**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**





**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**

[**I. Cấu trúc thư mục 4**](#_pbxrbly98a3i)

[1. Frontend 4](#_lsnudffrxjuk)

[2. Backend 4](#_ocb4d8ub6ti6)

[**II. Quy định chung 5**](#_d9yi6bpue7y)

[1. Các nguyên lý cần tuân theo 5](#_szme2tefwojv)

[2. Dart 5](#_6r9acbat02pn)

[a) Quy tắc đặt tên 5](#_8i2v2w4aepl1)

[b) Quy tắc code 6](#_xz6rwqkmql25)

[c) Quy tắc khi merge code 6](#_5i0y8clkhhw7)

[3. Java 7](#_do2hyngwxns4)

[a) Quy tắc chung 7](#_gd2tkefnf2lq)

[b) Quy tắc đặt tên 7](#_g9g6uu4q1o5e)

[c) Quy tắc code 8](#_rk3l3j5ulzm9)

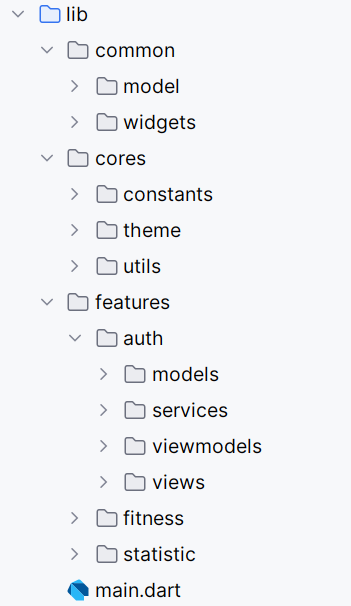
[d) Quy tắc merge code 10](#_42xz1z83a27t)

**THÔNG TIN THÀNH VIÊN**

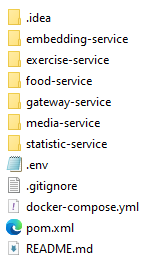
|  |  |  |
| --- | --- | --- |
| MSSV | Họ Và Tên | Email |
| 20120626 | Phạm Khánh Hoàng Việt | phamviet12092002@gmail.com |
| 20120627 | Hoàng Vinh | vinhtenbivn@gmail.com |
| 21120093 | Trần Anh Kiệt | anhkiet07012003@gmail.com |
| 21120525 | Cao Nhật Phong | 21120525@student.hcmus.edu.vn |
| 21120540 | 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 |

# Cấu trúc thư mục

## Frontend



## Backend



Tuỳ chỉnh nhưng đảm bảo theo cấu trúc Springboot nếu không có yêu cầu gì đặc biệt:

* controller
* service
* repository
* dto
* model
* config
* exception

# Quy định chung

## Các nguyên lý cần tuân theo

* Ưu tiên sử dụng cú pháp rút gọn khi có thể.
* Dùng khoảng cách theo tùy chọn định dạng mặc định (ví dụ: Prettier, công cụ tích hợp sẵn của Visual Studio, ...).
* Đặt tên có ý nghĩa (tên file, hàm, biến,...). Chỉ dùng viết tắt cho các trường hợp quen thuộc hoặc đã từng sử dụng.
* Cấu trúc thư mục rõ ràng và có ý nghĩa nhằm đảm bảo dễ dàng quản lý và mở rộng.
* Nội dung file cần gọn gàng và mang ý nghĩa. Mỗi class, module, hoặc package chỉ nên thuộc một file.
* Ưu tiên mã dễ hiểu hơn là phải chú thích để người khác hiểu được mã.
* Chỉ chú thích khi cần thiết, và nếu có, hãy giải thích lý do (“why”) thay vì nội dung chi tiết (“what”).
* Áp dụng các nguyên tắc lập trình (DRY, SOLID, YAGNI, ...) tối đa khi có thể.
* Áp dụng các mẫu thiết kế (design patterns) khi cần thiết để cải thiện khả năng mở rộng và bảo trì.
* Tránh commit dở dang trên Git (sử dụng tùy chọn Amend khi commit lại). Mỗi commit cần có thông điệp ý nghĩa trong tên của nó.
* Tránh sử dụng các “magic numbers” (các số không rõ ý nghĩa), thay vào đó hãy sử dụng hằng số hoặc enum.
* Áp dụng các công cụ kiểm tra mã tự động (CI/CD), bao gồm kiểm tra unit, tích hợp và bảo mật, để đảm bảo mã luôn đạt tiêu chuẩn.
* Sử dụng chuẩn phiên bản (version control) nhất quán (ví dụ: Gitflow hoặc Trunk-based Development) để dễ quản lý phiên bản và phối hợp công việc.
* Luôn kiểm tra và loại bỏ mã không sử dụng (dead code) để giữ mã sạch và dễ đọc.

## Dart

### a) Quy tắc đặt tên

* Sử dụng camelCase cho tên biến và hàm. Ví dụ: userName, getUserData().
* Sử dụng Pascal Case cho tên lớp. Ví dụ: LoginPage, UserProfile.
* Tên tệp và tên thư mục nên sử dụng snake\_case và phản ánh nội dung của tệp. Ví dụ: login\_screen.dart, email\_validator.dart.

### b) Quy tắc code

* Tổ chức mã nguồn theo chức năng (feature-based) kết hợp với kiến trúc micro-frontend. Đặt các tệp liên quan đến một chức năng cụ thể trong cùng một thư mục.
* Các lớp nên có một trách nhiệm rõ ràng, không làm quá nhiều việc trong cùng một lớp.
* Mã nguồn nên được viết sao cho không có phần trùng lặp không cần thiết. Sử dụng các phương thức và hàm để tái sử dụng mã nguồn
* Giao diện (UI) và logic xử lý nghiệp vụ (business logic) nên được tách biệt rõ ràng. Thực hiện theo nguyên lý MVVM
* Sử dụng comment để giải thích các đoạn mã phức tạp hoặc không rõ ràng và để mô tả các hàm và lớp.
* Sử dụng Provider để quản lý trạng thái ứng dụng.
* Sử dụng pubspec.yaml để quản lý các gói phụ thuộc và đảm bảo rằng các gói được cập nhật thường xuyên.
* Giao diện (UI) và logic xử lý nghiệp vụ (business logic) nên được tách biệt rõ ràng. Thực hiện theo nguyên lý MVVM
* Sử dụng các cơ chế xử lý lỗi phù hợp như try-catch khi làm việc với các thao tác có thể phát sinh lỗi, ví dụ như các thao tác I/O, network request
* Tuân thủ các nguyên lý thiết kế giao diện Material Design để đảm bảo tính đồng nhất và dễ sử dụng cho người dùng
* Khi gọi api cần sử dụng lớp Dio Client đã được viết sẵn để được xử lý các vấn đề về timeout, token,...

### c) Quy tắc khi merge code

* Cần kiểm tra lỗi tự động bằng câu lệnh flutter analyze để xử lý lỗi và warning trước khi merge code
* Cần chạy lệnh dart format . để format lại code cho dễ đọc trước khi merge code
* Cần pull code mới nhất về trước khi merge code

## Java

### a) Quy tắc chung

* Ưu tiên code dễ hiểu hơn comment: Viết code tự giải thích. Comment chỉ nên dùng để giải thích lý do (why) chứ không phải nội dung chi tiết (what) của code.
* DRY (Don't Repeat Yourself): Tránh trùng lặp code. Tái sử dụng code thông qua các hàm, phương thức, hoặc thư viện dùng chung.
* SOLID Principles: Áp dụng các nguyên tắc thiết kế SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) để code dễ bảo trì và mở rộng.
* YAGNI (You Ain't Gonna Need It): Chỉ thêm chức năng khi thực sự cần thiết, tránh viết code cho những tính năng có thể sẽ không bao giờ được dùng.
* Sử dụng Design Patterns phù hợp: Áp dụng các mẫu thiết kế khi chúng giúp cải thiện cấu trúc, khả năng mở rộng và bảo trì của ứng dụng.
* Loại bỏ "Magic Numbers": Thay thế các số có ý nghĩa không rõ ràng bằng hằng số hoặc enum để tăng tính dễ đọc và dễ bảo trì.
* Loại bỏ Dead Code: Thường xuyên kiểm tra và loại bỏ code không còn sử dụng để giữ cho code sạch và gọn gàng.

### b) Quy tắc đặt tên

**Tên Package:**

* Sử dụng chữ thường và số (ví dụ: com.example.myproject). Không dùng dấu gạch dưới hay chữ hoa.
* Phản ánh cấu trúc và chức năng của ứng dụng.

**Tên Class và Interface:**

* Sử dụng UpperCamelCase (PascalCase) (ví dụ: UserService, OrderRepository).
* Thường là danh từ hoặc cụm danh từ (ví dụ: Character, ImmutableList). Interface cũng có thể là tính từ (ví dụ: Readable).

**Tên Method:**

* Sử dụng lowerCamelCase (ví dụ: getUserById, calculateTotal).
* Thường là động từ hoặc cụm động từ (ví dụ: sendMessage, stop).
* Đối với các phương thức truy cập thuộc tính, nên bắt đầu bằng get, set (ví dụ: getName(), setName(String name)).
* Đối với biến boolean hoặc phương thức trả về boolean, nên bắt đầu bằng is, has, can, should (ví dụ: isActive(), hasPermission()).

**Tên Hằng số (Constants):**

* Sử dụng UPPER\_SNAKE\_CASE (tất cả chữ hoa, các từ phân tách bằng dấu gạch dưới) (ví dụ: MAX\_RETRIES, DEFAULT\_TIMEOUT).
* Áp dụng cho các trường static final mà giá trị không thay đổi và không có tác dụng phụ.

**Tên Biến cục bộ (Local Variables) và Tham số:**

* Sử dụng lowerCamelCase (ví dụ: userName, orderId).
* Đặt tên có ý nghĩa, tránh các từ chung chung như data, thing, stuff.

**Tên Biến kiểu (Type Variables - Generics):**

* Sử dụng một chữ cái viết hoa, đôi khi theo sau bởi một chữ số (ví dụ: T, E, K, V, T2).

### c) Quy tắc code

**Cấu trúc file source:**

* Mỗi file Java phải tuân thủ thứ tự sau, phân cách bởi một dòng trống:

1. Thông tin bản quyền/Giấy phép (nếu có).

2. Khai báo package.

3. Các câu lệnh import (nhóm static import riêng, sau đó là non-static import, cách nhau bởi một dòng trống). Không sử dụng import \*.

4. Lớp cấp cao (top-level class).

* Mỗi lớp cấp cao (top-level class) phải nằm trong một file riêng biệt.
* Thứ tự thành viên trong lớp: Sắp xếp theo một thứ tự logic. Các phương thức có cùng tên (overloaded methods) và các hàm tạo (constructors) nên được nhóm lại với nhau.

**Định dạng:**

* Dấu ngoặc nhọn {}:
* Luôn sử dụng dấu ngoặc nhọn cho if, else, for, do, while, ngay cả khi thân rỗng hoặc chỉ có một câu lệnh.
* Kiểu K&R: Không xuống dòng trước dấu ngoặc mở {, xuống dòng sau dấu ngoặc mở, xuống dòng trước dấu ngoặc đóng }, xuống dòng sau dấu ngoặc đóng, trừ khi nó đi kèm với else hoặc dấu phẩy.

if (condition) {

doSomething();

} else {

doSomethingElse();

}

* Khối rỗng: void doNothing() {}
* Một câu lệnh/khai báo trên mỗi dòng:

int count; // Đúng

int x, y; // Sai

* Thụt lề: Mỗi dòng tiếp theo sau dòng đầu tiên bị ngắt được thụt lề 4 khoảng trắng.
* Khoảng trắng ngang: Sử dụng để tăng tính dễ đọc:
* Giữa từ khóa và dấu ngoặc mở (ví dụ: if ().
* Trước dấu ngoặc nhọn mở {.
* Hai bên của bất kỳ toán tử nhị phân/tam phân nào (+, -, &&, ||, ?:).
* Giữa kiểu và tên biến trong khai báo (ví dụ: String name).

**Khai báo biến:** Khai báo biến cục bộ gần nơi chúng được sử dụng nhất để giảm thiểu phạm vi.

**Mảng:** Tránh khai báo mảng kiểu C (ví dụ: String args[]). Luôn sử dụng String[] args.

**Switch:** Luôn bao gồm default case, ngay cả khi nó không chứa code.

**Annotations:** Đặt annotations trên dòng riêng biệt trước phần tử mà chúng chú thích.

@Override

@Autowired

public void someMethod() { /\* ... \*/ }

**Modifiers:** Sắp xếp theo thứ tự sau:

public protected private abstract default static final

transient volatile synchronized native strictfp

**Xử lý ngoại lệ:**

* Sử dụng try-catch một cách có trách nhiệm.
* Không "nuốt" ngoại lệ mà không có xử lý hoặc log phù hợp. Ném các ngoại lệ phù hợp với ngữ cảnh.

**Logging:**

* Sử dụng một framework logging chuẩn (ví dụ: Logback/SLF4J).
* Sử dụng các cấp độ log thích hợp (DEBUG, INFO, WARN, ERROR).

### d) Quy tắc merge code

Để đảm bảo chất lượng và sự nhất quán trong quá trình tích hợp code:

* Chạy unit tests: Đảm bảo tất cả các unit test liên quan đã vượt qua trước khi merge.
* Code format: Chạy công cụ định dạng code (ví dụ: tích hợp sẵn trong IDE hoặc Maven/Gradle plugin như Spotless) để đảm bảo code tuân thủ chuẩn format trước khi commit và merge.
* Pull code mới nhất: Luôn kéo code mới nhất từ nhánh đích (ví dụ: develop hoặc main) về nhánh của bạn và giải quyết xung đột trước khi tạo Pull Request.
* Thông điệp commit rõ ràng:
* Mỗi commit nên đại diện cho một thay đổi logic duy nhất.
* Thông điệp commit phải ngắn gọn, súc tích và có ý nghĩa, giải thích mục đích của commit.
* Tránh commit dở dang; sử dụng git commit --amend khi cần sửa commit trước đó.