ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

ARCHITECTURE

FIT TRACK

Ứng dụng tính toán và theo dõi dinh dưỡng & vận động cá nhân

(Personal Nutrition & Activity Tracker)

Thành Phố Hồ Chí Minh – 06/2025

MỤC LỤC

THÔNG TIN THÀNH VIÊN

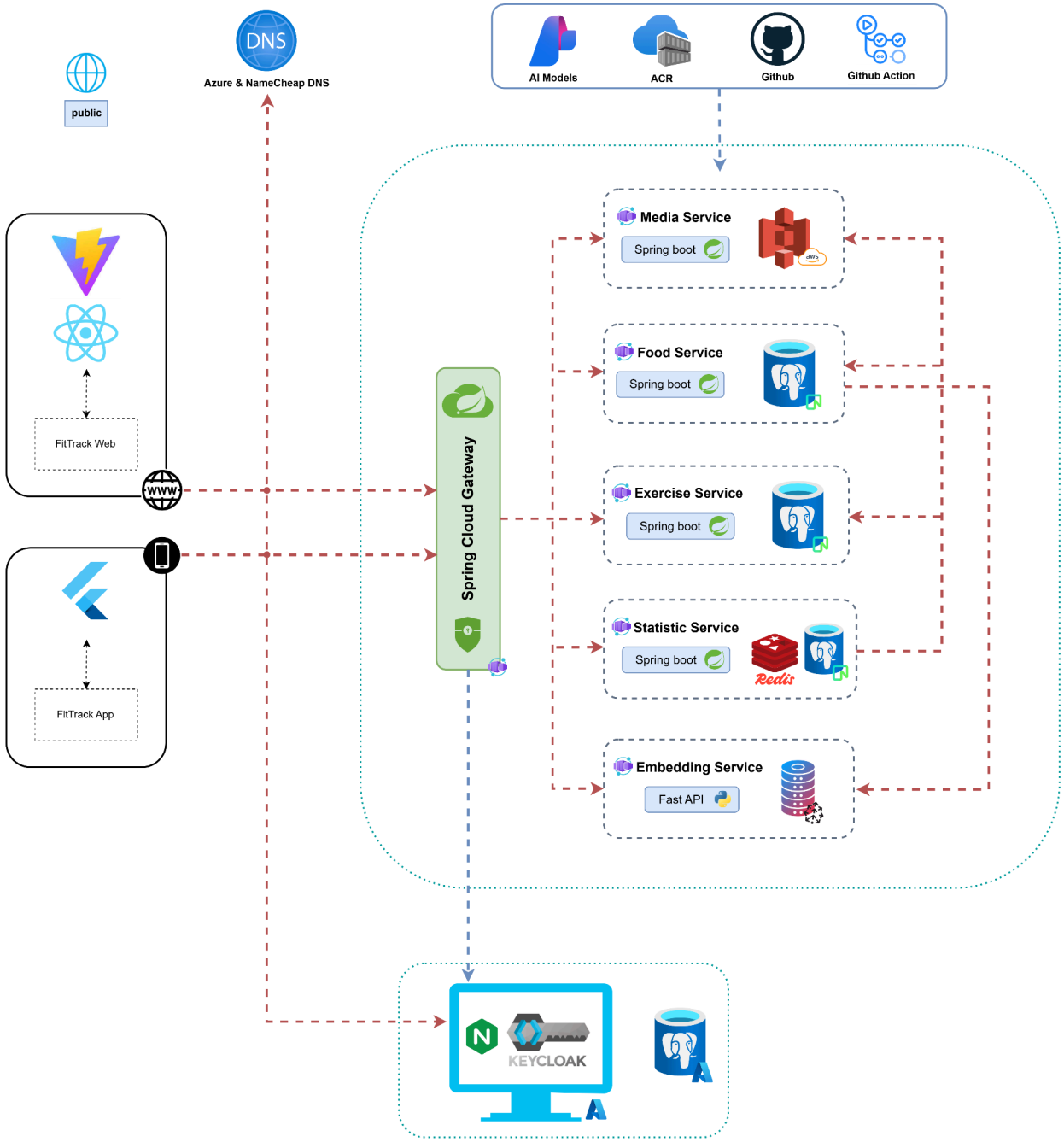
MSSV	Họ Và Tên	Email
20120626	Phạm Khánh Hoàng Việt	phamviet12092002@gmail.com
20120627	Hoàng Vinh	vinhtenbivn@gmail.com
21120093	Trần Anh Kiệt	anhkiet07012003@gmail.com
21120525	Cao Nhật Phong	21120525@student.hcmus.edu.vn
21120540	Trần Tôn Bửu Quang	buuquang102@gmail.com
21120543	Nguyễn Đặng Quốc	ndquocstudy@gmail.com
21120560	Nguyễn Đức Thiện	ndtkhtnk21@gmail.com
21120585	Lê Anh Tú	cubeaholic03@gmail.com
21120596	Trần Đoàn Thanh Vinh	thanhvinh.htn2020@gmail.com

I. Kiến trúc hệ thống theo Logical View

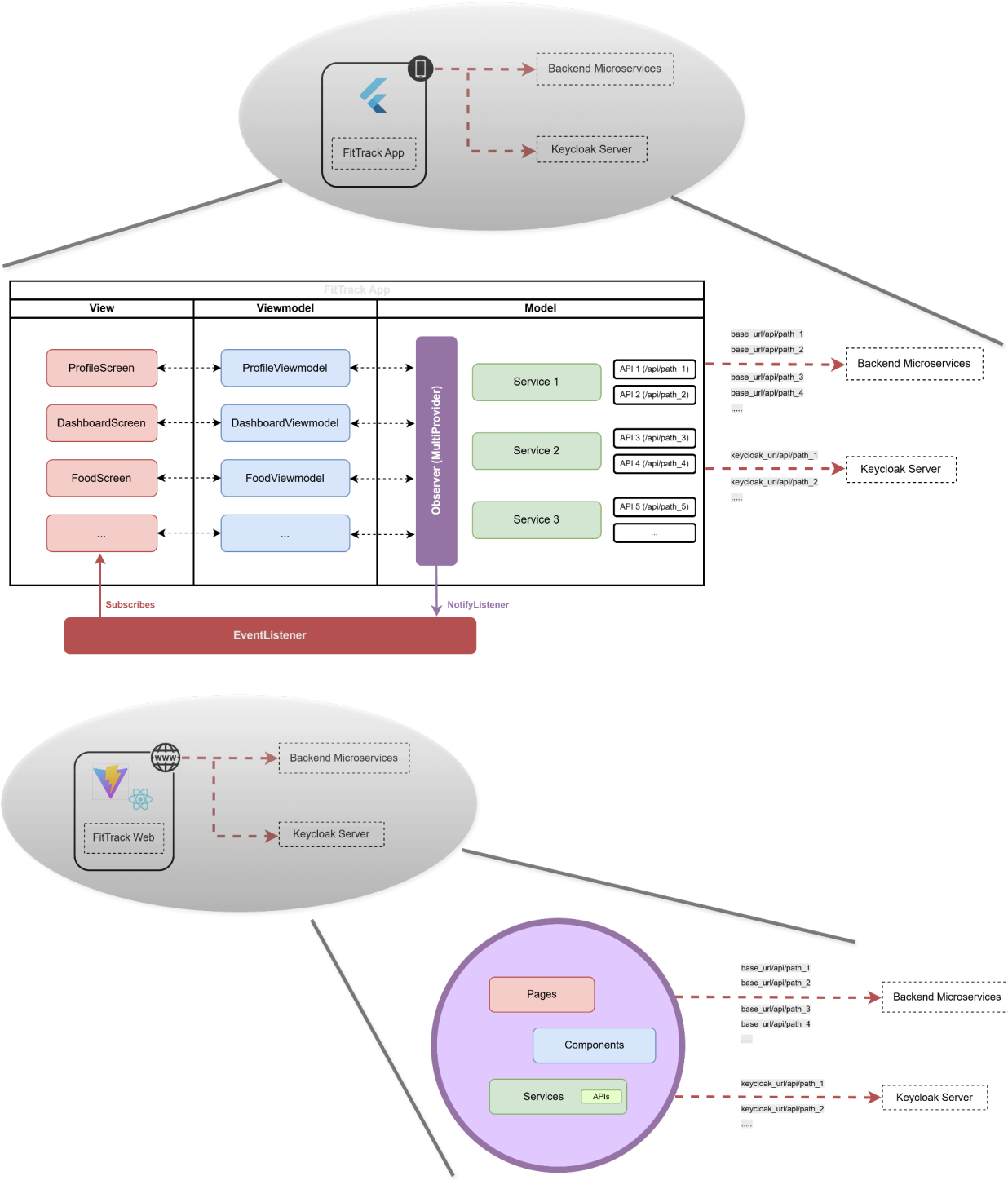
1. Kiến trúc tổng thể

Ứng dụng FitTrack được xây dựng theo kiến trúc Microservices, cho phép mỗi service hoạt động độc lập, dễ dàng mở rộng và bảo trì. Các service này giao tiếp với nhau thông qua API RESTful là chủ yếu.

Tổng thể:



Trong đó:


Backend Microservices
FitTrack App
Keycloak Server
FitTrack App
View
Viewmodel
Model
ProfileScreen
DashboardScreen
FoodScreen
...
ProfileViewmodel
DashboardViewmodel
FoodViewmodel
...
Observer (MultiProvider)
Service 1
Service 2
Service 3
API 1 (/api/path_1)
API 2 (/api/path_2)
API 3 (/api/path_3)
API 4 (/api/path_4)
API 5 (/api/path_5)
...
base_url/api/path_1
base_url/api/path_2
base_url/api/path_3
base_url/api/path_4
.....
Backend Microservices
keycloak_url/api/path_1
keycloak_url/api/path_2
.....
Keycloak Server
Subscribes
NotifyListener
EventListener
Backend Microservices
FitTrack Web
Keycloak Server
Pages
Components
Services
APIs
base_url/api/path_1
base_url/api/path_2
base_url/api/path_3
base_url/api/path_4
.....
Backend Microservices
keycloak_url/api/path_1
keycloak_url/api/path_2
.....
Keycloak Server

2. Frontend

a) App Mobile

Có sự phân lớp chức năng. Ứng dụng sẽ được chia thành ba nhóm chức năng chính, thể hiện sự tách biệt rõ ràng về mặt nghiệp vụ:

- **Authentication:** Quản lý việc tạo tài khoản, xác thực và ủy quyền người dùng như đăng ký, đăng nhập, quên mật khẩu, ... Quản lý thông tin cá nhân của người dùng. Nhóm này sẽ call api tới keycloak server.
- **Fitness:** Cung cấp các tính năng liên quan đến theo dõi và quản lý hoạt động dinh dưỡng của người dùng như tìm và ghi lại bài tập exercise, ghi lại bữa ăn meal & food. Có thể ngầm hiểu qua "food" và "exercise". Nhóm này sẽ call api tới microservice backend, sử dụng host name của api gateway, các service bên trong chỉ có thể call nội bộ.
- **Statistic:** Hiển thị các số liệu thống kê và phân tích liên quan đến dữ liệu người dùng về các hoạt động dinh dưỡng và thể chất như cân nặng, bước chân, goal, calories, … Nhóm này sẽ call api tới microservice backend, sử dụng host name của api gateway, các service bên trong chỉ có thể call nội bộ.

Vai trò: Xử lý trực tiếp tương tác của người dùng, cung cấp giao diện người dùng trực quan và gửi các yêu cầu API đến backend để thực hiện các nghiệp vụ.
Công nghệ: Sử dụng Flutter framework, cho phép phát triển ứng dụng đa nền tảng mà trong trường hợp này là Android.

b) Web Admin

Web React, sử dụng kiến trúc monolith. Web dành riêng cho admin để quản lý danh sách người dùng, nhập liệu food và exercise một cách hiệu quả.

3. Backend

Xây dựng một hệ thống **Identity and Access Management** bằng giải pháp mã nguồn mở Keycloak. Khi cài đặt và config xong, nó cung cấp các API đăng nhập, đăng ký, hỗ trợ quản lý tài khoản bằng UI sẵn có, … Các hệ thống FitTrack sẽ uỷ quyền các tính năng về authentication và security cho Keycloak quản lý. Các client (web và mobile) sau khi đăng nhập, sẽ sử dụng các token mà Keycloak cung cấp để đính kèm vào các request gửi đi.
Áp dụng xây dựng kiến trúc **Microservices** cho Backend:

- GatewayService: Sử dụng framework của Spring là Spring Cloud Gateway để xây dựng một API Gateway riêng, giúp chủ động trong việc config để giao tiếp với

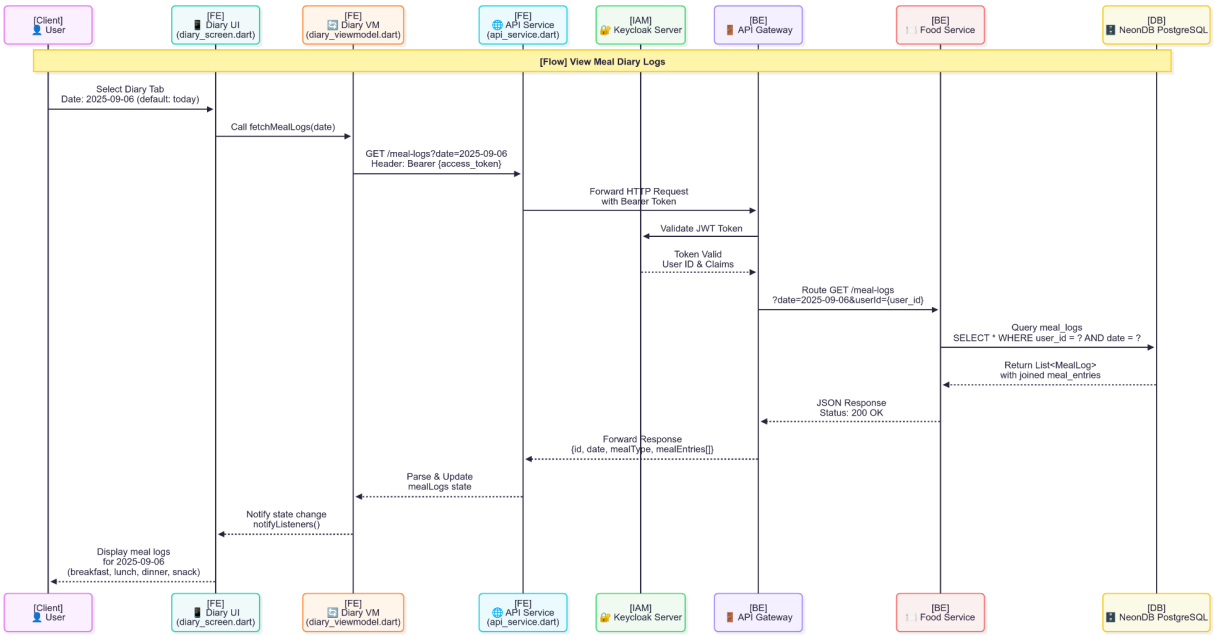
Keycloak server. Đồng thời cũng hỗ trợ config các tính năng mà một API Gateway thường có như Routing, Logging, … Đây là điểm truy cập duy nhất của hệ thống backend, chuyển hướng yêu cầu đến các service phù hợp. Các service bên trong chỉ có thể giao tiếp nội bộ với nhau, không thể được gọi trực tiếp từ bên ngoài.

- FoodService: Service giúp quản lý tất cả các khía cạnh liên quan thực phẩm, dinh dưỡng món ăn, nhật ký món ăn, …
- ExerciseService: Service giúp xử lý mọi thứ liên quan đến hoạt động exercise, theo dõi hoạt động, …
- StatisticService: Service tập trung vào việc thu thập, xử lý và trình bày số liệu thống kê về sức khỏe của người dùng và thống kê liên quan tới admin.
- MediaService: Service chịu trách nhiệm quản lý việc xử lý file ảnh.
- EmbeddingService: Service hỗ trợ tính năng phân tích món ăn thông minh (Xem POC để biết thêm chi tiết).

Các công nghệ sử dụng:

- Các service đa số áp dụng Spring framework. Riêng EmbeddingService áp dụng FastAPI framework vì nó có tính năng liên quan tới AI.
- Database của các service đa số là PostgreSQL. EmbeddingService thì sử dụng Vector Database.

II. Kiến trúc hệ thống theo Process View



Luồng xử lý:

- Khởi động ứng dụng: Người dùng mở app Android, đăng nhập qua Keycloak, nhận access token.
- Tương tác UI:
  + Component: diary_screen.dart (Flutter).
  + Chọn tab Diary, ngày (mặc định: 2025-09-06), gọi fetchMealLogs trong diary_viewmodel.dart.
- Xử lý ViewModel:
  + Component: diary_viewmodel.dart.
  + Quản lý state, gọi API qua api_service.dart với GET /meal-logs?date=2025-09-06, kèm Bearer {access_token}.
- Gọi API:
  + Component: api_service.dart.
  + Gửi request tới API Gateway, xử lý response, trả dữ liệu cho ViewModel.
- Xác thực và định tuyến:
  + Component: GatewayService (Spring Cloud Gateway). Một Service trong Microservice.
  + Nhận request gửi tới từ app mobile. Lấy và gửi token tới Keycloak Server để xác thực, trích xuất user ID truyền vào header, định tuyến tới FoodService.

- Xử lý FoodService:
    + Component chính: FoodController.java, FoodService.java, FoodRepository.java.
    + Truy vấn tới Database trên Neon là fooddb: SELECT * FROM meal_logs WHERE user_id = ? AND date = ?.
- Trả kết quả:
    + Database trả về List<MealLog> qua FoodService, Gateway, tới api_service.dart.
    + ViewModel cập nhật state, thông báo UI hiển thị meal logs (breakfast, lunch, dinner, snack) của ngày đã chọn.

III. Kiến trúc hệ thống theo Development View

Source code: Github

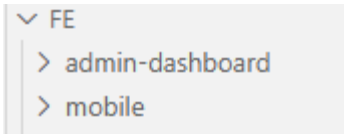
Hệ thống FitTrack là một ứng dụng toàn diện hỗ trợ người dùng theo dõi dinh dưỡng, tập luyện và thống kê liên quan đến sức khỏe. Mã nguồn của hệ thống được tổ chức một cách rõ ràng thành hai phần chính: Frontend (FE) và Backend (BE).

Dưới đây là cấu trúc thư mục tổng quan của dự án:

```
Source_Code
├── FE/                          # Thư mục chứa mã nguồn Frontend
│   ├── admin-dashboard/         # Giao diện quản trị web
│   └── mobile/                  # Ứng dụng di động
└── BE/                          # Thư mục chứa mã nguồn Backend
    └── services/                # Chứa các Microservices
        ├── embedding-service/   # Python - FastAPI
        ├── exercise-service/    # Java - Spring Boot
        ├── food-service/        # Java - Spring Boot
        ├── gateway-service/     # Java - Spring Boot
        ├── media-service/       # Java - Spring Boot
        └── statistic-service/   # Java - Spring Boot
```

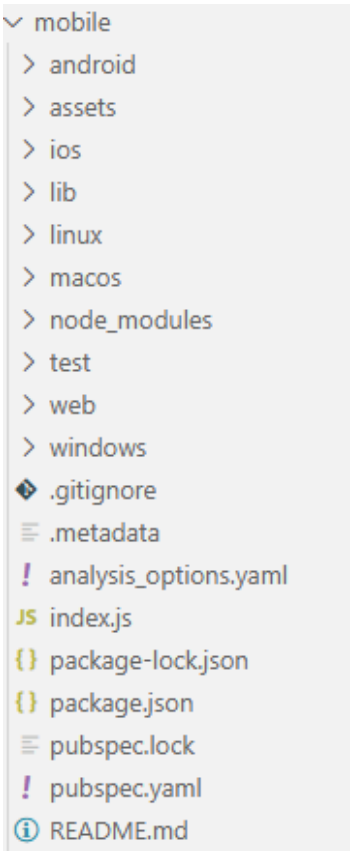
1. Frontend (FE)

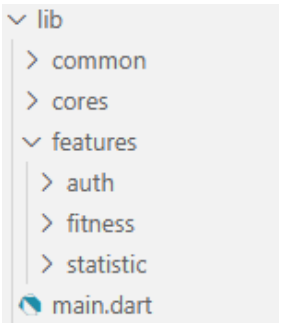
Phần Frontend của hệ thống FitTrack bao gồm hai ứng dụng riêng biệt: một ứng dụng di động (Mobile App) và một giao diện quản trị web (Web Admin).



a) Mobile App

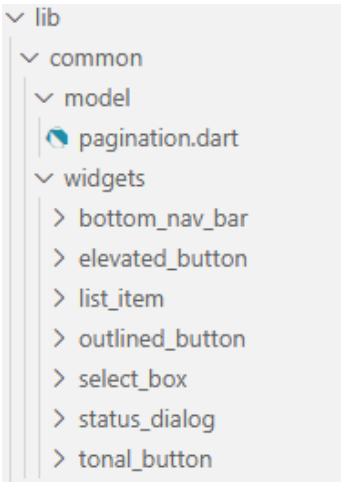
Ứng dụng di động được phát triển bằng Flutter, cho phép triển khai trên nhiều nền tảng từ một cơ sở mã duy nhất.



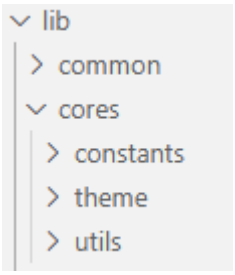


Giới thiệu:

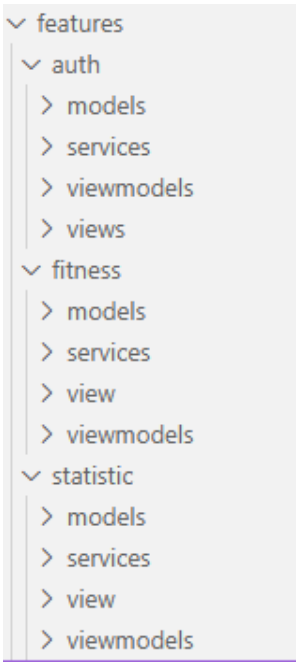
- **pubspec.yaml** chứa thông tin các dependencies và thông tin ứng dụng
- **lib/common** chứa các widget dùng chung trong app



- **lib/cores** chứa constant, theme, thông tin endpoint api và cách gọi api.

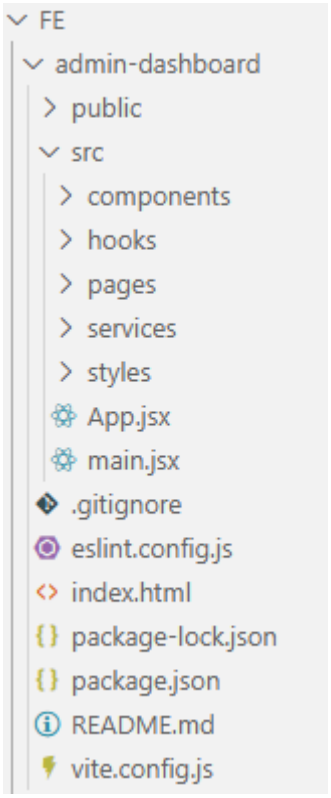


- **lib/features/..** mỗi feature bao gồm:
    + model: lớp dữ liệu (class user, food,...)
    + service: tương tác với backend bằng api
    + view: widget UI màn hình
    + viewmodel: cung cấp dữ liệu cho view, lấy dữ liệu từ service



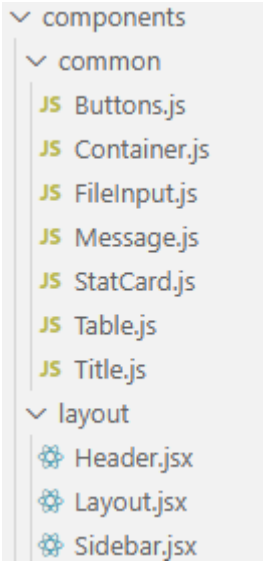
b) Web Admin

Giao diện quản trị web được phát triển bằng React (JavaScript/JSX), tập trung vào việc quản lý dữ liệu và người dùng cho hệ thống.

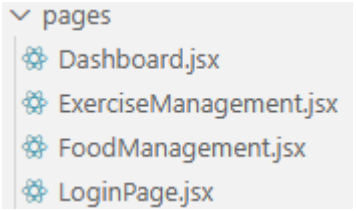


Trong đó:

- src chứa source code chính cho web, sử dụng context hook cho one-way-dataflow.
- components chứa các thành phần UI có thể tái sử dụng



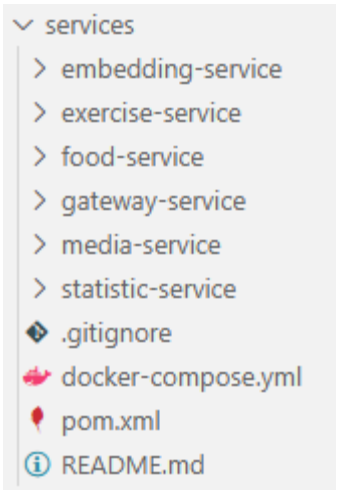
- pages là các component đại diện cho các trang riêng biệt trong ứng dụng quản trị (Dashboard, Quản lý Bài tập, Quản lý Thực phẩm, Đăng nhập)



- services chứa logic giao tiếp với các API Backend (sử dụng Axios)

2. Backend

Phần Backend của hệ thống FitTrack được xây dựng dưới dạng các Microservices, mỗi service tập trung vào một chức năng nghiệp vụ cụ thể. Các service này được phát triển chủ yếu bằng Spring Boot (Java) và FastAPI (Python).



Mỗi service đa số bao gồm:

- pom.xml: chứa thông tin dependencies và ứng dụng
- config: cấu hình ứng dụng
- controller: tương tác với request client, là endpoint kết nối frontend
- dto: định hình dữ liệu request và response
- exception: bắt lỗi exception
- model: đại diện cho các bảng trong database và chứa dữ liệu của ứng dụng
- repository: chịu trách nhiệm tương tác với database
- service: tương tác với repository và trả kết quả về controller

a) Embedding Service (Python - FastAPI)

Chịu trách nhiệm tạo và quản lý các embedding cho dữ liệu food, phục vụ cho các chức năng tìm kiếm thành phần bữa ăn thông minh.

Cấu trúc thư mục:

```
BE/
└── services/
    └── embedding-service/
        ├── app/                    # Mã nguồn chính của ứng dụng FastAPI
        │   ├── main.py             # Điểm vào chính của ứng dụng FastAPI
        │   ├── api/                # Định nghĩa các API endpoints
        │   │   └── endpoints/      # Các endpoint cụ thể
        │   │       ├── food.py     # Endpoint liên quan đến thực phẩm
        │   │       └── search.py   # Endpoint liên quan đến phân tích thành phần món ăn
        │   ├── config/             # Cấu hình ứng dụng
```

```
│   │       settings.py          # Chứa các cấu hình của ứng dụng
│   ├── core/                    # Logic cốt lõi của dịch vụ
│   │       embedding.py         # Xử lý việc tạo và quản lý các vector embedding
│   │       search.py            # Chứa logic cho tính năng AI
│   │       vectordb.py          # Tương tác với Vector DB, thao tác với các embedding.
│   └── models/                  # Pydantic models cho request/response API
│           food.py
│           search.py
├── Dockerfile                   # Dockerfile để xây dựng Docker image
├── Dockerfile.dev               # Dockerfile cho môi trường phát triển
├── requirements.txt             # Danh sách các thư viện Python cần thiết
└── .env                         # File cấu hình môi trường chứa các biến môi trường
```

Database (DB) của Embedding Service: Service này tương tác với một Vector Database (Pinecone) để lưu trữ các embedding vector. Cấu hình kết nối DB sẽ nằm trong app/config/settings.py, và logic tương tác (ORM hoặc truy vấn trực tiếp) sẽ được triển khai trong app/core/vectordb.py.

b) Exercise Service (Java - Spring Boot)

Quản lý thông tin về các bài tập, nhật ký tập luyện và các thống kê liên quan đến tập luyện.

Cấu trúc thư mục:

```
BE/
└── services/
    └── exercise-service/
        ├── src/
        │   ├── main/
        │   │   ├── java/
        │   │   │   └── com/hcmus/exerciseservice/
        │   │   │       ├── ExerciseServiceApplication.java
        │   │   │       ├── config/                  # Cấu hình Spring
        │   │   │       │   SecurityConfig.java
        │   │   │       ├── controller/              # Các REST Controller
        │   │   │       │   ExerciseController.java
        │   │   │       │   ExerciseLogController.java
```

```
│   │   │   │   ExerciseLogEntryController.java
│   │   │   │   ExerciseReportController.java
│   │   │   ├── dto/                    # Data Transfer Objects
│   │   │   │   ├── request/
│   │   │   │   │   ExerciseLogEntryRequest.java
│   │   │   │   │   ExerciseRequest.java
│   │   │   │   │   InitiateExerciseLogRequest.java
│   │   │   │   └── response/
│   │   │   │       ApiResponse.java
│   │   │   │       ExerciseCaloriesResponse.java
│   │   │   │       ExerciseLogEntryResponse.java
│   │   │   │       ExerciseLogResponse.java
│   │   │   │       ExerciseReportResponse.java
│   │   │   │       TotalCaloriesBurnedResponse.java
│   │   │   ├── exception/              # Các lớp xử lý ngoại lệ tùy chỉnh
│   │   │   │   GlobalExceptionHandler.java
│   │   │   │   InvalidTokenException.java
│   │   │   │   ResourceAlreadyExistsException.java
│   │   │   │   ResourceNotFoundException.java
│   │   │   ├── mapper/                 # Các lớp ánh xạ DTO và Model
│   │   │   │   ExerciseLogEntryMapper.java
│   │   │   │   ExerciseLogMapper.java
│   │   │   │   ExerciseMapper.java
│   │   │   ├── model/                  # ORM
│   │   │   │   Exercise.java
│   │   │   │   ExerciseLog.java
│   │   │   │   ExerciseLogEntry.java
│   │   │   ├── repository/             # Các Spring Data JPA Repository
│   │   │   │   ExerciseLogEntryRepository.java
│   │   │   │   ExerciseLogRepository.java
│   │   │   │   ExerciseRepository.java
│   │   │   ├── security/               # Cấu hình các handler
│   │   │   │   CustomAccessDeniedHandler.java
```

```
│   │   │       │       CustomAuthenticationEntryPoint.java
│   │   │       ├── service/                    # Logic nghiệp vụ chính
│   │   │       │   │   ExerciseLogEntryService.java
│   │   │       │   │   ExerciseLogService.java
│   │   │       │   │   ExerciseReportService.java
│   │   │       │   │   ExerciseService.java
│   │   │       │   └── impl/
│   │   │       │           ExerciseLogEntryServiceImpl.java
│   │   │       │           ExerciseLogServiceImpl.java
│   │   │       │           ExerciseReportServiceImpl.java
│   │   │       │           ExerciseServiceImpl.java
│   │   │       └── util/                       # Các lớp tiện ích chung (.java)
│   │   │               CustomSecurityContextHolder.java
│   │   └── resources/                          # Tài nguyên ứng dụng
│   │       application.yml
│   └── test/                                   # Các tệp kiểm thử đơn vị và tích hợp
│       └── java/
│           └── com/hcmus/exerciseservice/
│                   ExerciseserviceApplicationTests.java
├── pom.xml                                     # Tệp cấu hình Maven
├── Dockerfile                                  # Dockerfile để build Docker image
├── Dockerfile.dev                              # Dockerfile cho môi trường phát triển
└── .env                                        # File cấu hình môi trường
```

Database (DB) của Exercise Service: Service này sử dụng PostgreSQL làm cơ sở dữ liệu quan hệ. Spring Data JPA (ORM) được sử dụng để tương tác với cơ sở dữ liệu, và cấu hình kết nối nằm trong src/main/resources/application.yml.

c) Food Service (Java - Spring Boot)

Tương tự với Exercise Service

d) Media Service (Java - Spring Boot)

Tương tự với Exercise Service

e) Gateway Service (Java - Spring Boot)

Hoạt động như một điểm truy cập duy nhất cho tất cả các yêu cầu từ Frontend đến các Microservices Backend. Nó xử lý định tuyến, bảo mật (xác thực và ủy quyền).

Cấu trúc thư mục:

```
BE/
└── services/
    └── gateway-service/
        ├── src/
        │   ├── main/
        │   │   ├── java/
        │   │   │   └── com/hcmus/gatewayservice/
        │   │   │       ├── GatewayServiceApplication.java
        │   │   │       ├── config/                          # Cấu hình Gateway
        │   │   │       │   CorsConfig.java                  # Cấu hình CORS
        │   │   │       │   RequestLoggingFilter.java        # Ghi log các yêu cầu đi qua
        │   │   │       │   SecurityConfig.java              # Cấu hình bảo mật
        │   │   │       │   TokenHeaderFilter.java          # Xử lý và chuyển tiếp JWT
        │   │   │       ├── controller/                      # Các Fallback Controller
        │   │   │       │   FallbackController.java         # Xử lý khi có lỗi
        │   │   │       ├── dto/                             # Data Transfer Objects
        │   │   │       │   ApiResponse.java
        │   │   │       └── security/                        # Cấu hình các handler
        │   │   │           CustomAccessDeniedHandler.java
        │   │   │           CustomAuthenticationEntryPoint.java
        │   │   └── resources/                               # Tài nguyên ứng dụng
        │   │       application.yml                          # Cấu hình định tuyến và bảo mật
        │   └── test/                                        # Các tệp kiểm thử
        │       └── java/
        │           └── com/hcmus/gatewayservice/
        │               GatewayServiceApplicationTests.java
        ├── pom.xml                                          # Tệp cấu hình Maven
        ├── Dockerfile
        ├── Dockerfile.dev
```

└── .env

Database (DB) của Gateway Service: Service này không tương tác trực tiếp với cơ sở dữ liệu nghiệp vụ mà chỉ định tuyến các yêu cầu đến các microservices khác.

f) Statistic Service (Java - Spring Boot)

Chuyên về tổng hợp và cung cấp các thống kê, báo cáo về hoạt động của người dùng (dinh dưỡng, tập luyện, cân nặng, bước chân).

Áp dụng DDD (Domain-Driven Design) với các lớp chính:

- Domain Layer (domain/): Chứa logic nghiệp vụ cốt lõi và định nghĩa miền:
    + model/: Các Entities (thực thể) chính của service (ví dụ: FitProfile, WeightLog, NutritionGoal).
    + exception/: Các ngoại lệ đặc trưng của miền.
    + repository/: Các Interface (hợp đồng) định nghĩa cách thức ứng dụng tương tác với cơ sở dữ liệu cho các Aggregate Root.
- Application Layer (application/): Điều phối các tác vụ nghiệp vụ và sử dụng Domain Layer:
    + dto/: Các Data Transfer Objects (DTOs) cho request và response API.
    + mapper/: Các lớp ánh xạ giữa DTO và Domain Model để đảm bảo cấu trúc dữ liệu.
    + service/: Xử lý các trường hợp sử dụng (use cases) và logic tính toán nghiệp vụ không thuộc trực tiếp về một Entity cụ thể.
- Infrastructure Layer (infrastructure/): Chứa tất cả các chi tiết kỹ thuật bên ngoài liên quan đến việc triển khai các yêu cầu của lớp domain và application:
    + client/: Feign Clients để gọi các microservices khác (Exercise, Food, Media).
    + config/: Cấu hình cho PostgreSQL, Redis, và các dịch vụ khác.
    + exception/: Các ngoại lệ ở tầng hạ tầng.
    + repository/: Triển khai cụ thể các interface repository của Domain (sử dụng Spring Data JPA).
    + security/: Cấu hình bảo mật và các lớp liên quan.
- Presentation Layer (presentation/): Chịu trách nhiệm hiển thị thông tin và nhận đầu vào:
    + controller/: Các REST Controller định nghĩa các endpoint API để Frontend hoặc các hệ thống khác tương tác.

Cấu trúc thư mục:

```
BE/
└── services/
    └── statistic-service/
        ├── src/
        │   ├── main/
        │   │   ├── java/
        │   │   │   └── com/hcmus/statisticservice/
        │   │   │       ├── StatisticServiceApplication.java
        │   │   │       ├── application/
        │   │   │       │   ├── dto/
        │   │   │       │   │   ├── request/
        │   │   │       │   │   │   AddWeightLogRequest.java
        │   │   │       │   │   │   CreateWeightGoalRequest.java
        │   │   │       │   │   │   ExerciseRequest.java
        │   │   │       │   │   │   FoodRequest.java
        │   │   │       │   │   │   SaveSurveyRequest.java
        │   │   │       │   │   │   StepLogRequest.java
        │   │   │       │   │   │   UpdateProfileRequest.java
        │   │   │       │   │   │   UpdateWeightGoalRequest.java
        │   │   │       │   │   │   WeightLogRequest.java
        │   │   │       │   │   └── response/
        │   │   │       │   │       AdminReportResponse.java
        │   │   │       │   │       ApiResponse.java
        │   │   │       │   │       CheckSurveyResponse.java
        │   │   │       │   │       DashboardResponse.java
        │   │   │       │   │       ExerciseReportResponse.java
        │   │   │       │   │       FitProfileResponse.java
        │   │   │       │   │       FoodReportResponse.java
        │   │   │       │   │       GetNutritionGoalResponse.java
        │   │   │       │   │       GetWeightGoalResponse.java
        │   │   │       │   │       StepLogResponse.java
        │   │   │       │   │       TotalCaloriesBurnedResponse.java
        │   │   │       │   │       TotalCaloriesConsumedResponse.java
```

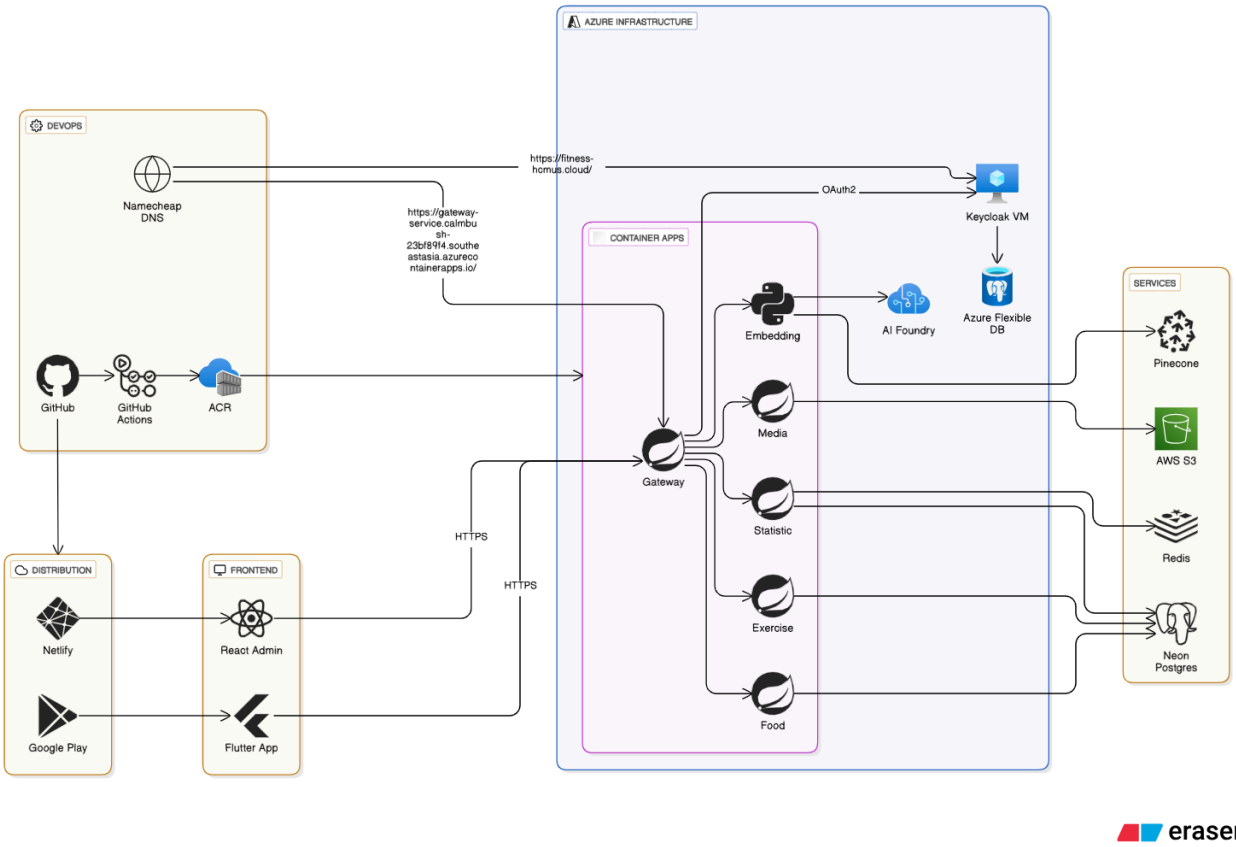
```
│   │   │   │   │   WeightGoalResponse.java
│   │   │   │   │   WeightLogResponse.java
│   │   │   │   ├── mapper/
│   │   │   │   │   FitProfileMapper.java
│   │   │   │   │   WeightGoalMapper.java
│   │   │   │   │   WeightLogMapper.java
│   │   │   │   └── service/
│   │   │   │       │   AdminReportService.java
│   │   │   │       │   DashboardService.java
│   │   │   │       │   FitProfileService.java
│   │   │   │       │   LatestLoginService.java
│   │   │   │       │   NutritionGoalService.java
│   │   │   │       │   StatisticService.java
│   │   │   │       │   StepLogService.java
│   │   │   │       │   SurveyService.java
│   │   │   │       │   WeightGoalService.java
│   │   │   │       │   WeightLogService.java
│   │   │   │       └── impl/
│   │   │   │               AdminReportServiceImpl.java
│   │   │   │               DashboardServiceImpl.java
│   │   │   │               FitProfileServiceImpl.java
│   │   │   │               LatestLoginServiceImpl.java
│   │   │   │               NutritionGoalServiceImpl.java
│   │   │   │               StepLogServiceImpl.java
│   │   │   │               SurveyServiceImpl.java
│   │   │   │               WeightGoalServiceImpl.java
│   │   │   │               WeightLogServiceImpl.java
│   │   │   ├── domain/
│   │   │   │   ├── exception/
│   │   │   │   │   NutritionException.java
│   │   │   │   │   StatisticException.java
│   │   │   │   ├── model/
│   │   │   │   │   │   FitProfile.java
```

```
│   │   │   │   │   │   LatestLogin.java
│   │   │   │   │   │   NutritionData.java
│   │   │   │   │   │   NutritionGoal.java
│   │   │   │   │   │   StepLog.java
│   │   │   │   │   │   WeightGoal.java
│   │   │   │   │   │   WeightLog.java
│   │   │   │   │   │   └── type/
│   │   │   │   │   │       ActivityLevel.java
│   │   │   │   │   │       Gender.java
│   │   │   │   │   │       Goal.java
│   │   │   │   │   └── repository/
│   │   │   │   │       FitProfileRepository.java
│   │   │   │   │       LatestLoginRepository.java
│   │   │   │   │       NutritionGoalRepository.java
│   │   │   │   │       StepLogRepository.java
│   │   │   │   │       WeightGoalRepository.java
│   │   │   │   │       WeightLogRepository.java
│   │   │   │   ├── infrastructure/
│   │   │   │   │   ├── client/
│   │   │   │   │   │   ExerciseServiceClient.java
│   │   │   │   │   │   FeignConfig.java
│   │   │   │   │   │   FoodServiceClient.java
│   │   │   │   │   │   MediaServiceClient.java
│   │   │   │   │   ├── config/
│   │   │   │   │   │   RedisConfig.java
│   │   │   │   │   │   RepositoryConfig.java
│   │   │   │   │   ├── exception/
│   │   │   │   │   │   GlobalExceptionHandler.java
│   │   │   │   │   ├── repository/
│   │   │   │   │   │   FitProfileRepositoryAdapter.java
│   │   │   │   │   │   JpaFitProfileRepository.java
│   │   │   │   │   │   JpaLatestLoginRepository.java
│   │   │   │   │   │   JpaNutritionGoalRepository.java
```

```
│   │   │       │   │       JpaStepLogRepository.java
│   │   │       │   │       JpaWeightGoalRepository.java
│   │   │       │   │       JpaWeightLogRepository.java
│   │   │       │   │       LatestLoginRepositoryAdapter.java
│   │   │       │   │       NutritionGoalRepositoryAdapter.java
│   │   │       │   │       StepLogRepositoryAdapter.java
│   │   │       │   │       WeightGoalRepositoryAdapter.java
│   │   │       │   │       WeightLogRepositoryAdapter.java
│   │   │       │   └── security/
│   │   │       │           CustomAccessDeniedHandler.java
│   │   │       │           CustomAuthenticationEntryPoint.java
│   │   │       │           CustomSecurityContextHolder.java
│   │   │       │           SecurityConfig.java
│   │   │       └── presentation/
│   │   │           └── controller/
│   │   │                   AdminReportController.java
│   │   │                   DashboardController.java
│   │   │                   FitProfileController.java
│   │   │                   LatestLoginController.java
│   │   │                   NutritionGoalController.java
│   │   │                   StatisticController.java
│   │   │                   StepLogController.java
│   │   │                   SurveyController.java
│   │   │                   WeightGoalController.java
│   │   │                   WeightLogController.java
│   │   └── resources/
│   │           application.yml
│   └── test/
│       └── java/
│           └── com/hcmus/statisticservice/
│                   StatisticSerivceApplicationTests.java
│                   ├── application/
│                   │   └── service/
```

```
│               │           FitProfileServiceTest.java
│               ├── infrastructure/
│               │   └── repository/
│               │           FitProfileRepositoryAdapterTest.java
│               └── presentation/
│                   └── controller/
│                           FitProfileControllerTest.java
├── pom.xml
├── Dockerfile
├── Dockerfile.dev
└── .env
```

IV. Kiến trúc hệ thống theo Deployment View



V. Công nghệ, công cụ được lựa chọn để xây dựng hệ thống FitTrack

1. Công nghệ

Ngôn ngữ lập trình:

- Java (Spring Boot): Ngôn ngữ chính cho backend microservices với khả năng mở rộng cao và hệ sinh thái phong phú
- Python (FastAPI): Ngôn ngữ cho Embedding service, tối ưu cho xử lý AI và machine learning
- Dart (Flutter): Ngôn ngữ cho phát triển ứng dụng mobile đa nền tảng
- JavaScript (React): Ngôn ngữ cho phát triển web admin interface

Database:

- PostgreSQL (NeonDB): Database quan hệ serverless, tương thích hoàn toàn với PostgreSQL, hỗ trợ auto-scaling
    + Statistics Database: Lưu trữ dữ liệu thống kê người dùng.

    + Exercise Database: Quản lý thông tin bài tập và workout.
    + Food Database: Lưu trữ dữ liệu dinh dưỡng và thực phẩm.
- Redis (cloud.redis.io): In-memory cache database cho việc tăng tốc truy vấn thống kê.
- Vector Database (Pinecone): Specialized database cho AI embedding và similarity search
- Azure Flexible Database: PostgreSQL database cho Keycloak authentication service

Authentication & Authorization:

- Keycloak: Open-source identity and access management solution
- JWT (JSON Web Token)

Cloud Infrastructure:

- Microsoft Azure: Primary cloud platform
- Azure Container Apps: Serverless container hosting cho microservices
- Azure Linux VM: Hosting Keycloak server
- Azure AI Foundry: AI models và services
- AWS S3: Object storage cho media files
- Netlify: Static site hosting cho React web admin

2. Công cụ

Trình soạn thảo:

- Visual Studio Code: IDE chính cho JavaScript/TypeScript và Python development
- IntelliJ IDEA: IDE chuyên nghiệp cho Java Spring Boot development
- Android Studio: IDE chính thức cho Flutter mobile development

Quản lý mã nguồn:

- GitHub: Version control system và source code repository
- Git: Distributed version control system

Tích hợp, kiểm thử, triển khai:

- GitHub Actions: CI/CD pipeline automation
    + Automated testing và code quality checks
    + Docker image building và pushing to ACR
    + Automated deployment to Azure Container Apps
- Azure Container Registry (ACR): Container image registry
- Docker: Containerization platform cho microservices

- Postman: API testing và documentation tool

Monitoring & DevOps:

- Azure Application Insights: Application performance monitoring
- Docker Compose: Local development environment orchestration
- Namecheap DNS: Domain name management

Development & Testing Tools:

- Postman: RESTful API testing và documentation
- Flutter Test: Unit và integration testing cho mobile app
- Jest: JavaScript testing framework cho React components
- JUnit: Unit testing framework cho Spring Boot services

Package Managers:

- npm/yarn: JavaScript package management
- pub: Dart/Flutter package management
- Maven: Java dependency management
- pip: Python package management