**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**RESTful API với HTTP và Dio**

**Sinh viên thực hiện:**

**01. Nguyễn Quốc Khánh Lớp: 22KTMT2** **MSSV:**  106220255

**02. Hà Tiến Đạt**  **Lớp: 22KTMT2**  **MSSV:**  106220248

**Người hướng dẫn:**

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | HÀ TIẾN ĐẠT | Xử lý authentication, interceptors với Dio  Error handling và retry mechanisms | 50% |
| 02 | NGUYỄN QUỐC KHÁNH | Tạo ứng dụng lấy danh sách bài viết từ JSONPlaceholder API  So sánh http vs dio về tính năng, performance | 50% |

**MỤC LỤC**

[**I. Giới thiệu chung về RESTful API và JSONPlaceholder API** 5](#_Toc212570896)

[**1.Giới thiệu về API** 5](#_Toc212570897)

[**2. Giới thiệu chung về RESTful API** 5](#_Toc212570898)

[**a. RESTful API là gì?** 5](#_Toc212570899)

[**b. Các thành phần của RESTFul API** 6](#_Toc212570900)

[**c. Cách hoạt động của RESTFul API** 6](#_Toc212570901)

[**d. Các nguyên tắc của RESTFul API** 7](#_Toc212570902)

[**3. JSONPlaceholder API – API mẫu để thực hành** 7](#_Toc212570903)

[**II. So sánh HTTP và DIO** 8](#_Toc212570904)

[**1. Sử dụng HTTP package trong Flutter** 8](#_Toc212570905)

[**a. Giới thiệu** 8](#_Toc212570906)

[**b. Các bước thực hiện theo tài liệu** 8](#_Toc212570907)

[**c. Phân tích ưu và nhược điểm khi sử dụng http trong Flutter** 11](#_Toc212570908)

[**2. Sử dụng Dio package trong Flutter** 11](#_Toc212570909)

[**a. Giới thiệu** 11](#_Toc212570910)

[**b. Các bước thực hiện theo tài liệu** 11](#_Toc212570911)

[**c. Phân tích ưu và nhược điểm** 13](#_Toc212570912)

[**3. Kết quả chạy được** 13](#_Toc212570913)

[**4. So sánh http vs dio về tính năng, performance** 15](#_Toc212570914)

[**a. So sánh theo tính năng** 15](#_Toc212570915)

[**b. So sánh hiệu năng (Performance)** 16](#_Toc212570916)

[**III. Xử lý authentication, intercreptors với Dio** 16](#_Toc212570917)

[**1. Authentication** 16](#_Toc212570918)

[**2. Intercreptors** 17](#_Toc212570919)

[**3. Mối quan hệ giữa Authentication và Interceptors** 18](#_Toc212570920)

[**4. Quy trình hoạt động tổng quát** 19](#_Toc212570921)

[**IV. Error handling và retry mechnisms** 20](#_Toc212570922)

[**1. Xử lý lỗi (Error Handling)** 20](#_Toc212570923)

[**2. Cơ chế thử lại (Retry Mechanisms)** 21](#_Toc212570924)

[**3. Ý nghĩa và lợi ích** 22](#_Toc212570925)

[**TÀI LIỆU THAM KHẢO** 23](#_Toc212570926)

# **I. Giới thiệu chung về RESTful API và JSONPlaceholder API**

## **1.Giới thiệu về API**

API (Application Programming Interface) là giao diện lập trình ứng dụng, cho phép các phần mềm hoặc hệ thống khác nhau giao tiếp, kết nối và trao đổi dữ liệu với nhau thông qua một tập hợp các định nghĩa và giao thức.

API đóng vai trò trung gian giữa client (người dùng, ứng dụng, website) và server (máy chủ, cơ sở dữ liệu). Khi client gửi yêu cầu đến API, API sẽ xử lý và trả lại phản hồi từ server về cho client.

API hỗ trợ nhiều loại thao tác như truy xuất (GET), tạo mới (POST), cập nhật (PUT/PATCH) và xóa dữ liệu (DELETE). Dữ liệu trao đổi giữa hai bên thường được định dạng dưới dạng JSON hoặc XML nhằm đảm bảo tính tương thích và dễ xử lý.

Trong phát triển phần mềm hiện nay, API giữ vai trò cốt lõi trong việc kết nối giữa các dịch vụ, tích hợp hệ thống, và phát triển ứng dụng web, mobile. Việc sử dụng API giúp tiết kiệm thời gian phát triển, tăng tính mở rộng và tái sử dụng mã nguồn.

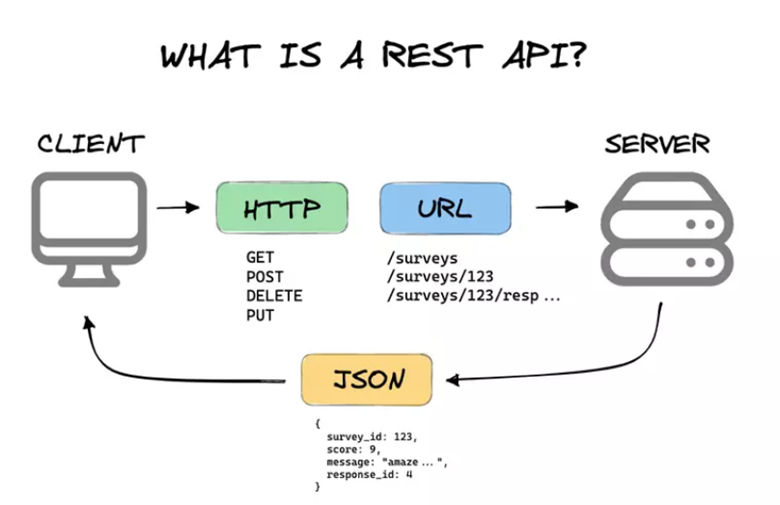
## **2. Giới thiệu chung về RESTful API**

### **a. RESTful API là gì?**

RESTful API (Representational State Transfer API) là một phong cách kiến trúc và phương pháp tiếp cận được sử dụng rộng rãi trong việc xây dựng các dịch vụ web. RESTful API dựa trên giao thức HTTP để thực hiện các hoạt động CRUD (Create, Read, Update, Delete) trên các tài nguyên. Sự phổ biến của RESTful API đến từ tính đơn giản, dễ hiểu và khả năng mở rộng dễ dàng.

RESTFul API (còn gọi là REST API) là tiêu chuẩn được dùng để thiết kế API cho trang web và ứng dụng và được sử dụng để truyền tải, trao đổi dữ liệu trong tài nguyên hệ thống với nhiều định dạng như tệp văn bản, âm thanh, ảnh, video, dữ liệu di động,... giữa máy chủ và máy khách thông qua giao thức HTTP.

RESTFul API sử dụng định dạng dữ liệu XML, JSON để trao đổi thông tin giữa máy chủ và máy khách đồng thời cho phép các website, ứng dụng khác nhau có thể truy cập và sử dụng hệ thống tài nguyên dễ dàng.



### **b. Các thành phần của RESTFul API**

RESTFul API gồm 2 thành phần là API và REST, cụ thể như sau:

* API (Application Programming Interface) là cơ chế cho phép 2 thành phần phần mềm giao tiếp với nhau bằng một tập hợp các định nghĩa và giao thức. API sẽ trả dữ liệu về ứng dụng cho người dùng những kiểu dữ liệu như XML hoặc JSON.
* REST (Representational State Transfer) là dạng chuyển đổi cấu trúc dữ liệu, một kiến trúc viết API và thường sử dụng giao thức HTTP đơn giản để trao đổi giữa các máy. REST sử dụng URL để xử lý nhanh chóng dữ liệu, thông tin người dùng khi REST gửi yêu cầu HTTP (GET, POST, DELETE, PUT,...).

RESTFul APIlà tiêu chuẩn thiết kế các API cho các ứng dụng website để quản lý tài nguyên, trong đó RESTFul là một kiểu thiết kế API thông dụng, phổ biến được áp dụng trong nhiều ứng dụng như Mobile, Website,... để giao tiếp với nhau.

RESTFul không có quy tắc logic code ứng dụng và không bị giới hạn ngôn ngữ lập trình nên bất kỳ ngôn ngữ hay framework nào cũng sử dụng thiết kế RESTFul API dễ dàng.

### **c. Cách hoạt động của RESTFul API**

RESTFul API hoạt động chủ yếu dựa vào phương thức HTTP với các hoạt động cơ bản như:

* **GET (Read):** Truy xuất và trả về tài nguyên từ máy chủ.
* **POST (Create):** Gửi yêu cầu tạo mới một tài nguyên.
* **PUT (Update):** Cập nhật hoặc thay thế toàn bộ thông tin tài nguyên.
* **DELETE (Delete):** Xóa tài nguyên khỏi máy chủ.

Hoạt động của RESTFul API còn được gọi là CRUD có nghĩa là Create, Read, Update, Delete (Tạo, Đọc, Sửa, Xóa). Tuy nhiên, đại đa số lập trình viên viết RESTFul API muốn nhanh gọn và tiện lợi, dễ thực hiện thường chọn JSON hoặc XML để làm format chính.

### **d. Các nguyên tắc của RESTFul API**

Những nguyên tắc của RESTFul API giúp phát triển API hiệu quả hơn, cụ thể như sau:

* Tương tác giữa Server và Client:Các tài nguyên (resource) của server được người dùng truy cập và sử dụng bởi client thông qua các yêu cầu HTTP.
* Các phương thức HTTP: Phương thức HTTP như GET, POST, PUT và DELETE là tiêu chuẩn để người dùng truy cập, quản lý tài nguyên của máy chủ, được dùng để lấy thông tin, tạo mới, cập nhật hoặc xóa dữ liệu.
* Resource: Đây là đối tượng được quản lý trên máy chủ với mỗi URI đại diện cho 1 tài nguyên, ví dụ như blog, sản phẩm, người dùng,...
* URI - Uniform Resource Identifier: URI dùng để định danh, địa chỉ tài nguyên trên máy chủ hoặc thực hiện các yêu cầu thao tác trên tài nguyên đó.
* Representation: Các tài nguyên trong RESTFul API đều có định dạng cụ thể như HTML, XML, JSON và YAML. Các định dạng này được dùng để thông tin được truyền đi dễ dàng giữa client và server.
* Metadata: Được dùng để cung cấp những thông tin như định dạng, ngôn ngữ, thời gian tạo hoặc sửa đổi của tài nguyên giúp người dùng quản lý tốt dữ liệu trên máy chủ.

# **3. JSONPlaceholder API – API mẫu để thực hành**

JSONPlaceholder là một dịch vụ web giả lập miễn phí được sử dụng rộng rãi để thực hành REST API.

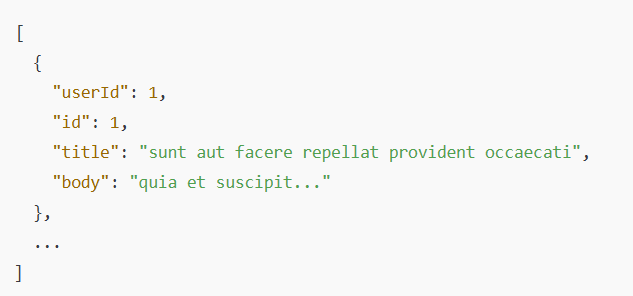
Một số endpoint phổ biến:

* /posts → Lấy danh sách bài viết
* /posts/{id} → Lấy chi tiết bài viết
* /users → Lấy danh sách người dùng

**Ví dụ:** [jsonplaceholder.typicode.com/posts](https://jsonplaceholder.typicode.com/posts) và [Fake Store API](https://fakestoreapi.com/)



**Kết quả trả về dạng JSON:**



# **II. So sánh HTTP và DIO**

## **1. Sử dụng HTTP package trong Flutter**

### **a. Giới thiệu**

Giao tiếp mạng (networking) là một phần thiết yếu trong hầu hết các ứng dụng hiện đại — ví dụ như lấy dữ liệu từ server, gửi dữ liệu, v.v. Trang tài liệu của Flutter nhấn mạnh rằng việc fetch dữ liệu từ Internet là rất phổ biến và Flutter/Dart cung cấp các công cụ như gói http để thực hiện điều đó.

Gói http là thư viện đơn giản và hiệu quả để thực hiện các request HTTP trong ứng dụng Flutter.

### **b. Các bước thực hiện theo tài liệu**

Theo hướng dẫn trên trang “Fetch data from the internet” của Flutter, khi sử dụng http package ta thường thực hiện theo 4 bước chính:

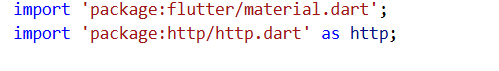
**Bước 1: Thêm gói và import**

* Bạn thêm http vào pubspec.yaml

A computer screen shot of a computer program

AI-generated content may be incorrect.

* Rồi import trong file:



**Bước 2: Thực hiện HTTP GET request mạng.**

Gửi yêu cầu GET đến API:



**http.get()** trả về một **Future<Response>** nên phải **await.**

**Bước 3.Thực hiện HTTP POST**

Sử dụng API POST để tạo một đối tượng mới trong database, ví dụ: tạo một tệp tin trong thư mục cụ thể hoặc một hàng trong bảng cơ sở dữ liệu. Nói một cách nghiêm túc về REST, các phương thức POST được sử dụng để tạo đối tượng mới vào cơ sở dữ liệu.

A screen shot of a computer code

AI-generated content may be incorrect.

**Bước 4: Thực hiện HTTP PUT**

Sử dụng API PUT chủ yếu để cập nhật đối tượng hiện có (nếu dữ liệu không tồn tại, thì API có thể quyết định tạo đối tượng mới hay không).

A screen shot of a computer code

AI-generated content may be incorrect.

**Bước 5: Thực hiện HTTP DELETE**

API DELETE được sử dụng để xóa đối tượng (được xác định bởi URI yêu cầu).

A screen shot of a computer code

AI-generated content may be incorrect.

### **c. Phân tích ưu và nhược điểm khi sử dụng http trong Flutter**

**\***Ưu điểm**:**

* Dễ tích hợp: chỉ cần thêm gói và import.
* Đơn giản rõ ràng cho các thao tác GET/POST cơ bản.
* Hoạt động tốt khi dữ liệu và yêu cầu mạng đơn giản.

**\***Nhược điểm**:**

* Khi ứng dụng phức tạp (nhiều request, cần logging, retry, cancel) sẽ thiếu các tính năng nâng cao.
* Cần tự xử lý nhiều logic như timeout, lỗi, parse JSON phức tạp.
* Không có interceptor sẵn để chặn request/response hoặc thêm header chung.

## **2. Sử dụng Dio package trong Flutter**

### **a. Giới thiệu**

Dio là thư viện HTTP client mạnh mẽ được Flutter community sử dụng rộng rãi. So với http, Dio hỗ trợ nhiều tính năng nâng cao như:

* Interceptor: theo dõi, thay đổi request/response.
* Timeout & Cancel request: dễ dàng quản lý lỗi mạng.
* Logging & Error Handling: tích hợp sẵn.
* FormData, Upload file, Token authentication: tiện lợi cho ứng dụng lớn.

Nhờ vậy, Dio thường được chọn trong các ứng dụng Flutter cần quản lý mạng chuyên nghiệp, có nhiều request phức tạp.

### **b. Các bước thực hiện theo tài liệu**

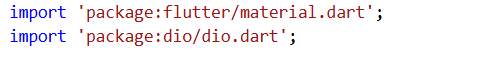
**Bước 1: Thêm gói Dio**

* Thêm vào pubspec.yaml:

A computer screen shot of a computer program

AI-generated content may be incorrect.

* Rồi import trong file:



**Bước 2: Dio GET**

Phương thức GET trong Dio được sử dụng để lấy dữ liệu từ server. Khi gọi API, phương thức này trả về một Future<Response> chứa dữ liệu phản hồi (response data), giúp ứng dụng hiển thị thông tin nhận được từ server.

A computer code with text

AI-generated content may be incorrect.

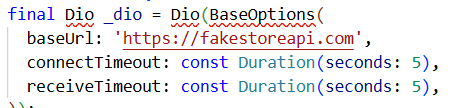
**Bước 3: Dio POST**

Được sử dụng để cập nhật dữ liệu hiện có trên server. Phương thức put() được gọi đến endpoint cụ thể (ví dụ: /products/1) và gửi dữ liệu cập nhật trong phần data.

Server sẽ thay thế nội dung cũ bằng dữ liệu mới, hoặc tùy cấu hình có thể tạo đối tượng mới nếu chưa tồn tại.



**Bước 4: Cấu hình chung**



### **c. Phân tích ưu và nhược điểm**

\*Ưu điểm:

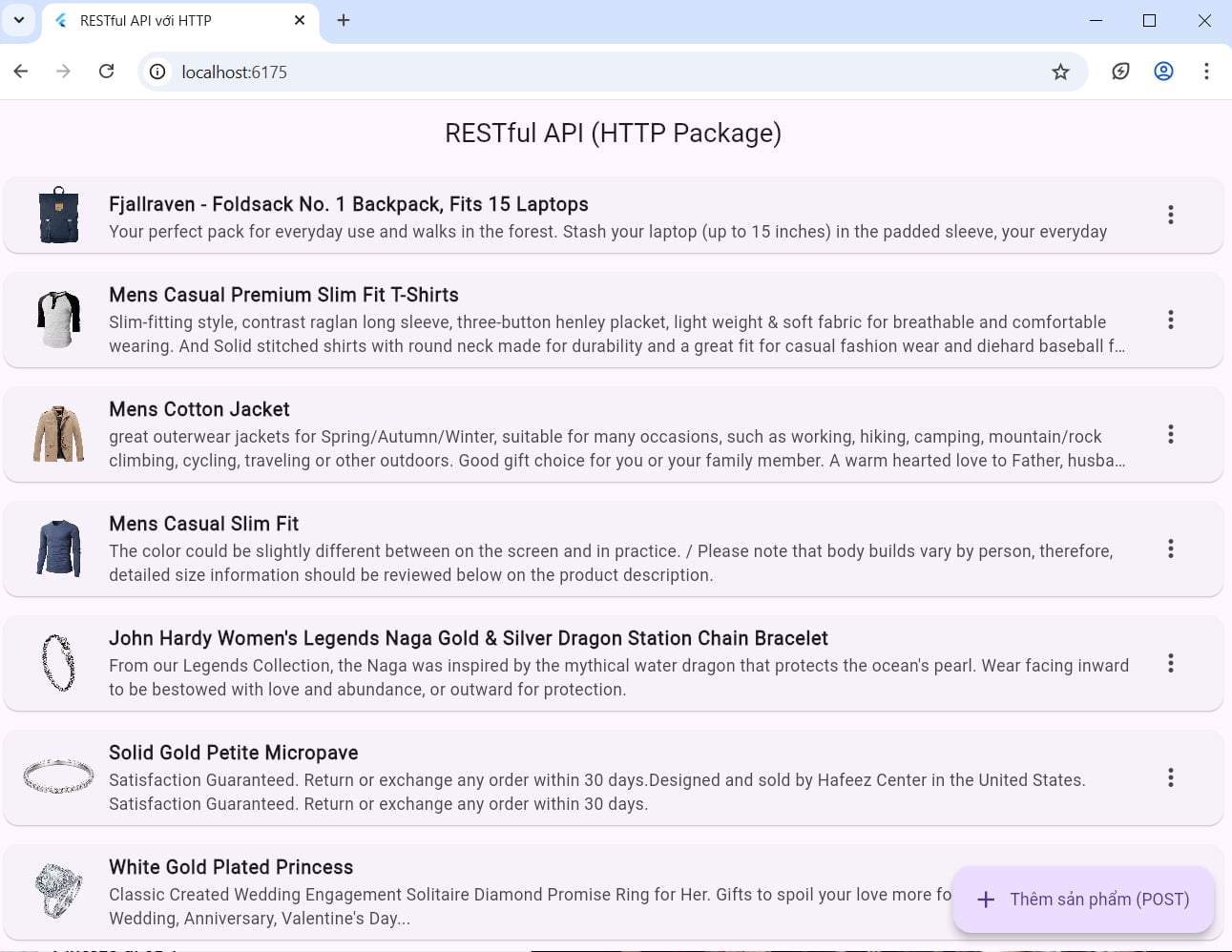
* Có interceptor – dễ dàng thêm token, log, header.
* Tự động decode JSON, không cần jsonDecode().
* Hỗ trợ timeout, cancel request, upload file, form data.
* Tùy chỉnh linh hoạt cho ứng dụng lớn.

\*Nhược điểm:

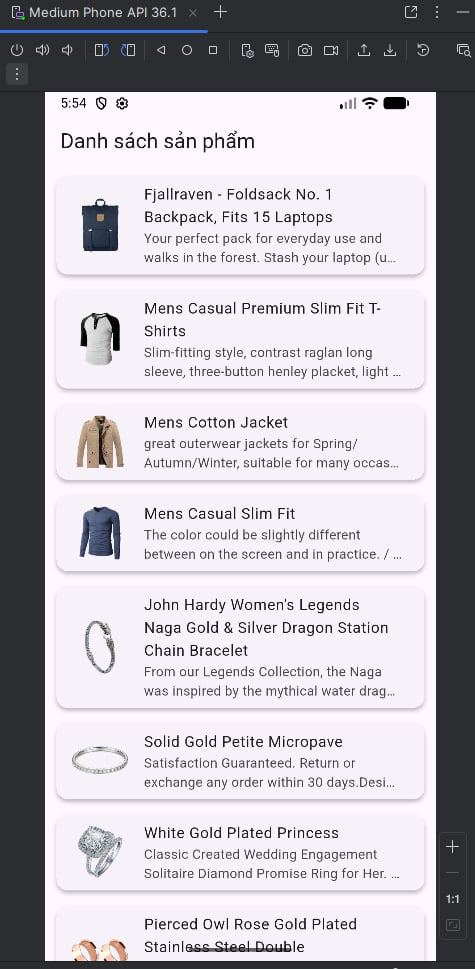
* Kích thước gói lớn hơn http.
* Cấu hình phức tạp hơn, không cần thiết cho dự án nhỏ.

## **3. Kết quả chạy được**

* **Kết quả chạy của HTTP**



* **Kết quả chạy của Dio**



## **4. So sánh http vs dio về tính năng, performance**

### **a. So sánh theo tính năng**

**HTTP:**

* Giao diện đơn giản, dễ sử dụng.
* Hỗ trợ các phép HTTP cơ bản như GET, POST, PUT, DELETE.
* Nhẹ, ít phụ thuộc, phù hợp cho các dự án nhỏ.

**Dio:**

* Cung cấp API toàn diện hơn: hỗ trợ interceptor, cấu hình toàn cục (headers, baseURL)
* Hỗ trợ upload/download file, FormData, cancel request, timeout, retry mechanism.
* Hỗ trợ JSON tự động serialisation/deserialisation, xử lý request/response phức tạp.

**Hạn chế:**

* HTTP: Không có interceptor, không cấu hình global, không hủy request được, khả năng tùy chỉnh thấp.
* Dio: Có đường cong học tập cao hơn, nhiều tính năng có thể “quá” nếu chỉ cần các request đơn giản, kích thước bundle lớn hơn.

### **b. So sánh hiệu năng (Performance)**

**HTTP:**

* Setup nhanh hơn do đơn giản.
* Sử dụng bộ nhớ và CPU thấp hơn cho các request cơ bản.

**Dio:**

* Có overhead cao hơn một chút do nhiều tính năng hơn.
* Tuy nhiên, trong các kịch bản phức tạp (upload/download file lớn, nhiều request đồng thời, xử lý dữ liệu lớn) thì Dio hoạt động tốt hơn.

# **III. Xử lý authentication, intercreptors với Dio**

## **1. Authentication**

Authentication là quá trình xác minh danh tính của người dùng nhằm đảm bảo rằng chỉ những người có quyền hợp lệ mới được phép truy cập vào hệ thống. Thông thường, quá trình này được thực hiện thông qua việc đăng nhập bằng tên người dùng (username) và mật khẩu (password).

A screen shot of a computer code

AI-generated content may be incorrect.

Khi người dùng đăng nhập thành công, máy chủ (server) sẽ trả về các thông tin xác thực cần thiết để ứng dụng duy trì phiên làm việc, bao gồm:

* Access Token: là mã truy cập ngắn hạn, được gửi kèm trong các yêu cầu (request) đến server để chứng minh rằng người dùng đã được xác thực. Token này thường có thời hạn ngắn nhằm giảm rủi ro bảo mật nếu bị lộ.
* Refresh Token: là mã làm mới Access Token khi nó hết hạn. Thay vì bắt người dùng đăng nhập lại, ứng dụng sẽ gửi Refresh Token đến server để yêu cầu cấp một Access Token mới, giúp duy trì trạng thái đăng nhập liên tục.
* Expired Time: là thời điểm hoặc khoảng thời gian mà Access Token (hoặc đôi khi cả Refresh Token) còn hiệu lực. Sau khi hết hạn, Access Token sẽ không thể sử dụng để truy cập API nữa, và ứng dụng phải thực hiện quá trình làm mới token hoặc yêu cầu người dùng đăng nhập lại.

Nhờ cơ chế sử dụng kết hợp Access Token và Refresh Token, hệ thống có thể vừa đảm bảo bảo mật dữ liệu, vừa nâng cao trải nghiệm người dùng bằng cách tránh phải xác thực thủ công nhiều lần.

## **2. Intercreptors**

Interceptors là cơ chế trung gian trong thư viện Dio (Flutter), cho phép chặn và can thiệp vào quá trình xử lý các yêu cầu (request), phản hồi (response) hoặc lỗi (error) trước khi chúng được gửi đi hoặc trả về cho ứng dụng. Nhờ đó, Interceptors giúp lập trình viên có thể dễ dàng quản lý các tác vụ lặp lại hoặc xử lý logic chung cho toàn bộ hệ thống mạng.

Cụ thể, Interceptors thường được sử dụng để thực hiện các chức năng sau:

* Thêm hoặc chỉnh sửa thông tin trong yêu cầu (Request Interceptor):  
  Cho phép tự động gắn thêm Access Token vào phần header của mọi request nhằm đảm bảo các yêu cầu gửi đến server đều được xác thực. Ngoài ra, lập trình viên có thể thêm các thông tin khác như Content-Type, Accept-Language, hoặc log thông tin yêu cầu phục vụ mục đích theo dõi và kiểm thử.

A screen shot of a computer code

AI-generated content may be incorrect.

* Xử lý phản hồi từ server (Response Interceptor):

Giúp kiểm tra, phân tích, hoặc chuẩn hóa dữ liệu phản hồi trước khi chuyển đến tầng xử lý logic của ứng dụng. Ví dụ, có thể lọc dữ liệu, xử lý định dạng JSON, hoặc ghi log kết quả phản hồi để dễ dàng theo dõi trạng thái hệ thống.

* Xử lý lỗi (Error Interceptor):

Khi xảy ra lỗi (như mất kết nối, lỗi 401 – Unauthorized, hoặc lỗi máy chủ 500), Interceptor cho phép ứng dụng can thiệp trước khi lỗi được hiển thị ra giao diện. Trường hợp lỗi 401 do Access Token hết hạn, Interceptor có thể tự động gọi Refresh Token để lấy token mới và thử gửi lại yêu cầu ban đầu, giúp người dùng không bị gián đoạn quá trình sử dụng.

A screen shot of a computer code

AI-generated content may be incorrect.

## **3. Mối quan hệ giữa Authentication và Interceptors**

Hai cơ chế Authentication và Interceptors trong Dio có mối quan hệ chặt chẽ và bổ trợ cho nhau trong quá trình quản lý xác thực người dùng.

Authentication đảm nhiệm việc xác minh danh tính và cấp phát token sau khi người dùng đăng nhập, trong khi Interceptors chịu trách nhiệm quản lý, duy trì và sử dụng token đó trong toàn bộ vòng đời của phiên làm việc.

Mối quan hệ này có thể tóm gọn như sau:

* Authentication cung cấp dữ liệu xác thực (Access Token, Refresh Token, Expired Time).
* Interceptors tự động gắn Access Token vào mọi request, phát hiện khi token hết hạn, và kích hoạt quá trình làm mới token bằng Refresh Token.
* Khi quá trình làm mới thành công, Interceptor sẽ cập nhật token mới và thực hiện lại request ban đầu, giúp trải nghiệm người dùng liền mạch, không bị yêu cầu đăng nhập lại.

Nhờ sự phối hợp này, ứng dụng vừa đảm bảo tính bảo mật cao, vừa đơn giản hóa xử lý xác thực trong mã nguồn, tránh việc lập trình viên phải thao tác thủ công với từng API riêng lẻ.

## **4. Quy trình hoạt động tổng quát**

Quy trình xử lý authentication kết hợp interceptors trong Dio có thể mô tả qua các bước sau:

1. Người dùng đăng nhập bằng tên đăng nhập và mật khẩu tại giao diện đăng nhập.
2. Server kiểm tra thông tin đăng nhập và trả về Access Token nếu hợp lệ.
3. Access Token được lưu tạm thời trong lớp DioClient để sử dụng cho các yêu cầu tiếp theo.
4. Mọi request gửi đi đều được Interceptor tự động gắn Access Token vào phần header để xác thực với server.
5. Khi xảy ra lỗi mạng hoặc hết thời gian chờ, Interceptor sẽ tự động thử lại request sau một khoảng thời gian ngắn.
6. Sau khi đăng nhập thành công, người dùng có thể truy cập dữ liệu (ví dụ: danh sách sản phẩm) thông qua các request đã được gắn token hợp lệ.

Quy trình này giúp hệ thống:

* Duy trì trạng thái xác thực ổn định, không bị ngắt quãng.
* Giảm thiểu rủi ro bảo mật, vì token có thời hạn ngắn.
* Tối ưu hiệu suất phát triển, do chỉ cần cấu hình Interceptors một lần cho toàn bộ ứng dụng.

# **IV. Error handling và retry mechnisms**

Trong quá trình giao tiếp giữa ứng dụng và máy chủ thông qua API, việc xảy ra lỗi là điều không thể tránh khỏi. Các lỗi có thể đến từ kết nối mạng, xác thực không hợp lệ, lỗi phía máy chủ, hoặc vấn đề dữ liệu.

Do đó, việc xây dựng cơ chế xử lý lỗi (Error Handling) và thử lại yêu cầu (Retry Mechanism) là rất quan trọng nhằm đảm bảo ứng dụng hoạt động ổn định và mang lại trải nghiệm tốt cho người dùng.

## **1. Xử lý lỗi (Error Handling)**

Dio cung cấp một cơ chế mạnh mẽ để phát hiện và xử lý lỗi thông qua Error Interceptor hoặc khối try-catch. Khi một request thất bại, Dio sẽ trả về đối tượng DioError, cho phép lập trình viên kiểm tra chi tiết nguyên nhân lỗi thông qua các thuộc tính như:

* error.type: xác định loại lỗi (mất kết nối, timeout, lỗi phản hồi, lỗi hủy request, v.v.).
* error.response: chứa thông tin phản hồi từ server (status code, message, body).

Một số lỗi phổ biến thường gặp:

| Loại lỗi | Mô tả | Hướng xử lý đề xuất |
| --- | --- | --- |
| DioErrorType.connectionTimeout | Kết nối quá thời gian | Thông báo người dùng kiểm tra mạng, thử lại |
| DioErrorType.sendTimeout | Quá thời gian khi gửi dữ liệu lên server | Kiểm tra tốc độ mạng, thử gửi lại request |
| DioErrorType.receiveTimeout | Server phản hồi quá chậm | Hiển thị thông báo “Server quá tải” |
| DioErrorType.badResponse | Server trả phản hồi lỗi (mã 4xx hoặc 5xx) | Tuỳ từng mã lỗi:  • **401:** Access Token hết hạn → gọi Refresh Token. • **403:** Không đủ quyền truy cập → thông báo “Access Denied.” • **404:** Không tìm thấy tài nguyên. • **500:** Lỗi máy chủ → báo người dùng thử lại. |
| DioErrorType.cancel | Request bị hủy thủ công trước khi hoàn thành | Bỏ qua hoặc ghi log (nếu cần debug). |
| DioErrorType.connectionError | Không thể kết nối mạng, lỗi DNS, server unreachable. | Hiển thị thông báo *“Không có kết nối mạng, vui lòng kiểm tra Internet.”* |
| DioErrorType.unknown | Lỗi không xác định | Ghi log để debug, hiển thị thông báo chung |

A screen shot of a computer code

AI-generated content may be incorrect.

Việc xử lý có thể được thực hiện ngay trong Interceptor, giúp toàn bộ request đều được bảo vệ và quản lý lỗi thống nhất, thay vì xử lý rời rạc tại từng API call.

## **2. Cơ chế thử lại (Retry Mechanisms)**

Sau khi phát hiện lỗi, đặc biệt là các lỗi mạng tạm thời (*network failure*) hoặc lỗi hết thời gian chờ (*timeout*), ứng dụng có thể tự động thử gửi lại request thay vì báo lỗi ngay cho người dùng.

Cơ chế này giúp hệ thống trở nên ổn định và thân thiện hơn, đặc biệt trong các trường hợp mạng yếu hoặc không ổn định.

Trong ứng dụng hiện tại, cơ chế retry được triển khai trực tiếp trong Interceptor của lớp DioClient, hoạt động theo quy trình sau:

1. Khi request gặp lỗi mạng hoặc timeout, Interceptor của DioClient sẽ bắt được lỗi dưới dạng DioException.
2. Hệ thống tạm dừng 2 giây trước khi thực hiện gửi lại request để tránh việc gửi liên tục.
3. Sau thời gian chờ, Interceptor gửi lại đúng request cũ, giữ nguyên toàn bộ thông tin như URL, phương thức (method), headers, body và query parameters.
4. Nếu request gửi lại thành công, Interceptor sẽ trả kết quả phản hồi (response) về luồng xử lý chính thông qua handler.resolve().
5. Nếu request gửi lại vẫn thất bại, lỗi mới sẽ được chuyển tiếp ra ngoài thông qua handler.next(), để các tầng khác như AuthService xử lý và hiển thị thông báo lỗi cho người dùng.

A screen shot of a computer code

AI-generated content may be incorrect.

## **3. Ý nghĩa và lợi ích**

* Tăng độ tin cậy của ứng dụng, đảm bảo yêu cầu quan trọng không thất bại do lỗi tạm thời.
* Giảm thao tác người dùng, do ứng dụng tự động khôi phục trạng thái.
* Chuẩn hóa quản lý lỗi nhờ sử dụng Interceptor tập trung.

# **TÀI LIỆU THAM KHẢO**

* RESTFul API : <https://fptcloud.com/restful-api-la-gi/>
* Flutter Docs : <https://docs.flutter.dev/cookbook/networking/fetch-data>
* Flutter - Fetching List of Data From API Through Dio :

<https://www.geeksforgeeks.org/flutter/flutter-fetching-list-of-data-from-api-through-dio/>

* Tìm hiểu về Interceptor trong Dio và triển khai cơ chế Authentication

<https://viblo.asia/p/dio-flutter-tim-hieu-ve-interceptor-trong-dio-va-trien-khai-co-che-authentication-018J2vzqJYK>

* Handling errors : <https://pub.dev/packages/dio#handling-errors>
* dio\_smart\_retry : <https://pub.dev/packages/dio_smart_retry>