
Tài liệu Định hướng Kỹ thuật Dự án ITAPIA (v1.1)

1. Mục đích

Tài liệu này đóng vai trò là bản thiết kế kỹ thuật (technical blueprint) cho việc triển khai các thành phần xử lý dữ liệu và trí tuệ nhân tạo của hệ thống ITAPIA. Nó định nghĩa rõ ràng vai trò, đầu vào, đầu ra, và sự tương tác giữa các module, làm cơ sở cho việc phát triển và tích hợp.

2. Nguyên tắc Kiến trúc Cốt lõi

- **Tách biệt Backend:** Hệ thống backend được chia thành 2 service chính:
    - **api_service (CPU-based):** Đóng vai trò API Gateway, điều phối luồng dữ liệu và xử lý logic nghiệp vụ nhẹ. Đây là "bộ não" điều khiển.
    - **ai_service (GPU-based):** Đóng vai trò "nhà máy AI", chỉ chịu trách nhiệm thực thi các mô hình tính toán nặng.
- **"UTC Everywhere":** Toàn bộ backend và CSDL sẽ xử lý và lưu trữ thời gian dưới dạng UTC (TIMESTAMP WITH TIME ZONE) để đảm bảo tính nhất quán.
- **Giao tiếp Phi trạng thái (Stateless Communication):** api_service và ai_service giao tiếp qua các API nội bộ. Mọi yêu cầu từ api_service đến ai_service đều chứa đầy đủ dữ liệu cần thiết cho việc xử lý.
- **Bảo mật Dữ liệu:** Các service AI không được phép truy cập CSDL trực tiếp.

---

3. Chi tiết các Thành phần và Luồng Dữ liệu

3.1. Data Pipeline (data_processing)

Đây là thành phần độc lập, chịu trách nhiệm cung cấp "nhiên liệu" cho toàn bộ hệ thống. Nó được điều khiển bởi một công cụ lập lịch bên ngoài (ví dụ: cron hoặc Airflow).

A. Pipeline Dữ liệu Lịch sử (Batch - fetch_eod_data.py)

- **Mục đích:** Cập nhật dữ liệu giá cuối ngày (EOD) đã được xác nhận.
- **Kích hoạt:** Chạy theo lịch trình cho từng khu vực sau khi thị trường của khu vực đó đóng cửa (ví dụ: americas chạy lúc 21:00 UTC).
- **Đầu vào:** Tham số region từ dòng lệnh (americas, europe, asia_pacific).
- **Tương tác:**

1. **Đọc từ CSDL:** Truy vấn bảng daily_prices trong PostgreSQL để tìm ngày cuối cùng đã được lưu cho khu vực đó.
2. **Gọi API ngoài:** Sử dụng yf.Tickers(...).history() để lấy toàn bộ dữ liệu từ ngày cuối cùng đó đến ngày hiện tại.
3. **Ghi vào CSDL:** Sử dụng hàm db_utils.bulk_upsert_with_temp_table để thực hiện UPSERT dữ liệu mới vào bảng daily_prices.

- **Đầu ra:** Bảng daily_prices trong PostgreSQL được cập nhật với dữ liệu mới nhất.

B. Pipeline Dữ liệu Gần Thời gian thực (Real-time - process_realtime_data.py)

- **Mục đích:** Cung cấp cây nến ngày "tạm thời" dựa trên diễn biến trong ngày.
- **Kích hoạt:** Chạy trong một service Docker liên tục (realtime_processor), với logic lặp vô hạn bên trong, chỉ hoạt động khi thị trường của khu vực tương ứng mở cửa.
- **Đầu vào:** Tham số region từ dòng lệnh.
- **Tương tác:**
    1. **Gọi API ngoài:** Mỗi 5 phút, lặp qua các ticker trong khu vực và gọi yf.Ticker(...).fast_info để lấy các giá trị open, day_high, day_low, last_price, volume.
    2. **Ghi vào Cache:** Tổng hợp các giá trị trên thành một cây nến ngày tạm thời và ghi (HSET) vào **Redis**. Key có dạng provisional_daily:{ticker} (ví dụ: provisional_daily:AAPL).
    3. **Tự động hủy:** Đặt TTL (Time To Live) cho mỗi key là 24 giờ.
- **Đầu ra:** Dữ liệu cây nến ngày tạm thời được cập nhật liên tục trong Redis.

C. Pipeline Dữ liệu Tin tức (Batch - fetch_news.py)

- **Mục đích:** Thu thập và lưu trữ tin tức liên quan đến các cổ phiếu.
- **Kích hoạt:** Chạy theo lịch trình, thường xuyên hơn pipeline giá (ví dụ: mỗi 30 phút).
- **Đầu vào:** Tham số region.
- **Tương tác:**
    1. Gọi yf.Ticker(...).get_news() để lấy N tin tức mới nhất cho mỗi cổ phiếu.
    2. Đối với mỗi tin tức, chuẩn bị một dictionary có cấu trúc khớp với bảng relevant_news.

3. Sử dụng hàm db_utils.bulk_insert_on_conflict_do_nothing để chèn tin tức vào bảng relevant_news trong PostgreSQL, tự động bỏ qua nếu uuid đã tồn tại.

- **Đầu ra:** Bảng relevant_news được cập nhật với các tin tức mới.

3.2. Dịch vụ AI (ai_service)

Đây là "nhà máy" chạy trên GPU, chỉ nhận dữ liệu và trả về kết quả tính toán.

A. Technical Module

Thay vì là một khối đơn lẻ, Technical Module sẽ được tổ chức như một "package" hoặc một thư mục chứa các sub-module chuyên biệt:

1. Sub-module: Analysis (Phân tích & Thống kê)

- **Vai trò cốt lõi:** Cung cấp một cái nhìn **toàn diện về tình trạng hiện tại và quá khứ** của một tài sản dựa trên dữ liệu giá lịch sử. Nó trả lời câu hỏi: **"Điều gì đang xảy ra và đã xảy ra?"**
- **Chức năng chính:**
    1. **Tính toán Chỉ báo Kỹ thuật (Indicator Calculation):**
        - Đây là chức năng "Calculate On-the-fly" mà chúng ta đã thảo luận.
        - Cung cấp một hàm nhận đầu vào là DataFrame OHLCV và một danh sách các chỉ báo cần tính (ví dụ: {'indicator': 'SMA', 'period': 20}), trả về DataFrame đã được làm giàu.
    2. **Phát hiện Mẫu hình Giá (Pattern Recognition - Nâng cao):**
        - Nhận diện các mẫu hình giá kinh điển như "Vai-đầu-vai", "Hai đỉnh/đáy", "Cờ", "Tam giác".
        - Có thể được triển khai bằng các thuật toán so khớp chuỗi hoặc các mô hình ML đơn giản.
    3. **Xác định Hỗ trợ & Kháng cự (Support & Resistance Identification):**
        - Xác định các vùng giá quan trọng nơi giá có xu hướng phản ứng. Có thể dựa trên các điểm xoay (pivot points), các vùng tập trung khối lượng (volume profile), hoặc các mức Fibonacci.
    4. **Phân tích Xu hướng (Trend Analysis):**
        - Xác định xu hướng chính (tăng, giảm, đi ngang) dựa trên các đường MA, chỉ số ADX, hoặc phân tích các đỉnh/đáy.

- **Đầu ra:** Một cấu trúc dữ liệu (ví dụ: JSON hoặc dict) chứa đựng một "bản báo cáo tình trạng kỹ thuật" của cổ phiếu.

Generated json

```
    {
  "indicators": {"SMA_20": 150.5, "RSI_14": 55.8, ...},
  "patterns": ["Rising Wedge Detected"],
  "support_resistance": {"support": [145.0, 142.5], "resistance": [155.0]},
  "trend": {"primary": "Uptrend", "strength": "Weakening"}
}
```

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Json

IGNORE_WHEN_COPYING_END

2. Sub-module: ForecastingModel (Mô hình Dự đoán)

- **Vai trò cốt lõi:** Cố gắng đưa ra **dự báo về diễn biến giá trong tương lai gần**. Nó trả lời câu hỏi: **"Điều gì có thể xảy ra tiếp theo?"**
- **Quan trọng:** Cần phải hiểu rằng việc dự đoán giá chính xác là cực kỳ khó. Kết quả từ module này nên được xem là một **tín hiệu xác suất**, một yếu tố đầu vào bổ sung cho Decision Maker Agent, chứ không phải là một "lời tiên tri".
- **Các công nghệ có thể áp dụng:**
    1. **Mô hình Thống kê Chuỗi thời gian (Time Series Models):**
        - **ARIMA/SARIMA:** Các mô hình kinh điển, tốt cho các chuỗi thời gian có tính mùa vụ và xu hướng rõ ràng. Có thể không hiệu quả với thị trường chứng khoán vốn rất "nhiễu".
        - **Prophet:** Thư viện của Facebook, dễ sử dụng, xử lý tốt dữ liệu có tính mùa vụ và ngày nghỉ.
    2. **Machine Learning (ML) cho Hồi quy (Regression):**
        - Sử dụng các giá trị quá khứ và các chỉ báo kỹ thuật làm đặc trưng để dự đoán giá đóng cửa của N ngày tiếp theo.

        - **Mô hình:** Linear Regression, Random Forest Regressor, Gradient Boosting (XGBoost).
    3. **Deep Learning (DL) cho Chuỗi thời gian:**
        - Đây là hướng tiếp cận mạnh mẽ nhất cho dữ liệu tuần tự.
        - **Mô hình:**
            - **LSTM (Long Short-Term Memory):** "Ông vua" một thời của các bài toán chuỗi thời gian, có khả năng ghi nhớ các phụ thuộc dài hạn.
            - **GRU (Gated Recurrent Unit):** Một phiên bản đơn giản hơn của LSTM, đôi khi hiệu quả tương đương.
            - **Transformers (ví dụ: Informer, Autoformer - Nâng cao):** Các kiến trúc mới hơn, có khả năng xử lý các chuỗi dài hơn và các mối quan hệ phức tạp hơn, nhưng cũng khó triển khai hơn.
- **Đầu ra:** Một dự báo, có thể bao gồm:

Generated json

```
    {
  "forecast_horizon": "5 days",
  "predicted_price_range": [152.0, 158.0],
  "predicted_direction": "Up",
  "confidence_score": 0.65
}
```

Sự Tương tác giữa các Module

Với cấu trúc này, luồng làm việc của hệ thống trở nên rất rõ ràng:

1. **Decision Maker Agent** cần đưa ra quyết định cho cổ phiếu XYZ.
2. Nó gửi một yêu cầu đến **Technical Module**.
3. Bên trong Technical Module:
    - Nó gọi đến **Analysis** để có được "bức tranh toàn cảnh" về tình hình hiện tại.
    - Nó gọi đến **ForecastingModel** để có được một "cái nhìn về tương lai".

4. **Decision Maker Agent** nhận kết quả từ cả hai sub-module này, kết hợp chúng với thông tin từ **News Module** và hồ sơ người dùng để đưa ra quyết định cuối cùng.

Cách cấu trúc này không chỉ giúp bạn dễ phát triển mà còn làm cho báo cáo đồ án của bạn trở nên chuyên nghiệp và có hệ thống hơn rất nhiều.

B. News Analysis Module

- **Đầu vào:** Một danh sách các đối tượng tin tức (mỗi đối tượng chứa title, summary).
- **Tương tác:**
    1. Lặp qua từng tin tức.
    2. Gọi đến **LLM Inference Service** (phiên bản FinGPT gốc) với một prompt được thiết kế để yêu cầu gán nhãn direction và level.
    3. Tổng hợp kết quả của tất cả các tin tức (ví dụ: lấy trung bình có trọng số dựa trên level và độ mới của tin) để tạo ra một kết quả cuối cùng.
- **Đầu ra:** Một đối tượng JSON chứa kết quả phân tích tổng hợp, ví dụ: {"sentiment_score": 0.65, "impact_level": "Medium"}.

C. Advisor Module (Phần AI)

- **Explain Agent (LLM Fine-tuned):**
    - **Đầu vào:** Một đối tượng "sự thật thô" (raw facts) từ Decision Maker, bao gồm quyết định, các quy tắc được kích hoạt, các chỉ số quan trọng, và **ngữ cảnh từ Knowledge Base** (đọc từ file, ví dụ: định nghĩa "Giao cắt vàng là gì?").
    - **Tương tác:** Gọi đến **LLM Inference Service** (phiên bản đã được fine-tune cho việc giải thích).
    - **Đầu ra:** Một chuỗi văn bản tự nhiên, dễ hiểu.
- **Decision Maker:** Sử dụng Rule-based hoặc LLM
    - **Đầu vào:** Toàn bộ EvaluationContext (bao gồm kết quả từ Technical, News, và Profile người dùng).
    - **Tương tác:** Gọi đến **Rule-based Agent** hoặc **LLM Inference Service** (phiên bản đã được fine-tune cho việc ra quyết định), có thể được cung cấp thêm ngữ cảnh RAG từ các file chiến lược chung.
    - **Đầu ra:** Một đối tượng JSON quyết định có cấu trúc (action, confidence, rationale).

- **Risk Management:** Sử dụng Rule-based để dựa vào các thông số hiện tại để đo lường rủi ro.
- **Opportunity Finding Agent:** Sử dụng rule-based và các tính toán đơn giản để ước lượng các cổ phiếu tiềm năng với yêu cầu của người sử dụng (có thể xem xét tích hợp LLM sau vì cần tổng hợp nhiều thông tin nhưng k ưu tiên trong giai đoạn hiện tại để giảm độ phức tạp).
- **Leader:** Thành phần điều phối đơn giản, nó phân tích request từ chatbot interface do người dùng nhập và xác định các thành phần AI tham gia, điều phối, phối hợp chúng và tổng hợp kết quả cuối cùng.
- **Các kĩ thuật có thể được sử dụng:**
    - Distillation (chưng cất tri thức): Sử dụng các mô hình LLM cỡ lớn như Gemini 2.5 Pro hay Deepseek (đều có miễn phí 1 phần) để tạo ra dữ liệu fine-tuned cần thiết cho các module AI sử dụng LLM.
    - RAG: Dùng RAG để lưu trữ bộ nhớ ngoài chứa các tài liệu tài chính và đầu tư phục vụ cho việc giảm ảo giác trong các model nhỏ hơn như FinGPT_mistral (optional)

D. Evolution Module (Evo Agent)

- **Đầu vào:** Một "gói tiến hóa" từ api_service chứa userProfile và localRules (dạng AST JSON).
- **Tương tác:**
    1. Gọi Simulation Module hàng ngàn lần để chạy backtest. Simulation Module sẽ nhận dữ liệu lịch sử (đã được cache trong RAM của ai_service) và một cây AST quy tắc.
    2. (Tùy chọn) Gọi LLM Inference Service (vai trò "Reflector") để đánh giá định tính các chiến lược ứng viên tốt nhất.
- **Đầu ra:** Một danh sách các cây AST JSON đã được tối ưu hóa.
- **Lưu ý:** Mỗi rule luôn bao gồm 2 phần: template và constant (threshold). Template là cấu trúc, threshold là những ngưỡng hằng số (chứ không phải parameter cho rule được truyền vào). Vì vậy có thể có 2 quần thể tiến hóa hỗ trợ nhau: template là cần tối ưu cấu trúc, có thể dùng GP, còn threshold về bản chất là khi có kiến trúc rồi thì chọn các hằng số ngưỡng là bao nhiêu (việc chọn hằng số này còn có thể phân ra theo nhóm các cổ phiếu liên quan, vì các cổ phiếu thuộc các nhóm khác nhau sẽ có các “ngưỡng” khác nhau) là bài toán tối ưu số thực, có thể dùng các thuật toán Evo mạnh về số thực như DE, CMA-ES.

---

3. Dịch vụ API (api_service)

Đây là "nhạc trưởng" điều phối. Luồng phân tích một cổ phiếu sẽ như sau:

1. **Đầu vào:** Nhận request từ Frontend: POST /analyze với body chứa {"ticker": "AAPL", "userProfile": {...}, "active_ruleset": [...]}.
2. **Tương tác (Thu thập Dữ liệu):**
   a. Truy vấn **PostgreSQL** (daily_prices) để lấy dữ liệu lịch sử.
   b. Truy vấn **Redis** (provisional_daily:AAPL) để lấy cây nến ngày hiện tại.
   c. **Ghép nối** hai nguồn dữ liệu trên thành một DataFrame hoàn chỉnh.
   d. Truy vấn **PostgreSQL** (relevant_news) để lấy 10 tin tức gần nhất của AAPL.
3. **Tương tác (Gọi AI):**
   a. Tạo một payload lớn chứa (DataFrame giá, danh sách tin tức).
   b. Gọi đến ai_service qua API nội bộ: POST http://ai_service:8001/get_full_analysis.
4. **Tương tác (Xử lý Kết quả):**
   a. Nhận lại kết quả phân tích AI có cấu trúc từ ai_service.
   b. Xây dựng đối tượng EvaluationContext.
   c. Thực thi các quy tắc trong active_ruleset (dạng AST) trên EvaluationContext để ra quyết định cuối cùng.
5. **Đầu ra:** Gửi một JSON response hoàn chỉnh về cho Frontend.

Đây là "nhà máy" chạy trên GPU, được triển khai như một service Docker riêng biệt (itapia_ai_service). Nó **KHÔNG** truy cập CSDL trực tiếp và chỉ giao tiếp với thế giới bên ngoài thông qua các API nội bộ được api_service gọi đến.

- **Quản lý Mô hình:**
  - Service sẽ quản lý nhiều phiên bản mô hình LLM đã được fine-tune cho các tác vụ khác nhau (Explain, Decide, News Analysis).
  - Áp dụng kỹ thuật **Lazy Loading** và **LoRA Adapter Swapping**:
    1. Base model (ví dụ: FinGPT/Mistral-7B) được tải vào VRAM một lần và giữ lại.
    2. Khi có yêu cầu cho một tác vụ cụ thể (ví dụ: /explain), service sẽ "gắn" (attach) LoRA adapter tương ứng vào base model để thực thi. Việc này giúp chuyển đổi giữa các tác vụ gần như tức thì và tiết kiệm VRAM.
- **Tối ưu hóa Hiệu suất:** Áp dụng kỹ thuật **Dynamic Batching**. Các request đến sẽ được đưa vào hàng đợi và gom lại thành một batch để xử lý song song trên GPU, giúp tăng thông lượng hệ thống.
- **Các API Nội bộ chính:**
  - POST /analyze/technical: Nhận một chuỗi thời gian, trả về kết quả phân tích kỹ thuật và dự báo.

- POST /analyze/news: Nhận một danh sách tin tức, trả về phân tích cảm xúc tổng hợp.
- POST /explain: Nhận một tập hợp "sự thật thô", trả về một lời giải thích tự nhiên.
- POST /decide: (Nâng cao) Nhận EvaluationContext, trả về một đối tượng quyết định.
- POST /evolve: Nhận userProfile và localRules, chạy Evo Agent và trả về các quy tắc đã được tối ưu hóa.

3.3. Dịch vụ API (api_service)

Đây là "nhạc trưởng" điều phối, chạy trên một container Docker chỉ cần CPU (itapia_api_service). Nó là điểm vào duy nhất cho các yêu cầu từ Frontend.

A. Quản lý Kết nối và Vòng đời

- **Cơ sở dữ liệu (PostgreSQL & Redis):**
  - api_service sẽ khởi tạo một **đối tượng Engine (SQLAlchemy) duy nhất** và một **đối tượng Redis Client duy nhất** khi ứng dụng khởi động.
  - Các đối tượng này đã tích hợp sẵn **Connection Pooling** để quản lý kết nối một cách hiệu quả. Không cần tự xây dựng pool thủ công.
- **Dependency Injection với FastAPI:**
  - Sử dụng cơ chế Depends và các hàm generator (yield) để "tiêm" các session CSDL và kết nối Redis vào từng request API.
  - Cơ chế này đảm bảo mỗi request được xử lý trong một session độc lập và tài nguyên kết nối luôn được giải phóng một cách an toàn sau khi request hoàn tất, ngay cả khi có lỗi.

B. Cấu trúc Code (Theo kiến trúc gợi ý)

- **app/db/session.py:** Nơi duy nhất chịu trách nhiệm tạo Engine, SessionLocal, và redis_client. Cung cấp các dependency get_db() và get_redis().
- **app/schemas/:** Chứa các class Pydantic để định nghĩa hình dạng dữ liệu cho request và response, giúp tự động validation và serialize.
- **app/crud/:** Chứa các hàm logic truy cập dữ liệu thuần túy (chỉ SELECT). Ví dụ: crud_prices.get_history_prices, crud_news.get_news. Các hàm này sẽ nhận session DB làm tham số.
- **app/api/v1/endpoints/:** Chứa các file định nghĩa router API. Các endpoint sẽ nhận request, gọi đến các hàm trong crud để lấy dữ liệu, sau đó gọi đến ai_service nếu cần, và trả về response.

C. Luồng Phân tích một Cổ phiếu (Chi tiết)

Đây là luồng làm việc chính, thể hiện sự tương tác giữa các thành phần:

1. **Đầu vào:** api_service nhận request từ Frontend: POST /api/v1/analyze/{ticker} với body chứa {"userProfile": {...}, "active_ruleset": [...]}.

2. **Tương tác (Thu thập Dữ liệu):**
   a. Gọi hàm crud.crud_prices.get_history_prices() để truy vấn **PostgreSQL** và lấy N ngày dữ liệu lịch sử gần nhất.
   b. Gọi hàm crud.crud_prices.get_intraday_price() để truy vấn **Redis** và lấy cây nến ngày tạm thời.
   c. **Ghép nối** hai nguồn dữ liệu trên bằng Pandas để thành một DataFrame lịch sử hoàn chỉnh và cập nhật nhất.
   d. Gọi hàm crud.crud_news.get_news() để truy vấn **PostgreSQL** và lấy 10 tin tức gần nhất.
3. **Tương tác (Gọi AI):**
   a. api_service tạo một payload lớn chứa DataFrame giá đã được chuẩn bị và danh sách các đối tượng tin tức.
   b. Nó thực hiện một lời gọi API nội bộ đến ai_service qua mạng Docker: POST http://ai_service:8001/get_full_analysis với payload trên. *(ai_service là tên của service trong docker-compose.yml)*.
4. **Tương tác (Xử lý Kết quả AI):**
   a. api_service nhận lại một đối tượng JSON lớn từ ai_service, chứa kết quả đã được tính toán từ các module Technical và News. Ví dụ:
   json { "technicals": { "analysis": {...}, "forecasts": {...} }, "news": { "sentiment_score": 0.65, ... } }
   b. Nó xây dựng một đối tượng EvaluationContext hoàn chỉnh bằng cách kết hợp kết quả AI này với userProfile từ request ban đầu.
   c. Nó thực thi các quy tắc trong active_ruleset (dạng AST) trên EvaluationContext để ra quyết định cuối cùng (ví dụ: BUY, SELL, HOLD). Đây là lúc logic **Rule-based** được áp dụng.
5. **Tương tác (Tạo lời giải thích - RAG):**
   a. Sau khi có quyết định, api_service sẽ chuẩn bị một "gói sự thật" (raw facts) bao gồm: quyết định cuối cùng, quy tắc AST đã được kích hoạt, và các chỉ số quan trọng đã dẫn đến quyết định đó.
   b. (Nâng cao) Nó có thể đọc thêm ngữ cảnh từ các file kiến thức tài chính (ví dụ: "Giao cắt vàng là một tín hiệu mua...") để làm giàu thêm cho "gói sự thật".
   c. Nó gửi "gói sự thật" này đến một endpoint khác của ai_service: POST http://ai_service:8001/explain.
   d. ai_service sử dụng Explain Agent (LLM đã được fine-tune) để tạo ra một lời giải thích tự nhiên.
6. **Đầu ra:** api_service tổng hợp tất cả thông tin (quyết định, lời giải thích, dữ liệu gốc) vào một JSON response cuối cùng và gửi về cho Frontend.