

HỌC VIÊN BUỔI CHÍNH VIỄN THÔNG
KHOA ĐÀO TẠO SAU ĐẠI HỌC



BÁO CÁO BÀI TẬP LỚN
BỘ MÔN: CÁC HỆ THỐNG PHÂN TÁN

Nền Tảng Lập Kế Hoạch Du Lịch Microservices

Giảng viên hướng dẫn:

TS. Kim Ngọc Bách

Lớp:

M25CQHT01-B

Nhóm thực hiện:

15

Học viên thực hiện:

Hoàng Anh Tuấn
Đinh Hữu Tường
Nguyễn Đình Khả

B25CHT061
B25CHT067
B25CHT030

Hà Nội, Tháng 12 Năm 2024

Mục lục

1 GIỚI THIỆU TỔNG QUAN	4
1.1 Đặt vấn đề	4
1.2 Mục tiêu dự án	4
1.3 Công nghệ sử dụng	4
1.3.1 Backend Framework	4
1.3.2 Databases	5
1.3.3 External APIs	5
1.3.4 DevOps & Infrastructure	5
2 PHÂN TÍCH YÊU CẦU HỆ THỐNG	6
2.1 Yêu cầu chức năng	6
2.1.1 User Management	6
2.1.2 Destination Discovery	6
2.1.3 Weather Information	6
2.1.4 Flight & Hotel Booking	7
2.1.5 Itinerary Planning	7
2.2 Yêu cầu phi chức năng	7
2.2.1 Security	7
2.2.2 Performance	7
2.2.3 Scalability	8
2.2.4 Reliability	8
2.2.5 Maintainability	8
3 KIẾN TRÚC MICROSERVICES	9
3.1 Tổng quan kiến trúc	9
3.1.1 System Architecture Diagram	9
3.2 Services Overview	10
3.3 Các Design Patterns được áp dụng	10
3.3.1 API Gateway Pattern	10
3.3.2 Database per Service Pattern	10
3.3.3 Repository Pattern	11
3.3.4 Clean Architecture	11
4 CHI TIẾT CÁC MICROSERVICES	12
4.1 Middleware Service (API Gateway)	12
4.1.1 Vai trò và chức năng	12
4.1.2 Kiến trúc chi tiết	12
4.1.3 Routing Rules	13
4.1.4 Security Considerations	13

4.2	Destination Service	13
4.2.1	Chức năng	13
4.2.2	Database Schema (MySQL)	14
4.2.3	Geoapify Integration	14
4.3	Weather Service	14
4.3.1	Chức năng	14
4.3.2	OpenWeather API Integration	14
4.3.3	Error Handling	15
4.4	Booking Service	15
4.4.1	Chức năng	15
4.4.2	Amadeus API Integration	15
4.4.3	Entities và Domain Models	16
4.5	Itinerary Service	17
4.5.1	Chức năng	17
4.5.2	Database Schema (PostgreSQL)	17
4.5.3	Repository Pattern Implementation	18
5	DEPLOYMENT VÀ TESTING	19
5.1	Docker Containerization	19
5.1.1	Docker Compose Configuration	19
5.1.2	Deployment Steps	20
5.2	Health Checks	20
5.3	API Documentation	21
5.4	Testing Strategy	21
5.4.1	Manual Testing	21
5.4.2	Automated Testing (Planned)	22
6	PHÂN TÍCH VÀ ĐÁNH GIÁ	23
6.1	Ưu điểm của hệ thống	23
6.1.1	Kiến trúc	23
6.1.2	Clean Architecture	23
6.1.3	API Gateway Pattern	23
6.2	Hạn chế và vấn đề	23
6.2.1	Security Issues	23
6.2.2	Architecture Limitations	24
6.2.3	Missing Features	24
6.3	Bài học kinh nghiệm	24
6.3.1	Microservices Best Practices	24
6.3.2	Security Lessons	25
6.3.3	Development Process	25
7	HƯỚNG PHÁT TRIỂN	26
7.1	Improvements ngắn hạn	26
7.1.1	Security Enhancements (Priority 1)	26
7.1.2	Caching Layer	26
7.1.3	Complete CRUD Operations	26
7.2	Improvements dài hạn	27
7.2.1	Resilience Patterns	27

7.2.2	Observability	27
7.2.3	Event-Driven Architecture	27
7.2.4	Advanced Features	28
7.3	DevOps Improvements	28
7.3.1	CI/CD Pipeline	28
7.3.2	Kubernetes Deployment	29
7.3.3	Monitoring & Alerting	29
8	KẾT LUẬN	30
8.1	Tổng kết	30
8.1.1	Thành tựu đạt được	30
8.1.2	Kiến thức thu được	30
8.2	Dánh giá khách quan	31
8.2.1	Điểm mạnh	31
8.2.2	Điểm yếu	31
8.3	Khuyến nghị	31
8.3.1	Cho Development	31
8.3.2	Cho Production Deployment	31
8.4	Lời kết	32
A	TÀI LIỆU THAM KHẢO	33
B	CODE SNIPPETS QUAN TRỌNG	34
B.1	API Gateway Proxy Implementation	34
B.2	JWT Authentication	35
B.3	Repository Pattern	35

Chương 1

GIỚI THIỆU TỔNG QUAN

1.1 Đặt vấn đề

Trong bối cảnh công nghệ phát triển nhanh chóng, kiến trúc microservices đã trở thành xu hướng chủ đạo trong việc xây dựng các hệ thống phân tán quy mô lớn. Trip Hub là một dự án minh họa triển khai hoàn chỉnh kiến trúc microservices áp dụng vào lĩnh vực du lịch, cung cấp nền tảng toàn diện cho việc lập kế hoạch và quản lý chuyến du lịch.

Hệ thống được thiết kế với 5 microservices độc lập, mỗi service đảm nhận một chức năng cụ thể trong quy trình du lịch, từ khám phá điểm đến, dự báo thời tiết, đặt vé máy bay và khách sạn, đến quản lý lịch trình chi tiết. Các services tương tác với nhau thông qua API Gateway, tạo nên một hệ sinh thái phần mềm linh hoạt, dễ mở rộng và bảo trì.

1.2 Mục tiêu dự án

Dự án Trip Hub được xây dựng với các mục tiêu sau:

- **Nghiên cứu và triển khai:** Áp dụng kiến trúc microservices vào thực tế, hiểu rõ các nguyên tắc thiết kế và thách thức khi xây dựng hệ thống phân tán.
- **Tích hợp API bên ngoài:** Làm việc với các API thực tế từ OpenWeatherMap và Amadeus, học cách xử lý authentication, rate limiting và error handling.
- **API Gateway Pattern:** Triển khai gateway làm single entry point, centralized authentication và request routing.
- **Database per Service:** Áp dụng nguyên tắc polyglot persistence với PostgreSQL và MySQL cho các services khác nhau.
- **Containerization:** Sử dụng Docker và Docker Compose để đóng gói và orchestrate các microservices.
- **Clean Architecture:** Thiết kế mỗi service theo nguyên tắc separation of concerns, dependency injection và repository pattern.

1.3 Công nghệ sử dụng

1.3.1 Backend Framework

- **Python 3.11+:** Ngôn ngữ lập trình chính

- **FastAPI**: Framework web hiện đại với hiệu năng cao, hỗ trợ async/await và tự động sinh OpenAPI documentation
- **Pydantic**: Validation và serialization dữ liệu với type hints

1.3.2 Databases

- **PostgreSQL**: Database cho user management và itinerary service
- **MySQL**: Database cho destination service
- **Redis**: Caching layer (đã config nhưng chưa implement)

1.3.3 External APIs

- **Amadeus API**: Cung cấp dữ liệu chuyến bay và khách sạn
- **OpenWeatherMap API**: Dữ liệu thời tiết hiện tại và dự báo
- **Geoapify API**: Geocoding và places search

1.3.4 DevOps & Infrastructure

- **Docker & Docker Compose**: Containerization và orchestration
- **SQLAlchemy**: ORM cho Python
- **httpx**: Async HTTP client
- **JWT (JSON Web Tokens)**: Authentication mechanism

Chương 2

PHÂN TÍCH YÊU CẦU HỆ THỐNG

2.1 Yêu cầu chức năng

2.1.1 User Management

- User registration với username và password
- User login phát hành JWT token
- JWT token-based authentication cho protected endpoints
- Shared user database giữa Middleware và Itinerary services
- Token expiration: 1 giờ

2.1.2 Destination Discovery

- Browse danh sách điểm đến du lịch
- Tìm kiếm destinations theo keyword (tên địa điểm, quốc gia)
- Filter theo country, category, rating
- Pagination support
- Lấy thông tin chi tiết destination với description và attractions
- Tìm kiếm điểm tham quan và khách sạn gần một địa điểm

2.1.3 Weather Information

- Lấy thời tiết hiện tại cho một location
- Dự báo thời tiết 5 ngày với interval 3 giờ
- Hiển thị temperature, humidity, wind speed, conditions
- Weather icons và descriptions
- Support multiple cities worldwide

2.1.4 Flight & Hotel Booking

- Tìm kiếm chuyến bay giữa 2 sân bay (origin → destination)
- Filter theo travel class (Economy, Business, First), non-stop flights, price range
- Tìm kiếm khách sạn theo thành phố với city code (IATA)
- Room availability và pricing với check-in/check-out dates
- Support multiple adults, children, rooms
- City/airport reference data với 50+ major cities
- Real-time data từ Amadeus API

2.1.5 Itinerary Planning

- Tạo travel itineraries với title, date range, description
- Thêm activities vào itineraries với time slots
- Date range management (start_date → end_date)
- Activity scheduling với start_time, end_time, location
- User-specific data isolation (chỉ xem được data của mình)
- UUID primary keys cho uniqueness

2.2 Yêu cầu phi chức năng

2.2.1 Security

- JWT authentication cho protected endpoints
- User data isolation (mỗi user chỉ truy cập data của mình)
- CORS support cho web client
- **Limitation hiện tại:** Plain text passwords (chỉ cho development, không production-ready)

2.2.2 Performance

- External API caching (planned với Redis)
- Connection pooling cho databases
- Async/await pattern cho tất cả I/O operations
- HTTP timeout handling (10-30 seconds)
- Minimize blocking operations

2.2.3 Scalability

- Stateless services (horizontal scaling ready)
- Database per service pattern (independent scaling)
- Containerized deployment với Docker
- Microservices có thể deploy và scale độc lập

2.2.4 Reliability

- Comprehensive error handling và logging
- Health check endpoints cho mỗi service
- Automatic database table creation on startup
- Transaction support với SQLAlchemy
- Graceful error responses cho API consumers

2.2.5 Maintainability

- Clean Architecture patterns
- Repository pattern cho data access layer
- Dependency injection với FastAPI Depends
- Type safety với Python type hints và Pydantic
- Comprehensive documentation (README cho mỗi service)
- Auto-generated OpenAPI/Swagger documentation

Chương 3

KIẾN TRÚC MICROSERVICES

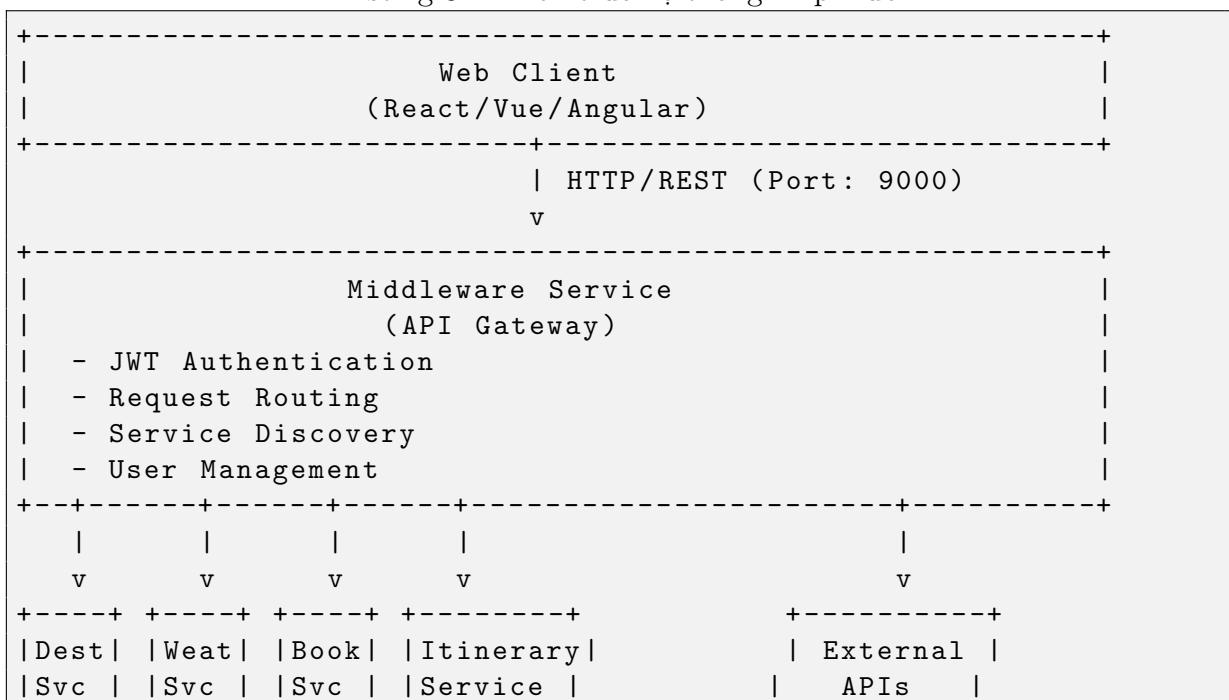
3.1 Tổng quan kiến trúc

Hệ thống Trip Hub được thiết kế theo kiến trúc microservices với 5 services độc lập, tương tác qua HTTP/REST API thông qua một API Gateway. Kiến trúc này mang lại nhiều lợi ích:

- **Loose Coupling:** Mỗi service độc lập, có thể phát triển và deploy riêng biệt
- **Technology Flexibility:** Mỗi service có thể sử dụng database và công nghệ phù hợp
- **Scalability:** Scale từng service độc lập dựa trên load
- **Resilience:** Lỗi ở một service không làm sập toàn bộ hệ thống
- **Team Organization:** Các team khác nhau phát triển các services khác nhau

3.1.1 System Architecture Diagram

Listing 3.1: Kiến trúc hệ thống Trip Hub





3.2 Services Overview

Bảng 3.1: Tổng quan các Microservices

Service	Port	Database	External API	Responsibility
Middleware	9000	PostgreSQL	-	API Gateway, Auth, Routing
Destination	8001	MySQL	Geoapify	Destination catalog
Weather	8002	-	OpenWeatherMap	Weather forecasts
Booking	8000	-	Amadeus	Flight/Hotel search
Itinerary	8000	PostgreSQL	-	Trip planning

3.3 Các Design Patterns được áp dụng

3.3.1 API Gateway Pattern

Middleware Service hoạt động như API Gateway, cung cấp:

- **Single Entry Point:** Tất cả client requests qua port 9000
- **Authentication:** JWT validation trước khi routing
- **Service Discovery:** Map service names → URLs
- **Request Proxying:** Forward requests với preserved data
- **Error Handling:** Unified error responses (401, 404, 502, 504)
- **CORS Handling:** Centralized CORS configuration

3.3.2 Database per Service Pattern

Mỗi service có database riêng (hoặc không có database):

- **Middleware & Itinerary:** Shared PostgreSQL database `trip_hub`
- **Destination:** MySQL database
- **Booking & Weather:** Stateless, không có database (data từ external APIs)

Lưu ý: Middleware và Itinerary services share database để đồng bộ user table, đây là compromise cho đơn giản hóa authentication.

3.3.3 Repository Pattern

Tất cả các services có database đều sử dụng Repository Pattern:

Listing 3.2: Repository Pattern Example

```
class ItineraryRepo:
    def __init__(self, session: Session):
        self.session = session

    def create(self, username: str, payload) -> dict:
        id = str(uuid.uuid4())
        item = Itinerary(id=id, username=username, ...)
        self.session.add(item)
        self.session.commit()
        self.session.refresh(item)
        return self._to_dict(item)

    def list_by_user(self, username: str) -> List[dict]:
        rows = self.session.query(Itinerary) \
            .filter(Itinerary.username == username) \
            .order_by(Itinerary.created_at.desc()) \
            .all()
        return [self._to_dict(row) for row in rows]
```

3.3.4 Clean Architecture

Mỗi service được tổ chức theo Clean Architecture:

- **API Layer (api/)**: HTTP endpoints, request/response handling
- **Core Layer (core/)**: Business logic, entities, use cases
- **Infrastructure Layer (infrastructure/)**: External APIs, databases
- **Schemas Layer (schemas/)**: Pydantic models cho validation

Chương 4

CHI TIẾT CÁC MICROSERVICES

4.1 Middleware Service (API Gateway)

4.1.1 Vai trò và chức năng

Middleware Service là trái tim của hệ thống, đóng vai trò API Gateway với các chức năng:

1. **Single Entry Point:** Port 9000 là điểm truy cập duy nhất từ client
2. **JWT Authentication:** Quản lý user registration, login và JWT tokens
3. **Request Routing:** Forward requests đến đúng downstream service
4. **Service Discovery:** Quản lý mapping giữa service names và URLs
5. **Error Translation:** Chuyển đổi errors từ downstream services

4.1.2 Kiến trúc chi tiết

Components chính:

- **ServiceRouter:** Quản lý service registry và build target URLs
- **Authentication System:** JWT creation và validation
- **Proxy Mechanism:** Forward HTTP requests với httpx AsyncClient
- **User Repository:** Quản lý user data trong PostgreSQL

Authentication Flow:

Listing 4.1: JWT Authentication Flow

- ```
1. User registers/login -> Middleware
2. Middleware returns JWT token (expires in 1 hour)
3. Client includes token in subsequent requests:
 Authorization: Bearer <token>
4. Middleware validates token
5. If valid -> proxy to downstream service
6. If invalid -> 401 Unauthorized
```

Bảng 4.1: API Routing trong Middleware

| Client Request        | Downstream Service         |
|-----------------------|----------------------------|
| /api/v1/auth/*        | Middleware (local)         |
| /api/v1/destination/* | Destination Service (8001) |
| /api/v1/weather/*     | Weather Service (8002)     |
| /api/v1/booking/*     | Booking Service (8000)     |
| /api/v1/itinerary/*   | Itinerary Service (8000)   |

### 4.1.3 Routing Rules

### 4.1.4 Security Considerations

**Limitations hiện tại (Development Mode):**

- Passwords được lưu dưới dạng plain text (KHÔNG hash!)
- JWT secret key hardcoded: "SECRET"
- Không có rate limiting
- Không có HTTPS/TLS

**Khuyến nghị cho Production:**

- Implement bcrypt hoặc argon2 password hashing
- Move JWT secret sang environment variables
- Thêm rate limiting middleware
- Enable HTTPS/TLS termination
- Implement API key rotation

## 4.2 Destination Service

### 4.2.1 Chức năng

Destination Service quản lý thông tin điểm đến du lịch với các tính năng:

- Tìm kiếm destinations theo keyword và country
- Lấy chi tiết một destination cụ thể
- Tìm attractions (điểm tham quan) gần một địa điểm
- Tìm hotels gần một địa điểm
- Tích hợp với Geoapify API (geocoding và places search)

### 4.2.2 Database Schema (MySQL)

Service không lưu trữ data nội bộ, tất cả query trực tiếp từ Geopify API. Configuration có MySQL nhưng không được sử dụng trong implementation hiện tại.

### 4.2.3 Geopify Integration

**API Endpoints sử dụng:**

1. /v1/geocode/search: Tìm kiếm địa điểm và chuyển thành tọa độ
2. /v2/places: Tìm places theo category (tourism, accommodation.hotel)

**Flow tìm attractions:**

Listing 4.2: Attractions Search Flow

- ```

1. Client request: GET /api/v1/attractions?location=Paris
2. Geocode "Paris" -> (lon, lat)
3. Call Places API:
   - categories=tourism
   - filter=circle:lon,lat,5000m
   - limit=20
4. Map results to Attraction entities
5. Return list of attractions

```

4.3 Weather Service

4.3.1 Chức năng

Weather Service cung cấp thông tin thời tiết với OpenWeatherMap API:

- Thời tiết hiện tại cho một location
- Dự báo thời tiết 5 ngày (interval 3 giờ)
- Temperature (Celsius), description, conditions
- Stateless service (không cache)

4.3.2 OpenWeather API Integration

Endpoints:

1. /weather: Current weather
2. /forecast: 5-day forecast

API Request Example:

```
GET https://api.openweathermap.org/data/2.5/weather
?q=Paris
&appid={api_key}
&units=metric
```

4.3.3 Error Handling

Service map các HTTP errors từ OpenWeather:

- 401 Unauthorized → Invalid API key
- 404 Not Found → Location không tìm thấy
- 5xx Server Error → OpenWeather API down

4.4 Booking Service

4.4.1 Chức năng

Booking Service là service phức tạp nhất, tích hợp với Amadeus API:

- Flight search với origin, destination, dates, travel class
- Hotel search theo city code
- Cities reference data (50+ major cities)
- OAuth2 token management cho Amadeus API
- Real-time availability và pricing

4.4.2 Amadeus API Integration

Authentication Flow:

Listing 4.3: Amadeus OAuth2 Flow

```

1. Request access token:
POST /v1/security/oauth2/token
grant_type=client_credentials
client_id={api_key}
client_secret={api_secret}

2. Response:
{
    "access_token": "abc123...",
    "expires_in": 1799
}

3. Cache token in memory (expires - 60s buffer)

4. Use token cho subsequent API calls:
Authorization: Bearer abc123...

```

Flight Search:

Listing 4.4: Flight Search API

```
GET /v2/shopping/flight-offers
?originLocationCode=HAN
&destinationLocationCode=BKK
&departureDate=2025-02-15
&returnDate=2025-02-20
&adults=2
&currencyCode=USD
```

Hotel Search (2-Step Process):

1. Call `/v1/reference-data/locations/hotels/by-city` để lấy hotel IDs
2. Call `/v3/shopping/hotel-offers` với hotel IDs để lấy pricing

Lưu ý: Amadeus không cho phép search hotels trực tiếp bằng city code, phải qua 2 bước.

4.4.3 Entities và Domain Models**FlightEntity:**

```
@dataclass
class FlightEntity:
    id: str
    source: str
    one_way: bool
    segments: List[Segment]
    price: Price
    validating_airline_codes: List[str]

    def get_total_duration(self) -> str
    def is_direct_flight(self) -> bool
    def get_total_stops(self) -> int
```

HotelEntity:

```
@dataclass
class HotelEntity:
    hotel_id: str
    name: str
    city_code: str
    rating: Optional[str]
    location: Optional[HotelLocation]
    amenities: List[HotelAmenity]
    rooms: List[Room]

    def get_min_price(self) -> Optional[float]
    def has_amenity(self, amenity_name: str) -> bool
```

4.5 Itinerary Service

4.5.1 Chức năng

Itinerary Service quản lý lịch trình du lịch của users:

- User authentication (shared với Middleware)
- CRUD operations cho itineraries
- CRUD operations cho activities trong itinerary
- User data isolation (chỉ xem data của mình)
- UUID primary keys

4.5.2 Database Schema (PostgreSQL)

Table: users (shared với Middleware)

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(150) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Table: itineraries

```
CREATE TABLE itineraries (
    id VARCHAR(36) PRIMARY KEY, -- UUID
    username VARCHAR(150) NOT NULL,
    title VARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Table: activities

```
CREATE TABLE activities (
    id VARCHAR(36) PRIMARY KEY, -- UUID
    itinerary_id VARCHAR(36) NOT NULL,
    username VARCHAR(150) NOT NULL,
    title VARCHAR(255) NOT NULL,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL,
    location VARCHAR(255) NOT NULL,
    note TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);
```

4.5.3 Repository Pattern Implementation

Listing 4.5: Itinerary Repository

```
class ItineraryRepo:  
    def __init__(self, session: Session):  
        self.session = session  
  
    def create(self, username: str, payload: ItineraryCreate):  
        id = str(uuid.uuid4())  
        item = Itinerary(  
            id=id,  
            username=username,  
            title=payload.title,  
            start_date=payload.start_date,  
            end_date=payload.end_date,  
            description=payload.description  
        )  
        self.session.add(item)  
        self.session.commit()  
        self.session.refresh(item)  
        return self._to_dict(item)  
  
    def list_by_user(self, username: str):  
        rows = self.session.query(Itinerary) \  
            .filter(Itinerary.username == username) \  
            .order_by(Itinerary.created_at.desc()) \  
            .all()  
        return [self._to_dict(row) for row in rows]
```

Chương 5

DEPLOYMENT VÀ TESTING

5.1 Docker Containerization

5.1.1 Docker Compose Configuration

Toàn bộ hệ thống được containerize với Docker Compose:

Listing 5.1: docker-compose.yml (simplified)

```
version: '3.8'

services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: trip_hub
      POSTGRES_USER: trip
      POSTGRES_PASSWORD: trip
    ports:
      - "5432:5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data

  mysql:
    image: mysql:8
    environment:
      MYSQL_DATABASE: destination_db
      MYSQL_USER: destination
      MYSQL_PASSWORD: destination
    volumes:
      - mysql-data:/var/lib/mysql

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  middleware-service:
    build: ./services/middleware-service
    ports:
      - "9000:9000"
    depends_on:
```

```

    - postgres
  environment:
    DATABASE_URL: postgresql://trip:trip@postgres:5432/trip_hub

  booking-service:
    build: ./services/booking-service
    ports:
      - "8000:8000"
    environment:
      AMADEUS_API_KEY: ${AMADEUS_API_KEY}
      AMADEUS_API_SECRET: ${AMADEUS_API_SECRET}

    # ... other services ...

  volumes:
    postgres-data:
    mysql-data:
    redis-data:

```

5.1.2 Deployment Steps

Listing 5.2: Quick Start Commands

```

# Clone repository
git clone <repo-url>
cd trip-hub

# Start all services
docker compose up -d --build

# Verify services
docker compose ps

# Check logs
docker compose logs -f middleware-service

# Stop all services
docker compose down

# Stop and remove volumes
docker compose down -v

```

5.2 Health Checks

Mỗi service có health check endpoint:

Bảng 5.1: Health Check Endpoints

Service	Health Check URL
Middleware	http://localhost:9000/health
Destination	http://localhost:8001/health
Weather	http://localhost:8002/health
Booking	http://localhost:8000/health
Itinerary	http://localhost:8003/health

5.3 API Documentation

FastAPI tự động sinh Swagger UI documentation:

- **Middleware**: http://localhost:9000/api/docs
- **Destination**: http://localhost:8001/api/docs
- **Weather**: http://localhost:8002/api/docs
- **Booking**: http://localhost:8000/api/docs
- **Itinerary**: http://localhost:8003/api/docs

5.4 Testing Strategy

5.4.1 Manual Testing

Complete User Journey Test:

Listing 5.3: End-to-End Testing

```
# 1. Register user
curl -X POST http://localhost:9000/api/v1/auth/register \
-H "Content-Type: application/json" \
-d '{"username": "traveler1", "password": "pass123"}'

# 2. Login
TOKEN=$(curl -X POST http://localhost:9000/api/v1/auth/login \
-H "Content-Type: application/json" \
-d '{"username": "traveler1", "password": "pass123"}' \
| jq -r '.access_token')

# 3. Search destinations
curl -H "Authorization: Bearer $TOKEN" \
"http://localhost:9000/api/v1/destination/destinations?query=paris"

# 4. Get weather forecast
curl -H "Authorization: Bearer $TOKEN" \
"http://localhost:9000/api/v1/weather/forecast?location=Paris"
```

```
# 5. Search flights
curl -X POST -H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
"http://localhost:9000/api/v1/booking/flights/search" \
-d '{
    "origin": "HAN",
    "destination": "CDG",
    "departure_date": "2025-06-01",
    "return_date": "2025-06-07",
    "adults": 2,
    "currency": "USD"
}'

# 6. Create itinerary
curl -X POST -H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
"http://localhost:9000/api/v1/itinerary/itineraries" \
-d '{
    "title": "Paris Trip",
    "start_date": "2025-06-01",
    "end_date": "2025-06-07",
    "description": "Summer vacation"
}'
```

5.4.2 Automated Testing (Planned)

Future improvements:

- Unit tests cho mỗi service với pytest
- Integration tests với test containers
- Contract testing giữa services
- Load testing với Locust hoặc k6
- E2E tests với Playwright

Chương 6

PHÂN TÍCH VÀ ĐÁNH GIÁ

6.1 Ưu điểm của hệ thống

6.1.1 Kiến trúc

1. **Loose Coupling:** Các services độc lập, thay đổi một service không ảnh hưởng services khác
2. **Technology Diversity:** Mỗi service có thể chọn database phù hợp (PostgreSQL, MySQL)
3. **Independent Deployment:** Deploy và scale từng service riêng biệt
4. **Clear Boundaries:** Mỗi service có responsibility rõ ràng

6.1.2 Clean Architecture

1. **Separation of Concerns:** API, business logic, infrastructure tách biệt
2. **Testability:** Dễ dàng unit test với mock dependencies
3. **Maintainability:** Code dễ đọc, dễ bảo trì
4. **Type Safety:** Python type hints và Pydantic validation

6.1.3 API Gateway Pattern

1. **Single Entry Point:** Đơn giản hóa client integration
2. **Centralized Auth:** JWT validation ở một nơi
3. **Service Abstraction:** Client không cần biết internal topology
4. **Error Handling:** Consistent error responses

6.2 Hạn chế và vấn đề

6.2.1 Security Issues

1. **CRITICAL - Plain Text Passwords:**

- Passwords không được hash, lưu trữ dạng plain text
- Cực kỳ nguy hiểm cho production
- Cần implement bcrypt/argon2 ngay lập tức

2. CRITICAL - Hardcoded Secrets:

- JWT secret = "SECRET" hardcoded trong code
- Amadeus API credentials trong code
- Cần move tất cả secrets sang environment variables

3. No Rate Limiting: Vulnerable to brute force attacks

4. No HTTPS: Production cần TLS/SSL termination

6.2.2 Architecture Limitations

1. Shared Database: Middleware và Itinerary share PostgreSQL, vi phạm "Database per Service"
2. Synchronous Communication: Chỉ có HTTP/REST, không có message queue
3. No Circuit Breaker: Khi downstream service fail, không có protection
4. No Service Discovery: Static configuration, không dynamic
5. No Distributed Tracing: Khó debug cross-service issues

6.2.3 Missing Features

1. No Caching: Redis configured nhưng chưa implement
2. No Event Bus: Không có async messaging
3. No CRUD Complete: Itinerary service chỉ có Create và Read
4. No Data Validation: Không validate date ranges, time conflicts
5. No Pagination: List endpoints không có pagination

6.3 Bài học kinh nghiệm

6.3.1 Microservices Best Practices

1. Start Simple: Không cần phức tạp hóa ban đầu, có thể bắt đầu với monolith
2. API Gateway Essential: Single entry point rất quan trọng
3. Database per Service: Nên tuân thủ nghiêm ngặt để tránh coupling
4. External API Challenges: Rate limits, timeouts, errors phải handle cẩn thận

6.3.2 Security Lessons

1. **Security First:** Không bao giờ compromise security vì "convenience"
2. **Use Secrets Management:** Vault, AWS Secrets Manager, etc.
3. **Implement Proper Auth:** OAuth2, OpenID Connect cho production

6.3.3 Development Process

1. **Documentation Important:** README chi tiết giúp onboarding dễ dàng
2. **Docker Compose Powerful:** Orchestration nhiều services rất hiệu quả
3. **OpenAPI/Swagger:** Auto-generated docs tiết kiệm thời gian

Chương 7

HƯỚNG PHÁT TRIỂN

7.1 Improvements ngắn hạn

7.1.1 Security Enhancements (Priority 1)

Listing 7.1: Implement Password Hashing

```
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash_password(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(plain: str, hashed: str) -> bool:
    return pwd_context.verify(plain, hashed)
```

- Move JWT secret sang environment variables
- Implement rate limiting với slowapi hoặc Redis
- Add HTTPS/TLS với nginx reverse proxy
- Implement refresh tokens

7.1.2 Caching Layer

- Implement Redis caching cho external API responses
- Cache flight search results (TTL: 15 minutes)
- Cache weather data (TTL: 30 minutes)
- Cache destination search (TTL: 1 hour)

7.1.3 Complete CRUD Operations

- Add UPDATE endpoints cho itineraries và activities
- Add DELETE endpoints với soft delete
- Add validation cho date ranges và time conflicts
- Implement pagination cho list endpoints

7.2 Improvements dài hạn

7.2.1 Resilience Patterns

1. Circuit Breaker:

- Implement với pybreaker library
- Prevent cascading failures
- Fallback responses khi service down

2. Retry Logic:

- Exponential backoff cho failed requests
- Configurable retry attempts
- Idempotency keys

3. Bulkhead Pattern:

- Isolate thread pools cho external API calls
- Prevent resource exhaustion

7.2.2 Observability

1. Centralized Logging:

- ELK stack (Elasticsearch, Logstash, Kibana)
- Structured logging với JSON format
- Correlation IDs cho cross-service tracing

2. Distributed Tracing:

- Jaeger hoặc Zipkin
- OpenTelemetry instrumentation
- Visualize request flows

3. Metrics:

- Prometheus cho metrics collection
- Grafana dashboards
- Application metrics (latency, throughput, errors)

7.2.3 Event-Driven Architecture

1. Message Queue:

- RabbitMQ hoặc Apache Kafka
- Async communication giữa services
- Event sourcing pattern

2. Use Cases:

- User created event → sync sang các services
- Itinerary created → send notifications
- Booking confirmed → update multiple systems

7.2.4 Advanced Features

1. Payment Integration:

- Stripe hoặc PayPal integration
- Secure payment processing
- PCI compliance

2. Notification Service:

- Email notifications (SendGrid)
- SMS notifications (Twilio)
- Push notifications

3. Recommendation Engine:

- ML-based destination recommendations
- Collaborative filtering
- Personalized suggestions

7.3 DevOps Improvements

7.3.1 CI/CD Pipeline

Listing 7.2: GitHub Actions CI/CD

```
name: CI/CD Pipeline

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: |
          docker-compose up -d
```

```
pytest tests/  
  
build:  
  runs-on: ubuntu-latest  
  steps:  
    - name: Build and push Docker images  
      run: |  
        docker build -t trip-hub/middleware:latest .  
        docker push trip-hub/middleware:latest  
  
deploy:  
  runs-on: ubuntu-latest  
  needs: [test, build]  
  steps:  
    - name: Deploy to Kubernetes  
      run: kubectl apply -f k8s/
```

7.3.2 Kubernetes Deployment

- Migrate từ Docker Compose sang Kubernetes
- Horizontal Pod Autoscaling
- Load balancing với Ingress
- ConfigMaps và Secrets management
- Health checks và liveness probes

7.3.3 Monitoring & Alerting

- PagerDuty hoặc Opsgenie integration
- Alert rules cho:
 - High error rates
 - Response time degradation
 - Service unavailability
 - Database connection issues

Chương 8

KẾT LUẬN

8.1 Tổng kết

Dự án Trip Hub đã thành công trong việc triển khai một hệ thống microservices hoàn chỉnh với 5 services độc lập, minh họa các nguyên tắc và best practices của kiến trúc phân tán. Hệ thống cung cấp đầy đủ tính năng cho một nền tảng du lịch, từ khám phá điểm đến, kiểm tra thời tiết, đặt vé máy bay và khách sạn, đến quản lý lịch trình chi tiết.

8.1.1 Thành tựu đạt được

1. **Kiến trúc Microservices:** Successfully implement API Gateway pattern, service isolation, và database per service
2. **External API Integration:** Tích hợp với 3 external APIs (Amadeus, OpenWeatherMap, Geoapify) với proper authentication và error handling
3. **Clean Architecture:** Áp dụng separation of concerns, dependency injection, repository pattern trong tất cả services
4. **Containerization:** Hoàn chỉnh Docker Compose orchestration với multiple services và databases
5. **Documentation:** Comprehensive README files và auto-generated OpenAPI documentation

8.1.2 Kiến thức thu được

Qua dự án này, chúng ta đã học được:

- Thiết kế và triển khai microservices từ đầu
- Trade-offs giữa microservices và monolithic architecture
- Challenges trong distributed systems (network latency, partial failures, data consistency)
- API Gateway pattern và centralized authentication
- Working với external APIs và handling rate limits
- Docker containerization và orchestration
- Security best practices (và những gì KHÔNG nên làm)

8.2 Đánh giá khách quan

8.2.1 Điểm mạnh

- Kiến trúc rõ ràng, dễ hiểu
- Code quality tốt với type hints và documentation
- FastAPI framework hiện đại và hiệu quả
- Docker Compose setup đơn giản và functional
- Real-world external API integration

8.2.2 Điểm yếu

- Critical security issues (plain text passwords, hardcoded secrets)
- Thiếu automated testing
- Không có caching layer
- Limited error resilience patterns
- Shared database giữa 2 services

8.3 Khuyến nghị

8.3.1 Cho Development

1. **Fix security issues ngay lập tức** trước khi deploy production
2. **Implement comprehensive testing** (unit, integration, E2E)
3. **Add caching layer** để giảm external API calls
4. **Implement monitoring và logging** để debug production issues

8.3.2 Cho Production Deployment

1. **Kubernetes deployment** thay vì Docker Compose
2. **Proper secrets management** với Vault hoặc AWS Secrets Manager
3. **HTTPS/TLS termination** với nginx hoặc cloud load balancer
4. **Rate limiting và throttling** để protect APIs
5. **Database backups và disaster recovery plan**
6. **Monitoring và alerting** với Prometheus/Grafana

8.4 Lời kết

Trip Hub là một dự án học tập có giá trị, demonstrate kiến thức về microservices architecture, distributed systems, và modern software engineering practices. Mặc dù có một số limitations (đặc biệt về security), dự án đã thành công trong việc minh họa cách xây dựng một hệ thống phân tán functional và có thể mở rộng.

Các kiến thức và kinh nghiệm thu được từ dự án này là nền tảng quan trọng cho việc phát triển các production-grade microservices systems trong tương lai. Với các improvements đã đề xuất, hệ thống hoàn toàn có thể được nâng cấp lên production-ready status.

— *HẾT* —

Phụ lục A

TÀI LIỆU THAM KHẢO

1. FastAPI Documentation: <https://fastapi.tiangolo.com/>
2. Microservices Patterns - Chris Richardson
3. Building Microservices - Sam Newman
4. Amadeus for Developers: <https://developers.amadeus.com/>
5. OpenWeatherMap API: <https://openweathermap.org/api>
6. Docker Documentation: <https://docs.docker.com/>
7. PostgreSQL Documentation: <https://www.postgresql.org/docs/>
8. Python Type Hints: <https://docs.python.org/3/library/typing.html>
9. OAuth 2.0 RFC: <https://tools.ietf.org/html/rfc6749>
10. RESTful API Design Best Practices

Phụ lục B

CODE SNIPPETS QUAN TRỌNG

B.1 API Gateway Proxy Implementation

Listing B.1: Proxy Request Handler

```
async def proxy_request(
    service: str,
    request: Request,
    path: str = ""
) -> Response:
    target_url = service_router.build_target(service, path)
    if not target_url:
        raise HTTPException(404, f"Unknown service '{service}'")

    headers = {k: v for k, v in request.headers.items()
               if k.lower() != "host"}
    body = await request.body()

    try:
        async with httpx.AsyncClient(timeout=10.0) as client:
            upstream_response = await client.request(
                method=request.method,
                url=target_url,
                content=body,
                params=request.query_params,
                headers=headers
            )

        filtered_headers = {
            k: v for k, v in upstream_response.headers.items()
            if k.lower() not in excluded_headers
        }

        return Response(
            content=upstream_response.content,
            status_code=upstream_response.status_code,
            headers=filtered_headers
        )
    except httpx.TimeoutException:
        raise HTTPException(504, f"Request to {service} timed out"
                           )
```

```

    except httpx.HTTPError:
        raise HTTPException(502, f"Error forwarding to {service}")

```

B.2 JWT Authentication

Listing B.2: JWT Token Management

```

from jose import jwt, JWSError
from datetime import datetime, timedelta

SECRET_KEY = "SECRET" # Should be in env vars!
ALGORITHM = "HS256"

def create_access_token(data: dict) -> str:
    to_encode = data.copy()
    expire = datetime.utcnow() + timedelta(hours=1)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def get_current_user(credentials: HTTPAuthorizationCredentials):
    token = credentials.credentials
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username = payload.get("sub")
        if username is None:
            raise HTTPException(401, "Invalid token")
        return {"username": username}
    except JWSError:
        raise HTTPException(401, "Invalid token")

```

B.3 Repository Pattern

Listing B.3: SQLAlchemy Repository

```

from sqlalchemy.orm import Session
import uuid

class ItineraryRepo:
    def __init__(self, session: Session):
        self.session = session

    def create(self, username: str, payload) -> dict:
        id = str(uuid.uuid4())
        item = Itinerary(
            id=id,
            username=username,
            title=payload.title,

```

```
        start_date=payload.start_date,
        end_date=payload.end_date,
        description=payload.description
    )
    self.session.add(item)
    self.session.commit()
    self.session.refresh(item)
    return {
        "id": item.id,
        "user": item.username,
        "title": item.title,
        "start_date": item.start_date.isoformat(),
        "end_date": item.end_date.isoformat(),
        "description": item.description
    }

def list_by_user(self, username: str) -> List[dict]:
    rows = self.session.query(Itinerary) \
        .filter(Itinerary.username == username) \
        .order_by(Itinerary.created_at.desc()) \
        .all()
    return [self._to_dict(row) for row in rows]
```