Toolkit), phù hợp để xây dựng Giao diện Người dùng Đơn trang (SPA) phức tạp.

- **Thư viện Biểu đồ: TradingView Lightweight Charts** hoặc **Chart.js** với các plugin tài chính.
    - *Lý do:* Cung cấp các biểu đồ tài chính hiệu năng cao, dễ dàng tùy chỉnh và tích hợp.
- **Trách nhiệm chính:**
    - Hiển thị giao diện người dùng, bao gồm dashboard, chatbot và các view chi tiết.
    - Lưu trữ và quản lý **Bộ nhớ Quy tắc Cá nhân (Local Rule Set)** và **Hồ sơ Người dùng (User Profile)** cục bộ (sử dụng localStorage hoặc IndexedDB).
    - Thực thi **Personal Analysis Agent** (dưới dạng các logic JavaScript đơn giản) để phân tích hành vi người dùng tại client-side.
    - Tạo và gửi các **"Contextual API Call"** chứa đầy đủ bối cảnh (ticker, profile, ruleset) đến Backend.

4.2. Backend (Server)

- **Ngôn ngữ & Framework: Python & FastAPI**.
    - *Lý do:* Python có hệ sinh thái AI/ML/Data Science mạnh nhất. FastAPI cung cấp hiệu năng vượt trội, tự động sinh tài liệu API (Swagger UI), và tích hợp chặt chẽ với Pydantic để xác thực dữ liệu.
- **Kiến trúc Backend: Monolithic**.
    - *Lý do:* Đơn giản hóa việc phát triển và triển khai cho đồ án một người. Các module logic bên trong vẫn được tách biệt rõ ràng.
- **Chi tiết các Module Backend:**
    - **API Layer:** Chịu trách nhiệm nhận request, xác thực dữ liệu đầu vào bằng Pydantic, điều phối đến các module xử lý tương ứng và định dạng dữ liệu trả về.
    - **Technical Module:** Sử dụng **Pandas, NumPy, và TA-Lib/bta-lib** để tính toán các chỉ báo kỹ thuật từ dữ liệu giá. Là một module tính toán thuần túy.
    - **News Module:** Sử dụng **VADER** (cho sentiment cơ bản) hoặc một mô hình nhỏ từ **Hugging Face Transformers** (ví dụ: distilbert-base-uncased-finetuned-sst-2-english) để gán nhãn direction và level cho tin tức.
    - **Decision Maker Agent:** Nhận "bối cảnh" từ client, áp dụng bộ quy tắc được gửi lên (active_ruleset) vào dữ liệu thị trường mới nhất để đưa ra quyết định. Nếu sử dụng LLM, nó sẽ được fine-tune để nhận bối cảnh này và đưa ra quyết định.
    - **Explain Agent (LLM-powered):**
        - Sử dụng một mô hình 7B (như **Mistral 7B** hoặc **Llama 3 8B**) đã được fine-tune bằng **Unsloth** để tối ưu hiệu suất.
        - Mô hình này được fine-tune trên một bộ dữ liệu nhỏ để học cách "dịch" các "sự thật thô" (raw facts) từ Decision Maker thành lời giải thích tự nhiên, tuân thủ giọng văn của hệ thống.
        - Hoạt động theo cơ chế RAG, lấy ngữ cảnh từ Knowledge Base để tăng độ chính xác.
    - **Evo Agent (Central Lab Model):**
        - Sử dụng thư viện **PyGAD** hoặc một thuật toán Genetic Algorithm (GA) tự triển khai đơn giản.
        - Thực hiện quá trình tiến hóa trên server.

        - **Fitness Function** sẽ được tùy chỉnh động dựa trên userProfile nhận được từ client.
        - Tương tác với Simulation Module để đánh giá các chiến lược.
    - **Simulation Module (Backtester):**
        - Được xây dựng từ đầu bằng **Pandas/NumPy** để đảm bảo sự linh hoạt và hiểu sâu về logic.
        - Nhận một bộ quy tắc và một khoảng thời gian lịch sử, trả về các chỉ số hiệu suất (lợi nhuận, max drawdown, Sharpe ratio...).

4.3. Data Persistence Layer (Lớp Lưu trữ Dữ liệu)

- **Hệ quản trị CSDL: PostgreSQL**.
    - *Lý do:* Mạnh mẽ, ổn định, đáng tin cậy. Hỗ trợ tốt kiểu dữ liệu JSONB (để lưu các cấu hình linh hoạt) và có các extension mạnh mẽ cho dữ liệu chuỗi thời gian như TimescaleDB (nếu cần mở rộng).
- **Cấu trúc Bảng (Schema) chính:**
    - stocks: Thông tin cơ bản về các mã cổ phiếu hệ thống theo dõi.
    - daily_prices: Lưu dữ liệu OHLCV hàng ngày.
    - news_articles: Lưu trữ tiêu đề tin tức và các nhãn đã được phân tích.
    - global_rules: Lưu trữ "Bộ nhớ Quy tắc Chung" cho Evo Agent.

4.4. Data Ingestion Pipeline (Luồng Thu thập Dữ liệu)

- **Công nghệ:** Các script **Python** độc lập, sử dụng thư viện requests và yfinance.
- **Lập lịch:** Sử dụng APScheduler trong Python hoặc cron trên hệ điều hành để chạy tự động theo lịch (ví dụ: mỗi ngày một lần sau khi thị trường đóng cửa).
- **Nhiệm vụ:**
    1. Gọi API từ Yahoo Finance và các nguồn tin tức.
    2. Làm sạch và chuẩn hóa dữ liệu.
    3. Lưu dữ liệu mới vào các bảng tương ứng trong PostgreSQL.

5. PHÂN TÍCH LUỒNG DỮ LIỆU CHÍNH (KEY DATA FLOWS)

5.1. Luồng Phân tích Cổ phiếu (Decision & Explanation Flow)

1. **Client:** Người dùng yêu cầu phân tích cổ phiếu "AAPL". Frontend tạo một JSON context_package chứa {"ticker": "AAPL", "userProfile": {...}, "active_ruleset": [...]}.
2. **API Call:** Client gửi POST /api/decision/analyze với context_package trong body.
3. **Backend:**
   a. **API Layer** nhận và xác thực request.
   b. **Advisor Module** được gọi. Nó yêu cầu dữ liệu mới nhất cho AAPL từ **Technical Module** và **News Module**.
   c. **Decision Maker Agent** áp dụng active_ruleset vào dữ liệu để tạo ra "quyết định thô" và "lý do thô".
   d. **Explain Agent (LLM)** nhận "quyết định thô" và "lý do thô", sau đó "dịch" chúng

thành một lời giải thích tự nhiên.
e. **API Layer** đóng gói kết quả (quyết định + giải thích) và trả về cho client.

4. **Client:** Nhận JSON response và hiển thị kết quả cho người dùng trong chatbot/dashboard.

5.2. Luồng Tối ưu hóa Chiến lược (Evolution Flow)

1. **Client:** Người dùng nhấn "Tối ưu hóa". Frontend tạo evolution_package chứa {"userProfile": {...}, "localRules": [...]}.
2. **API Call:** Client gửi POST /api/evo/personalize_evolution với evolution_package.
3. **Backend:**
   a. **Evo Agent** nhận yêu cầu và khởi tạo một phiên tiến hóa.
   b. Nó kết hợp các quy tắc từ **Bộ nhớ Quy tắc Chung** và localRules để tạo quần thể ban đầu.
   c. Nó tùy chỉnh **Fitness Function** dựa trên userProfile.
   d. Trong vòng lặp tiến hóa, nó liên tục gọi **Simulation Module** để đánh giá từng chiến lược.
   e. Sau khi hoàn tất, nó chọn ra các quy tắc tốt nhất.
4. **Client:** Nhận về bộ quy tắc mới đã được cá nhân hóa và gợi ý cho người dùng cập nhật.

6. TÓM TẮT CÔNG NGHỆ (TECHNOLOGY STACK SUMMARY)

Lớp	Công nghệ	Lý do chính
Frontend	React.js, TradingView Lightweight Charts	Hệ sinh thái mạnh, UI linh hoạt, biểu đồ chuyên nghiệp
Backend	Python, FastAPI, Pydantic	Hiệu suất cao, hệ sinh thái AI mạnh mẽ, xác thực dữ liệu
Database	PostgreSQL	Ổn định, mạnh mẽ, hỗ trợ tốt JSONB và chuỗi thời gian
AI/ML/NLP	Pandas, TA-Lib, Transformers, Unsloth, PyGAD	Tiêu chuẩn ngành, hiệu quả, tối ưu hóa cho AI
Deployment	Docker, Nginx	Đóng gói, dễ triển khai, quản lý request

7. KẾT LUẬN

Kiến trúc và các công nghệ được lựa chọn được thiết kế để tạo ra một hệ thống mạnh mẽ, linh hoạt, và quan trọng nhất là khả thi để triển khai trong khuôn khổ một đồ án tốt nghiệp. Thiết kế này trực tiếp hỗ trợ các mục tiêu cốt lõi của dự án như tính giải thích, bảo mật, chi phí thấp và khả năng học hỏi, đặt nền móng vững chắc cho việc phát triển và mở rộng trong tương lai.