**TRƯỜNG ĐẠI HỌC LÂM NGHIỆP**

KHOA KINH TẾ VÀ QUẢN TRỊ KINH DOANH

**===o0o===**

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**MÔN: LẬP TRÌNH TRÊN CÁC THIẾT BỊ DI ĐỘNG**

**ĐỀ TÀI: PHÂN TÍCH VÀ XÂY DỰNG PHẦN MỀM QUẢN LÝ TRUY VẾT**



**Giảng viên hướng dẫn: ThS. Mai Hà An**

**Sinh viên thực hiện: Bùi Văn Mạnh**

**Mã sinh viên: 1851070199**

**Lớp: K63 - HTTT**

**Hà Nội, 2021**

**MỤC LỤC**

MỤC LỤC

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH ẢNH

LỜI CẢM ƠN

[CHƯƠNG I: TỔNG QUAN ĐỀ TÀI NGHIÊN CỨU 1](#_Toc91945010)

[1.1. Lý do chọn đề tài 1](#_Toc91945011)

[1.2. Mục đích chọn đề tài 1](#_Toc91945012)

[1.3. Mục tiêu và phạm vi nghiên cứu 1](#_Toc91945013)

[1.3. Phương pháp nghiên cứu 1](#_Toc91945014)

[CHƯƠNG II: CƠ SỞ LÝ THUYẾT 2](#_Toc91945015)

[2.1. Công nghệ Flutter 2](#_Toc91945016)

[2.1.1. Ngôn ngữ dart 2](#_Toc91945017)

[2.2. SQLite 4](#_Toc91945018)

[2.3. API, RESTful API 5](#_Toc91945019)

[CHƯƠNG III: PHÂN TÍCH THIẾT KẾ HỆ THỐNG 10](#_Toc91945020)

[3.1. Cơ sở dữ liệu 10](#_Toc91945021)

[3.2. Biểu đồ UC hệ thống 11](#_Toc91945022)

[3.3. Biểu đồ hoạt động 12](#_Toc91945023)

[CHƯƠNG IV: LẬP TRÌNH VÀ KẾT QUẢ 17](#_Toc91945024)

[4.1. Các hàm chức năng 17](#_Toc91945025)

[4.1.1. User\_API 17](#_Toc91945026)

[4.1.2. Hiển thị bản đồ người dùng di chuyển 19](#_Toc91945027)

[4.1.3. Xem lịch sử di chuyển 21](#_Toc91945028)

[4.1.4. Quản lý truy vết 23](#_Toc91945029)

[4.2. Kết quả lập trình 25](#_Toc91945030)

[4.2.1 Giao diện Người dùng chưa đăng nhập 25](#_Toc91945031)

[4.2.2. Giao diện người dùng 26](#_Toc91945032)

[4.2.3. Giao diện quản trị viên 27](#_Toc91945033)

[4.2.4. Giao diện quản lý truy vết 28](#_Toc91945034)

[4.2.6. Giao diện lịch sử di chuyển 29](#_Toc91945035)

[4.2.6. Giao diện xem lịch sử di chuyển trên bản đồ 30](#_Toc91945036)

[CHƯƠNG V: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 31](#_Toc91945037)

[5.1. Kết luận 31](#_Toc91945038)

[5.2. Hướng phát triển 31](#_Toc91945039)

TÀI LIỆU THAM KHẢO

**DANH MỤC BẢNG BIỂU**

[Bảng 1: Bảng Lịch sử 10](#_Toc91944926)

[Bảng 2: Bảng người dùng 10](#_Toc91944927)

**DANH MỤC HÌNH ẢNH**

[Hình 1: Công nghệ Flutter 2](file:///D:\K63HTTH\Subjects\LapTrinhThietBiDiDong\Final_Proejct\BC\BC_LTTBDD_Edit_L1.docx#_Toc91944993)

[Hình 2: Ngôn ngữ dart 2](file:///D:\K63HTTH\Subjects\LapTrinhThietBiDiDong\Final_Proejct\BC\BC_LTTBDD_Edit_L1.docx#_Toc91944994)

[Hình 3: Web APIs 5](#_Toc91944995)

[Hình 4: REST API 7](#_Toc91944996)

[Hình 5: Mô hình hoạt động của RESTful 8](#_Toc91944997)

[Hình 6: Biểu đồ UC Hệ thống 11](#_Toc91944998)

[Hình 7: Biểu đồ hoạt động đăng ký tài khoản 12](#_Toc91944999)

[Hình 8: Biểu đồ hoạt động đăng nhập 13](#_Toc91945000)

[Hình 9: Biểu đồ hoạt động thêm vị trí hiện tại 14](#_Toc91945001)

[Hình 10: Biểu đồ hoạt động xem lịch sử di chuyển 15](#_Toc91945002)

[Hình 11: Biểu đồ hoạt động quản lý truy vết 16](#_Toc91945003)

[Hình 12: Giao diện người dùng chưa đăng nhập 25](#_Toc91945004)

[Hình 13: Giao diện người dùng 26](#_Toc91945005)

[Hình 14: Giao diện quản trị viên 27](#_Toc91945006)

[Hình 15: Giao diện quản lý truy vết 28](#_Toc91945007)

[Hình 16: Giao diện lịch sử di chuyển 29](#_Toc91945008)

[Hình 17: Giao diện xem lịch sử di chuyển trên bản đồ 30](#_Toc91945009)

**LỜI CẢM ƠN**

Em xin gửi lời cảm ơn chân thành tới các Thầy, Cô trong trường Đại học lâm nghiệp và thầy cô giáo tại bộ môn Tin Học. Đặc biệt, em xin gửi lời cảm ơn chân thành tới **Thầy** **Mai Hà An** – đã tận tình giúp đỡ, trực tiếp chỉ bảo, hướng dẫn nhóm trong suốt quá trình nghiên cứu và hoàn thành tốt môn học Lập trình trên các thiết bị di động trong đề tài: ***“Phân tích và Xây dựng phần mềm quản lý truy vết”.***

Em xin chân thành cảm ơn!

*Hà Nội, ngày 30 tháng 12 năm 2021*

**Sinh viên thực hiện**

**Bùi Văn Mạnh**

# TỔNG QUAN ĐỀ TÀI NGHIÊN CỨU

## 1.1. Lý do chọn đề tài

Trong thời gian qua, sự bùng phát của dịch COVID-19 đã gây ra những tác động không nhỏ đối với sự phát triển kinh tế - xã hội. Việc thực hiện cách ly xã hội trên diện rộng khiến cuộc sống sinh hoạt của người dân bị gián đoạn, nhiều trường học, công sở phải tạm đóng cửa, hoạt động sản xuất kinh doanh của doanh nghiệp bị đinh trệ, v.v… và khiến tổn thất đối với sự phát triển kinh tế - xã hội của các địa phương trên cả nước là ngày càng lớn. Trong khi diễn biến của dịch bệnh vẫn đang rất phức tạp, chưa có sự dự liệu chính xác về thời điểm kết thúc, và càng không thể tiên liệu được sẽ còn có những dịch bệnh nào khác sau COVID-19 có thể xuất hiện, thì một trong những giải pháp được cho là hiệu quả nhất đối với xã hội chính là sử dụng những ứng dụng công nghệ số để giúp toàn xã hội tiếp tục vận hành song song với sự tồn tại của dịch bệnh.

Vì lý do trên em đã lựa chọn đề tài:

***“Phân tích và Xây dựng phần mềm quản lý truy vết”***

Để làm đồ án cuối kỳ cho môn học Lập trình với các thiết bị di động do thầy **Mai Hà An** giảng dạy.

## 1.2. Mục đích chọn đề tài

* Rèn luyện kỹ năng phân tích và thiết kế hệ thống.
* Xây dựng hệ thống quản lý truy vết đơn giản hiệu quả đáp ứng được nhu cầu đề ra.

## 1.3. Mục tiêu và phạm vi nghiên cứu

#### Mục tiêu

Ứng dụng các công nghệ đã được học xây dựng thành công hệ thống quản lý truy vết.

#### Phạm vi nghiên cứu

* Phạm vi công nghệ: Lưu trữ cơ sở dữ liệu trên SQLite và lập trình trên phần mềm Android studio, Pycharm.
* Phạm vi con người: Sinh viên Đại học lâm nghiệp.

## 1.3. Phương pháp nghiên cứu

* Phương pháp kế thừa tài liệu
* Phương pháp chuyên gia

# Flutter là gì?CƠ SỞ LÝ THUYẾT

Hình 1: Công nghệ Flutter

## Công nghệ Flutter

Flutter là một công cụ mới được cung cấp bởi Google cho phép các nhà phát triển xây dựng các ứng dụng đa nền tảng có thể được thực hiện trong các hệ thống khác nhau chẳng hạn như Android hay iOS chỉ với một codebase chung.

Công cụ này được được xây dựng trong C và C ++ và cung cấp một cơ chế rendering 2D, một funtional-reactive framework là React-inspired, và một tập hợp các Material Design widget. Nó hiện đang được distribute bản alpha: version 0.0.20, tuy vậy giai đoạn đầu của nó đã được cho phép để tạo ra interfacing phức tạp, thực hiện kết nối mạng và thậm chí quản lý tập tin.

Cách tiếp cận Flutter's là khác nhau từ các solution khác, ví dụ Cordova chạy trên một WebView là HTML, CSS và Javascript. Không giống như những công cụ này, nó chỉ sử dụng Dart như một ngôn ngữ lập trình duy nhất. Dart là khá dễ dàng để tìm hiểu và nếu bạn có kiến thức Java, 75% của công việc được gần như hoàn tất và làm quen với Dart sẽ chỉ mất một vài ngày.

Ứng dụng sẽ không thực thi mã Dart trực tiếp. Theo thời gian, một ứng dụng được release xây dựng, mã sẽ được biên dịch để local nhận được hiệu suất như một kết quả tốt hơn và đáp ứng giao diện người dùng tốt hơn. Trong khi phát triển trong chế độ debug (kiểm tra đối với các lỗi tiềm năng) Flutter cũng thực hiện một số nhiệm vụ mà có thể làm cho các ứng dụng chạy chậm hơn. Nếu trải qua tình trạng này, Flutter sẽ cho biết cách đặt một dải ruy băng màu đỏ đầu ngay trong màn hình với dòng chữ "Slow Mode" ở trên đó.

### Ngôn ngữ dart



Hình 2: Ngôn ngữ dart

Dart là ngôn ngữ lập trình đa mục đích, xây dựng và phát triển bởi Google. Sau đó, nó được Ecma (ECMA-408) phê duyệt làm tiêu chuẩn.

Dart thường được dùng trong việc xây dựng những ứng dụng web, server trên máy tính để bàn và nhiều thiết bị di động khác.

Ngoài ra, Dart còn là một ngôn ngữ hướng đối tượng, được xác định theo lớp và hoạt động dưới cơ chế garbage-collected. Nó sử dụng cú pháp kiểu C để dịch mã sang JavaScript đúng như yêu cầu.

Dart cũng nổi tiếng như là một ngôn ngữ mã nguồn mở, hỗ trợ sử dụng miễn phí và được phát triển trên GitHub.

Ngôn ngữ Dart hoạt động tương tự với Java và là phiên bản cải tiến từ JavaScript. Do cùng là hệ C nên cú pháp của chúng có nét tương đồng nhau.

#### Ngôn ngữ Python

Python là ngôn ngữ lập trình hướng đối tượng, cấp cao, mạnh mẽ, được tạo ra bởi Guido van Rossum. Nó dễ dàng để tìm hiểu và đang nổi lên như một trong những ngôn ngữ lập trình nhập môn tốt nhất cho người lần đầu tiếp xúc với ngôn ngữ lập trình. Python hoàn toàn tạo kiểu động và sử dụng cơ chế cấp phát bộ nhớ tự động. Python có cấu trúc dữ liệu cấp cao mạnh mẽ và cách tiếp cận đơn giản nhưng hiệu quả đối với lập trình hướng đối tượng. Cú pháp lệnh của Python là điểm cộng vô cùng lớn vì sự rõ ràng, dễ hiểu và cách gõ linh động làm cho nó nhanh chóng trở thành một ngôn ngữ lý tưởng để viết script và phát triển ứng dụng trong nhiều lĩnh vực, ở hầu hết các nền tảng.

**Lịch sự phát triển của lập trình giao diện Python:**

Sự phát triển của ngôn ngữ lập trình Python phát triển theo các giai đoạn sau:

Giai đoạn 1: Là các bản phát hành python 1x. Giai đoạn này bắt đầu từ năm 1990 cho đến 2000. Năm 1990 đến 1995 , Guido van Rossum làm việc tại CWI trung tâm toán tin tại Amsterdam, Hà Lan. Do vậy phiên bản Python đầu tiên do trung tâm toán tin phát hành. Phiên bản cuối cùng tại CWI là python đời 1.2.

Năm 1995, ông chuyển sang CNRI ở Reston, Virginia. Ông đã nâng cấp đến Python 1.6 – đây cũng là bản cuối cùng của ông tại CNRI.

Sau đó ông rời CNRI và làm việc với các lập trình viên tự do chuyên viết phần mềm thương mại. Lúc đó ông nảy sinh ý tưởng kết hợp giữa Python với các phần mềm tuân theo GPL. Và chính CNRI và FSF hợp tác để làm bản quyền này của ông. Cũng trong năm này, ông được giải thưởng vì sự phát triển phần mềm tự do của FSF trao tặng.

Một thời gian sau phiên bản 16.1 ra đời tuân theo bản quyền GPL.

Giai đoạn 2: Năm 2000, Guido van Rossum cùng nhóm phát triển dời đến BeOpen.com để thành lập team phát triển Python. Và phiên bản 2.0 được phát triển tại đây. Sau đó Guido và các thành viên tham gia vào Digital Creations.

Sau đó phiên bản 2.1 ra đời dựa trên Python 1.6.1 và phiên bản 2.0. Từ thời điểm này Python thuộc sở hữu của PSF- đây là một tổ chức phi lợi nhuận.

Giai đoạn 3: Phiên bản 3x không tương thích hoàn toàn với 2x nhưng lại có công cụ hỗ trợ chuyển đổi từ phiên bản này sang phiên bản kia. Nguyên tắc chủ đạo trong việc phát triển Python ở phiên bản 3X chính là hạn chế sự trùng lặp về chức năng của Python. Ở phiên bản 3X có những thay đổi trong cú pháp và bổ sung thêm một số cú pháp mới.

## SQLite

SQLite là hệ quả trị cơ sở dữ liệu (DBMS) quan hệ tương tự như Mysql, ... Đặc điểm nổi bật của SQLite so với các DBMS khác là gọn, nhẹ, đơn giản, đặt biệt không cần mô hình server-client, không cần cài đặt, cấu hình hay khởi động nên không có khái niệm user, password hay quyền hạn trong SQLite Database. Dữ liệu cũng được lưu ở một file duy nhất.

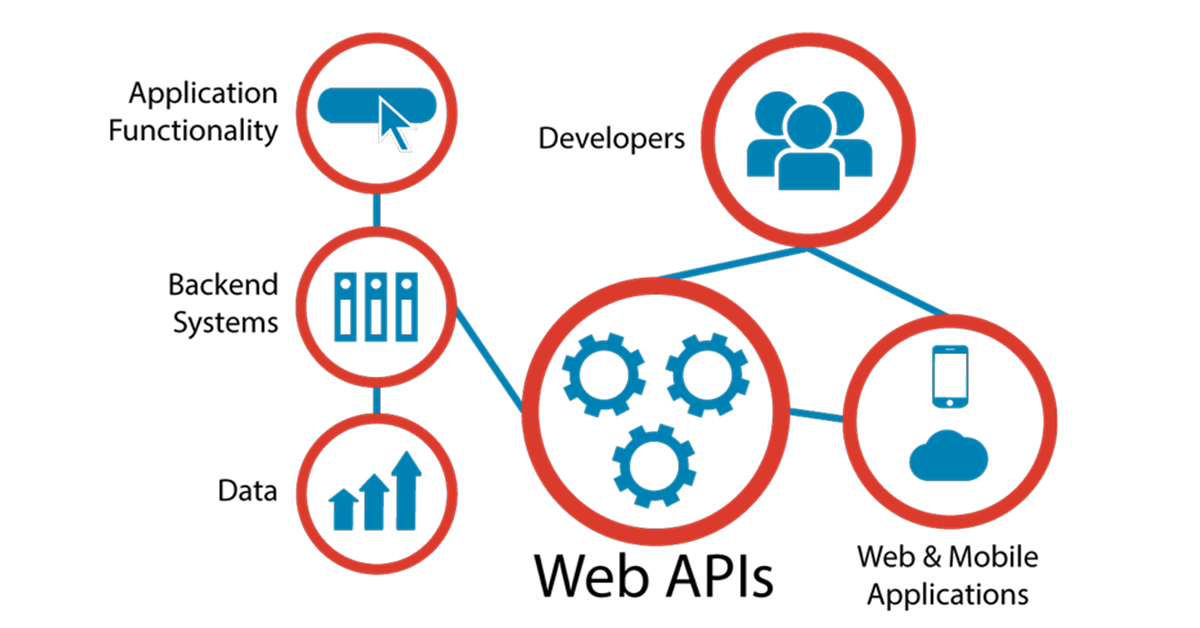
SQLite thường không được sử dụng với các hệ thống lớn nhưng với những hệ thống ở quy mô vùa và nhỏ thì SQLite không thua các DBMS khác về chức năng hay tốc độ. Vì không cần cài đặt hay cấu hình nên SQLite được sử dụng nhiều trong việc phát triển, thử nghiệm … vì tránh được những rắc rối trong quá trình cài đặt.

#### Tính năng của SQLite

* Giao dịch trong SQLite tuân thủ theo nguyên tắc (ACID) ngay cả sau hi hệ thống treo và mất điện.
* Không cấu hình:
* Không cần thiết lập hoặc quản trị
* SQLite hỗ trợ với đầy đủ tính năng với các khả năng nâng cao như các chỉ mục 1 phần, các chỉ mục về các biểu thức, JSON và các biểu thức bảng chung.
* Một sở dữ liệu hoàn chỉnh được lưu trữ trong một tệp đa nền tảng duy nhất. Phù hợp với sử dụng dưới dạng định dạng tệp ứng dụng
* Hỗ trợ các cơ sở dữ liệu có kích thước terabyte và các chuỗi có kích thước gigabyte.
* API
* Đơn giản dễ sử dụng
* Nhanh: Trong một số trường hợp, SQLite nhanh hơn hệ thống tệp tin trực tiếp I/O.
* Được viết bằng ANSI-C.
* Bindings cho hàng chục ngôn ngữ khác có sẵn 1 cách riêng biệt.
* Mã nguồn đầy, nguồn mở đủ có thể kiểm tra nhánh 100%.
* Nền tảng đa nền tảng:
* SQLite là có sẵn trên Android, \*BSD, iOS, Linux, Mac, Solaris, Windows,.. Dễ dàng dịch chuyển sang các hệ thống khác.

## API, RESTful API

**API** là các phương thức, giao thức kết nối với các thư viện và ứng dụng khác. Nó là viết tắt của **Application Programming Interface** – giao diện lập trình ứng dụng. API cung cấp khả năng cung cấp khả năng truy xuất đến một tập các hàm hay dùng. Và từ đó có thể trao đổi dữ liệu giữa các ứng dụng.



Hình 3: Web APIs

**API thường ứng dụng vào đâu?**

* Web API: là hệ thống API được sử dụng trong các hệ thống website. Hầu hết các website đều ứng dụng đến Web API cho phép bạn kết nối, lấy dữ liệu hoặc cập nhật cơ sở dữ liệu. Ví dụ: Bạn thiết kế chức nằng login thông Google, Facebook, Twitter, Github… Điều này có nghĩa là bạn đang gọi đến API của. Hoặc như các ứng dụng di động đều lấy dữ liệu thông qua API.
* API trên hệ điều hành: Windows hay Linux có rất nhiều API, họ cung cấp các tài liệu API là đặc tả các hàm, phương thức cũng như các giao thức kết nối. Nó giúp lập trình viên có thể tạo ra các phần mềm ứng dụng có thể tương tác trực tiếp với hệ điều hành.
* API của thư viện phần mềm hay framework: API mô tả và quy định các hành động mong muốn mà các thư viện cung cấp. Một API có thể có nhiều cách triển khai khác nhau và nó cũng giúp cho một chương trình viết bằng ngôn ngữ này có thể sử dụng thư viện được viết bằng ngôn ngữ khác. Ví dụ bạn có thể dùng Php để yêu cầu một thư viện tạo file PDF được viết bằng C++.

**Web API là gì?**

**Web API** là một phương thức dùng để cho phép các ứng dụng khác nhau có thể giao tiếp, trao đổi dữ liệu qua lại. Dữ liệu được Web API trả lại thường ở dạng [JSON](https://topdev.vn/blog/json-la-gi/) hoặc XML thông qua giao thức HTTP hoặc HTTPS.

**Những điểm nổi bật của Web API**

Web API hỗ trợ restful đầy đủ các phương thức: Get/Post/put/delete dữ liệu. Nó giúp bạn xây dựng các HTTP service một cách rất đơn giản và nhanh chóng. Nó cũng có khả năng hỗ trợ đầy đủ các thành phần HTTP: URI, request/response headers, caching, versioning, content forma.

Hỗ trợ đầy đủ các thành phần MVC như: routing, controller, action result, filter, model binder, IoC container, [dependency injection](https://topdev.vn/blog/dependency-injection-la-gi/), unit test.

**Web API hoạt động như thế nào?**

1. Đầu tiên là xây dựng URL API để bên thứ ba có thể gửi request dữ liệu đến máy chủ cung cấp nội dung, dịch vụ thông qua giao thức HTTP hoặc HTTPS.
2. Tại web server cung cấp nội dung, các ứng dụng nguồn sẽ thực hiện kiểm tra xác thực nếu có và tìm đến tài nguyên thích hợp để tạo nội dung trả về kết quả.
3. Server trả về kết quả theo định dạng JSON hoặc XML thông qua giao thức HTTP/HTTPS.
4. Tại nơi yêu cầu ban đầu là ứng dụng web hoặc ứng dụng di động , dữ liệu JSON/XML sẽ được parse để lấy data. Sau khi có được data thì thực hiện tiếp các hoạt động như lưu dữ liệu xuống Cơ sở dữ liệu, hiển thị dữ liệu…

Có thể nói nguyên lí **REST** và cấu trúc dữ liệu **RESTful** được biết đến rộng rãi trong giới [lập trình web](https://topdev.vn/blog/lap-trinh-web/) nói chung và lập trình ứng dụng nói riêng.

Có thể nói bản thân **REST** không phải là một loại công nghệ. Nó là phương thức tạo API với nguyên lý tổ chức nhất định. Những nguyên lý này nhằm hướng dẫn lập trình viên tạo môi trường xử lý API request được toàn diện.

Để hiểu rõ hơn về **RESTful API** ta sẽ đi lần lượt giải thích các khái niệm nhở **API** , **REST** hay **RESTful**.

[RESTful API là gì?](https://topdev.vn/blog/restful-api-la-gi/)

**RESTful API** là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động…), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.



Hình 4: REST API

Diễn giải các thành phần

**API** ( **A** pplication **P** rogramming **I** nterface) là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như [JSON](https://topdev.vn/blog/json-la-gi/) hay XML.

**REST** ( **RE**presentational **S**tate **T**ransfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE, vv đến một URL để xử lý dữ liệu.

**RESTful API** là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource. RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau.

Chức năng quan trọng nhất của **REST** là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE…) và cách định dạng các URL cho ứng dụng web để quản các resource. RESTful không quy định logic code ứng dụng và không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một **RESTful API**.

RESTful hoạt động như thế nào?



Hình 5: Mô hình hoạt động của RESTful

REST hoạt động chủ yếu dựa vào giao thức HTTP. Các hoạt động cơ bản nêu trên sẽ sử dụng những phương thức HTTP riêng.

* GET (SELECT): Trả về một Resource hoặc một danh sách Resource.
* POST (CREATE): Tạo mới một Resource.
* PUT (UPDATE): Cập nhật thông tin cho Resource.
* DELETE (DELETE): Xoá một Resource.

Những phương thức hay hoạt động này thường được gọi là **CRUD** tương ứng với Create, Read, Update, Delete – Tạo, Đọc, Sửa, Xóa.

Hiện tại đa số lập trình viên viết **RESTful API** giờ đây đều chọn JSON là format chính thức nhưng cũng có nhiều người chọn XML làm format, nói chung dùng thế nào cũng được miễn tiện và nhanh.

Authentication và dữ liệu trả về

**RESTful API** không sử dụng [session](https://topdev.vn/blog/session-la-gi/) và cookie, nó sử dụng một access\_token với mỗi request. Dữ liệu trả về thường có cấu trúc như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | {      "data":{          "id":"1",          "name":"maihaan"      }  } |

Status code

Khi chúng ta request một API nào đó thường thì sẽ có vài status code để nhận biết sau:

* 200 OK – Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
* 201 Created – Trả về khi một Resouce vừa được tạo thành công.
* 204 No Content – Trả về khi Resource xoá thành công.
* 304 Not Modified – Client có thể sử dụng dữ liệu cache.
* 400 Bad Request – Request không hợp lệ
* 401 Unauthorized – Request cần có auth.
* 403 Forbidden – bị từ chối không cho phép.
* 404 Not Found – Không tìm thấy resource từ URI
* 405 Method Not Allowed – Phương thức không cho phép với user hiện tại.
* 410 Gone – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
* 415 Unsupported Media Type – Không hỗ trợ kiểu Resource này.
* 422 Unprocessable Entity – Dữ liệu không được xác thực
* 429 Too Many Requests – Request bị từ chối do bị giới hạn

**Các Package**

* **body-parser** (parse các request tới server)
* **express** (làm cho ứng dụng chạy)
* **nodemon** (restart khi có thay đổi xảy ra)
* **mongoose** (mô hình hóa object data để đơn giản hóa các tương tác với MongoDB)
* **bcrypt** (hashing và salting passwords)
* **express session** (xử lý sessions)
* **connect-mongo** (lưu trữ session trong MongoDB)
* **dotenv** (sử dụng .env)
* **express-validator**
* **morgan**

# PHÂN TÍCH THIẾT KẾ HỆ THỐNG

## 3.1. Cơ sở dữ liệu

* **Bảng Lịch sử (LichSu):**

|  |  |  |
| --- | --- | --- |
| Tên trường | Kiểu dữ liệu | Mô tả |
| uuid | Char | Khóa chính |
| Lat | Varchar | Vĩ độ |
| Lng | Varchar | Kinh độ |
| Date | Date | Ngày tracking |
| uuidUser | Char | Id người dùng |

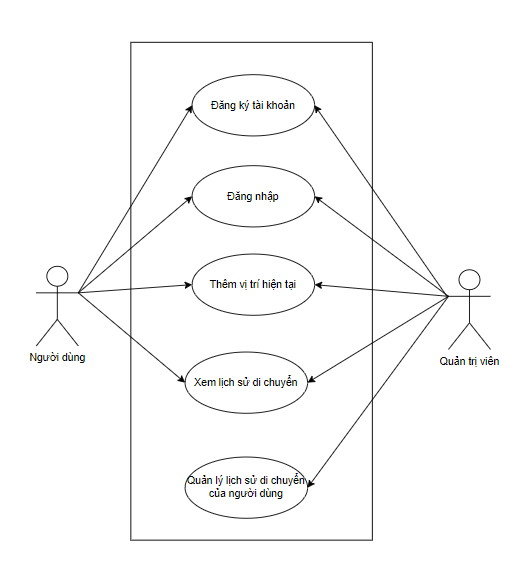
Bảng 1: Bảng Lịch sử

* **Bảng Người dùng (User):**

|  |  |  |
| --- | --- | --- |
| Tên trường | Kiểu dữ liệu | Mô tả |
| Uuid | Char | Khóa chính |
| Username | Varchar | Tên đăng nhập |
| Password | Varchar | Mật khẩu |
| Name | Varchar | Tên người dùng |
| Role | Int | Phân quyền |
| Diachi | Nvarchar | Địa chỉ |
| Cmnd | Varchar | Chứng minh nhân dân |

Bảng 2: Bảng người dùng

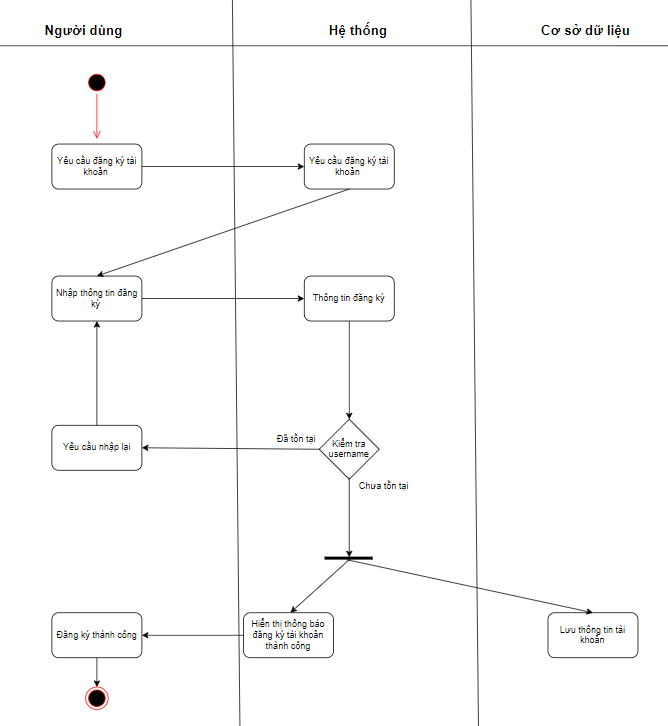
## 3.2. Biểu đồ UC hệ thống



Hình 6: Biểu đồ UC Hệ thống

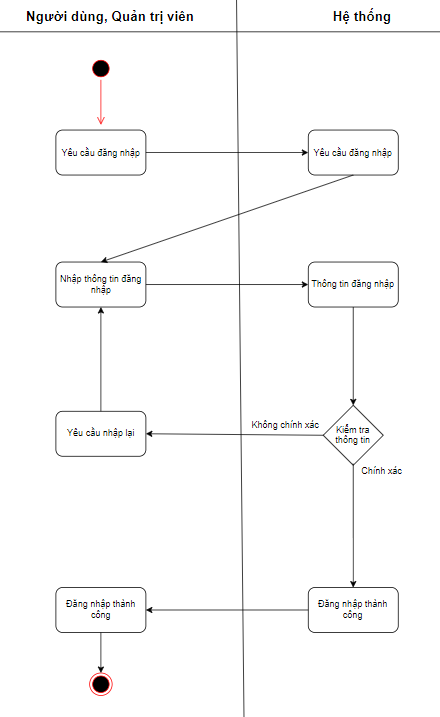
## 3.3. Biểu đồ hoạt động

* **Biểu đồ hoạt động đăng ký tài khoản**



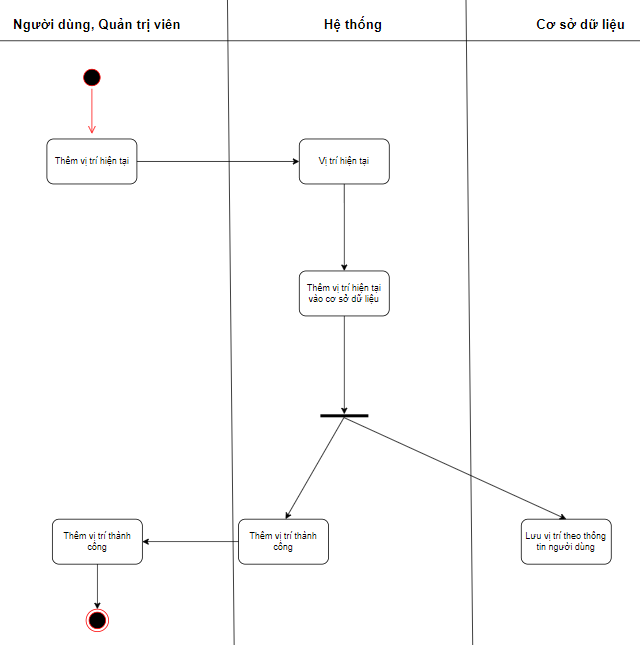
Hình 7: Biểu đồ hoạt động đăng ký tài khoản

* **Biểu đồ hoạt động đăng nhập**



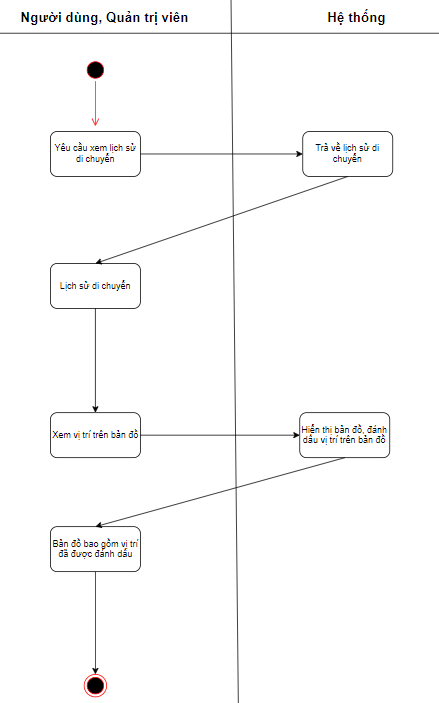
Hình 8: Biểu đồ hoạt động đăng nhập

* **Biểu đồ hoạt động thêm vị trí hiện tại**



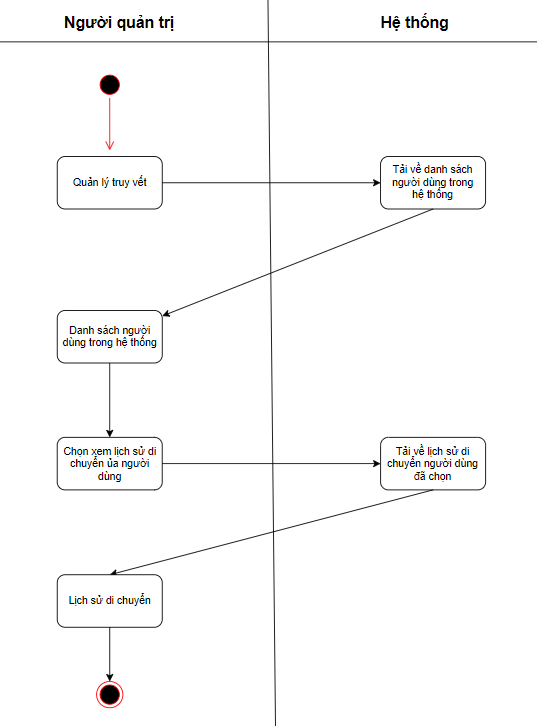
Hình 9: Biểu đồ hoạt động thêm vị trí hiện tại

* **Biểu đồ hoạt động xem lịch sử di chuyển**

****

Hình 10: Biểu đồ hoạt động xem lịch sử di chuyển

* **Biểu đồ hoạt động quản lý truy vết**



Hình 11: Biểu đồ hoạt động quản lý truy vết

# LẬP TRÌNH VÀ KẾT QUẢ

## 4.1. Các hàm chức năng

### 4.1.1. User\_API

**Mô tả:** Api được đặt ở phía server giúp giao tiếp giữa người dùng và hệ thống thông qua chuẩn dịch vụ REST API. API thực hiện các chức năng thêm, sửa, xóa, trả về danh sách thông qua các phương thức: GET, POST, PUT, …

**Chức năng:** API

**from** rest\_framework.views **import** APIView  
**from** app\_tracking\_location.serializers **import** \*  
**from** rest\_framework.response **import** Response  
**from** app\_tracking\_location.models.users **import** User  
  
  
**class** User\_Api\_View(APIView):  
 **def** get(self, request):  
 **try**:  
 type = request.GET[**'type'**]  
 **if** type == **"get-all-user"**:  
 queryset = User.objects.all()  
 serializer = User\_Serializer(queryset, many=**True**)  
 **return** Response(serializer.data)  
 **elif** type == **"get-user-by-cmnd"**:  
 cmnd = request.GET[**'cmnd'**]  
 queryset = User.objects.filter(cmnd=cmnd)  
 serializer = User\_Serializer(queryset, many=**True**)  
 **return** Response(serializer.data)  
 **except**:  
 queryset = User.objects.all()  
 serializer = User\_Serializer(queryset, many=**True**)  
 **return** Response(serializer.data)  
  
 **def** post(self, request):  
 **try**:  
 type = request.data[**'type'**]  
 **if** type == **'login'**:  
 username = request.data[**'username'**]  
 password = request.data[**'password'**]  
 **try**:  
 user = User.objects.get(username=username, password=password)  
  
 *# True* **if** user **is not None**:  
 *# User* **if** user.role == 2:  
 serializer = User\_Serializer(user)  
 **return** Response({  
 **'uuid'**: user.uuid,  
 **'name'**: user.name,  
 **'diachi'**: user.diachi,  
 **'role'**: user.role,  
 **'status'**: 200  
 })  
 *# Admin* **else**:  
 serializer = User\_Serializer(user)  
 **return** Response({  
 **'uuid'**: user.uuid,  
 **'name'**: user.name,  
 **'diachi'**: user.diachi,  
  
 **'role'**: user.role,  
 **'status'**: 200  
 })  
 *# Fail* **else**:  
 **return** Response({  
 **'data'**: -1,  
 **'role'**: -1,  
 **'status'**: 500  
 })  
 **except**:  
 **return** Response({  
 **'data'**: -1,  
 **'role'**: -1,  
 **'status'**: 500  
 })  
 **elif** type == **'create-user'**:  
 **try**:  
 username = request.data[**'username'**]  
 password = request.data[**'password'**]  
 name = request.data[**'name'**]  
 diachi = request.data[**'diachi'**]  
 role = 2  
 user = {**'username'**: username, **'password'**: password, **'name'**: name, **'role'**: role, **'diachi'**: diachi}  
 serializer = User\_Serializer(data=user)  
 **if** serializer.is\_valid():  
 serializer.save()  
 **return** Response({  
 **'role'**: 1  
 })  
 **else**:  
 **return** Response({  
 **'role'**: -1  
 })  
 **except**:  
 **return** Response({  
 **'role'**: -1  
 })  
 **elif** type == **'edit-user'**:  
 **try**:  
 *# Get user* uuid = request.data[**'uuid'**]  
 user = User.objects.get(uuid=uuid)  
  
 *# Get Edit value* password = request.data[**'password'**]  
 name = request.data[**'name'**]  
 diachi = request.data[**'diachi'**]  
  
 *# Assign Value* **if** password != **""**:  
 user.password = password  
 **if** name != **""**:  
 user.name = name  
 **if** diachi != **""**:  
 user.diachi = diachi  
 user.save()  
 **return** Response({  
 **'status'**: 200  
 })  
 **except**:  
 **return** Response({  
 **'status'**: 500  
 })  
 **elif** type == **'delete-user'**:  
 **try**:  
 uuid = request.data[**'uuid'**]  
 user = User.objects.get(uuid=uuid)  
 user.delete()  
 **return** Response({  
 **'status'**: 200  
 })  
 **except**:  
 **return** Response({  
 **'status'**: 500  
 })  
 **except**:  
 **return** Response(500)

### 4.1.2. Hiển thị bản đồ người dùng di chuyển

**Mô tả:** Hàm này sẽ tạo ra bản đồ lịch sử di chuyển của người dùng và đánh dấu vị trí đó trên bản đồ.

**Chức năng:** Hiển thị bản đồ

import 'package:flutter/material.dart';  
import 'package:map4d\_map/map4d\_map.dart';  
import 'dart:async';  
  
class Map4d extends StatefulWidget {  
 Map4d({Key? key, required this.lat, required this.lng});  
  
 final lat;  
 final lng;  
  
 @override  
 \_Map4dState createState() => \_Map4dState(lat: lat, lng: lng);  
}  
  
class \_Map4dState extends State<Map4d> {  
 \_Map4dState({Key? key, required this.lat, required this.lng});  
  
 final lat;  
 final lng;  
  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Map4D Map',  
 home: Map4dSample(lat: lat, lng: lng),  
 );  
 }  
}  
  
class Map4dSample extends StatefulWidget {  
 Map4dSample({Key? key, required this.lat, required this.lng});  
  
 final lat;  
 final lng;  
  
 @override  
 \_Map4dSampleState createState() => \_Map4dSampleState();  
}  
  
class \_Map4dSampleState extends State<Map4dSample> {  
 Completer<MFMapViewController> \_controller = Completer();  
 bool \_is3dMode = false;  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 body: MFMapView(  
 initialCameraPosition: MFCameraPosition(  
 target: MFLatLng(double.*parse*(widget.lat), double.*parse*(widget.lng)), zoom: 16),  
 onMapCreated: (MFMapViewController controller) {  
 \_controller.complete(controller);  
 },  
 onPOITap: \_onPOITap,  
 markers: {  
 MFMarker(  
 consumeTapEvents: true,  
 markerId: MFMarkerId('marker\_id\_0'),  
 position: MFLatLng(double.*parse*(widget.lat), double.*parse*(widget.lng)),  
 anchor: Offset(0.5, 1.0),  
 infoWindow: MFInfoWindow(  
 snippet: "Snippet",  
 title: "Map4D",  
 anchor: const Offset(0.5, 0.0),  
 onTap: () {}),  
 zIndex: 1.0,  
 )  
 },  
 ),  
 floatingActionButton: FloatingActionButton(  
 onPressed: \_switch3dMode,  
 tooltip: '3D Mode',  
 child: Icon(Icons.*threed\_rotation*),  
 ),  
 );  
 }  
  
 void \_onPOITap(String placeId, String name, MFLatLng location) {  
 print('Tap on place: $placeId, name: $name, location: $location');  
 }  
  
 void \_switch3dMode() async {  
 final MFMapViewController controller = await \_controller.future;  
 \_is3dMode = !\_is3dMode;  
 controller.enable3DMode(\_is3dMode);  
 }  
}

### 4.1.3. Xem lịch sử di chuyển

**Mô tả:** Người dùng sau khi thêm vị trí người mình đã tới, có thể xem lại nhưng nơi mình đã tới ở chức năng xem lịch sử di chuyển trên bản đồ

**Chức năng:** Xem lịch sử di chuyển

import 'package:flutter/material.dart';  
import 'package:http/http.dart' as http;  
import 'dart:convert';  
  
import 'package:tracking\_location\_v3/templates/home/map.dart';  
  
class History extends StatefulWidget {  
 const History({Key? key, required this.uuid}) : super(key: key);  
 final uuid;  
  
 @override  
 \_HistoryState createState() => \_HistoryState(uuid: uuid);  
}  
  
class \_HistoryState extends State<History> {  
 \_HistoryState({Key? key, required this.uuid});  
  
 final uuid;  
 var url = 'http://10.0.2.2:1001';  
 var \_getJson = [];  
  
 void fetchGet() async {  
 try {  
 var queryParameters = {  
 'uuid': uuid,  
 'type': "get-history-by-uuid"  
 };  
 var uri = Uri.http(  
 '10.0.2.2:1001', "/tracking-location/history/api/", queryParameters);  
 final response = await http.get(uri);  
 final jsonData = jsonDecode(utf8.decode(response.bodyBytes)) as List;  
 setState(() {  
 \_getJson = jsonData;  
 });  
 } catch (err) {  
 print(err);  
 }  
 }  
  
 @override  
 void initState() {  
 super.initState();  
 fetchGet();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 fetchGet();  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text("NƠI ĐÃ ĐẾN"),  
 centerTitle: true,  
 ),  
 body: ListView.builder(  
 itemCount: \_getJson.length,  
 itemBuilder: (context, i) {  
 final get = \_getJson[i];  
 *// return Text(  
 // "\nNgày (date): ${get["date"]}\nVĩ độ (Latitude): ${get["lat"]}\nKinh độ (Longitude): ${get["lng"]}\n");* return Card(  
 child: Column(  
 children: [  
 ListTile(  
 title: Text("${get["date"]}"),  
 subtitle: Text("Kinh độ (Longitude): ${get["lng"]}"  
 "\nVĩ độ (Latitude): ${get["lat"]}"),  
 leading: IconButton(  
 icon: const Icon(Icons.*map*),  
 color: Colors.*blue*,  
  
 onPressed: ()  
 {  
 print("${get["lng"]}"  
 ",${get["lat"]}");  
 final lat = get["lat"];  
 final lng = get["lng"];  
  
 Navigator.*push*(context,  
 MaterialPageRoute(builder: (context) => Map4d(lat: lat, lng: lng,)));  
 },  
 ),  
 ),  
 Padding(padding: const EdgeInsets.all(3.0)),  
 ],  
 ),  
 );  
 },  
 ));  
 }  
}

### 4.1.4. Quản lý truy vết

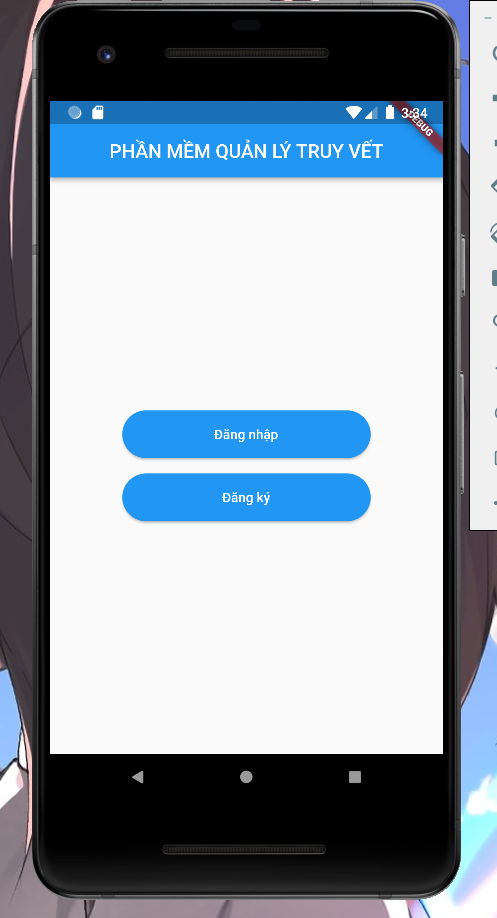
**Mô tả:** Quản lý truy vết, giúp quản lý được lịch sử di chuyển của tất cả người dùng trong hệ thống

**Chức năng:** Quản lý lịch sử di chuyển của người dùng

import 'package:flutter/material.dart';  
import 'package:http/http.dart' as http;  
import 'dart:convert';  
  
import 'package:tracking\_location\_v3/templates/home/history.dart';  
  
class QuanLy\_TruyVet extends StatefulWidget {  
 const QuanLy\_TruyVet({Key? key}) : super(key: key);  
  
 @override  
 \_QuanLy\_TruyVetState createState() => \_QuanLy\_TruyVetState();  
}  
  
class \_QuanLy\_TruyVetState extends State<QuanLy\_TruyVet> {  
 var url = 'http://10.0.2.2:1001/tracking-location/user/api/';  
  
 var \_getJson = [];  
  
 void fetchGet() async {  
 try {  
 final response = await http.get(Uri.parse(url));  
 final jsonData = jsonDecode(utf8.decode(response.bodyBytes)) as List;  
 setState(() {  
 \_getJson = jsonData;  
 });  
 } catch (err) {}  
 }  
  
 @override  
 void initState() {  
 super.initState();  
 fetchGet();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 fetchGet();  
 return Scaffold(  
 appBar: AppBar(  
 title: Text("QUẢN LÝ TRUY VẾT"),  
 centerTitle: true,  
 ),  
 body: ListView.builder(  
 itemCount: \_getJson.length,  
 itemBuilder: (context, i) {  
 final get = \_getJson[i];  
 return Card(  
 child: Column(  
 children: [  
 ListTile(  
 title: Text("${get["name"]}"),  
 subtitle:  
 Text("Tên tài khoản (Username): ${get["username"]}"  
 "\nĐịa chỉ (Address): ${get["diachi"]}"),  
 leading: IconButton(  
 icon: const Icon(Icons.remove\_red\_eye),  
 color: Colors.blue,  
 onPressed: () {  
 Navigator.push(  
 context,  
 MaterialPageRoute(  
 builder: (context) =>  
 History(uuid: get["uuid"])));  
 },  
 ),  
 ),  
 Padding(padding: const EdgeInsets.all(3.0)),  
 ],  
 ),  
 );  
 },  
 ));  
 }  
}

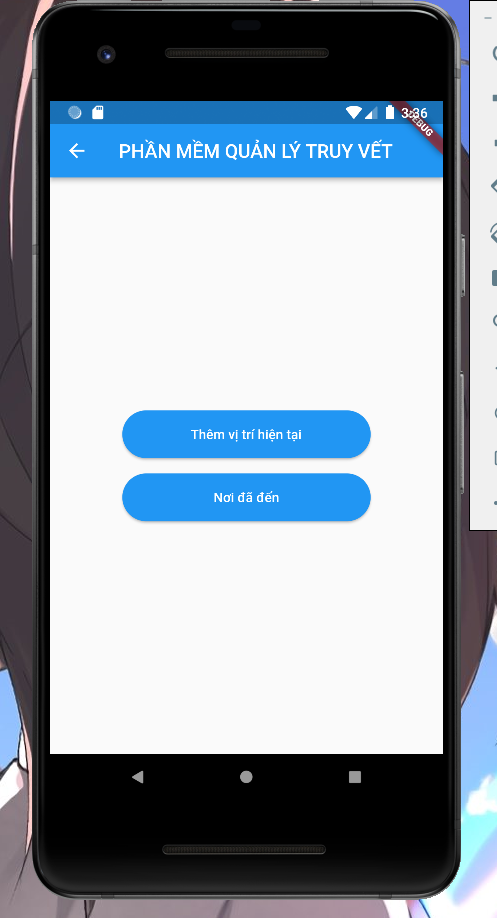
## 4.2. Kết quả lập trình

### 4.2.1 Giao diện Người dùng chưa đăng nhập



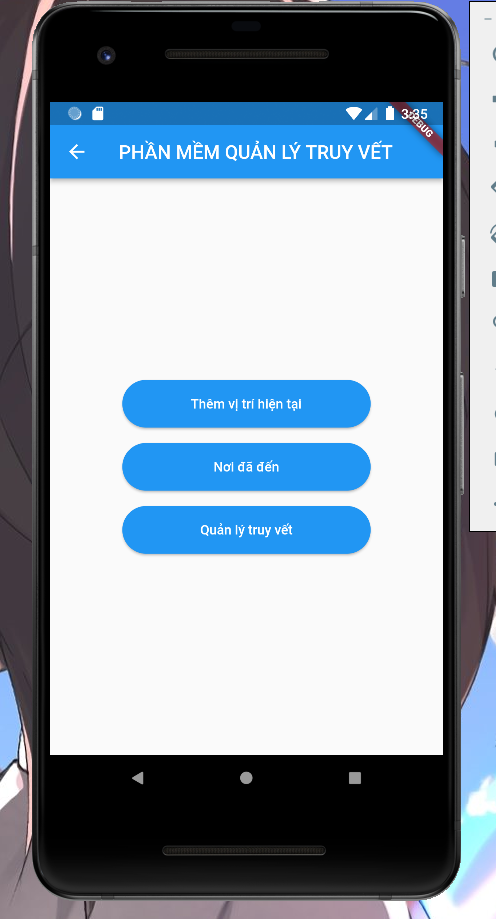
Hình 12: Giao diện người dùng chưa đăng nhập

### 4.2.2. Giao diện người dùng



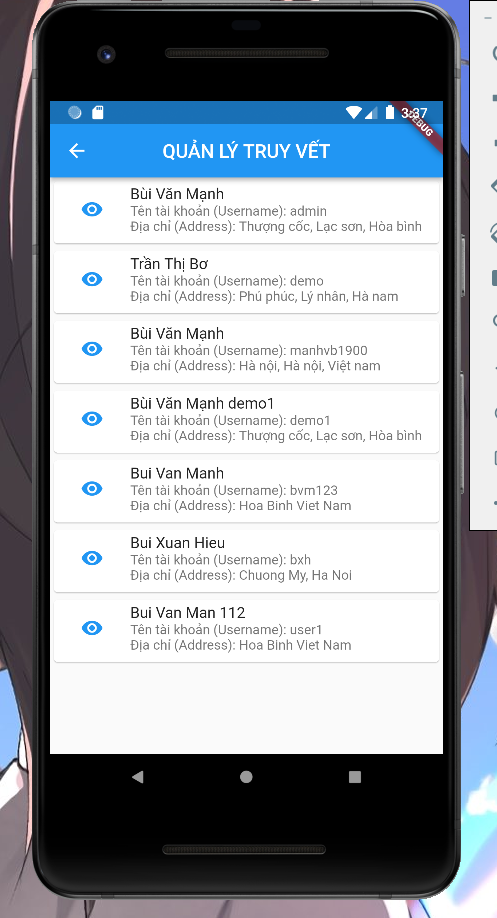
Hình 13: Giao diện người dùng

### 4.2.3. Giao diện quản trị viên



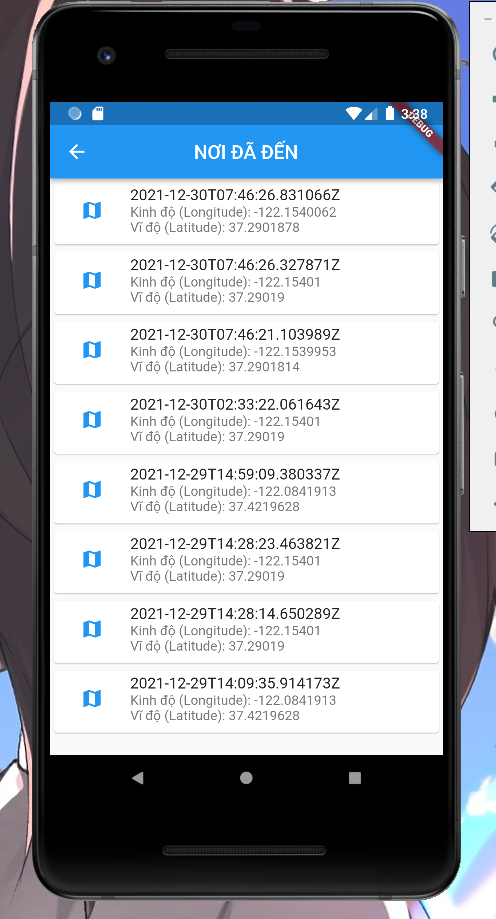
Hình 14: Giao diện quản trị viên

### 4.2.4. Giao diện quản lý truy vết

\ 

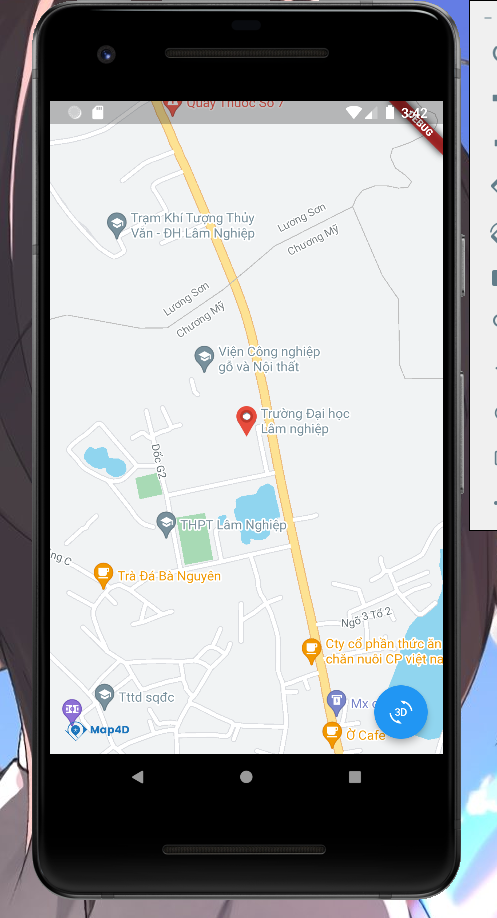
Hình 15: Giao diện quản lý truy vết

### 4.2.6. Giao diện lịch sử di chuyển



Hình 16: Giao diện lịch sử di chuyển

### 4.2.6. Giao diện xem lịch sử di chuyển trên bản đồ



Hình 17: Giao diện xem lịch sử di chuyển trên bản đồ

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 5.1. Kết luận

Như vậy, Trong quá trình xây dựng phần mềm quản lý truy vết, em đã đạt được những mục tiêu sau:

* Nâng cao được lập trình cũng như phát triển hệ thống.
* Xây dựng thành công phần mềm quản lý truy vết đơn giản hiệu quả.

Trong quá trình xây dựng hệ thống vì chưa có nhiều kinh nghiệm nên vẫn còn chưa thực sự hoàn thiện em mong thầy có thể góp ý và chỉ ra những thứ còn thiếu sót của em. Em trân thành cảm ơn!.

## 5.2. Hướng phát triển

Những hướng phát triển trong tương lai của em gồm có:

* Xây dựng hệ thống có thể đáp ứng được nhu cầu do cho nhiều người dùng hơn.
* Xây dựng hệ thống có độ bảo mật cao hơn.
* Xây dựng hệ thống có nhiều chức năng đáp ứng tính mới

**TÀI LIỆU THAM KHẢO**

[1]. ***.Thư viện Flutter, Dart*** https://pub.dev/

[2]. ***Python***  https://docs.python.org/3/