Tài liệu thiết kế hệ thống

Hệ thống theo dõi xe và phát hiện xe lạ thông qua biển số xe ở nhà trọ

Các thành viên trong nhóm:

- Ngô Đình Linh - 23020394.
- Hoàng Mạnh Hùng - 23020371.
- Đoàn Quang Huy - 23020374.
- Nguyễn Quý Bắc - 23020334.

▼ Thiết kế kiến trúc

Hệ thống được thiết kế theo kiến trúc **Microservices**, trong đó các chức năng chính được chia thành các dịch vụ độc lập, mỗi dịch vụ chịu trách nhiệm cho một nhiệm vụ cụ thể. Các dịch vụ này được triển khai trong các container Docker riêng biệt và được quản lý thông qua Docker Compose. Kiến trúc này đảm bảo tính linh hoạt, dễ mở rộng và khả năng bảo trì cao, phù hợp với một hệ thống có thể phát triển trong tương lai.

Các thành phần chính:

Hệ thống bao gồm 3 dịch vụ chính và một giao diện người dùng:

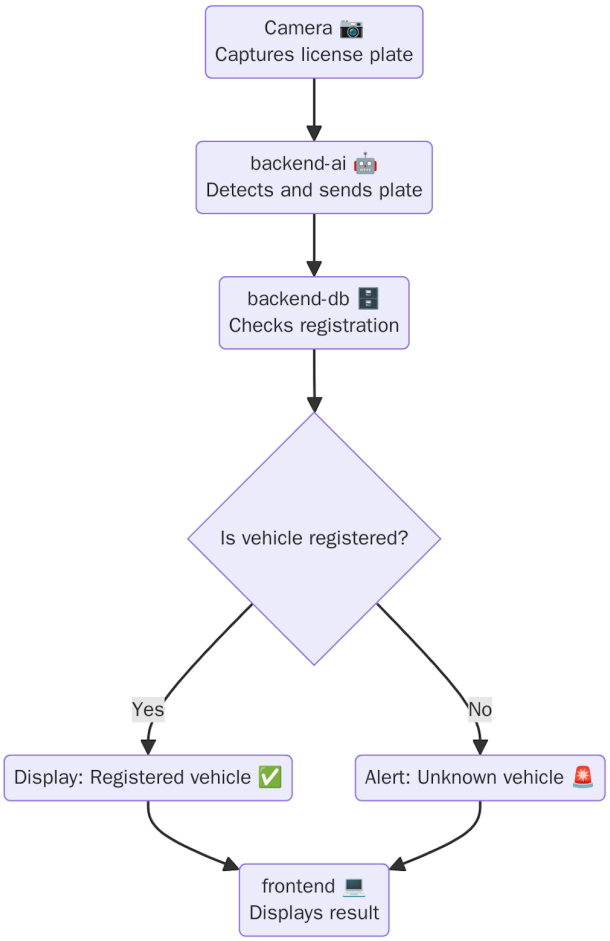
- `backend-ai` (Dịch vụ nhận diện biển số):
  - **Chức năng:** Sử dụng công nghệ AI/ML để nhận diện biển số xe từ hình ảnh hoặc video do camera cung cấp.
  - **Công nghệ:** Python với OpenCV, PyTorch và TensorFlow để xử lý hình ảnh.
  - **Đầu vào:** Hình ảnh/video từ camera.
  - **Đầu ra:** Biển số xe (dạng văn bản) và gửi đến `backend` để kiểm tra.
- `backend` - backend-db (Dịch vụ quản lý dữ liệu):
  - **Chức năng:** Quản lý dữ liệu của hệ thống, bao gồm thông tin xe, người dùng, lịch sử ra vào và thông báo.
  - **Công nghệ:** API RESTful (Java Spring Boot, Node.js) kết hợp với cơ sở dữ liệu quan hệ PostgreSQL.

    - **Đầu vào:** Yêu cầu từ `backend-ai` (kiểm tra biển số).
    - **Đầu ra:** Kết quả kiểm tra (xe lạ/xe đã đăng ký), dữ liệu lịch sử, hoặc thông báo.
- `frontend-admin` (Giao diện người dùng):
    - **Chức năng:** Cung cấp giao diện cho người dùng (chủ nhà trọ) để tương tác với hệ thống.
    - **Công nghệ sử dụng:** React.js cho giao diện web.
    - **Tương tác:** Giao tiếp với `backend-ai` qua API để hiển thị thông tin.
- **Camera** (Phần cứng bên ngoài):
    - Cung cấp hình ảnh/video chứa biển số xe, gửi đến `backend-ai` qua giao thức (như RTSP) hoặc API.

Luồng hoạt động giữa các dịch vụ:

- **Camera gửi dữ liệu:**
    - Camera ghi nhận hình ảnh biển số xe và gửi đến `backend-ai` qua API.
- `backend-ai` **xử lý:**
    - Nhận diện biển số từ hình ảnh/video và trả về kết quả (biển số dạng văn bản).
    - Gửi biển số đến `backend` qua API REST để kiểm tra trạng thái xe.
- `backend` **kiểm tra và lưu trữ:**
    - So sánh biển số với danh sách xe đã đăng ký.
    - Nếu xe không có trong danh sách, gán nhãn “xe lạ” và lưu vào nhật ký ra vào.
    - Gửi thông báo đến chủ nhà trọ (qua email/SMS nếu được cấu hình).
- `frontend` **hiển thị:**
    - Giao diện quản trị (`admin/`) gọi API của `backend-ai` để hiển thị thông tin.
    - Hiển thị thông báo "Phát hiện xe lạ: [Biển số]" hoặc cập nhật danh sách xe ra vào.

Sơ đồ kiến trúc:



- **Camera**: Gửi hình ảnh/video đến `backend-ai`.
- **`backend-ai`**: Gửi biển số đã nhận diện đến `backend` qua API REST.
- **`backend`**: Quản lý dữ liệu và trả kết quả (xe lạ/xe đã đăng ký) về cho `backend-ai`.
- **`frontend-admin`**: Giao tiếp với `backend-ai` qua API để lấy thông tin và hiển thị cho người dùng qua giao diện web.

Triển khai:

- Các dịch vụ (`backend-ai`, `backend`, `frontend-admin`) được triển khai trong các container Docker riêng biệt, được quản lý bởi Docker Compose (qua file `docker-compose.yml`).
- **Cấu hình `docker-compose.yml`:**

```
services:
  # Backend AI
  backend-ai:
```

```
  build:
    context: ./Backend_AI
  container_name: license-plate-backend-ai
  ports:
    - "8000:8000"
  environment:
    - BACKEND_DB_URL=http://backend:5000/api
    - SAVE_IMAGES=true
    - ENABLE_CAMERA_DETECTION=true
    - CAMERA_CONFIG=/app/config/cameras.json
    - DEBUG_SAVE_FRAMES=true
  volumes:
    - ai-images:/app/images
    - ./Backend_AI/config:/app/config
  restart: always
  networks:
    - license-plate-network

# Backend DB - Dịch vụ quản lý dữ liệu
backend:
  build:
    context: ./Backend_DB
  container_name: license-plate-backend-db
  restart: always
  environment:
    - DB_HOST=db
    - DB_USER=postgres
    - DB_PASS=linh2005
    - DB_NAME=BTL_THPTHT
    - REDIS_HOST=redis
    - REDIS_PORT=6379
    - JWT_SECRET=Ngodinhlinh1@
  ports:
    - "5001:5000"
  depends_on:
    db:
      condition: service_healthy
    redis:
```

```
      condition: service_started
  networks:
    - license-plate-network

# PostgreSQL Database
db:
  image: docker.io/postgres:17
  container_name: license-plate-postgres
  restart: always
  environment:
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=linh2005
    - POSTGRES_DB=BTL_THPTHT
  volumes:
    - postgres_data:/var/lib/postgresql/data
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql
  ports:
    - "5432:5432"
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U postgres"]
    interval: 5s
    timeout: 5s
    retries: 5
  networks:
    - license-plate-network

# Redis for high-performance data exchange
redis:
  image: docker.io/redis:7
  container_name: license-plate-redis
  restart: always
  volumes:
    - redis_data:/data
  ports:
    - "6379:6379"
  networks:
    - license-plate-network
```

```
  # Frontend Admin
  frontend-admin:
    build:
      context: ./frontend/admin
    container_name: license-plate-frontend-admin
    ports:
      - "80:80"
    environment:
      - REACT_APP_API_BASE=http://backend-ai:8000
    depends_on:
      - backend
    networks:
      - license-plate-network

networks:
  license-plate-network:
    driver: bridge
    # external: true

volumes:
  postgres_data:
  redis_data:
  ai-images:
```

- `backend-ai`: Chạy trên cổng 8000, xử lý nhận diện biển số xe từ camera, lưu hình ảnh vào volume `ai-images`, và giao tiếp với `backend` qua API (`http://backend:5000/api`).
- `backend` (backend-db): Chạy trên cổng 5001 (ánh xạ từ 5000 trong container), kết nối với cơ sở dữ liệu PostgreSQL (`db`) để quản lý dữ liệu, tích hợp Redis (`redis`) để tăng hiệu suất trao đổi dữ liệu, và cung cấp API cho các dịch vụ khác.
- `frontend-admin`: Chạy trên cổng 80, giao tiếp với backend-ai qua API (http://backend-ai:8000) để hiển thị thông tin (có thể cần điều chỉnh để giao tiếp với backend nếu cần lấy dữ liệu như lịch sử ra vào).
- `db`: Sử dụng PostgreSQL (`postgres:17`) để lưu trữ dữ liệu chính (thông tin xe, lịch sử ra vào), chạy trên cổng 5432, với volume `postgres_data`

để đảm bảo dữ liệu không bị mất.

- `redis`: Sử dụng Redis (`redis:7`) để lưu trữ dữ liệu tạm thời (cache), chạy trên cổng 6379, với volume `redis_data` để lưu dữ liệu.

Cách các luồng hoạt động trong hệ thống:

1. *Luồng xử lý AI nhận diện biển số và liên kết với quản lý xe*

   A. Xử lý dữ liệu từ camera:

   `backend-ai` sử dụng CameraManager để lấy frame từ các camera (IP/webcam/video file). Mỗi frame sẽ được gửi tới API/api/recognize (nội bộ FastAPI).
   API này sử dụng 2 model YOLOv5:
   LP Detector: Phát hiện vùng chứa biển số trên ảnh.
   LP OCR: Nhận diện ký tự trên biển số (OCR).

   B. Gửi kết quả vào database:
   Sau khi nhận diện được biển số, API sẽ:
   Gọi API Backend DB để kiểm tra biển số đã đăng ký chưa (`/vehicles/check/{license_plate}`).
   Gửi lịch sử nhận diện vào DB qua API
   `/vehicles-in` (lưu lịch sử xe vào).
   Nếu là xe lạ (chưa đăng ký), gửi cảnh báo qua API
   `/alerts` và broadcast WebSocket cho frontend.
   Dữ liệu được lưu vào các bảng:
   `vehicles`, `history`, `alerts` trong PostgreSQL (xem file `init.sql`).

   C. Quản lý xe dựa vào biển số:
   Khi có biển số mới, hệ thống sẽ:
   - Kiểm tra biển số có trong bảng vehicles không.
   - Nếu có, lấy thông tin chủ xe, loại xe, tòa nhà, v.v.
   Nếu không, tạo cảnh báo xe lạ.
   Frontend sẽ hiển thị realtime các cảnh báo, lịch sử xe ra vào, quản lý danh sách xe, v.v.

2. *Cách test với dữ liệu thật (biển số xe):*

A. Test với ảnh/video thật:

Cách 1: Thêm đường dẫn file video/ảnh thật vào `cameras.json` (dạng "url": `./test_videos/your_video.mp4`).

Cách 2: Dùng IP Camera thật, cấu hình đúng RTSP/HTTP URL trong `cameras.json`.

Cách 3: Dùng webcam, cấu hình "url": 0 hoặc "url": 1 (tùy máy).

B. Test qua API
Gửi ảnh biển số thật qua API /api/recognize bằng Postman hoặc script Python:
Apply to
`cameras.json`

C. Test frontend
Truy cập trang quản lý camera, chọn camera đã cấu hình, kiểm tra realtime stream, snapshot, cảnh báo, lịch sử xe ra vào.

D. Kiểm tra dữ liệu trong database
Dùng DBeaver, pgAdmin, hoặc lệnh SQL để kiểm tra bảng `vehicles`, `history`, `alerts` xem dữ liệu có được ghi nhận đúng không.

3. *Tóm tắt quy trình test dữ liệu thật:*
   Chuẩn bị ảnh/video biển số thật hoặc camera thật.
   Cấu hình vào
   `cameras.json`.
   Khởi động hệ thống.

```
docker-compose up -d
```

▼ Thiết kế chi tiết

▼ Các chức năng

▼ Cập nhật danh sách xe đã đăng ký của người thuê trọ ở nhà trọ

Mô tả ngắn gọn: Quản trị viên (chủ thể) thực hiện cập nhật danh sách xe mà người thuê trọ đã đăng ký ở nhà trọ, tùy theo nhu cầu của người thuê trọ.

Actor:

- Chính: Chủ thể (Quản trị viên).
- Phụ: Giao diện chính, giao diện quản lý xe đã đăng ký, giao diện thêm xe mới, giao diện cập nhật thông tin xe, giao diện xác nhận xóa xe đăng ký.

Mục tiêu: Quản trị viên có thể quản lý và cập nhật tình hình về các xe của người thuê trọ, đồng thời hỗ trợ trong việc phát hiện xe lạ.

Pre-condition:

- Quản trị viên đã có tài khoản và đã đăng nhập vào hệ thống.
- Thiết bị đăng nhập hệ thống của quản trị viên có kết nối Internet.
- Hệ thống không trong thời gian bảo trì.

Main Flow:

1. Tại giao diện chính, quản trị viên chọn “Quản lý xe”.
2. Hệ thống hiển thị giao diện quản lý xe đã đăng ký, bao gồm danh sách xe mà người thuê trọ đã đăng ký.
3. Nếu quản trị viên muốn thêm xe đăng ký, quản trị viên chọn “Thêm xe mới”.
4. Hệ thống hiển thị giao diện thêm xe mới.
5. Quản trị viên điền các thông tin về xe muốn thêm và chọn “Thêm mới”.
6. Hệ thống đối chiếu với danh sách xe đã đăng ký trước đó.
7. Hệ thống xác nhận và cập nhật danh sách xe đăng ký.

Alternative Flow:

- *Bước 3*: *Nếu quản trị viên muốn cập nhật thông tin một xe đã đăng ký*:
    - Quản trị viên chọn “Sửa” ở dòng xe mà quản trị viên muốn cập nhật thông tin.
    - Hệ thống hiển thị giao diện cập nhật thông tin xe.

    - Quản trị viên sửa lại những thông tin về xe cần cập nhật và chọn “Cập nhật”.
    - Chuyển sang bước 6.
- *Bước 3*: *Nếu quản trị viên muốn xóa một xe đã đăng ký*:
    - Quản trị viên chọn “Xóa” ở dòng xe mà quản trị viên muốn xóa.
    - Hệ thống hiển thị giao diện xác nhận xóa xe đăng ký.
    - Quản trị viên chọn “Xác nhận”.
    - Chuyển sang bước 7.

Post-condition: Quản trị viên có thể quản lý các xe đăng ký ở nhà trọ tốt hơn, từ đó giúp ích cho việc kiểm soát xe ra vào nhà trọ.

Exceptions:

- *Nếu hệ thống trong thời gian bảo trì*:
    - Hệ thống hiển thị “Hệ thống đang bảo trì. Vui lòng thử lại sau.”.
- *Nếu thiết bị của quản trị viên không được kết nối Internet:*
    - Hệ thống hiển thị “Không có kết nối Internet. Vui lòng kiểm tra lại.”.


Registered_Vehicles_List_Updating
Admin
Frontend (Vehicles.js)
Backend (API)
Database (PostgreSQL)
Main Flow: Thêm xe mới
Truy cập giao diện Vehicles
render()
fetchVehicles() [GET /]
alt [Không có kết nối Internet]
showError("Không có kết nối Internet. Vui lòng kiểm tra lại.")
[Hệ thống đang bảo trì]
returnErrorResponse(503, "Hệ thống đang bảo trì. Vui lòng thử lại sau.")
setError("Hệ thống đang bảo trì. Vui lòng thử lại sau.")
showError("Hệ thống đang bảo trì. Vui lòng thử lại sau.")
[Thành công]
authenticateToken()
query("SELECT v.*, b.name FROM vehicles v JOIN buildings b ON v.building_id = b.id")
returnVehicles()
returnVehiclesResponse()
fetchBuildings() [GET /buildings]
alt [Không có kết nối Internet]
showError("Không có kết nối Internet. Vui lòng kiểm tra lại.")
[Hệ thống đang bảo trì]
returnErrorResponse(503, "Hệ thống đang bảo trì. Vui lòng thử lại sau.")
setError("Hệ thống đang bảo trì. Vui lòng thử lại sau.")
showError("Hệ thống đang bảo trì. Vui lòng thử lại sau.")
[Thành công]
query("SELECT * FROM buildings")
returnBuildings()
returnBuildingsResponse()
handleOpenDialog() [Button: Thêm xe mới]
setOpenDialog(true)
handleInputChange() + handleSubmit() [Button: Thêm mới]
addVehicle(vehicleData) [POST /]
alt [Không có kết nối Internet]
showError("Không có kết nối Internet. Vui lòng kiểm tra lại.")
[Hệ thống đang bảo trì]
returnErrorResponse(503, "Hệ thống đang bảo trì. Vui lòng thử lại sau.")
setSnackbar({message: "Hệ thống đang bảo trì. Vui lòng thử lại sau.", severity: "error"})
Hiển thị thông báo lỗi
[Thành công]
authenticateToken()
query("SELECT * FROM vehicles WHERE license_plate = $1")
returnExistingVehicle()
alt [Biển số xe đã tồn tại]
returnErrorResponse(400, "Biển số xe đã được đăng ký")
setSnackbar({message: "Lỗi: Biển số xe đã được đăng ký", severity: "error"})
Hiển thị thông báo lỗi
[Biển số xe chưa tồn tại]
query("INSERT INTO vehicles (...) VALUES (...) RETURNING *")
returnNewVehicle()
returnNewVehicleResponse()
fetchVehicles()
setSnackbar({message: "Thêm xe mới thành công", severity: "success"})
Hiển thị thông báo thành công
Alternative Flow 1: Cập nhật thông tin xe
opt [Quản trị viên muốn cập nhật thông tin xe]
handleOpenDialog(params.row) [IconButton: Edit]
setOpenDialog(true)
handleInputChange() + handleSubmit() [Button: Cập nhật]
updateVehicle(id, vehicleData) [PUT /:id]
alt [Không có kết nối Internet]
showError("Không có kết nối Internet. Vui lòng kiểm tra lại.")
[Hệ thống đang bảo trì]
returnErrorResponse(503, "Hệ thống đang bảo trì. Vui lòng thử lại sau.")
setSnackbar({message: "Hệ thống đang bảo trì. Vui lòng thử lại sau.", severity: "error"})
Hiển thị thông báo lỗi
[Thành công]
authenticateToken()
query("SELECT * FROM vehicles WHERE id = $1")
returnVehicle()
alt [Xe không tồn tại]
returnErrorResponse(404, "Không tìm thấy xe")
setSnackbar({message: "Lỗi: Không tìm thấy xe", severity: "error"})
Hiển thị thông báo lỗi
[Xe tồn tại]
query("UPDATE vehicles SET ... WHERE id = $1 RETURNING *")
returnUpdatedVehicle()
returnUpdatedVehicleResponse()
fetchVehicles()
setSnackbar({message: "Cập nhật thông tin xe thành công", severity: "success"})
Hiển thị thông báo thành công
Alternative Flow 2: Xóa xe
opt [Quản trị viên muốn xóa xe]
handleDeleteVehicle(params.row.id) [IconButton: Delete]
window.confirm()
alt [Admin xác nhận xóa]
deleteVehicle(id) [DELETE /:id]
alt [Không có kết nối Internet]
showError("Không có kết nối Internet. Vui lòng kiểm tra lại.")
[Hệ thống đang bảo trì]
returnErrorResponse(503, "Hệ thống đang bảo trì. Vui lòng thử lại sau.")
setSnackbar({message: "Hệ thống đang bảo trì. Vui lòng thử lại sau.", severity: "error"})
Hiển thị thông báo lỗi
[Thành công]
authenticateToken()
query("SELECT * FROM vehicles WHERE id = $1")
returnVehicle()
alt [Xe không tồn tại]
returnErrorResponse(404, "Không tìm thấy xe")
setSnackbar({message: "Lỗi: Không tìm thấy xe", severity: "error"})
Hiển thị thông báo lỗi
[Xe tồn tại]
query("DELETE FROM vehicles WHERE id = $1")
confirmDelete()
confirmDeleteResponse()
fetchVehicles()
setSnackbar({message: "Xóa xe thành công", severity: "success"})
Hiển thị thông báo thành công
[Admin hủy xóa]
Đóng dialog xác nhận
Admin
Frontend (Vehicles.js)
Backend (API)
Database (PostgreSQL)

▼ **Theo dõi lịch sử các xe ra vào nhà trọ**

Mô tả ngắn gọn: Quản trị viên (chủ thể) thực hiện theo dõi lịch sử các xe ra vào nhà trọ qua hệ thống.

Actor:

- Chính: Chủ thể (quản trị viên).
- Phụ: Giao diện chính, giao diện lịch sử xe ra vào nhà trọ.

Mục tiêu: Quản trị viên có thể theo dõi những xe ra vào nhà trọ, từ đó có thể xác định số đầu xe trong nhà trọ hay những thời điểm khả nghi (xe chưa đăng ký ra vào nhà trọ).

Pre-condition:

- Quản trị viên đã có tài khoản và đã đăng nhập vào hệ thống.
- Thiết bị đăng nhập hệ thống của quản trị viên có kết nối Internet.
- Hệ thống không trong thời gian bảo trì.

Main Flow:

1. Tại giao diện chính, quản trị viên chọn “Lịch sử ra vào”.
2. Hệ thống hiển thị giao diện lịch sử xe ra vào nhà trọ với danh sách các lượt ra vào của các xe từ khi lập ra hệ thống đến thời điểm xem.
3. Nếu quản trị viên muốn rút ngắn danh sách để dễ dàng tra cứu, quản trị viên điền vào các thông tin cần điền ở bộ lọc (filter) và chọn “Tìm kiếm”.
4. Hệ thống rút ngắn lại danh sách theo yêu cầu của quản trị viên.

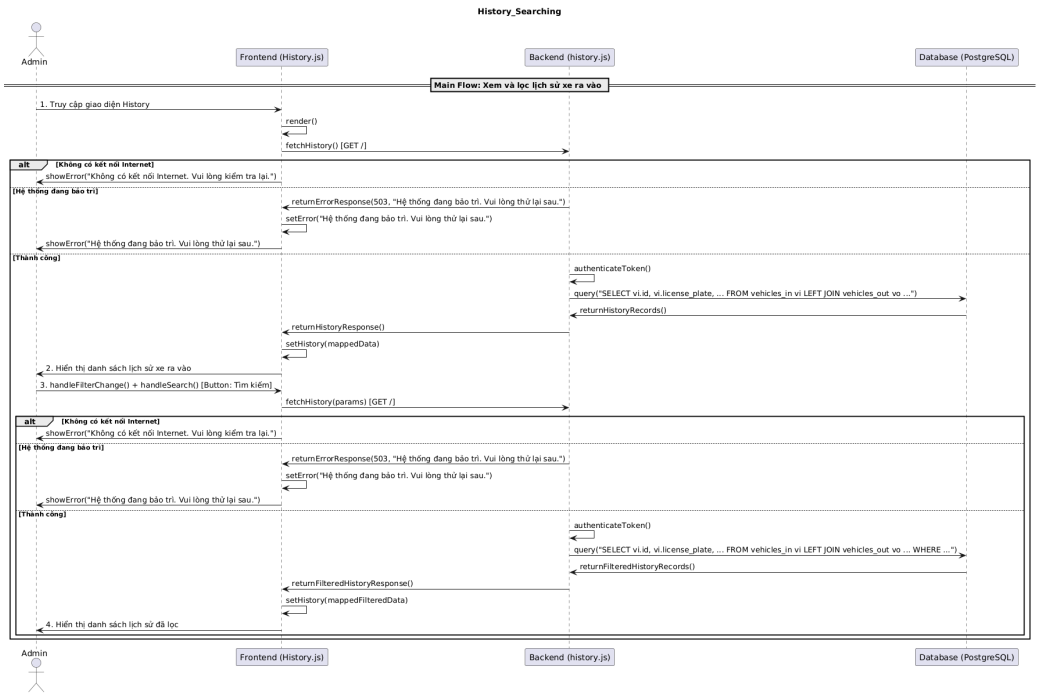
Alternative Flow:

Post-condition: Quản trị viên có thể dễ dàng theo dõi tình hình về các lượt ra vào nhà trọ của các xe và kịp thời phát hiện những trường hợp khả nghi (nếu có).

Exceptions:

- *Nếu hệ thống trong thời gian bảo trì*:
    - Hệ thống hiển thị “Hệ thống đang bảo trì. Vui lòng thử lại sau.”.

- *Nếu thiết bị của quản trị viên không được kết nối Internet:*
    - Hệ thống hiển thị “Không có kết nối Internet. Vui lòng kiểm tra lại.”.



▼ Phê duyệt các xe ra vào nhà trọ

Mô tả ngắn gọn: Quản trị viên (chủ thể) thực hiện phê duyệt các xe ra vào nhà trọ qua hệ thống.

Actor:

- Chính: Chủ thể (quản trị viên).
- Phụ: Giao diện chính, giao diện camera realtime, giao diện cảnh báo xe lạ.

Mục tiêu: Quản trị viên có thể trực tiếp phát hiện xe lạ đi vào nhà trọ thông qua nhận diện và cảnh báo từ hệ thống.

Pre-condition:

- Quản trị viên đã có tài khoản và đã đăng nhập vào hệ thống.
- Thiết bị đăng nhập hệ thống của quản trị viên có kết nối Internet.
- Hệ thống không trong thời gian bảo trì.
- Camera hoạt động bình thường.

Main Flow:

1. Tại giao diện chính, quản trị viên chọn “Camera”.
2. Hệ thống hiển thị giao diện camera realtime với stream realtime (camera quay trực tiếp) và Frame mới nhất (cứ 3 giây thì chụp và thu ảnh từ camera).
3. Hệ thống chuyển ảnh thu được từ Frame về mô hình AI để nhận diện biển số.
4. Nếu AI nhận diện được biển số của một xe đi vào hoặc ra khỏi nhà trọ, hệ thống đối chiếu biển số xe với danh sách các xe đã đăng ký.
5. Hệ thống xác nhận và lưu biển số, trạng thái, hướng di chuyển cùng thời điểm chụp ảnh Frame vào dữ liệu lịch sử ra vào nhà trọ.

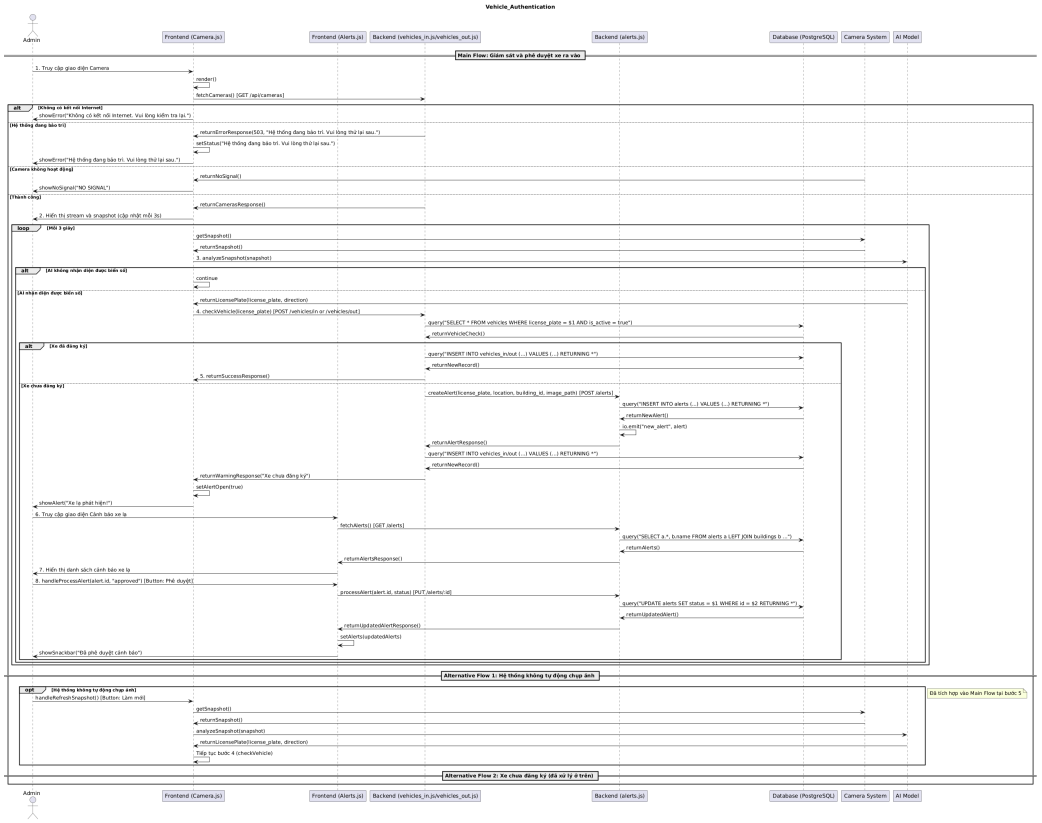
Alternative Flow:

- *Bước 3*: *Nếu hệ thống không thể tự động chụp ảnh và chuyển về mô hình AI*:
    - Quản trị viên chọn “Làm mới” ở mục “Frame mới nhất” để chụp ảnh.
    - Hệ thống chuyển ảnh chụp được về mô hình AI để nhận diện biển số.
    - Chuyển sang bước 4.
- *Bước 5*: *Nếu có một xe đi vào nhà trọ và có biển số được xác định là không trùng với biển số xe nào được đăng ký*:
    - Hệ thống lưu vào dữ liệu cảnh báo xe lạ và đưa ra thông báo với quản trị viên.
    - Quản trị viên chọn “Cảnh báo xe lạ”.
    - Hệ thống hiển thị giao diện cảnh báo xe lạ với ảnh biển số cùng thông tin về thời điểm chụp ảnh biển số của xe đó.
    - Quản trị viên chọn “Từ chối” hoặc “Phê duyệt”.
    - Nếu quản trị viên chọn “Phê duyệt”, hệ thống lưu biển số, trạng thái, hướng di chuyển cùng thời điểm chụp ảnh Frame vào dữ liệu lịch sử ra vào nhà trọ.

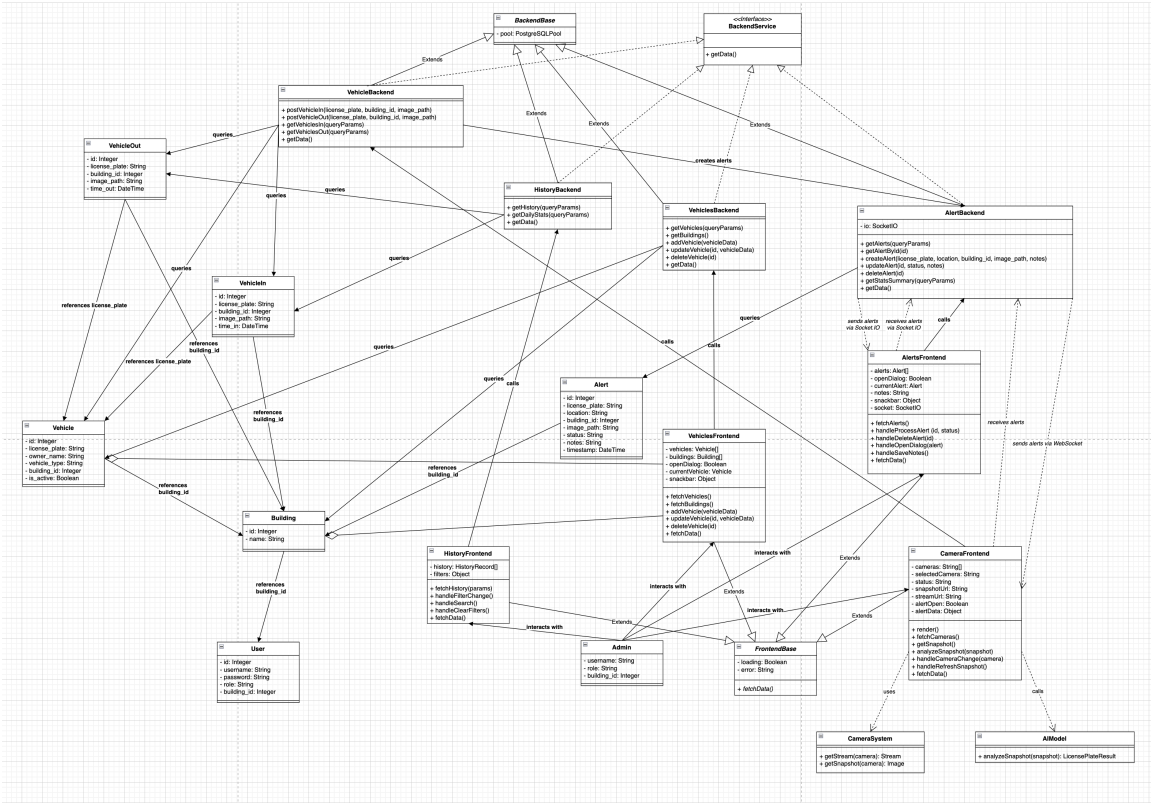
Post-condition: Mọi lượt di chuyển ra vào nhà trọ của các phương tiện đều được kiểm soát và lưu lại, từ đó công tác quản lý xe ra vào nhà trọ của quản trị viên trở nên dễ dàng, thuận tiện và hiệu quả hơn.

Exceptions:

- *Nếu hệ thống trong thời gian bảo trì*:
  - Hệ thống hiển thị “Hệ thống đang bảo trì. Vui lòng thử lại sau.”.
- *Nếu thiết bị của quản trị viên không được kết nối Internet:*
  - Hệ thống hiển thị “Không có kết nối Internet. Vui lòng kiểm tra lại.”.
- *Nếu camera không hoạt động*:
  - Hệ thống hiển thị “NO SIGNAL” với nền đen ở các ô camera và Frame.



Các lớp và các hàm trong lớp:


BackendBase
- pool: PostgreSQLPool
<<Interface>>
BackendService
+ getData()
Extends
VehicleBackend
+ postVehicleIn(license_plate, building_id, image_path)
+ postVehicleOut(license_plate, building_id, image_path)
+ getVehiclesIn(queryParams)
+ getVehiclesOut(queryParams)
+ getData()
VehicleOut
- id: Integer
- license_plate: String
- building_id: Integer
- image_path: String
- time_out: DateTime
queries
creates alerts
HistoryBackend
+ getHistory(queryParams)
+ getDailyStats(queryParams)
+ getData()
VehiclesBackend
+ getVehicles(queryParams)
+ getBuildings()
+ addVehicle(vehicleData)
+ updateVehicle(id, vehicleData)
+ deleteVehicle(id)
+ getData()
AlertBackend
- io: SocketIO
+ getAlerts(queryParams)
+ getAlertById(id)
+ createAlert(license_plate, location, building_id, image_path, notes)
+ updateAlert(id, status, notes)
+ deleteAlert(id)
+ getStatsSummary(queryParams)
+ getData()
VehicleIn
- id: Integer
- license_plate: String
- building_id: Integer
- image_path: String
- time_in: DateTime
references license_plate
references building_id
Alert
- id: Integer
- license_plate: String
- location: String
- building_id: Integer
- image_path: String
- status: String
- notes: String
- timestamp: DateTime
calls
sends alerts via Socket.IO
receives alerts via Socket.IO
AlertsFrontend
- alerts: Alert[]
- openDialog: Boolean
- currentAlert: Alert
- notes: String
- snackbar: Object
- socket: SocketIO
+ fetchAlerts()
+ handleProcessAlert (id, status)
+ handleDeleteAlert(id)
+ handleOpenDialog(alert)
+ handleSaveNotes()
+ fetchData()
receives alerts
sends alerts via WebSocket
Vehicle
- id: Integer
- license_plate: String
- owner_name: String
- vehicle_type: String
- building_id: Integer
- is_active: Boolean
VehiclesFrontend
- vehicles: Vehicle[]
- buildings: Building[]
- openDialog: Boolean
- currentVehicle: Vehicle
- snackbar: Object
+ fetchVehicles()
+ fetchBuildings()
+ addVehicle(vehicleData)
+ updateVehicle(id, vehicleData)
+ deleteVehicle(id)
+ fetchData()
Building
- id: Integer
- name: String
HistoryFrontend
- history: HistoryRecord[]
- filters: Object
+ fetchHistory(params)
+ handleFilterChange()
+ handleSearch()
+ handleClearFilters()
+ fetchData()
interacts with
CameraFrontend
- cameras: String[]
- selectedCamera: String
- status: String
- snapshotUrl: String
- streamUrl: String
- alertOpen: Boolean
- alertData: Object
+ render()
+ fetchCameras()
+ getSnapshot()
+ analyzeSnapshot(snapshot)
+ handleCameraChange(camera)
+ handleRefreshSnapshot()
+ fetchData()
User
- id: Integer
- username: String
- password: String
- role: String
- building_id: Integer
Admin
- username: String
- role: String
- building_id: Integer
FrontendBase
- loading: Boolean
- error: String
+ fetchData()
uses
calls
CameraSystem
+ getStream(camera): Stream
+ getSnapshot(camera): Image
AIModel
+ analyzeSnapshot(snapshot): LicensePlateResult