**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. Lê Thành Duy Lớp: 22KTMT2**  **MSSV:**  106220250

**02. Nguyễn Ngọc Duy** **Lớp: 22KTMT2**  **MSSV:**  106220251

**Người hướng dẫn:**

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

**THUYẾT MINH**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**RESTful API với HTTP và Dio**

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | LÊ THÀNH DUY | Tìm hiểu về RESTful API, HTTP package, Dio package , code gọi api lấy bài viết, like, comment và giao diện | 50% |
| 02 | NGUYỄN NGỌC DUY | Tìm hiểu về xử lý authentication, interceptors với Dio, Error handling và retry mechanisms. Code demo chức năng authentication, đăng kí, đăng nhập và giao diện | 50% |

 Link code github:

**Mục lục**

…

Mục lục báo báo

[**I. Tổng quan RESTful API:** 1](#_Toc212499558)

[**II. HTTP và Dio trong Flutter:** 2](#_Toc212499559)

[**1. HTTP Package:** 3](#_Toc212499560)

[**2. Dio Package** 4](#_Toc212499561)

[**3. So sánh giữa HTTP và Dio**: 5](#_Toc212499562)

[**III. Cơ chế authentication bằng interceptors của thư viện dio:** 6](#_Toc212499563)

[**1. Khái niệm:** 6](#_Toc212499564)

[**2. Mục tiêu của cơ chế** 6](#_Toc212499565)

[**3. Quy trình hoạt động** 6](#_Toc212499566)

[**4. Cấu trúc các loại Interceptor liên quan:** 7](#_Toc212499567)

[**5. Ví dụ minh họa cơ chế hoạt động:** 7](#_Toc212499568)

[**6. Ưu điểm** 8](#_Toc212499569)

[**7. Nhược điểm và lưu ý** 8](#_Toc212499570)

[**8. Kết luận:** 9](#_Toc212499571)

[**IV. Error handling và retry mechanism** 9](#_Toc212499572)

[**1. Khái niệm** 9](#_Toc212499573)

[**2. Mục tiêu của cơ chế** 9](#_Toc212499574)

[**3. Các loại lỗi thường gặp trong Dio** 10](#_Toc212499575)

[**4. Quy trình xử lý lỗi trong Dio** 10](#_Toc212499576)

[**5. Cấu trúc Error Handling trong Dio** 10](#_Toc212499577)

[**6. Retry Mechanism – Cơ chế thử lại** 11](#_Toc212499578)

[**7. Ưu điểm của cơ chế Retry** 12](#_Toc212499579)

[**8. Lưu ý và hạn chế** 12](#_Toc212499580)

[**9. Kết luận** 12](#_Toc212499581)

[**V. Sản phẩm demo:** 13](#_Toc212499582)

[**TÀI LIỆU THAM KHẢO** 14](#_Toc212499583)

**Giới thiệu:**

Trong bối cảnh phát triển ứng dụng di động hiện nay, việc giao tiếp giữa ứng dụng khách (client) và máy chủ (server) thông qua API đóng vai trò quan trọng trong việc xử lý dữ liệu và cung cấp dịch vụ đến người dùng. RESTful API là một trong những kiến trúc phổ biến nhất nhờ tính đơn giản, linh hoạt và dễ triển khai.

Flutter là framework đa nền tảng mạnh mẽ của Google, hỗ trợ xây dựng giao diện và ứng dụng hiệu năng cao trên Android, iOS và web. Để giao tiếp với RESTful API trong Flutter, lập trình viên có thể sử dụng nhiều thư viện khác nhau. Trong đó, http package và dio package là hai lựa chọn phổ biến nhất:

* http: thư viện cơ bản, dễ dùng cho các yêu cầu GET/POST đơn giản.
* dio: thư viện nâng cao, hỗ trợ interceptors, timeout, retry, upload/download tracking, authentication,…

# **I. Tổng quan RESTful API:**

API (Application Programming Interface) là giao diện lập trình ứng dụng, cho phép các phần mềm hoặc hệ thống khác nhau có thể giao tiếp và trao đổi dữ liệu với nhau. API đóng vai trò trung gian giữa client (ứng dụng người dùng) và server (máy chủ xử lý dữ liệu), giúp việc tích hợp và mở rộng ứng dụng trở nên linh hoạt hơn.

REST (Representational State Transfer) là phong cách kiến trúc cho phép các hệ thống giao tiếp qua giao thức HTTP, dựa trên việc thao tác các tài nguyên (resource) được định danh bằng URI.  
RESTful API là API tuân theo các nguyên tắc REST, giúp client và server trao đổi dữ liệu linh hoạt và mở rộng tốt. Dữ liệu thường được truyền dưới dạng JSON do gọn nhẹ và dễ xử lý.

Về nguyên tắc, một RESTful API cần tuân thủ các đặc điểm cơ bản sau:

* Client–Server: Giao diện người dùng (client) và xử lý dữ liệu (server) được tách biệt rõ ràng, giúp hai bên có thể phát triển và bảo trì độc lập.
* Stateless: Mỗi request từ client phải chứa đủ thông tin để server hiểu và xử lý, server không lưu trạng thái của client giữa các lần yêu cầu.
* Cacheable: Phản hồi từ server có thể được lưu tạm (cache) để giảm tải và tăng hiệu năng.
* Uniform Interface: Giao diện truy cập tài nguyên thống nhất, được định danh bằng URL rõ ràng.
* Layered System: Cho phép phân tầng giữa các thành phần (gateway, proxy, load balancer), giúp hệ thống dễ mở rộng và linh hoạt hơn.

Trong RESTful API, tài nguyên (resource) là đối tượng được quản lý như bài viết, người dùng hoặc sản phẩm. Mỗi tài nguyên được xác định duy nhất bằng URI (Uniform Resource Identifier). Ví dụ:

https://jsonplaceholder.typicode.com/posts

Địa chỉ này mô tả tài nguyên “posts” trong API JSONPlaceholder, còn hành động (lấy, tạo, xóa, cập nhật) được xác định bằng phương thức HTTP tương ứng.

|  |  |  |
| --- | --- | --- |
| Phương thức | Chức năng | Ví dụ Endpoint |
| GET | Lấy dữ liệu | /posts hoặc /posts/1 |
| POST | Tạo mới | /posts |
| PUT/PATCH | Cập nhật | /posts/1 |
| DELETE | Xóa | /posts/1 |

Cách tiếp cận này giúp API rõ ràng, dễ hiểu và nhất quán, vì mỗi hành động đều được ánh xạ trực tiếp với phương thức HTTP quen thuộc.

RESTful API không ràng buộc định dạng dữ liệu, nhưng JSON được sử dụng phổ biến nhất nhờ ưu điểm: tương thích tốt với web và mobile, kích thước nhỏ, tốc độ truyền tải nhanh và dễ phân tích trên client. Ví dụ phản hồi từ API có thể ở dạng:

{

"userId": 1,

"id": 1,

"title": "Post title",

"body": "Content of the post"

}

**Bảng 1. Ưu điểm và nhược điểm của RESTful API**

|  |  |
| --- | --- |
| Ưu điểm | Nhược điểm |
| Dễ sử dụng, dễ tích hợp trên nhiều nền tảng nhờ giao thức HTTP. | Không hỗ trợ giao tiếp thời gian thực (real-time). |
| Cấu trúc rõ ràng, dễ hiểu, tuân theo các phương thức HTTP (GET, POST, PUT, DELETE). | Cần nhiều request khi xử lý dữ liệu phức tạp. |
| Hoạt động tốt với mô hình microservices, dễ mở rộng và bảo trì. | Stateless nên mỗi request đều cần xác thực lại. |
| Hỗ trợ cache, tăng hiệu năng và giảm tải cho server. | Nếu cấu hình cache sai có thể gây lỗi dữ liệu. |
| Dữ liệu thường ở dạng JSON – gọn nhẹ, dễ xử lý. | Bảo mật phải tự cấu hình, không tích hợp sẵn. |

# **II. HTTP và Dio trong Flutter:**

Trong phát triển ứng dụng Flutter, giao tiếp với backend thông qua RESTful API là yêu cầu quan trọng. Để thực hiện các yêu cầu HTTP, hai thư viện phổ biến được sử dụng là HTTP package và Dio package.

## **1. HTTP Package:**

HTTP là thư viện cơ bản và phổ biến nhất được Flutter cung cấp để thực hiện các yêu cầu mạng thông qua giao thức HTTP . Thư viện này giúp ứng dụng có thể gửi và nhận dữ liệu từ server, thường dùng để giao tiếp với RESTful API. Do tính đơn giản, HTTP package phù hợp cho các dự án nhỏ hoặc các tác vụ mạng không quá phức tạp.

Cấu trúc hoạt động của HTTP package khá trực tiếp. Mỗi khi ứng dụng cần lấy hoặc gửi dữ liệu, tạo một yêu cầu HTTP (như GET, POST, PUT, DELETE), gửi đến địa chỉ API, và nhận lại phản hồi dưới dạng JSON hoặc text. Dữ liệu nhận về thường được giải mã thủ công (parse JSON) để chuyển đổi thành đối tượng trong Dart.

Các bước cơ bản:

1. Gửi yêu cầu HTTP đến server.
2. Server xử lý và trả về phản hồi.
3. Ứng dụng nhận phản hồi, giải mã dữ liệu, và hiển thị cho người dùng.

Quá trình này không có nhiều lớp trung gian, giúp việc triển khai nhanh và dễ hiểu, tuy nhiên hạn chế ở khả năng mở rộng, xử lý lỗi nâng cao, và quản lý cấu hình phức tạp.

Ví dụ sau minh họa cách sử dụng HTTP package để lấy dữ liệu từ API JSONPlaceholder:

A computer screen shot of a program

AI-generated content may be incorrect.

Trong ví dụ trên:

* http.get() được dùng để gửi yêu cầu GET đến API.
* Kết quả trả về ở dạng JSON và được chuyển đổi thủ công bằng json.decode.
* statusCode giúp kiểm tra trạng thái phản hồi từ server.

Đặc điểm nổi bật

* Cấu trúc đơn giản, dễ sử dụng cho người mới bắt đầu.
* Không có hỗ trợ sẵn cho interceptor, cache hay retry.
* Phù hợp cho các ứng dụng nhỏ hoặc các request đơn lẻ.
* Đòi hỏi lập trình viên tự xử lý các tác vụ nâng cao như timeout, parse JSON, hoặc xử lý lỗi.

## **2. Dio Package**

Dio là một thư viện mạng mạnh mẽ dành cho Flutter, được phát triển như một giải pháp nâng cao so với HTTP package. Dio hỗ trợ đầy đủ các tính năng hiện đại như Interceptor, Request/Response Transformer, Timeout linh hoạt, Upload/Download có theo dõi tiến trình, và quản lý lỗi thông minh. Nhờ đó, Dio phù hợp với các ứng dụng có quy mô lớn, yêu cầu hiệu năng cao và kiểm soát chi tiết luồng truyền dữ liệu.

Cấu trúc hoạt động

Cấu trúc hoạt động của Dio dựa trên cơ chế request–interceptor–response, giúp quản lý và xử lý dữ liệu ở từng giai đoạn của vòng đời yêu cầu.  
Quá trình hoạt động gồm ba bước chính:

* Gửi yêu cầu (Request): Ứng dụng tạo đối tượng Dio và gửi yêu cầu HTTP (GET, POST, PUT, DELETE, …) đến API. Có thể cấu hình sẵn Base URL, Header, và Timeout để tái sử dụng trong toàn bộ ứng dụng.
* Xử lý trung gian (Interceptor): Trước khi yêu cầu được gửi hoặc phản hồi được trả về, Dio cho phép chèn các interceptor để xử lý dữ liệu hoặc xác thực (Authentication). Ví dụ: thêm token vào header, ghi log, hoặc bắt lỗi tự động.
* Nhận và xử lý phản hồi (Response): Phản hồi từ server được Dio tự động phân tích (parse JSON) và trả về ở dạng có cấu trúc. Nếu gặp lỗi mạng, timeout hoặc status code không hợp lệ, Dio tự động phát sinh Exception để lập trình viên xử lý.

Cơ chế này giúp Dio hoạt động linh hoạt, giảm thiểu code lặp, đồng thời tăng tính bảo mật và khả năng mở rộng hệ thống.

Ví dụ dưới đây minh họa cách sử dụng Dio để lấy dữ liệu từ API JSONPlaceholder:

A computer screen shot of a code

AI-generated content may be incorrect.

Trong ví dụ trên:

* dio.get() gửi yêu cầu HTTP, trả về dữ liệu đã được parse sẵn.
* try–catch bắt và xử lý lỗi mạng, giúp ứng dụng ổn định hơn.

Đặc điểm nổi bật

* Hỗ trợ Interceptor để chèn xử lý trung gian như token, log, hoặc caching.
* Quản lý timeout, retry, và error handling toàn diện.
* Hỗ trợ FormData, multipart upload/download với tiến trình theo dõi.
* Có thể cấu hình Base URL, header mặc định, và tự động parse JSON.
* Phù hợp cho các dự án lớn, có nhiều tầng xử lý mạng hoặc cần tối ưu hiệu năng.

## **3. So sánh giữa HTTP và Dio**:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | HTTP | Dio |
| Mức độ phức tạp / Hiệu năng | Nhẹ, đơn giản, phù hợp dự án nhỏ; ít overhead. | Mạnh mẽ, nhiều tính năng nâng cao; tối ưu nhiều request song song. |
| Xử lý JSON | Phải tự jsonDecode() và parse thủ công. | Tự decode JSON vào response.data, tiện lợi. |
| Cấu hình toàn cục | Không có — mỗi request cấu hình riêng. | BaseOptions cấu hình 1 lần cho toàn bộ request. |
| Interceptors | Không hỗ trợ sẵn. | Có logging, auth token, retry, pipeline xử lý linh hoạt. |
| Upload / Download nâng cao | Không theo dõi progress; hạn chế I/O. | Có FormData + onSendProgress / onReceiveProgress. |
| Quản lý lỗi & Retry | Tự xử lý thủ công dựa vào statusCode. | DioException chi tiết + hỗ trợ retry linh hoạt. |
| BaseURL & API Routing | Phải khai báo URL đầy đủ mỗi request. | baseUrl giúp code gọn, dễ bảo trì. |

# **III. Cơ chế authentication bằng interceptors của thư viện dio:**

## **1. Khái niệm:**

|  |  |  |
| --- | --- | --- |
| **Khái niệm** | **Giải thích** | **Chức năng chính trong ứng dụng Flutter** |
| Interceptor | Bộ chặn trung gian trong Dio, can thiệp vào vòng đời của request/response/lỗi. | • onRequest: thêm token, logging, chỉnh headers. • onResponse: chuẩn hoá dữ liệu. • onError: bắt lỗi, refresh token, retry request. |
| Authentication | Quy trình xác thực danh tính người dùng trước khi truy cập tài nguyên/dịch vụ. | • Đăng nhập nhận access\_token/refresh\_token. • Lưu token an toàn (flutter\_secure\_storage). • Gửi token trong header mỗi request. • Token hết hạn: tự refresh bằng refresh\_token. |

## **2. Mục tiêu của cơ chế**

* Giúp ứng dụng tự động quản lý token mà không cần thao tác thủ công.
* Giảm rủi ro bảo mật do quên gửi token hoặc gửi sai.
* Cải thiện trải nghiệm người dùng: khi token hết hạn, hệ thống tự làm mới mà không yêu cầu đăng nhập lại.
* Dễ dàng bảo trì và mở rộng, vì toàn bộ logic xác thực tập trung tại một nơi (Interceptors).

## **3. Quy trình hoạt động**

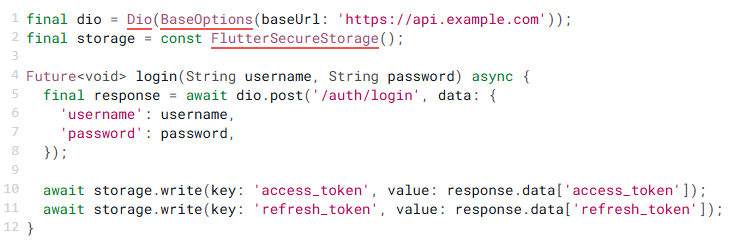
1. Người dùng đăng nhập vào ứng dụng.
2. Server trả về access\_token và refresh\_token.
3. Ứng dụng lưu token vào flutter\_secure\_storage để đảm bảo an toàn.
4. Khi gửi request, Interceptor tự động thêm Authorization: Bearer <access\_token> vào header.
5. Nếu server phản hồi 401 Unauthorized, Interceptor:
   * Gọi API /auth/refresh kèm refresh\_token.
   * Cập nhật token mới và lưu lại.
   * Gửi lại request ban đầu với token hợp lệ.
6. Nếu refresh thất bại, token bị xóa, và người dùng được chuyển về màn hình đăng nhập.

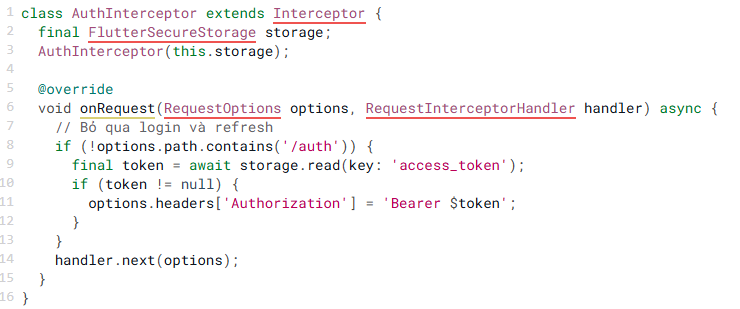
## **4. Cấu trúc các loại Interceptor liên quan:**

| Loại Interceptor | Chức năng |
| --- | --- |
| AuthInterceptor | Thêm access token vào header của mỗi request, trừ các API login/refresh. |
| RefreshInterceptor | Bắt lỗi 401 Unauthorized, gọi refresh token và gửi lại request. |
| LogInterceptor | Ghi lại thông tin request – response để kiểm tra quá trình xác thực. |

## **5. Ví dụ minh họa cơ chế hoạt động:**

**Bước 1 – Đăng nhập và lưu token**

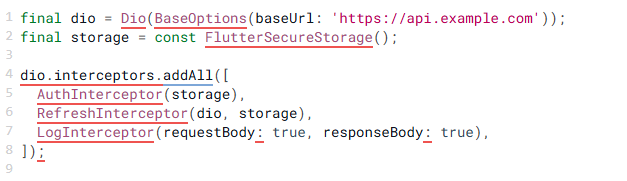
**Bước 2 – Interceptor tự thêm token**



**Bước 3 – Interceptor xử lý refresh token**



**Bước 4 – Gắn Interceptors vào Dio**

→ Sau khi gắn, mọi request sẽ tự động chèn token, tự làm mới token khi hết hạn, và ghi log toàn bộ quá trình xác thực.

## **6. Ưu điểm**

* Tự động hoá toàn bộ quá trình Authentication.
* Giảm sai sót khi gọi API.
* Dễ dàng bảo trì, tách biệt logic xác thực với giao diện.
* Giúp hệ thống an toàn và mượt mà hơn với người dùng.

## **7. Nhược điểm và lưu ý**

* Cần tránh vòng lặp vô hạn khi refresh token thất bại.
* Không log token thật trong môi trường production.
* Cần tách riêng Dio cho /auth để tránh việc interceptor tự gọi chính nó.
* Khi refresh token hết hạn, phải xử lý logout ngay lập tức để đảm bảo an toàn.

## **8. Kết luận:**

Cơ chế Authentication bằng Interceptors trong thư viện Dio là giải pháp hiệu quả giúp tự động hóa quá trình xác thực người dùng trong ứng dụng Flutter.  
Việc sử dụng Interceptor cho phép quản lý token tập trung, giảm lỗi bảo mật, đồng thời tối ưu trải nghiệm người dùng khi làm việc với các API bảo vệ bằng token.

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

## **1. Khái niệm**

**1.1. Error Handling (Xử lý lỗi)**

Error Handling là quá trình phát hiện, phân loại và xử lý lỗi khi ứng dụng giao tiếp với server.  
Trong Flutter (sử dụng Dio), lỗi có thể đến từ:

* Lỗi kết nối: mạng yếu, timeout, mất Internet.
* Lỗi server: máy chủ quá tải hoặc lỗi nội bộ (5xx).
* Lỗi client: request sai, thiếu dữ liệu (4xx).
* Lỗi xác thực: token hết hạn hoặc sai (401, 403).

Mục tiêu của xử lý lỗi là:

* Giúp ứng dụng ổn định và dễ hiểu với người dùng.
* Phân biệt rõ lỗi tạm thời (có thể retry) và lỗi cố định (phải xử lý logic).
* Không để lỗi kỹ thuật gây crash hoặc hiển thị thông tin nhạy cảm.

**1.2. Retry Mechanism (Cơ chế thử lại)**

Retry Mechanism là cơ chế tự động gửi lại request khi gặp lỗi tạm thời như timeout, mạng yếu hoặc server bận.  
Mục tiêu là tăng độ tin cậy của ứng dụng mà không cần người dùng thao tác lại.

Nguyên tắc retry:

* Chỉ áp dụng cho lỗi tạm thời (timeout, 5xx, 429).
* Giới hạn số lần retry (thường 2–3 lần).
* Dùng exponential backoff: thời gian chờ tăng dần (0.5s → 1s → 2s).
* Không retry với lỗi 4xx (request sai, dữ liệu lỗi).
* Với 401, cần refresh token trước khi gửi lại request.

**1.3. Mối quan hệ giữa Error Handling và Retry**

* Error Handling giúp phân loại lỗi chính xác.
* Retry là một phản ứng trong xử lý lỗi, chỉ dùng cho lỗi có thể phục hồi.
* Với lỗi xác thực, cần xử lý qua Interceptor (refresh token) thay vì retry tự động.

## **2. Mục tiêu của cơ chế**

* Đảm bảo ứng dụng ổn định và đáng tin cậy khi giao tiếp với API.
* Tự động xử lý các lỗi thường gặp mà không làm gián đoạn trải nghiệm người dùng.
* Giúp lập trình viên dễ dàng phân loại, hiển thị thông báo lỗi hợp lý (ví dụ: “Mất kết nối mạng”, “Token hết hạn”).
* Giảm lỗi người dùng gửi lại request thủ công khi gặp sự cố.

## **3. Các loại lỗi thường gặp trong Dio**

| **Loại lỗi** | **Mô tả** | **Ví dụ mã trạng thái** |
| --- | --- | --- |
| **Lỗi mạng (Network Error)** | Không kết nối được internet, server không phản hồi. | Timeout, DNS, SocketException |
| **Lỗi từ server (Server Error)** | Server xử lý sai hoặc gặp sự cố nội bộ. | 500, 502, 503 |
| **Lỗi xác thực (Auth Error)** | Token hết hạn hoặc sai định dạng. | 401 Unauthorized |
| **Lỗi dữ liệu (Client Error)** | Request sai hoặc thiếu tham số. | 400, 404 |
| **Lỗi không xác định (Unknown)** | Không thuộc các nhóm trên, thường do exception nội bộ. | null |

## **4. Quy trình xử lý lỗi trong Dio**

1. Phát hiện lỗi:  
   Dio sẽ ném ra một đối tượng DioException khi request thất bại.
2. Phân loại lỗi:  
   Dựa vào error.type và response.statusCode.
3. Xử lý phù hợp:
   * Nếu lỗi là mạng → hiển thị thông báo “Kiểm tra kết nối Internet”.
   * Nếu 401 → gọi interceptor refresh token.
   * Nếu lỗi server (5xx) → kích hoạt cơ chế retry.
   * Nếu lỗi dữ liệu (4xx) → hiển thị lỗi cho người dùng.

## **5. Cấu trúc Error Handling trong Dio**

Ví dụ viết một lớp tiện ích để chuẩn hoá lỗi:



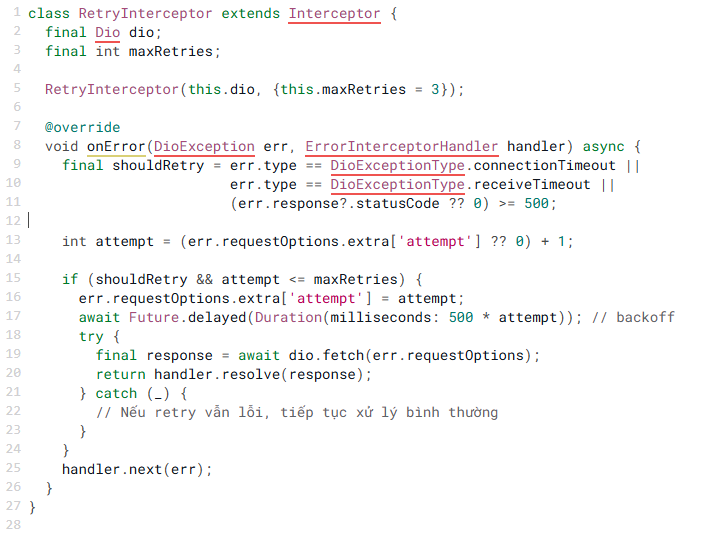
## **6. Retry Mechanism – Cơ chế thử lại**

Khi lỗi là tạm thời, ứng dụng có thể tự động gửi lại request sau một khoảng thời gian chờ nhất định.  
Ví dụ như khi mất mạng tạm thời hoặc server bận (502, 503).

Cơ chế hoạt động

1. Khi request thất bại, Interceptor bắt lỗi.
2. Kiểm tra loại lỗi có được phép retry không.
3. Nếu có, chờ một khoảng thời gian (theo backoff: 0.5s, 1s, 2s, …).
4. Gửi lại request với cùng dữ liệu ban đầu.
5. Nếu retry nhiều lần vẫn thất bại → thông báo lỗi cho người dùng.

Ví dụ Interceptor Retry đơn giản



## **7. Ưu điểm của cơ chế Retry**

* Tăng độ ổn định cho ứng dụng khi mạng không ổn định.
* Giảm lỗi ảo do server quá tải hoặc mất kết nối tạm thời.
* Giúp trải nghiệm người dùng “mượt” hơn khi gặp sự cố nhẹ.

## **8. Lưu ý và hạn chế**

* Không nên retry với lỗi logic (4xx) vì request sai từ phía client.
* Cần giới hạn số lần retry để tránh vòng lặp vô hạn.
* Nên dùng exponential backoff (mỗi lần chờ lâu hơn lần trước) để tránh gây quá tải cho server.
* Ghi log đầy đủ để kiểm tra nguyên nhân lỗi khi debug.

## **9. Kết luận**

Cơ chế Error Handling và Retry trong Dio giúp lập trình viên:

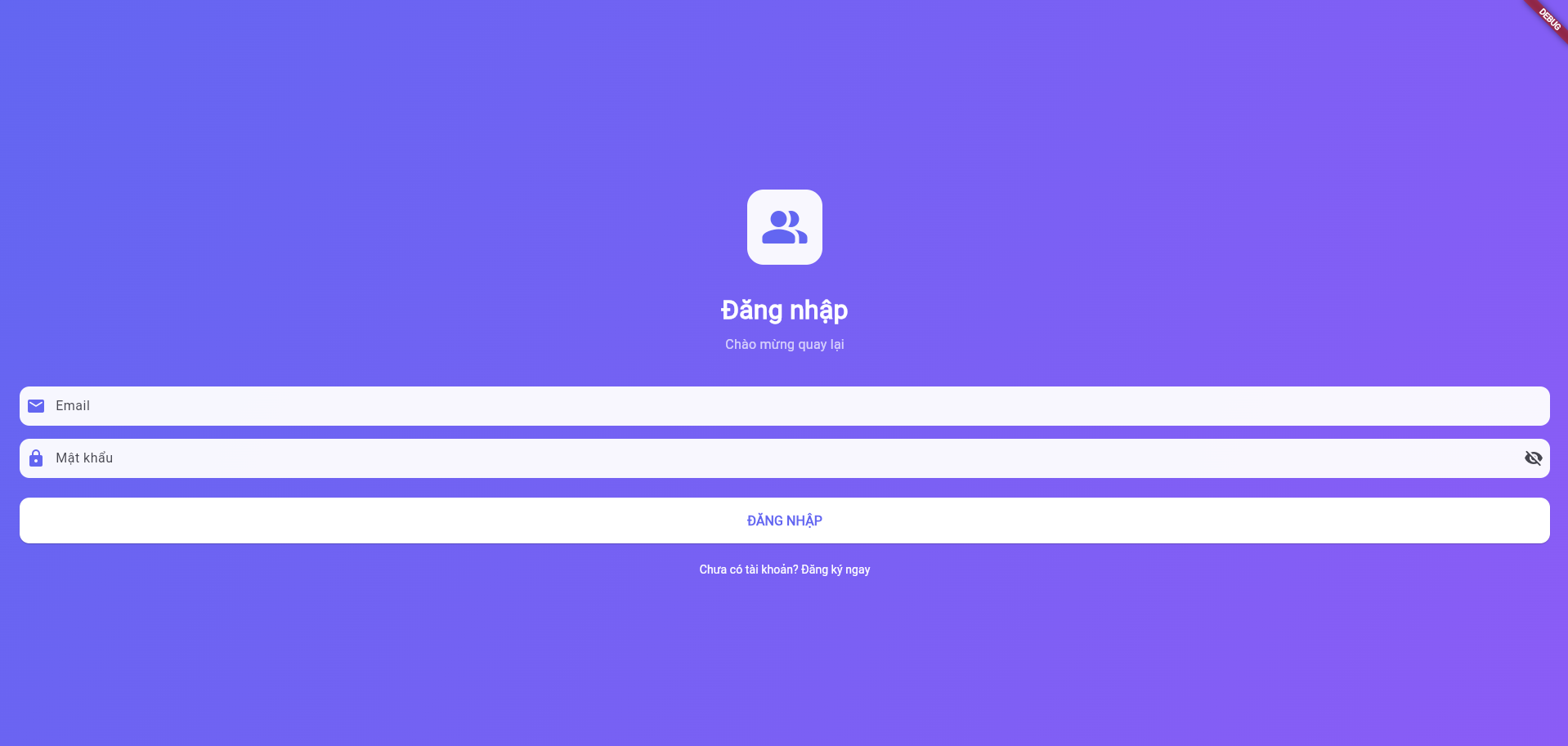
* Dễ dàng kiểm soát các tình huống lỗi phức tạp.
* Giảm crash, cải thiện độ ổn định của ứng dụng.
* Kết hợp với Interceptor, cơ chế này giúp hệ thống API trở nên linh hoạt, tự phục hồi và thân thiện hơn với người dùng.

# **V. Sản phẩm demo:**

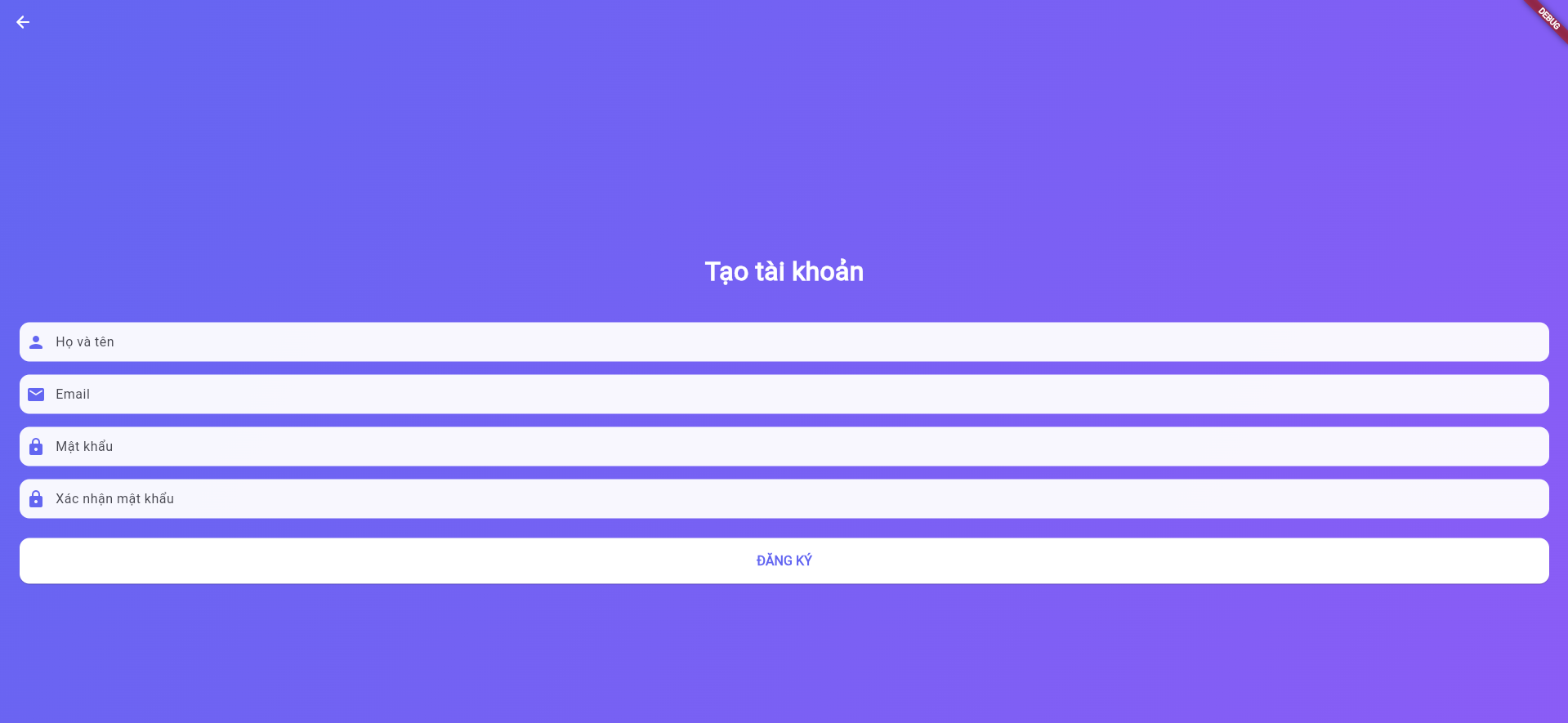
Đây là 1 sản phẩm demo về 1 mạng xã hội mà các bài post được lấy dio get từ trang web sau: <https://jsonplaceholder.typicode.com/posts>. Đồng thời phần đăng nhập và đăng ký sử dụng cơ chế authentication.

- Các trang widget chính của sản phẩm:

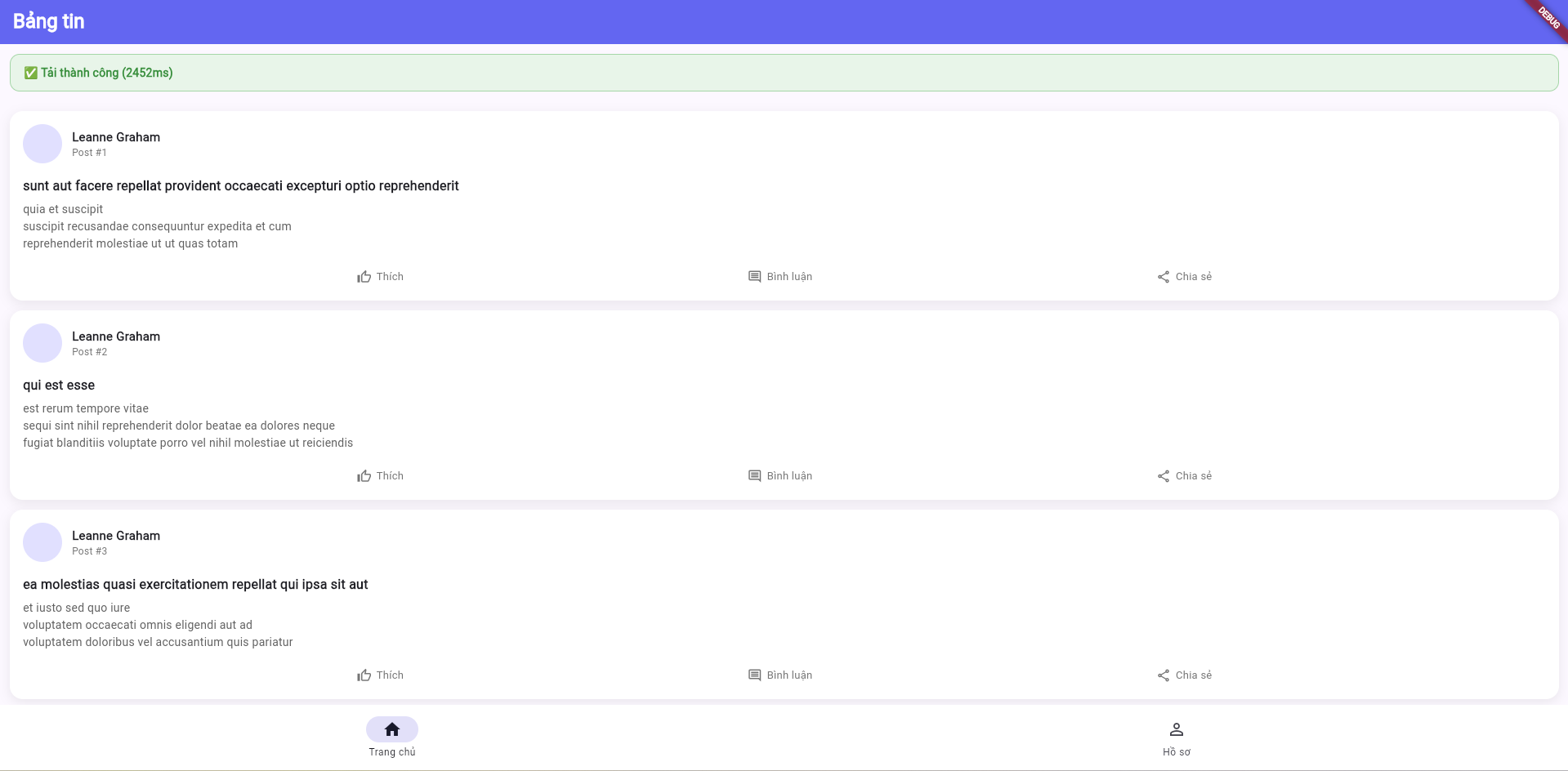
1. Trang đăng nhập:

****

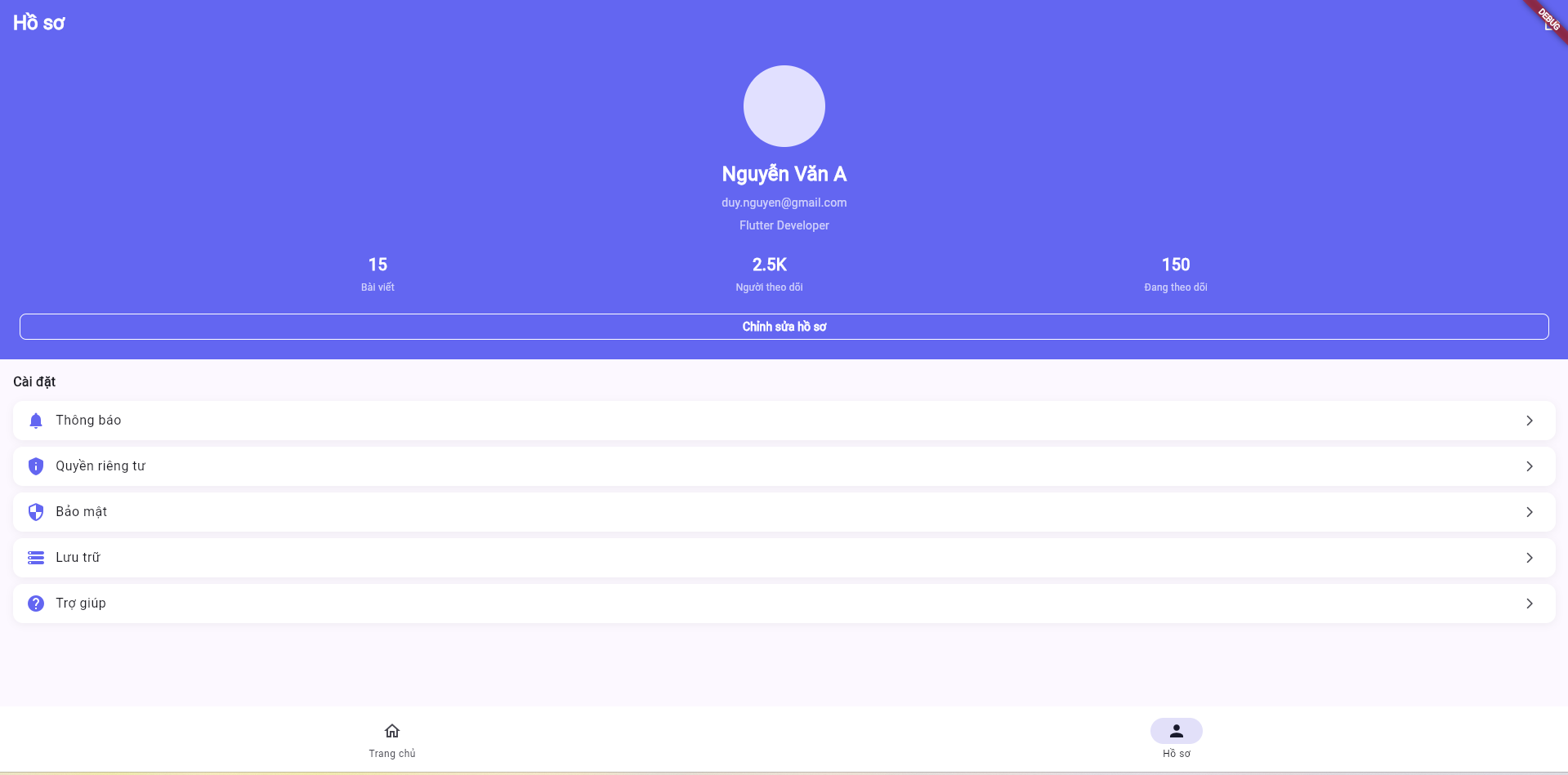
1. Trang đăng ký:



1. Trang bảng tin



1. Trang hồ sơ:



# **TÀI LIỆU THAM KHẢO**

[1] Flutter Team, “dio package,” *pub.dev*, 2025. [Online]. Available:https://pub.dev/packages/dio  
[2] Dart Team, “http package,” *pub.dev*, 2025. [Online]. Available: https://pub.dev/packages/http

[3] “RESTful API Tutorial.” *restfulapi.net*, 2025.

[4] Flutter Community, “Retry failed HTTP requests with Dio,” *Medium*, 2023.

[5] Flutter Team, *Flutter Documentation*, Google LLC, 2025. [Online]. Available: <https://docs.flutter.dev>

[6] K. Pardeshi, “Flutter Dio Tutorial: The Ultimate HTTP Client for Flutter Development,” *Mobisoft Infotech Blog*, Jun. 3, 2025. [Online]. Available: [https://mobisoftinfotech.com/resources/blog/flutter-development/flutter-dio-tutorial-http-client](https://mobisoftinfotech.com/resources/blog/flutter-development/flutter-dio-tutorial-http-client?utm_source=chatgpt.com)