**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG TP.HCM**

**KHOA: HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**

****

**BÁO CÁO ĐỀ TÀI CUỐI KÌ**

**LẬP TRÌNH ỨNG DỤNG**

**“TÊN TIẾNG ANH CỦA BẠN LÀ GÌ”**

Giảng viên: **ThS. Nguyễn Thanh Truyền**

Sinh viên thực hiện: **Trần Xuân Phúc**

**Tào Gia Khánh**

**Nguyễn Quốc Khánh**

**Nguyễn Ngọc Lam**

**Nguyễn Ngọc Sơn Tuyền**

**Đinh Ngọc Huyền**

Lớp: **08\_ĐH\_TMĐT**

Khóa: **2019 - 2023**

***TP. Hồ Chí Minh, tháng 8 năm 2023***

**MỞ ĐẦU**

Trong thế giới ngày nay, sự giao thoa và gắn kết giữa các quốc gia đã khiến việc giao tiếp và hiểu biết trở nên vô cùng quan trọng. Tuy nhiên, rào cản về ngôn ngữ vẫn đôi khi gây trở ngại cho việc truyền tải thông tin và tương tác hiệu quả. Nhằm giải quyết vấn đề này, báo cáo này sẽ đưa ra một cuộc hành trình thú vị vào thế giới phát triển ứng dụng Flutter, tập trung vào việc xây dựng một ứng dụng độc đáo có khả năng chuyển đổi tên tiếng Việt sang tiếng Anh.

Mục tiêu của dự án là tận dụng sức mạnh của Flutter - một công cụ phát triển giao diện người dùng mã nguồn mở do Google phát triển - để xây dựng một ứng dụng thân thiện và linh hoạt, giải quyết những thách thức về chuyển đổi tên giữa hai ngôn ngữ tiếng Việt và tiếng Anh. Bằng cách cung cấp việc chuyển đổi tên một cách trơn tru và chính xác, ứng dụng của nhóm em nhằm hỗ trợ việc tương tác đa văn hóa một cách dễ dàng hơn và nối liền sự giao tiếp giữa hai ngôn ngữ này.

Trong suốt báo cáo này, nhóm em sẽ đào sâu vào kiến trúc của Flutter và tìm hiểu cách nó giúp nhà phát triển xây dựng ứng dụng đa nền tảng một cách hiệu quả. Thêm vào đó, nhóm em sẽ xem xét các phương pháp và thuật toán được sử dụng để giải quyết những khó khăn trong việc chuyển đổi tên, đảm bảo tính chính xác và tôn trọng văn hóa trong quá trình này.

Bây giờ, hãy bắt đầu hành trình thú vị này vào thế giới phát triển ứng dụng Flutter và khám phá cách chúng ta có thể tạo ra một công cụ mạnh mẽ, góp phần thúc đẩy giao tiếp đa ngôn ngữ và sự hiểu biết.

**LỜI CẢM ƠN**

Trước hết nhóm em xin gửi lời cảm ơn chân thành và sâu sắc đến Thầy Nguyễn Thanh Truyền đã dành thời gian, kiến thức và hỗ trợ tận tâm trong suốt quá trình thực hiện dự án này. Sự hướng dẫn tận tâm và tâm huyết của Thầy đã đóng góp không nhỏ vào thành công của dự án và giúp nhóm em đạt được những kết quả đáng kể.

Qua sự chỉ dạy và hướng dẫn của Thầy, nhóm chúng em đã được tiếp cận những kiến thức chuyên sâu về Flutter và phát triển ứng dụng đa nền tảng. Thầy đã tận tình giải đáp các thắc mắc và hỗ trợ chúng tôi vượt qua những khó khăn và thách thức trong quá trình thực hiện dự án.

Sự đồng hành cùng Thầy trong dự án này không chỉ giúp chúng em hoàn thiện ứng dụng chuyển đổi tên tiếng Việt sang tiếng Anh một cách thành công mà còn đem lại cho chúng em những kinh nghiệm quý báu trong lĩnh vực phát triển ứng dụng và công nghệ.

Một lần nữa, nhóm em xin chân thành cảm ơn Thầy Nguyễn Thanh Truyền đã dành tâm huyết và kiến thức giúp đỡ chúng em trong dự án này. Sự hỗ trợ của Thầy đã góp phần không nhỏ vào thành công của dự án và là nguồn động lực to lớn để chúng em tiếp tục vươn tới những thành công mới trong tương lai.

Xin chân thành cảm ơn và kính chúc Thầy sức khỏe, hạnh phúc và thành công trong công việc giảng dạy và nghiên cứu.

**NHẬN XÉT**

**CÔNG VIỆC CỦA CÁC THÀNH VIÊN TRONG NHÓM 1**

|  |  |
| --- | --- |
| **Tên thành viên** | **Công việc thực hiện** |
| Trần Xuân Phúc | Thiết kế giao diện, tổng hợp báo cáo. |
| Tào Gia Khánh | Thiết kế giao diện. |
| Nguyễn Quốc Khánh | Xử lý code logic chuyển tên tiếng Việt sang tiếng Anh. |
| Nguyễn Ngọc Sơn Tuyền | Tìm hiểu lý thuyết về Container, Textfeild, Button. |
| Nguyễn Ngọc Lam | Tìm hiểu lý thuyết về Center, MediaQuery. |
| Đinh Ngọc Huyền | Tìm hiểu lý thuyết về ListView TabbleData, Grid. |

**MỤC LỤC**

[**CHƯƠNG 1: TỔNG QUAN** 1](#_Toc142634569)

[**1.1** **Lý do chọn đề tài** 1](#_Toc142634570)

[**1.2** **Tính thực tiễn của đề tài** 1](#_Toc142634571)

[**1.3** **Phạm vi đề tài** 1](#_Toc142634572)

[**1.4** **Cấu trúc bài báo cáo** 1](#_Toc142634573)

[**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT** 2](#_Toc142634574)

[**2.1** **Tìm hiểu về Container** 2](#_Toc142634575)

[**2.1.1** **Định nghĩa** 2](#_Toc142634576)

[**2.1.2** **Thuộc tính** 2](#_Toc142634577)

[**2.1.3** **Sử dụng Container trong Flutter** 2](#_Toc142634578)

[**2.1.4** **Kết luận** 3](#_Toc142634579)

[**2.2** **Tìm hiểu về TextField** 4](#_Toc142634580)

[**2.2.1** **Định nghĩa** 4](#_Toc142634581)

[**2.2.2** **Thuộc tính** 4](#_Toc142634582)

[**2.2.3** **Sử dụng TextField trong Flutter** 4](#_Toc142634583)

[**2.2.4** **Kết luận** 5](#_Toc142634584)

[**2.3** **Tìm hiểu về Button** 6](#_Toc142634585)

[**2.3.1** **Định nghĩa** 6](#_Toc142634586)

[**2.3.2** **Sử dụng Button trong Flutter** 6](#_Toc142634587)

[**2.4** **Tìm hiểu về Center** 8](#_Toc142634588)

[**2.5** **Tìm hiểu về MediaQuery** 11](#_Toc142634589)

[**2.6** **Tìm hiểu về ListView** 15](#_Toc142634590)

[**2.7** **Tìm hiều về TableData** 19](#_Toc142634591)

[**2.8** **Tìm hiều về Grid** 21](#_Toc142634592)

[**CHƯƠNG 3: MÔ TẢ ỨNG DỤNG** 23](#_Toc142634593)

[**3.1** **Mô tả giao diện** 23](#_Toc142634594)

[**3.2** **Mô tả thao tác** 23](#_Toc142634595)

[**CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN** 26](#_Toc142634596)

[**4.1** **Kết luận** 26](#_Toc142634597)

[**4.2** **Hướng phát triển** 26](#_Toc142634598)

[**PHỤ LỤC** 27](#_Toc142634599)

[**TÀI LIỆU THAM KHẢO** 29](#_Toc142634600)

**DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH**

[HÌNH 2. 1. Container trong Flutter 3](#_Toc142604749)

[HÌNH 2. 2. TextField trong Flutter 5](#_Toc142604750)

[HÌNH 2. 3. ElevatedButton trong Flutter 7](#_Toc142604751)

[HÌNH 2. 4. Widget Center trong Flutter 8](#_Toc142604752)

[HÌNH 2. 5. Widget Center có chứa thuộc tính widthFactor 9](#_Toc142604753)

[HÌNH 2. 6. Widget Center có chứa thuộc tính heightFactor 10](#_Toc142604754)

[HÌNH 2. 7. Widget Center có chứa Icon 11](#_Toc142604755)

[HÌNH 2. 8. ListView trong Flutter 16](#_Toc142604756)

[HÌNH 2. 9. ListView.builder trong Flutter 17](#_Toc142604757)

[HÌNH 2. 10. ListView.separated trong Flutter 18](#_Toc142604758)

[HÌNH 2. 11. Giao diện khi sử dụng DataTable trong Flutter 20](#_Toc142604759)

[HÌNH 2. 12. DataTable với các màu thay thế trên mỗi hàng 20](#_Toc142604760)

[HÌNH 2. 13. GridView với hai cột 22](#_Toc142604761)

[HÌNH 2. 14. ListTile trong GridView hoặc ListView 22](#_Toc142604762)

[HÌNH 3. 1. Giao diện trang nhập thông tin 23](#_Toc142604706)

[HÌNH 3. 2. Giao diện trang nhập thông tin báo lỗi 23](#_Toc142604707)

[HÌNH 3. 3. Giao diện trang lịch sử khi chưa có dữ liệu 24](#_Toc142604708)

[HÌNH 3. 4. Trang kết quả dịch tên tiếng việt sang tên tiếng anh 24](#_Toc142604709)

[HÌNH 3. 5. Trang giao diện lịch sử khi có dữ liệu 25](#_Toc142604710)

[HÌNH 3. 6. Giao diện khi xóa dữ liệu 25](#_Toc142604711)

# **CHƯƠNG 1: TỔNG QUAN**

* 1. **Lý do chọn đề tài**

Việc lựa chọn đề tài "Lập trình ứng dụng 'Tên tiếng Anh của bạn là gì'" được thực hiện dựa trên nhận thấy về sự gia tăng của ứng dụng di động trong cuộc sống hàng ngày. Với sự phát triển vượt bậc của công nghệ thông tin, ứng dụng di động đã trở thành một phần quan trọng của cuộc sống hiện đại, cung cấp nhiều tiện ích và giải pháp cho người dùng. Do đó, việc nghiên cứu và phát triển một ứng dụng đơn giản nhưng thú vị như "Tên tiếng Anh của bạn là gì" là một cách tốt để giới thiệu người dùng vào quá trình lập trình và trải nghiệm sự hứng thú từ việc tạo ra một ứng dụng thực tế**.**

* 1. **Tính thực tiễn của đề tài**

Ứng dụng di động đã thay đổi cách chúng ta giao tiếp, làm việc và giải trí hàng ngày. Việc lập trình ứng dụng không chỉ mang lại cơ hội tạo ra những sản phẩm hữu ích mà còn giúp người dùng phát triển kỹ năng lập trình và logic. Việc phát triển một ứng dụng nhỏ như "Tên tiếng Anh của bạn là gì" một ứng dụng đơn giản với tính giải trí như việc đoán tên tiếng Anh của người dùng cũng có thể thu hút sự quan tâm và tham gia của nhiều người, đặc biệt là các bạn trẻ.

* 1. **Phạm vi đề tài**

Phạm vi của đề tài "Lập trình ứng dụng 'Tên tiếng Anh của bạn là gì'" sẽ tập trung vào việc phát triển một ứng dụng di động trên nền tảng Android và iOS. Ứng dụng sẽ cho phép người dùng nhập tên của họ và sau đó hiển thị một thông điệp thú vị dựa trên việc đoán tên tiếng Anh của người dùng. Ứng dụng cũng sẽ cung cấp một giao diện đơn giản và thân thiện với người dùng, giúp họ dễ dàng tương tác và trải nghiệm.

* 1. **Cấu trúc bài báo cáo**

Chương 1: Tổng Quan

Chương 2: Cơ sở lý thuyết

Chương 3: Mô tả ứng dụng

Chương 4: Kết luận và Hướng Phát Triển.

# **CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

* 1. **Tìm hiểu về Container**
     1. **Định nghĩa**

Trong Flutter, Container là một widget dùng để tạo và điều khiển các hộp chứa (box) có hình dạng và kích thước tùy ý. Nó là một thành phần cơ bản được sử dụng rộng rãi trong xây dựng giao diện người dùng với mục tiêu tạo ra các thành phần giao diện linh hoạt và đẹp mắt.

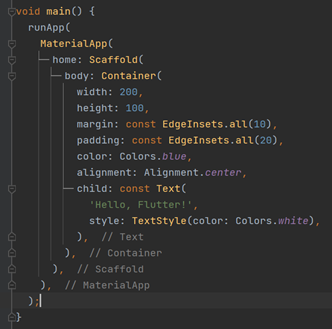
* + 1. **Thuộc tính**

Container có một loạt các thuộc tính quan trọng để điều chỉnh hình dạng, kích thước và vị trí của nó:

* **width** và **height**: Đây là các thuộc tính để xác định chiều rộng và chiều cao của Container. Nếu không xác định kích thước, Container sẽ mở rộng dọc theo nội dung bên trong.
* **color** và **decoration**: Thuộc tính color cho phép đặt màu nền cho Container bằng cách sử dụng Color hoặc Gradient. Nếu chúng ta muốn tạo nền phức tạp hơn, chúng ta có thể sử dụng thuộc tính decoration để tạo các hiệu ứng nâng cao như gradient, hình ảnh nền, hay viền.
* **padding** và **margin**: Thuộc tính padding cho phép chúng ta đặt lề bên trong Container, là khoảng cách giữa nội dung và viền của nó. Thuộc tính margin, ngược lại, xác định khoảng cách giữa Container và các phần tử lân cận trong giao diện.
* **alignment**: Thuộc tính này điều chỉnh vị trí của Container trong cha của nó. Bạn có thể căn chỉnh theo các giá trị như topLeft, topCenter, topRight,…
* **border**: Thuộc tính này cho phép chúng ta thêm viền cho Container, đồng thời xác định màu sắc và chiều rộng của viền. Chúng ta cũng có thể đặt góc bo tròn cho Container sử dụng thuộc tính borderRadius.
  + 1. **Sử dụng Container trong Flutter**
* **Xây dựng giao diện đơn giản:** Container là một trong những widget cơ bản nhất, dễ sử dụng và có thể sử dụng riêng lẻ hoặc kết hợp với các widget khác để xây dựng các giao diện đơn giản như hộp chứa (container), hình ảnh, văn bản, nút, và nhiều thành phần khác.
* **Tạo các thành phần phức tạp hơn:** Container có khả năng chồng lớp (stacking) và lồng vào nhau, cho phép chúng ta xây dựng các giao diện phức tạp hơn bằng cách sắp xếp và kết hợp nhiều Container với các thuộc tính và vị trí khác nhau.
* **Kiểm soát vị trí và căn chỉnh:** Container cho phép chúng ta kiểm soát vị trí của nó trong giao diện và căn chỉnh nội dung bên trong một cách linh hoạt, giúp tạo ra các giao diện gọn gàng và chính xác.

**Ví dụ về sử dụng Container**

Sử dụng Container trong Flutter để tạo một hộp chứa (container) với màu nền xanh và chữ màu trắng:

 A white background with black dots

Description automatically generated

HÌNH 2. 1. Container trong Flutter

* + 1. **Kết luận**

Container là một trong những widget quan trọng và mạnh mẽ nhất trong Flutter, cho phép tạo và kiểm soát các thành phần giao diện với các thuộc tính về hình dạng, kích thước, màu sắc và vị trí. Bằng cách sử dụng Container, chúng ta có thể tạo các giao diện phức tạp, đẹp mắt và linh hoạt cho ứng dụng của mình, đồng thời tối ưu hóa trải nghiệm người dùng.

* 1. **Tìm hiểu về TextField**
     1. **Định nghĩa**

TextField là một widget trong Flutter cho phép người dùng nhập dữ liệu văn bản thông qua bàn phím của thiết bị. Nó là một thành phần cơ bản được sử dụng rộng rãi trong việc xây dựng các biểu mẫu (form) và các tương tác liên quan đến dữ liệu đầu vào từ người dùng.

* + 1. **Thuộc tính**

TextField có các thuộc tính quan trọng để điều chỉnh cách nó hoạt động và hiển thị:

* **controller:** Đây là một đối tượng TextEditingController để liên kết và quản lý dữ liệu đầu vào của TextField.
* **decoration:** Thuộc tính này cho phép chúng ta tùy chỉnh hình dạng và giao diện của TextField, bao gồm cả tiêu đề (label), viền, màu sắc và hình ảnh nền.
* **obscureText:** Nếu được đặt thành true, dữ liệu đầu vào của TextField sẽ được ẩn bằng các dấu chấm hoặc dấu sao. Thường được sử dụng để nhập mật khẩu.
  + 1. **Sử dụng TextField trong Flutter**
* **Tạo trường nhập liệu (input field):** Để tạo một trường nhập liệu, chúng ta cần sử dụng widget TextField và gán một đối tượng TextEditingController cho nó. Điều này cho phép chúng ta lấy giá trị của TextField sau khi người dùng nhập dữ liệu.
* **Tùy chỉnh hình dạng và giao diện:** TextField có khả năng tùy chỉnh hình dạng và giao diện thông qua thuộc tính decoration. Chúng ta có thể đặt màu sắc, viền, tiêu đề, hình ảnh nền và nhiều thuộc tính khác để tạo một trường nhập liệu phù hợp với thiết kế của ứng dụng.

**Ví dụ về sử dụng TextField**

Sử dụng TextField trong Flutter để tạo một trường nhập liệu cho người dùng nhập tên:

 A screenshot of a keyboard

Description automatically generated

HÌNH 2. 2. TextField trong Flutter

* + 1. **Kết luận**

TextField là một widget cơ bản và quan trọng trong Flutter, cho phép chúng ta tạo các trường nhập liệu để người dùng nhập dữ liệu văn bản. Bằng cách sử dụng các thuộc tính như controller và decoration, chúng ta có thể tùy chỉnh hình dạng và giao diện của TextField để phù hợp với thiết kế và nhu cầu của ứng dụng. TextField là một công cụ hữu ích giúp tối ưu hóa tương tác giữa ứng dụng và người dùng, đồng thời cung cấp khả năng thu thập và sử dụng dữ liệu người dùng một cách dễ dàng.

* 1. **Tìm hiểu về Button**
     1. **Định nghĩa**

Trong Flutter, Button là một widget dùng để tạo các nút và xử lý các tương tác từ người dùng, như nhấn, chạm, hoặc kéo thả. Nó là một thành phần cơ bản và quan trọng trong việc xây dựng các giao diện tương tác và điều hướng trong ứng dụng.

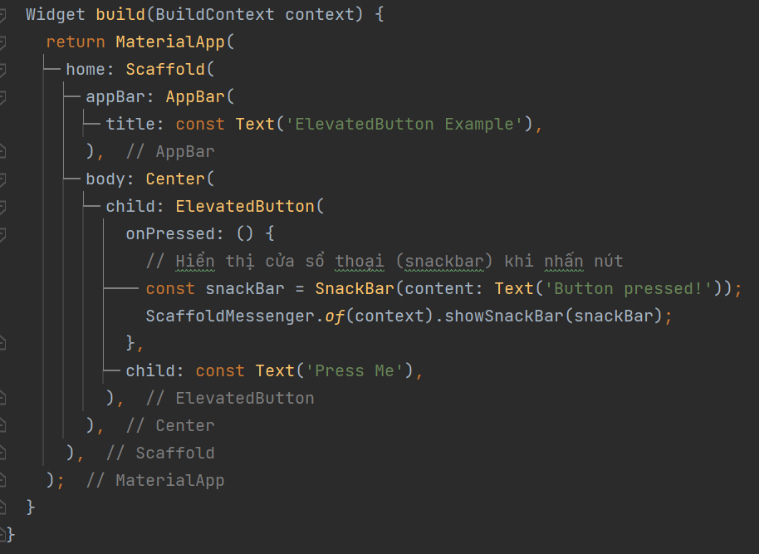
**Các loại Button**

Flutter cung cấp nhiều loại Button khác nhau, bao gồm:

* **ElevatedButton:** Nút có hình dạng nổi bật với màu nền và chữ màu trắng.
* **TextButton:** Nút không có đường viền, thường có màu sắc trong suốt với chữ màu tùy chỉnh.
* **IconButton:** Nút có hình ảnh hoặc biểu tượng, thường được sử dụng trong AppBar hoặc các vị trí có ít không gian.
* **FloatingActionButton:** Nút tròn nổi bật thường được đặt ở dưới cùng màn hình để thực hiện các hành động quan trọng.
  + 1. **Sử dụng Button trong Flutter**
* **Tạo Button:** Để tạo một Button trong Flutter, chúng ta chỉ cần sử dụng một trong các widget RaisedButton, FlatButton, IconButton, hoặc FloatingActionButton tùy theo nhu cầu và thiết kế của ứng dụng.
* **Xử lý sự kiện:** Để xử lý sự kiện khi người dùng nhấn vào Button, chúng ta cần sử dụng thuộc tính onPressed và truyền một hàm xử lý sự kiện vào đó.

**Ví dụ về sử dụng Button**

Sử dụng ElevatedButton trong Flutter:

 A blue rectangle with white text

Description automatically generated

HÌNH 2. 3. ElevatedButton trong Flutter

* + 1. **Kết luận**

Button là một widget cơ bản và quan trọng trong Flutter, cho phép chúng ta tạo các nút và xử lý các tương tác từ người dùng. Bằng cách sử dụng các loại Button như RaisedButton, FlatButton, IconButton và FloatingActionButton, chúng ta có thể tùy chỉnh hình dạng, màu sắc và vị trí của các nút trong ứng dụng của mình. Button giúp tối ưu hóa tương tác giữa người dùng và ứng dụng, làm cho ứng dụng trở nên trực quan và dễ sử dụng hơn.

* 1. **Tìm hiểu về Center**

Một widget giúp căn giữa nội dung của nó bên trong chính nó.

Center và Align khá giống nhau, chúng chỉ có duy nhất một widget con,

nhưng Align cho phép tùy biến vị trí của widget con bên trong nó.

* Center Contructors(Hàm tạo)

const Center(

{Key? key,

double? widthFactor,

double? heightFactor,

Widget? child}

)

Hàm tạo này được đánh dấu là const, có nghĩa rằng widget Center được tạo bằng cách sử dụng hàm tạo này sẽ không thay đổi sau khi được khởi tạo.

Ví dụ:

Center(

child: Icon (

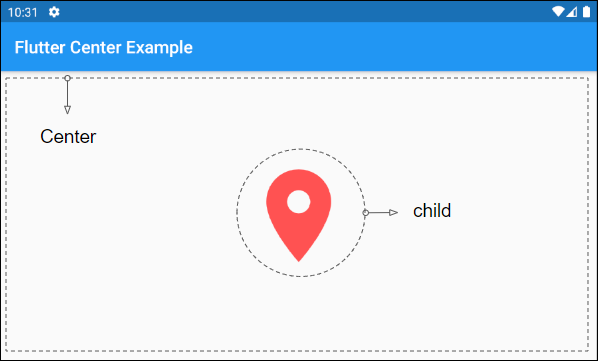
Icons.place,

size: 128,

color: Colors.redAccent

)

)



HÌNH 2. 4. Widget Center trong Flutter

* Các tham số của hàm tạo bao gồm:
* **key**: Định danh duy nhất của widget, giúp Flutter xác định các widget trong cây widget khi thay đổi cấu trúc.
* **widthFactor**: là một hệ số được sử dụng để tính toán chiều rộng của Center dựa trên chiều rộng của child (Con của nó). Nếu **widthFactor** không null chiều rộng của **Center** bằng chiều rộng của **child** nhân với hệ số này. Nếu **widthFactor** không được chỉ định chiều rộng của Center sẽ lớn nhất có thể. Ví dụ:

Center(

child: ElevatedButton (

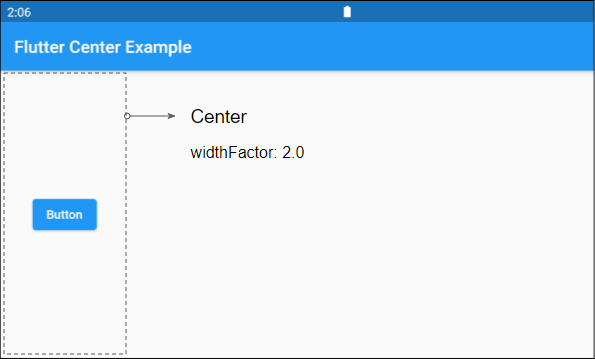
child: Text("Button"),

onPressed: () {}

),

widthFactor: 2.0

)



HÌNH 2. 5. Widget Center có chứa thuộc tính widthFactor

* **heightFactor**: là một hệ số được sử dụng để tính toán chiều cao của **Center** dựa trên chiều cao của **child** (Con của nó). Nếu **heightFactor** không null chiều cao của **Cente**r bằng chiều cao của **child** nhân với hệ số này. Nếu **heightFactor** không được chỉ định chiều cao của **Center** sẽ lớn nhất có thể. Ví dụ:

Center(

child: ElevatedButton (

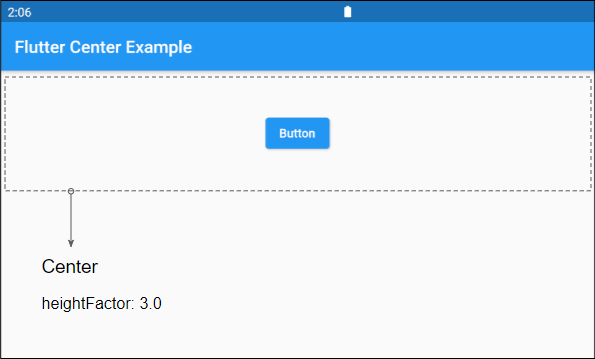
child: Text("Button"),

onPressed: () {}

),

heightFactor: 3.0

)



HÌNH 2. 6. Widget Center có chứa thuộc tính heightFactor

* **child:** Đây là widget con(duy nhất) sẽ được căn giữa bên trong **Center**, trong một số tình huống sử dụng nó có thể là đối tượng **Row**, **Column** hoặc **Stack** để có thể chứa được nhiều widget khác.Ví dụ với child là một đối tượng Row, nó có thể chứa nhiều widget con trên một hàng.

Center (

child: Row (

mainAxisAlignment: MainAxisAlignment.center,

children: [

Icon (

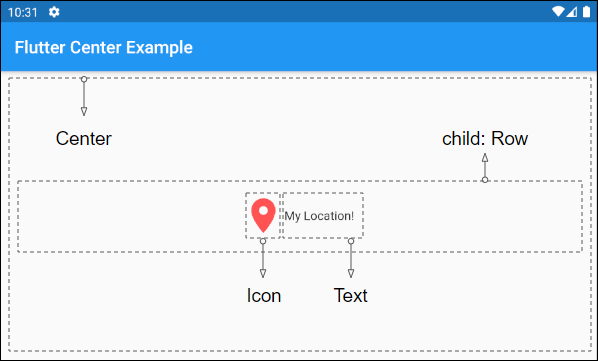
Icons.place,

size: 48,

color: Colors.redAccent

),

Text("My Location!") ], ) )



HÌNH 2. 7. Widget Center có chứa Icon

* 1. **Tìm hiểu về MediaQuery**

**MediaQuery** là một lớp trong Flutter cung cấp thông tin về kích thước màn hình hiện tại, hướng màn hình và các thông tin liên quan khác về phương tiện. Nó cho phép bạn xây dựng giao diện người dùng linh hoạt có thể thích ứng với các kích thước và hướng màn hình khác nhau của thiết bị.

* Contructors(Hàm tạo)

MediaQuery({Key? key, required MediaQueryData data, required Widget child})

//Tạo một widget cung cấp `MediaQueryData` cho các widget con của nó.

const MediaQuery.removePadding({Key? key, required BuildContext context, bool removeLeft = false, bool removeTop = false, bool removeRight = false, bool removeBottom = false, required Widget child})

Tạo một `MediaQuery` mới kế thừa từ `MediaQuery` hiện tại từ context đã cho, nhưng loại bỏ các giá trị padding đã chỉ định.

- `key`: Khóa duy nhất cho widget.

- `context`: BuildContext của widget hiện tại.

- `removeLeft`: Xác định liệu có loại bỏ giá trị padding bên trái hay không. Mặc định là `false`.

- `removeTop`: Xác định liệu có loại bỏ giá trị padding phía trên hay không. Mặc định là `false`.

- `removeRight`: Xác định liệu có loại bỏ giá trị padding bên phải hay không. Mặc định là `false`.

- `removeBottom`: Xác định liệu có loại bỏ giá trị padding phía dưới hay không. Mặc định là `false`.

- `child`: Widget con được bao bọc bởi `MediaQuery`.

factory MediaQuery.removeViewInsets({Key? key, required BuildContext context, bool removeLeft = false, bool removeTop = false, bool removeRight = false, bool removeBottom = false, required Widget child})

Tạo một `MediaQuery` mới kế thừa từ `MediaQuery` hiện tại từ context đã cho, nhưng loại bỏ các giá trị view insets đã chỉ định.

- `key`: Khóa duy nhất cho widget.

- `context`: BuildContext của widget hiện tại.

- `removeLeft`: Xác định liệu có loại bỏ giá trị view insets bên trái hay không. Mặc định là `false`.

- `removeTop`: Xác định liệu có loại bỏ giá trị view insets phía trên hay không. Mặc định là `false`.

- `removeRight`: Xác định liệu có loại bỏ giá trị view insets bên phải hay không. Mặc định là `false`.

- `removeBottom`: Xác định liệu có loại bỏ giá trị view insets phía dưới hay không. Mặc định là `false`.

- `child`: Widget con được bao bọc bởi `MediaQuery`.

factory MediaQuery.removeViewPadding({Key? key, required BuildContext context, bool removeLeft = false, bool removeTop = false, bool removeRight = false, bool removeBottom = false, required Widget child})

Tạo một `MediaQuery` mới kế thừa từ `MediaQuery` hiện tại từ context đã cho, nhưng loại bỏ các giá trị view padding đã chỉ định.

- `key`: Khóa duy nhất cho widget.

- `context`: BuildContext của widget hiện tại.

- `removeLeft`: Xác định liệu có loại bỏ giá trị view padding bên trái hay không. Mặc định là `false`.

- `removeTop`: Xác định liệu có loại bỏ giá trị view padding phía trên hay không. Mặc định là `false`.

- `removeRight`: Xác định liệu có loại bỏ giá trị view padding bên phải hay không. Mặc định là `false`.

- `removeBottom`: Xác định liệu có loại bỏ giá trị view padding phía dưới hay không. Mặc định là `false`.

- `child`: Widget con được bao bọc bởi `MediaQuery`.

Cách sử dụng phổ biến nhất của **MediaQuery** là trong phương thức build của widget. Bằng cách sử dụng phương thức tĩnh **MediaQuery.of(context),** ta có thể truy cập **MediaQueryData** hiện tại, chứa các thông tin về thiết bị. Ví dụ:

Widget build(BuildContext context) {

// Truy cập MediaQueryData

MediaQueryData mediaQueryData = MediaQuery.of(context);

// Lấy kích thước màn hình

Size screenSize = mediaQueryData.size;

// Lấy hướng màn hình của thiết bị

Orientation orientation = mediaQueryData.orientation;

// Lấy tỷ lệ điểm ảnh của thiết bị

double pixelRatio = mediaQueryData.devicePixelRatio;

// ... bạn có thể truy cập nhiều thuộc tính khác từ lớp MediaQueryData

return YourWidget(); }

Với thông tin thu được từ **MediaQuery**, ta có thể điều chỉnh các thành phần giao diện người dùng của mình một cách linh hoạt dựa trên kích thước màn hình hiện có. Ví dụ, có thể đặt kích thước hoặc bố cục của một số widget dựa trên kích thước màn hình và hướng màn hình.

Widget build(BuildContext context) {

MediaQueryData mediaQueryData = MediaQuery.of(context);

// Kiểm tra hướng màn hình và điều chỉnh giao diện theo đó

if (mediaQueryData.orientation == Orientation.portrait) {

// Điều chỉnh cho hướng dọc

return YourPortraitUI();

} else {

// Điều chỉnh cho hướng ngang

return YourLandscapeUI(); } }

Chúng ta có thể truy cập **MediaQueryData** bất cứ nơi nào trong ứng dụng của mình miễn là có thể truy cập **BuildContext.** Điều này có nghĩa là chúng ta có thể sử dụng nó trong một widget có trạng thái hoặc không có trạng thái, trong quá trình tính toán bố cục, hoặc thậm chí trong các widget tùy chỉnh.

**Lưu ý:** Nếu chúng ta đang sử dụng một widget con mà thay đổi giao diện thường xuyên, nó có thể hiệu quả hơn nếu trích xuất **MediaQueryData** trong widget cha và truyền nó xuống như một đối số để tránh việc tái xây dựng không cần thiết.

* 1. **Tìm hiểu về ListView**

ListView là tiện ích cuộn được sử dụng phổ biến nhất. Nó hiển thị các phần tử con của nó lần lượt theo hướng cuộn. Trong trục chéo, các phần tử con được yêu cầu điền vào ListView .

Có bốn tùy chọn để xây dựng một ListView :

* Hàm tạo mặc định lấy một Danh sách <Widget> rõ ràng của các phần tử con. Phương thức khởi tạo này phù hợp với dạng xem danh sách có ít phần tử con vì việc xây dựng Danh sách yêu cầu thực hiện công việc cho mọi phần tử con có thể được hiển thị trong dạng xem danh sách thay vì chỉ những phần tử con thực sự hiển thị.
* Hàm tạo ListView.builder lấy một IndexedWidgetBuilder , hàm tạo phần tử con theo yêu cầu. Hàm tạo này phù hợp với dạng xem danh sách có số lượng phần tử con lớn (hoặc vô hạn) vì trình tạo chỉ được gọi cho những phần tử con thực sự hiển thị.
* Hàm tạo ListView.separated nhận hai chỉ số IndexedWidgetBuilder s: itemBuilderxây dựng các mục con theo yêu cầu và separatorBuilder tương tự xây dựng các phần tử con của dấu phân cách xuất hiện ở giữa các mục con. Hàm tạo này phù hợp với dạng xem danh sách có số lượng phần tử con cố định.
* Hàm tạo ListView.custom nhận SliverChildDelegate , cung cấp khả năng tùy chỉnh các khía cạnh bổ sung của mô hình con. Ví dụ: SliverChildDelegate có thể kiểm soát thuật toán được sử dụng để ước tính kích thước của trẻ em không thực sự nhìn thấy được.

Ví dụ này sử dụng hàm tạo mặc định cho ListView nhận một List<Widget> rõ ràng của các phần tử con. Các phần tử con của ListView này được tạo thành từ các Container có Văn bản .

ListView(

padding: const EdgeInsets.all(8),

children: <Widget>[

Container(

height: 50,

color: Colors.amber[600],

child: const Center(child: Text('Entry A')),

),

Container(

height: 50,

color: Colors.amber[500],

child: const Center(child: Text('Entry B')),

),

Container(

height: 50,

color: Colors.amber[100],

child: const Center(child: Text('Entry C')), ), ], )

A screenshot of a computer screen

Description automatically generated

HÌNH 2. 8. ListView trong Flutter

Ví dụ này phản chiếu ví dụ trước, tạo cùng một danh sách bằng cách sử dụng hàm tạo ListView.builder .

final List<String> entries = <String>['A', 'B', 'C'];

final List<int> colorCodes = <int>[600, 500, 100];

Widget build(BuildContext context) {

return ListView.builder(

padding: const EdgeInsets.all(8),

itemCount: entries.length,

itemBuilder: (BuildContext context, int index) {

return Container(

height: 50,

color: Colors.amber[colorCodes[index]],

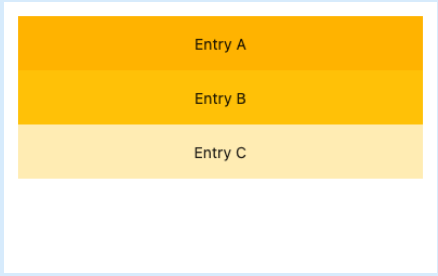
child: Center(child: Text('Entry ${entries[index]}')),

);

}

);

}



HÌNH 2. 9. ListView.builder trong Flutter

Ví dụ này tiếp tục xây dựng từ những ví dụ trước của chúng tôi, tạo một danh sách tương tự bằng cách sử dụng ListView.separated . Ở đây, Dải phân cách được sử dụng làm dấu phân cách.

final List<String> entries = <String>['A', 'B', 'C'];

final List<int> colorCodes = <int>[600, 500, 100];

Widget build(BuildContext context) {

return ListView.separated(

padding: const EdgeInsets.all(8),

itemCount: entries.length,

itemBuilder: (BuildContext context, int index) {

return Container(

height: 50,

color: Colors.amber[colorCodes[index]],

child: Center(child: Text('Entry ${entries[index]}')),

);

},

separatorBuilder: (BuildContext context, int index) => const Divider(),

); }

Ảnh có chứa văn bản, ảnh chụp màn hình, màu vàng, Phông chữ

Mô tả được tạo tự động

HÌNH 2. 10. ListView.separated trong Flutter

* 1. **Tìm hiều về TableData**
* Hiển thị dữ liệu trong một bảng rất tốn kém, bởi vì để sắp xếp bảng, tất cả dữ liệu phải được đo hai lần, một lần để thương lượng các kích thước sẽ sử dụng cho mỗi cột và một lần để thực sự sắp xếp bảng dựa trên kết quả thương lượng.
* Vì lý do này, nếu bạn có nhiều dữ liệu (giả sử hơn chục hàng với một tá cột, mặc dù giới hạn chính xác phụ thuộc vào thiết bị đích), bạn nên sử dụng PaginatedDataTable tự động chia dữ liệu thành nhiều trang.
* Mẫu này cho biết cách hiển thị một DataTable có ba cột: tên, tuổi và vai trò. Các cột được xác định bởi ba đối tượng DataColumn . Bảng chứa ba hàng dữ liệu cho ba người dùng ví dụ, dữ liệu được xác định bởi ba đối tượng DataRow.

DataColumn(

label: Expanded(

child: Text( 'Name', style: TextStyle(fontStyle: FontStyle.italic), ),),

),

DataColumn(

label: Expanded(

child: Text('Age', style: TextStyle(fontStyle: FontStyle.italic),),),

),

DataColumn(

label: Expanded(

child: Text('Role', style: TextStyle(fontStyle: FontStyle.italic),),),

),

rows: const <DataRow>[

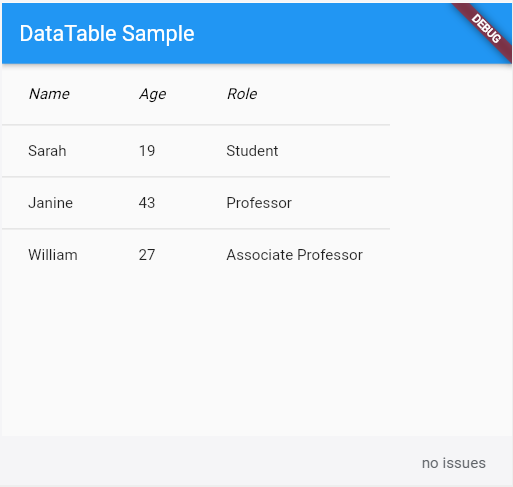
DataRow(

cells: <DataCell>[

DataCell(Text('Sarah')),

DataCell(Text('19')),

DataCell(Text('Student')), ], ),



HÌNH 2. 11. Giao diện khi sử dụng DataTable trong Flutter

Mẫu này cho biết cách hiển thị một DataTable với các màu thay thế trên mỗi hàng và một màu tùy chỉnh khi hàng được chọn.

columns: const <DataColumn>[

DataColumn(

label: Text('Number'), ), ],

rows: List<DataRow>.generate( numItems,

(int index) => DataRow(

color: MaterialStateProperty.resolveWith<Color?>(

(Set<MaterialState> states) {

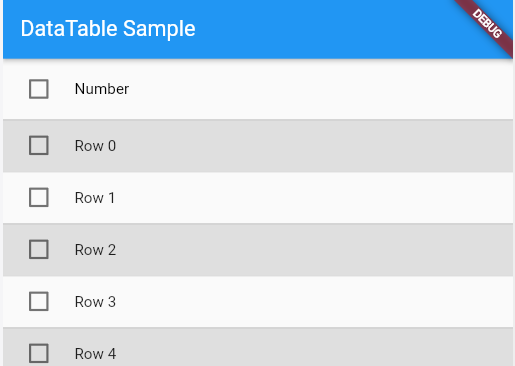
if (states.contains(MaterialState.selected)) {

return Theme.of(context).colorScheme.primary.withOpacity(0.08);}

if (index.isEven) { return Colors.grey.withOpacity(0.3); }

return null; // Use default value for other states and odd rows.

}),



HÌNH 2. 12. DataTable với các màu thay thế trên mỗi hàng

* 1. **Tìm hiều về Grid**

GridView trong Flutter là một widget được sử dụng để hiển thị một lưới (grid) của các widget con. Grid có thể được sắp xếp dọc hoặc ngang, và nó cho phép bạn hiển thị một danh sách các widget con trong một cách có cấu trúc

Có một số loại GridView khác nhau mà bạn có thể sử dụng

* **GridView.count**: Hiển thị một lưới với một số cố định của hàng hoặc cột. Bạn có thể xác định số lượng hàng hoặc cột bằng cách sử dụng thuộc tính crossAxisCount
* **GridView.builder**: Tương tự như ListView.builder, GridView.builder cung cấp một cách tối ưu hóa để xây dựng lưới với các phần tử động. Bạn cần xác định số lượng phần tử thông qua itemCount và sử dụng hàm itemBuilder để xây dựng từng phần tử.
* **GridView.extent:** Tương tự như GridView.count, nhưng thay vì xác định số lượng hàng hoặc cột, bạn xác định kích thước của mỗi phần tử (theo chiều ngang hoặc dọc).

Ví dụ này trình bày cách tạo GridView với hai cột. Những children được đặt cách nhau bằng cách sử dụng các thuộc tính crossAxisSpacingvà mainAxisSpacing .

GridView.count(primary: false,

padding: const EdgeInsets.all(20),

crossAxisSpacing: 10,

mainAxisSpacing: 10,

crossAxisCount: 2,

children: <Widget>[

Container(

padding: const EdgeInsets.all(8), color: Colors.teal[100],

child: const Text("He'd have you all unravel at the"),

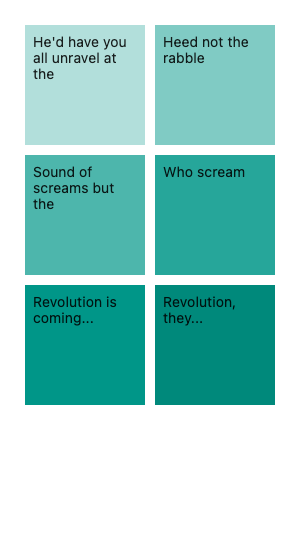
),

Container(

padding: const EdgeInsets.all(8), color: Colors.teal[200],

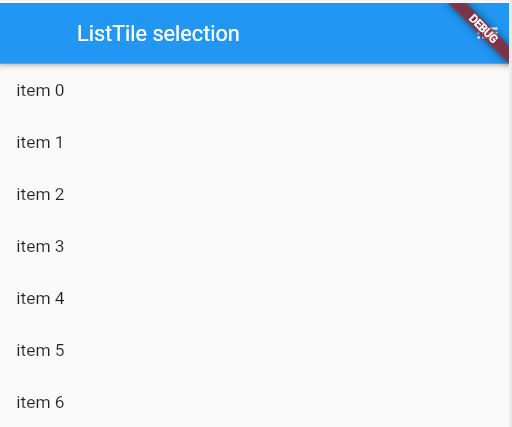
child: const Text('Heed not the rabble'),

),



HÌNH 2. 13. GridView với hai cột

Ví dụ này cho thấy việc triển khai tùy chỉnh lựa chọn ListTile trong GridView hoặc ListView . Nhấn và giữ bất kỳ ListTile nào để bật chế độ lựa chọn.



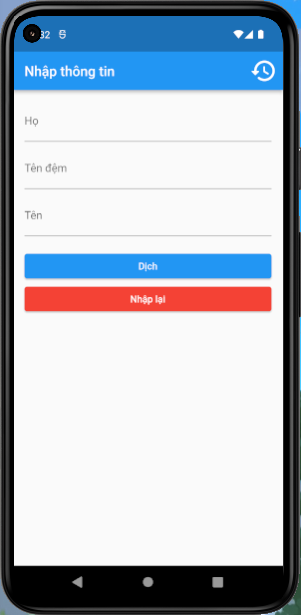
HÌNH 2. 14. ListTile trong GridView hoặc ListView

# **CHƯƠNG 3: MÔ TẢ ỨNG DỤNG**

* 1. **Mô tả giao diện**

Màn hình nhập thông tin bao gồm 3 TextField và 3 Button:

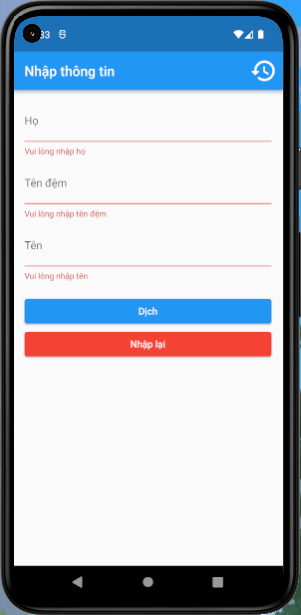
* Họ: nhập họ
* Tên đệm: nhập tên đệm
* Tên: nhập tên
* Nút Dịch: thực hiện chuyển họ tên của người dùng sang tiếng Anh
* Nút Nhập lại: thực hiện xóa dữ liệu đã nhập ở 3 ô TextField phía trên
* Nút Lịch sử: xem danh sách các họ tên đã được dịch trước đó



HÌNH 3. 1. Giao diện trang nhập thông tin

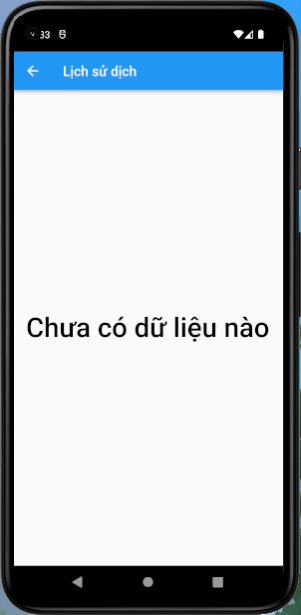
* 1. **Mô tả thao tác**

Nếu chưa nhập thông tin mà nhấn nút **Dịch** hệ thống sẽ báo lỗi và yêu cầu người dùng nhập thông tin.



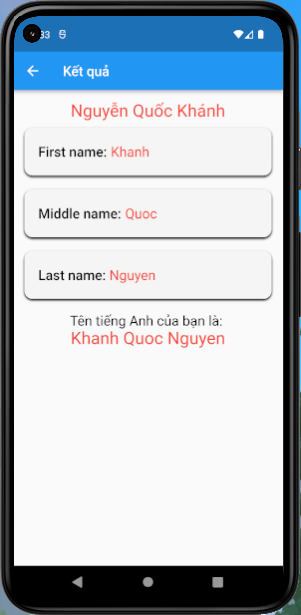
HÌNH 3. 2. Giao diện trang nhập thông tin báo lỗi

Màn hình trang Lịch sử dịch sẽ hiển thị danh sách các lần dịch thông tin họ tên trước đó. Nếu chưa thực hiện dịch lần nào thì hệ thống sẽ báo “*Chưa có dữ liệu nào*”.



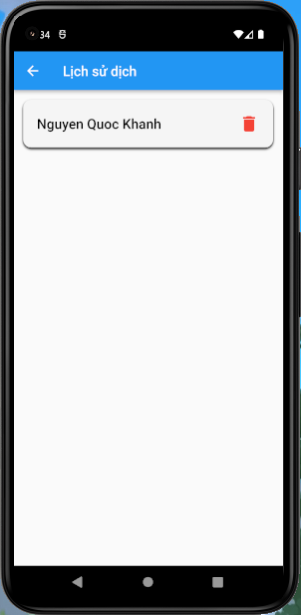
HÌNH 3. 3. Giao diện trang lịch sử khi chưa có dữ liệu

Khi thực hiện dịch, hệ thống sẽ trả về kết quả. Tại đây, sẽ hiển thị thông tin họ tên đã nhập trước đó, First name, Middle name, Last name và kết quả dịch.



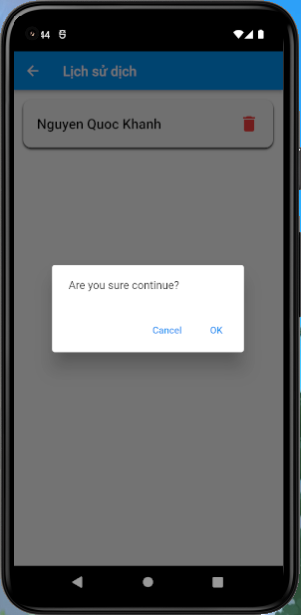
HÌNH 3. 4. Trang kết quả dịch tên tiếng việt sang tên tiếng anh

Khi đã thực hiện dịch, hệ thống sẽ tự động lưu lại kết quả. Có thể xem kết quả trong trang Lịch sử dịch và xóa lịch sử đi nếu muốn.



HÌNH 3. 5. Trang giao diện lịch sử khi có dữ liệu

Khi xóa hệ thống sẽ hỏi xác nhận có muốn xóa hay không, chọn Cancel để hủy hoặc OK để tiến hành xóa lịch sử.



HÌNH 3. 6. Giao diện khi xóa dữ liệu

# **CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

* 1. **Kết luận**
  + Nhóm em đã xây dựng thành công một ứng dụng di động Flutter hoàn chỉnh với giao diện thân thiện người dùng và chức năng chuyển đổi tên tiến Việt sang tiếng Anh.
  + Ứng dụng của nhóm đã được kiểm thử và đánh giá độ chính xác trong việc chuyển đổi tên với một loạt các tên phổ biến trong tiếng Việt, và kết quả cho thấy độ chính xác cao.
  + Người dùng có thể dễ dàng nhập tên tiến Việt và nhận kết quả chuyển đổi ngay lập tức, giúp tiết kiệm thời gian và nâng cao hiệu quả trong việc sử dụng tên tiếng Anh trong giao tiếp quốc tế.
  + Việc sử dụng Flutter làm nền tảng phát triển giúp ứng dụng có khả năng hoạt động trên cả nền tảng Android và iOS, tận dụng tích hợp đa nền tảng một cách hiệu quả.
  1. **Hướng phát triển**

Để nâng cao tính ứng dụng và đáp ứng tốt hơn nhu cầu giao tiếp đa ngôn ngữ, nhóm đề xuất hướng phát triển sau:

Mở rộng phạm vi ngôn ngữ: Mở rộng tích hợp để hỗ trợ nhiều ngôn ngữ khác nhau, giúp người dùng có thể chuyển đổi tên giữa các ngôn ngữ khác nhau một cách linh hoạt.

# **PHỤ LỤC**

**Câu hỏi các nhóm hỏi nhóm 1:**

Câu 1: ListView và Column/Row có điểm gì khác biệt? (nhóm 2)

Câu 2: DataTable trong Flutter được sử dụng để làm gì? (nhóm 3)

Câu 3: DataTable và ListView có điểm gì khác biệt? (nhóm 4)

**Câu hỏi nhóm 1 hỏi các nhóm còn lại:**

Câu 1: Hàm loadHistory được gọi trong phương thức initState trong file main.dart. Tại sao chúng ta cần gọi hàm loadHistory trong initState?

**A screen shot of a computer

Description automatically generated**

Câu 2: : Trong đoạn mã ở file second\_page.dart, hàm handleAddItem được gọi trong phương thức build. Điều này có thể gây ra những vấn đề gì và làm thay đổi cách hoạt động của ứng dụng như thế nào?

**A screen shot of a computer program

Description automatically generated**

Câu hỏi 3: Trong hàm checkData(), tại sao chúng ta kiểm tra các trường nhập liệu trước khi thực hiện chuyển hướng đến trang SecondPage?

**A screen shot of a computer

Description automatically generated**

# **TÀI LIỆU THAM KHẢO**

[1] **Frank Zammetti (**2019**)**, *Practical Flutter Improve Your Mobile Development with Google’s Latest Open-Source SDK ,* **Nhà xuất bản Apress, New York.**

[2] **Hans Kokx** (2023), *Flutter for Jobseekers: Learn Flutter and take your cross-platform app development skills to the next level (English Edition)*, **Nhà xuất bản BPB Publications**

[3] **Pouya Hosseini** (2019), *Flutter: Zero to App,* **flutterbooks.com**

[4] Trang web của Flutter <https://flutter.dev/> truy cập ngày 25/07/2023