

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, ĐHQG-HCM

KHOA CÔNG NGHỆ PHẦN MỀM



BÁO CÁO ĐỒ ÁN MÔN HỌC

ĐỀ TÀI:

Xây dựng nền tảng "UIT-Go" Cloud-Native

Môn học: SE360.Q11

Giảng viên hướng dẫn: Lê Văn Tuấn

Thực hiện bởi các thành viên dưới đây, bao gồm:

- Lê Văn Khôi 23520770 Trưởng nhóm
- Lưu Quang Huy 23520619 Thành viên

Thời gian thực hiện: Tháng 09/2025 – Tháng 11/2025

MỤC LỤC

MỤC LỤC	2
TÓM TẮT	4
Chương I. TỔNG QUAN	5
1. Giới thiệu đề tài	5
1.1. Tổng quan	5
1.2. Phạm vi thực hiện	5
2. Cơ sở lý thuyết.....	5
2.1. Kiến trúc Microservices.....	5
2.2. Kiến trúc Microservices.....	6
2.3. Kiến trúc Kubernetes (K8s).....	6
2.4. Azure Cloud Services	6
2.5. Terraform - Infrastructure as Code	6
2.6. CI/CD với GitHub Actions	6
Chương II. THIẾT KẾ HỆ THỐNG.....	8
1. Kiến trúc tổng quan	8
1.1. Các microservices	8
1.2. Luồng nghiệp vụ chính	9
1.3. Các luồng nghiệp vụ của hệ thống.....	10
2. Chi tiết kỹ thuật	40
2.1. Authentication & Authorization.....	40
2.2. Database Design	41
2.3. WebSocket Architecture	41
2.4. Payment Flow với VNPay	41
Chương III. TRIỀN KHAI HỆ THỐNG	41
Giai đoạn 1: "Bộ Xương" Microservices.....	41
1. Chuẩn bị môi trường Azure	41
1.1. Tạo Azure Account và cài đặt công cụ	41
1.2. Tạo Service Principal cho GitHub Actions.....	42
2. Infrastructure as Code với Terraform	42
2.1. Cấu trúc Terraform code	42

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

2.2 Nguyên tắc tổ chức	42
2.3. Các tài nguyên được tạo	50
2.4. Kiến trúc tổng quan sơ đồ K8s	52
3. CI/CD Pipeline với GitHub Actions	53
3.1. Workflow configuration.....	53
3.2. Kubernetes Manifests	56
3.3 Chi tiết các resources	57
4. Local Development với Docker Compose	74
5. Monitoring trên Azure	74
Chương IV. HƯỚNG ĐI MODULE CHUYÊN SÂU	75
1. Tổng quan mô hình	76
2. Phân tích Ban đầu: Mô hình hóa Mối đe dọa (Threat Modeling)	76
3. Trụ cột 1: Kiến trúc Mạng Zero Trust.....	76
3.1 Phân đoạn Mạng (Network Segmentation)	76
3.2 Ân Cơ sở dữ liệu (Private Endpoints)	77
3.3 Tường lửa Ứng dụng Web (WAF)	77
4. Trụ cột 2: Tích hợp Bảo mật CI/CD (Shift-Left).....	77
5. Trụ cột 3: Phòng thủ theo chiều sâu (Defense-in-Depth)	78

TÓM TẮT

Đồ án môn học "Xây dựng nền tảng UIT-Go Cloud-Native" triển khai một hệ thống gọi xe hoàn chỉnh dựa trên kiến trúc microservices và được triển khai trên nền tảng đám mây Azure với Kubernetes (AKS). Hệ thống gồm 5 microservice chính:

- UserService quản lý người dùng và xác thực trên PostgreSQL
- TripService xử lý logic chuyến đi và tích hợp Mapbox API
- DriverService quản lý tài xế,
- LocationService cung cấp khả năng theo dõi vị trí thời gian thực qua WebSocket và Redis
- PaymentService tích hợp công thanh toán VNPay.

Dự án sử dụng Terraform cho Infrastructure as Code, GitHub Actions cho quy trình CI/CD tự động, và CosmosDB (MongoDB API) làm cơ sở dữ liệu chính cho các service. Kiến trúc tổng thể được thiết kế nhằm đảm bảo khả năng mở rộng, độ tin cậy cao và dễ dàng bảo trì hệ thống.

Chương I. TỔNG QUAN.

1. Giới thiệu đề tài.

1.1. Tổng quan

Tên đề tài: Xây dựng nền tảng "UIT-Go" Cloud-Native

Mục tiêu: Xây dựng hệ thống gọi xe hiện đại tương tự Grab/Uber với các tính năng chính:

- Đăng ký, đăng nhập cho hành khách và tài xế với JWT authentication
- Tạo chuyến đi với tính năng ước tính cước phí dựa trên Mapbox API
- Matching tài xế tự động dựa trên vị trí địa lý (Redis GeoSearch)
- Tracking vị trí real-time qua WebSocket
- Thanh toán điện tử tích hợp VNPay
- Quản lý ví tài xế và phân chia doanh thu tự động

Link github: <https://github.com/khoilv2005/SE360>

1.2. Phạm vi thực hiện

Dự án được triển khai hoàn chỉnh trên Azure Cloud với các thành phần:

- **Backend:** 5 microservices viết bằng Python FastAPI
- **Database:** Azure CosmosDB (MongoDB API), Azure PostgreSQL, Azure Redis Cache
- **Container Registry:** Azure Container Registry (ACR)
- **Orchestration:** Azure Kubernetes Service (AKS)
- **CI/CD:** GitHub Actions
- **Infrastructure as Code:** Terraform

2. Cơ sở lý thuyết.

2.1. Kiến trúc Microservices

Microservices là phương pháp phát triển phần mềm trong đó ứng dụng được chia thành các service nhỏ, độc lập, mỗi service chạy trong process riêng và giao tiếp qua các cơ chế nhẹ như HTTP REST API hoặc message queue. Các ưu điểm chính:

- **Độc lập trong phát triển và deploy:** Mỗi team có thể phát triển và deploy service của mình độc lập
- **Khả năng mở rộng linh hoạt:** Scale từng service theo nhu cầu thực tế
- **Fault isolation:** Lỗi ở một service không làm sập toàn hệ thống
- **Technology diversity:** Mỗi service có thể dùng công nghệ phù hợp nhất

2.2. Kiến trúc Microservices

Docker là nền tảng containerization cho phép đóng gói ứng dụng cùng dependencies thành container image. Container cung cấp môi trường chạy nhất quán trên mọi hạ tầng. Trong dự án, mỗi microservice được đóng gói thành Docker image riêng với Dockerfile tối ưu:

- Base image: python3:11-slim (nhẹ nhàng, bảo mật)
- Multi-stage build để giảm kích thước image
- Layer caching để tăng tốc build

2.3. Kiến trúc Kubernetes (K8s)

Kubernetes là hệ thống orchestration mã nguồn mở để tự động hóa deploy, scale và quản lý ứng dụng container. Các khái niệm chính được sử dụng:

- Pod: Đơn vị nhỏ nhất, chứa 1 hoặc nhiều container
- Deployment: Quản lý replica set và rolling update
- Service: Load balancer nội bộ cho pods
- Secret: Quản lý thông tin nhạy cảm (password, API key)
- ConfigMap: Lưu trữ cấu hình ứng dụng

2.4. Azure Cloud Services

Các dịch vụ Azure được sử dụng trong dự án:

- Azure Kubernetes Service (AKS): Managed Kubernetes cluster
- Azure Container Registry (ACR): Private Docker registry
- Azure CosmosDB: NoSQL database với MongoDB API compatibility
- Azure PostgreSQL: Managed relational database
- Azure Redis Cache: In-memory cache với GeoSearch support

2.5. Terraform - Infrastructure as Code

Terraform là công cụ IaC cho phép định nghĩa và quản lý infrastructure bằng code. Ưu điểm:

- Version control cho infrastructure
- Tái tạo môi trường nhanh chóng và nhất quán
- Plan và preview trước khi apply changes
- Multi-cloud support (không bị lock-in)

2.6. CI/CD với GitHub Actions

GitHub Actions cung cấp workflow automation trực tiếp trên GitHub repository. Pipeline CI/CD của dự án gồm 4 giai đoạn:

- **Giai đoạn 1:** test (Unit Test): Chạy pytest để kiểm tra logic code (bỏ qua smoke_test.py) và dừng pipeline nếu code lỗi.
- **Giai đoạn 2:** build (Build): Build 5 Docker image (cho 5 service) và đẩy (push) chúng lên kho Azure Container Registry (ACR).
- **Giai đoạn 3:** deploy (Deploy): Lấy "bí mật" (chuỗi kết nối DB), tạo K8s Secret, và kubectl apply 5 service (đã cập nhật image tag) lên AKS.
- **Giai đoạn 4:** smoke_test (Kiểm thử sau deploy): Lấy IP public động của userservice và chạy script smoke_test.py (test đăng ký/đăng nhập) để xác nhận hệ thống hoạt động.

Chương II. THIẾT KẾ HỆ THỐNG.

1. Kiến trúc tổng quan

Hệ thống UIT-Go được thiết kế theo kiến trúc microservices với 5 service chính, mỗi service có responsibility rõ ràng và database riêng biệt (database per service pattern). Các service giao tiếp với nhau qua HTTP REST API và WebSocket cho real-time communication.

1.1. Các microservices

Service	Chức năng
UserService	Quản lý user authentication, authorization với JWT Cấp Service Token cho inter-service communication Database: Azure PostgreSQL
TripService	Quản lý logic chuyến đi: tạo, matching, tracking status Tích hợp Mapbox API để tính toán tuyến đường và ước tính cước Orchestrate các service khác (LocationService, DriverService, PaymentService) Database: Azure CosmosDB (MongoDB API)
DriverService	Quản lý hồ sơ tài xế và trạng thái (ONLINE/OFFLINE/ON_TRIP) Quản lý ví tài xế (balance, transactions) Cung cấp internal API để TripService truy vấn thông tin tài xế Database: Azure CosmosDB (MongoDB API)
LocationService	Quản lý vị trí tài xế real-time qua WebSocket Lưu trữ GeoLocation trong Redis với GEOSEARCH Tìm kiếm tài xế gần nhất dựa trên bán kính Relay vị trí giữa passenger và driver trong chuyến đi Database: Azure Redis Cache

Service	Chức năng
PaymentService	Tích hợp VNPay payment gateway Xử lý callback và IPN từ VNPay Tự động tính toán hoa hồng (20%) và cộng tiền vào ví tài xế Database: Azure CosmosDB (MongoDB API)

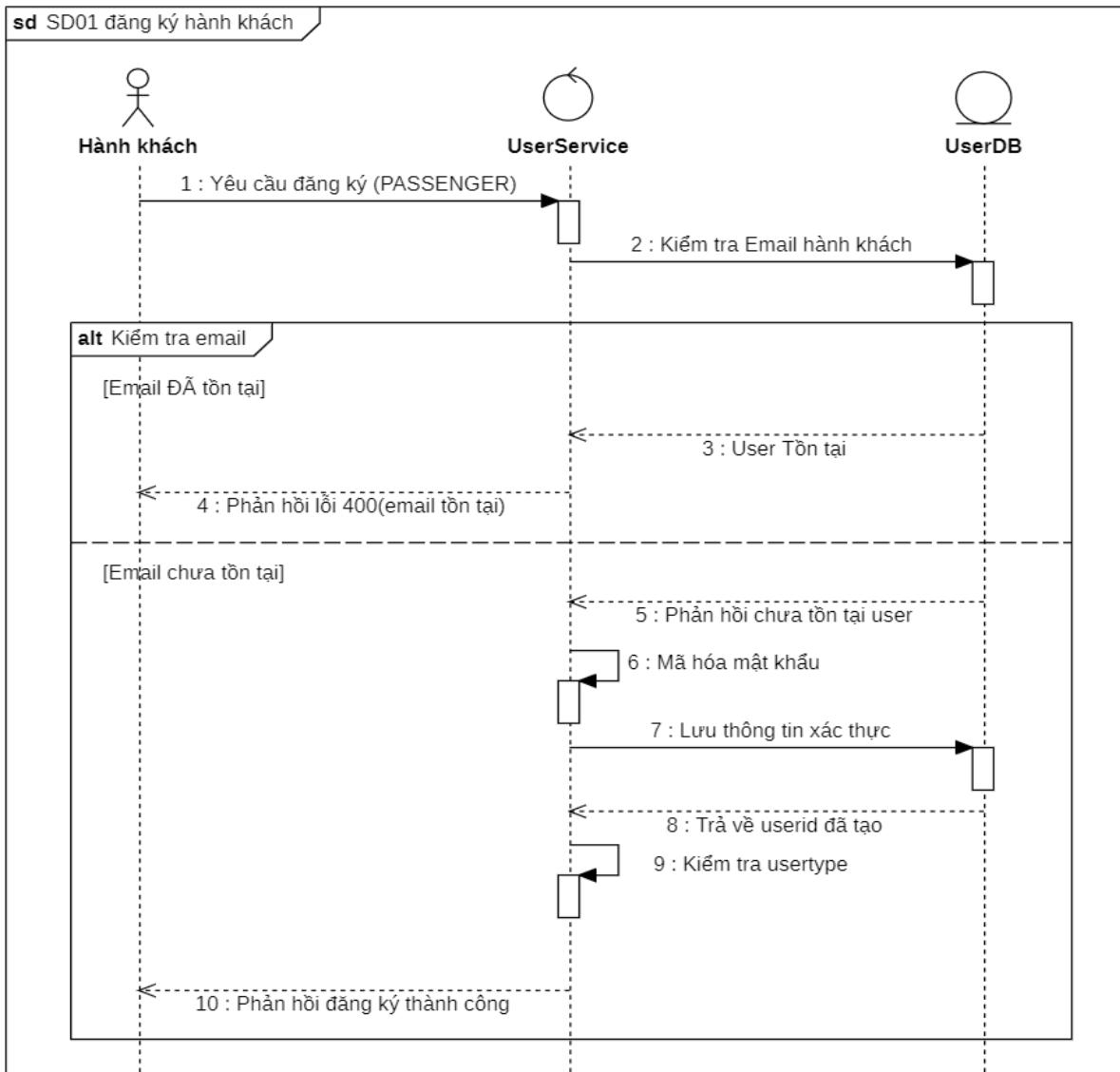
1.2. Luồng nghiệp vụ chính

Luồng tạo và thực hiện chuyến đi:

1. Passenger tạo chuyến đi qua TripService (POST /trip-requests/complete)
2. TripService gọi Mapbox API để tính toán route và estimated fare
3. TripService gọi LocationService để tìm tài xế gần nhất (GET /drivers/nearby)
4. LocationService query Redis GEOSEARCH và trả về danh sách driver_ids
5. TripService gửi thông báo TRIP_OFFER tới các tài xế qua LocationService WebSocket
6. Tài xế đầu tiên accept sẽ được assign (PUT /trips/{id}/assign-driver) - race condition handling
7. TripService lấy thông tin tài xế từ DriverService (GET /drivers/internal/{id}) với Service Token
8. TripService thông báo DRIVER_ASSIGNED cho passenger qua LocationService WebSocket
9. Trong chuyến đi, 2 bên kết nối WebSocket /ws/trip/{id}/{role} để tracking vị trí real-time
10. Khi hoàn thành (POST /trips/{id}/complete), nếu payment_method = 'E-Wallet':
 - TripService gọi PaymentService để tạo link VNPay
 - Passenger thanh toán qua VNPay
 - PaymentService nhận callback, tính hoa hồng (20%), cộng tiền vào ví tài xế

1.3. Các luồng nghiệp vụ của hệ thống

1.3.1. Đăng ký mới tài khoản Hành khách



1.3.1.1. Triển khai mô hình

```

# UserService/main.py

@auth_router.post("/register", response_model=schemas.UserResponse,
status_code=status.HTTP_201_CREATED)
async def register_user(
    user_in: schemas.UserCreate,
    db: AsyncSession = Depends(get_db)
):
    try:
        created_user = await crud.create_user(db=db, user_data=user_in)
        return created_user
    except ValueError as e:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail=str(e))
    except Exception as e:
        logger.error(f"UserService: Lỗi khi tạo user: {e}", exc_info=True)
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Lỗi server khi đăng ký.")

```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# UserService/crud.py

async def get_user_by_email(db: AsyncSession, email: str) -> Optional[User]:
    query = select(UserTable).where(UserTable.email == email)
    result = await db.execute(query)
    user_row = result.scalar_one_or_none()

    if user_row:
        return User.model_validate(user_row)
    return None

# UserService/auth.py

def get_password_hash(password):
    return pwd_context.hash(password)
```

```
db_user = UserTable(**user_data_dict)

try:
    db.add(db_user)
    await db.commit()
    await db.refresh(db_user)
except Exception as e:
    await db.rollback()
    if 'duplicate key value violates unique constraint' in str(e):
        raise ValueError("Số điện thoại này đã được đăng ký.")
    logger.error(f"PostgreSQL: Lỗi khi tạo user: {e}", exc_info=True)
    raise Exception("Không thể tạo user do lỗi database.")

created_user = User.model_validate(db_user)

if created_user.user_type == UserTypeEnum.DRIVER:
    await call_create_driver_profile(
        user_id=created_user.id,
        full_name=created_user.full_name,
        phone=created_user.phone,
        email=created_user.email
    )

return created_user
```

```
# UserService/crud.py

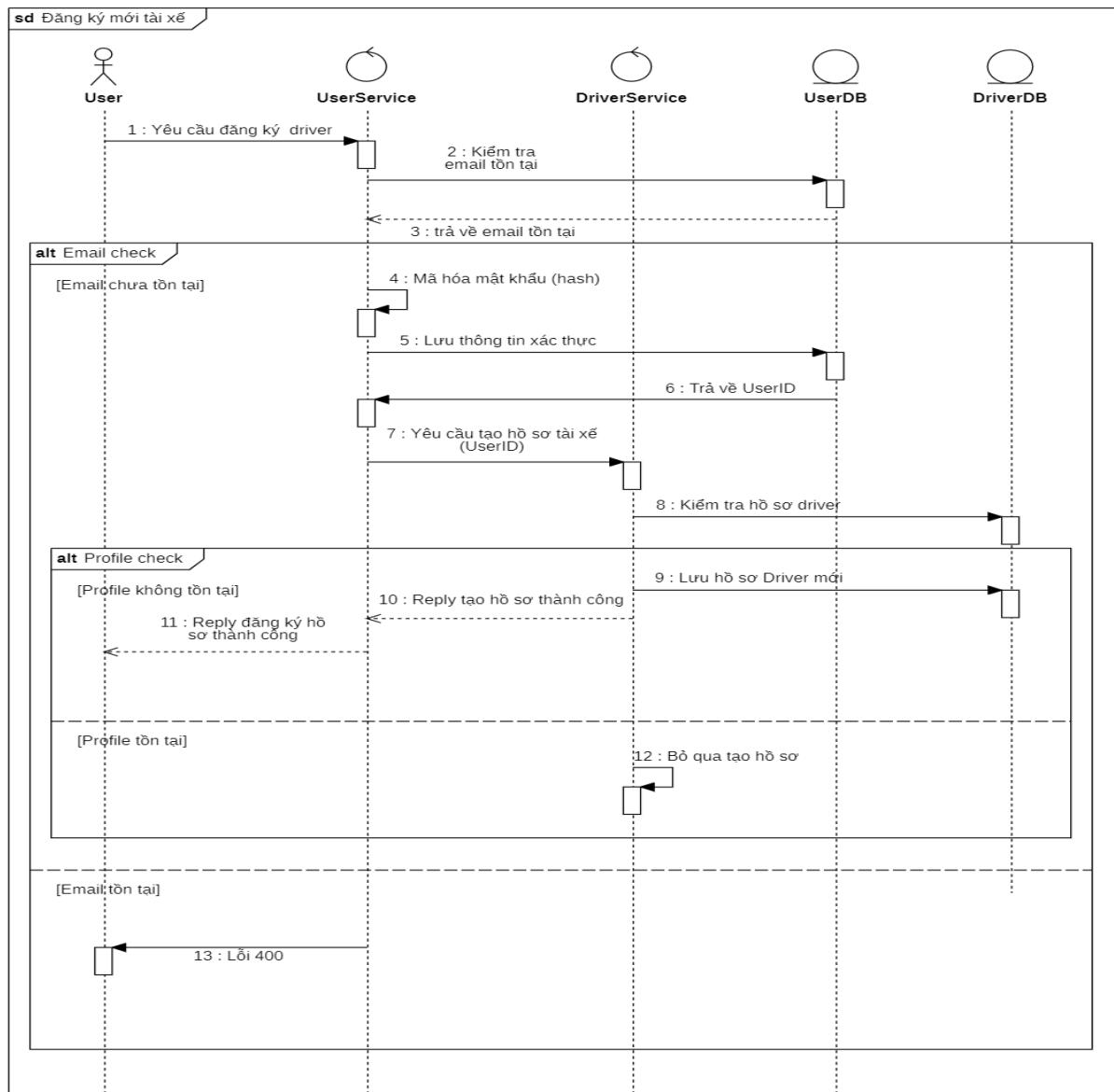
async def get_user_by_email(db: AsyncSession, email: str) -> Optional[User]:
    query = select(UserTable).where(UserTable.email == email)
    result = await db.execute(query)
    user_row = result.scalar_one_or_none()

    if user_row:
        return User.model_validate(user_row)
    return None

# UserService/auth.py

def get_password_hash(password):
    return pwd_context.hash(password)
```

1.3.2. Đăng ký mới tài khoản tài xế



1.3.2.1. Triển khai mô hình

```

# UserService/main.py

@auth_router.post("/register", response_model=schemas.UserResponse,
status_code=status.HTTP_201_CREATED)
async def register_user(
    user_in: schemas.UserCreate,
    db: AsyncSession = Depends(get_db)
):
    try:
        created_user = await crud.create_user(db=db, user_data=user_in)
        return created_user
    except ValueError as e:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail=str(e))
    except Exception as e:
        logger.error(f"UserService: Lỗi khi tạo user: {e}", exc_info=True)
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Lỗi server khi đăng ký.")

```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# UserService/main.py

@auth_router.post("/register", response_model=schemas.UserResponse,
status_code=status.HTTP_201_CREATED)
async def register_user(
    user_in: schemas.UserCreate,
    db: AsyncSession = Depends(get_db)
):
    try:
        created_user = await crud.create_user(db=db, user_data=user_in)
        return created_user
    except ValueError as e:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail=str(e))
    except Exception as e:
        logger.error(f"UserService: Lỗi khi tạo user: {e}", exc_info=True)
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Lỗi server khi
đăng ký.")
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# UserService/crud.py

async def get_user_by_email(db: AsyncSession, email: str) -> Optional[User]:
    query = select(UserTable).where(UserTable.email == email)
    result = await db.execute(query)
    user_row = result.scalar_one_or_none()

    if user_row:
        return User.model_validate(user_row)
    return None

# UserService/crud.py
# (Service-to-Service Call)

async def call_create_driver_profile(user_id: str, full_name: Optional[str], phone: Optional[str],
email: str):
    if not DRIVER_SERVICE_URL:
        logger.error("DRIVER_SERVICE_URL chưa được cấu hình. Bỏ qua việc tạo hồ sơ tài xế.")
        return
    url = f"{DRIVER_SERVICE_URL}/drivers/?user_id={user_id}"
    driver_payload = {
        "name": full_name or "Tài xế mới",
        "phone": phone or "Chưa cập nhật",
        "email": email,
        "vehicle": {
            "license_plate": "Chưa cập nhật",
            "seat_type": 4
        }
    }
    logger.info(f"Đang gọi DriverService để tạo hồ sơ cho user {user_id} tại {url}...")
    try:
        async with httpx.AsyncClient() as client:
            response = await client.post(url, json=driver_payload, timeout=10.0)
            if response.status_code == 201:
                logger.info(f"Tạo hồ sơ tài xế thành công cho user {user_id}.")
            elif response.status_code == 400:
                logger.warning(f"Hồ sơ tài xế cho user {user_id} có thể đã tồn tại.")
            else:
                response.raise_for_status()
    except Exception as e:
        logger.error(f"Lỗi không xác định khi gọi DriverService: {e}")

# UserService/crud.py

async def get_user_by_email(db: AsyncSession, email: str) -> Optional[User]:
    query = select(UserTable).where(UserTable.email == email)
    result = await db.execute(query)
    user_row = result.scalar_one_or_none()

    if user_row:
        return User.model_validate(user_row)
    return None

async def call_create_driver_profile(user_id: str, full_name: Optional[str], phone: Optional[str],
email: str):
    if not DRIVER_SERVICE_URL:
        logger.error("DRIVER_SERVICE_URL chưa được cấu hình. Bỏ qua việc tạo hồ sơ tài xế.")
        return
    url = f"{DRIVER_SERVICE_URL}/drivers/?user_id={user_id}"
    driver_payload = {
        "name": full_name or "Tài xế mới",
        "phone": phone or "Chưa cập nhật",
        "email": email,
        "vehicle": {
            "license_plate": "Chưa cập nhật",
            "seat_type": 4
        }
    }
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
logger.info(f"Đang gọi DriverService để tạo hồ sơ cho user {user_id} tại {url}...")
try:
    async with httpx.AsyncClient() as client:
        response = await client.post(url, json=driver_payload, timeout=10.0)
        if response.status_code == 201:
            logger.info(f"Tạo hồ sơ tài xế thành công cho user {user_id}.")
        elif response.status_code == 400:
            logger.warning(f"Hồ sơ tài xế cho user {user_id} có thẻ đã tồn tại.")
        else:
            response.raise_for_status()
except Exception as e:
    logger.error(f"Lỗi không xác định khi gọi DriverService: {e}")

# UserService/auth.py

def get_password_hash(password):
    return pwd_context.hash(password)

# DriverService/main.py

@app.post("/drivers/", response_model=schemas.DriverResponse, status_code=status.HTTP_201_CREATED)
async def create_driver_profile_endpoint(
    driver_create: schemas.DriverCreate,
    user_id: str
):
    driver = await crud.create_driver_profile(driver_create, user_id)
    if not driver:
        raise HTTPException(status_code=400, detail="Không thể tạo hồ sơ tài xế (có thẻ đã tồn tại hoặc user_id không hợp lệ)")
    return driver

# DriverService/crud.py
# (Sử dụng _id là string UUID từ UserService)

async def create_driver_profile(driver_create: schemas.DriverCreate, user_id_str: str) ->
Optional[models.Driver]:
    logger = logging.getLogger(__name__)
    if drivers_collection is None:
        logger.error("Lỗi: drivers_collection chưa được khởi tạo.")
        return None

    existing = await drivers_collection.find_one({"_id": user_id_str})
    if existing:
        logger.info(f"DriverService: Hồ sơ cho {user_id_str} đã tồn tại.")
        return driver_helper(existing)

    try:
        if hasattr(driver_create.vehicle, 'model_dump'):
            vehicle_data = driver_create.vehicle.model_dump()
        elif isinstance(driver_create.vehicle, dict):
            vehicle_data = driver_create.vehicle
        else:
            logger.error(f"Dữ liệu vehicle không hợp lệ: {driver_create.vehicle}")
            raise TypeError("Dữ liệu vehicle không phải dict hoặc Pydantic model")

        vehicle_data.setdefault("license_plate", "Chưa cập nhật")
        vehicle_data.setdefault("seat_type", 4)
        vehicle_info_obj = models.VehicleInfo(**vehicle_data)

        driver_obj = models.Driver(
            id=user_id_str,
            name=driver_create.name,
            phone=driver_create.phone,
            email=driver_create.email,
            vehicle=vehicle_info_obj,
            status=models.DriverStatusEnum.OFFLINE
        )
    
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

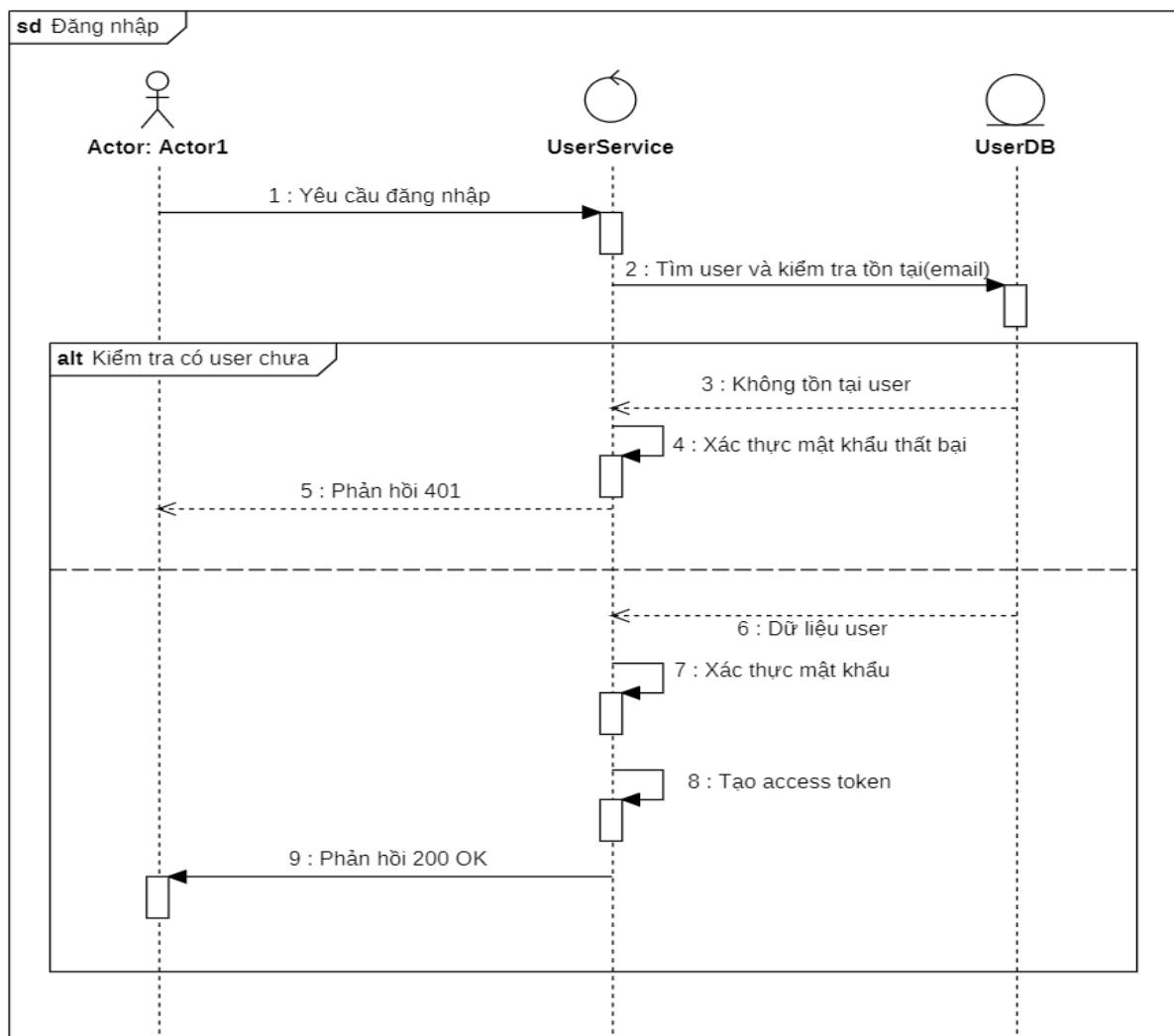
```
except Exception as e:
    logger.error(f'Lỗi khi xử lý dữ liệu đầu vào để tạo Driver: {e}', exc_info=True)
    return None

try:
    driver_doc = driver_obj.model_dump(by_alias=True, exclude={"id"}, exclude_none=True)
    driver_doc["_id"] = user_id_str

    await drivers_collection.insert_one(driver_doc)

    new_doc = await drivers_collection.find_one({"_id": user_id_str})
    if new_doc:
        return driver_helper(new_doc)
    else:
        logger.error(f'Không thể tìm thấy tài xế vừa tạo với ID: {user_id_str}')
        return None
except Exception as e:
    logger.error(f'Lỗi khi lưu tài xế vào database: {e}', exc_info=True)
    return None
```

1.3.3. Đăng nhập



1.3.3.1. Triển khai mô hình

```
#Lưỡng user
#UserService/main.py
@auth_router.post("/login", response_model=schemas.Token)
async def login_for_access_token(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: AsyncSession = Depends(get_db)
):
    user = await crud.get_user_by_email(db=db, email=form_data.username)
    if not user or not auth.verify_password(form_data.password, user.password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Email hoặc mật khẩu không đúng.",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=auth.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = auth.create_access_token(
        data={"sub": user.email},
        expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}
#UserService/crud.py
async def get_user_by_email(db: AsyncSession, email: str) -> Optional[User]:
    query = select(UserTable).where(UserTable.email == email)
    result = await db.execute(query)
    user_row = result.scalar_one_or_none()
    if user_row:
        return User.model_validate(user_row)
    return None
```

```
#UserService/auth.py

def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)
# --- Tạo JWT ---
def create_access_token(data: dict, expires_delta: Union[timedelta, None] = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.now(timezone.utc) + expires_delta
    else:
        expire = datetime.now(timezone.utc) + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

```
#UserService/auth.py
def create_service_access_token(data: dict, expires_delta: timedelta | None = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.now(timezone.utc) + expires_delta
    else:
        expire = datetime.now(timezone.utc) + timedelta(minutes=15)

    to_encode.update({"exp": expire})
    to_encode.update({"type": "service"})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

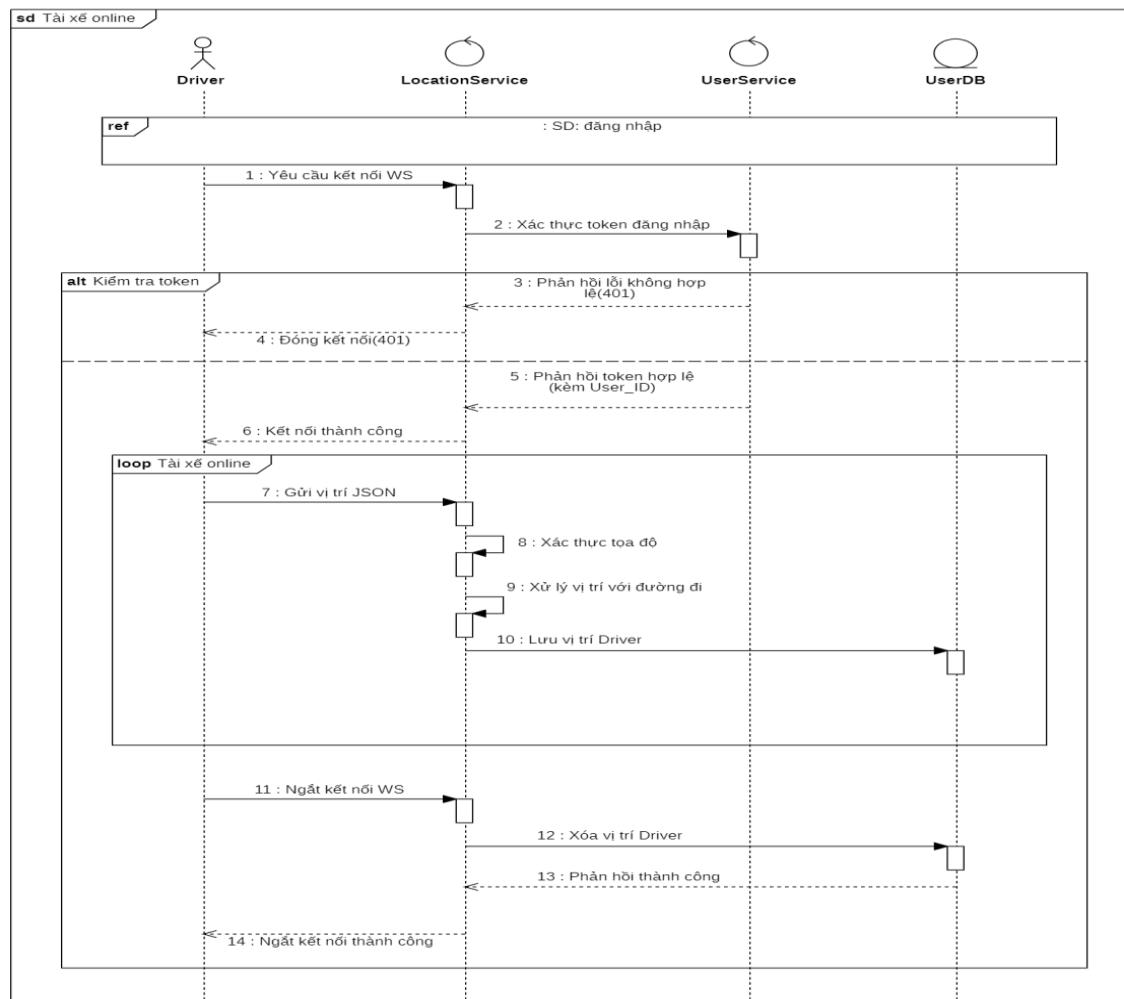
# Luồng service to service
# UserService/main.py

@auth_router.post("/token", response_model=schemas.Token)
async def login_for_service_token(
    form_data: OAuth2PasswordRequestForm = Depends()
):
    client_id = form_data.username
    client_secret = form_data.password
    if (client_id == EXPECTED_TRIPSVC_CLIENT_ID and
        client_secret == EXPECTED_TRIPSVC_CLIENT_SECRET):
        service_token_data = {
            "sub": client_id,
            "aud": "driveservice"

        }
        service_token = auth.create_service_access_token(data=service_token_data)
        logger.info(f"Đã cấp Service Token cho client: {client_id}")
        return {"access_token": service_token, "token_type": "bearer"}
    else:
        logger.warning(f"Client Credentials không hợp lệ: {client_id}")
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Client Credentials không hợp lệ",
            headers={"WWW-Authenticate": "Bearer"},
        )
#UserService/auth.py
```

1.3.4. Tài xế online

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ



Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# LocationService/main.py

class DriverConnectionManager:
    def __init__(self):
        self.active_drivers: Dict[str, WebSocket] = {}

    async def connect(self, websocket: WebSocket, driver_id: str):
        await websocket.accept()
        self.active_drivers[driver_id] = websocket
        logger.info(f"Tài xế RÀNH {driver_id} đã kết nối WSS.")

    def disconnect(self, driver_id: str):
        if driver_id in self.active_drivers:
            del self.active_drivers[driver_id]
            logger.info(f"Tài xế RÀNH {driver_id} đã ngắt kết nối WSS.")

    async def send_notification(self, driver_id: str, payload: dict):
        websocket = self.active_drivers.get(driver_id)
        if websocket:
            try:
                await websocket.send_json(payload)
                logger.info(f"Đã gửi thông báo cho tài xế {driver_id}: {payload.get('type')}")
                return True
            except Exception as e:
                logger.warning(f"Lỗi khi gửi thông báo cho {driver_id}: {e}")
                self.disconnect(driver_id)
            return False
        else:
            logger.error(f"Không tìm thấy tài xế {driver_id} để gửi thông báo")

    def get_driver_location(self, driver_id: str) -> Optional[LocationUpdate]:
        websocket = self.active_drivers.get(driver_id)
        if websocket:
            try:
                data = await websocket.receive_json()
                location = schemas.LocationUpdate(**data)
                await crud.update_driver_location(
                    driver_id,
                    location.longitude,
                    location.latitude
                )
                return location
            except Exception:
                logger.warning(f"Tài xế {driver_id}: Dữ liệu nhận được không phải định dạng Vị trí: {data}")
                continue

        raise WebSocketDisconnect
        logger.info(f"Tài xế {driver_id} (matching) ngắt kết nối WSS.")
        driver_manager.disconnect(driver_id)
        await crud.remove_driver_location(driver_id)

    def disconnect_all(self):
        for driver_id, websocket in self.active_drivers.items():
            self.disconnect(driver_id)
            await crud.remove_driver_location(driver_id)

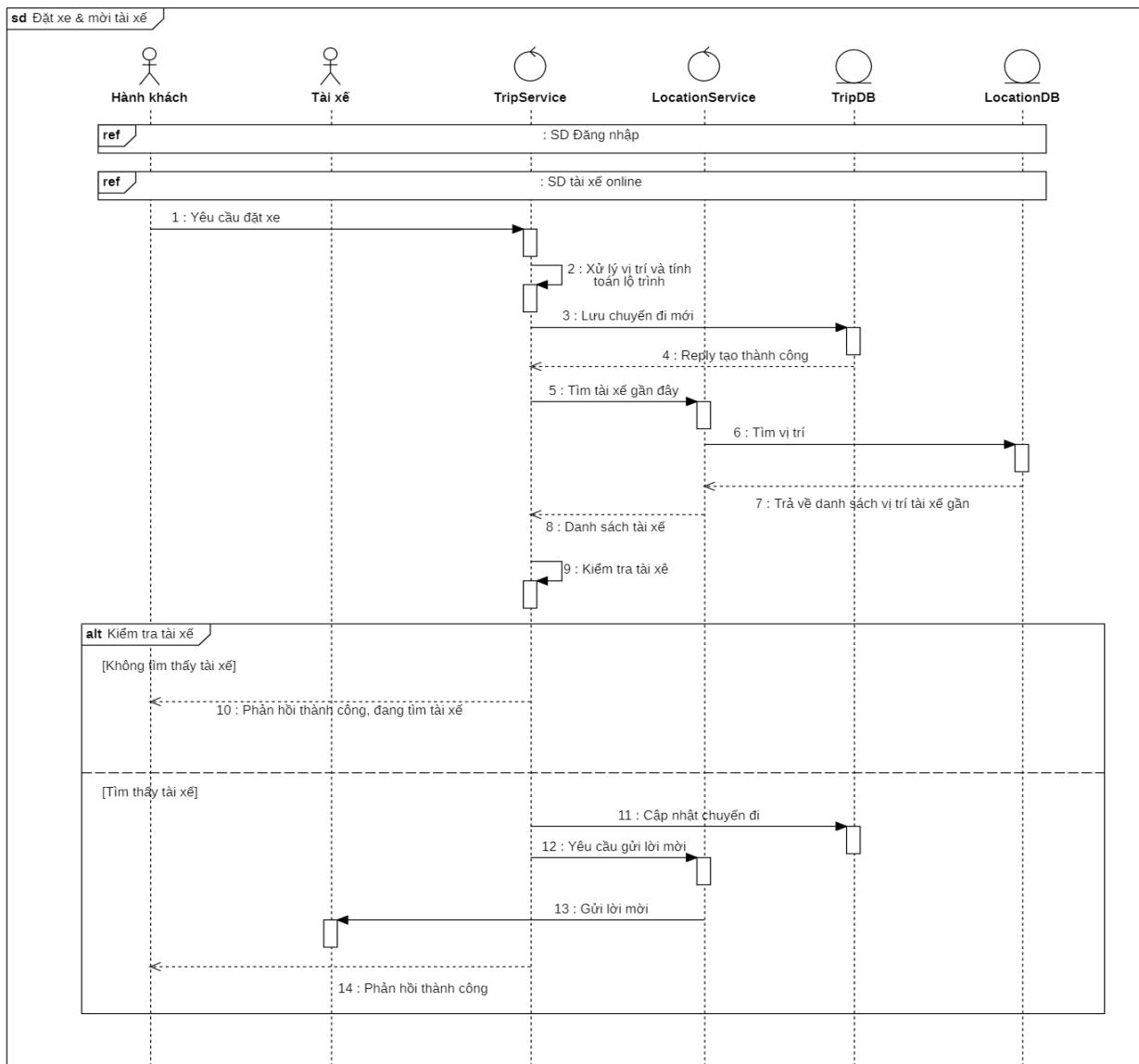
# LocationService/crud.py

async def update_driver_location(driver_id: str, longitude: float, latitude: float):
    if redis_client:
        await redis_client.geoadd(
            DRIVER_GEO_KEY,
            (longitude, latitude, driver_id)
        )

async def remove_driver_location(driver_id: str):
    if redis_client:
        await redis_client.zrem(DRIVER_GEO_KEY, driver_id)
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

1.3.5. Đặt xe và mời tài xế



1.3.5.1.

Triển khai mô hình

```

# TripService/main.py
# (Endpoint chính cho SD-05)

@app.post(
    "/trip-requests/complete/",
    response_model=schemas.TripCreationResponse,
)
async def create_complete_trip_request(trip_request: schemas.TripRequestComplete):
    try:
        trip_data = await crud.create_trip_request_complete(trip_request)
        if not trip_data:
            raise HTTPException(status_code=500, detail="Lỗi không xác định khi tạo chuyến đi.")
        return {"trip": trip_data}

    except ValueError as e:
        logger.error(f"Lỗi khi tạo chuyến đi: {e}")
        raise HTTPException(status_code=400, detail=str(e))
    except HTTPException as http_exc:
        raise http_exc
    except Exception as e:
        logger.error(f"Lỗi không xác định khi tạo chuyến đi: {e}", exc_info=True)
        raise HTTPException(status_code=500, detail="Lỗi server khi tạo chuyến đi.")

```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
async def create_trip_request_complete(trip_request: schemas.TripRequestComplete) -> dict:
    pickup_location = models.LocationInfo(
        address=trip_request.pickup.address,
        location=models.GeoLocation(
            coordinates=[trip_request.pickup.longitude, trip_request.pickup.latitude]
        )
    )
    dropoff_location = models.LocationInfo(
        address=trip_request.dropoff.address,
        location=models.GeoLocation(
            coordinates=[trip_request.dropoff.longitude, trip_request.dropoff.latitude]
        )
    )
    pickup_coords = (trip_request.pickup.longitude, trip_request.pickup.latitude)
    dropoff_coords = (trip_request.dropoff.longitude, trip_request.dropoff.latitude)

    route_info_data = await get_route_info(pickup_coords, dropoff_coords, trip_request.vehicle_type)
    if not route_info_data:
        raise ValueError("Could not calculate route between coordinates")

    route_info = models.RouteInfo(**route_info_data)
    estimated_fare = calculate_estimated_fare(route_info_data["distance"], trip_request.vehicle_type)
    fare_info = models.FareInfo(estimated=estimated_fare)
    payment_info = models.PaymentInfo(
        method=trip_request.payment_method,
        status=models.PaymentStatusEnum.PENDING
    )
    initial_history = [models.StatusHistory(status=models.TripStatusEnum.PENDING)]

    trip_obj = models.Trip(
        passenger_id=trip_request.passenger_id,
        driver_id="",
        vehicle_type=trip_request.vehicle_type,
        status=models.TripStatusEnum.PENDING,
        pickup=pickup_location,
        dropoff=dropoff_location,
        created_at=datetime.now(timezone.utc),
        fare=fare_info,
        route_info=route_info,
        payment=payment_info,
        history=initial_history,
        notes=trip_request.notes,
        notified_driver_ids=[],
        rejected_driver_ids=[],
        offer_sent_at=None
    )
    trip_dict = trip_obj.model_dump(by_alias=True, exclude={"id"})

    try:
        result = await trips_collection.insert_one(trip_dict)
        trip_id = str(result.inserted_id)
        trip_dict["_id"] = trip_id
        logger.info(f"Đã tạo chuyến đi mới với ID: {trip_id}")
    except Exception as e:
        logger.error(f"Lỗi khi insert chuyến đi vào DB: {e}", exc_info=True)
        raise HTTPException(status_code=500, detail="Lỗi server khi tạo chuyến đi.")

    nearby_drivers_raw = await find_nearby_drivers_from_location_service(
        trip_request.pickup.latitude,
        trip_request.pickup.longitude
    )

    if nearby_drivers_raw:
        driver_ids = [driver['driver_id'] for driver in nearby_drivers_raw]
        logger.info(f"Tìm thấy {len(driver_ids)} tài xế gần đó cho chuyến đi {trip_id}. Bắt đầu mời...")
        await add_notified_drivers_to_trip(trip_id, driver_ids)
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
trip_payload = {
    "type": "TRIP_OFFER",
    "trip_id": trip_id,
    "pickup_address": trip_dict["pickup"]["address"],
    "dropoff_address": trip_dict["dropoff"]["address"],
    "estimated_fare": trip_dict["fare"]["estimated"],
    "distance_meters": trip_dict.get("route_info", {}).get("distance")
}

await notify_drivers_via_location_service(driver_ids, trip_payload)

offer_timestamp = datetime.now(timezone.utc)
try:
    await trips_collection.update_one(
        {"_id": ObjectId(trip_id)},
        {"$set": {"offer_sent_at": offer_timestamp}}
    )
    logger.info(f"Đã cập nhật offer_sent_at cho chuyến đi {trip_id}.")
    trip_dict["offer_sent_at"] = offer_timestamp
except Exception as e:
    logger.error(f"Lỗi khi cập nhật offer_sent_at cho chuyến đi {trip_id}: {e}")
else:
    logger.warning(f"Không tìm thấy tài xế nào cho chuyến đi {trip_id} khi tạo.")

final_trip_data = await get_trip_by_id(trip_id)
return final_trip_data if final_trip_data else trip_dict

# TripService/crud.py
# (Hàm gọi Mapbox - Ân chi tiết)

async def get_route_info(pickup_coords: tuple[float, float], dropoff_coords: tuple[float, float],
vehicle_type: models.VehicleTypeEnum) -> dict | None:
    directions_url = "https://api.mapbox.com/directions/v5/mapbox/driving" # (URL bị ẩn)
    coordinates = f'{pickup_coords[0]},{pickup_coords[1]};{dropoff_coords[0]},{dropoff_coords[1]}'
    params = {
        'access_token': MAPBOX_ACCESS_TOKEN,
        'geometries': 'polyline',
        'overview': 'full'
    }
    if vehicle_type == models.VehicleTypeEnum.TWO_SEATER:
        params['exclude'] = 'motorway'

    logger.info(f"Mapbox: Requesting directions from {pickup_coords} to {dropoff_coords}")
    try:
        async with httpx.AsyncClient() as client:
            response = await client.get(f"{directions_url}/{coordinates}", params=params)
            response.raise_for_status()
            data = response.json()
            if data.get("routes"):
                route = data["routes"][0]
                return {
                    "distance": route["distance"],
                    "duration": route["duration"],
                    "geometry": route["geometry"]
                }
            else:
                logger.warning("Mapbox: No routes found")
                return None
    except httpx.RequestError as e:
        logger.error(f"Mapbox API (Directions) error: {e}")
        raise e
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# TripService/crud.py
# (Hàm tính giá)

def calculate_estimated_fare(distance_meters: float, vehicle_type: models.VehicleTypeEnum) -> float:
    distance_km = distance_meters / 1000
    base_fares = {
        models.VehicleTypeEnum.TWO_SEATER: 15000,
        models.VehicleTypeEnum.FOUR_SEATER: 20000,
        models.VehicleTypeEnum.SEVEN_SEATER: 30000
    }
    per_km_rates = {
        models.VehicleTypeEnum.TWO_SEATER: 8000,
        models.VehicleTypeEnum.FOUR_SEATER: 10000,
        models.VehicleTypeEnum.SEVEN_SEATER: 15000
    }
    base_fare = base_fares.get(vehicle_type, 20000)
    per_km_rate = per_km_rates.get(vehicle_type, 10000)
    estimated_fare = base_fare + (distance_km * per_km_rate)
    return round(estimated_fare / 1000) * 1000

# TripService/crud.py
# (Hàm này gọi S2S đến LocationService)

async def find_nearby_drivers_from_location_service(latitude: float, longitude: float) ->
List[Dict[str, Any]]:
    # Service-to-Service Call (1)
    search_radii = [3, 7, 15]
    limit_per_search = 10
    nearby_drivers = []
    for radius_km in search_radii:
        logger.info(f"Dang tim tai xe trong ban kinh {radius_km}km...")
        url = f"{LOCATION_SERVICE_URL}/drivers/nearby"
        params = {
            "latitude": latitude,
            "longitude": longitude,
            "radius_km": radius_km,
            "limit": limit_per_search
        }
        try:
            async with httpx.AsyncClient() as client:
                response = await client.get(url, params=params)
                if response.status_code == 200:
                    nearby_drivers = response.json()
                    logger.info(f"Tim thay {len(nearby_drivers)} tai xe trong ban kinh {radius_km}km.")
                    break
                elif response.status_code == 404:
                    logger.warning(f"Khong tim thay tai xe nao trong ban kinh {radius_km}km. Mở rộng
tim kiem...")
                    continue
                else:
                    response.raise_for_status()
        except httpx.HTTPStatusError as e:
            logger.error(f"Lỗi khi gọi LocationService (HTTP {e.response.status_code}):
{e.response.text}")
            return []
        except httpx.RequestError as e:
            logger.error(f"Không thể kết nối đến LocationService: {e}")
            return []
    return nearby_drivers
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# TripService/crud.py
# (Hàm cập nhật DB)

async def add_notified_drivers_to_trip(trip_id: str, driver_ids: List[str]):
    if not ObjectId.is_valid(trip_id) or not driver_ids:
        return
    await trips_collection.update_one(
        {"_id": ObjectId(trip_id)},
        {"$set": {"notified_driver_ids": driver_ids}}
    )

# TripService/crud.py
# (Hàm này gọi S2S đến LocationService)

async def notify_drivers_via_location_service(driver_ids: List[str], payload: Dict[str, Any]):
    # Service-to-Service Call (2)
    if not driver_ids:
        return
    url = f"{LOCATION_SERVICE_URL}/notify/drivers"
    request_data = {"driver_ids": driver_ids, "payload": payload}
    try:
        async with httpx.AsyncClient() as client:
            response = await client.post(url, json=request_data, timeout=10.0)
            response.raise_for_status()
            logger.info(f"TripService: Đã yêu cầu LocationService thông báo (loại: {payload.get('type')}) cho {len(driver_ids)} tài xế.")
    except httpx.RequestError as e:
        logger.error(f"TripService: Không thể kết nối LocationService (đã thông báo): {e}")
    except httpx.HTTPStatusError as e:
        logger.error(f"TripService: LocationService trả lỗi khi thông báo: {e.response.status_code} - {e.response.text}")
    except Exception as e:
        logger.error(f"TripService: Lỗi không xác định khi thông báo tài xế: {e}")

# LocationService/crud.py
# (Hàm xử lý logic cho S2S Call 1)

async def get_nearby_drivers(longitude: float, latitude: float, radius_km: int, limit: int) -> List[NearbyDriver]:
    if not redis_client:
        return []
    try:
        drivers = await redis_client.geosearch(
            DRIVER_GEO_KEY,
            longitude=longitude,
            latitude=latitude,
            radius=radius_km,
            unit="km",
            withdist=True,
            withcoord=True,
            count=limit,
            sort="ASC"
        )
        result_list = []
        for d in drivers:
            driver_id, distance, (lon, lat) = d
            result_list.append(
                NearbyDriver(
                    driver_id=driver_id,
                    distance_km=round(distance, 2),
                    longitude=lon,
                    latitude=lat
                )
            )
        return result_list
    except Exception as e:
        print(f"LỖI: Không thể thực hiện GEOSEARCH: {e}")
        return []
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# LocationService/main.py
# (Endpoint cho S2S Call 2)

@app.post("/notify/drivers")
async def notify_drivers_endpoint(request: schemas.NotificationRequest = Body(...)):
    if not request.driver_ids:
        logger.warning("NotifyDrivers: Nhận được yêu cầu nhưng không có driver_ids.")
        return {"message": "Không có tài xế nào để thông báo."}

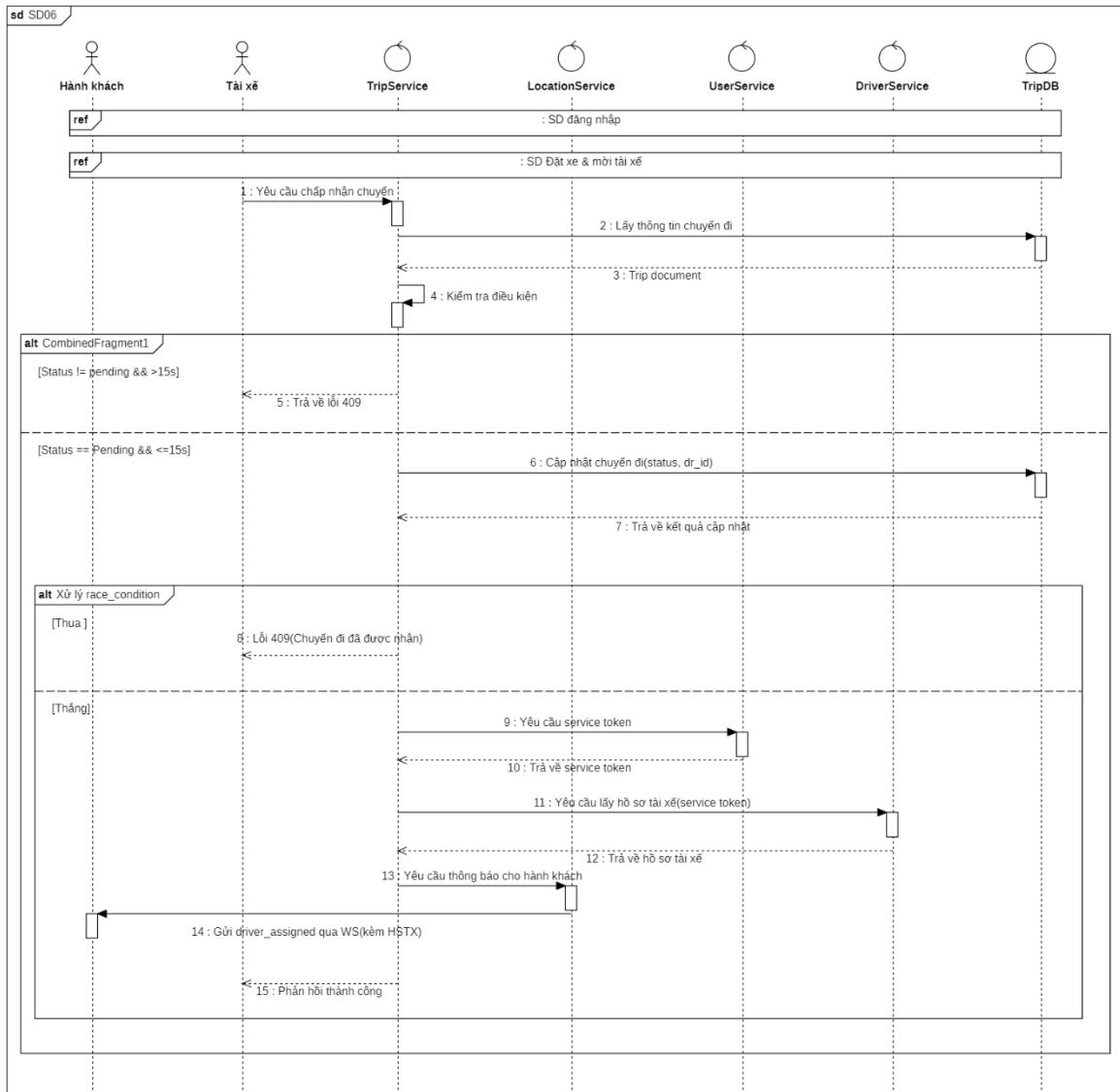
    sent_count = 0
    failed_ids = []

    logger.info(f"NotifyDrivers: Bắt đầu gửi thông báo '{request.payload.get('type')}' đến
{len(request.driver_ids)} tài xế.")
    for driver_id in list(request.driver_ids):
        success = await driver_manager.send_notification(driver_id, request.payload)
        if success:
            sent_count += 1
        else:
            failed_ids.append(driver_id)

    logger.info(f"NotifyDrivers: Gửi thành công {sent_count}/{len(request.driver_ids)}. Thất bại:
{failed_ids}")
    return {
        "message": f"Đã gửi thông báo cho {sent_count} tài xế.",
        "sent_count": sent_count,
        "failed_driver_ids": failed_ids
    }
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

1.3.6. Tài xế chấp nhận chuyến



1.3.6.1. Triển khai mô hình

```
# TripService/main.py
# (Endpoint chính cho SD-06)

@app.put("/trips/{trip_id}/assign-driver", response_model=schemas.TripResponse)
async def assign_driver(trip_id: str, assign_data: schemas.AssignDriver):
    trip_data = await crud.assign_driver_to_trip(trip_id, assign_data.driver_id)
    if trip_data is None:
        raise HTTPException(status_code=409, detail="Chuyến đi không hợp lệ, đã được nhận, bị hủy hoặc
đã quá hạn chấp nhận.")
    return schemas.TripResponse(**trip_data)
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# TripService/crud.py
# (Hàm "Nhạc trưởng" Orchestrator cho SD-06)

async def assign_driver_to_trip(trip_id: str, driver_id: str) -> Optional[dict]:
    if not ObjectId.is_valid(trip_id):
        logger.warning(f"assign_driver_to_trip: trip_id không hợp lệ: {trip_id}")
        return None
    try:
        current_trip = await trips_collection.find_one({"_id": ObjectId(trip_id)})
        if not current_trip:
            logger.warning(f"Tài xế {driver_id} có nhận chuyến {trip_id} không tồn tại.")
            return None
    except Exception as e:
        logger.error(f"Lỗi khi lấy thông tin chuyến đi {trip_id} để kiểm tra: {e}")
        return None

    current_status = current_trip.get("status")
    if current_status != models.TripStatusEnum.PENDING.value:
        logger.warning(f"Tài xế {driver_id} có nhận chuyến {trip_id} không còn PENDING (status: {current_status}).")
        return None

    offer_sent_time = current_trip.get("offer_sent_at")
    if offer_sent_time:
        if offer_sent_time.tzinfo is None:
            offer_sent_time = offer_sent_time.replace(tzinfo=timezone.utc)

        time_now = datetime.now(timezone.utc)
        time_elapsed = time_now - offer_sent_time
        acceptance_limit = timedelta(seconds=16)

        if time_elapsed > acceptance_limit:
            logger.warning(f"Tài xế {driver_id} có nhận chuyến {trip_id} QUÁ HẠN {acceptance_limit.total_seconds()} giây ({time_elapsed.total_seconds():.1f}s).")
            return None
        else:
            logger.info(f"Tài xế {driver_id} chấp nhận chuyến {trip_id} trong thời hạn ({time_elapsed.total_seconds():.1f}s).")
    else:
        logger.warning(f"Chuyến đi {trip_id} thiếu 'offer_sent_at'. Bỏ qua kiểm tra thời gian.")

    new_history_entry = {
        "status": models.TripStatusEnum.ACCEPTED.value,
        "timestamp": datetime.now(timezone.utc)
    }
    try:
        result = await trips_collection.update_one(
            {"_id": ObjectId(trip_id), "status": models.TripStatusEnum.PENDING.value},
            {
                "$set": {
                    "driver_id": driver_id,
                    "status": models.TripStatusEnum.ACCEPTED.value
                },
                "$push": {"history": new_history_entry}
            }
        )
    except Exception as e:
        logger.error(f"Lỗi khi update_one để gán tài xế {driver_id} cho chuyến {trip_id}: {e}")
        return None

    if result.modified_count == 0:
        logger.warning(f"Tài xế {driver_id} THẤT BẠI khi nhận chuyến {trip_id} (Race condition - người khác nhanh hơn).")
        return None

    logger.info(f"Tài xế {driver_id} THÀNH CÔNG nhận chuyến {trip_id} (Thắng race condition).")
    updated_trip = await get_trip_by_id(trip_id)
    if not updated_trip:
        return None

    notified_ids = updated_trip.get("notified_driver_ids", [])
    winner_id = driver_id
    loser_ids = [id for id in notified_ids if id != winner_id]
    if loser_ids:
        logger.info(f"Thông báo 'TRIP_CANCELLED' cho {len(loser_ids)} tài xế thua cuộc.")
        cancel_payload = {"type": "TRIP_CANCELLED", "trip_id": trip_id, "reason": "Đã được tài xế khác nhận"}
        await notify_drivers_via_location_service(loser_ids, cancel_payload)

    logger.info(f"Lấy thông tin tài xế {winner_id} để báo cho hành khách.")
    driver_details = await get_driver_details_from_driver_service(winner_id)
    if driver_details is None:
        driver_details = {"name": "Tài xế", "vehicle": {"license_plate": "N/A"}}

    return driver_details
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# DriverService/main.py
# (Endpoint nhận S2S Call 2)

@app.get(
    "/drivers/internal/{driver_id}",
    response_model=schemas.DriverResponse,
    dependencies=[Depends(verify_service_jwt)])
)
async def get_driver_internal(driver_id: str):
    logger.info(f"Yêu cầu nội bộ (JWT hợp lệ): Lấy thông tin cho tài xế {driver_id}")

    driver = await crud.get_driver_by_id(driver_id)

    if not driver:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Không tìm thấy tài xế với ID này"
        )
    return driver

# DriverService/crud.py
# (Hàm xử lý logic cho S2S Call 2)

async def get_driver_by_id(driver_id_str: str) -> Optional[models.Driver]:
    if drivers_collection is None: return None

    driver_data = await drivers_collection.find_one({"_id": driver_id_str})
    if driver_data:
        return driver_helper(driver_data)

    logger.warning(f"DriverService: Không tìm thấy tài xế với _id string: {driver_id_str}")
    return None

# TripService/crud.py
# (Hàm này gọi S2S đến LocationService)

async def notify_passenger_via_location_service(trip_id: str, payload: Dict[str, Any]):
    # Service-to-Service Call (3)
    url = f"{LOCATION_SERVICE_URL}/notify/trip/{trip_id}/passenger"
    request_data = {"payload": payload}

    try:
        async with httpx.AsyncClient() as client:
            response = await client.post(url, json=request_data, timeout=10.0)
            response.raise_for_status()
            logger.info(f"TripService: Đã yêu cầu LocationService thông báo cho hành khách (chuyến {trip_id}, loại: {payload.get('type')}).")
    except Exception as e:
        logger.error(f"TripService: Lỗi khi thông báo hành khách: {e}")

# LocationService/main.py
# (Endpoint nhận S2S Call 3)

@app.post("/notify/trip/{trip_id}/{user_type}")
async def notify_trip_participant(
    trip_id: str,
    user_type: str,
    request: schemas.SingleNotificationRequest = Body(...)
):
    if user_type == "passenger":
        await trip_manager.broadcast_to_passenger(trip_id, request.payload)
        logger.info(f"NotifTrip: Đã gửi thông báo cho passenger trong phòng {trip_id}")
        return {"message": f"Đã gửi thông báo cho passenger trong phòng {trip_id}"}
    elif user_type == "driver":
        await trip_manager.broadcast_to_driver(trip_id, request.payload)
        logger.info(f"NotifTrip: Đã gửi thông báo cho driver trong phòng {trip_id}")
        return {"message": f"Đã gửi thông báo cho driver trong phòng {trip_id}"}
    else:
        raise HTTPException(status_code=400, detail="user_type phải là 'driver' hoặc 'passenger'")
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

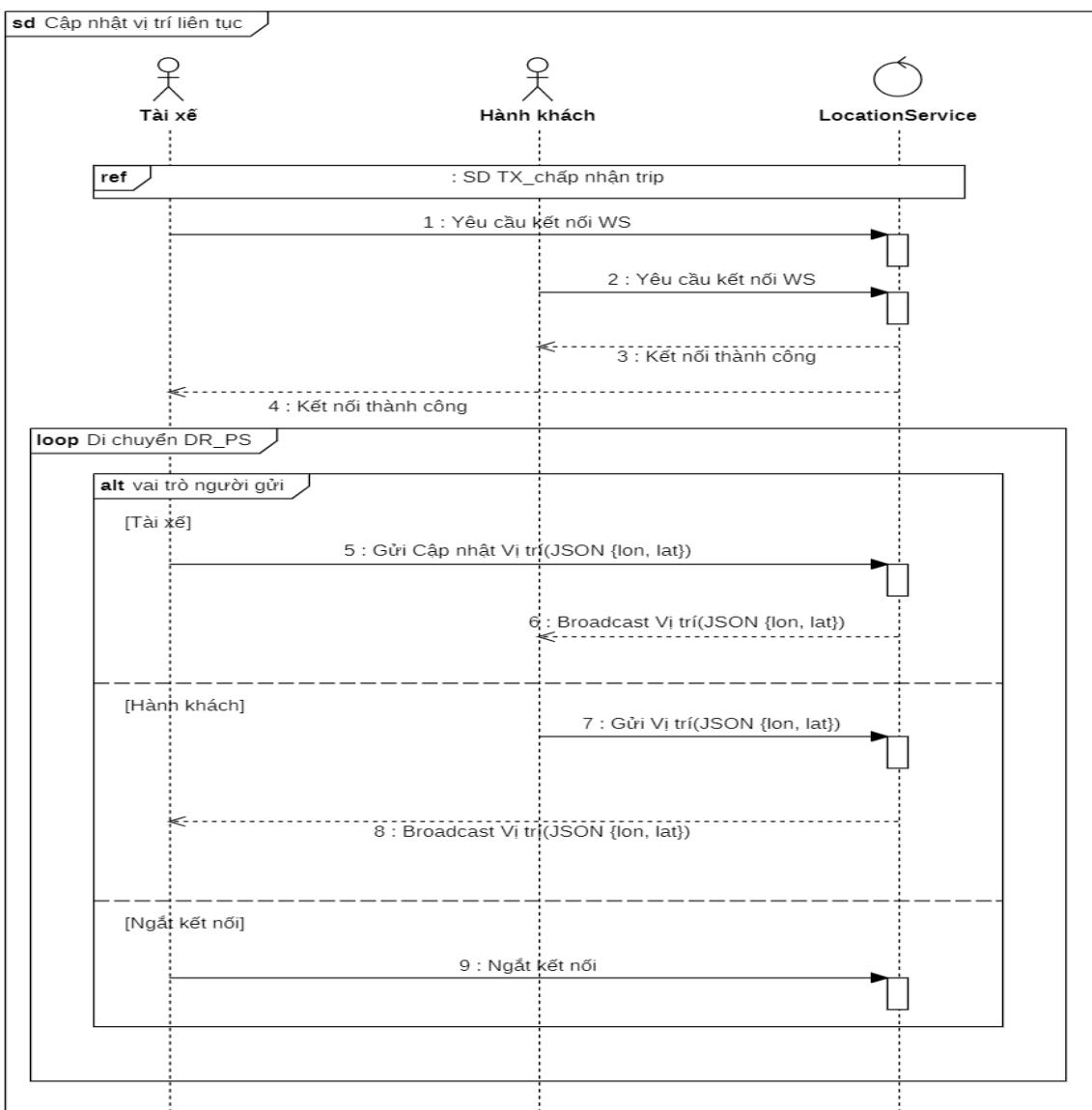
```
# LocationService/main.py
# (Hàm xử lý logic cho S2S Call 3)

class TripConnectionManager:
    # ... (hàm init, connect, disconnect ...)
    async def broadcast_to_passenger(self, trip_id: str, message: dict):
        if trip_id in self.active_rooms:
            passenger_ws = self.active_rooms[trip_id].get("passenger")
            if passenger_ws:
                await passenger_ws.send_json(message)

    async def broadcast_to_driver(self, trip_id: str, message: dict):
        if trip_id in self.active_rooms:
            driver_ws = self.active_rooms[trip_id].get("driver")
            if driver_ws:
                await driver_ws.send_json(message)

trip_manager = TripConnectionManager()
```

1.3.7. Cập nhật vị trí liên tục



1.3.7.1

Triển khai mô hình

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# LocationService/main.py

class TripConnectionManager:
    def __init__(self):
        self.active_rooms: Dict[str, Dict[str, WebSocket]] = {}

    async def connect(self, websocket: WebSocket, trip_id: str, user_type: str):
        await websocket.accept()
        if trip_id not in self.active_rooms:
            self.active_rooms[trip_id] = {}
        self.active_rooms[trip_id][user_type] = websocket
        logger.info(f"Phòng {trip_id}: {user_type} đã kết nối.")

    def disconnect(self, trip_id: str, user_type: str):
        if trip_id in self.active_rooms and user_type in self.active_rooms[trip_id]:
            del self.active_rooms[trip_id][user_type]
            if not self.active_rooms[trip_id]:
                del self.active_rooms[trip_id]
        logger.info(f"Phòng {trip_id}: {user_type} đã ngắt kết nối.")

    async def broadcast_to_passenger(self, trip_id: str, message: dict):
        if trip_id in self.active_rooms:
            passenger_ws = self.active_rooms[trip_id].get("passenger")
            if passenger_ws:
                await passenger_ws.send_json(message)

    async def broadcast_to_driver(self, trip_id: str, message: dict):
        if trip_id in self.active_rooms:
            driver_ws = self.active_rooms[trip_id].get("driver")
            if driver_ws:
                await driver_ws.send_json(message)

    trip_manager = TripConnectionManager()

@app.websocket("/ws/trip/{trip_id}/{user_type}")
@app.websocket("/ws/trip/{trip_id}/{user_type}")
async def ws_trip_tracking(websocket: WebSocket, trip_id: str, user_type: str):
    if user_type not in ["driver", "passenger"]:
        logger.warning(f"Kết nối thất bại: user_type không hợp lệ '{user_type}'")
        return

    await trip_manager.connect(websocket, trip_id, user_type)

    try:
        while True:
            data = await websocket.receive_json()

            try:
                location = schemas.LocationUpdate(**data)
            except Exception:
                logger.warning(f"Phòng {trip_id}: Dữ liệu vị trí sai định dạng: {data}")
                continue
```

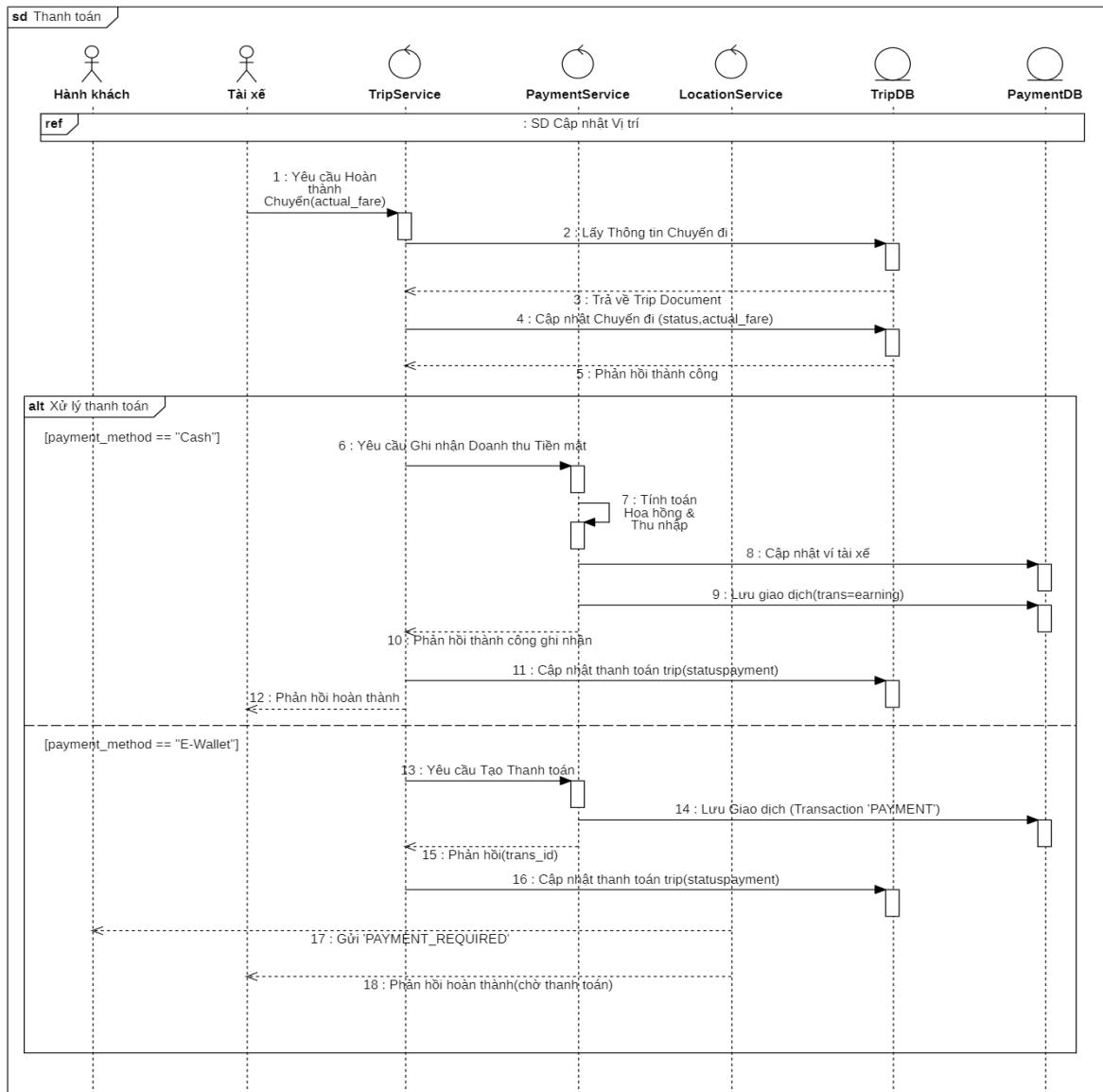
Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```

        if user_type == "driver":
            await trip_manager.broadcast_to_passenger(trip_id,
location.model_dump())
        elif user_type == "passenger":
            await trip_manager.broadcast_to_driver(trip_id, location.model_dump())

    except WebSocketDisconnect:
        trip_manager.disconnect(trip_id, user_type)
    except Exception as e:
        logger.error(f'Lỗi WebSocket chuyển đi {trip_id} ({user_type}): {e}')
        trip_manager.disconnect(trip_id, user_type)
    
```

1.3.8. Hoàn thành trip và thanh toán



1.3.8.1

Triển khai mô hình

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# TripService/main.py

@app.post("/trips/{trip_id}/complete")
async def complete_trip(trip_id: str, data: dict = Body(...)):
    actual_fare = data.get("actual_fare")
    if actual_fare is None:
        raise HTTPException(status_code=400, detail="actual_fare is required in the request body")

    trip = await crud.get_trip_by_id(trip_id)
    if trip is None:
        raise HTTPException(status_code=404, detail="Trip not found")

    payment_method = trip.get("payment", {}).get("method", "Cash")

    if payment_method == "E-Wallet":
        payment_request_data = {
            "trip_id": trip_id,
            "user_id": trip.get("passenger_id"),
            "driver_id": trip.get("driver_id"),
            "amount": actual_fare
        }

    PAYMENT_SERVICE_URL = os.getenv("PAYMENT_SERVICE_URL")
    if not PAYMENT_SERVICE_URL:
        raise HTTPException(status_code=500, detail="PAYMENT_SERVICE_URL is not configured")

    try:
        async with httpx.AsyncClient() as client:
            response = await client.post(
                f"{PAYMENT_SERVICE_URL}/process-payment", json=payment_request_data, timeout=20.0
            )
            response.raise_for_status()
            payment_result = response.json()
    except (httpx.RequestError, httpx.TimeoutException):
        raise HTTPException(status_code=503, detail="Could not connect to Payment Service")
    except httpx.HTTPStatusError as e:
        raise HTTPException(status_code=e.response.status_code, detail=e.response.json())

    await crud.update_trip_status(trip_id, models.TripStatusEnum.COMPLETED)
    await crud.update_trip_fare(trip_id, actual_fare)

    return {
        "message": "Trip completed and payment processed successfully",
        "payment_details": payment_result
    }

else:
    # Service-to-Service (Cash payment flow)
    await crud.update_trip_status(trip_id, models.TripStatusEnum.COMPLETED)
    await crud.update_trip_fare(trip_id, actual_fare)

    return {"message": "Trip completed (Cash payment)"}

# TripService/crud.py

async def update_trip_status(trip_id: str, new_status: models.TripStatusEnum) -> Optional[dict]:
    if not ObjectId.is_valid(trip_id):
        return None

    current_trip = await get_trip_by_id(trip_id)
    if not current_trip:
        return None
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
new_history_entry = {
    "status": new_status.value,
    "timestamp": datetime.now()
}

set_data = {
    "status": new_status.value
}

if new_status == models.TripStatusEnum.ON_TRIP:
    set_data["startTime"] = datetime.now()
elif new_status == models.TripStatusEnum.COMPLETED:
    set_data["endTime"] = datetime.now()

result = await trips_collection.update_one(
    {"_id": ObjectId(trip_id)},
    {
        "$set": set_data,
        "$push": {"history": new_history_entry}
    }
)

if result.modified_count:
    return await get_trip_by_id(trip_id)
return None

# TripService/trip_crud.py
# TripService/crud.py

async def update_trip_fare(trip_id: str, actual_fare: float, discount: float = 0, tax: float = 0) -> Optional[dict]:
    if not ObjectId.is_valid(trip_id):
        return None

    update_data = {
        "fare.actual": actual_fare,
        "fare.discount": discount,
        "fare.tax": tax
    }

    result = await trips_collection.update_one(
        {"_id": ObjectId(trip_id)},
        {"$set": update_data}
    )

    if result.modified_count:
        return await get_trip_by_id(trip_id)
    return None

# PaymentService/main.py

@app.post("/v1/payment/process", response_model=schemas.PaymentLinkResponse, tags=["Payment"])
async def handle_payment_processing(request: schemas.ProcessPaymentRequest):
    logger.info(f"Nhận yêu cầu xử lý thanh toán cho chuyến đi: {request.trip_id}")
    result = await crud.process_vnpay_payment(request)

    if result.get("status") == "FAILED":
        logger.error(f"Lỗi khi tạo link VNPay: {result.get('message')}")
        raise HTTPException(
            status_code=500,
            detail=result.get("message", "Lỗi không xác định khi xử lý thanh toán")
        )
    return result
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# PaymentService/crud.py

def _get_base_url() -> str:
    explicit_base_url = os.getenv("BASE_URL")
    if explicit_base_url:
        logger.info(f"Sử dụng BASE_URL được cấu hình: {explicit_base_url}")
        if not explicit_base_url.startswith(("http://", "https://")):
            return f"https://{explicit_base_url}"
        return explicit_base_url

    azure_hostname = os.getenv("WEBSITE_HOSTNAME")
    if azure_hostname:
        base_url = f"https://{azure_hostname}"
        logger.info(f"Phát hiện môi trường Azure App Service. Sử dụng BASE_URL: {base_url}")
        return base_url

    local_base_url = "http://localhost:8004"
    logger.warning(f"Không tìm thấy BASE_URL hoặc WEBSITE_HOSTNAME. Mặc định dùng URL local: {local_base_url}. Callback IPN từ VNPAY sẽ không hoạt động.")
    return local_base_url

# PaymentService/crud.py

async def process_vnpay_payment(request: schemas.ProcessPaymentRequest) -> Dict:
    base_url = _get_base_url()

    if not all([VNP_TMN_CODE, VNP_HASH_SECRET, VNP_URL]):
        logger.error("Lỗi: Thiếu cấu hình VNP_TMN_CODE, VNP_HASH_SECRET hoặc VNP_URL.")
        return {"status": "FAILED", "message": "Lỗi cấu hình thanh toán VNPay."}

    try:
        order_id = datetime.now(timezone.utc).strftime("%Y%m%d%H%M%S%f")
        amount = int(request.amount) * 100
        order_desc = f'Thanh toán cho chuyến đi {request.trip_id}'
        ip_addr = '127.0.0.1'

        return_url = f"{base_url}/v1/payment/vnpay_return?order_id={order_id}&trip_id={request.trip_id}"
        ipn_url = f"{base_url}/v1/payment/vnpay_ipn"

        vnp_params = {
            'vnp_Version': '2.1.0',
            'vnp_Command': 'pay',
            'vnp_TmnCode': VNP_TMN_CODE,
            'vnp_Amount': amount,
            'vnp_CurrCode': 'VND',
            'vnp_TxnRef': order_id,
            'vnp_OrderInfo': order_desc,
            'vnp_OrderType': 'other',
            'vnp_Locale': 'vn',
            'vnp_ReturnUrl': return_url,
            'vnp_IpAddr': ip_addr,
            'vnp_CreateDate': datetime.now(timezone.utc).strftime('%Y%m%d%H%M%S'),
            'vnp_IpnURL': ipn_url
        }

        sorted_params = sorted(vnp_params.items())
        sorted_params = sorted(vnp_params.items())
        hash_data_string = "&".join([
            f"{key}={quote_plus(str(value))}" for key, value in sorted_params
        ])
        h = hmac.new(VNP_HASH_SECRET.encode('utf-8'), hash_data_string.encode('utf-8'), hashlib.sha512)
        secure_hash = h.hexdigest()
        vnp_params['vnp_SecureHash'] = secure_hash
        payment_url = VNP_URL + "?" + urlencode(vnp_params, quote_via=quote_plus)

        transactions_coll: Optional[AsyncIOMotorCollection] = await get_transactions_collection()
        if transactions_coll:
            pending_transaction = models.Transaction(
                transaction_id=order_id,
                user_id=request.user_id,
                trip_id=request.trip_id,
                amount=request.amount,
                transaction_type=models.TransactionType.PAYMENT,
                payment_method="VNPAY",
                status=models.TransactionStatus.PENDING,
                vnpay_txnref=order_id
            )
    
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```

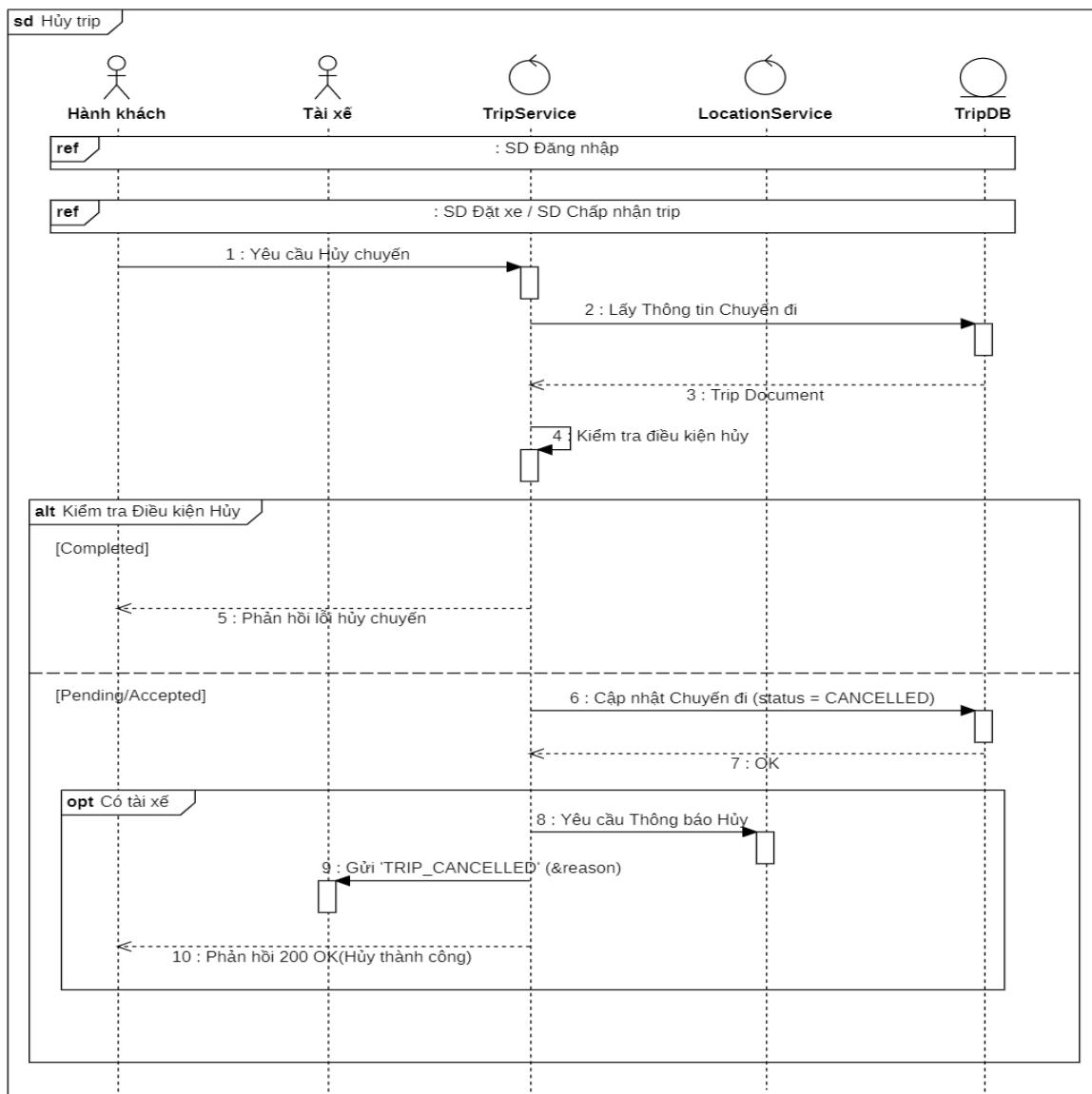
        await transactions_coll.insert_one(pending_transaction.model_dump(by_alias=True, exclude={"id"}))
        logger.info(f"PaymentService: Đã lưu giao dịch PENDING {order_id} cho chuyến {request.trip_id}")
    else:
        logger.error("Lỗi: Không lấy được transactions_collection để lưu giao dịch PENDING.")
        return {"status": "FAILED", "message": "Lỗi hệ thống khi tạo giao dịch."}

    return {"status": "PENDING", "payUrl": payment_url, "transaction_id": order_id}

except Exception as e:
    logger.error(f"Loi khi tao link VNPay: {e}", exc_info=True)
    return {"status": "FAILED", "message": "Có lỗi xảy ra khi tạo yêu cầu thanh toán."}

```

1.3.9 Hủy trip



Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
# TripService/main.py

@app.post("/trips/{trip_id}/cancel")
async def cancel_trip(trip_id: str, cancellation: schemas.CancellationCreate):

    current_trip = await crud.get_trip_by_id(trip_id)
    if not current_trip:
        raise HTTPException(status_code=404, detail="Trip not found")

    current_status = current_trip.get("status")
    allowed_cancel_statuses = [models.TripStatusEnum.PENDING.value,
                                models.TripStatusEnum.ACCEPTED.value]

    if current_status not in allowed_cancel_statuses:
        raise HTTPException(status_code=400, detail=f"Cannot cancel trip in status: {current_status}")

    trip_data = await crud.cancel_trip(trip_id, cancellation)
    if trip_data is None:
        raise HTTPException(status_code=500, detail="Failed to update trip status to CANCELLED")

    driver_id = current_trip.get("driver_id")

    if cancellation.cancelled_by == models.CancelledByEnum.PASSENGER and driver_id:
        logger.info(f"Hành khách hủy chuyến {trip_id}. Thông báo cho tài xế {driver_id}.")
        cancel_payload = {
            "type": "TRIP_CANCELLED",
            "trip_id": trip_id,
            "reason": cancellation.reason or "Chuyến đi đã bị hủy bởi hành khách."
        }
        await crud.notify_driver_in_trip_via_location_service(trip_id, cancel_payload)

    return {"message": "Trip cancelled successfully", "trip_id": trip_id, "status": "CANCELLED"}


# TripService/crud.py

async def cancel_trip(trip_id: str, cancellation: schemas.CancellationCreate) -> Optional[dict]:
    if not ObjectId.is_valid(trip_id):
        return None

    cancellation_data = models.CancellationInfo(
        cancelled_by=cancellation.cancelled_by,
        reason=cancellation.reason,
        cancelled_at=datetime.now()
    )

    update_data = {
        "status": models.TripStatusEnum.CANCELLED.value,
        "cancellation": cancellation_data.dict()
    }

    push_history = {
        "$push": {
            "history": {
                "status": models.TripStatusEnum.CANCELLED.value,
                "timestamp": datetime.now()
            }
        }
    }

    result = await trips_collection.update_one(
        {"_id": ObjectId(trip_id)},
        {"$set": update_data, **push_history}
    )

    if result.modified_count:
        return await get_trip_by_id(trip_id)
    return None
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
async def notify_driver_in_trip_via_location_service(trip_id: str, payload: Dict[str, Any]):
    url = f"{LOCATION_SERVICE_URL}/notify/trip/{trip_id}/driver"
    request_data = {"payload": payload}
    try:
        async with httpx.AsyncClient() as client:
            response = await client.post(url, json=request_data, timeout=10.0)
            response.raise_for_status()
            logger.info(f"TripService: Đã yêu cầu LocationService thông báo cho tài xế (chuyến {trip_id}, loại: {payload.get('type')})")
    except Exception as e:
        logger.error(f"TripService: Lỗi khi thông báo tài xế trong chuyến {trip_id}: {e}")

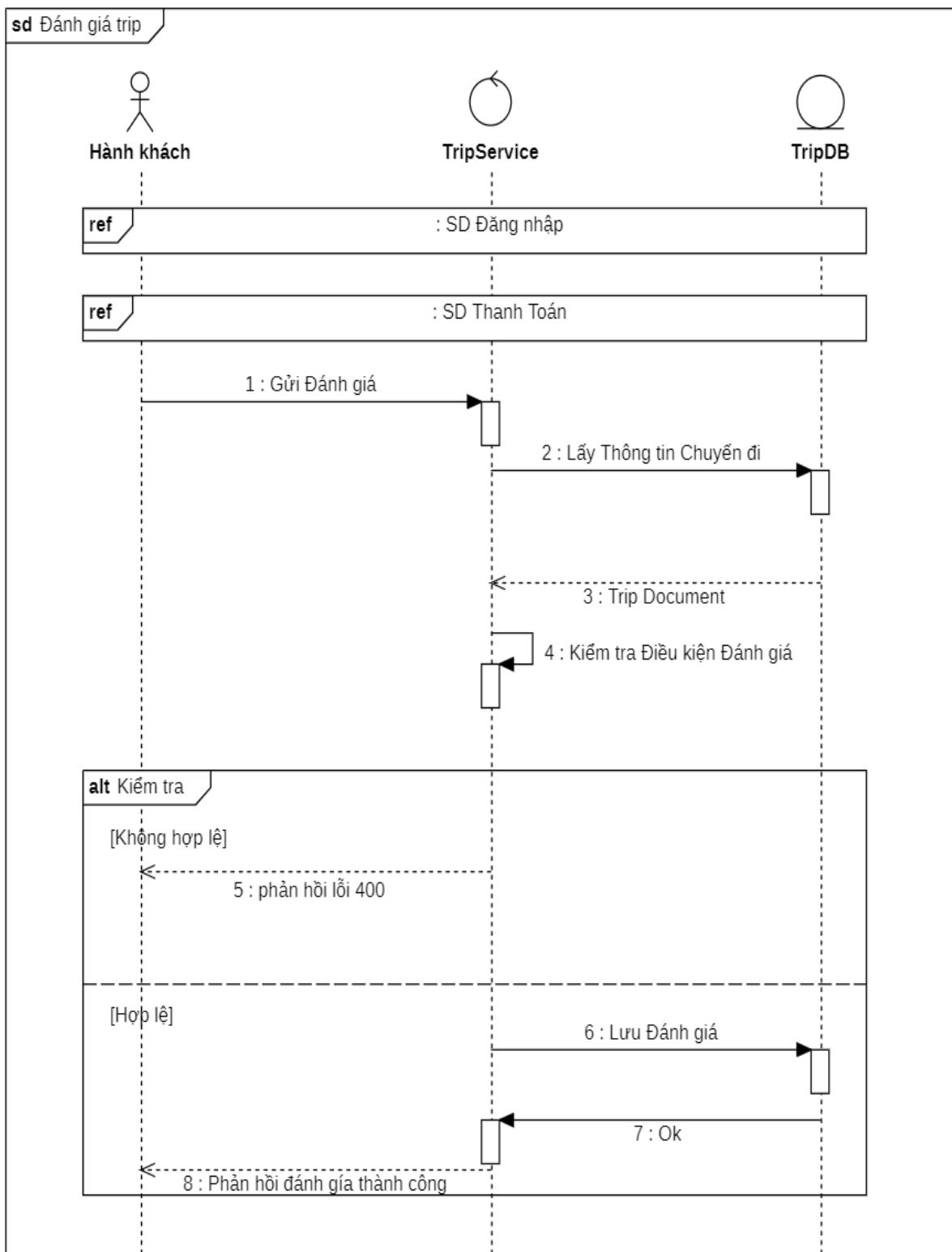
# LocationService/main.py
# (Hàm này đã có từ SD-06)

@app.post("/notify/trip/{trip_id}/{user_type}")
async def notify_trip_participant(
    trip_id: str,
    user_type: str,
    request: schemas.SingleNotificationRequest = Body(...)
):
    if user_type == "passenger":
        await trip_manager.broadcast_to_passenger(trip_id, request.payload)
        logger.info(f"NotifyTrip: Đã gửi thông báo cho passenger trong phòng {trip_id}")
        return {"message": f"Đã gửi thông báo cho passenger trong phòng {trip_id}"}
    elif user_type == "driver":
        await trip_manager.broadcast_to_driver(trip_id, request.payload)
        logger.info(f"NotifyTrip: Đã gửi thông báo cho driver trong phòng {trip_id}")
        return {"message": f"Đã gửi thông báo cho driver trong phòng {trip_id}"}
    else:
        raise HTTPException(status_code=400, detail="user_type phải là 'driver' hoặc 'passenger'")

# LocationService/main.py
# (Hàm này đã có từ SD-07)

async def broadcast_to_driver(self, trip_id: str, message: dict):
    if trip_id in self.active_rooms:
        driver_ws = self.active_rooms[trip_id].get("driver")
        if driver_ws:
            await driver_ws.send_json(message)
```

1.3.10. Đánh giá trip sau khi hoàn thành



```
# TripService/main.py

@app.post("/trips/{trip_id}/rating")
async def add_trip_rating(trip_id: str, rating: schemas.RatingCreate):
    trip_data = await crud.get_trip_by_id(trip_id)
    if trip_data is None:
        raise HTTPException(status_code=404, detail="Trip not found")

    if trip_data["status"] != models.TripStatusEnum.COMPLETED.value:
        raise HTTPException(status_code=400, detail="Can only rate completed trips")

    if trip_data.get("rating"):
        raise HTTPException(status_code=400, detail="Trip already rated")

    updated_trip = await crud.add_trip_rating(trip_id, rating)
    return {"message": "Rating added successfully", "trip_id": trip_id, "rating": rating.stars}

# TripService/crud.py

async def add_trip_rating(trip_id: str, rating: schemas.RatingCreate) -> Optional[dict]:
    if not ObjectId.is_valid(trip_id):
        return None

    rating_data = models.RatingInfo(
        stars=rating.stars,
        comment=rating.comment,
        rated_at=datetime.now()
    )

    result = await trips_collection.update_one(
        {"_id": ObjectId(trip_id)},
        {"$set": {"rating": rating_data.dict()}}
    )

    if result.modified_count:
        return await get_trip_by_id(trip_id)
    return None

# TripService/crud.py
# (Hàm này được gọi ở trên)

async def get_trip_by_id(trip_id: str) -> Optional[dict]:
    if not ObjectId.is_valid(trip_id):
        return None
    doc = await trips_collection.find_one({"_id": ObjectId(trip_id)})
    return convert_objectid(doc)
```

2. Chi tiết kỹ thuật

2.1. Authentication & Authorization

Hệ thống sử dụng 2 loại JWT token:

- **User JWT:** Được cấp sau khi login (POST /auth/login), có claim 'sub' chứa email, dùng cho API của người dùng. Expire sau 30 phút.

- **Service JWT:** Được cấp khi service (ví dụ TripService) cung cấp client credentials (POST /auth/token). Có claim 'type': 'service' và 'aud': 'driveservice'. Dùng cho internal API như /drivers/internal/{id}. Expire sau 15 phút và được cache.

2.2. Database Design

Áp dụng Database per Service pattern:

- **UserService:** Azure PostgreSQL - Relational data cho users với ACID compliance
- **TripService:** Azure CosmosDB (MongoDB API) - Document store cho trips với GeoJSON support
- **DriverService:** Azure CosmosDB (MongoDB API) - Lưu driver profiles và wallets
- **PaymentService:** Azure CosmosDB (MongoDB API) - Transactions và wallet data
- **LocationService:** Azure Redis Cache - In-memory store với GEOSEARCH capability

2.3. WebSocket Architecture

LocationService quản lý 2 loại WebSocket connection:

- **Driver location tracking:** WS /ws/driver/{driver_id}/location - Driver gửi vị trí định kỳ, server lưu vào Redis GEOADD
- **Trip tracking:** WS /ws/trip/{trip_id}/{user_type} - Relay vị trí giữa driver và passenger trong cùng trip room

2.4. Payment Flow với VNPay

1. PaymentService tạo payment URL với secure hash (HMAC-SHA512)
2. VNPay redirect về return_url sau khi user thanh toán (hiển thị message)
3. VNPay gọi IPN (Instant Payment Notification) về server để confirm transaction
4. PaymentService verify secure hash, cập nhật transaction status
5. Tính toán: driver_earning = amount * (1 - 0.20), cộng vào ví tài xế

Chương III. TRIỂN KHAI HỆ THỐNG.

Giai đoạn 1: "Bộ Xương" Microservices

1. Chuẩn bị môi trường Azure

1.1. Tạo Azure Account và cài đặt công cụ

Các bước chuẩn bị:

1. Đăng ký Azure account (Azure for Students hoặc Free Tier)
2. Cài đặt Azure CLI: az --version
3. Cài đặt kubectl: kubectl version --client

4. Cài đặt Docker Desktop
5. Cài đặt Terraform: `terraform version`
6. Login Azure CLI: `az login`

1.2. Tạo Service Principal cho GitHub Actions

Service Principal cung cấp identity để GitHub Actions có thể tương tác với Azure:

```
az ad sp create-for-rbac --name "github-actions-uitgo" --role contributor --scopes /subscriptions/{subscription-id} --sdk-auth
```

Output JSON được lưu vào GitHub Secret với tên AZURE_CREDENTIALS

2. Infrastructure as Code với Terraform

2.1. Cấu trúc Terraform code

Project sử dụng các file Terraform sau:

- **provider.tf:** Cấu hình Azure provider và remote backend (Azure Storage)
- **variables.tf:** Định nghĩa biến (prefix, location)
- **main.tf:** Resource Group, Virtual Network, Subnet, AKS Cluster
- **acr.tf:** Azure Container Registry và role assignment
- **outputs.tf:** Export thông tin cần thiết (ACR login server, AKS credentials)
- **database.tf:** Tạo các database PaaS (Postgres, CosmosDB, Redis) và cấu hình mạng (Private DNS).

2.2 Nguyên tắc tổ chức

- **Separation of concerns:** Mỗi file quản lý một nhóm tài nguyên liên quan
- **Single responsibility:** File database.tf chỉ quản lý databases, không quản lý networking
- **Dependencies:** Terraform tự động xác định thứ tự tạo resource dựa trên dependencies
- **State management:** Remote state lưu trên Azure Storage để team collaboration

2.2.1 Cấu hình provider và backend – provider.tf

File này cấu hình Azure provider và remote backend để lưu trữ Terraform state.

Nội dung file:

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
● ● ●
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 4.0"
    }
  }

  backend "azurerm" {
    resource_group_name  = "rg-uitgo-tfstate"
    storage_account_name = "stuitgotfstate"
    container_name        = "tfstate"
    key                  = "prod.terraform.tfstate"
  }
}

provider "azurerm" {
  features {}
  subscription_id = "d8ece151-084a-418c-a446-0ff133a2d388"
}
```

Hình ảnh file cấu hình provider.tf

Giải thích chi tiết:

Khai báo sử dụng provider azurerm Azure với phiên bản tối thiểu 4.0

Cấu hình backend remote lưu file trạng thái Terraform trên Azure Storage, gồm resource group, storage account, container và tên file state

Thiết lập provider Azure với subscription ID cụ thể

Mục đích: Cho phép quản lý tập trung trạng thái hạ tầng trên Azure, dễ dàng triển khai và duy trì cơ sở hạ tầng dưới dạng mã nguồn (IaC)

2.2.2 Định nghĩa biến - variables.tf

File này định nghĩa tất cả input variables cho Terraform configuration, giúp tái sử dụng code cho nhiều môi trường khác nhau (dev, staging, prod).

Nội dung file:

```
● ● ●
variable "prefix" {
  description = "Tiền tố cho tất cả tài nguyên (ví dụ: uitgo)"
  type        = string
  default     = "uitgo"
}

variable "location" {
  description = "Khu vực Azure (ví dụ: East US, Southeast Asia)"
  type        = string
  default     = "Southeast Asia"
}

variable "db_password" {
  description = "Mật khẩu cho PostgreSQL và MongoDB (nên set qua TF_VAR_db_password hoặc terraform.tfvars)"
  type        = string
  sensitive   = true
}
```

Giải thích chi tiết:

File này khai báo các biến đầu vào cho Terraform như sau:

- prefix: tiền tố để đặt tên các tài nguyên, mặc định "uitgo"
- location: vị trí khu vực triển khai Azure, mặc định "Southeast Asia"
- db_password: mật khẩu của database, được đánh dấu là nhạy cảm nên không hiển thị trong output hoặc log

Mục đích: Giúp cấu hình linh hoạt theo từng môi trường và bảo mật thông tin nhạy cảm. Giá trị biến có thể được truyền từ bên ngoài khi chạy Terraform.

2.2.3 Hạ tầng cốt lõi - main.tf

File này định nghĩa các resource cốt lõi: Resource Group, Virtual Network, Subnet và AKS Cluster.

Resource Group

```
resource "azurerm_resource_group" "rg" {  
    name      = "rg-${var.prefix}-prod"  
    location  = var.location  
}
```

Virtual Network

```
resource "azurerm_virtual_network" "vnet" {  
    name          = "vnet-${var.prefix}-prod"  
    address_space = ["172.16.0.0/16"]  
    location      = azurerm_resource_group.rg.location  
    resource_group_name = azurerm_resource_group.rg.name  
}
```

Subnet cho AKS

```
resource "azurerm_subnet" "aks_subnet" {  
    name          = "snet-aks-prod"  
    resource_group_name = azurerm_resource_group.rg.name  
    virtual_network_name = azurerm_virtual_network.vnet.name  
    address_prefixes  = ["172.16.1.0/24"] #
```

Subnet cho

PostgresSQL

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
● ● ●
resource "azurerm_subnet" "postgres_subnet" {
  name           = "snet-postgres-prod"
  resource_group_name = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes      = ["172.16.2.0/24"]
  delegation {
    name = "fs"
    service_delegation {
      name = "Microsoft.DBforPostgreSQL/flexibleServers"
      actions = [
        "Microsoft.Network/virtualNetworks/subnets/join/action",
      ]
    }
  }
}
```

Azure Kubernetes Service cluster

```
● ● ●
resource "azurerm_kubernetes_cluster" "aks" {
  name           = "aks-${var.prefix}-prod"
  location       = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  dns_prefix     = "${var.prefix}-prod"

  default_node_pool {
    name           = "default"
    node_count     = 1
    vm_size        = "Standard_B2s"
    vnet_subnet_id = azurerm_subnet.aks_subnet.id
  }

  identity {
    type = "SystemAssigned"
  }
  oms_agent {
    log_analytics_workspace_id = azurerm_log_analytics_workspace.logs.id
    msi_auth_for_monitoring_enabled = true
  }
}
```

Log Analytics workspace

```
● ● ●
resource "azurerm_log_analytics_workspace" "logs" {
  name           = "logs-${var.prefix}-prod"
  location       = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  sku            = "PerGB2018"
  retention_in_days  = 30 # Lưu log trong 30 ngày
}
```

Giải thích chi tiết:

Resource Group: Tạo một nhóm tài nguyên (resource group) trên Azure, làm nơi tập trung quản lý tất cả các tài nguyên liên quan. Tên nhóm được đặt theo tiền tố và môi trường (prod), vị trí lấy từ biến đầu vào.

Virtual Network: Tạo một mạng ảo (virtual network) với dải địa chỉ lớn 172.16.0.0/16, dùng để chứa các subnet phục vụ hạ tầng, nằm trong resource group đã tạo.

Subnet cho AKS: Tạo một mạng con (subnet) riêng biệt cho dịch vụ Kubernetes Azure (AKS), dùng dải IP 172.16.1.0/24, gắn vào mạng ảo vnet.

Subnet cho PostgreSQL: Tạo subnet cho dịch vụ PostgreSQL, dải IP 172.16.2.0/24 riêng biệt, có ủy quyền đặc biệt (delegation) cho phép dịch vụ Postgres sử dụng subnet này trong VNet.

Azure Kubernetes Service cluster: Tạo cụm Kubernetes trên Azure, cấu hình node pool 1 node với VM size Standard_B2s sử dụng subnet AKS, quản lý bằng identity hệ thống, kích hoạt agent log analytics để thu thập log.

Log Analytics workspace: Tạo workspace lưu log hệ thống trên Azure Monitor với khu vực, tên, và thời gian lưu trữ log được cấu hình rõ ràng.

2.2.4 Container registry - acr.tf

File này tạo Azure Container Registry (ACR) và cấu hình quyền để AKS có thể pull images.

Container Registry

```
resource "azurerm_container_registry" "acr" {
  name          = "acruitgoprod"
  resource_group_name = azurerm_resource_group.rg.name
  location      = azurerm_resource_group.rg.location
  sku           = "Basic" # Dùng "Basic" cho tiết kiệm
  admin_enabled = true
}
```

Role Assignment cho AKS

```
resource "azurerm_role_assignment" "aks_pull_acr" {
  scope          = azurerm_container_registry.acr.id
  role_definition_name = "AcrPull"
  # Sử dụng kubelet identity của AKS để kéo image từ ACR
  principal_id    = azurerm_kubernetes_cluster.aks.kubelet_identity[0].object_id
}
```

Giải thích chi tiết:

Tạo Azure Container Registry (ACR) tên là "acruitgoprod" với SKU loại "Basic" để lưu trữ các container image, kích hoạt quản trị (admin_enabled = true) để dễ quản lý.

Đặt quyền (role assignment) cho cụm Azure Kubernetes Service (AKS) sử dụng identity kubelet của nó để có thể kéo (pull) container image từ ACR này với vai trò "AcrPull". Điều này giúp cụm AKS tự động lấy image cần thiết để chạy ứng dụng.

2.2.5 Database services - database.tf

File này provisioning tất cả database services: CosmosDB (MongoDB API), PostgreSQL, và Redis Cache, kèm theo cấu hình mạng và Private DNS.

Private DNS Zone cho PostgreSQL

```
resource "azurerm_private_dns_zone" "postgres_dns" {
  name          = "private.postgres.database.azure.com"
  resource_group_name = azurerm_resource_group.rg.name
}
```

Liên kết DNS Zone với Virtual Network

```
resource "azurerm_private_dns_zone_virtual_network_link" "postgres_dns_link" {
  name          = "postgres-dns-link"
  resource_group_name = azurerm_resource_group.rg.name
  private_dns_zone_name = azurerm_private_dns_zone.postgres_dns.name
  virtual_network_id    = azurerm_virtual_network.vnet.id
}
```

Azure Database for PostgreSQL Flexible Server

```
resource "azurerm_postgresql_flexible_server" "postgres" {
  name          = "psql-${var.prefix}-prod"
  resource_group_name = azurerm_resource_group.rg.name
  location      = azurerm_resource_group.rg.location
  administrator_login  = "postgresadmin"
  administrator_password = var.db_password
  sku_name      = "B_Standard_B1ms"
  version        = "15"
  storage_mb     = 32768
  public_network_access_enabled = false
  delegated_subnet_id      = azurerm_subnet.postgres_subnet.id
  private_dns_zone_id       = azurerm_private_dns_zone.postgres_dns.id
  depends_on   = [azurerm_private_dns_zone_virtual_network_link.postgres_dns_link]
  backup_retention_days     = 7
  geo_redundant_backup_enabled = false
  zone = "1"
}
```

Cơ sở dữ liệu trong PostgreSQL server

```
resource "azurerm_postgresql_flexible_server_database" "postgres_db" {
  name      = "mydb"
  server_id = azurerm_postgresql_flexible_server.postgres.id
  charset   = "UTF8"
  collation = "en_US.utf8"
}
```

CosmosDB MongoDB API

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
resource "azurerm_cosmosdb_account" "cosmos" {
  name          = "cosmos-${var.prefix}-prod"
  location      = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  offer_type    = "Standard"
  kind          = "MongoDB"
  consistency_policy {
    consistency_level = "Session"
  }
  geo_location {
    location      = azurerm_resource_group.rg.location
    failover_priority = 0
  }
  public_network_access_enabled = true
  is_virtual_network_filter_enabled = false
  capabilities {
    name = "EnableMongo"
  }
  capabilities {
    name = "EnableServerless"
  }
}
```

CosmosDB Mongo databases cho từng service

```
resource "azurerm_cosmosdb_mongo_database" "trips_db" {
  name          = "uitgo_trips"
  resource_group_name = azurerm_resource_group.rg.name
  account_name   = azurerm_cosmosdb_account.cosmos.name
}
resource "azurerm_cosmosdb_mongo_database" "drivers_db" {
  name          = "uitgo_drivers"
  resource_group_name = azurerm_resource_group.rg.name
  account_name   = azurerm_cosmosdb_account.cosmos.name
}
resource "azurerm_cosmosdb_mongo_database" "payments_db" {
  name          = "uitgo_payments"
  resource_group_name = azurerm_resource_group.rg.name
  account_name   = azurerm_cosmosdb_account.cosmos.name
}
```

Azure Cache for Redis

```
resource "azurerm_redis_cache" "redis" {
  name          = "redis-${var.prefix}-prod"
  location      = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  capacity      = 0
  family        = "C"
  sku_name      = "Basic"
  minimum_tls_version = "1.2"
  public_network_access_enabled = true
  redis_configuration {
  }
}
```

Firewall rule cho Redis

```
resource "azurerm_redis_firewall_rule" "allow_azure_services" {
  name          = "AllowAzureServices"
  redis_cache_name = azurerm_redis_cache.redis.name
  resource_group_name = azurerm_resource_group.rg.name
  start_ip      = "0.0.0.0"
  end_ip        = "0.0.0.0"
}
```

Giải thích các cấu hình quan trọng

CosmosDB

- Consistency Level: "Session" — cân bằng giữa độ nhất quán và hiệu năng, phù hợp phần lớn use case.
- EnableServerless: true — Chủ yếu dùng cho phát triển thử nghiệm và không yêu cầu mức độ cao, chi phí tiết kiệm.
- Indexes: Tạo trên các trường passenger_id, driver_id, status giúp tăng tốc truy vấn.

PostgreSQL (Flexible Server)

- Loại: Thay thế Single Server (đã ngừng hỗ trợ), nhiều tính năng hơn, phù hợp workload không đều.
- SKU: B_Standard_B1ms — loại burstable, phù hợp workload nhẹ hoặc trung bình.
- Backup: Giữ dữ liệu backup trong 7 ngày, không bật geo-redundant backup (backup đa vùng).
- Firewall: Cho phép truy cập từ mọi Azure services (0.0.0.0/0), đảm bảo kết nối từ AKS và các dịch vụ nội bộ.
- CIDR host: Tính toán IP range của subnet để whitelisting an toàn, không để mở rộng quá mức.

Redis

- Loại: Standard (thay vì Basic C1), 1GB cache, không có chế độ HA (High Availability).
- SSL: Bật chỉ cho kết nối mã hóa, enable_non_ssl_port = false.
- Eviction policy: allkeys-lru — xóa key ít dùng nhất khi hết bộ nhớ, tối ưu hiệu suất lưu trữ.

2.2.6 Xuất thông tin - outputs.tf

File này định nghĩa các output values sau khi Terraform apply thành công, giúp dễ dàng lấy thông tin cần thiết cho deployment.

ACR login server

```
● ● ●  
output "acr_login_server" {  
  description = "The ACR login server (e.g., acruiitgoprod.azurecr.io)"  
  value       = azurerm_container_registry.acr.login_server  
}
```

ACR name

```
● ● ●  
output "acr_name" {  
  description = "The name of the Azure Container Registry"  
  value       = azurerm_container_registry.acr.name  
}
```

AKS kubelet identity object ID

```
● ● ●  
output "aks_kubelet_object_id" {  
  description = "Object ID of the AKS kubelet identity used to pull images from ACR"  
  value       = azurerm_kubernetes_cluster.aks.kubelet_identity[0].object_id  
}
```

PostgreSQL server FQDN

```
● ● ●  
output "postgres_fqdn" {  
  description = "PostgreSQL server FQDN"  
  value       = azurerm_postgresql_flexible_server.postgres.fqdn  
}
```

PostgreSQL connection string (sensitive)

```
● ● ●  
output "postgres_connection_string" {  
  description = "PostgreSQL connection string (dùng trong K8s secrets)"  
  value       =  
  "postgresql+asyncpg://postgresadmin:${var.db_password}@${azurerm_postgresql_flexible_server.postgres.fqdn}:5432/mydb"  
  sensitive   = true  
}
```

CosmosDB connection string (sensitive)

```
● ● ●  
output "cosmos_connection_string" {  
  description = "CosmosDB (MongoDB) connection string"  
  value       = azurerm_cosmosdb_account.cosmos.primary_mongodb_connection_string  
  sensitive   = true  
}
```

Redis cache hostname

```
● ● ●  
output "redis_hostname" {  
  description = "Redis cache hostname"  
  value       = azurerm_redis_cache.redis.hostname  
}
```

Redis primary access key (sensitive)

```
● ● ●  
output "redis_connection_string" {  
  description = "Redis connection string"  
  value       = "redis://${azurerm_redis_cache.redis.primary_access_key}@${azurerm_redis_cache.redis.hostname}:6380?  
  ssl=true"  
  sensitive   = true  
}
```

2.3. Các tài nguyên được tạo

- **Resource Group:** rg-uitgo-prod (Southeast Asia)
- **Virtual Network:** vnet-uitgo-prod (172.16.0.0/16)
- **AKS Cluster:** aks-uitgo-prod (1 node, Standard_B2s)
- **Container Registry:** acruitgoprod.azurecr.io

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

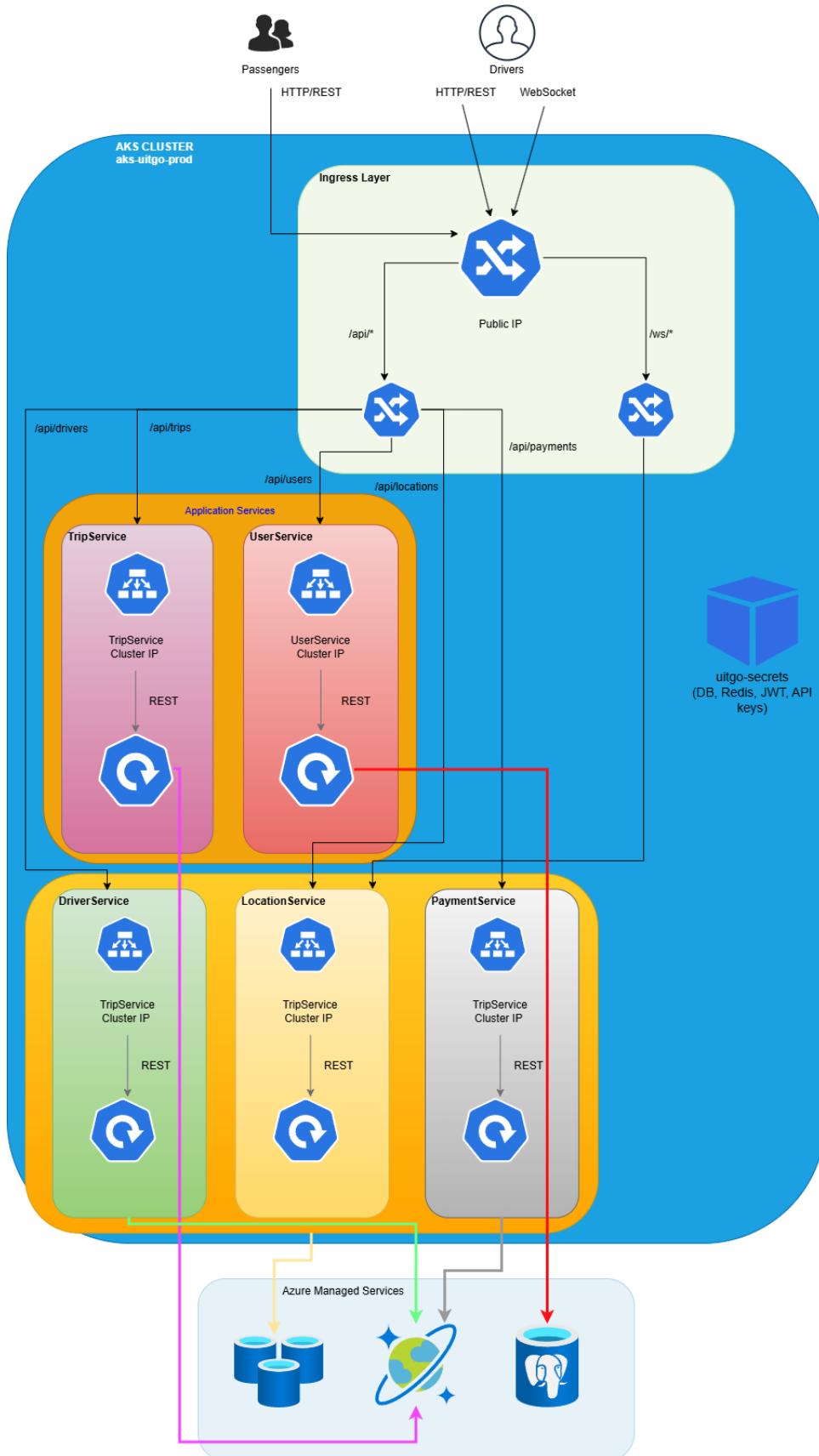
- **CosmosDB:** cosmos-uitgo-prod (MongoDB API)
- **PostgreSQL:** psql-uitgo-prod
- **Redis Cache:** redis-uitgo-prod

The screenshot shows the Azure Resource Group Overview page for 'rg-uitgo-prod'. The left sidebar includes links for Home, rg-uitgo-prod (Resource group), Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Cost Management, Monitoring, Automation, and Help. The main content area displays the 'Essentials' section with details about the subscription (Azure for Students, d8ce151-084a-418c-a446-0ff133a2d388) and location (Southeast Asia). It also shows 'Tags' (Add tags) and 'Resources' (Recommendations). A search bar at the top allows filtering by field type and location. Below is a table listing resources:

Name	Type	Location
acruiitgoprod	Container registry	Southeast Asia
aks-uitgo-prod	Kubernetes service	Southeast Asia
cosmos-uitgo-prod	Azure Cosmos DB for MongoDB...	Southeast Asia
logs-uitgo-prod	Log Analytics workspace	Southeast Asia
private.postgres.database.azure.com	Private DNS zone	Global
psql-uitgo-prod	Azure Database for PostgreSQL ...	Southeast Asia
redis-uitgo-prod	Azure Cache for Redis	Southeast Asia
vnet-uitgo-prod	Virtual network	Southeast Asia

Hình ảnh các tài nguyên được tạo

2.4. Kiến trúc tổng quan sơ đồ K8s



UIT-GO Kubernetes Architecture

3. CI/CD Pipeline với GitHub Actions

3.1. Workflow configuration

File `.github/workflows/deploy.yml` định nghĩa 4-stage pipeline:

Giai đoạn 1: Test (Unit Test)

Mục đích: Đảm bảo code mới không làm hỏng logic nội bộ trước khi build.

1. Checkout code.
2. Cài đặt Python và các thư viện trong `tests/requirements.txt`.
3. Chạy `pytest tests/ --deselect tests/smoke_test.py` (chỉ chạy unit test, bỏ qua smoke test).
4. (Nếu thất bại, toàn bộ pipeline sẽ dừng lại tại đây.)

```
test:
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.11'

    - name: Install test dependencies
      run: |
        python -m pip install --upgrade pip
        # Cài đặt các thư viện cần thiết cho Unit Test
        pip install -r tests/requirements.txt
        # Cài thêm các thư viện mà code của bạn dùng (nếu test cần)
        pip install fastapi uvicorn pydantic python-dotenv "passlib[bcrypt]"

    - name: Run unit tests
      run: |
        # Giả sử các file test của bạn nằm trong thư mục tests/
        # Bỏ qua file smoke_test.py (vì đây là integration test)
        pytest tests/ --deselect tests/smoke_test.py
  continue-on-error: false # Dừng lại nếu test fail
```

Giai đoạn 1: Test

Giai đoạn 2: Build (CI)

Mục đích: Xây dựng và lưu trữ 5 image (ảnh) của 5 microservice.

1. Checkout code.
2. Đăng nhập vào Azure (dùng `azure/login`) với secret `AZURE_CREDENTIALS`.
3. Đăng nhập vào Azure Container Registry (dùng `az acr login`).
4. Build 5 Docker image (Location, Trip, Driver, Payment, User) với tag là mã `github.sha` (mã của commit).
5. Đẩy (push) cả 5 image này lên kho ACR (`acruitgoprod`).

```
● ● ●
build:
  runs-on: ubuntu-latest
  needs: test
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Log in to Azure
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: Log in to ACR (Dùng az acr login)
      run: |
        az acr login --name ${{ env.AC_NAME }}

    - name: Build and push (LocationService)
      run: |
        docker build ./LocationService -t ${{ env.AC_NAME }}.azurecr.io/locationservice:${{ github.sha }}
        docker push ${{ env.AC_NAME }}.azurecr.io/locationservice:${{ github.sha }}

    - name: Build and push (TripService)
      run: |
        docker build ./TripService -t ${{ env.AC_NAME }}.azurecr.io/tripservice:${{ github.sha }}
        docker push ${{ env.AC_NAME }}.azurecr.io/tripservice:${{ github.sha }}

    - name: Build and push (DriverService)
      run: |
        docker build ./DriverService -t ${{ env.AC_NAME }}.azurecr.io/driverservice:${{ github.sha }}
        docker push ${{ env.AC_NAME }}.azurecr.io/driverservice:${{ github.sha }}

    - name: Build and push (PaymentService)
      run: |
        docker build ./PaymentService -t ${{ env.AC_NAME }}.azurecr.io/paymentservice:${{ github.sha }}
        docker push ${{ env.AC_NAME }}.azurecr.io/paymentservice:${{ github.sha }}

    - name: Build and push (UserService)
      run: |
        docker build ./UserService -t ${{ env.AC_NAME }}.azurecr.io/userservice:${{ github.sha }}
        docker push ${{ env.AC_NAME }}.azurecr.io/userservice:${{ github.sha }}
```

Giai đoạn 2: Build

Giai đoạn 3: Deploy (CD)

Mục đích: Triển khai 5 image mới lên cụm K8s và "nối dây" chúng với database.

1. Checkout code.
2. Đăng nhập vào Azure (dùng azure/login).
3. Kết nối kubectl với cụm AKS (az aks get-credentials).
4. **(Bước quan trọng)** Lấy các chuỗi kết nối (connection string) động từ các dịch vụ Azure PaaS (CosmosDB, Redis, Postgres) bằng az cli.
5. Tạo hoặc cập nhật K8s Secret uitgo-secrets, "bơm" vào đó:
 - a. Chuỗi kết nối (từ bước 13).
 - b. Các secret khác (DB_PASSWORD, JWT_SECRET_KEY, VNP_TMN_CODE...) từ GitHub Secrets.
6. Cập nhật 5 file k8s/*.yaml (dùng sed) để trả đến tag image mới nhất (từ github.sha).
7. Áp dụng 5 file manifests đã sửa (kubectl apply -f k8s/userservice.yaml...)

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
● ● ●

deploy:
  runs-on: ubuntu-latest
  needs: build # Phát chờ 'build' xong
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Log in to Azure
      uses: azure/login@v1
      with:
        creds: ${ secrets.AZURE_CREDENTIALS }

    - name: Set AKS context (Kết nối kubectl)
      uses: azure/aks-set-context@v3
      with:
        resource-group: ${ env.RESOURCE_GROUP_NAME }
        cluster-name: ${ env.AKS_CLUSTER_NAME }

# --- LẤY CHUỖI KẾT NỐI TỪ AZURE ---
- name: Get Azure Service Connection Strings
  id: azure-keys # Đặt ID để các bước sau gọi
  run: |
    # Lấy Connection String của CosmosDB (Mongo)
    COSMOS_CS=$(az cosmosdb keys list --name ${ env.COSMOS_DB_NAME } --resource-group ${ env.RESOURCE_GROUP_NAME } --type connection-strings --query "connectionStrings[0].connectionString" -o tsv)

    # Lấy Host và Key của Redis
    REDIS_HOST=$(az redis show --name ${ env.REDIS_CACHE_NAME } --resource-group ${ env.RESOURCE_GROUP_NAME } --query "hostName" -o tsv)
    REDIS_KEY=$(az redis list-keys --name ${ env.REDIS_CACHE_NAME } --resource-group ${ env.RESOURCE_GROUP_NAME } --query "primaryKey" -o tsv)

    # Lấy thông tin Postgres (Host và User là cố định)
    POSTGRES_HOST=${ env.POSTGRES_SERVER_NAME }.postgres.database.azure.com
    POSTGRES_USER="postgresadmin"

    # Đặt các giá trị này làm output (SỬ DỤNG CÚ PHÁP $() )
    echo "COSMOS_CONNECTION_STRING=$COSMOS_CS" >> $GITHUB_OUTPUT
    echo "REDIS_HOST=$REDIS_HOST" >> $GITHUB_OUTPUT
    echo "REDIS_KEY=$REDIS_KEY" >> $GITHUB_OUTPUT
    echo "POSTGRES_HOST=$POSTGRES_HOST" >> $GITHUB_OUTPUT
    echo "POSTGRES_USER=$POSTGRES_USER" >> $GITHUB_OUTPUT
```

Giai đoạn kết nối tới Azure và lấy chuỗi kết nối Azure.

```
# --- TẠO K8S SECRETS ---
- name: Create K8s Secrets
  env:
    # Lấy output từ bước 'azure-keys'
    DB_PASSWORD: ${ secrets.DB_PASSWORD } # Lấy từ GitHub Secret
    COSMOS_CONNECTION_STRING: ${ steps.azure-keys.outputs.COSMOS_CONNECTION_STRING }
    REDIS_HOST: ${ steps.azure-keys.outputs.REDIS_HOST }
    REDIS_KEY: ${ steps.azure-keys.outputs.REDIS_KEY }
    POSTGRES_HOST: ${ steps.azure-keys.outputs.POSTGRES_HOST }
    POSTGRES_USER: ${ steps.azure-keys.outputs.POSTGRES_USER }

    # Lấy các secret khác từ GitHub Secrets
    JWT_SECRET_KEY: ${ secrets.JWT_SECRET_KEY }
    MAPBOX_ACCESS_TOKEN: ${ secrets.MAPBOX_ACCESS_TOKEN }
    TRIPSPVC_CLIENT_ID: ${ secrets.TRIPSPVC_CLIENT_ID }
    TRIPSPVC_CLIENT_SECRET: ${ secrets.TRIPSPVC_CLIENT_SECRET }
    USER_SERVICE_BASE_URL: ${ secrets.USER_SERVICE_BASE_URL }
    VNP_TMN_CODE: ${ secrets.VNP_TMN_CODE }
    VNP_HASH_SECRET: ${ secrets.VNP_HASH_SECRET }
    VNP_URL: ${ secrets.VNP_URL }

  run: |
    kubectl create secret generic uitgo-secrets --dry-run=client -o yaml \
      --from-literal=DB_PASSWORD=$DB_PASSWORD \
      --from-literal=COSMOS_CONNECTION_STRING=$COSMOS_CONNECTION_STRING \
      --from-literal=REDIS_HOST=$REDIS_HOST \
      --from-literal=REDIS_KEY=$REDIS_KEY \
      --from-literal=POSTGRES_HOST=$POSTGRES_HOST \
      --from-literal=POSTGRES_USER=$POSTGRES_USER \
      --from-literal=JWT_SECRET_KEY=$JWT_SECRET_KEY \
      --from-literal=MAPBOX_ACCESS_TOKEN=$MAPBOX_ACCESS_TOKEN \
      --from-literal=TRIPSPVC_CLIENT_ID=$TRIPSPVC_CLIENT_ID \
      --from-literal=TRIPSPVC_CLIENT_SECRET=$TRIPSPVC_CLIENT_SECRET \
      --from-literal=USER_SERVICE_BASE_URL=$USER_SERVICE_BASE_URL \
      --from-literal=VNP_TMN_CODE=$VNP_TMN_CODE \
      --from-literal=VNP_HASH_SECRET=$VNP_HASH_SECRET \
      --from-literal=VNP_URL=$VNP_URL \
    | kubectl apply -f -

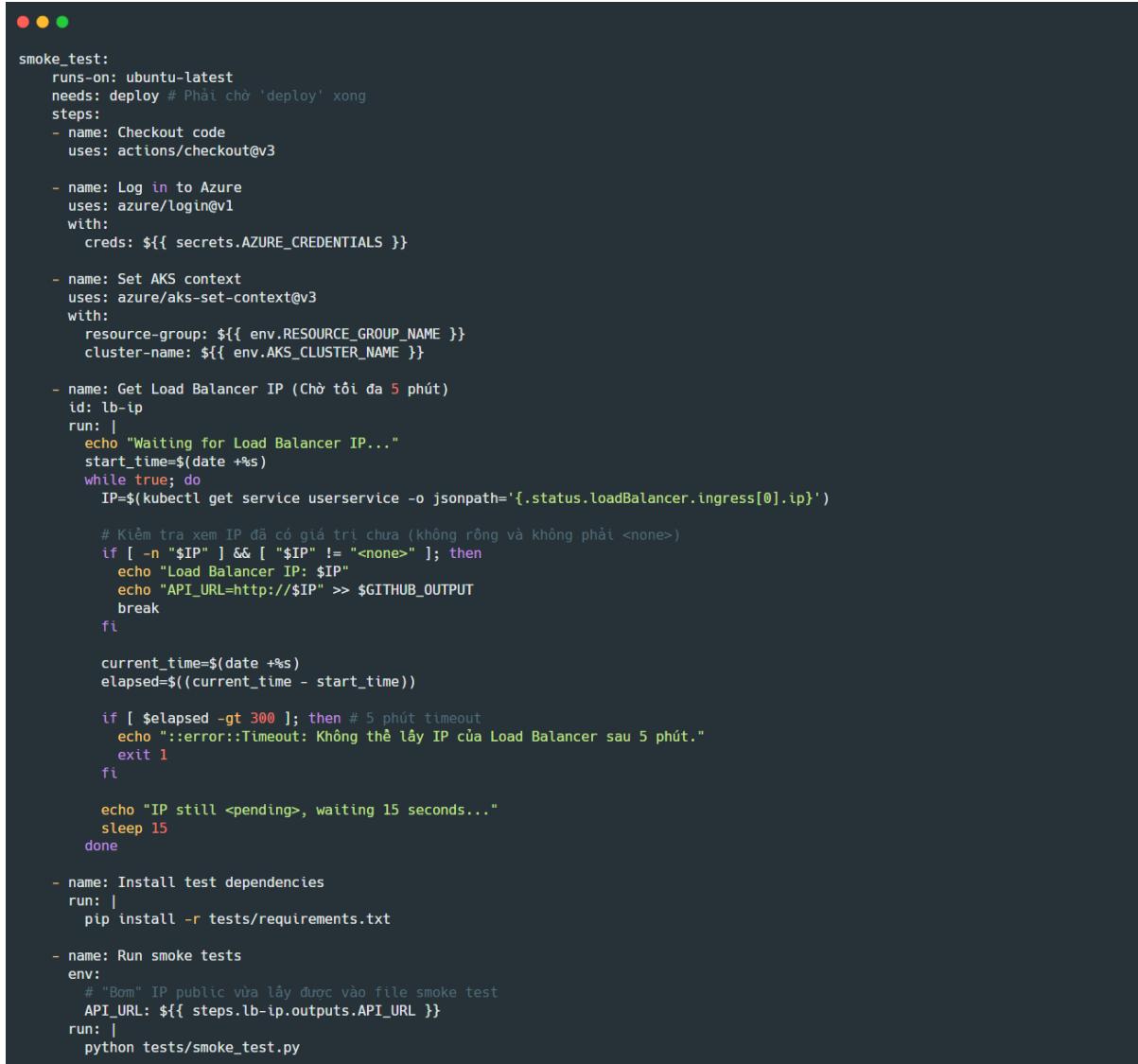
# --- CẬP NHẬT TAG IMAGE MỚI ---
- name: Update K8s manifests with new image tag
  run: |
    sed -i 's|image: ${ env.ACR_NAME }.azurecr.io/locationservice:.*|image: ${ env.ACR_NAME }.azurecr.io/locationservice:${{ github.sha }}|g' k8s/locationservice.yaml
    sed -i 's|image: ${ env.ACR_NAME }.azurecr.io/tripservice:.*|image: ${ env.ACR_NAME }.azurecr.io/tripservice:${{ github.sha }}|g' k8s/tripservice.yaml
    sed -i 's|image: ${ env.ACR_NAME }.azurecr.io/driverservice:.*|image: ${ env.ACR_NAME }.azurecr.io/driverservice:${{ github.sha }}|g' k8s/driverservice.yaml
    sed -i 's|image: ${ env.ACR_NAME }.azurecr.io/paymentservice:.*|image: ${ env.ACR_NAME }.azurecr.io/paymentservice:${{ github.sha }}|g' k8s/paymentservice.yaml
    sed -i 's|image: ${ env.ACR_NAME }.azurecr.io/userservice:.*|image: ${ env.ACR_NAME }.azurecr.io/userservice:${{ github.sha }}|g' k8s/userservice.yaml
```

Giai đoạn tạo K8s secrets và cập nhật Image Tag

Giai đoạn 4: Smoke Test

Mục đích: Xác nhận hệ thống (đã deploy) đang chạy và IP public hoạt động.

1. Checkout code.
2. Lấy IP public (External IP) của userservice (chạy vòng lặp while để chờ IP).
3. "Bom" IP này vào biến môi trường API_URL.
4. Chạy script python tests/smoke_test.py (sử dụng API_URL để test đăng ký và đăng nhập).
5. (Nếu thất bại, pipeline sẽ báo lỗi đó, cho phép bạn rollback.)



```
smoke_test:
  runs-on: ubuntu-latest
  needs: deploy # Phải chờ 'deploy' xong
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Log in to Azure
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: Set AKS context
      uses: azure/aks-set-context@v3
      with:
        resource-group: ${{ env.RESOURCE_GROUP_NAME }}
        cluster-name: ${{ env.AKS_CLUSTER_NAME }}

    - name: Get Load Balancer IP (Chờ tối đa 5 phút)
      id: lb-ip
      run: |
        echo "Waiting for Load Balancer IP..."
        start_time=$(date +%s)
        while true; do
          IP=$(kubectl get service userservice -o jsonpath='{.status.loadBalancer.ingress[0].ip}')

          # Kiểm tra xem IP đã có giá trị chưa (không rỗng và không phải <none>)
          if [ -n "$IP" ] && [ "$IP" != "<none>" ]; then
            echo "Load Balancer IP: $IP"
            echo "API_URL=http://$IP" >> $GITHUB_OUTPUT
            break
          fi

          current_time=$(date +%s)
          elapsed=$((current_time - start_time))

          if [ $elapsed -gt 300 ]; then # 5 phút timeout
            echo "::error::Timeout: Không thể lấy IP của Load Balancer sau 5 phút."
            exit 1
          fi

          echo "IP still <pending>, waiting 15 seconds..."
          sleep 15
        done
      done

    - name: Install test dependencies
      run: |
        pip install -r tests/requirements.txt

    - name: Run smoke tests
      env:
        # "Bom" IP public vừa lấy được vào file smoke test
        API_URL: ${{ steps.lb-ip.outputs.API_URL }}
      run: |
        python tests/smoke_test.py
```

Giai đoạn Kiểm thử sau deploy

3.2. Kubernetes Manifests

Thư mục k8s/ chứa 8 file YAML, trong đó có 5 file cho kiến trúc microservice, mỗi file định nghĩa "bản thiết kế" cho một microservice. Mỗi file bao gồm 2 đối tượng chính:

- **kind: Deployment:**

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

- Khai báo container image nào sẽ được chạy (ví dụ: acruitgoprod.azurecr.io/userservice).
- Thêm vào các biến môi trường và secrets (như chuỗi kết nối database) từ uitgo-secrets vào container.
- **kind: Service:**
 - Cung cấp một tên DNS nội bộ cố định cho Deployment.
 - **5 service (Trip, Driver, v.v.)** dùng type: ClusterIP. Đây là chế độ *Nội bộ*, giúp các service gọi nhau an toàn (ví dụ: <http://tripservice:8000>).

Ngoài ra, có 3 file YAML định nghĩa:

- NGINX Ingress Controller - API Gateway cho toàn bộ hệ thống
- Định nghĩa routing rules cho REST API endpoints (/api/*)
- Định nghĩa routing rules riêng cho WebSocket connections (/ws/*)

3.3 Chi tiết các resources

3.3.1 File: nginx-ingress-controller.yaml

Mục đích: Triển khai NGINX Ingress Controller - API Gateway cho toàn bộ hệ thống

Resource 1: Namespace

```
apiVersion: v1
kind: Namespace
metadata:
  name: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
```

Giải thích: Tạo namespace riêng tên "ingress-nginx" để cài đặt Ingress Controller khỏi application services.

Resource 2: ServiceAccount

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
```

Giải thích: ServiceAccount để NGINX pod có identity riêng, dùng cho RBAC authorization.

Resource 3: ClusterRole

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ingress-nginx
rules:
  - apiGroups: [""]
    resources: ["configmaps", "endpoints", "nodes", "pods", "secrets", "namespaces"]
    verbs: ["list", "watch", "get"]
  - apiGroups: ["coordination.k8s.io"]
    resources: ["leases"]
    verbs: ["list", "watch", "get", "create", "update"]
  - apiGroups: [""]
    resources: ["services"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["networking.k8s.io"]
    resources: ["ingresses", "ingresses/status", "ingressclasses"]
    verbs: ["get", "list", "watch", "update"]
```

Giải thích: Định nghĩa quyền cho Ingress Controller:

- list, watch, get pods, services, endpoints, configmaps, secrets: Để discover services
- get, create, update leases: Cho leader election (HA)
- get, list, watch, update ingresses: Để đọc Ingress rules và update status
- get, list, watch ingressclasses: Để biết ingress class nào thuộc về controller này

Resource 4: ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx
subjects:
  - kind: ServiceAccount
    name: ingress-nginx
    namespace: ingress-nginx
```

Giải thích: Bind ClusterRole (quyền) với ServiceAccount (identity). Cho phép ingress-nginx ServiceAccount có các quyền đã define ở ClusterRole.

Resource 5: Deployment

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/component: controller
  template:
    metadata:
      labels:
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/component: controller
    spec:
      serviceAccountName: ingress-nginx
      containers:
        - name: controller
          image: registry.k8s.io/ingress-nginx/controller:v1.9.4
          args:
            - /nginx-ingress-controller
            - --election-id=ingress-controller-leader
            - --controller-class=k8s.io/ingress-nginx
            - --ingress-class=nginx
            - --configmap=$(POD_NAMESPACE)/ingress-nginx-controller
            - --publish-service=$(POD_NAMESPACE)/ingress-nginx-controller
          env:
            - name: POD_NAME
              valueFrom: {fieldRef: {fieldPath: metadata.name}}
            - name: POD_NAMESPACE
              valueFrom: {fieldRef: {fieldPath: metadata.namespace}}
      ports:
        - name: http
          containerPort: 80
        - name: https
          containerPort: 443
      livenessProbe:
        httpGet: {path: /healthz, port: 10254}
        initialDelaySeconds: 10
        periodSeconds: 10
      readinessProbe:
        httpGet: {path: /healthz, port: 10254}
        initialDelaySeconds: 10
        periodSeconds: 10
```

Giải thích từng phần:

Tham số	Giải thích
replicas: 1	Chạy 1 pod NGINX (có thể scale lên 3-5 cho HA)
image: ...controller:v1.9.4	NGINX Ingress Controller version 1.9.4 từ Kubernetes registry
--election-id=...	Leader election ID cho HA (khi có nhiều replicas)

--controller-class=k8s.io/ingress-nginx	Controller class để biết Ingress nào thuộc controller này
--ingress-class=nginx	Ingress class name (dùng trong Ingress annotation)
--configmap=.../ingress-nginx-controller	ConfigMap chứa NGINX config
--publish-service=.../ingress-nginx-controller	Publish LoadBalancer IP vào Ingress status
env: POD_NAME, POD_NAMESPACE	Inject pod metadata vào container
ports: 80, 443	HTTP và HTTPS ports
livenessProbe	Kubernetes restart pod nếu /healthz fail
readinessProbe	Kubernetes không route traffic nếu /healthz fail

Resource 6: Service (LoadBalancer)

```
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 80
      targetPort: 80
      protocol: TCP
    - name: https
      port: 443
      targetPort: 443
      protocol: TCP
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/component: controller
```

Giải thích: Service type LoadBalancer tạo Azure Load Balancer với Public IP 135.171.210.33. Port 80 (HTTP) và 443 (HTTPS) từ internet forward vào container port 80/443. Selector chọn pods có label app.kubernetes.io/name=ingress-nginx.

Resource 7: ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ingress-nginx-controller
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
namespace: ingress-nginx
data:
  allow-snippet-annotations: "true"
```

Giải thích: ConfigMap chứa configuration cho NGINX. allow-snippet-annotations: "true" cho phép dùng nginx.ingress.kubernetes.io/configuration-snippet annotation trong Ingress để custom NGINX config.

Resource 8: IngressClass

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: nginx
spec:
  controller: k8s.io/ingress-nginx
```

Giải thích: IngressClass định nghĩa class "nginx" được handle bởi controller k8s.io/ingress-nginx. Các Ingress resources sẽ dùng annotation kubernetes.io/ingress.class: nginx để chỉ định dùng NGINX Ingress Controller này.

3.3.2 File: ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: uitgo-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "300"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "300"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "300"
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/cors-allow-origin: "*"
    nginx.ingress.kubernetes.io/proxy-body-size: "10m"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  rules:
  - http:
      paths:
      - path: /api/users(/|$)(.*)
        pathType: ImplementationSpecific
        backend:
          service:
            name: userservice
            port: {number: 80}
      - path: /api/drivers(/|$)(.*)
        pathType: ImplementationSpecific
        backend:
          service:
            name: driverservice
            port: {number: 8000}
      - path: /api/trips(/|$)(.*)
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
pathType: ImplementationSpecific
backend:
  service:
    name: tripservice
    port: {number: 8000}
- path: /api/locations(/|$)(.*)
  pathType: ImplementationSpecific
  backend:
    service:
      name: locationservice
      port: {number: 8000}
- path: /api/payments(/|$)(.*)
  pathType: ImplementationSpecific
  backend:
    service:
      name: paymentservice
      port: {number: 8000}
```

Annotations:

Annotation	Giải thích
kubernetes.io/ingress.class: nginx	Chỉ định dùng NGINX Ingress Controller (match với IngressClass)
proxy-connect-timeout: "300"	Timeout kết nối đến backend: 300 giây (5 phút)
proxy-send-timeout: "300"	Timeout gửi request đến backend: 300 giây
proxy-read-timeout: "300"	Timeout đọc response từ backend: 300 giây
enable-cors: "true"	Bật CORS (Cross-Origin Resource Sharing) cho mobile app
cors-allow-origin: "*"	Cho phép tất cả origins (có thể giới hạn thành domain cụ thể)
proxy-body-size: "10m"	Giới hạn request body size max 10MB (tránh upload file quá lớn)
rewrite-target: /\$2	Rewrite URL: /api/users/auth/login → /auth/login (remove prefix /api/users)
use-regex: "true"	Enable regex trong path pattern

Routing Paths:

Mỗi path rule có cấu trúc:

path: /api/users(/|\$)(.*) → Regex: /api/users (bắt buộc), sau đó / hoặc kết thúc, capture group \$2 là phần còn lại

backend: service name + port → Route đến Kubernetes Service

Path Pattern	Backend	Ví dụ
/api/users(/ \$)(.*)	userservice:80	/api/users/auth/login → userservice /auth/login
/api/drivers(/ \$)(.*)	driverservice:8000	/api/drivers/me → driverservice /me
/api/trips(/ \$)(.*)	tripservice:8000	/api/trips/123 → trippservice /123
/api/locations(/ \$)(.*)	locationservice:8000	/api/locations/drivers/nearby → locationservice /drivers/nearby
/api/payments(/ \$)(.*)	paymentservice:8000	/api/payments/process → paymentservice /process

3.3.3 File: ingress-websocket.yaml

Mục đích: Định nghĩa routing rules riêng cho WebSocket connections (/ws/*)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: uitgo-websocket-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/websocket-services: "locationservice"
    nginx.ingress.kubernetes.io/proxy-http-version: "1.1"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "300"
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/cors-allow-origin: "*"
spec:
  rules:
  - http:
      paths:
      - path: /ws
        pathType: Prefix
        backend:
          service:
            name: locationservice
            port: {number: 8000}
```

Tại sao tách riêng file cho WebSocket?

- WebSocket cần timeout lâu hơn (3600s = 1 giờ) so với REST API (300s = 5 phút)
- WebSocket cần HTTP/1.1 (không dùng HTTP/2)
- WebSocket KHÔNG rewrite URL (giữ nguyên /ws/... không xóa prefix)
- Dễ quản lý và debug hơn khi tách riêng

Annotations đặc biệt:

Annotation	Giải thích
websocket-services: "locationservice"	Chỉ định service này handle WebSocket (enable WebSocket upgrade)
proxy-http-version: "1.1"	Force HTTP/1.1 (WebSocket yêu cầu HTTP/1.1, không dùng HTTP/2)
proxy-read-timeout: "3600"	Timeout đọc 1 giờ (WebSocket connection lâu dài)
proxy-send-timeout: "3600"	Timeout gửi 1 giờ

Path:

path: /ws

pathType: Prefix → Match tất cả paths bắt đầu bằng /ws

KHÔNG có rewrite-target → Giữ nguyên URL /ws/driver/123/location

WebSocket Endpoints:

Endpoint	Mục đích
/ws/driver/{driver_id}/location	Driver (ranh) cập nhật vị trí real-time
/ws/trip/{trip_id}/driver	Driver tracking chuyến đi, gửi vị trí cho passenger
/ws/trip/{trip_id}/passenger	Passenger tracking chuyến đi, nhận vị trí driver

3.3.4 File: userservice.yaml

Mục đích: Quản lý users (passengers & drivers), authentication, JWT tokens

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: userservice
spec:
  replicas: 1
  selector:
    matchLabels: {app: userservice}
  template:
    metadata:
      labels: {app: userservice}
    spec:
      containers:
        - name: userservice
          image: acruiitgoprod.azurecr.io/userservice:latest
          ports:
            - containerPort: 8000
          envFrom:
            - secretRef: {name: uitgo-secrets}
          env:
            - name: POSTGRES_HOST
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: POSTGRES_HOST}}
            - name: POSTGRES_USER
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: POSTGRES_USER}}
            - name: POSTGRES_PASSWORD
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: DB_PASSWORD}}
            - name: POSTGRES_DB
              value: "mydb"
            - name: ACCESS_TOKEN_EXPIRE_MINUTES
              value: "30"
            - name: DRIVER_SERVICE_URL
              value: "http://driverservice:8000"
            - name: SECRET_KEY
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: JWT_SECRET_KEY}}
---
apiVersion: v1
kind: Service
metadata:
  name: userservice
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 8000
  selector: {app: userservice}
```

Deployment:

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

Field	Value	Giải thích
metadata.name	userservice	Tên deployment
spec.replicas	1	Số lượng pods
spec.selector.matchLabels	app: userservice	Selector chọn pods
template.metadata.labels	app: userservice	Labels cho pods
spec.containers[0].image	acruiitgoprod.azurecr.io/userservice:latest	Docker image từ ACR
spec.containers[0].ports	containerPort: 8000	Container listen port

Environment Variables:

Biến	Source	Mục đích
POSTGRES_HOST	secretKeyRef: uitgo-secrets.POSTGRES_HOST	Hostname PostgreSQL server
POSTGRES_USER	secretKeyRef: uitgo-secrets.POSTGRES_USER	Username PostgreSQL
POSTGRES_PASSWORD	secretKeyRef: uitgo-secrets.DB_PASSWORD	Password PostgreSQL
POSTGRES_DB	value: "mydb"	Database name
ACCESS_TOKEN_EXPIRE_MINUTES	value: "30"	JWT token expire time
SECRET_KEY	secretKeyRef: uitgo-secrets.JWT_SECRET_KEY	JWT signing key
DRIVER_SERVICE_URL	value: "http://driveservice:8000"	URL DriverService

Service (ClusterIP):

Field	Value	Giải thích
metadata.name	userservice	Tên service (dùng làm DNS: userservice:80)
spec.type	ClusterIP	Internal service, chỉ access được trong cluster
spec.ports[0].port	80	Port expose cho services khác gọi
spec.ports[0].targetPort	8000	Port container đang listen
spec.selector	app: userservice	Forward traffic đến pods có label này

Database Connection:

Database: PostgreSQL (Azure Database for PostgreSQL: psql-uitgo-prod)

3.3.5 File: driverservice.yaml

Mục đích: Quản lý driver profiles, online/offline status, wallet

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: driverservice
spec:
  replicas: 1
  selector:
    matchLabels: {app: driverservice}
  template:
    metadata:
      labels: {app: driverservice}
    spec:
      containers:
        - name: driverservice
          image: acruiitgoprod.azurecr.io/driverservice:latest
          ports:
            - containerPort: 8000
          envFrom:
            - secretRef: {name: uitgo-secrets}
          env:
            - name: MONGODB_URL
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: COSMOS_CONNECTION_STRING}}
            - name: SECRET_KEY
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: JWT_SECRET_KEY}}
            - name: LOCATION_SERVICE_URL
              value: "http://locationservice:8000"
            - name: USER_SERVICE_URL
              value: "http://userservice:8000"
---
apiVersion: v1
kind: Service
metadata:
  name: driverservice
spec:
  type: ClusterIP
  ports:
    - port: 8000
      targetPort: 8000
  selector: {app: driverservice}

```

Deployment:

Field	Value	Giải thích
metadata.name	driverservice	Tên deployment
spec.replicas	1	Số lượng pods
spec.selector.matchLabels	app: driverservice	Selector chọn pods
template.metadata.labels	app: driverservice	Labels cho pods

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

spec.containers[0].image	acruiitgoprod.azurecr.io/driverservice:latest	Docker image từ ACR
spec.containers[0].ports	containerPort: 8000	Container listen port

Environment Variables:

Biến	Source	Mục đích
MONGODB_URL	secretKeyRef: uitgo-secrets.COSMOS_CONNECTION_STRING	Cosmos DB connection string
SECRET_KEY	secretKeyRef: uitgo-secrets.JWT_SECRET_KEY	JWT verification key
LOCATION_SERVICE_URL	value: "http://locationservice:8000"	URL LocationService
USER_SERVICE_URL	value: "http://userservice:8000"	URL UserService

Service (ClusterIP):

Field	Value	Giải thích
metadata.name	driverservice	Tên service (dùng làm DNS: driverservice:8000)
spec.type	ClusterIP	Internal service, chỉ access được trong cluster
spec.ports[0].port	8000	Port expose cho services khác gọi
spec.ports[0].targetPort	8000	Port container đang listen
spec.selector	app: driverservice	Forward traffic đến pods có label này

Database Connection:

Database: Cosmos DB - MongoDB API (cosmos-uitgo-prod)

3.3.6 File: tripservice.yaml

Mục đích: Quản lý trip lifecycle, fare estimation, driver-passenger matching. Orchestrator chính gọi 4 services khác

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tripservice
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```

spec:
  replicas: 1
  selector:
    matchLabels: {app: tripservice}
  template:
    metadata:
      labels: {app: tripservice}
    spec:
      containers:
        - name: tripservice
          image: acruiitgoprod.azurecr.io/tripservice:latest
          ports:
            - containerPort: 8000
          envFrom:
            - secretRef: {name: uitgo-secrets}
          env:
            - name: MONGODB_URL
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: COSMOS_CONNECTION_STRING}}
            - name: LOCATION_SERVICE_URL
              value: "http://locationservice:8000"
            - name: DRIVER_SERVICE_URL
              value: "http://driverservice:8000"
            - name: PAYMENT_SERVICE_URL
              value: "http://paymentservice:8000"
            - name: MY_CLIENT_ID
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: TRIPSVC_CLIENT_ID}}
            - name: MY_CLIENT_SECRET
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: TRIPSVC_CLIENT_SECRET}}
            - name: USER_SERVICE_URL
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: USER_SERVICE_BASE_URL}}
---
apiVersion: v1
kind: Service
metadata:
  name: tripservice
spec:
  ports:
    - port: 8000
      targetPort: 8000
  selector: {app: tripservice}

```

Deployment:

Field	Value	Giải thích
metadata.name	tripservice	Tên deployment
spec.replicas	1	Số lượng pods
spec.selector.matchLabels	app: tripservice	Selector chọn pods
template.metadata.labels	app: tripservice	Labels cho pods
spec.containers[0].image	acruiitgoprod.azurecr.io/tripservice:latest	Docker image từ ACR
spec.containers[0].ports	containerPort: 8000	Container listen port

Environment Variables:

Biến	Source	Mục đích
MONGODB_URL	secretKeyRef: uitgo-secrets.COSMOS_CONNECTION_STRING	Cosmos DB connection string
LOCATION_SERVICE_URL	value: "http://locationservice:8000"	Gọi LocationService
DRIVER_SERVICE_URL	value: "http://driverservice:8000"	Gọi DriverService
PAYMENT_SERVICE_URL	value: "http://paymentservice:8000"	Gọi PaymentService
USER_SERVICE_URL	secretKeyRef: uitgo-secrets.USER_SERVICE_BASE_URL	Gọi UserService
MY_CLIENT_ID	secretKeyRef: uitgo-secrets.TRIPSVC_CLIENT_ID	Service token client ID
MY_CLIENT_SECRET	secretKeyRef: uitgo-secrets.TRIPSVC_CLIENT_SECRET	Service token secret
MAPBOX_ACCESS_TOKEN	secretKeyRef: uitgo-secrets.MAPBOX_ACCESS_TOKEN	Mapbox API key (routing)

Service (ClusterIP):

Field	Value	Giải thích
metadata.name	tripservice	Tên service (dùng làm DNS: tribservice:8000)
spec.type	ClusterIP	Internal service, chỉ access được trong cluster
spec.ports[0].port	8000	Port expose cho services khác gọi
spec.ports[0].targetPort	8000	Port container đang listen
spec.selector	app: tribservice	Forward traffic đến pods có label này

Database Connection:

Database: Cosmos DB - MongoDB API (cosmos-uitgo-prod)

3.3.7 File: locationservice.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: locationservice
spec:
  replicas: 1
  selector:
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
matchLabels: {app: locationservice}
template:
  metadata:
    labels: {app: locationservice}
  spec:
    containers:
      - name: locationservice
        image: acruitgoprod.azurecr.io/locationservice:latest
        ports:
          - containerPort: 8000
        envFrom:
          - secretRef: {name: uitgo-secrets}
        env:
          - name: REDIS_HOST
            valueFrom: {secretKeyRef: {name: uitgo-secrets, key: REDIS_HOST}}
          - name: REDIS_KEY
            valueFrom: {secretKeyRef: {name: uitgo-secrets, key: REDIS_KEY}}
---
apiVersion: v1
kind: Service
metadata:
  name: locationservice
spec:
  ports:
    - port: 8000
      targetPort: 8000
  selector: {app: locationservice}
```

Deployment:

Field	Value	Giải thích
metadata.name	locationservice	Tên deployment
spec.replicas	1	Số lượng pods
spec.selector.matchLabels	app: locationservice	Selector chọn pods
template.metadata.labels	app: locationservice	Labels cho pods
spec.containers[0].image	acruitgoprod.azurecr.io/locationservice:latest	Docker image từ ACR
spec.containers[0].ports	containerPort: 8000	Container listen port

Environment Variables:

Biến	Source	Mục đích
REDIS_HOST	secretKeyRef: uitgo-secrets.REDIS_HOST	Redis hostname
REDIS_KEY	secretKeyRef: uitgo-secrets.REDIS_KEY	Redis access key

Service (ClusterIP):

Field	Value	Giải thích
-------	-------	------------

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

metadata.name	locationservice	Tên service (dùng làm DNS: locationservice:8000)
spec.type	ClusterIP	Internal service, chỉ access được trong cluster
spec.ports[0].port	8000	Port expose cho services khác gọi
spec.ports[0].targetPort	8000	Port container đang listen
spec.selector	app: locationservice	Forward traffic đến pods có label này

Database Connection:

Database: Redis Cache (redis-uitgo-prod)

3.3.8 File paymentservice.yaml

Mục đích: Xử lý thanh toán E-Wallet, VNPay integration, transaction history

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: paymentservice
spec:
  replicas: 1
  selector:
    matchLabels: {app: paymentservice}
  template:
    metadata:
      labels: {app: paymentservice}
    spec:
      containers:
        - name: paymentservice
          image: acruitgoprod.azurecr.io/paymentservice:latest
          ports:
            - containerPort: 8000
          envFrom:
            - secretRef: {name: uitgo-secrets}
          env:
            - name: MONGODB_URL
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: COSMOS_CONNECTION_STRING}}
            - name: VNP_TMN_CODE
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: VNP_TMN_CODE}}
            - name: VNP_HASH_SECRET
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: VNP_HASH_SECRET}}
            - name: VNP_URL
              valueFrom: {secretKeyRef: {name: uitgo-secrets, key: VNP_URL}}
```

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

```
---
apiVersion: v1
kind: Service
metadata:
  name: paymentservice
spec:
  ports:
  - port: 8000
    targetPort: 8000
    selector: {app: paymentservice}
```

Deployment:

Field	Value	Giải thích
metadata.name	paymentservice	Tên deployment
spec.replicas	1	Số lượng pods
spec.selector.matchLabels	app: paymentservice	Selector chọn pods
template.metadata.labels	app: paymentservice	Labels cho pods
spec.containers[0].image	acruijtgoprod.azurecr.io/paymentservice:latest	Docker image từ ACR
spec.containers[0].ports	containerPort: 8000	Container listen port

Environment Variables:

Biến	Source	Mục đích
MONGODB_URL	secretKeyRef: uitgo-secrets.COSMOS_CONNECTION_STRING	Cosmos DB connection string
VNP_TMN_CODE	secretKeyRef: uitgo-secrets.VNP_TMN_CODE	VNPay merchant code
VNP_HASH_SECRET	secretKeyRef: uitgo-secrets.VNP_HASH_SECRET	VNPay hash secret
VNP_URL	secretKeyRef: uitgo-secrets.VNP_URL	VNPay API URL
TRIP_SERVICE_URL	value: "http://tripservice:8000"	URL TripService

Service (ClusterIP):

Field	Value	Giải thích
metadata.name	paymentservice	Tên service (dùng làm DNS: paymentservice:8000)
spec.type	ClusterIP	Internal service, chỉ access được trong cluster
spec.ports[0].port	8000	Port expose cho services khác gọi

Báo cáo Đồ án môn Điện toán đám mây và phát triển ứng dụng hướng dịch vụ

spec.ports[0].targetPort	8000	Port container đang listen
spec.selector	app: paymentservice	Forward traffic đến pods có label này

Database Connection:

Database: Cosmos DB - MongoDB API (cosmos-uitgo-prod)

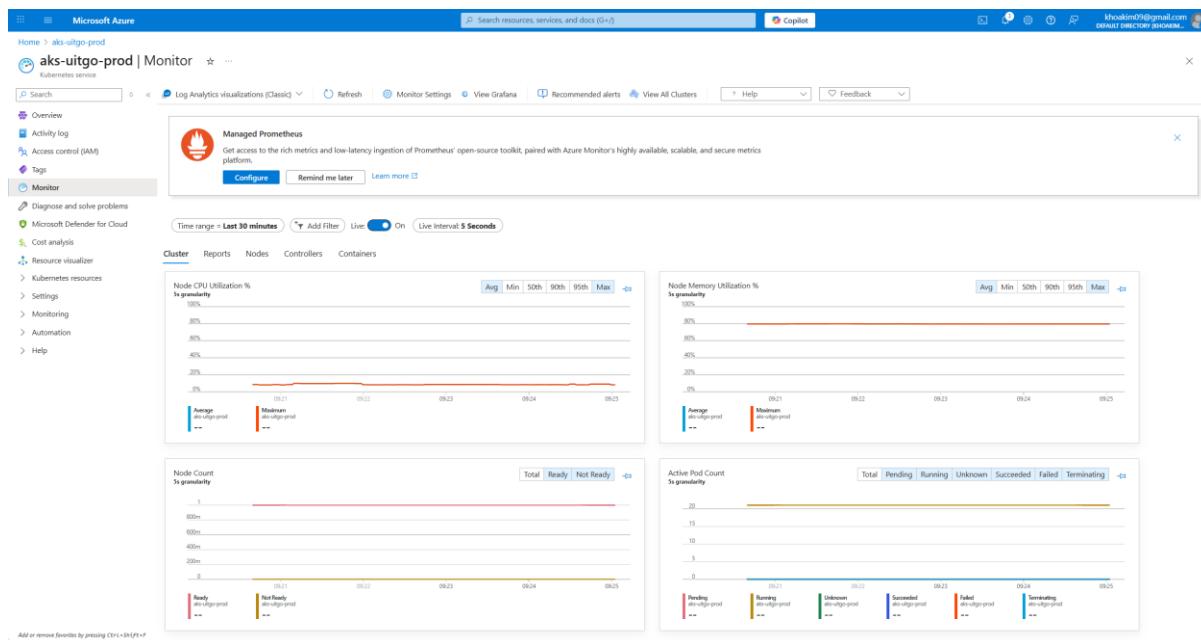
4. Local Development với Docker Compose

File docker-compose.yml cho phép chạy toàn bộ hệ thống trên local:

- MongoDB container (port 27017)
- Redis container (port 6379)
- PostgreSQL container (port 5432)
- 5 microservices (ports 8000-8004)

5. Monitoring trên Azure

- **AKS Logs:** kubectl logs -f deployment/{service-name}
- **Pod status:** kubectl get pods
- **Azure Portal:** Monitor metrics (CPU, Memory, Network) của AKS cluster



Hình ảnh trang monitor

Chương IV. HƯỚNG ĐI MODULE CHUYÊN SÂU

Sau khi bàn bạc và cân nhắc bọn em đã lập ra 1 bảng so sánh về các giá trị mà các module mang lại.

Module	Vai trò Kỹ sư	Mục tiêu chính	Câu hỏi cốt lõi cần giải quyết
A: Scalability	System Architect	Thiết kế để đạt "hyper-scale" (chịu tải cực lớn)	"Làm sao để hệ thống chạy NHANH khi có 1 triệu người dùng?"
B: Reliability	Site Reliability Engineer (SRE)	Thiết kế hệ thống "chống chịu và tự phục hồi" sự cố[cite: 78].	"Làm sao để hệ thống KHÔNG SẬP khi có lỗi xảy ra?"
C: Security	Security Engineer	Thiết kế theo triết lý "Zero Trust" và "phòng thủ theo chiều sâu"	"Làm sao để hệ thống AN TOÀN trước các cuộc tấn công?"
D: Observability	Operations Engineer	Thiết kế hệ thống có khả năng "tự chẩn đoán" sức khỏe.	"Làm sao để BIẾT hệ thống đang vận hành tốt hay xấu?"
E: FinOps	FinOps Engineer	Thiết kế quy trình tự động hóa và "tối ưu hóa chi phí".	"Làm sao để vận hành RẺ và HIỆU QUẢ ?"

Phân tích sự khác biệt:

- Các module A (Nhanh), B (Bền), D (Để chẩn đoán), E (Rẻ) đều tập trung vào việc làm cho hệ thống **vận hành tốt hơn (better, faster, stronger, cheaper)**.
- Chỉ riêng Module C (An toàn) tập trung vào việc đảm bảo hệ thống có thể **ổn định, an toàn**.

Việc ưu tiên **Module C (Security)** là hoàn toàn hợp lý, đặc biệt là trong bối cảnh "thế giới đầy rẫy rủi ro và hacker". **Bảo mật là Điều kiện Tiên quyết, không phải là Tính năng**

1. Tổng quan mô hình

Chuyển đổi kiến trúc hiện tại của UIT-Go sang một mô hình DevSecOps toàn diện, tuân thủ triết lý Zero Trust. Mục tiêu là tích hợp bảo mật vào mọi giai đoạn của vòng đời phát triển (shift-left) và xây dựng một hệ thống có khả năng phòng thủ theo chiều sâu (defense-in-depth).

Ba trụ cột chính của kế hoạch bao gồm:

1. **Thiết kế Mạng Zero Trust:** Phân đoạn mạng triệt để, loại bỏ các điểm truy cập public không cần thiết và triển khai Tường lửa Ứng dụng Web (WAF).
2. **Tích hợp Bảo mật vào CI/CD (Shift-Left):** Tự động hóa việc quét và phát hiện lỗ hổng ngay trong pipeline GitHub Actions.
3. **Triển khai Phòng thủ theo chiều sâu:** Áp dụng 7 lớp bảo mật từ tầng mạng, định danh, đến ứng dụng và dữ liệu.

2. Phân tích Ban đầu: Mô hình hóa Mối đe dọa (Threat Modeling)

Kế hoạch bắt đầu bằng bước nền tảng là **Threat Modeling**, sử dụng các phương pháp tiêu chuẩn:

- **Sơ đồ luồng dữ liệu (DFD):** Vẽ chi tiết các luồng dữ liệu quan trọng (xác thực, thanh toán, theo dõi vị trí) để xác định các bề mặt tấn công (attack surface).
- **Phân tích STRIDE:** Đánh giá hệ thống qua 6 hạng mục (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) để vạch ra các mối đe dọa cụ thể và biện pháp giảm thiểu (ví dụ: dùng aud claim trong JWT, mã hóa secret, rate limiting).

3. Trụ cột 1: Kiến trúc Mạng Zero Trust

3.1 Phân đoạn Mạng (Network Segmentation)

Kế hoạch đề xuất chia nhỏ VNet 172.16.0.0/16 thành các subnet riêng biệt cho từng thành phần:

- AKS Subnet (đã có)
- PostgreSQL Subnet (đã có)

- **Subnet mới:** CosmosDB Private Endpoint, Redis, Management (Bastion), Application Gateway.
- **Network Security Groups (NSGs):** Sẽ được áp dụng nghiêm ngặt cho từng subnet. Ví dụ: Subnet CSDL sẽ chỉ cho phép traffic đến từ Subnet AKS và chặn toàn bộ traffic ra ngoài (outbound).

3.2 Ân Cơ sở dữ liệu (Private Endpoints)

Triển khai **Private Link** cho PostgreSQL, CosmosDB và Redis. Điều này đảm bảo rằng các CSDL không còn lộ diện trên Internet, mọi giao tiếp chỉ diễn ra bên trong VNet qua IP private.

3.3 Tường lửa Ứng dụng Web (WAF)

Đây là một quyết định quan trọng. Kế hoạch lựa chọn **ModSecurity (Open-Source)** thay vì Azure Application Gateway WAF.

- **Lý do:** Miễn phí, linh hoạt, tích hợp trực tiếp với NGINX Ingress Controller và sử dụng OWASP Core Rule Set (CRS) mới nhất.
- **Kiến trúc:** ModSecurity sẽ được tích hợp thẳng vào NGINX Ingress, hoạt động như một lớp bảo vệ đầu tiên, quét *toàn bộ* traffic (HTTP và WebSocket) trước khi chúng chạm đến các service (UserService, TripService, v.v.).
- **Quy tắc (Rules):** Sẽ áp dụng bộ quy tắc OWASP CRS 4.0 (chống SQLi, XSS, RCE...) và các quy tắc tùy chỉnh (custom rules) cho UIT-Go, ví dụ:
 - Rate limiting (giới hạn 100 req/phút/IP).
 - Chặn các user-agent độc hại (sqlmap, nikto).
 - Giới hạn đăng nhập (5 lần/phút).
 - Xác thực định dạng (validation) cho các API thanh toán.

4. Trụ cột 2: Tích hợp Bảo mật CI/CD (Shift-Left)

Mục tiêu là phát hiện lỗ hổng sớm nhất có thể bằng cách tích hợp 6 công kiểm soát bảo mật tự động vào GitHub Actions:

1. **SAST (Static Application Security Testing):** Dùng **Bandit** để quét mã nguồn Python tìm các lỗ hổng phổ biến.
2. **Dependency Scanning (Quét thư viện):** Dùng **Safety** và **Trivy** để kiểm tra requirements.txt xem có thư viện nào dính lỗ hổng đã biết (CVE) hay không.
3. **Container Image Scanning (Quét ảnh Docker):** Dùng **Trivy** để quét ảnh Docker sau khi build, tìm các lỗ hổng trong HDH hoặc thư viện hệ thống.

4. **Secrets Scanning (Quét bí mật):** Dùng **TruffleHog** để quét toàn bộ lịch sử commit, đảm bảo không có API key, password nào bị lộ.
5. **IaC Scanning (Quét Infrastructure as Code):** Dùng **Checkov** để quét các file Terraform, tìm các cấu hình sai (ví dụ: mở cổng không an toàn).
6. **DAST (Dynamic... Testing):** Dùng **OWASP ZAP** để chạy quét tự động trên môi trường staging sau khi deploy, mô phỏng các đợt tấn công cơ bản.

Pipeline sẽ được cấu hình để **thất bại (fail build)** nếu phát hiện lỗ hổng ở mức "Critical" hoặc "High".

5. Trụ cột 3: Phòng thủ theo chiều sâu (Defense-in-Depth)

Kế hoạch cũng vạch ra 7 lớp phòng thủ bổ trợ, đảm bảo rằng nếu một lớp bị vượt qua, các lớp khác sẽ ngăn chặn cuộc tấn công:

- **Lớp 1 (Perimeter):** Azure DDoS Protection.
- **Lớp 2 (Network):** NSGs, Private Endpoints, Azure Firewall.
- **Lớp 3 (Identity):** Tích hợp Azure AD, RBAC cho Kubernetes, Managed Identities.
- **Lớp 4 (Application):** Dùng Pydantic để validation, mã hóa đầu ra (chống XSS), rate limiting.
- **Lớp 5 (Data):** Mã hóa CSDL (at-rest) và mã hóa đường truyền (in-transit) bằng TLS 1.3.
- **Lớp 6 (Logging):** Dùng Azure Monitor và Azure Sentinel (SIEM) để phát hiện mối đe dọa.
- **Lớp 7 (Incident Response):** Xây dựng Playbook (kịch bản) và Runbook (hướng dẫn) để tự động hóa và xử lý sự cố.