**BỘ TÀI NGUYÊN VÀ MÔI TRƯỜNG**
**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG**
**KHOA HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**

**BÁO CÁO ĐỒ ÁN MÔN HỌC: CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI ĐỘNG**

# XÂY DỰNG UI FLOW CƠ BẢN

Lớp**: 08_ĐH_CNPM**

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

| | | |
|---|---|---|
| Sinh viên thực hiện**:** | **Nguyễn Thanh Ý** | **MSSV: 0850080056** |
| | **Trần Lê Tiến Hoà** | **MSSV: 0850080019** |
| | **Nguyễn Xuân Diệu** | **MSSV: 0850080010** |
| | **Thái Lai** | **MSSV: 0850080027** |
| | **Mai Anh Lộc** | **MSSV: 0850080028** |
| | **Võ Trần Uy** | **MSSV: 0850080053** |

*TP. Hồ Chí Minh, tháng 8 năm 2023*

**BỘ TÀI NGUYÊN VÀ MÔI TRƯỜNG**
**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG**
**KHOA HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**

**BÁO CÁO ĐỒ ÁN MÔN HỌC: CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI ĐỘNG**

# XÂY DỰNG UI FLOW CƠ BẢN

Lớp**: 08_ĐH_CNPM**

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

| | | |
|---|---|---|
| Sinh viên thực hiện**:** | **Nguyễn Thanh Ý** | **MSSV: 0850080056** |
| | **Trần Lê Tiến Hoà** | **MSSV: 0850080019** |
| | **Nguyễn Xuân Diệu** | **MSSV: 0850080010** |
| | **Thái Lai** | **MSSV: 0850080027** |
| | **Mai Anh Lộc** | **MSSV: 0850080028** |
| | **Võ Trần Uy** | **MSSV: 0850080053** |

*TP. Hồ Chí Minh, tháng 8 năm 2023*

# MỞ ĐẦU

Trong thế giới công nghệ ngày nay, ứng dụng di động và web đóng vai trò quan trọng trong việc kết nối và cung cấp trải nghiệm tuyệt vời cho người dùng. Để xây dựng những ứng dụng đa nền tảng linh hoạt và hiệu quả, việc thiết kế UI (User Interface) Flow chính là yếu tố cơ bản và quan trọng.

Đề tài "Xây dựng UI Flow cơ bản" nhằm nghiên cứu và tìm hiểu về quá trình xây dựng luồng giao diện người dùng (UI Flow) cho ứng dụng di động và web bằng cách sử dụng Flutter và Dart - hai công nghệ phát triển đa nền tảng hàng đầu. Chúng ta sẽ tiếp cận các khái niệm cơ bản của Flutter và Dart, học cách tạo ra các thành phần giao diện tương tác và cải thiện trải nghiệm người dùng. Chúng ta sẽ tập trung vào nghiên cứu về Flutter, một framework mã nguồn mở của Google, và Dart, ngôn ngữ lập trình chính của Flutter. Chúng ta sẽ bắt đầu từ việc giới thiệu về Flutter và Dart, sau đó khám phá kiến trúc và cấu trúc ứng dụng trong Flutter, cùng với tính đa nền tảng của nó.

Tiếp theo, chúng ta sẽ nghiên cứu về các thành phần quan trọng của giao diện người dùng như Drawer Menu, List, Inherited Widget và Build Context. Chúng ta sẽ hiểu cách sử dụng chúng để xây dựng giao diện linh hoạt và tương tác cho ứng dụng.

Qua đề tài này, chúng ta hy vọng nắm vững cách xây dựng UI Flow cơ bản cho các ứng dụng di động và web, từ đó tạo ra những trải nghiệm tuyệt vời và đáp ứng nhu cầu ngày càng cao của người dùng trong thế giới kỹ thuật số ngày nay.

# LỜI CẢM ƠN

Trong lời đầu tiên báo cáo đồ án cuối kỳ này, nhóm em muốn gửi lời cảm ơn và biết ơn chân thành nhất đến Ban giám hiệu nhà trường, các thầy cô giảng viên trong khoa Hệ thống thông tin và Viễn Thám của trường Đại học Tài nguyên và Môi trường Tp. HCM đã giảng dạy, hướng dẫn, trang bị kiến thức cho chúng em trong suốt quá trình học đại học, từ các kiến thức cơ bản đến vấn đề chuyên sâu. Từ đó đã tạo điều kiện tốt nhất cho nhóm em những tài liệu liên quan đến đề tài mà nhóm em thực hiện trong thời gian làm đồ án.

Chúng em xin chân thành cảm ơn giảng viên hướng dẫn thầy ThS. Nguyễn Thanh Truyền giảng viên khoa Hệ thống thông tin và Viễn Thám của trường Đại học Tài nguyên và Môi trường Tp.HCM, người đã trực tiếp hướng dẫn, đóng góp ý kiến, nhận xét và giúp đỡ nhóm em trong suốt quá trình thực hiện đồ án. Thầy còn động viên tinh thần và ân cần nhắc nhở tiến độ để nhóm em thực hiện xong báo cáo đồ án công nghệ phần mềm.

Nhóm em xin chân thành cảm ơn!

# NHẬN XÉT

……

……

……

……

……

……

……

……

……

……

……

……

*……., ngày….tháng….năm……*

*Giảng viên*

*(ký tên)*

## DANH MỤC PHÂN CÔNG

| STT | Họ và tên | Nhiệm vụ | Ghi chú |
|---|---|---|---|
| **1** | **Võ Trần Uy (C)** | **Phân công nhiệm vụ**<br>**Nghiên cứu DrawerMenu**<br>• Lý thuyết thực nghiệm<br>• Code áp dụng đề tài<br>**Xây dưng nội dung báo cáo – thuyết trình**<br>**Tham gia trả lời – đặt câu hỏi cho các nhóm** | **Hoàn thành tốt** |
| **2** | **Mai Anh Lộc** | **Nghiên cứu OrderList**<br>• Lý thuyết thực nghiệm<br>• Code áp dụng đề tài<br>**Xây dưng nội dung báo cáo – thuyết trình**<br>**Tham gia trả lời – đặt câu hỏi cho các nhóm** | **Hoàn thành tốt** |
| **3** | **Thái Lai** | **Nghiên cứu BuildContext**<br>• Lý thuyết thực nghiệm<br>• Code áp dụng đề tài<br>**Xây dưng nội dung báo cáo – thuyết trình**<br>**Tổng hợp ModuleCode** | **Hoàn thành tốt** |
| **4** | **Nguyễn Thanh Ý** | **Nghiên cứu BuildContext**<br>• Thuyết trình nội dung<br>• Code áp dụng đề tài<br>**Xây dưng nội dung báo cáo – thuyết trình**<br>**Tổng hợp ModuleCode** | **Hoàn thành tốt** |
| **5** | **Trần Lê Tiến Hòa** | **Nghiên cứu InheritedWidget**<br>• Lý thuyết thực nghiệm<br>• Code áp dụng đề tài<br>**Xây dưng nội dung báo cáo – thuyết trình**<br>**Tham gia trả lời – đặt câu hỏi cho các nhóm** | **Hoàn thành tốt** |
| **6** | **Nguyễn Xuân Diệu** | **Nghiên cứu InheritedWidget**<br>• Lý thuyết thực nghiệm<br>• Code áp dụng đề tài<br>**Xây dưng nội dung báo cáo – thuyết trình**<br>**Tham gia trả lời – đặt câu hỏi cho các nhóm** | **Hoàn thành tốt** |

# MỤC LỤC

# DANH MỤC HÌNH ẢNH

-

# CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

## 1.1. Định nghĩa vấn đề

Flutter và Dart là hai công nghệ ngày càng phổ biến trong việc phát triển ứng dụng di động. Với sự gia tăng đáng kể về lượng người dùng di động và sự đa dạng của các thiết bị, việc phát triển các ứng dụng chất lượng, mượt mà và có khả năng chạy trên nhiều nền tảng là một thách thức đối với các nhà phát triển.

Trước đây, việc phát triển ứng dụng di động đòi hỏi các nhà phát triển phải học nhiều ngôn ngữ lập trình khác nhau để xây dựng các ứng dụng cho các hệ điều hành khác nhau như Android và iOS. Điều này không chỉ tốn thời gian và công sức mà còn làm tăng chi phí phát triển và bảo trì ứng dụng. Hơn nữa, việc đảm bảo tính nhất quán và chất lượng giữa các phiên bản ứng dụng trên các nền tảng khác nhau cũng gặp nhiều khó khăn. Flutter và Dart ra đời với mục tiêu giải quyết những vấn đề này. Flutter cung cấp một framework đa nền tảng, cho phép nhà phát triển xây dựng một lần và chạy trên nhiều nền tảng một cách mượt mà và hiệu quả. Dart, là ngôn ngữ lập trình chính của Flutter, được tối ưu hóa để xây dựng các ứng dụng di động với hiệu suất cao và tính nhất quán.

Từ khi Flutter và Dart được giới thiệu, chúng đã thu hút sự quan tâm và ủng hộ đáng kể từ cộng đồng lập trình và các công ty công nghệ. Tính đến hiện tại, nhiều ứng dụng nổi tiếng đã được xây dựng bằng Flutter và Dart, điển hình là Google Ads, Alibaba, Reflectly, và nhiều ứng dụng khác. Mặc dù Flutter và Dart đã có những điểm mạnh và những thành công đáng kể, vẫn có một số ý kiến trái chiều xoay quanh việc sử dụng framework và ngôn ngữ này. Có những người cho rằng Flutter và Dart mới chỉ mới nổi và cần thời gian để chứng minh tính ổn định và đáng tin cậy. Ngoài ra, việc chọn Flutter và Dart cũng đòi hỏi nhà phát triển phải đầu tư thời gian học tập và làm quen với các công nghệ mới này.

### 1.2. Phạm vi đề tài

Phạm vi đề tài của chúng ta sẽ tập trung vào nghiên cứu và khám phá các khái niệm quan trọng trong việc phát triển ứng dụng Flutter và Dart. Cụ thể, chúng ta sẽ xem xét các yếu tố sau:

1. Drawer Menu: Drawer Menu là một thành phần giao diện người dùng phổ biến trong ứng dụng di động, cho phép hiển thị một menu ẩn từ cạnh trái hoặc cạnh phải của màn hình. Chúng ta sẽ tìm hiểu cách tạo và tùy chỉnh Drawer Menu trong Flutter, cũng như cách kết hợp nó với các màn hình chính để cung cấp trải nghiệm người dùng tốt hơn.

2. List: List là một thành phần quan trọng để hiển thị danh sách dữ liệu trong ứng dụng. Chúng ta sẽ nghiên cứu về các loại danh sách khác nhau trong Flutter, bao gồm ListView, GridView và Custom List, và cách sử dụng chúng để hiển thị dữ liệu một cách hiệu quả và linh hoạt.

3. Inherited Widget: Inherited Widget là một mô hình quan trọng trong Flutter để chia sẻ dữ liệu và trạng thái giữa các widget con trong cây widget. Chúng ta sẽ tìm hiểu cách sử dụng Inherited Widget để truyền dữ liệu xuống các widget con một cách hiệu quả, giúp tối ưu hóa hiệu năng và giảm thiểu việc tái tạo không cần thiết.

4. Build Context: Build Context là một đối tượng quan trọng trong Flutter, đại diện cho ngữ cảnh xây dựng (build context) của một widget trong cây widget. Chúng ta sẽ tìm hiểu về ý nghĩa của Build Context và cách sử dụng nó trong việc truy cập và thao tác với các widget cha và các widget anh chị em trong cây widget.

Thông qua việc nghiên cứu các khái niệm trên, chúng ta sẽ có cơ hội hiểu rõ hơn về cách xây dựng ứng dụng di động linh hoạt và mạnh mẽ bằng cách sử dụng Flutter và Dart. Đồng thời, việc áp dụng những kiến thức này vào viết code và xây dựng các ứng dụng thực tế sẽ giúp chúng ta trở thành những nhà phát triển ứng dụng di động hiệu quả và thành công.

### 1.3. Mục tiêu đề tài

Mục tiêu của đề tài này là tìm hiểu và khám phá sâu hơn về Flutter và Dart, hai công nghệ ngày càng phổ biến trong việc phát triển ứng dụng di động. Chúng ta sẽ tập trung vào nghiên cứu các khái niệm quan trọng như Drawer Menu, List, Inherited Widget và Build Context trong Flutter, để hiểu rõ hơn về cách xây dựng các ứng dụng di động linh hoạt và hiệu quả.

Bằng cách tìm hiểu về cú pháp và tính năng của ngôn ngữ Dart, chúng ta sẽ hiểu cách đơn giản hóa việc viết code và tăng cường khả năng phát triển ứng dụng di động. Chúng ta sẽ cùng khám phá cách Flutter cung cấp môi trường phát triển tốt nhất cho việc xây dựng giao diện người dùng đa nền tảng, đồng thời đảm bảo tính nhất quán và hiệu năng cao. Bên cạnh đó, chúng ta sẽ nghiên cứu cách sử dụng Inherited Widget để quản lý trạng thái và dữ liệu trong ứng dụng, giúp tái sử dụng code và tối ưu hóa hiệu năng. Việc áp dụng Drawer Menu và List trong ứng dụng giúp cung cấp trải nghiệm người dùng tốt hơn, tiết kiệm không gian màn hình và tăng tính tương tác của ứng dụng.

Cuối cùng, chúng ta sẽ tìm hiểu cách sử dụng Build Context để truy cập và thao tác với các widget trong cây widget, từ đó cải thiện khả năng tương tác và phản hồi của ứng dụng. Đề tài này hướng đến việc trình bày các khái niệm và ứng dụng thực tế, giúp tăng cường kỹ năng lập trình và xây dựng ứng dụng di động hiệu quả với Flutter và Dart.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## 2.1. Flutter và Dart

### 2.1.1. Tìm hiểu Flutter

#### 2.1.1.1. Flutter?

Flutter là một framework mã nguồn mở do Google phát triển, cho phép bạn xây dựng ứng dụng di động nhanh chóng và dễ dàng trên nhiều nền tảng như Android, iOS, Web và máy tính để bàn. Flutter sử dụng ngôn ngữ lập trình Dart, được tối ưu hóa để xây dựng các ứng dụng chạy mượt mà và nhanh chóng.

#### 2.1.1.2. Tính đa nền tảng Flutter

Một trong những điểm mạnh của Flutter là tính đa nền tảng, cho phép viết code một lần và triển khai ứng dụng trên nhiều nền tảng như Android, iOS, Web và máy tính để bàn. Điều này giúp tiết kiệm thời gian và công sức cho việc phát triển và bảo trì ứng dụng, đồng thời đảm bảo tính nhất quán và hiệu năng cao trên mọi nền tảng.

Tính đa nền tảng là lý do chính khiến nó trở thành một lựa chọn phổ biến cho việc phát triển ứng dụng di động. Điều này cho phép nhà phát triển viết code một lần và triển khai ứng dụng trên nhiều nền tảng khác nhau mà không cần phải viết lại code hoàn toàn từ đầu cho từng nền tảng riêng biệt.

Một số lợi ích quan trọng của tính đa nền tảng trong Flutter là:

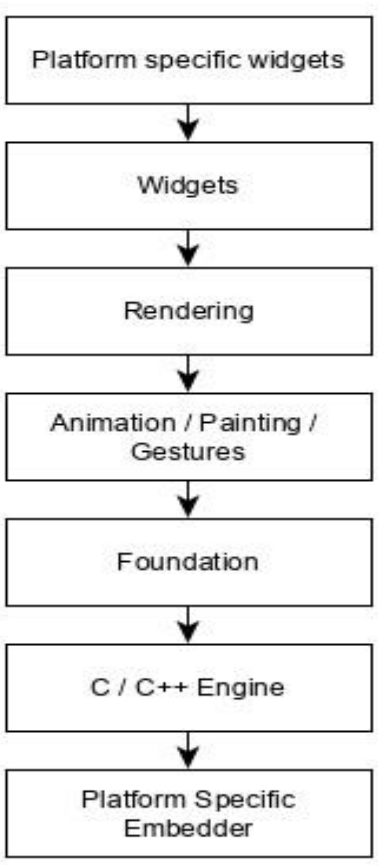
- Thay vì phải phát triển ứng dụng riêng lẻ cho từng hệ điều hành như Android và iOS, tính đa nền tảng của Flutter cho phép viết một mã nguồn chung và triển khai nó trên nhiều nền tảng chỉ với một lần viết. Điều này giúp tiết kiệm thời gian và công sức cho quá trình phát triển ứng dụng, đồng thời giảm thiểu chi phí bảo trì và cập nhật ứng dụng.
- Với tính đa nền tảng, các ứng dụng Flutter trên các nền tảng khác nhau sẽ có giao diện và trải nghiệm người dùng nhất quán. Điều này giúp tạo ra một ứng dụng chất lượng cao và đồng nhất, giúp tăng cường độ tin cậy và hài lòng của người dùng.
- Flutter hỗ trợ không chỉ Android và iOS, mà còn Web và máy tính để bàn. Điều này cho phép nhà phát triển mở rộng phạm vi ứng dụng của họ ra nhiều nền tảng và đáp ứng nhu cầu của người dùng trên nhiều thiết bị khác nhau.

- Mặc dù đa nền tảng, Flutter không giảm đi hiệu năng của ứng dụng. Nhờ vào việc sử dụng Dart, một ngôn ngữ được tối ưu hóa cho việc phát triển ứng dụng di động, Flutter cung cấp hiệu năng cao và tương thích tốt trên nhiều nền tảng.

Nhờ vào tính đa nền tảng của Flutter, các nhà phát triển có thể tiết kiệm thời gian, nỗ lực và tiền bạc khi phát triển ứng dụng di động. Việc này cũng giúp tạo ra các ứng dụng linh hoạt và mở rộng được đáp ứng nhu cầu của người dùng trên nhiều nền tảng, đồng thời đảm bảo tính nhất quán và hiệu năng cao trong trải nghiệm người dùng.

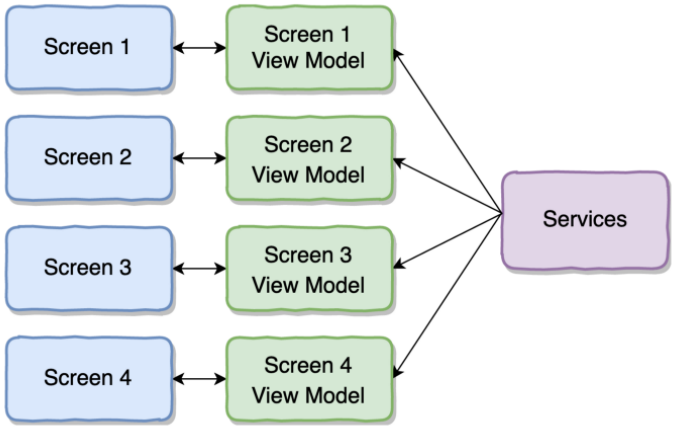
### 2.1.1.3. Kiến trúc và cấu trúc ứng dụng Flutter

Flutter được thiết kế như một hệ thống nhiều lớp, có thể mở rộng. Nó tồn tại dưới dạng một loạt các thư viện độc lập mà mỗi thư viện phụ thuộc vào lớp bên dưới. Không có lớp nào có đặc quyền truy cập vào các lớp bên dưới, mỗi phần framework được thiết kế để có thể tùy chọn và thay thế. Một khái niệm quan trọng của Flutter framework đó là các thành phần sẽ được nhóm lại theo độ phức tạp và được sắp xếp rõ ràng trong các tầng có độ phức tạp giảm dần. Một layer (tầng, lớp) được tạo thành bằng việc sử dụng các class tiếp theo ngay canh nó. Top của tất cả các layer là các widget đặc biệt cho Android và iOS. Layer tiếp theo là widget gốc của flutter. Tiếp lữa là Rendering layer, đây là level thấp nhất trong việc sinh các thành phần của flutter app. Layer tiếp theo là nền tảng gốc hệ điều hành.




***Kiến trúc Flutter**

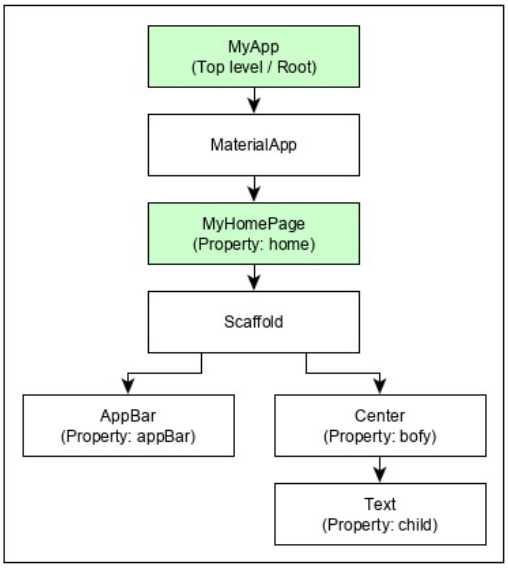
Nhìn hình ảnh kiến trúc Flutter trên có thể thấy Widget là thành phần giao diện cơ bản nhất để tạo giao diện người dùng của ứng dụng.




Trong ứng dụng Flutter, bản thân chính ứng dụng đã là một widget, một ứng dụng cũng có thể là tập hợp của nhiều ứng dụng nhỏ khác (nhiều widget con khác) được lồng vào nhau, liên kết với nhau để tạo ra một ứng dụng chính.

**Ví dụ minh họa:*

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
   // This widget is the root of your application.
   @override
   Widget build(BuildContext context) {
      return MaterialApp(
         title: 'Hello World Demo Application',
         theme: ThemeData(
            primarySwatch: Colors.blue,
         ),
         home: MyHomePage(title: 'Home page'),
      );
   }
}
class MyHomePage extends StatelessWidget {
   MyHomePage({Key key, this.title}) : super(key: key);
   final String title;

   @override
   Widget build(BuildContext context) {
      return Scaffold(
         appBar: AppBar(
            title: Text(this.title),
         ),
         body: Center(
            child:
            Text(
               'Hello World',
            )
         ),
      );
   }
}
```




*Giải thích sơ đồ trên*

- *MyApp là một widget được tạo ra bằng widget gốc của Flutter, MaterialApp.*
- *MaterialApp có các thuộc tính của màn hình home và mô tả giao diện người dùng, nó lại được tạo ra bởi một widget khác, MyHomePage.*
- *MyHomePage được tạo bởi một widget gốc của flutter, Scaffold*
- *Scaffold có 2 thuộc tính – body và appBar*
- *body chứa giao diện chính còn appBar chứa phần đầu (header) của ứng dụng*
- *Header UI là một widget gốc của flutter, AppBar và Body UI sử dụng Center widget.*
- *Center* widget có một thuộc tính, *Child*, nó chứa phần nội dung chính là một *Text* widget

### 2.1.2. Tìm hiểu Dart

#### 2.1.2.1. Giới thiệu về Dart, ngôn ngữ lập trình chính của Flutter.

Dart là một ngôn ngữ lập trình hiện đại có mục đích chung, cấp cao, được phát triển ban đầu bởi Google. Đây là ngôn ngữ lập trình mới xuất hiện vào năm 2011, nhưng phiên bản ổn định của nó đã được phát hành vào tháng 6 năm 2017. Dart không quá phổ biến vào thời điểm đó, nhưng nó đã trở nên phổ biến khi được sử dụng bởi Flutter.

Dart là một ngôn ngữ lập trình động, dựa trên lớp, hướng đối tượng với phạm vi đóng và từ vựng. Về mặt cú pháp, nó khá giống với Java, C và JavaScript. Nếu bạn biết bất kỳ ngôn ngữ lập trình nào trong số này, bạn có thể dễ dàng học ngôn ngữ lập trình Dart.

Dart là một ngôn ngữ lập trình mã nguồn mở được sử dụng rộng rãi để phát triển ứng dụng di động, ứng dụng web hiện đại, ứng dụng máy tính để bàn và Internet of Things (IoT) bằng cách sử dụng khung Flutter. Nó cũng hỗ trợ một số khái niệm nâng cao như giao diện, mixin, lớp trừu tượng, tổng thể trường và giao diện kiểu. Nó là một ngôn ngữ biên dịch và hỗ trợ hai loại kỹ thuật biên dịch.

- AOT (Ahead of Time) – Nó chuyển đổi mã Dart sang mã JavaScript được tối ưu hóa với sự trợ giúp của trình biên dịch dar2js và chạy trên tất cả các trình duyệt web hiện đại. Nó biên dịch mã tại thời điểm xây dựng.
- JOT (Just-In-Time) – Nó chuyển đổi mã byte trong mã máy (mã gốc), nhưng chỉ mã cần thiết.

#### 2.1.2.2. Đặc điểm – Cú pháp của Dart

Dart là một ngôn ngữ lập trình hướng đối tượng, có cú pháp gần giống với nhiều ngôn ngữ lập trình phổ biến như Java, JavaScript và C#. Tuy nhiên, Dart cũng có những đặc điểm riêng biệt, bao gồm:

- Tính nhất quán: Cú pháp của Dart được thiết kế để dễ đọc, gọn gàng và nhất quán, giúp tăng cường tính dễ hiểu và bảo trì mã nguồn.
- Typing tĩnh và dynamic: Dart hỗ trợ kiểu dữ liệu tĩnh và kiểu dữ liệu động. Nhà phát triển có thể sử dụng kiểu dữ liệu tĩnh để kiểm tra lỗi tại thời điểm biên dịch, trong khi kiểu dữ liệu động cho phép linh hoạt trong việc xử lý dữ liệu.
- Thư viện phong phú: Dart có một hệ sinh thái thư viện phong phú, cung cấp nhiều tính năng và chức năng tiện ích cho việc phát triển ứng dụng.
- Async/Await: Dart hỗ trợ các khái niệm Async/Await, giúp viết mã bất đồng bộ một cách dễ dàng và rõ ràng.

Cú pháp Dart:

- Mọi lớp đều kế thừa từ lớp cơ sở có tên Object
- Mọi thứ lưu trong biến đều là đối tượng (kể cả số, kể cả null)
- Sử dụng từ khóa dynamic khi khai báo biến có chấp nhận mọi kiểu
- Dart hỗ trợ định nghĩa kiểu generic
- Dart cho phép định nghĩa hàm trong hàm (lồng nhau)
- Dart không có từ khóa public, private, protected khi khai báo phương thức, thuộc tính lớp. Nếu thuộc tính, tên lớp bắt đầu bằng _ thì hiểu đó là private
- Tên biến, tên hàm, tên lớp … là chuỗi chữ có thể kế hợp với số và có thể bắt đầu bằng _

#### 2.1.2.3. Dart và ngôn ngữ lập trình khác

- So với Java và Kotlin (ngôn ngữ phát triển Android): Dart có cú pháp dễ đọc và gọn gàng hơn, đồng thời hỗ trợ cả kiểu dữ liệu tĩnh và động. Tuy nhiên, Kotlin đã phát triển mạnh mẽ và trở thành ngôn ngữ ưu tiên cho việc phát triển ứng dụng Android.
- So với Objective-C và Swift (ngôn ngữ phát triển iOS): Dart giống Swift trong việc hỗ trợ kiểu dữ liệu tĩnh và cú pháp nhất quán. Dart không phổ biến bằng Swift, nhưng Flutter vẫn có thể sử dụng để phát triển ứng dụng iOS.
- So với JavaScript và TypeScript (ngôn ngữ phát triển web): Dart có cú pháp tương tự như JavaScript và TypeScript, nhưng lại hỗ trợ kiểu dữ liệu tĩnh, giúp tăng cường kiểm tra lỗi trong quá trình phát triển ứng dụng web.

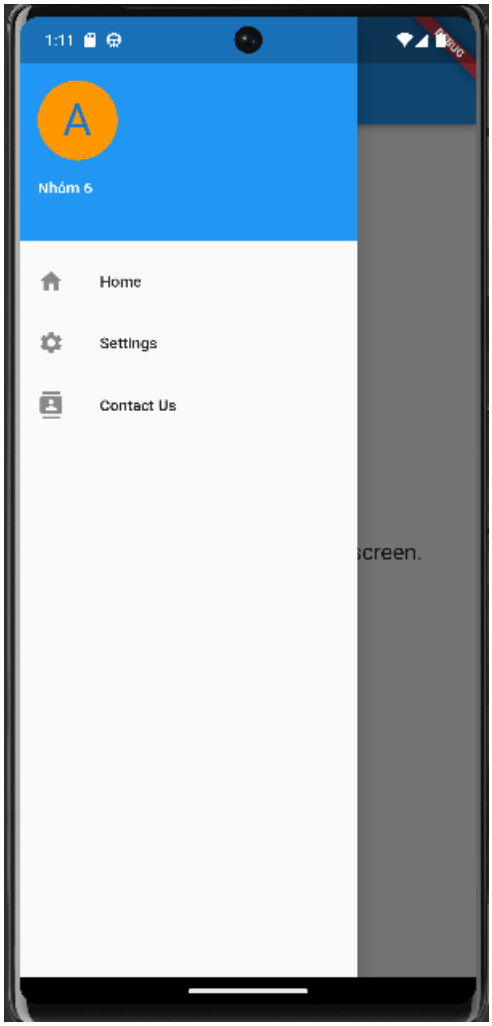
Tổng kết, Dart là một ngôn ngữ lập trình mạnh mẽ và linh hoạt, với cú pháp dễ đọc và nhất quán. Nó đã trở thành ngôn ngữ chính của Flutter và được sử dụng rộng rãi để phát triển các ứng dụng di động đa nền tảng và các ứng dụng web.

## 2.2. Các mục tiêu tìm hiểu

### 2.2.1. Drawer Menu

Drawer là một thành phần UI trong Flutter được sử dụng để hiển thị một một màn hình bên vô hình. Đây là một menu trượt bên thường được sử dụng để chứa các tùy chọn điều hướng và chứa các liên kết quan trọng trong ứng dụng và chiếm một nửa màn hình khi hiển thị.

Drawer menu thường xuất hiện khi người dùng vuốt từ cạnh bên trái (hoặc bên phải) của màn hình hoặc khi nhấp vào biểu tượng menu (thường là biểu tượng ba đường gạch ngang) trong thanh ứng dụng hoặc thanh tiêu đề của ứng dụng




*Hình 2. 1 Hình ảnh minh họa drawer menu*

#### 2.2.1.1. Xây dựng một Drawer Menu cơ bản

Trước tiên, hãy tạo ra một drawer trống. Cũng như các ứng dụng Flutter khác với thiết kế material-ui, chúng ta sẽ tạo ra một MaterialApp cơ bản, màn hình chính sẽ chứa một Scaffold với Drawer.

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  final appTitle = 'Flutter Drawer Demo';

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: appTitle,
      home: DWidget(),
    );  // MaterialApp
  }
}
```

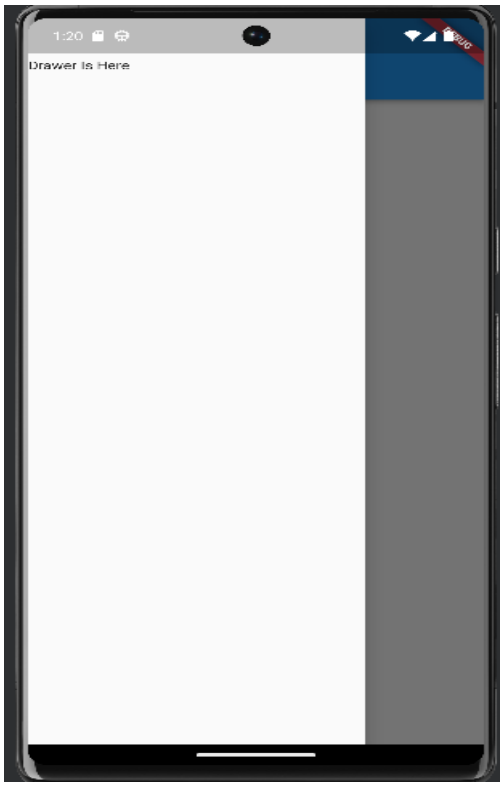
*Hình 2. 2 Xây dựng Drawer Menu*

DWidget ở đây là widget sẽ chứa các drawers. Cần lưu ý rằng các drawers là một phần của Scaffold cùng với appBar và body. Khi chúng ta thêm drawer vào Scaffold, nó sẽ tạo ra một mục menu ba dòng ở góc trên cùng bên trái của thanh ứng dụng, khi nhấp vào sẽ hiển thị màn hình drawer.

```
class DWidget extends StatelessWidget {
  @override
  Widget build (BuildContext ctxt) {
    return Scaffold(
      drawer:  const Drawer(
        child:  Text("\n\n\nDrawer Is Here"),
      ),  // Drawer
      appBar:  AppBar(
        title:  Text("Drawer Demo"),
      ),  // AppBar
      body:  Text("Drawer Body"),
    );  // Scaffold
  }
}
```

*Hình 2. 3 Đoạn code Demo Drawer Menu*

Kết quả:




*Hình 2. 4 Kết quả đoạn code Drawer Menu demo*

Có thể nhận thấy rằng drawer, có child: và không phải children:, có nghĩa là tại bất kỳ thời điểm nào, chúng ta chỉ có thể có một widget trong drawer.

Để có nhiều hơn một widget, chúng ta cần sử dụng các widget có khả năng chứa nhiều widget con như Column.

```
class DWidget extends StatelessWidget {
  @override
  Widget build (BuildContext ctxt) {
    return const Scaffold(
      drawer:  Drawer(
        child:  Column(
          children: <Widget>[
            Text("\n\n\n\Drawer Is Here => 1"),
            Text("Drawer Is Here => 2"),
            Text("Drawer Is Here => 3"),
          ],  // <Widget>[]
        )  // Column
      ),  // Drawer

    );  // Scaffold
  }
}
```

*Hình 2. 5 Đoạn code thêm các option vào Drawer Menu*

**Kết quả:*

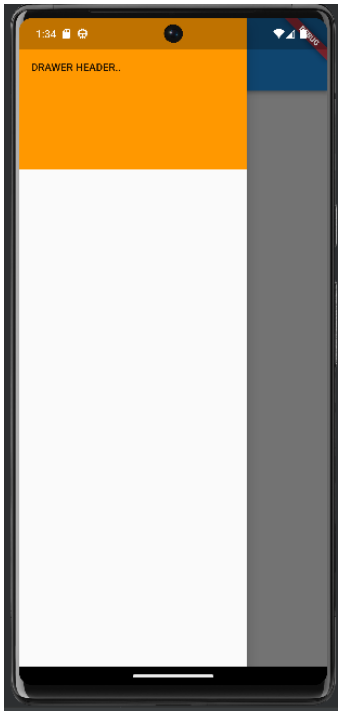

*Hình 2. 6 Kết quả đoạn code thêm các option vào Drawer Menu*

Trong Flutter, chúng ta có thể tạo một tiêu đề tương tự bằng cách sử dụng tiện ích DrawerHeader có một child và cho phép chúng ta trang trí tiêu đề. Ở đây, sử dụng BoxDecoration để chúng ta có thể phân biệt ranh giới hoàn chỉnh của widget.

```
class DWidget extends StatelessWidget {
  @override
  Widget build(BuildContext ctxt) {
    return Scaffold(
      appBar: AppBar(
        title: Text("App Title"),
      ),  // AppBar
      drawer: Drawer(
        child: ListView(
          padding: EdgeInsets.zero,
          children: const <Widget>[
            DrawerHeader(
              child: Text("DRAWER HEADER.."),
              decoration: BoxDecoration(
                color: Colors.orange,
              ),  // BoxDecoration
            ),  // DrawerHeader
          ],  // <Widget>[]
        ),  // ListView
      ),  // Drawer
      body: const Center(
        child: Text("Content of the page"),
      ),  // Center
    );  // Scaffold
  }
}
```

*Hình 2. 7 Đoạn code tạo header Drawer Menu*

*Kết quả đạt đượcs :*



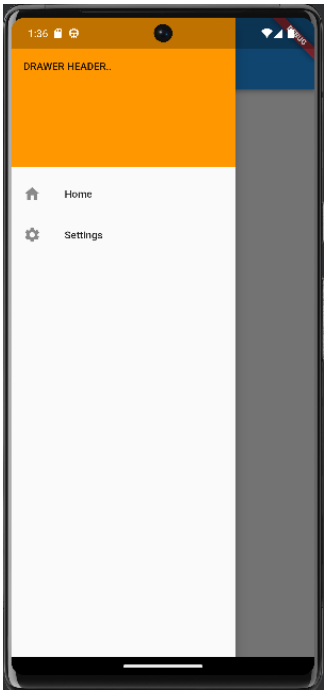

*Hình 2. 8 Kết quả đoạn code tạo header Drawer Menu*

Thêm các tùy chọn vào Drawer Menu

```
const DrawerHeader(
  child: Text("DRAWER HEADER.."),
  decoration: BoxDecoration(
    color: Colors.orange,
  ),  // BoxDecoration
),  // DrawerHeader
ListTile(
  leading: const Icon(Icons.home),
  title: const Text("Home"),
  onTap: () {
    // Điều hướng đến trang chủ
  },
),  // ListTile
ListTile(
  leading: Icon(Icons.settings),
  title:const  Text("Settings"),
  onTap: () {
    // Điều hướng đến trang cài đặt
  },
),  // ListTile
],  // <Widget>[]
),  // ListView
),  // Drawer
```

*Hình 2. 9 Đoạn code thêm các option vào Drawer Menu*

**Kết quả đạt được:*




*Hình 2. 10 Kết quả Drawer Menu cơ bản*

Như vậy, vừa rồi chúng ta đã hoàn tất việc xây dựng 1 drawer menu cơ bản

### 2.2.2. Order List

Order List là một nhóm có thứ tự các đối tượng. Một List sử dụng để chứa nhiều đối tượng / giá trị – ta gọi chung là phần tử – trong một biến duy nhất.

List(Danh sách) là yếu tố phổ biến nhất của mọi ứng dụng web hoặc di động. Chúng được tạo thành từ nhiều hàng mục, bao gồm văn bản, nút, nút chuyển đổi, biểu tượng, hình thu nhỏ và nhiều hàng khác. Chúng ta có thể sử dụng nó để hiển thị các thông tin khác nhau như menu, tab hoặc để phá vỡ sự đơn điệu của các tệp văn bản thuần túy.

Flutter cho phép bạn làm việc với List theo các cách khác nhau, được đưa ra dưới đây:

- List cơ bản
- List dài (Long Lists)
- List lưới (Grid Lists)
- List ngang(Horizontal Lists)

#### 2.2.2.1. List cơ bản

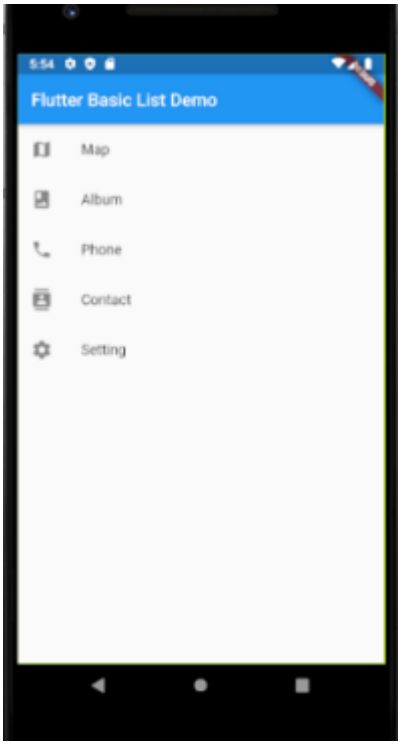
Flutter bao gồm một tiện ích **ListView** để làm việc với Lists, đây là khái niệm cơ bản về hiển thị dữ liệu trong ứng dụng dành cho thiết bị di động. ListView là một tiêu chuẩn hoàn hảo để hiển thị List chỉ chứa một vài mục. ListView cũng bao gồm tiện ích **ListTitle** , cung cấp nhiều thuộc tính hơn cho cấu trúc trực quan vào List dữ liệu.

*Ví dụ sau đây hiển thị một List cơ bản trong ứng dụng Flutter.*

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final appTitle = 'Flutter Basic List Demo';

    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: Text(appTitle),
        ),
        body: ListView(
          children: <Widget>[
            ListTile(
              leading: Icon(Icons.map),
              title: Text('Map'),
            ),
            ListTile(
              leading: Icon(Icons.photo_album),
              title: Text('Album'),
            ),
            ListTile(
              leading: Icon(Icons.phone),
              title: Text('Phone'),
            ),
            ListTile(
              leading: Icon(Icons.contacts),
              title: Text('Contact'),
            ),
            ListTile(
              leading: Icon(Icons.settings),
              title: Text('Setting'),
            ),
          ],
        ),
      ),
    );
  }
}
```

*Hình 2. 11 Đoạn code List cơ bản trong Fultter*

**Kết quả của đoạn code là:*




*Hình 2. 12 Kết quả đoạn code List cơ bản*

#### 2.2.2.2. Làm việc với LongList

Đôi khi bạn muốn hiển thị một List rất dài trong một màn hình ứng dụng của mình, thì trong trường hợp đó, phương pháp hiển thị List ở trên không hoàn hảo. Để làm việc với một List chứa một số lượng rất lớn các mục, chúng ta cần sử dụng một hàm tạo **ListView.builder()** . Sự khác biệt chính giữa ListView và ListView.builder là ListView tạo tất cả các mục cùng một lúc, trong khi hàm tạo ListView.builder() tạo các mục khi chúng được cuộn trên màn hình.

**Một ví dụ minh họa cho LongList:*

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp(
    products: List<String>.generate(500, (i) => "Product List: $i"),
  ));
}

class MyApp extends StatelessWidget {
  final List<String> products;

  MyApp({Key key, @required this.products}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final appTitle = 'Flutter Long List Demo';

    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: Text(appTitle),
        ),
        body: ListView.builder(
          itemCount: products.length,
          itemBuilder: (context, index) {
            return ListTile(
              title: Text('${products[index]}'),
            );
          },
        ),
      ),
    );
  }
}
```

*Hình 2. 13 Đoạn code tạo một Long List cơ bản*

**Kết quả đạt được:*


*Hình 2. 14 Kết quả đoạn code tạo Long List cơ bản*

#### 2.2.2.3. Tạo Grid List

Đôi khi chúng ta muốn hiển thị các mục trong bố cục lưới thay vì List bình thường xuất hiện lần lượt. Một widget **GridView** cho phép bạn tạo một List lưới trong Flutter. Cách đơn giản nhất để tạo lưới là sử dụng hàm tạo **GridView.count()** , hàm này chỉ định số hàng và cột trong lưới.

**Một ví dụ minh họa cho Grid List*

```dart
import 'package:flutter/material.dart';

void main() {runApp(MyApp());}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final appTitle = "Flutter Grid List Demo";

    return MaterialApp(
        title: appTitle,
        home: Scaffold(appBar: AppBar(
          title: Text(appTitle),
        ),
            body: GridView.count(
                crossAxisCount: 3,
                children: List.generate(choices.length, (index) {
                  return Center(
                    child: SelectCard(choice: choices[index]),
                  );
                }
                )
            )
        )
    );
  }
}

class Choice {
  const Choice({this.title, this.icon});
  final String title;
  final IconData icon;
}
```

*Hình 2. 15 Đoạn code tạo List  Grild View cơ bản (đoạn 1)*

```
const List<Choice> choices = const <Choice>[
  const Choice(title: 'Home', icon: Icons.home),
  const Choice(title: 'Contact', icon: Icons.contacts),
  const Choice(title: 'Map', icon: Icons.map),
  const Choice(title: 'Phone', icon: Icons.phone),
  const Choice(title: 'Camera', icon: Icons.camera_alt),
  const Choice(title: 'Setting', icon: Icons.settings),
  const Choice(title: 'Album', icon: Icons.photo_album),
  const Choice(title: 'WiFi', icon: Icons.wifi),
  const Choice(title: 'GPS', icon: Icons.gps_fixed),
];

class SelectCard extends StatelessWidget {
  const SelectCard({Key key, this.choice}) : super(key: key);
  final Choice choice;

  @override
  Widget build(BuildContext context) {
    final TextStyle textStyle = Theme.of(context).textTheme.display1;
    return Card(
        color: Colors.lightGreenAccent,
        child: Center(child: Column(
            mainAxisSize: MainAxisSize.min,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
              Expanded(child: Icon(choice.icon, size:50.0, color: textStyle.color)),
              Text(choice.title, style: textStyle),
            ]
        ),
        )
    );
  }
}
```

*Hình 2. 16 Đoạn code tạo List Gird View cơ bản (đoạn 2)*

**Kết quả đạt được*


*Hình 2. 17 Kết quả của đoạn code tạo List Gird View cơ bản*

#### 2.2.2.4. Tạo Horizontal List

Tiện ích ListView cũng hỗ trợ List ngang. Đôi khi chúng ta muốn tạo một List có thể cuộn theo chiều ngang chứ không phải theo chiều dọc. Trong trường hợp đó, ListView cung cấp hướng **cuộn** ngang ghi đè hướng dọc.

*Ví dụ minh họa cho Tạo Horizontal List*

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final title = 'Flutter Horizontal Demo List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: Text(title),
        ),
        body: Container(
          margin: EdgeInsets.symmetric(vertical: 25.0),
          height: 150.0,
          child: ListView(
            scrollDirection: Axis.horizontal,
            children: <Widget>[
              Container(
                width: 150.0,
                color: Colors.blue,
                child: new Stack(
                  children: <Widget>[
                    ListTile(
                      leading: Icon(Icons.home),
                      title: Text('Home'),
                    ),
                  ],
                ),
              ),
              Container(
                width: 148.0,
                color: Colors.green,
                child: new Stack(
                  children: <Widget>[
                    ListTile(
                      leading: Icon(Icons.camera_alt),
                      title: Text('Camera'),
                    ),
                  ],
                ),
              ),
```

*Hình 2. 18 Đoạn code tạo Horizontal List (đoạn 1)*

```
                Container(
                  width: 148.0,
                  color: Colors.yellow,
                  child: new Stack(
                    children: <Widget>[
                      ListTile(
                        leading: Icon(Icons.phone),
                        title: Text('Phone'),
                      ),
                    ],
                  ),
                ),
                Container(
                  width: 148.0,
                  color: Colors.red,
                  child: new Stack(
                    children: <Widget>[
                      ListTile(
                        leading: Icon(Icons.map),
                        title: Text('Map'),
                      ),
                    ],
                  ),
                ),
                Container(
                  width: 148.0,
                  color: Colors.orange,
                  child: new Stack(
                    children: <Widget>[
                      ListTile(
                        leading: Icon(Icons.settings),
                        title: Text('Setting'),
                      ),
                    ],
                  ),
                ),
              ],
            ),
          ),
        ),
      );
    }
}
```

*Hình 2. 19 Đoạn code tạo Horizontal List (đoạn 1)*

**Kết quả đạt được*


*Hình 2. 20 Kết quả đoạn code tạo Horizontal List (đoạn 1)*

### 2.2.3. Inherited Widget

#### 2.2.3.1. Tìm hiểu – Hình dùng về Inherited Widget

Làm thế nào để truyền data từ một widget cha nào đó xuống thẳng widget chắt mà không phải sử dụng constructor để truyền xuống từ từ từng widget một. Đó là sử dụng BuildContext kết hợp với InheritedWidget. BuildContext đã được giải thích trong phần trên. Bây giờ chúng ta sẽ tìm hiểu InheritedWidget.

**Chúng ta có ví dụ sau:**

- Chúng ta sẽ thấy để truyền được data từ MyHomePage xuống child widget của nó là CounterWidget thì trong class CounterWidget, ta phải tạo 1 constructor có vẻ giống hệt constructor của MyHomePage.
- Kỹ thuật truyền data từ widget cha xuống Widget con thông qua constructor như vậy được gọi là "Passing state down".

```dart
class MyHomePage extends StatefulWidget {
  final bool isLoading;
  final int counter;

  const MyHomePage({
    required this.isLoading,
    required this.counter,
  });

  @override
  State<MyHomePage> createState() {...}
}

class CounterWidget extends StatelessWidget {
  final bool isLoading;
  final int counter;

  const CounterWidget({
    required this.isLoading,
    required this.counter,
  });

  @override
  Widget build(BuildContext context) {...}
}
```

```dart
class MyCenterWidget extends StatelessWidget {
  final bool isLoading;
  final int counter;

  const MyCenterWidget({
    required this.isLoading,
    required this.counter,
  });

  @override
  Widget build(BuildContext context) {
    return Center(
      child: CounterWidget(
        isLoading: isLoading,
        counter: counter,
      ), // CounterWidget
    ); // Center
  }
}
```

- Giả sử, trong đoạn code trên, nếu chúng ta extract widget Center ra một Widget đặt tên là MyCenterWidget thì ta cũng phải tạo một constructor với 2 property là counter và isLoading để nó có thể nhận data từ MyHomePage truyền xuống cho CounterWidget.

=>>> ***Nếu cứ làm theo cách như vậy, từ một widget ông muốn truyền data xuống widget cháu, ta phải truyền sang tay từng người một, từ ông → ba → con → cháu. Chúng ta đã thấy vấn đề nghiêm trọng ở đây chưa?***

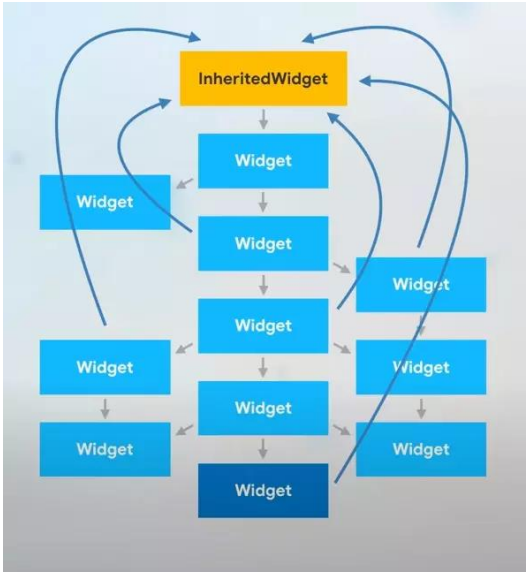
Có 2 vấn đề xảy ra ở đây là:

1. Nếu khoảng cách giữa MyHomePage và CounterWidget trên Widget tree càng xa thì chúng ta sẽ càng cực khổ để truyền được data từ MyHomePage xuống CounterWidget qua constructor của nhiều Widget trung gian.
2. Khi MyHomePage gọi hàm setState, nó sẽ gọi lại hàm build khiến cho cả MyCenterWidget và CounterWidget đều được khởi tạo lại và gọi hàm build. Đây là 1 sự lãng phí vì ta chỉ cần widget CounterWidget được rebuild mà thôi.

Vậy làm thế nào để khắc phục được 2 nhược điểm trên. Có một loại Widget có thể giúp ta làm điều đó là InheritedWidget

#### 2.2.3.2. Cấu trúc Inherited Widget

InheritedWidget là một nơi lưu trữ data và cung cấp data cho widget con trong widget tree. Tất cả widget con của InheritedWidget đều có thể truy cập vào InheritedWidget để lấy data. Tức là từ vị trí InheritedWidget, bạn không cần thiết phải truyền data xuống từng 1 widget con một nữa mà Widget con ở bất kỳ vị trí nào trên widget tree muốn lấy data từ InheritedWidget.




*Hình 2. 21 Cấu trúc **InheritedWidget***

Trong Flutter, **InheritedWidget** là một lớp rất quan trọng và mạnh mẽ trong việc quản lý trạng thái và truyền dữ liệu giữa các thành phần trong cây widget. Nó giúp bạn chia sẻ thông tin qua các thành phần con mà không cần truyền thông tin qua nhiều cấp widget cha. Một số trạng thái chung, chẳng hạn như đa ngôn ngữ, chế độ tối, các thiết lập toàn cục, ... thường được quản lý thông qua **InheritedWidget**.

*Để hiểu rõ hơn về **InheritedWidget** ta sẽ đưa ra một ví dụ cụ thể như sau:*

```
import 'package:flutter/material.dart';

// Bước 1: Tạo một lớp InheritedWidget chứa thông tin bạn muốn chia sẻ
class MyInheritedWidget extends InheritedWidget {
  final String data;
  MyInheritedWidget({required this.data, required Widget child})
      : super(child: child);

  @override
  bool updateShouldNotify(MyInheritedWidget oldWidget) {
    return data != oldWidget.data;
  }
  static MyInheritedWidget? of(BuildContext context) {
    return context.dependOnInheritedWidgetOfExactType<MyInheritedWidget>();
  }
}
// Bước 2: Sử dụng MyInheritedWidget để chia sẻ thông tin trong các widget con
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Sử dụng MyInheritedWidget.of để truy cập dữ liệu từ MyInheritedWidget
    final inheritedData = MyInheritedWidget.of(context)?.data ?? 'Default Data';
    return Scaffold(
      appBar: AppBar(title: Text('InheritedWidget Example')),
      body: Center(
        child: Text('Data from InheritedWidget: $inheritedData'),
      ),
    );
  }
}
```

Để sử dụng InheritedWidget, bạn cần làm hai bước:

1. Tạo một lớp InheritedWidget chứa thông tin bạn muốn chia sẻ.
2. Sử dụng InheritedWidget để bao bọc widget con và truy xuất thông tin trong các widget con bằng cách sử dụng InheritedWidget kế thừa.

- *Trong ví dụ, **MyInheritedWidget** chứa thông tin dạng chuỗi (ở đây là data). Sau đó, chúng ta sử dụng **MyInheritedWidget**.of(context) trong **MyWidget** để truy cập dữ liệu được chia sẻ. Nếu dữ liệu trong **MyInheritedWidget** thay đổi, các widget con*

*như **MyWidget** sẽ tự động được cập nhật lại với dữ liệu mới mà không cần phải thực hiện bất kỳ công việc cập nhật nào.*

#### 2.2.3.3. Đặc điểm Inherited Widget

Khi sử dụng InheritedWidget, có một số đặc điểm và vấn đề cần lưu ý để tránh các lỗi và tối ưu hóa hiệu suất ứng dụng:

- Tránh quá sâu cây InheritedWidget: Nếu cây InheritedWidget quá sâu, việc duyệt cây để tìm đối tượng được chia sẻ có thể trở nên rất tốn kém. Hãy đảm bảo chỉ chia sẻ những thông tin cần thiết và tránh đặt quá nhiều lớp InheritedWidget trong cây của bạn.
- Tối ưu việc cập nhật: Khi một InheritedWidget thay đổi giá trị, tất cả các widget con phụ thuộc sẽ được cập nhật. Điều này có thể gây ra nhiều việc cập nhật không cần thiết, đặc biệt nếu widget con không thực sự sử dụng dữ liệu được chia sẻ. Để giảm thiểu số lượng cập nhật không cần thiết, hãy sử dụng InheritedModel hoặc tối ưu hơn là sử dụng provider package cùng với ChangeNotifier.
- Cẩn thận khi thay đổi dữ liệu: Khi bạn thay đổi dữ liệu trong InheritedWidget, hãy đảm bảo rằng bạn đã gọi phương thức setState() hoặc tương tự để thông báo cho Flutter rằng dữ liệu đã thay đổi. Nếu không, widget con có thể không được cập nhật và hiển thị dữ liệu mới.
- Sử dụng kết hợp với Stateful Widgets khi cần thiết: InheritedWidget không phải là giải pháp hoàn hảo cho tất cả các trạng thái trong ứng dụng của bạn. Trong một số trường hợp, bạn có thể muốn sử dụng StatefulWidget để quản lý các trạng thái cục bộ của các widget và sử dụng InheritedWidget chỉ để chia sẻ thông tin giữa các widget.
- Sử dụng setState() trong InheritedWidget: Việc sử dụng setState() trong InheritedWidget có thể gây ra một số vấn đề, bởi vì setState() chỉ làm việc với các widget kế thừa từ StatefulWidget. Thay vào đó, hãy xem xét sử dụng ChangeNotifier hoặc provider package để quản lý trạng thái và cập nhật trong InheritedWidget.
- InheritedWidget và Route Rebuilds: Khi sử dụng Navigator.push để điều hướng sang một route mới, toàn bộ widget tree của route trước đó có thể được xây dựng lại khi

route quay lại. Điều này có thể làm cho các thay đổi trong InheritedWidget không được duyệt đúng cách. Để giải quyết vấn đề này, hãy cân nhắc sử dụng ModalRoute.of(context)?.addScopedWillPopCallback() để giữ các thay đổi của InheritedWidget sau khi route quay lại.

### 2.2.4. Build Context

Trong Flutter, "build context" là một đối tượng được sử dụng để tạo và quản lý các thành phần giao diện người dùng. Nó đại diện cho cây widget (widget tree) hiện tại mà Flutter đang xây dựng để hiển thị giao diện người dùng. Build context là một đối tượng không thể thay đổi trong quá trình thực thi của ứng dụng và được sử dụng để tìm và tương tác với các widget trong cây hiện tại.

#### 2.2.4.1. Các trường hợp trong Build Context

- Trong hàm build(): Mỗi widget trong Flutter có một phương thức build() tương ứng, nơi bạn xác định giao diện người dùng mà widget đó sẽ hiển thị. Khi build() được gọi, nó nhận một tham số là một build context để xây dựng cây widget.
- Trong hàm BuildContext.inheritFromWidgetOfExactType<T>(): Đây là một phương thức tìm kiếm các widget cha trong cây widget và trả về widget gần nhất có kiểu T (hoặc kiểu con của T). Điều này rất hữu ích khi bạn muốn truy cập dữ liệu hoặc trạng thái từ widget cha trong widget con.

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Example using BuildContext'),
        ),
        body: Center(
          child: MyWidget(),
        ),
```

```
        ),
      );
    }
  }
  class MyWidget extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
      // Sử dụng BuildContext để truy cập các thuộc tính của widget cha
      String appTitle = context.findAncestorWidgetOfExactType<AppBar>()?.title?.data ??
  'Unknown';
      return Text(
        'App Title: $appTitle', // Hiển thị tiêu đề của AppBar
        style: TextStyle(fontSize: 20),
      );
    }
  }
```

Trong ví dụ trên, chúng ta tạo một ứng dụng Flutter đơn giản với một tiêu đề **AppBar** và một **MyWidget** trong thân của **Scaffold**. Trong **MyWidget**, chúng ta sử dụng **BuildContext** để truy cập và hiển thị tiêu đề của **AppBar** của widget cha (**MyApp**).

#### 2.2.4.2. Một số đặc điểm chính của Build Context

- Vị trí của BuildContext: Mỗi widget trong cây widget của Flutter đều có một build context riêng. Build context cho biết widget đó đang nằm ở đâu trong cây widget và giúp Flutter xác định cách xây dựng cây và quản lý tối ưu việc cập nhật giao diện khi có thay đổi.
- Tính không thay đổi của BuildContext: BuildContext là một đối tượng không thể thay đổi. Điều này có nghĩa là khi bạn xây dựng cây widget, một build context sẽ không bao giờ thay đổi từ khi nó được tạo ra cho đến khi widget đó bị xóa khỏi cây widget. Nói cách khác, một widget không thể thay đổi build context của nó khi đang trong quá trình xây dựng.

- Sử dụng BuildContext trong hàm build():
  - Mỗi khi Flutter cần xây dựng lại giao diện của widget, nó sẽ gọi hàm build() tương ứng của widget đó và chuyển vào một build context.
  - Bạn sẽ sử dụng build context để tạo các widget và xây dựng cây widget của bạn. Ví dụ: sử dụng Scaffold, Container, Text,... là các widget được xây dựng bằng cách sử dụng build context.
- Sử dụng BuildContext để tìm và truy cập các widget khác:
  - Bên cạnh việc xây dựng giao diện, build context cũng được sử dụng để tìm và truy cập các widget khác trong cây widget.
  - BuildContext.inheritFromWidgetOfExactType<T>(): Bạn có thể sử dụng phương thức này để tìm và truy cập một widget có kiểu cụ thể (hoặc kiểu con của nó) từ widget hiện tại trở lên cây widget.
  - BuildContext.findAncestorWidgetOfExactType<T>(): Tương tự như inheritFromWidgetOfExactType, nhưng phương thức này trả về widget cha gần nhất có kiểu cụ thể (hoặc kiểu con của nó).
- Sử dụng BuildContext trong các tình huống khác: Trong các tình huống khác, như khi bạn muốn tạo một overlay, tạo dialog, hoặc gửi một sự kiện tới widget cha thông qua callback, build context cũng có thể được sử dụng.

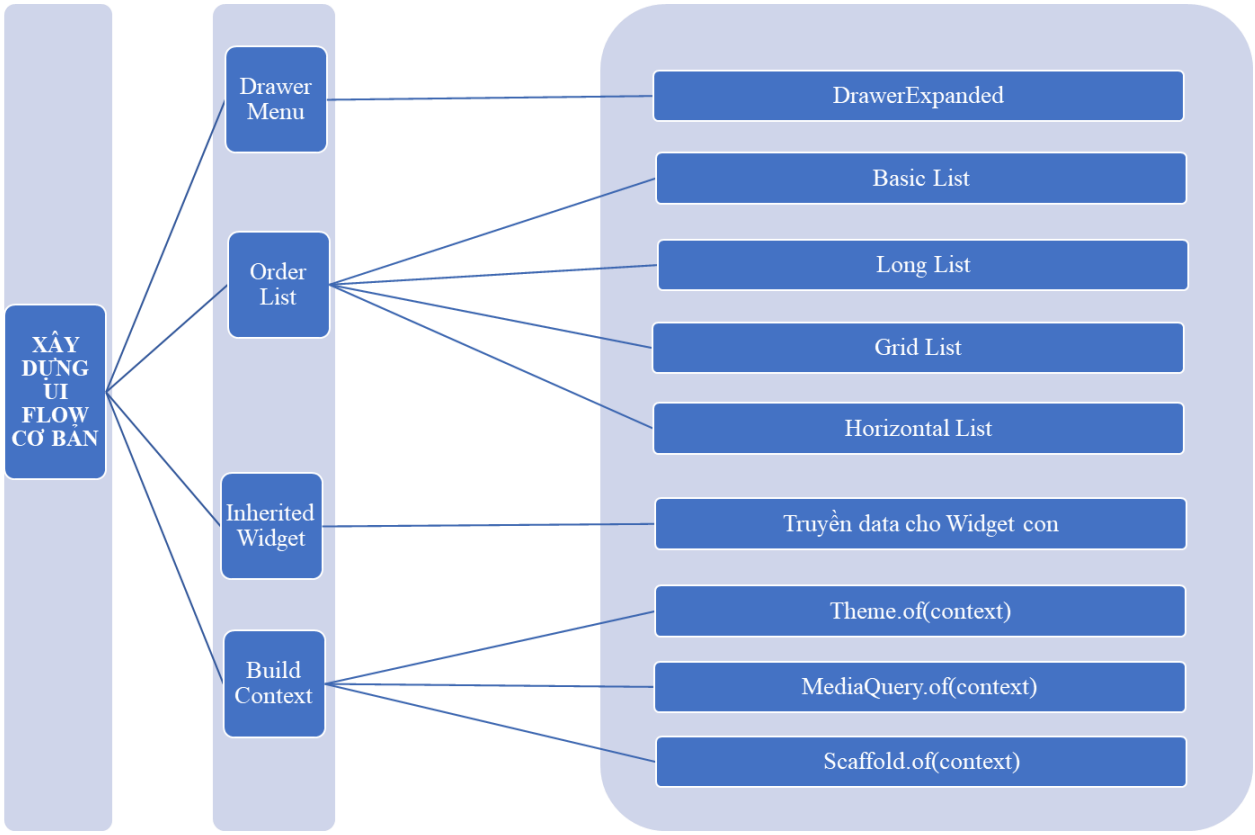
**Cảnh báo khi sử dụng BuildContext*

- *Trong quá trình phát triển ứng dụng, bạn cần chú ý tránh lưu trữ các build context dài hạn hoặc gửi chúng đi xa cây widget. Điều này có thể gây ra lỗi hoặc lãng phí bộ nhớ.*
- *Build context nên được sử dụng cẩn thận và chỉ trong phạm vi ngắn hạn, tức là trong quá trình xây dựng giao diện của một widget cụ thể.*

# CHƯƠNG 3: CÀI ĐẶT THỬ NGHIỆM

## 3.1. Phương pháp nghiên cứu

Đầu tiên, chúng em đã tiến hành thu thập thông tin về ứng dụng Netflix và các yêu cầu cơ bản của giao diện người dùng. Thông tin được lấy từ tài liệu tham khảo, trang web chính thức của Netflix, và cũng thông qua phỏng vấn một số người dùng mục tiêu để hiểu rõ hơn về các tính năng và ưu tiên của họ.




*Hình 3. 1: Sơ đồ nghiên cứu thực hiện*

Sau khi thu thập đủ thông tin, chúng em đã tiến hành phân tích yêu cầu để định rõ phạm vi và giới hạn của việc xây dựng UI flow. Bằng việc xác định rõ những chức năng chính cần có và đối tượng người dùng mục tiêu, chúng tôi đã thiết lập mục tiêu cụ thể cho việc nghiên cứu này.

Sau khi hoàn tất thiết kế, chúng em đã triển khai UI flow bằng Flutter. Mã nguồn được viết và kiểm thử kỹ lưỡng để đảm bảo rằng các tính năng hoạt động đúng và tương tác người dùng diễn ra một cách mượt mà. Các lỗi và vấn đề khác nhau đã được xác định và sửa chữa để cải thiện chất lượng của ứng dụng.

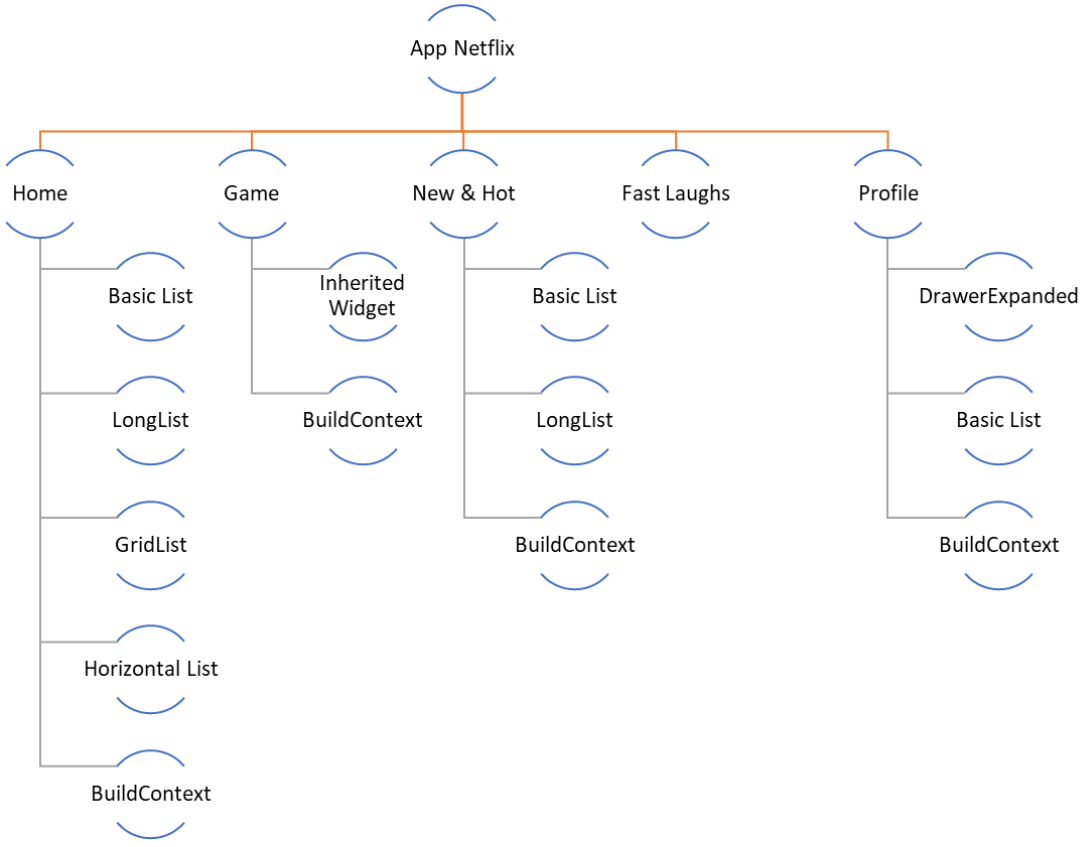
Chúng em đã so sánh UI flow mà chúng em xây dựng với ứng dụng Netflix gốc để đánh giá tính chính xác và tương đồng. Những điểm mạnh và yếu của UI flow cũng được đề cập để thấy rõ những gì đã đạt được và những điểm cần cải thiện trong quá trình nghiên cứu.

Phương pháp nghiên cứu trên đảm bảo tính toàn vẹn và độ tin cậy của kết quả nghiên cứu, đồng thời giúp chúng em có cái nhìn tổng quan và logic về việc xây dựng UI flow cơ bản cho ứng dụng Netflix trong Flutter.

### 3.2. Khảo sát chức năng hệ thống

App Netflit áp dụng các UI Flow cơ bản sử dụng ngôn ngữ dart với nền tảng công nghệ là Flutter cho phép chúng ta thực hiện các chức năng sau:

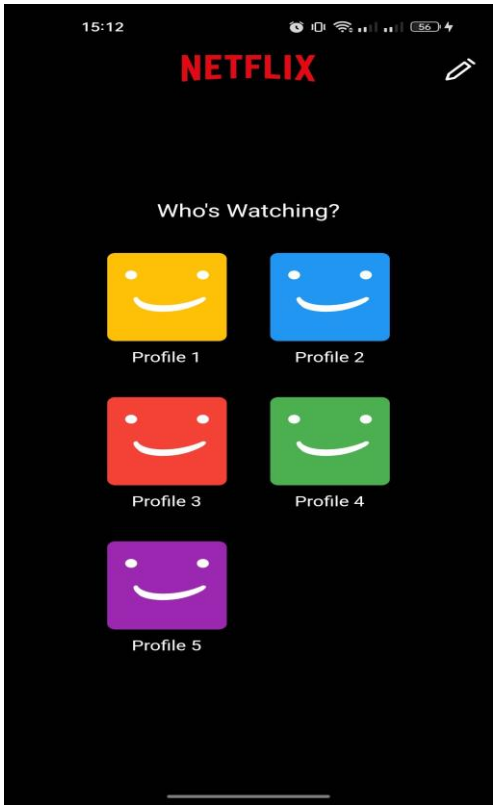
- Profile.
- Xem thông tin phim.
- Xem chi tiết phim.
- Top trending movie this weeks.
- Top trending movie today.
- Top trending TV-Shows this weeks.
- Top trending TV_Shows today
- Xem được những phim sắp chiếu.



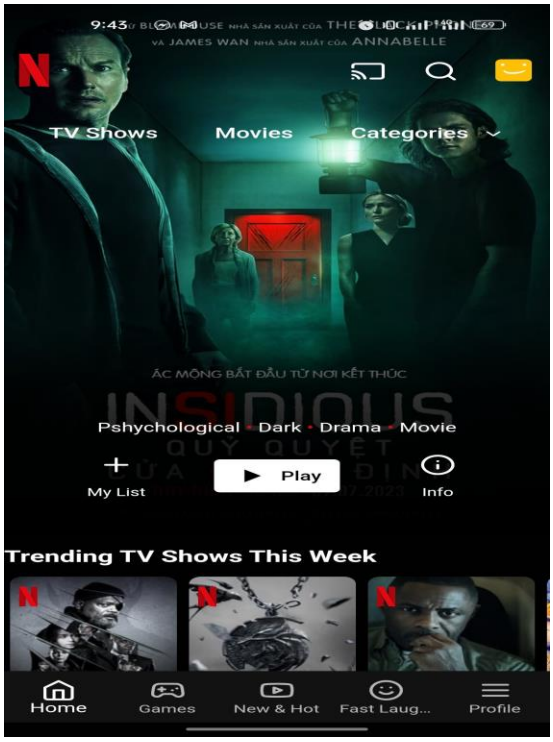

*Hình 3. 2: Sơ đồ chức năng nghiên cứu thực hiện áp dụng vào AppNetfix*

### 3.1 Kết quả đạt được

Với sự tìm hiểu và phân tích thu thập dữ liệu, cũng như là xây dựng thiết kế hệ thống và tìm hiểu về ngôn ngữ dart cũng như là flutter, thì nhóm em cũng đã áp dụng được ngôn ngữ dart và các UI Flow cơ bản vào ứng dụng Netflit. Gồm các chức năng sau.

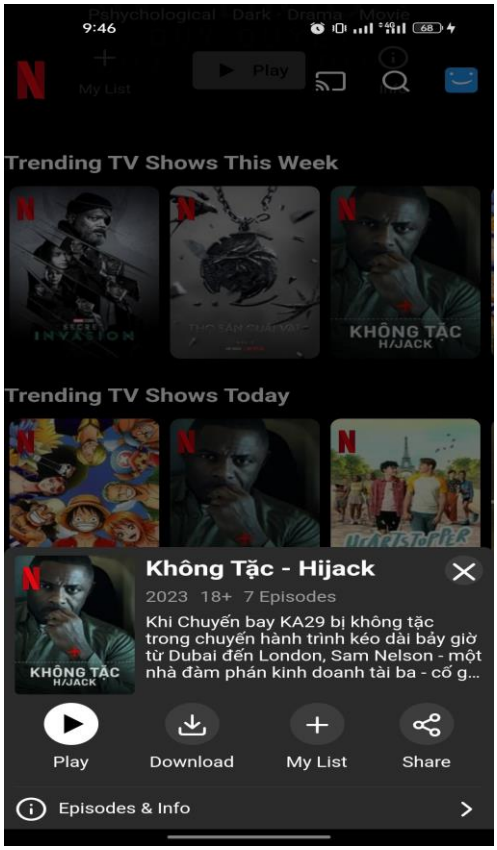



*Hình 3. 3 Giao diện chọn profile khi vào ứng dụng*

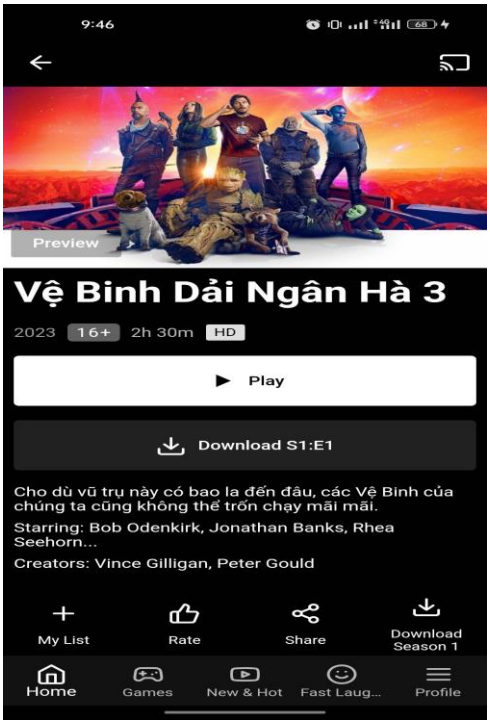



*Hình 3. 4 Màn hình chính khi vào ứng dụng*

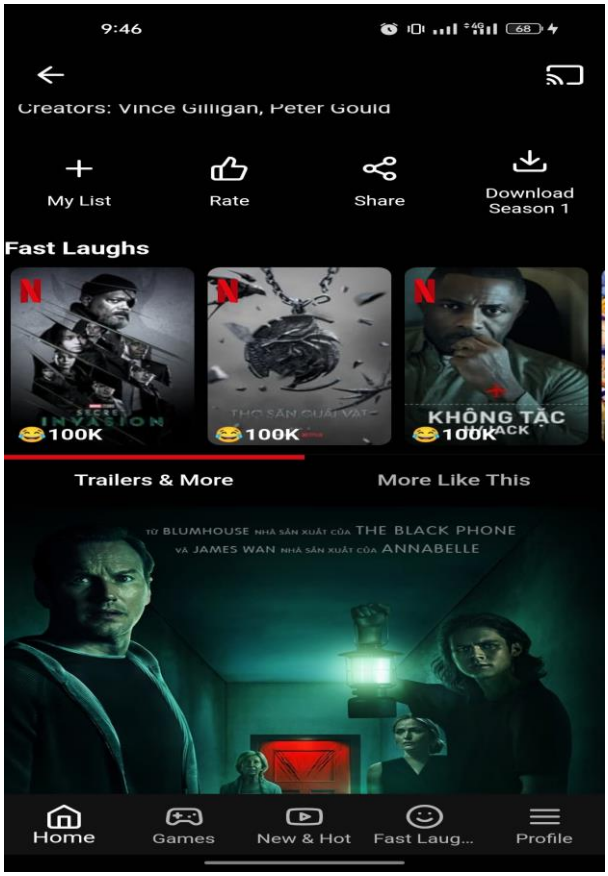

*Hình 3. 5 Giao diện Trending TV Shows today và movie this week*



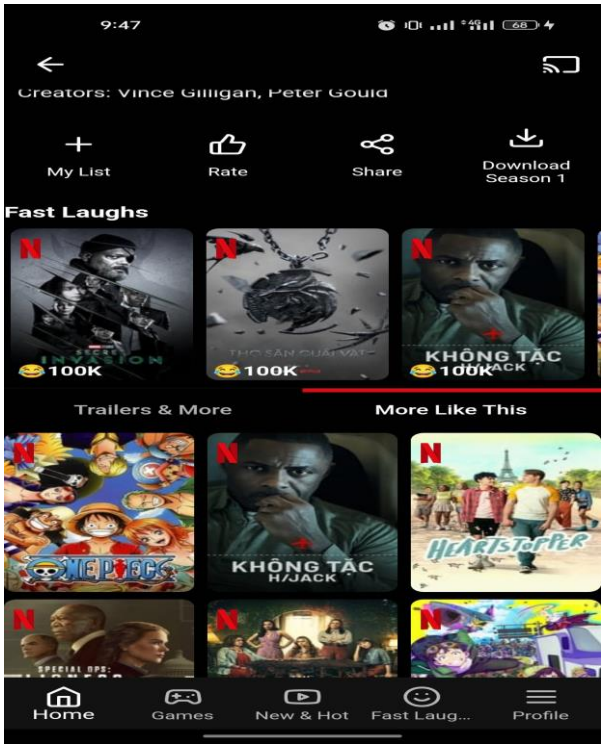

*Hình 3. 6 Giao diện xem trước thông tin phim*



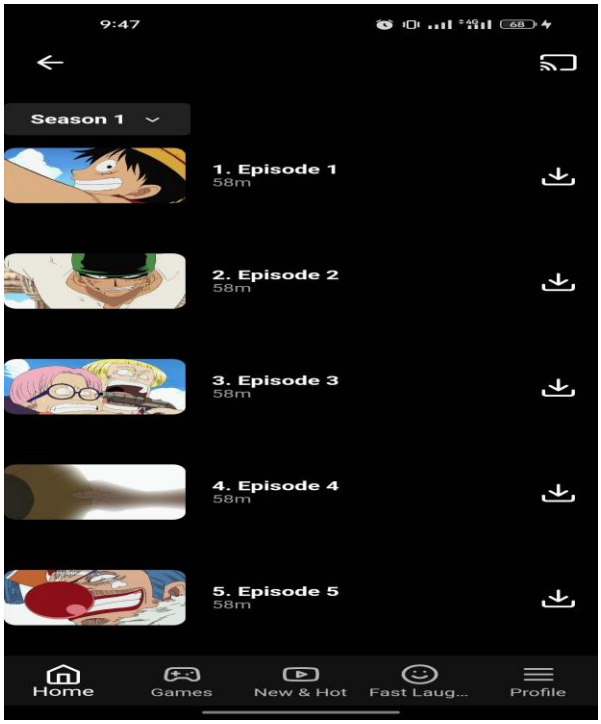

*Hình 3. 7 Giao diện khi bấm vào phim để xem chi tiết*



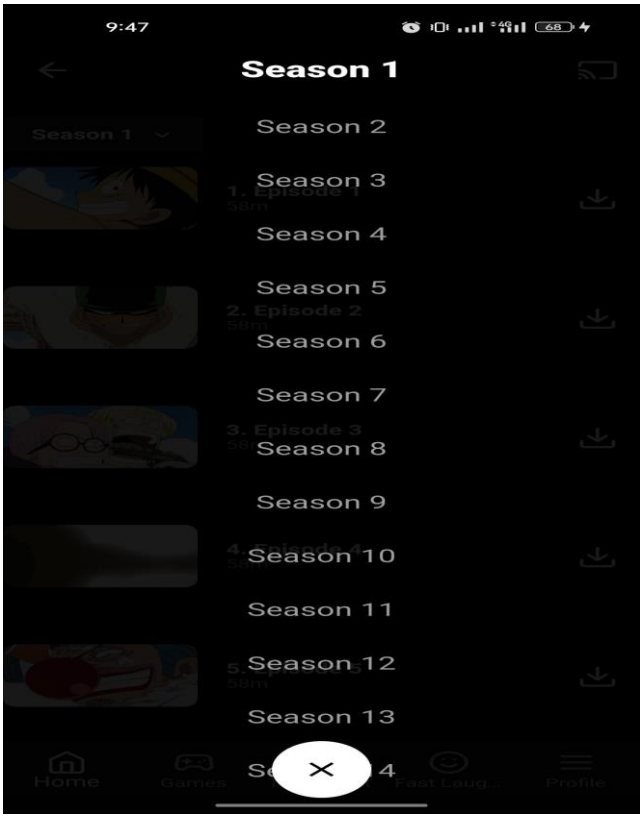

*Hình 3. 8 Giao diện trailers của phim*



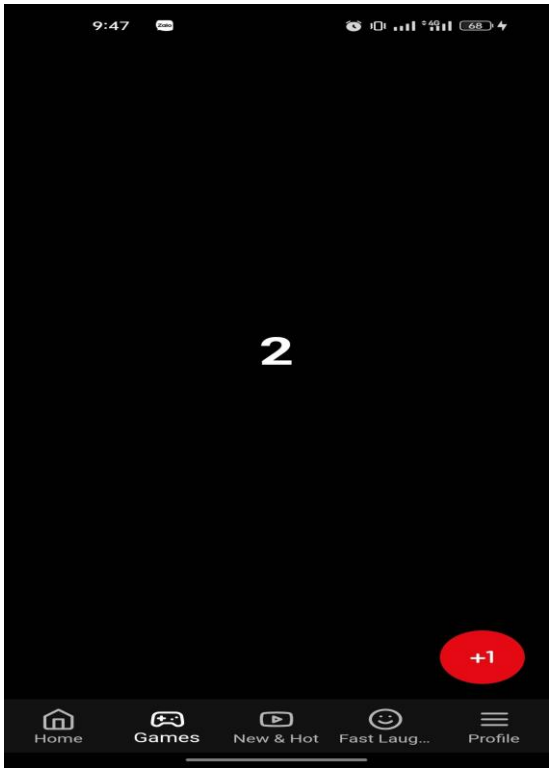

*Hình 3. 9 Giao diện những phim tương tự*




*Hình 3. 10 Giao diện các tập phim*

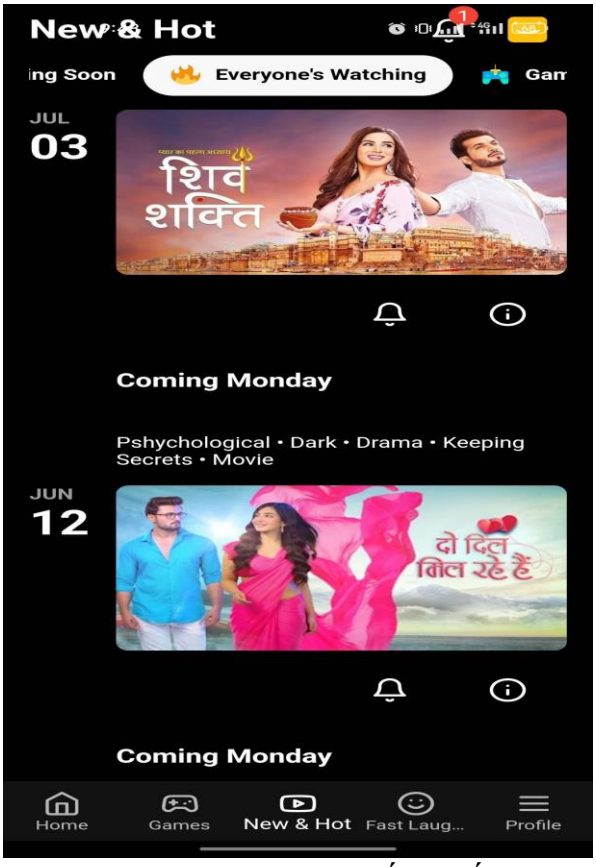



*Hình 3. 11 Giao diện drowdown season*

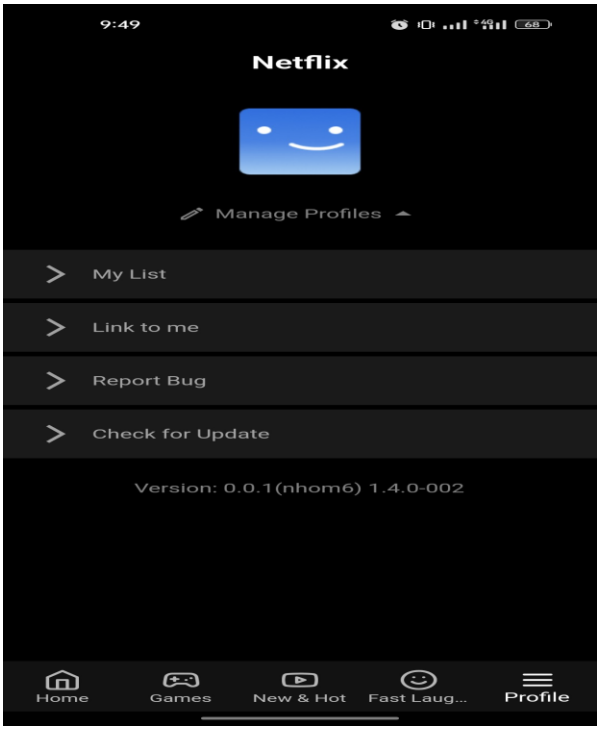



*Hình 3. 12 Giao diện demo inheritedWidget*


*Hình 3. 13 Giao diện các phim sắp chiếu trong tuần*




*Hình 3. 14 Giao diện các phim sắp chiếu trong tháng*




*Hình 3. 15 Giao diện profile drawer menu*

# CHƯƠNG 4: TỔNG KẾT

## 4.1. Kết quả đạt được

Khi hoàn thành đề tài "Xây dựng UI Flow cơ bản", chúng ta sẽ đạt được một số kết quả quan trọng và hữu ích trong việc nghiên cứu và ứng dụng Flutter và Dart vào xây dựng giao diện người dùng hiệu quả. Dưới đây là những kết quả chính mà chúng ta hy vọng sẽ đạt được:

- Hiểu biết sâu hơn về Flutter và Dart: Thông qua việc nghiên cứu và thực hành, chúng ta sẽ nắm vững kiến thức cơ bản về Flutter và Dart, hiểu rõ về tính đa nền tảng của Flutter và cú pháp đa dạng của Dart.
- Xây dựng giao diện người dùng đa dạng và tương tác: Chúng ta sẽ biết cách sử dụng các thành phần như Drawer Menu, List, Inherited Widget và Build Context để xây dựng giao diện đa dạng, tương tác và tối ưu cho ứng dụng. Điều này giúp cải thiện trải nghiệm người dùng và tăng tính hấp dẫn của ứng dụng.
- Tối ưu hóa hiệu năng và nhất quán của ứng dụng: Nhờ vào việc áp dụng các khái niệm và công nghệ của Flutter và Dart, chúng ta sẽ tối ưu hóa hiệu năng và đảm bảo tính nhất quán cho ứng dụng trên nhiều nền tảng, giúp ứng dụng hoạt động mượt mà và ổn định.
- Kỹ năng lập trình và phát triển ứng dụng tăng cường: Đề tài này giúp chúng ta rèn luyện kỹ năng lập trình và phát triển ứng dụng di động và web, từ đó nâng cao khả năng tạo ra các ứng dụng đa nền tảng chất lượng cao.
- Sáng tạo và ứng dụng thực tế: Sau khi hoàn thành đề tài, chúng ta sẽ có khả năng sáng tạo và xây dựng các ứng dụng thực tế với giao diện người dùng đa dạng và phong phú.

Kết quả đạt được từ đề tài này không chỉ giúp chúng ta nắm vững kiến thức cơ bản về Flutter và Dart mà còn trở thành nền tảng để phát triển và cải thiện các ứng dụng di động và web trong tương lai.

## 4.2. Đánh giá đề tài

### 4.2.1. Hạn chế

Trong quá trình thực hiện đề tài "Xây dựng UI Flow cơ bản" với các phần "List", "DrawerMenu", "InheritedWidget", và "BuildContext", có một số hạn chế mà bạn có thể gặp phải:

- Phạm vi hạn chế của đề tài: Đề tài "Xây dựng UI Flow cơ bản" có thể giới hạn về phạm vi và không đủ thời gian để khám phá sâu hơn về các khía cạnh phức tạp của các thành phần như "InheritedWidget". Bạn có thể chỉ có thể giới thiệu một phần cơ bản về chúng mà không đủ thời gian để đi vào chi tiết sâu hơn.
- Khả năng tương thích và phiên bản: Các công nghệ và thư viện trong việc xây dựng UI có thể thay đổi theo thời gian và có thể không tương thích với các phiên bản mới. Điều này có thể tạo ra khó khăn trong việc duy trì và cập nhật dự án sau này.
- Khả năng hiểu biết về UI/UX design: Xây dựng UI không chỉ liên quan đến việc triển khai code, mà còn đòi hỏi hiểu biết về thiết kế giao diện và trải nghiệm người dùng. Nếu bạn không có kiến thức sâu về thiết kế UI/UX, có thể dẫn đến giao diện không thân thiện hoặc khó sử dụng.
- Tương tác phức tạp: Nếu dự án yêu cầu các tương tác phức tạp giữa các phần, ví dụ như tương tác giữa "List" và "DrawerMenu", việc quản lý logic và tương tác có thể trở nên phức tạp và khó khăn để bảo trì.
- Hiệu năng và tương thích: Xây dựng UI có thể ảnh hưởng đến hiệu năng của ứng dụng và có thể cần tối ưu hóa để đảm bảo hoạt động mượt mà trên nhiều thiết bị và kích thước màn hình khác nhau.
- Khả năng tương tác với dữ liệu thực tế: Trong thực tế, việc xây dựng giao diện thường liên quan đến tương tác với dữ liệu thực tế từ nguồn dữ liệu như cơ sở dữ liệu hoặc API. Việc làm việc với dữ liệu có thể tạo ra thách thức riêng và yêu cầu kiến thức về quản lý trạng thái và xử lý lỗi.
- Khả năng debug và sửa lỗi: Khi xây dựng giao diện, sẽ có thời điểm gặp lỗi hoặc vấn đề trong quá trình phát triển. Khả năng debug và sửa lỗi trong các thành phần

như "InheritedWidget" và "BuildContext" có thể đòi hỏi kiến thức sâu về cách hoạt động của chúng.

- Hạn chế về học tập và thực hành: Nếu bạn mới bắt đầu với việc xây dựng UI, việc hiểu và áp dụng các khái niệm có thể đòi hỏi thời gian và nỗ lực học tập.

#### 4.2.2. Ưu điểm

Những ưu điểm quan trọng và giá trị của việc nghiên cứu và xây dựng UI Flow cơ bản bằng Flutter và Dart. Dưới đây là những ưu điểm chính mà chúng ta đã thu được từ đề tài này:

- Đa nền tảng và linh hoạt: Sử dụng Flutter và Dart cho việc xây dựng UI Flow cơ bản cho ứng dụng di động và web giúp ta tiết kiệm thời gian và công sức bằng cách viết mã một lần và triển khai trên nhiều nền tảng khác nhau. Điều này tăng tính linh hoạt và giảm chi phí phát triển và bảo trì ứng dụng.
- Trải nghiệm người dùng tối ưu: Nhờ vào các thành phần giao diện như Drawer Menu, List, Inherited Widget và Build Context, ta có thể xây dựng giao diện tương tác và đa dạng, cải thiện trải nghiệm người dùng và tăng cường tính hấp dẫn của ứng dụng.
- Hiệu năng và ổn định cao: Flutter với cú pháp Dart giúp tối ưu hóa hiệu năng và đảm bảo ứng dụng hoạt động mượt mà và ổn định trên nhiều nền tảng khác nhau.
- Tiện ích và phong phú: Dart có một hệ sinh thái thư viện phong phú, cùng với tích hợp các thành phần giao diện của Flutter, giúp ta xây dựng các ứng dụng phong phú với nhiều tính năng hấp dẫn và tiện ích.

### 4.2.3. Hướng phát triển

Hướng phát triển của đề tài "Xây dựng UI Flow cơ bản" bằng Flutter và Dart có thể bao gồm các bước và phương pháp sau đây:

1. Để phát triển đề tài này, chúng ta có thể mở rộng phạm vi nghiên cứu để tìm hiểu sâu hơn về các tính năng và chức năng nâng cao của Flutter và Dart. Chúng ta có thể tìm hiểu về các chủ đề như animations, responsive UI, localization, và state management để tăng cường tính năng và hiệu quả của ứng dụng.
2. Để ứng dụng các kiến thức và kỹ năng đã học, chúng ta có thể thực hiện các dự án ứng dụng thực tế. Điều này giúp rèn luyện kỹ năng lập trình và phát triển ứng dụng di động và web, cũng như hiểu rõ hơn về quá trình phát triển ứng dụng từ đầu đến cuối.
3. Tiếp tục tối ưu hiệu năng và trải nghiệm người dùng (UX) của ứng dụng bằng cách sử dụng các công nghệ và kỹ thuật mới của Flutter và Dart. Tìm hiểu về cách cải thiện hiệu suất và đáp ứng nhanh hơn, giúp ứng dụng hoạt động mượt mà và tốt hơn trên mọi nền tảng.
4. Tiến hành phân tích và đánh giá kết quả của việc xây dựng UI Flow cơ bản bằng Flutter và Dart. So sánh các kết quả đạt được với các tiêu chí và mục tiêu đã đề ra trong đề tài, từ đó đánh giá hiệu quả và khả năng ứng dụng trong thực tế.
5. Hướng phát triển cũng có thể bao gồm việc tìm hiểu thêm về các công nghệ liên quan như Firebase, GraphQL, Redux, và các thư viện bên thứ ba khác. Điều này giúp ta mở rộng kiến thức và tùy chỉnh ứng dụng một cách linh hoạt và phong phú hơn.
6. Hướng phát triển này cũng có thể bao gồm việc đóng góp vào cộng đồng Flutter và Dart bằng cách tham gia vào việc viết mã nguồn mở, tạo các dự án mã nguồn mở, và hỗ trợ cộng đồng lập trình thông qua việc trả lời câu hỏi và đóng góp kiến thức.

Tóm lại, việc phát triển đề tài "Xây dựng UI Flow cơ bản" bằng Flutter và Dart là một bước khởi đầu tuyệt vời để mở ra nhiều cơ hội và tiềm năng trong việc nghiên cứu và phát triển các ứng dụng di động và web đa nền tảng. Từ đó, chúng ta có thể tiếp tục mở rộng kiến thức, rèn luyện kỹ năng, và tạo ra những ứng dụng đáp ứng nhu cầu và mong đợi của người dùng trong thời đại số hóa ngày nay.

# TÀI LIỆU THAM KHẢO

[1] An toàn hệ thống thông tin https://nam.name.vn/bai-1-gioi-thieu-ve-an-toan-va-bao-mat-thong-tin.html#ftoc-heading-6

[2] Quy chế đào tạo trình độ đại học https://thuvienphapluat.vn/van-ban/Giao-duc/Thong-tu-08-2021-TT-BGDDT-Quy-che-dao-tao-trinh-do-dai-hoc-470013.aspx

[3] Giải pháp bảo mật HTTT https://cypresscom.vn/tin-tuc-1/giai-phap-bao-mat-thong-tin.html

[4] Xác thực thông tin https://m.antoanthongtin.vn/giai-phap-khac/xac-thuc-thong-tin-100042

[5] An toàn HTTT https://knacert.com.vn/blogs/tin-tuc/isms-la-gi-tim-hieu-he-thong-quan-ly-an-toan-thong-tin

[6] Văn bản chính sách mới https://moj.gov.vn/qt/tintuc/Pages/van-ban-chinh-sach-moi.aspx?ItemID=3739

[7] Luật trong bảo mật HTTT https://thuvienphapluat.vn/phap-luat/trong-he-thong-thong-tin-cap-do-5-viec-xay-dung-chinh-sach-an-toan-thong-tin-bao-gom-nhung-noi-dung-323988-42286.html

[8] Chính Phủ. (2017, 05 25). *Thư viện pháp luật.* Retrieved from Quy định chức năng, nhiệm vụ, quyền hạn và cơ cấu tổ chức của bộ giáo dục và đào tạo: https://thuvienphapluat.vn/van-ban/Bo-may-hanh-chinh/Nghi-dinh-69-2017-ND-CP-chuc-nang-nhiem-vu-quyen-han-co-cau-to-chuc-Bo-Giao-duc-va-Dao-tao-350206.aspx

[9] Giang, L. T. (2022, 03 09). *Thư viện pháp luật.* Retrieved from Giáo viên mầm non dạy tại trường công lập quy định giờ dạy như thế nào: https://thuvienphapluat.vn/phap-luat/giao-vien-mam-non-day-tai-truong-cong-lap-duoc-quy-dinh-gio-day-nhu-the-nao-578.html

[10] Hoa, L. K. (2023, 01 25). *Công ty luật TNHH Minh Khuê*. Retrieved from Chính sách miễn, giảm học phí, hỗ trợ phí học tập từ năm 2023: https://luatminhkhue.vn/chinh-sach-mien-giam-hoc-phi-ho-tro-phi-hoc-tap.aspx#3-doi-tuong-duoc-min-hoc-phi

[11] Luật Việt Nam. (2020, 10 26). *luatvietnam.vn.* Retrieved from Thông tư 40/2020/TT-BGDĐT xếp lương viên chức giảng dạy trong đại học công lập: https://luatvietnam.vn/giao-duc/thong-tu-40-2020-xep-luong-vien-chuc-giang-day-trong-dai-hoc-cong-lap-192983-d1.html

[12] Nguyễn Thị Nghĩa, Trương Chí Trung. (2014, 10 15). *Chinhphu.vn*. Retrieved from Thông tư liên tịch số 35/2014/TTLT-BGDĐT-BTC của Bộ Tài chính, Bộ Giáo dục và Đào tạo: Hướng dẫn thực hiện Quyết định số 66/2013/QĐ-TTg ngày 11 tháng 11 năm 2013 của Thủ tướng Chính phủ Quy định chính sách hỗ trợ chi phí học tập đối với sinh viên là người: https://vanban.chinhphu.vn/default.aspx?pageid=27160&docid=177426

[13] Nguyễn Thiện Nhân. (2013, 11 11). Chinhphu.vn. Retrieved from Quyết định số 66/2013/QĐ-TTg của Thủ tướng Chính phủ: Quy định chính sách hỗ trợ chi phí học tập đối với sinh viên là người dân tộc thiểu số học tại các cơ sở giáo dục đại học: https://vanban.chinhphu.vn/default.aspx?pageid=27160&docid=170795

[14] Nguyễn, Q. (2019, 11 5). *Thư viện pháp luật*. Retrieved from chính sách pháp luật mới: https://thuvienphapluat.vn/chinh-sach-phap-luat-moi/vn/thong-bao-van-ban-moi/email/25928/toan-bo-vb-ve-luong-phu-cap-va-che-do-lam-viec-cua-giao-vien

[15] Nhung, L. N. (2022, 7 30). *Thư viện pháp luật.* Retrieved from Bảng lương mới của giáo viên các cấp và chi tiết cách xếp lương năm 2022? Khi nào giáo viên được tăng lương?: https://thuvienphapluat.vn/phap-luat/thoi-su-phap-luat/bang-luong-moi-cua-giao-vien-cac-cap-va-chi-tiet-cach-xep-luong-nam-2022-khi-nao-giao-vien-duoc-tan-29155.html?rel=tbvbmemail

[16] *Phạm Thanh Hữu, T. T. (2023, 04 11). Thư viện pháp luật. Retrieved from Tiêu chuẩn Hiệu trưởng trường trung học phổ thông mới nhất: https://thuvienphapluat.vn/chinh-sach-phap-luat-moi/vn/thoi-su-phap-luat/tu-van-phap-luat/47736/tieu-chuan-hieu-truong-truong-trung-hoc-pho-thong-moi-nhat*

[17] Quốc hội. (2015, 11 19). *Chinhphu.vn*. Retrieved from Luật số 86/2015/QH13 của Quốc hội: Luật An toàn thông tin mạng: https://vanban.chinhphu.vn/default.aspx?pageid=27160&docid=183196

[18] Thư viện pháp luật. (2013, 12 31). *thư viện pháp luật*. Retrieved from Thông tư liên lịch 42/2013/TTLT-BGDĐT-BLĐTBXH-BTC quy định chính sách về giáo dục đối với người khuyết tật: https://thuvienphapluat.vn/van-ban/Giao-duc/Thong-tu-lien-tich-42-2013-TTLT-BGDDT-BLDTBXH-BTC-chinh-sach-giao-duc-nguoi-khuyet-tat-220707.aspx

[19] Thư viện pháp luật. (2013, 8 21). *Thư viện pháp luật.* Retrieved from Nghị định 94/2013/NĐ-CP chi tiết thi hành pháp luật dự trữ quốc gia: https://thuvienphapluat.vn/van-ban/Dich-vu-phap-ly/Nghi-dinh-94-2013-ND-CP-chi-tiet-thi-hanh-Luat-Du-tru-quoc-gia-205036.aspx

[20] Thư viện pháp luật. (2016, 7 15). Thư viện pháp luật. Retrieved from Thông tư 23/2016/TT-BLĐTBXH hướng dẫn điều chỉnh mức lương hưu, trợ cấp bảo hiểm xã hội: Thong-tu-23-2016-TT-BLDTBXH-huong-dan-dieu-chinh-luong-huu-tro-cap-bao-hiem-xa-hoi-55-2016-ND-CP-318467

[21] Thư viện pháp luật. (2020, 9 25). *Thư viện pháp luật*. Retrieved from Nghị định 115/2020/NĐ-CP quy định về tuyển dụng, sử dụng và quản lý viên chức: https://thuvienphapluat.vn/van-ban/Bo-may-hanh-chinh/Nghi-dinh-115-2020-ND-CP-tuyen-dung-su-dung-quan-ly-vien-chuc-453968.aspx

[22] Thư viện pháp luật. (2020, 12 10). *Thư viện pháp luật*. Retrieved from Quyết định 1066/QD-BNV 2020 công bố thủ tục hành chính tại nghị định số 115/2020/NĐ-CP: https://thuvienphapluat.vn/van-ban/Bo-may-hanh-chinh/Quyet-dinh-1066-QD-BNV-2020-cong-bo-thu-tuc-hanh-chinh-quy-dinh-tai-Nghi-dinh-115-2020-ND-CP-459151.aspx

[23] Thư viên pháp luật. (2021, 03 18). *Thư viện pháp luật*. Retrieved from Ban hành quy chế đào tạo trình độ đại học: https://thuvienphapluat.vn/van-ban/Giao-duc/Thong-tu-08-2021-TT-BGDDT-Quy-che-dao-tao-trinh-do-dai-hoc-470013.aspx

[24] Tuấn, P. Đ. (2014, 10 30). Retrieved from Quyết định 709/QĐ-TĐHTPHCM về việc ban hành quy chế đào tạo đại học và cao đẳng hệ chính quy theo hệ thống tín chỉ: https://hcmunre.edu.vn/Files/Home/TrangSinhVien/QuyDinh-QuyChe/Q%C4%90%20709%20Vv%20ban%20h%C3%A0nh%20Quy%20ch%E1%BA%BF%20%C4%91%C3%A0o%20t%E1%BA%A1o%20%C4%91%E1%BA

%A1i%20h%E1%BB%8Dc%20v%C3%A0%20cao%20%C4%91%E1%BA%B3ng%20h%E1%BB%87%20ch%C3%ADnh%20quy%20th

[25] Yến, N. (2021). Studocu. Retrieved from BT CHƯƠNG 4 NHÓM 17 - ôn tập an toàn thông tin 2021 2022: https://www.studocu.com/vn/document/truong-dai-hoc-cong-nghiep-thanh-pho-ho-chi-minh/an-toan-thong-tin/bt-chuong-4-nhom-17-on-tap-an-toan-thong-tin-2021-2022/29084420?origin=home-recent-1

## DANH MỤC CÂU HỎI

| | |
|---|---|
| Nhóm 1 | Làm thế nào để truy cập build context trong một widget? |
| Nhóm 2 | Hãy cho biết về cách bạn sẽ sử dụng Build Context để quản lý trạng thái và xây dựng giao diện trong ứng dụng của bạn |
| Nhóm 3 | Ưu và nhược điểm của Drawer Menu |
| Nhóm 4 | Có những phương thức nào để sắp xếp 1 list trong Dart |
| Nhóm 5 | Khi nào cần tránh sử dụng build context trong số trường hợp nhất định ? |
| Nhóm 7 | Làm thế nào để sắp xếp các phần tử trong một danh sách theo một tiêu chí cụ thể? |
| Nhóm 8 | Cách inheritied widget chia sẽ dữ liệu với các widget con |
| Nhóm 9 | Tại sao chúng ta cần build context trong widget tree? |