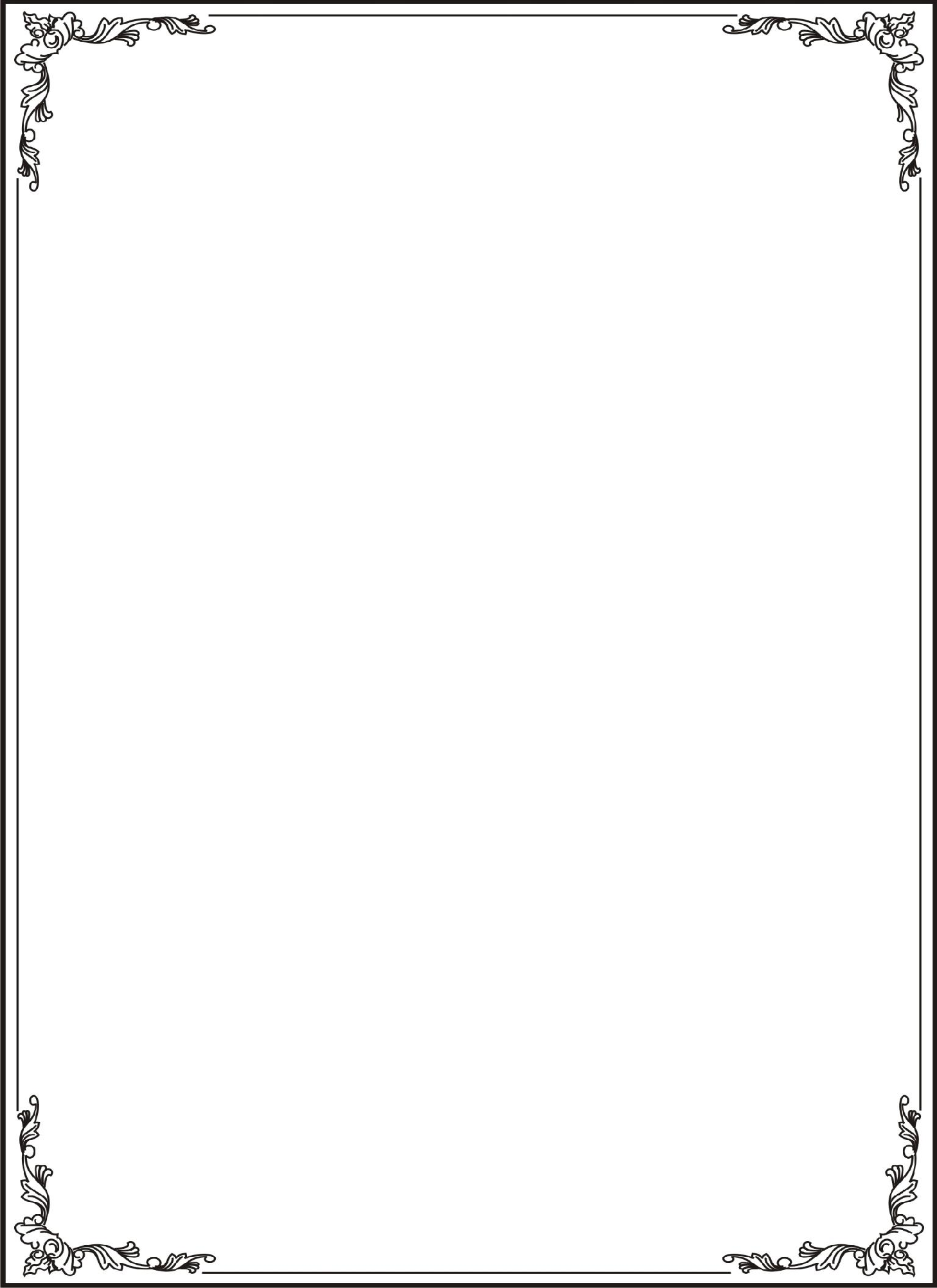
**BỘ TÀI NGUYÊN VÀ MÔI TRƯỜNG**

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

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



A logo with a person holding a book

Description automatically generated

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**MÔN: CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI ĐỘNG**

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG TODO CƠ BẢN**

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

Sinh viên thực hiện: **Đoàn Dzoãn Tường Anh**

**Thái Bình Dương**

**Phạm Thị Thảo Huyền**

**Lê Thị Bảo Yến**

Mã số sinh viên: **0850080002**

**0850080013**

**0850080024**

**0850080057**

Lớp**: 08\_ĐH\_CNPM**

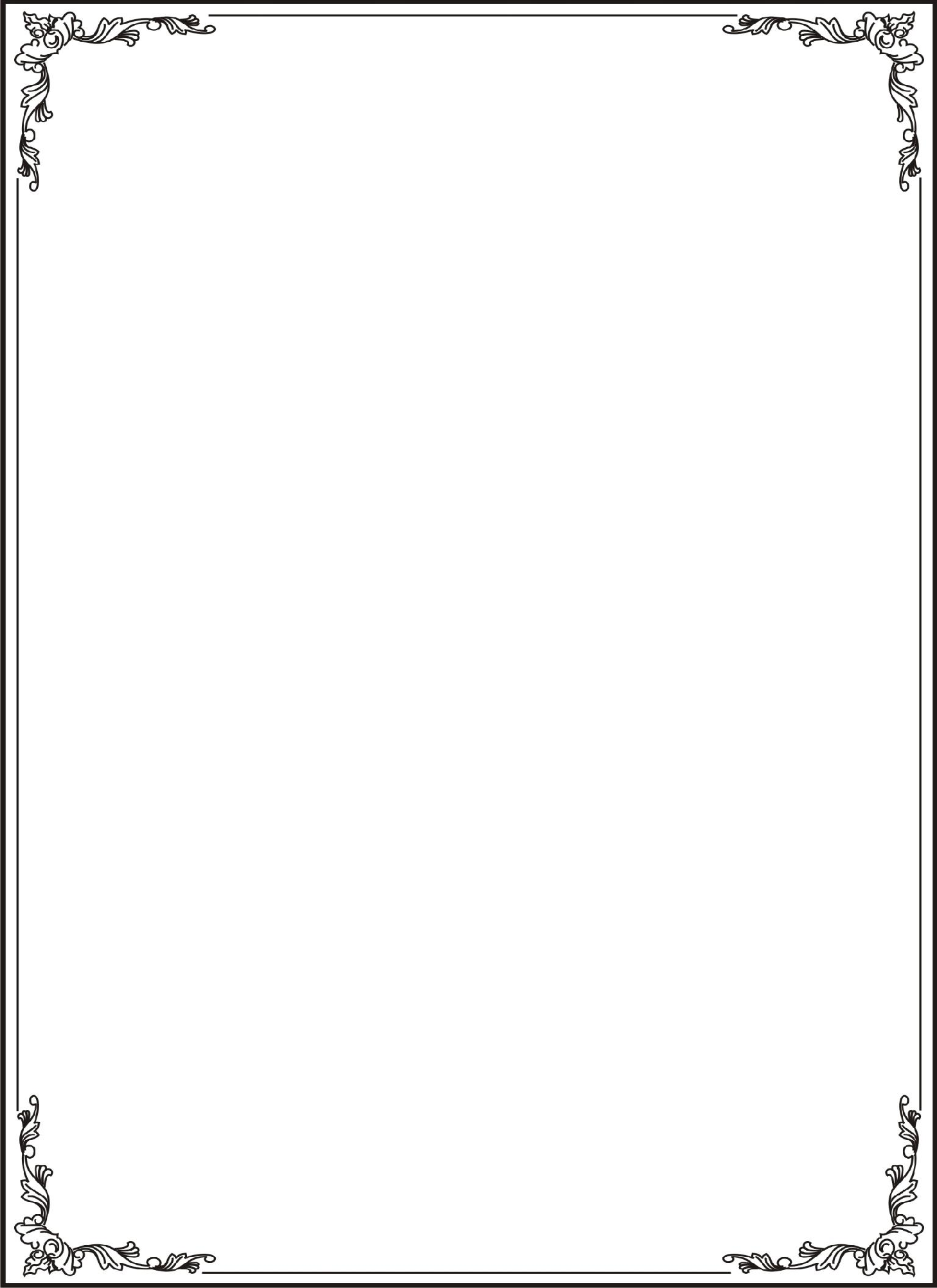
Khoá**: 08**

***TP. Hồ Chí Minh, tháng 08 năm 2023***

**BỘ TÀI NGUYÊN VÀ MÔI TRƯỜNG**

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

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



A picture containing text, logo, emblem, trademark

Description automatically generated

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**MÔN: CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI ĐỘNG**

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG TODO CƠ BẢN**

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

Sinh viên thực hiện: **Đoàn Dzoãn Tường Anh**

**Thái Bình Dương**

**Phạm Thị Thảo Huyền**

**Lê Thị Bảo Yến**

Mã số sinh viên: **0850080002**

**0850080013**

**0850080024**

**0850080057**

Lớp**: 08\_ĐH\_CNPM**

Khoá**: 08**

***TP. Hồ Chí Minh, tháng 08 năm 2023***

**MỞ ĐẦU**

Trong thời đại công nghệ phát triển nhanh chóng ngày nay, việc xây dựng ứng dụng di động đáp ứng yêu cầu ngày càng cao là một thách thức không nhỏ đối với các nhà phát triển. Đặc biệt, quản lý trạng thái và mô hình thiết kế trong ứng dụng di động đóng vai trò quan trọng để đảm bảo tính linh hoạt, mở rộng và hiệu suất của ứng dụng.

Đề tài "Xây dựng ứng dụng ToDo cơ bản" đã được chọn để tìm hiểu và áp dụng các phương pháp quản lý trạng thái và mô hình thiết kế trong Flutter, giúp đảm bảo rằng ứng dụng của chúng ta không chỉ đáp ứng được yêu cầu của người dùng mà còn đạt được hiệu suất tốt và dễ dàng mở rộng.

Trong đề tài này, chúng ta sẽ trải qua một cuộc hành trình qua các khái niệm và công nghệ cốt lõi của Flutter như Provider, State Management và BLoC Pattern. Chúng ta sẽ bắt đầu với việc xây dựng một ứng dụng ToDo đơn giản, sau đó áp dụng Provider để quản lý trạng thái của ứng dụng. Tiếp theo, chúng ta sẽ khám phá các phương pháp quản lý trạng thái khác như StatefulWidget, InheritedWidget và ValueNotifier để đạt được tính linh hoạt và mở rộng trong ứng dụng của chúng ta. Cuối cùng, chúng ta sẽ học cách áp dụng BLoC Pattern để tách biệt logic và giao diện người dùng trong ứng dụng, tạo ra một cấu trúc dễ bảo trì và sử dụng.

**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……

NGƯỜI NHẬN XÉT

*(ký tên)*

**MỤC LỤC**

[**CHƯƠNG 1: TỔNG QUAN 1**](#_Toc142598382)

[**1.1 Giới thiệu về đề tài 1**](#_Toc142598383)

[1.1.1 Giới thiệu chung 1](#_Toc142598384)

[**1.2 Phạm vi đề tài 2**](#_Toc142598385)

[**CHƯƠNG 2: CƠ SỞ LÝ LUẬN 3**](#_Toc142598386)

[**2.1 Lý thuyết 3**](#_Toc142598387)

[**2.2 Kỹ thuật 3**](#_Toc142598388)

[2.2.1 Giới thiệu về Flutter 3](#_Toc142598389)

[2.2.2 Giới thiệu về Android Studio 4](#_Toc142598390)

[2.2.3 Giới thiệu về Provider 9](#_Toc142598391)

[2.2.4 Giới thiệu về State Management 10](#_Toc142598392)

[2.2.5 Giới thiệu về BLoC Pattern 11](#_Toc142598393)

[2.2.6 Giới thiệu về ngôn ngữ Dart 12](#_Toc142598394)

[**CHƯƠNG 3: CÀI ĐẶT THỬ NGHIỆM 17**](#_Toc142598395)

[**3.1 Cài đặt triển khai ứng dụng 17**](#_Toc142598396)

[**3.2 Kết quả 24**](#_Toc142598397)

[**CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 30**](#_Toc142598398)

[**4.1 Kết luận 30**](#_Toc142598399)

[**4.2 Hạn chế 31**](#_Toc142598400)

[**4.3 Hướng phát triển 31**](#_Toc142598401)

[**DANH MỤC TÀI LIỆU THAM KHẢO 33**](#_Toc142598402)

**DANH MỤC HÌNH ẢNH**

[Hình 1 Sơ đồ sự khác biệt giữa Ephemeral State và App State 11](#_Toc142598338)

[Hình 2 BLoC Pattern 12](#_Toc142598339)

[Hình 3 Dart Features 14](#_Toc142598340)

[Hình 4 Các Dependencies 17](#_Toc142598341)

[Hình 5 Xây dựng các lớp Entities 19](#_Toc142598342)

[Hình 6 Xây dựng BLoC Pattern 20](#_Toc142598343)

[Hình 7 Luồng xử lý 20](#_Toc142598344)

[Hình 8 Các Services 21](#_Toc142598345)

[Hình 9 Triển khai API 23](#_Toc142598346)

[Hình 10 Hiển thị dữ liệu 23](#_Toc142598347)

[Hình 11 Xây dựng Router di chuyển 24](#_Toc142598348)

[Hình 12 Màn hình chính 25](file:///C:\Users\Admin\Documents\Zalo%20Received%20Files\Nhóm%207%20Mobile%20(1).docx#_Toc142598349)

[Hình 13 Màn hình các mục chưa hoàn thành 26](#_Toc142598350)

[Hình 14 Màn hình các mục đã hoàn thành 27](#_Toc142598351)

[Hình 15 Màn hình thêm công việc mới 28](#_Toc142598352)

[Hình 16 Màn hình hoàn thành công việc vừa tạo 29](#_Toc142598353)

# **C****HƯƠNG 1: TỔNG QUAN**

* 1. **Giới thiệu về đề tài**

Đề tài "Xây dựng ứng dụng ToDo cơ bản" nhằm mục đích tạo ra một ứng dụng Note đơn giản trong Flutter, áp dụng các phương pháp quản lý trạng thái và mô hình thiết kế để nâng cao hiệu suất và khả năng mở rộng của ứng dụng.

Trong đề tài này, chúng em sẽ sử dụng Provider, một gói phần mềm quản lý trạng thái trong Flutter, để quản lý trạng thái của ứng dụng. Provider giúp chúng ta tổ chức và chia sẻ các trạng thái trong toàn bộ ứng dụng một cách hiệu quả.

Ngoài ra, chúng em cũng sẽ sử dụng State Management để quản lý trạng thái trong ứng dụng. Flutter cung cấp nhiều phương pháp quản lý trạng thái như StatefulWidget, InheritedWidget và ValueNotifier. Trong dự án này, chúng em sẽ tìm hiểu và áp dụng các phương pháp này để quản lý trạng thái của các Note trong ứng dụng.

Cuối cùng, chúng em sẽ áp dụng BLoC (Business Logic Component) Pattern để quản lý logic của ứng dụng. BLoC Pattern giúp chúng ta tách biệt logic ứng dụng và giao diện người dùng, từ đó tạo ra một ứng dụng dễ dàng kiểm thử và mở rộng.

Trên cơ sở các kiến thức cơ bản về Flutter, dự án "Xây dựng ứng dụng ToDo cơ bản " giúp nắm vững các kỹ thuật quản lý trạng thái và mô hình thiết kế trong Flutter, từ đó xây dựng được các ứng dụng Flutter chất lượng cao và dễ dàng bảo trì.

* + 1. **Giới thiệu chung**

Trong đề tài này, chúng em sẽ tạo ra một ứng dụng ToDo đơn giản sử dụng Flutter và áp dụng các phương pháp quản lý trạng thái và mô hình thiết kế để làm cho ứng dụng trở nên mạnh mẽ và dễ dàng mở rộng.

Cụ thể, chúng em sẽ sử dụng Provider để quản lý trạng thái của ứng dụng. Provider giúp chúng ta tổ chức và chia sẻ các trạng thái trong toàn bộ ứng dụng một cách hiệu quả, từ đó tăng tính ổn định và đáng tin cậy của ứng dụng.

Ngoài ra, chúng em cũng sẽ sử dụng các phương pháp quản lý trạng thái khác như StatefulWidget, InheritedWidget và ValueNotifier để đạt được mục tiêu quản lý trạng thái trong ứng dụng Note.

Cuối cùng, chúng ta sẽ áp dụng BLoC Pattern để quản lý logic của ứng dụng. BLoC Pattern giúp chúng ta tách biệt logic và giao diện người dùng, từ đó tạo ra một ứng dụng dễ dàng kiểm thử và mở rộng.

Bằng cách tổng hợp những phương pháp quản lý trạng thái và mô hình thiết kế trong Flutter, dự án này sẽ giúp bạn hiểu rõ hơn về cách xây dựng ứng dụng sử dụng Provider, State Management và BLoC Pattern trong Flutter, và tạo ra các ứng dụng chất lượng cao và dễ bảo trì.

* 1. **Phạm vi đề tài**

Xây dựng một ứng dụng di động ToDo đơn giản bằng cách sử dụng các công nghệ và phương pháp quản lý trạng thái như Provider, State Management và BLoC Pattern trong Flutter.

Trong phạm vi của đề tài này, chúng em sẽ tạo ra một giao diện ứng dụng cơ bản và cung cấp các chức năng như thêm, sửa, xóa và xem danh sách các công việc. Chúng em sẽ sử dụng Provider để quản lý trạng thái của ứng dụng, sử dụng các phương pháp State Management để theo dõi và cập nhật trạng thái trong ứng dụng và sử dụng BLoC Pattern để quản lý logic của ứng dụng.

Đề tài này tập trung vào việc áp dụng các phương pháp và mô hình quản lý trạng thái trong Flutter và không đi sâu vào các chức năng phức tạp hoặc tích hợp các chức năng nâng cao như đồng bộ dữ liệu với máy chủ hoặc tích hợp các dịch vụ mạng.

Phạm vi của đề tài được giới hạn để tạo ra một ứng dụng di động Note đơn giản, nhưng cung cấp sự hiểu biết và sử dụng các kỹ thuật Provider, State Management và BLoC Pattern trong Flutter một cách hiệu quả.

# **CHƯƠNG 2: CƠ SỞ LÝ LUẬN**

* 1. **Lý thuyết**

Mục đích của việc tạo sản phẩm này là đánh giá khả năng tự áp dụng công nghệ, tư duy lập

trình và xử lý những vấn đề thực tế khi làm sản phẩm thật.

Để làm được, công việc đầu tiên cần làm là phải nghiên cứu kỹ cách vận hành của từng hệ

thống bằng cách trải nghiệm các sản phẩm, ứng dụng có sẵn. Tiếp đến là lên ý tưởng và mô

phỏng các chức năng làm việc. Sau đó là đọc và nghiên cứu tài liêu, cài đặt các công cụ hỗ

trợ. Cuối cùng là thực hiện phát triển hoàn thiện các chức năng.

* 1. **Kỹ thuật**

Về một số các kỹ thuật và công nghệ được áp dụng:

* Ngôn ngữ lập trình: Sử dụng ngôn ngữ Dart.
* Nền tảng Platform: Android Studio
  + 1. **Giới thiệu về Flutter**

Flutter là một khung nguồn mở do Google phát triển và hỗ trợ. Các nhà phát triển frontend và fullstack sử dụng Flutter để xây dựng giao diện người dùng (UI) của ứng dụng cho nhiều nền tảng chỉ với một nền mã duy nhất.

Tại thời điểm ra mắt vào năm 2018, Flutter chủ yếu hỗ trợ phát triển ứng dụng di động. Hiện nay, Flutter hỗ trợ phát triển ứng dụng trên sáu nền tảng: iOS, Android, web, Windows, MacOS và Linux.

1. **Các tính năng nổi bật của Flutter**

* **Phát triển ứng dụng gốc so với phát triển đa nền tảng**

Viết mã một ứng dụng cho một nền tảng cụ thể, chẳng hạn như iOS, được gọi là phát triển ứng dụng gốc. Ngược lại, phát triển ứng dụng đa nền tảng sẽ xây dựng một ứng dụng cho nhiều nền tảng với một nền mã duy nhất.

* **Phát triển ứng dụng gốc**

Vì các nhà phát triển viết mã cho một nền tảng cụ thể trong phát triển ứng dụng gốc, họ có toàn quyền truy cập vào chức năng của thiết bị gốc. Điều này thường mang lại hiệu suất và tốc độ cao hơn so với phát triển ứng dụng đa nền tảng.

Tuy nhiên, nếu bạn muốn khởi chạy một ứng dụng trên nhiều nền tảng, phát triển ứng dụng gốc cần nhiều mã hơn và nhiều nhà phát triển hơn. Ngoài những chi phí này, phát triển ứng dụng gốc khiến việc khởi chạy trên các nền tảng khác nhau cùng một lúc với trải nghiệm người dùng nhất quán trở nên khó khăn hơn. Đây là nơi mà các khung phát triển ứng dụng đa nền tảng như Flutter có thể hữu ích.

* **Phát triển ứng dụng đa nền tảng**

Phát triển ứng dụng đa nền tảng cho phép các nhà phát triển sử dụng một ngôn ngữ lập trình và một nền mã để xây dựng một ứng dụng cho nhiều nền tảng. Nếu bạn chuẩn bị phát hành một ứng dụng cho nhiều nền tảng, phát triển ứng dụng đa nền tảng sẽ ít tốn kém và đỡ mất thời gian hơn so với phát triển ứng dụng gốc

Quá trình này cũng cho phép các nhà phát triển tạo ra trải nghiệm nhất quán hơn cho người dùng trên các nền tảng.

Cách tiếp cận này có thể có những hạn chế so với phát triển ứng dụng gốc, đó là quyền truy cập hạn chế vào chức năng của thiết bị gốc. Tuy nhiên, Flutter có các tính năng giúp phát triển ứng dụng đa nền tảng mượt mà hơn và đạt hiệu suất cao.

1. **Ưu điểm của Flutter**

* **Hiệu suất gần với phát triển ứng dụng gốc**. Flutter sử dụng ngôn ngữ lập trình Dart và biên dịch thành mã máy. Các thiết bị máy chủ hiểu được mã này, điều này đảm bảo hiệu suất nhanh và hiệu quả.
* **Kết xuất nhanh, nhất quán và có thể tùy chỉnh**. Thay vì dựa vào các công cụ kết xuất theo nền tảng, Flutter sử dụng thư viện đồ họa Skia nguồn mở của Google để kết xuất UI. Điều này mang đến cho người dùng phương tiện trực quan nhất quán cho dù họ sử dụng nền tảng nào để truy cập ứng dụng.
* **Công cụ thân thiện với nhà phát triển**. Google đã xây dựng Flutter chú trọng vào tính dễ sử dụng. Với các công cụ như tải lại nóng, nhà phát triển có thể xem trước các thay đổi mã sẽ như thế nào mà không bị mất trạng thái. Các công cụ khác như widget inspector giúp dễ dàng trực quan hóa và giải quyết các vấn đề với bố cục UI.
  + 1. **Giới thiệu về Android Studio**

Android Studio là Môi trường phát triển tích hợp (IDE) chính thức để phát triển ứng dụng Android. Nhờ có công cụ cho nhà phát triển và trình soạn thảo mã mạnh mẽ của IntelliJ IDEA, Android Studio cung cấp thêm nhiều tính năng giúp bạn nâng cao năng suất khi xây dựng ứng dụng Android, chẳng hạn như:

* Một hệ thống xây dựng linh hoạt dựa trên Gradle
* Một trình mô phỏng nhanh và nhiều tính năng
* Một môi trường hợp nhất nơi bạn có thể phát triển cho mọi thiết bị Android
* Tính năng Live Edit (Chỉnh sửa trực tiếp) để cập nhật các thành phần kết hợp trong trình mô phỏng và thiết bị thực theo thời gian thực
* Mã mẫu và quá trình tích hợp GitHub để giúp bạn xây dựng các tính năng ứng dụng phổ biến cũng như nhập mã mẫu
* Đa dạng khung và công cụ thử nghiệm
* Công cụ tìm lỗi mã nguồn (lint) để nắm bắt hiệu suất, khả năng hữu dụng, khả năng tương thích với phiên bản và các vấn đề khác
* Hỗ trợ C++ và NDK
* Tích hợp sẵn tính năng hỗ trợ Google Cloud Platform, giúp dễ dàng tích hợp Google Cloud Messaging và App Engine

1. **Công cụ gỡ lỗi và phân tích tài nguyên**

Android Studio giúp bạn gỡ lỗi và cải thiện hiệu suất cho mã, bao gồm cả các công cụ gỡ lỗi cùng dòng và phân tích hiệu suất.

* **Gỡ lỗi cùng dòng**

Bạn có thể dùng tính năng gỡ lỗi cùng dòng để cải thiện kết quả kiểm tra toàn diện về mã (code walkthrough) trong khung hiển thị trình gỡ lỗi nhờ tính năng xác minh cùng dòng đối với nội dung tham chiếu, biểu thức và giá trị biến.

Thông tin gỡ lỗi cùng dòng bao gồm:

* Giá trị biến cùng dòng
* Các đối tượng tham chiếu đến một đối tượng đã chọn
* Giá trị trả về của phương thức
* Biểu thức toán tử và lambda
* Giá trị trong chú giải công cụ

Để bật tính năng gỡ lỗi cùng dòng, trong cửa sổ Debug (Gỡ lỗi), hãy nhấp vào biểu tượng Settings (Cài đặt) rồi chọn Show Variable Values in Editor (Hiện giá trị của các biến trong Trình chỉnh sửa).

* **Trình phân tích hiệu suất**

Android Studio cung cấp các trình phân tích hiệu suất để bạn có thể dễ dàng theo dõi mức sử dụng bộ nhớ và CPU của ứng dụng, tìm các đối tượng được giải phóng, xác định vị trí rò rỉ bộ nhớ, tối ưu hoá hiệu suất đồ hoạ và phân tích các yêu cầu về mạng.

Để sử dụng trình phân tích hiệu suất khi ứng dụng của bạn chạy trên thiết bị hoặc trình mô phỏng, hãy mở Android Profiler (Trình phân tích tài nguyên Android) bằng cách chọn View > Tool Windows > Profiler (Xem > Cửa sổ công cụ > Trình phân tích tài nguyên).

* **Tệp báo lỗi**

Khi phân tích mức sử dụng bộ nhớ trong Android Studio, bạn có thể đồng thời bắt đầu thu thập rác và báo lỗi Java (heap dump) vào một ảnh chụp nhanh của vùng nhớ khối xếp trong một tệp định dạng nhị phân HPROF dành riêng cho Android. Trình xem HPROF hiển thị các lớp, phiên bản của từng lớp và cây tham chiếu để giúp bạn theo dõi mức sử dụng bộ nhớ và tìm lỗi rò rỉ bộ nhớ.

* **Trình phân tích bộ nhớ**

Sử dụng Trình phân tích bộ nhớ để theo dõi quá trình phân bổ bộ nhớ và xem vị trí phân bổ các đối tượng khi thực hiện một số thao tác. Các cơ cấu phân bổ này giúp bạn tối ưu hoá hiệu suất và mức sử dụng bộ nhớ của ứng dụng bằng cách điều chỉnh các lệnh gọi phương thức có liên quan đến các thao tác đó.

* **Truy cập vào tệp dữ liệu**

Bộ công cụ SDK Android, chẳng hạn như Systrace và Logcat, tạo ra dữ liệu về hiệu suất và quá trình gỡ lỗi cho bản phân tích ứng dụng chi tiết.

Cách xem các tệp dữ liệu đã tạo hiện có:

1. Mở cửa sổ công cụ Captures (Ghi lại).
2. Trong danh sách các tệp đã tạo, hãy nhấp đúp vào một tệp để xem dữ liệu tương ứng.
3. Nhấp chuột phải vào tệp HPROF bất kỳ để chuyển đổi các tệp đó thành tệp chuẩn.
4. Kiểm tra định dạng tệp sử dụng RAM.

* **Kiểm tra mã**

Bất cứ khi nào bạn biên dịch chương trình, Android Studio sẽ tự động chạy các lượt kiểm tra lint (tìm lỗi mã nguồn) đã định cấu hình và các hoạt động kiểm tra IDE khác để giúp bạn dễ dàng xác định cũng như khắc phục vấn đề liên quan đến chất lượng cấu trúc của mã.

Công cụ tìm lỗi mã nguồn (lint tool) kiểm tra các tệp nguồn dự án Android để tìm ra các lỗi có thể xảy ra và cải thiện khả năng tối ưu hoá, nhằm đảm bảo tính chính xác, tính bảo mật, hiệu suất, khả năng hữu dụng, khả năng hỗ trợ tiếp cận và khả năng quốc tế hoá.

Ngoài các bước kiểm tra tìm lỗi mã nguồn (lint), Android Studio còn thực hiện quy trình kiểm tra mã IntelliJ và xác thực chú giải để đơn giản hoá quy trình lập trình của bạn.

* **Chú thích trong Android Studio**

Android Studio hỗ trợ tính năng chú giải cho các biến, tham số và giá trị trả về để giúp bạn phát hiện lỗi, chẳng hạn như ngoại lệ về con trỏ rỗng và xung đột loại tài nguyên.

Android SDK Manager (Trình quản lý SDK Android) đóng gói thư viện chú giải Jetpack trong Android Support Repository (Kho lưu trữ hỗ trợ của Android) để dùng trên Android Studio. Android Studio xác thực các chú giải đã định cấu hình trong quá trình kiểm tra mã.

* **Thông điệp nhật kí**

Khi xây dựng và chạy ứng dụng bằng Android Studio, bạn có thể xem thông điệp nhật ký về thiết bị và đầu ra adb trong cửa sổ Logcat.

1. **Hệ thống xây dựng Gradle**

Android Studio sử dụng Gradle làm nền tảng cho hệ thống xây dựng với nhiều tính năng dành riêng cho Android do Trình bổ trợ Android cho Gradle cung cấp. Hệ thống xây dựng này hoạt động như một công cụ tích hợp trên trình đơn Android Studio và độc lập với dòng lệnh. Bạn có thể sử dụng các tính năng của hệ thống xây dựng để làm những việc sau

* Tuỳ chỉnh, định cấu hình và mở rộng quy trình xây dựng.
* Tạo nhiều tệp APK cho ứng dụng với nhiều tính năng trong khi sử dụng cùng một dự án và mô-đun.
* Sử dụng lại mã và tài nguyên trên các nhóm tài nguyên (source set).
* Nhờ vận dụng tính linh hoạt của Gradle, bạn có thể làm được những việc này mà không cần sửa đổi các tệp nguồn cốt lõi của ứng dụng.

Tệp bản dựng Android Studio có tên build.gradle.kts nếu bạn sử dụng Kotlin (nên dùng), hoặc có tên là build.gradle nếu bạn sử dụng Groovy. Đây là các tệp văn bản thuần tuý sử dụng cú pháp Kotlin hoặc Groovy để định cấu hình bản dựng bằng các phần tử do trình bổ trợ Android cho Gradle cung cấp. Mỗi dự án có một tệp bản dựng cấp cao nhất cho toàn bộ dự án và các tệp bản dựng cấp mô-đun riêng cho từng mô-đun. Khi bạn nhập một dự án hiện có, Android Studio sẽ tự động tạo ra các tệp bản dựng cần thiết.

* **Biến thể bản dựng**

Hệ thống xây dựng có thể giúp bạn tạo nhiều phiên bản của cùng một ứng dụng trong một dự án duy nhất. Việc này khá hữu ích nếu ứng dụng của bạn có cả phiên bản miễn phí và phiên bản có tính phí, hoặc nếu bạn muốn phân phối nhiều tệp APK qua Google Play tuỳ theo cấu hình thiết bị.

* **Hỗ trợ nhiều APK**

Tính năng hỗ trợ nhiều APK cho phép bạn tạo nhiều APK một cách hiệu quả dựa trên mật độ màn hình hoặc ABI (Giao diện nhị phân ứng dụng). Ví dụ: bạn có thể tạo các APK riêng biệt của một ứng dụng cho mật độ màn hình hdpi và mdpi, trong khi vẫn xem xét các APK này là một biến thể duy nhất cũng như cho phép chúng dùng chung chế độ cài đặt của APK kiểm thử, javac, dx và ProGuard.

* **Rút gọn tài nguyên**

Tính năng rút gọn tài nguyên (resource shrinking) trong Android Studio sẽ tự động xoá các tài nguyên không dùng đến khỏi các phần phụ thuộc trong thư viện và ứng dụng đóng gói. Ví dụ: nếu ứng dụng của bạn dùng Dịch vụ Google Play để tiếp cận chức năng của Google Drive và bạn hiện không dùng tính năng Đăng nhập bằng Google, thì tính năng rút gọn tài nguyên có thể xoá các tài sản có thể vẽ cho nút SignInButton.

* **Quản lý phần phụ thuộc**

Bạn sẽ chỉ định các phần phụ thuộc cho dự án theo tên trong tập lệnh bản dựng cấp mô-đun. Gradle sẽ tìm các phần phụ thuộc đó và đưa vào trong bản dựng của bạn. Bạn có thể khai báo các phần phụ thuộc của mô-đun, phần phụ thuộc của tệp nhị phân từ xa và phần phụ thuộc của tệp nhị phân cục bộ trong tệp build.gradle.kts.

Android Studio định cấu hình các dự án để sử dụng Maven Central Repository theo mặc định. Cấu hình này có trong tệp bản dựng cấp cao nhất cho dự án.

* + 1. **Giới thiệu về Provider**
* **Provider**

Provider là một widget cơ bản nhất trong tất cả các Provider mà thư viện cung cấp. Bạn có thể sử dụng nó để cung cấp một giá trị (thường là một data model) cho bất kỳ vị trí nào trong widget tree. Tuy nhiên, nó sẽ không giúp bạn cập nhật widget tree khi mà giá trị đó thay đổi. Bạn có thể hình dung như việc nó chỉ set dữ liệu vào mà UI nó không có sự thay đổi gì, không nhận biết được sự thay đổi.

* **ChangeNotifierProvider**

Không giống như Provider, ChangeNotifierProvider lắng nghe các thay đổi trong data model. Khi có thay đổi, nó sẽ xây dựng lại bất kỳ widget nào trong Consumer.

Trong hàm build thay đổi Provider thành ChangeNotifierProvider. Lớp mô hình cần sử dụng extend ChangeNotifier (hoặc with ChangeNotifier). Điều này cung cấp cho bạn quyền truy cập vào notifyListeners() và bất kỳ lúc nào bạn gọi notifyListeners() thì ChangeNotifierProvider sẽ được thông báo và tất cả các widget bên trong Consumers sẽ được rebuild lại.

* **ValueListenableProvider**

Nó giống như ChangeNotifierProvider. . . nhưng phức tạp hơn. . . và không có bất kỳ giá trị gia tăng rõ ràng nào.

* **StreamProvider**

StreamProvider về cơ bản là một wrapper với bên trong là một StreamBuilder. Bạn cung cấp một Stream và sau đó Consumer được xây dựng lại khi có sự kiện trong steam. Thiết lập rất giống với FutureProvider ở trên.

* **FutureProvider**

FutureProvider về cơ bản chỉ là một wrapper với bên trong là FutureBuilder. Bạn cung cấp cho nó một số dữ liệu ban đầu để hiển thị trong giao diện người dùng và cũng có thể cung cấp cho nó một hoạt động bất đồng bộ Future của giá trị mà bạn muốn cung cấp. FutureProvider lắng nghe khi Future hoàn thành và sau đó thông báo cho Consumers để xây dựng lại các widget của nó.

* **MultiProvider**

Cho đến nay các ví dụ của tôi chỉ sử dụng một model object. Nếu bạn cần cung cấp loại model object thứ hai, bạn có thể lồng các provider. Tuy nhiên, có một cách khác gọn gàng hơn là sử dụng MultiProvider.

* **ProxyProvider**

Điều gì sẽ xảy ra nếu bạn có hai Model mà bạn muốn cung cấp, nhưng một trong các Model phụ thuộc vào mô hình còn lại? Trong trường hợp đó, bạn có thể sử dụng ProxyProvider. ProxyProvider lấy giá trị từ một provider và cho phép nó được đưa vào provider khác.

* + 1. **Giới thiệu về State Management**

Trạng thái là thông tin có thể đọc được khi widget được tạo và có thể thay đổi hoặc sửa đổi trong suốt thời gian tồn tại của ứng dụng. Nếu bạn muốn thay đổi widget của mình, bạn cần cập nhật đối tượng trạng thái, có thể được thực hiện bằng cách sử dụng hàm setState() có sẵn cho các widget Stateful. Hàm setState() cho phép chúng ta thiết lập các thuộc tính của đối tượng trạng thái kích hoạt vẽ lại giao diện người dùng.

Quản lý state(trạng thái) là một trong những quy trình phổ biến và cần thiết nhất trong vòng đời của một ứng dụng. Theo tài liệu chính thức, Flutter mang tính chất khai báo. Điều đó có nghĩa là Flutter xây dựng giao diện người dùng của mình bằng cách phản ánh trạng thái hiện tại của ứng dụng của bạn. Hình sau giải thích rõ hơn về nơi bạn có thể xây dựng giao diện người dùng từ trạng thái ứng dụng.

Trong Flutter, quản lý state(trạng thái) phân thành 2 loại khái niệm:

* **Trạng thái tức thời(Ephemeral State)**

Trạng thái này còn được gọi là Trạng thái giao diện người dùng hoặc trạng thái địa phương. Nó là một loại trạng thái có liên quan đến widget cụ thể, hoặc bạn có thể nói rằng nó là một trạng thái chứa trong một widget duy nhất. Trong loại trạng thái này, bạn không cần sử dụng các kỹ thuật quản lý state(trạng thái). Ví dụ phổ biến của trạng thái này là Text Field.

* **Trạng thái ứng dụng(App State)**

Nó khác với trạng thái tức thời. Đó là một loại trạng thái mà chúng ta muốn chia sẻ trên các phần khác nhau của ứng dụng và muốn giữ lại giữa các phiên của người dùng. Do đó, loại trạng thái này có thể được sử dụng trên toàn cầu. Đôi khi nó còn được gọi là trạng thái ứng dụng hoặc trạng thái chia sẻ. Một số ví dụ về trạng thái này là Tùy chọn người dùng, Thông tin đăng nhập, thông báo trong ứng dụng mạng xã hội, giỏ hàng trong ứng dụng thương mại điện tử, trạng thái đã đọc / chưa đọc của các bài báo trong ứng dụng tin tức, v.v.

A diagram of a data flow

Description automatically generated

*Hình 1 Sơ đồ sự khác biệt giữa Ephemeral State và App State*

* + 1. **Giới thiệu về BLoC Pattern**

BLoC Pattern (Bussiness Logic Component) đã được Paolo Soares công bố chính thức trong hội nghị Dart 2018. Nếu bạn đã xem video, có lẽ bạn đã nhận ra rằng đề xuất ban đầu là sử dụng lại code liên quan đến business logic trong các nền tảng khác, trong trường hợp này là Angular Dart. Ngay lập tức, những gì mà pattern này làm là lấy tất cả business logic code ra khỏi giao diện người dùng và chỉ sử dụng nó trong các lớp BLoC class. Nó giúp cho code có tính độc lập của môi trường và nền tảng, bên cạnh việc đặt các responsibility vào đúng component. Và bây giờ mọi thứ sẽ rõ ràng hơn nhiều, bởi vì BLoC Pattern chỉ dựa vào việc sử dụng Stream.

A diagram of a computer

Description automatically generated

*Hình 2 BLoC Pattern*

Nhìn vào hình trên, chúng ta có thể nhận ra kiến trúc flux. Các Widget gửi dữ liệu/sự kiện đến BLoC class thông qua Sink và được Stream thông báo. Nhận thấy rằng không có business logic nào trong widget, điều đó có nghĩa là những gì đã xảy ra trong BLoC không phải là mối quan tâm của giao diện người dùng. Kiến trúc này cải thiện để thực hiện Unit Test dễ dàng hơn, trong đó các trường hợp kiểm tra business logic chỉ cần áp dụng cho các BLoC class.

1. **Các khái niệm cốt lõi trong thư viện BloC**

* **Events**

Là đầu vào của bloc ví dụ : button presses

* **State**

Là đầu ra của Bloc, biểu diễn trạng thái của ứng dụng. Từ các state mà có thể đưa ra các bloc cần thiết.

* **Stream**

Là một chuỗi data bất đồng bộ ( async data ) , có thể ví stream như những ống nước, ống là stream còn nước là data bất đồng bộ ( asynchronous data ).

* **Blocs**

bloc được coi như bộ não với nhiệm vụ chuyển đổi ( convert ) các đầu vào là các stream Event thành các stream State ở đầu ra.

Sử dụng thư viện bloc , sau đó tạo class bloc kế thừa lớp Bloc từ thư viện bloc đã cài.

* + 1. **Giới thiệu về ngôn ngữ Dart**

1. **Dart là gì**

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.

1. **Tại sao lại sử dụng Dart**

Dart là một ngôn ngữ độc lập với nền tảng và hỗ trợ tất cả các hệ điều hành như Windows, Mac, Linux, v.v.

Nó là một ngôn ngữ mã nguồn mở, có nghĩa là nó có sẵn miễn phí cho tất cả mọi người. Nó đi kèm với giấy phép BSD và được công nhận bởi tiêu chuẩn ECMA.

Nó là một ngôn ngữ lập trình hướng đối tượng và hỗ trợ tất cả các tính năng của oops như kế thừa, giao diện và các tính năng kiểu tùy chọn.

Dart rất hữu ích trong việc xây dựng các ứng dụng thời gian thực vì tính ổn định của nó.

Dart đi kèm với trình biên dịch dar2js để truyền mã Dart thành mã JavaScript chạy trên tất cả các trình duyệt web hiện đại.

Máy ảo Dart độc lập cho phép mã Dart chạy trong môi trường giao diện dòng lệnh.

1. **Tính năng của Dart**

Dart là một ngôn ngữ lập trình hướng đối tượng, mã nguồn mở, chứa nhiều tính năng hữu ích. Đây là ngôn ngữ lập trình mới và hỗ trợ một loạt các tiện ích lập trình như giao diện, bộ sưu tập, lớp, kiểu gõ động và tùy chọn. Nó được phát triển cho máy chủ cũng như trình duyệt. Dưới đây là danh sách các tính năng quan trọng của Dart.

A diagram of dart features

Description automatically generated

*Hình 3 Dart Features*

* **Mã nguồn mở**

Dart là một ngôn ngữ lập trình mã nguồn mở, có nghĩa là nó có sẵn miễn phí. Nó được phát triển bởi Google, được phê duyệt bởi tiêu chuẩn ECMA và đi kèm với giấy phép BSD.

* **Nền tảng độc lập**

Dart hỗ trợ tất cả các hệ điều hành chính như Windows, Linux, Macintosh, v.v. Dart có Máy ảo riêng được gọi là Dart VM, cho phép chúng tôi chạy mã Dart trong mọi hệ điều hành.

* **Hướng đối tượng**

Dart là một ngôn ngữ lập trình hướng đối tượng và hỗ trợ tất cả các khái niệm oops như lớp, kế thừa, giao diện và các tính năng gõ tùy chọn. Nó cũng hỗ trợ các khái niệm nâng cao như mixin, abstract, các lớp, hệ thống kiểu chung được sửa đổi và mạnh mẽ.

* **Đồng nhất**

Dart là một ngôn ngữ lập trình không đồng bộ, có nghĩa là nó hỗ trợ đa luồng sử dụng Isolates. Các vùng cách ly là các thực thể độc lập có liên quan đến các luồng nhưng không chia sẻ bộ nhớ và thiết lập giao tiếp giữa các quá trình bằng cách truyền thông điệp. Thông điệp nên được nối tiếp nhau để tạo hiệu quả truyền thông. Việc tuần tự hóa thông báo được thực hiện bằng cách sử dụng một ảnh chụp nhanh được tạo ra bởi đối tượng đã cho và sau đó truyền đến một vùng cách ly khác để giải mã.

* **Thư viện mở rộng**

Dart bao gồm nhiều thư viện tích hợp hữu ích bao gồm SDK (Bộ phát triển phần mềm), lõi, toán học, không đồng bộ, toán học, chuyển đổi, html, IO, v.v. Nó cũng cung cấp cơ sở để tổ chức mã Dart thành các thư viện với không gian tên riêng. Nó có thể sử dụng lại bằng câu lệnh nhập.

* **Dễ học**

Như chúng ta đã thảo luận trong phần trước, học Dart không phải là nhiệm vụ của Hercules vì ​​chúng ta biết rằng cú pháp của Dart tương tự như Java, C #, JavaScript, kotlin, v.v. nếu bạn biết bất kỳ ngôn ngữ nào trong số này thì bạn có thể học Dart dễ dàng.

* **Biên dịch linh hoạt**

Dart cung cấp sự linh hoạt để biên dịch mã và nhanh chóng. Nó hỗ trợ hai loại quy trình biên dịch, AOT (Ahead of Time) và JIT (Just-in-Time). Mã Dart được truyền bằng ngôn ngữ khác có thể chạy trong các nhà sản xuất web hiện đại.

* **Nhập An toàn**

Dart là ngôn ngữ an toàn kiểu, có nghĩa là nó sử dụng cả kiểm tra kiểu tĩnh và kiểm tra thời gian chạy để xác nhận rằng giá trị của một biến luôn khớp với kiểu tĩnh của biến, đôi khi nó được gọi là kiểu gõ âm thanh.

Mặc dù loại là bắt buộc, nhưng chú thích loại là tùy chọn vì loại nhiễu. Điều này làm cho mã dễ đọc hơn. Ưu điểm khác của ngôn ngữ an toàn kiểu chữ là khi chúng ta thay đổi phần mã, hệ thống sẽ cảnh báo chúng ta về sửa đổi mà chúng ta đã sửa trước đó.

* **Các đối tượng**

Dart coi mọi thứ như một đồ vật. Giá trị gán cho biến là một đối tượng. Các hàm, số và chuỗi cũng là một đối tượng trong Dart. Tất cả các đối tượng kế thừa từ lớp Đối tượng.

* **Hỗ trợ trình duyệt**

Dart hỗ trợ tất cả các trình duyệt web hiện đại. Nó đi kèm với trình biên dịch dart2js để chuyển đổi mã Dart thành mã JavaScript được tối ưu hóa phù hợp với tất cả các loại trình duyệt web.

* **Cộng đồng**

Dart có một cộng đồng lớn trên toàn thế giới. Vì vậy, nếu bạn gặp vấn đề trong khi viết mã thì rất dễ dàng tìm được trợ giúp. Nhóm các nhà phát triển chuyên dụng đang làm việc để nâng cao chức năng của nó.

# **CHƯƠNG 3: CÀI ĐẶT THỬ NGHIỆM**

## Cài đặt triển khai ứng dụng

Thông qua việc áp dụng những kiến thức về State Managerment, Provider và Bloc Pattern để xây dựng ứng dụng Note đơn giản bằng Flutter và ngôn ngữ Dart, kết nối và lấy dữ liệu từ server bằng cách gọi API, JSON, bạn có thể làm theo các bước sau:

Bước 1: Chuẩn bị môi trường phát triển

- Cài đặt Flutter SDK và cấu hình môi trường để phát triển ứng dụng Flutter trên máy tính của bạn.

Bước 2: Tạo một ứng dụng Flutter mới

- Mở terminal hoặc command prompt và chạy lệnh sau để tạo một ứng dụng Flutter mới:

Flutter create chat\_app

Bước 3: Thêm các dependencies cần thiết

- Mở file `pubspec.yaml` trong thư mục gốc của dự án Flutter và thêm dependencies sau:

**A screen shot of a computer program

Description automatically generated**

*Hình 4 Các Dependencies*

- Chạy lệnh sau để cập nhật dependencies:

*Flutter pub get*

Bước 4: Thiết kế giao diện người dùng

Tạo các widget cần thiết để hiển thị danh sách ghi chú và các chức năng liên quan như thêm, sửa và xóa ghi chú.

Xây dựng màn hình thêm ghi chú, ghi chú và các widget khác.

Bước 5: Xây dựng các lớp Entities (class Todo) để tương tác với ứng dụng và dữ liệu

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

*Hình 5 Xây dựng các lớp Entities*

Bước 6: Xây dựng các Bloc Pattern

Tạo các sự kiện (events) liên quan đến ghi chú như LoadTodos, AddTodos, UpdateTodos, DeleteTodos.

A screen shot of a computer program

Description automatically generated

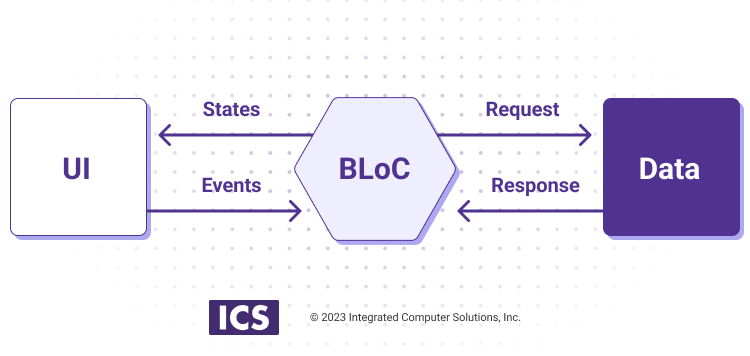
A screen shot of a computer program

Description automatically generated

*Hình 6 Xây dựng BLoC Pattern*

Xây dựng lớp NotesBloc để xử lý các sự kiện và cung cấp trạng thái.

Khi người dùng tương tác ở UI thì các Bloc sẽ gọi các events để thực thi chức năng và Request lên API để truy cập dữ liệu và nhận về Reponse tương ứng. Sau đó các Bloc sẽ thay đổi lại trạng thái (State) về cho UI để hiển thị cho người dùng.



*Hình 7 Luồng xử lý*

Bước 8: Các Service

A computer screen shot of a program code

Description automatically generated

*Hình 8 Các Services*

Bước 9: Triển khai API

A screen shot of a computer code

Description automatically generated

**A screen shot of a computer code

Description automatically generated**

**A screen shot of a computer code

Description automatically generated**

**A computer screen shot of code

Description automatically generated**

*Hình 9 Triển khai API*

Bước 9: Hiển thị dữ liệu

A screen shot of a computer program

Description automatically generated

*Hình 10 Hiển thị dữ liệu*

MultiBlocProvider: Widget này cho phép bạn cung cấp nhiều BlocProvider để quản lý trạng thái của ứng dụng. Trong đoạn mã của bạn, bạn cung cấp hai BlocProvider.

BlocProvider cho TodosBloc: Đây là nơi bạn tạo và cung cấp TodosBloc cho ứng dụng của bạn. Trong phần create, bạn tạo một thể hiện mới của TodosBloc và thêm sự kiện LoadTodos () ngay khi Bloc được tạo ra.

BlocProvider cho TodosFilterBloc: Đây là nơi bạn cung cấp TodosFilterBloc, một Bloc khác liên quan đến việc lọc các ghi chú (todos) trong danh sách. Bạn cần cung cấp một thể hiện của TodosBloc cho TodosFilterBloc để nó có thể tương tác với dữ liệu từ TodosBloc.

Bước 10: Điều hướng

Xây dựng Router di chuyển giữa các màn hình

A screen shot of a computer program

Description automatically generated

*Hình 11 Xây dựng Router di chuyển*

Bước 11: Thêm các chức năng tùy chọn

Thêm các chức năng như là hoàn thành công việc, chuyển đổi trạng thái của các Todo, Navigation giữa các Tab Đã hoàn thành và chưa hoàn thành , v.v

* 1. **Kết quả**

A white circle with blue text

Description automatically generatedMàn hình chính

*Hình 12 Màn hình chính*

A screenshot of a phone

Description automatically generated

*Hình 13 Màn hình các mục chưa hoàn thành*

A screenshot of a phone

Description automatically generated

*Hình 14 Màn hình các mục đã hoàn thành*

*A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generated*

*Hình 15 Màn hình thêm công việc mới*

*A screenshot of a phone

Description automatically generated*

*Hình 16 Màn hình hoàn thành công việc vừa tạo*

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

## 4.1 Kết luận

Trong đề tài "Xây dựng ứng dụng ToDo cơ bản", chúng em đã nghiên cứu và thực hiện xây dựng một ứng dụng ghi chú đơn giản bằng Flutter, sử dụng các công nghệ provider, state management và BLoC pattern.

Trước tiên, chúng em đã tìm hiểu về provider trong Flutter, một thư viện state management mạnh mẽ cho phép chúng ta chia sẻ và quản lý trạng thái của ứng dụng một cách dễ dàng. Chúng tôi đã sử dụng provider để quản lý trạng thái của danh sách ghi chú trong ứng dụng của chúng tôi. Sử dụng provider giúp chúng em truy cập và cập nhật dữ liệu từ khắp nơi trong ứng dụng một cách thuận tiện và hiệu quả.

Bên cạnh đó, chúng em đã áp dụng BLoC pattern trong ứng dụng để quản lý logic và trạng thái của các sự kiện. BLoC (Business Logic Component) là một kiến trúc state management giúp chúng ta tách biệt hoàn toàn giữa logic và giao diện người dùng. Chúng em đã sử dụng BLoC để quản lý việc tạo, xóa và cập nhật các ghi chú trong ứng dụng của chúng tôi. Với BLoC pattern, chúng tôi có thể dễ dàng kiểm thử và tái sử dụng các thành phần logic của ứng dụng một cách linh hoạt.

Thông qua việc sử dụng các công nghệ provider, state management và BLoC pattern, chúng em đã xây dựng thành công một ứng dụng ghi chú đơn giản và mạnh mẽ trong Flutter. Qua quá trình nghiên cứu và thực hiện đề tài này, chúng em nhận thấy rằng sử dụng các công nghệ này giúp ứng dụng linh hoạt, dễ dàng bảo trì và phát triển trong tương lai.

Tuy nhiên, cần lưu ý rằng việc lựa chọn công nghệ state management và pattern phù hợp với đề tài cụ thể là một quyết định quan trọng và cần được xem xét kỹ lưỡng. Việc sử dụng provider, state management và BLoC pattern chỉ là một trong số nhiều phương pháp có thể áp dụng trong Flutter, và việc tìm hiểu và thử nghiệm các công nghệ khác là điều cần thiết để chọn ra phương pháp tốt nhất cho việc xây dựng ứng dụng.

## 4.2 Hạn chế

- Độ phức tạp của việc thực hiện: Sử dụng provider, state management và BLoC pattern trong Flutter có thể đòi hỏi kiến thức kỹ thuật và thời gian nghiên cứu để hiểu và triển khai chúng một cách hiệu quả. Điều này có thể làm tăng độ phức tạp của việc phát triển và bảo trì ứng dụng.

- Khả năng mở rộng: Mặc dù provider, state management và BLoC pattern là các công nghệ mạnh mẽ, nhưng chúng có thể không phù hợp hoặc không đủ linh hoạt để mở rộng quy mô lớn hoặc phát triển các tính năng phức tạp trong ứng dụng.

- Tính tương thích: Cần kiểm tra tính tương thích và tương thích ngược của các công nghệ này với các phiên bản và thư viện Flutter khác nhau. Có thể xảy ra tình huống khi một số thư viện hoặc phiên bản Flutter không tương thích với cấu trúc code đã triển khai sử dụng provider, state management và BLoC pattern.

- Độ phổ biến: Provider, state management và BLoC pattern không phải là phương pháp duy nhất để quản lý trạng thái và logic trong Flutter. Vì vậy, sự phổ biến và hỗ trợ của cộng đồng có thể hạn chế, khiến việc tìm kiếm giải pháp và hỗ trợ có thể khó khăn hơn.

- Tốn tài nguyên: Việc sử dụng provider, state management và BLoC pattern có thể tăng tốn tài nguyên hệ thống và không gian lưu trữ. Điều này có thể ảnh hưởng đến hiệu suất và dung lượng của ứng dụng.

## 4.3 Hướng phát triển

- Tối ưu hóa hiệu suất: Nếu ứng dụng có dữ liệu lớn hoặc hoạt động chậm, có thể thử nghiệm các phương pháp tối ưu hóa như lazy loading dữ liệu, paging hoặc sử dụng công nghệ caching để cải thiện hiệu suất.

- Thêm tính năng đồng bộ: thêm chức năng đồng bộ dữ liệu ghi chú giữa các thiết bị khác nhau hoặc lưu trữ dữ liệu trên máy chủ để người dùng có thể truy cập từ nhiều thiết bị khác nhau.

- Tích hợp xác thực người dùng: Để tăng tính bảo mật và quản lý người dùng, có thể tích hợp các dịch vụ xác thực người dùng như Firebase Authentication hoặc Auth0.

- Tích hợp đánh giá và phản hồi của người dùng: Thêm một tính năng cho phép người dùng đánh giá và phản hồi ứng dụng giúp thu thập thông tin để cải thiện trải nghiệm người dùng và xây dựng một ứng dụng ngày càng tốt hơn.

- Thêm tính năng chia sẻ: Cho phép người dùng chia sẻ ghi chú qua các ứng dụng khác hoặc gửi ghi chú dưới dạng email hoặc tin nhắn.

- Cải thiện giao diện người dùng: Tìm hiểu và áp dụng các nguyên tắc thiết kế giao diện người dùng như Material Design hoặc Cupertino Design để cải thiện trải nghiệm người dùng và thêm tính thẩm mĩ cho ứng dụng.

- Kiểm thử và sửa lỗi: Thực hiện các bài kiểm tra đơn vị và kiểm tra tích hợp để đảm bảo tính ổn định và chất lượng của mã nguồn. Sửa lỗi và tối ưu hóa ứng dụng theo phản hồi từ người dùng.

# **DANH MỤC TÀI LIỆU THAM KHẢO**

1. Flutter Official Documentation - <https://docs.flutter.dev/>
2. Provider - <https://pub.dev/packages/provider>
3. FlutterBloc - <https://bloclibrary.dev/#/>
4. Bloc Pattern in Flutter - <https://docs.flutter.dev/data-and-backend/state-mgmt/simple>