BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC PHENIKAA

BÁO CÁO BÀI TẬP LỚN
HỌC PHẦN: KĨ THUẬT PHẦN MỀM

Thiết kế nền tảng dữ liệu hiện đại hỗ trợ xử lý và phân tích dữ liệu lớn

Sinh viên	MSSV	Khoá	Ngành	Hệ
Đặng Minh Trí	**22010157**	K16	KHMT	Chính quy
Nguyễn Minh Ánh	**23010718**	K17	CNTT	Chính quy

Giảng viên hướng dẫn: TS.Trương Anh Hoàng

Hà Nội, June 8, 2025

Mục lục

Chương 1

Giới thiệu

1.1 Bối cảnh và động lực của dự án

Trong bối cảnh chuyển đổi số ngày càng mạnh mẽ, dữ liệu đã trở thành tài sản chiến lược quan trọng của mọi tổ chức. Tuy nhiên, nhiều doanh nghiệp vừa và nhỏ vẫn gặp khó khăn trong việc thu thập, lưu trữ, xử lý và khai thác hiệu quả nguồn dữ liệu mà họ đang sở hữu. Việc dữ liệu bị phân tán, thiếu chuẩn hóa, khó truy cập hoặc không được kiểm soát rõ ràng là những rào cản lớn trong hành trình khai thác dữ liệu phục vụ phân tích và ra quyết định.

Dự án Data Platform ra đời nhằm xây dựng một nền tảng dữ liệu tập trung, giúp chuẩn hóa quy trình thu thập, lưu trữ, biến đổi và khai thác dữ liệu. Hệ thống này hướng tới việc xử lý dữ liệu theo hướng hiện đại với các đặc tính: linh hoạt, mở rộng dễ dàng, dễ bảo trì, đảm bảo tính toàn vẹn và bảo mật dữ liệu.

Với sự kết hợp của các công nghệ mã nguồn mở như MinIO (data lake), PostgreSQL (data warehouse), Apache Airflow (ETL orchestration), và DBT (data transformation) — nền tảng này không chỉ giúp tiết kiệm chi phí triển khai mà còn đảm bảo khả năng mở rộng trong tương lai, phù hợp với chiến lược phát triển dữ liệu lâu dài của doanh nghiệp.

Động lực của dự án không chỉ nằm ở nhu cầu kỹ thuật mà còn ở mục tiêu tạo ra một hệ thống dữ liệu hiện đại phục vụ trực tiếp cho các tác vụ phân tích, xây dựng dashboard BI, và triển khai các mô hình học máy (machine learning). Dự án giúp nhóm phát triển áp dụng kiến thức phần mềm vào bài toán thực tế, đồng thời rèn luyện tư duy hệ thống, kỹ năng teamwork và khả năng tích hợp công nghệ.

1.2 Mục tiêu của hệ thống

Mục tiêu chính của hệ thống là xây dựng một nền tảng dữ liệu tập trung (Data Platform) có khả năng thu thập, lưu trữ, xử lý, chuyển đổi và cung cấp dữ liệu phục vụ cho các nhu cầu phân tích, báo cáo và triển khai các mô hình học máy trong tương lai. Cụ thể, hệ thống hướng đến việc đạt được các mục tiêu sau:

- Tập trung hóa dữ liệu từ nhiều nguồn: Cho phép dễ dàng tích hợp dữ liệu từ nhiều hệ thống khác nhau (API, file CSV, cơ sở dữ liệu quan hệ, v.v) và lưu trữ tại một nơi duy nhất (data lake).

- Lưu trữ dữ liệu linh hoạt và an toàn: Sử dụng MinIO như một giải pháp thay thế S3 để lưu trữ dữ liệu dạng thô (raw data) và đã xử lý (clean data), đảm bảo tính bền vững và khả năng mở rộng.
- Xử lý và chuyển đổi dữ liệu có kiểm soát: Sử dụng Apache Airflow để điều phối luồng dữ liệu ETL/ELT, kết hợp với DBT để thực hiện các bước transform theo cách có thể kiểm tra, kiểm soát phiên bản và tái sử dụng.
- Tổ chức dữ liệu phục vụ phân tích và báo cáo: Dữ liệu sau xử lý được lưu trữ trong PostgreSQL như một kho dữ liệu (data warehouse), phục vụ trực tiếp cho việc khai thác thông tin bằng công cụ BI hoặc query trực tiếp.
- Đảm bảo khả năng mở rộng và dễ bảo trì: Thiết kế hệ thống theo hướng module hóa và triển khai bằng Docker Compose để dễ dàng tái sử dụng, nâng cấp và vận hành linh hoạt trên nhiều môi trường.
- Phục vụ mô hình học máy (ML) trong tương lai: Chuẩn bị hạ tầng dữ liệu sạch và đáng tin cậy để phục vụ cho việc huấn luyện mô hình AI/ML trên dữ liệu lịch sử.

Thông qua hệ thống này, người dùng có thể giảm thiểu thời gian làm sạch và tích hợp dữ liệu, tăng tốc độ truy cập và phân tích, đồng thời thiết lập một quy trình quản lý dữ liệu bài bản và có thể mở rộng.

1.3 Phạm vi thực hiện

Phạm vi dự án bao gồm việc xây dựng một nền tảng dữ liệu hiện đại (Data Platform) kết hợp với triển khai một mô hình trí tuệ nhân tạo (AI model), cung cấp API dự đoán và giao diện người dùng đơn giản phục vụ cho việc trình diễn, kiểm thử và ứng dụng thực tế. Cụ thể, phạm vi được chia thành các phần sau:

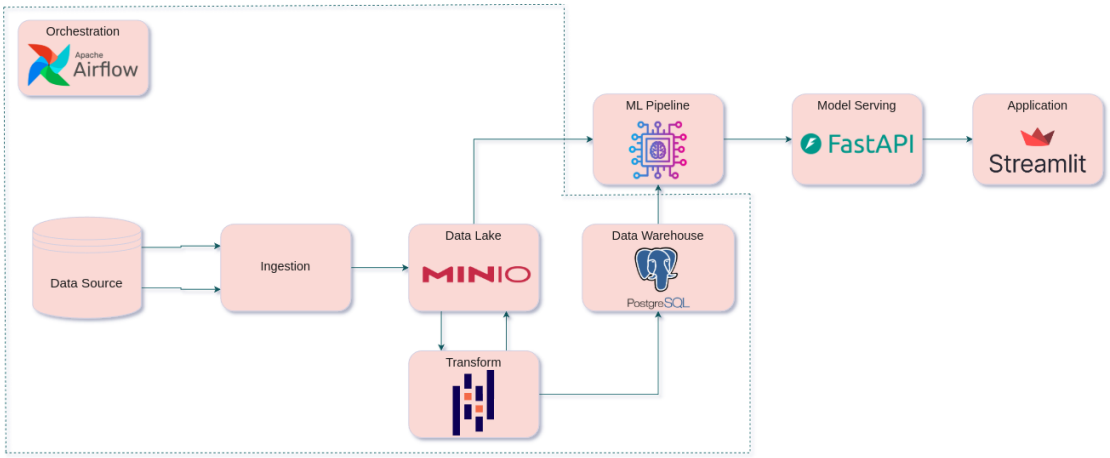
- Thu thập và lưu trữ dữ liệu
  - Thiết lập hệ thống lưu trữ dạng data lake sử dụng MinIO để lưu trữ dữ liệu đầu vào dưới dạng thô (raw data).
  - Hỗ trợ nhập dữ liệu, tự động hóa việc lấy dữ liệu định kỳ bằng Apache Airflow.
- Xử lý và chuyển đổi dữ liệu
  - Extract dữ liệu từ MinIO
  - Load dữ liệu vào PostgreSQL
  - Transform dữ liệu sử dụng Pandas để chuẩn hóa và tạo ra các bảng phân tích.
- Lưu trữ dữ liệu phân tích
  - Sử dụng PostgreSQL làm data warehouse để lưu trữ dữ liệu đã xử lý, phục vụ cho việc phân tích, trực quan hóa hoặc truy vấn theo thời gian thực.

- Tự động hóa và điều phối
  - Thiết lập quy trình tự động hóa với Apache Airflow để điều phối các tác vụ ETL, kiểm soát lỗi và log hoạt động.
  - Tất cả thành phần được đóng gói bằng Docker và điều phối thông qua Docker Compose để dễ dàng triển khai và bảo trì.
- Triển khai mô hình AI
  - Lưu mô hình đã huấn luyện và triển khai dự đoán qua RESTful API sử dụng FastAPI
- Tạo giao diện người dùng với Streamlit
  - Tải dữ liệu đầu vào
  - Gửi dữ liệu đến API để dự đoán
  - Hiển thị kết quả trực quan và dễ sử dụng với người không chuyên
- Bảo mật và phân quyền (cơ bản)
  - Cấu hình truy cập phân quyền mức cơ bản đối với MinIO và PostgreSQL để đảm bảo dữ liệu không bị truy cập trái phép trong môi trường phát triển.
- Triển khai hệ thống
  - Tất cả thành phần được đóng gói và chạy bằng Docker Compose.

1.4 Phương pháp tiếp cận

Dự án được tiếp cận theo hướng tích hợp từng thành phần độc lập trong một hệ sinh thái dữ liệu tổng thể, giúp đảm bảo khả năng mở rộng, dễ bảo trì và có thể triển khai nhanh trong môi trường phát triển. Phương pháp tiếp cận chính bao gồm các bước sau:

- Thiết kế kiến trúc tổng thể
  - Lập sơ đồ kiến trúc cho toàn bộ hệ thống, bao gồm các thành phần: ingestion, processing, storage, AI model, API, và giao diện.



Hình 1.1: Sơ đồ kiến trúc

  - Mỗi thành phần được đóng gói và triển khai độc lập bằng Docker Compose, cho phép dễ dàng chạy trên môi trường local hoặc cloud.

- Thu thập và xử lý dữ liệu
  - Sử dụng Apache Airflow để xây dựng pipeline tự động thu thập dữ liệu từ các nguồn như file CSV, API, v.v.
  - Dữ liệu được lưu trữ dưới dạng raw data trong MinIO (object storage).
  - Dữ liệu sau đó được xử lý bằng Pandas và lưu vào trong PostgreSQL (data warehouse)
- Triển khai mô hình AI
  - Triển khai mô hình inference thông qua REST API sử dụng FastAPI.
- Xây dựng giao diện người dùng
  - Phát triển giao diện đơn giản với Streamlit để người dùng có thể: Nhập dữ liệu đầu vào, gửi yêu cầu đến API dự đoán, Xem kết quả một cách trực quan
  - Giao diện Streamlit kết nối với API qua HTTP request, hoạt động như một client frontend nhẹ.
- Điều phối và triển khai
  - Tất cả các dịch vụ (MinIO, PostgreSQL, Airflow, API, Streamlit) được cấu hình chạy trên Docker Compose.
  - Sử dụng volume và mạng nội bộ của Docker để các dịch vụ có thể giao tiếp an toàn và hiệu quả.
  - Đảm bảo hệ thống có thể khởi chạy nhanh chóng chỉ với một lệnh docker-compose up.
- Kiểm thử và đánh giá

- Kiểm thử pipeline ETL, đảm bảo dữ liệu được xử lý đúng và đầy đủ.
- Kiểm thử mô hình dự đoán với các tập dữ liệu kiểm thử riêng biệt.
- Thử nghiệm giao diện người dùng để đảm bảo tính trực quan và dễ sử dụng.

Chương 2

Phân tích yêu cầu

2.1 Yêu cầu chức năng

Hệ thống được thiết kế nhằm hỗ trợ toàn bộ quy trình xử lý dữ liệu, triển khai mô hình trí tuệ nhân tạo, và cung cấp giao diện tương tác thân thiện với người dùng. Các chức năng chính bao gồm:

- **Nhập và lưu trữ dữ liệu (Data Ingestion & Storage)**
  - Hệ thống cho phép nhập dữ liệu từ nhiều nguồn như API, cơ sở dữ liệu,...
  - Dữ liệu thô được lưu trữ tại MinIO, đóng vai trò là nguồn dữ liệu chính cho các pipeline xử lý.
- **Chuyển đổi và chuẩn hóa dữ liệu (Data Transformation)**
  - Dữ liệu từ MinIO được trích xuất và xử lý bằng Pandas, sau đó nạp vào PostgreSQL (Data Warehouse).
  - Quy trình xử lý bao gồm chuẩn hóa dữ liệu (làm sạch, chuyển đổi kiểu dữ liệu, xử lý thiếu, loại bỏ nhiễu), xây dựng bảng trung gian và bảng phân tích phục vụ huấn luyện mô hình.
- **Triển khai API dự đoán (Model Inference API)**
  - Xây dựng REST API sử dụng FastAPI với endpoint `POST /models/predict_model`, nhận dữ liệu đầu vào và trả về kết quả dự đoán.
  - API hoạt động độc lập, dễ tích hợp với giao diện người dùng hoặc các hệ thống bên ngoài.
- **Giao diện người dùng (User Interface với Streamlit)**
  - Cung cấp giao diện đơn giản, dễ sử dụng cho người dùng không chuyên về kỹ thuật.
  - Cho phép người dùng nhập dữ liệu đầu vào, gửi đến API và hiển thị kết quả dự đoán theo thời gian thực.
- **Tự động hóa và điều phối quy trình (Orchestration)**

  - Tự động hóa toàn bộ quy trình ETL và huấn luyện mô hình định kỳ bằng Apache Airflow.
  - Các bước xử lý được mô hình hóa dưới dạng DAG bao gồm: thu thập dữ liệu → lưu vào MinIO → xử lý với Pandas → nạp vào PostgreSQL.

- **Triển khai và quản lý hệ thống**
  - Toàn bộ hệ thống được đóng gói và quản lý thông qua Docker Compose, dễ dàng triển khai và vận hành trong các môi trường dev/test/prod.
  - Cấu hình logging và giám sát cơ bản để hỗ trợ phát hiện lỗi và theo dõi quá trình xử lý dữ liệu.

2.2 Yêu cầu phi chức năng

- **Khả năng mở rộng (Scalability)**
  - Mỗi thành phần được triển khai dưới dạng service độc lập trong Docker, cho phép mở rộng riêng biệt theo nhu cầu.
  - Pipeline ETL và mô hình có thể xử lý dữ liệu lớn hoặc huấn luyện lại định kỳ dễ dàng.

- **Tính sẵn sàng và độ tin cậy (Availability & Reliability)**
  - Hệ thống có thể khởi chạy tất cả thành phần bằng một lệnh duy nhất: `docker-compose up`.
  - Cơ chế ghi log và retry được tích hợp trong Airflow để xử lý lỗi phát sinh trong pipeline.

- **Tính bảo mật (Security)**
  - API kiểm tra và xác thực đầu vào, từ chối các yêu cầu sai định dạng hoặc có nguy cơ injection.
  - Cấu hình mạng Docker chỉ mở các cổng cần thiết (ví dụ: 8501 cho Streamlit, 8000 cho FastAPI).

- **Khả năng bảo trì và mở rộng mã nguồn (Maintainability & Modularity)**
  - Mã nguồn được tổ chức theo module: ingestion, transformation, model, API, UI,...
  - Các module hoạt động độc lập, dễ nâng cấp hoặc thay thế.
  - Có tài liệu README hướng dẫn chi tiết cách cài đặt và vận hành hệ thống.

- **Khả năng tái sử dụng (Reusability)**
  - Có thể dễ dàng thay thế mô hình AI khác bằng cách cập nhật cấu hình.
  - Giao diện Streamlit và API được thiết kế tổng quát, cho phép tùy biến linh hoạt.

- **Hiệu suất xử lý (Performance)**
  - Hệ thống có thể xử lý batch dữ liệu lớn với độ trễ thấp trong quá trình dự đoán.
  - API dự đoán có thể phản hồi kết quả trong vòng < 1 giây trên dữ liệu đơn lẻ (theo kiểm thử nội bộ).

2.3 Các đối tượng sử dụng hệ thống

Hệ thống hướng tới phục vụ đa dạng người dùng với mục tiêu và nhu cầu khác nhau trong toàn bộ chuỗi xử lý dữ liệu và khai thác AI:

2.3.1 Nhà phân tích dữ liệu (Data Analyst)

- **Mục tiêu sử dụng:** Khai thác dữ liệu đã được xử lý để phân tích, trực quan hóa hoặc xây dựng báo cáo.
- **Tính năng:**
  - Truy vấn dữ liệu từ PostgreSQL.
  - Sử dụng kết quả từ bước chuyển đổi dữ liệu để phân tích chuyên sâu.
  - (Nếu có giao diện): Tải dữ liệu mới lên hệ thống để cập nhật phân tích.

2.3.2 Kỹ sư dữ liệu (Data Engineer)

- **Mục tiêu sử dụng:** Xây dựng, điều phối và duy trì các pipeline ETL.
- **Tính năng:**
  - Quản lý DAG trên Airflow.
  - Theo dõi log, giám sát trạng thái thực thi.
  - Cấu hình và cập nhật Docker Compose khi cần thiết.

2.3.3 Kỹ sư AI / Machine Learning Engineer

- **Mục tiêu sử dụng:** Huấn luyện mô hình, triển khai API dự đoán, theo dõi và cải tiến mô hình.
- **Tính năng:**
  - Truy cập dữ liệu chuẩn hóa từ Data Warehouse để train mô hình.
  - Huấn luyện, lưu trữ và triển khai mô hình dự đoán thông qua FastAPI.

2.3.4 Người dùng cuối (End-user / Khách hàng nội bộ)

- **Mục tiêu sử dụng:** Sử dụng hệ thống để nhận kết quả dự đoán mà không cần hiểu rõ về kỹ thuật phía sau.
- **Tính năng:**
  - Giao diện Streamlit để nhập dữ liệu và xem kết quả dự đoán theo thời gian thực.

2.3.5 Quản trị viên hệ thống (System Administrator)

- **Mục tiêu sử dụng:** Giám sát, cấu hình và duy trì toàn bộ hệ thống ở trạng thái ổn định.
- **Tính năng:**
  - Khởi động, dừng và kiểm tra trạng thái dịch vụ thông qua Docker Compose.
  - Theo dõi tài nguyên hệ thống (RAM, CPU) và giám sát bảo mật nội bộ.

Chương 3

Công nghệ và công cụ sử dụng

3.1 Ngôn ngữ lập trình

Trong quá trình xây dựng và triển khai hệ thống, dự án sử dụng chủ yếu ngôn ngữ Python kết hợp với các công cụ cấu hình và truy vấn dữ liệu khác như SQL, YAML và Dockerfile.

3.1.1 Python

- **Vai trò chính:**
  - Xử lý dữ liệu (ETL, chuẩn hóa, biến đổi dữ liệu)
  - Huấn luyện và triển khai mô hình học máy
  - Xây dựng API phục vụ mô hình (FastAPI)
  - Phát triển giao diện người dùng tương tác (Streamlit)
- **Lý do lựa chọn:**
  - Hệ sinh thái thư viện phong phú cho AI, xử lý dữ liệu và phát triển web (Pandas, NumPy, Scikit-learn, FastAPI, Streamlit, XGBoost, v.v.)
  - Dễ đọc, dễ học, dễ triển khai, cộng đồng hỗ trợ lớn
  - Tích hợp mượt với các công cụ dữ liệu như Airflow, MinIO, PostgreSQL,...

3.1.2 SQL

- **Vai trò chính:** Truy vấn và thao tác dữ liệu trong PostgreSQL.
- **Lý do lựa chọn:** Ngôn ngữ chuẩn và mạnh mẽ để xử lý dữ liệu quan hệ, dễ tích hợp với công cụ BI và Data Warehouse.

3.1.3 YAML & Dockerfile

- **Vai trò chính:**

- Định nghĩa cấu hình cho các pipeline và dịch vụ (FastAPI, Airflow, Streamlit,...)
- Xây dựng Docker images cho từng thành phần
- Quản lý Docker Compose để triển khai hệ thống đồng bộ

3.2 Framework và thư viện sử dụng

Hệ thống sử dụng các framework và thư viện mã nguồn mở phổ biến để xây dựng pipeline dữ liệu, huấn luyện mô hình, triển khai API và giao diện người dùng.

3.2.1 Xử lý dữ liệu

- **Pandas:** Phân tích và xử lý dữ liệu dạng bảng.
- **NumPy:** Hỗ trợ tính toán hiệu năng cao trên mảng và ma trận.
- **Apache Airflow:** Tự động hóa và điều phối các pipeline ETL theo lịch định kỳ.

3.2.2 Lưu trữ và quản lý dữ liệu

- **MinIO:** Dịch vụ lưu trữ dạng object (tương thích S3), dùng làm Data Lake chứa dữ liệu thô.
- **PostgreSQL:** Hệ quản trị cơ sở dữ liệu quan hệ, đóng vai trò Data Warehouse lưu trữ dữ liệu đã xử lý.
- **SQLAlchemy:** Thư viện ORM giúp kết nối và thao tác dữ liệu PostgreSQL trong Python.

3.2.3 Huấn luyện mô hình học máy

- **Scikit-learn:** Cung cấp các thuật toán ML cơ bản như Logistic Regression, Random Forest,...
- **XGBoost:** Thuật toán boosting mạnh, hiệu quả với các bài toán dự đoán tài chính.
- **Pickle:** Lưu trữ và nạp lại mô hình đã huấn luyện dưới dạng file nhị phân.

3.2.4 API và dịch vụ web

- **FastAPI:** Xây dựng REST API nhanh, hỗ trợ async và có sẵn tài liệu Swagger.
- **Uvicorn:** ASGI server chạy nền cho FastAPI, hiệu năng cao, nhẹ.

3.2.5 Giao diện người dùng

- **Streamlit:** Tạo giao diện web đơn giản, nhanh chóng, cho phép người dùng nhập dữ liệu và xem kết quả dự đoán một cách trực quan.

3.2.6 Triển khai và điều phối hệ thống

- **Docker:** Đóng gói từng thành phần thành container độc lập, giúp triển khai nhất quán.
- **Docker Compose:** Quản lý đa container (FastAPI, PostgreSQL, MinIO, Streamlit, Airflow,...) chỉ bằng một tệp cấu hình duy nhất.

3.2.7 Giám sát và logging

- **Logging (Python):** Ghi lại log chi tiết trong quá trình chạy pipeline, API và xử lý mô hình, phục vụ giám sát và gỡ lỗi.

Chương 4

Triển khai và đánh giá

4.1 Quy trình triển khai

Quy trình triển khai được thiết kế theo các giai đoạn tuần tự, nhằm đảm bảo hệ thống được xây dựng một cách linh hoạt, dễ bảo trì và mở rộng. Cụ thể bao gồm các bước sau:

4.1.1 Bước 1: Thiết kế kiến trúc tổng thể

- Xác định các thành phần chính của hệ thống: Data Lake (MinIO), Data Warehouse (PostgreSQL), công cụ ETL (Airflow + Pandas), mô hình AI, API (FastAPI), và giao diện người dùng (Streamlit).
- Lên sơ đồ kiến trúc tổng quan thể hiện dòng dữ liệu từ đầu vào đến khi ra kết quả dự đoán.

4.1.2 Bước 2: Triển khai hạ tầng và môi trường phát triển

- Tạo file `docker-compose.yml` để chạy đồng thời các dịch vụ: MinIO, PostgreSQL, FastAPI, Streamlit, DBT và Airflow.
- Cấu hình volume, network, biến môi trường và các thư mục chia sẻ giữa các container.
- Tạo script khởi động và cấu hình ban đầu (seed data nếu cần).

4.1.3 Bước 3: Xây dựng pipeline xử lý dữ liệu

- Viết DAG trong Airflow để tự động tải dữ liệu vào MinIO theo lịch định sẵn.
- Dùng Pandas để đọc dữ liệu từ MinIO (qua connector) và transform vào PostgreSQL.
- Kiểm tra dữ liệu sau transform đảm bảo định dạng đúng và đầy đủ.

4.1.4 Bước 4: Huấn luyện và triển khai mô hình AI

- Sử dụng dữ liệu đã được transform để huấn luyện mô hình (với scikit-learn hoặc các thư viện ML khác).
- Lưu mô hình đã train ra file `.pkl`.
- Triển khai mô hình thông qua một REST API sử dụng FastAPI.
- Tạo endpoint như `/predict` để sử dụng AI.

4.1.5 Bước 5: Phát triển giao diện người dùng với Streamlit

- Xây dựng một giao diện đơn giản cho phép người dùng nhập/tải lên dữ liệu đầu vào.
- Gửi dữ liệu tới API để nhận kết quả dự đoán.

4.1.6 Bước 6: Kiểm thử toàn hệ thống

- Kiểm tra các pipeline ETL hoạt động đúng theo lịch.
- Kiểm tra API có phản hồi đúng dữ liệu và mô hình dự đoán chính xác.
- Kiểm tra giao diện người dùng thân thiện, dễ thao tác và không lỗi.

4.2 Các điểm mạnh của hệ thống

4.2.1 Kiến trúc hiện đại, dễ mở rộng

- Sử dụng kiến trúc microservices với Docker Compose, giúp dễ triển khai và nâng cấp từng thành phần mà không ảnh hưởng đến toàn bộ hệ thống.
- Có thể dễ dàng mở rộng mô hình AI, thay đổi cơ sở dữ liệu, hoặc thêm module xử lý mà không cần viết lại hệ thống từ đầu.

4.2.2 Chuỗi xử lý dữ liệu hoàn chỉnh

- Hệ thống bao phủ toàn bộ pipeline xử lý dữ liệu:
  - Thu thập dữ liệu → Lưu trữ (MinIO)
  - Transform (Pandas)
  - Lưu trữ và truy xuất dữ liệu (PostgreSQL)
  - Huấn luyện mô hình → Triển khai API → Giao diện người dùng
- Đảm bảo luồng dữ liệu rõ ràng, có thể kiểm soát và truy vết (data lineage).

4.2.3 Tích hợp AI dễ dàng và hiệu quả

- Mô hình AI được đóng gói độc lập và có API dự đoán riêng.
- Có thể thay thế hoặc cập nhật mô hình mới mà không ảnh hưởng đến phần còn lại của hệ thống.

4.2.4 Giao diện thân thiện, dễ sử dụng

- Giao diện người dùng Streamlit đơn giản, trực quan, phù hợp cả với người không chuyên về kỹ thuật.
- Có thể chạy trực tiếp trên trình duyệt, không cần cài đặt phức tạp.

4.2.5 Dễ triển khai và bảo trì

- Tất cả dịch vụ được quản lý bằng Docker Compose → Chỉ cần 1 lệnh để khởi chạy toàn bộ hệ thống.
- Cấu trúc mã nguồn rõ ràng, tách biệt theo chức năng: ml/, api/, storage/, ui/, ...

Chương 5

Kết luận

5.1 Tóm tắt kết quả đạt được

Sau quá trình phân tích, thiết kế, triển khai và kiểm thử, hệ thống đã đạt được những kết quả chính như sau:

- Triển khai thành công toàn bộ pipeline dữ liệu
  - Dữ liệu được thu thập, lưu trữ tại MinIO (Data Lake).
  - Sử dụng Pandas để xử lý, làm sạch và chuyển đổi dữ liệu một cách tự động.
  - Dữ liệu sạch được đưa vào PostgreSQL làm Data Warehouse, phục vụ cho phân tích và dự đoán.
- Tích hợp thành công mô hình AI vào hệ thống
  - Mô hình Machine Learning đã được huấn luyện offline và triển khai dưới dạng API riêng biệt.
  - Hệ thống API hoạt động ổn định, cho phép nhận đầu vào, xử lý và trả kết quả dự đoán nhanh chóng.
- Giao diện người dùng thân thiện
  - Ứng dụng giao diện người dùng được xây dựng bằng Streamlit, cho phép người dùng upload dữ liệu.
  - Theo dõi kết quả xử lý.
  - Gửi yêu cầu dự đoán đến mô hình AI.
  - Hiển thị kết quả một cách trực quan.
- Triển khai hệ thống bằng Docker Compose
  - Toàn bộ hệ thống được đóng gói thành các dịch vụ Docker riêng biệt, có thể khởi chạy chỉ với một lệnh duy nhất.
  - Việc triển khai dễ dàng, tiết kiệm thời gian và phù hợp cho cả môi trường phát triển lẫn thử nghiệm.

5.2 Kết luận

Hệ thống đã hoàn thành đúng các mục tiêu ban đầu đề ra: Xây dựng một nền tảng xử lý dữ liệu tích hợp trí tuệ nhân tạo, có giao diện thân thiện, dễ triển khai và mở rộng. Đây là tiền đề quan trọng để tiếp tục phát triển hệ thống trong các ứng dụng thực tế quy mô lớn hơn trong tương lai.