**Tên: Nguyễn Thanh Ý**

**MSSV: 0850080056**

**LÍ THUYẾT BUỔI 9**

* **Storage**

Trong Flutter, "storage" thường được liên quan đến việc lưu trữ dữ liệu tạm thời hoặc lâu dài trên thiết bị của người dùng. Có hai loại chính của lưu trữ trong Flutter: lưu trữ tạm thời (Temporary Storage) và lưu trữ cục bộ (Local Storage).

1. **Lưu trữ tạm thời (Temporary Storage):**

Lưu trữ tạm thời thường được sử dụng để lưu trữ dữ liệu ngắn hạn mà bạn không cần giữ lại lâu dài. Trong Flutter, bạn có thể sử dụng các biến tạm thời như biến cục bộ để lưu trữ các giá trị tạm thời trong quá trình thực thi ứng dụng. Điều này thích hợp cho các dữ liệu như trạng thái tạm thời của giao diện người dùng, thông tin phiên làm việc, và những thứ tương tự.

1. **Lưu trữ cục bộ (Local Storage):**

Lưu trữ cục bộ là việc lưu trữ dữ liệu mà ứng dụng của bạn cần lưu giữ sau mỗi lần khởi động ứng dụng hoặc sau khi người dùng thoát khỏi ứng dụng. Để thực hiện lưu trữ cục bộ trong Flutter, bạn có thể sử dụng các thư viện như shared\_preferences hoặc sqflite.

* shared\_preferences: Thư viện này cho phép bạn lưu trữ dữ liệu dạng key-value (cặp khóa-giá trị) một cách đơn giản. Dữ liệu này được lưu trong cơ sở dữ liệu nhẹ gọi là "shared preferences" của ứng dụng, và nó không thích hợp cho việc lưu trữ lượng dữ liệu lớn hoặc phức tạp.
* sqflite: Đây là một thư viện SQLite dành cho Flutter, cho phép bạn tạo và quản lý cơ sở dữ liệu cục bộ trên thiết bị. Thư viện này thích hợp cho việc lưu trữ dữ liệu phức tạp và lớn hơn.

Ví dụ, bạn có thể sử dụng shared\_preferences để lưu trữ thông tin đăng nhập của người dùng (như tên người dùng), và sử dụng sqflite để lưu trữ danh sách các mục trong ứng dụng của bạn (như danh bạ).

Tóm lại, storage trong Flutter là quá trình lưu trữ và quản lý dữ liệu tạm thời và lâu dài trên thiết bị của người dùng, giúp ứng dụng của bạn có khả năng duy trì và khôi phục thông tin cần thiết sau mỗi lần chạy ứng dụng.

* **3rd Party, Librabies**

Trong ngữ cảnh của Flutter và phát triển ứng dụng di động nói chung, "3rd Party" thường đề cập đến các thư viện, gói mã nguồn mở hoặc phần mềm được phát triển bởi bên thứ ba, tức là không phải là bộ phận chính thức của Flutter Framework hoặc các công cụ liên quan của Google.

Trong Flutter, thuật ngữ "libraries" (thư viện) thường đề cập đến các tập hợp các mã nguồn đã được viết sẵn để cung cấp các chức năng, tính năng hoặc khả năng tái sử dụng cho các ứng dụng. Các thư viện này giúp nhà phát triển tiết kiệm thời gian và công sức bằng cách cung cấp mã nguồn đã được thử nghiệm và kiểm tra để giải quyết các vấn đề cụ thể.

Trong Flutter, có nhiều thư viện được cung cấp bởi cộng đồng và các nhà phát triển khác nhau. Các thư viện này có thể giúp bạn thực hiện các nhiệm vụ khác nhau, từ quản lý trạng thái ứng dụng đến tương tác với API, làm việc với hình ảnh và video, xử lý dữ liệu, và nhiều tính năng khác.Ví dụ về các thư viện 3rd Party trong Flutter bao gồm:

* http: Thư viện này cung cấp các phương thức để tương tác với các API HTTP, giúp bạn gửi và nhận dữ liệu từ máy chủ.
* provider: Đây là một thư viện quản lý trạng thái sử dụng kiến trúc "Provider". Nó giúp bạn quản lý và chia sẻ dữ liệu trạng thái giữa các thành phần khác nhau trong ứng dụng.
* firebase\_core và firebase\_auth: Đây là những thư viện cho phép tích hợp dễ dàng với các dịch vụ của Firebase, như xác thực người dùng, lưu trữ dữ liệu thời gian thực và nhiều tính năng khác.
* flutter\_bloc và rxdart: Các thư viện này giúp bạn quản lý trạng thái ứng dụng thông qua kiến trúc "BLoC" (Business Logic Component), cho phép tách biệt hoá logic kinh doanh và giao diện người dùng.
* flutter\_image\_compress: Thư viện này giúp bạn nén và xử lý hình ảnh để tối ưu hóa việc tải và hiển thị ảnh trong ứng dụng.
* shared\_preferences: Như đã đề cập ở trên, thư viện này giúp bạn lưu trữ dữ liệu tạm thời dạng key-value.

Các thư viện 3rd Party thường được quản lý thông qua hệ thống quản lý gói như pub.dev, nơi bạn có thể tìm và cài đặt các gói thư viện cho dự án Flutter của mình.

* **Behavior, Components**

1. **Behavior (Hành vi):**

Trong Flutter, "Behavior" đề cập đến cách mà các thành phần giao diện người dùng phản ứng và tương tác với hành động của người dùng hoặc các sự kiện xảy ra trong ứng dụng. Behavior quyết định cách mà các phần tử UI sẽ thay đổi, hiển thị thông tin, và tương tác với người dùng.

Ví dụ về các hành vi trong Flutter bao gồm:

* Button Behavior: Cách mà nút hoạt động khi được nhấn, ví dụ: khi người dùng nhấn vào nút, ứng dụng thực hiện một hành động cụ thể.
* Scrolling Behavior: Cách mà danh sách hoặc giao diện có khả năng cuộn reage khi người dùng vuốt lên hoặc xuống.
* Form Behavior: Cách mà các trường nhập liệu và nút gửi trong biểu mẫu tương tác với người dùng khi họ nhập liệu và ấn nút gửi.

1. **Components (Thành phần):**

Trong Flutter, "Components" là các phần tử xây dựng sẵn mà bạn có thể sử dụng để tạo giao diện người dùng của ứng dụng. Components là các khối xây dựng cơ bản để tạo thành các giao diện phức tạp. Các components có thể là các nút, biểu mẫu nhập liệu, danh sách, ảnh, v.v.

Ví dụ về một số thành phần trong Flutter:

* Button Components: ElevatedButton, TextButton, OutlinedButton là các thành phần nút với các phong cách khác nhau.
* Text Components: Text, RichText để hiển thị văn bản trong giao diện.
* Input Components: TextField, DropdownButton, Checkbox để nhập liệu từ người dùng.
* List Components: ListView, GridView để hiển thị danh sách hoặc lưới các phần tử.
* **Sate, Managements**

1. **State (Trạng thái):**

Trong Flutter, "State" đề cập đến các dữ liệu hoặc tình trạng hiện tại của một thành phần giao diện người dùng tại một thời điểm cụ thể. Ví dụ, trạng thái của một nút có thể là trạng thái được nhấn (pressed) hoặc không được nhấn (not pressed), trạng thái của một ứng dụng có thể là trạng thái đang đăng nhập hoặc đang đăng xuất.

1. **State Management (Quản lý trạng thái):**

"State Management" là quá trình quản lý, theo dõi và cập nhật trạng thái của các thành phần giao diện người dùng trong ứng dụng. Trong ứng dụng phức tạp, có thể có nhiều thành phần cần thay đổi trạng thái và cần đồng bộ hóa trạng thái giữa chúng. Quản lý trạng thái đúng cách là một phần quan trọng để đảm bảo hiệu suất và bảo trì của ứng dụng.

Flutter cung cấp một số phương pháp để quản lý trạng thái, bao gồm:

* setState(): Phương pháp đơn giản để cập nhật trạng thái của một StatefulWidget bằng cách gọi hàm setState(). Tuy nhiên, khi ứng dụng phức tạp hơn, việc sử dụng setState() có thể dẫn đến mã rối và khó quản lý.
* InheritedWidget: Đây là một phương pháp cổ điển để chia sẻ trạng thái giữa các thành phần con trong cây giao diện người dùng.
* Provider: Thư viện provider giúp bạn quản lý và chia sẻ trạng thái giữa các thành phần một cách hiệu quả hơn bằng cách sử dụng kiến trúc Provider.
* BLoC (Business Logic Component): Kiến trúc BLoC giúp bạn tách biệt logic kinh doanh và giao diện người dùng bằng cách sử dụng luồng dữ liệu.
* GetX: Một thư viện state management mới nhưng mạnh mẽ, giúp bạn quản lý trạng thái và định tuyến dễ dàng hơn.
* **Quality, Assurance**

1. **Quality (Chất lượng):**

Chất lượng trong Flutter liên quan đến cách mà ứng dụng được xây dựng, hoạt động và giao tiếp với người dùng. Điều này bao gồm việc đảm bảo rằng mã nguồn được viết đúng cách, giao diện người dùng được thiết kế một cách thẩm mỹ và trải nghiệm người dùng là mượt mà và dễ sử dụng.

1. **Quality Assurance (Đảm bảo chất lượng):**

Quality Assurance (QA) trong Flutter bao gồm một loạt các hoạt động và kiểm tra để đảm bảo rằng ứng dụng đáp ứng các tiêu chuẩn chất lượng. Điều này có thể bao gồm:

* Kiểm thử: Thực hiện các kiểm thử tự động và thủ công để đảm bảo rằng các chức năng và tính năng của ứng dụng hoạt động đúng như mong đợi và không có lỗi.
* Kiểm tra tương thích: Kiểm tra ứng dụng trên các thiết bị và môi trường khác nhau để đảm bảo rằng nó hoạt động một cách đáng tin cậy trên tất cả các nền tảng.
* Kiểm tra hiệu năng: Đảm bảo rằng ứng dụng có thời gian phản hồi nhanh, không gây ngốn tài nguyên quá mức và hoạt động mượt mà trong mọi tình huống.
* Kiểm tra bảo mật: Đảm bảo rằng ứng dụng được bảo vệ khỏi các lỗ hổng bảo mật và có các biện pháp bảo mật thích hợp.
* Kiểm tra tương tác người dùng: Đảm bảo rằng giao diện người dùng thân thiện và dễ sử dụng.
* **Version, Control**

1. **Version (Phiên bản):**

Phiên bản trong Flutter liên quan đến các biểu thị cho các phiên bản khác nhau của ứng dụng hoặc thư viện Flutter. Phiên bản thường được định dạng theo quy tắc major.minor.patch, ví dụ: 1.2.0. Trong đó:

* Major version thường thay đổi khi có các thay đổi lớn và không tương thích ngược với các phiên bản trước đó.
* Minor version thay đổi khi có các tính năng mới được thêm vào mà vẫn tương thích ngược.
* Patch version thay đổi khi có các sửa lỗi nhỏ hoặc cải thiện mà vẫn tương thích ngược.

1. **Version Control (Quản lý phiên bản):**

Version Control là quá trình quản lý và kiểm soát các thay đổi trong mã nguồn và tài liệu của dự án phần mềm theo thời gian. Một hệ thống quản lý phiên bản (VCS) cho phép bạn theo dõi lịch sử thay đổi, so sánh các phiên bản khác nhau, hợp nhất các thay đổi từ nhiều người cùng làm việc và phục hồi các phiên bản trước đó nếu cần.

Trong Flutter, các hệ thống quản lý phiên bản phổ biến bao gồm:

* Git: Là một hệ thống quản lý phiên bản phân tán rất phổ biến. Flutter và nhiều dự án phát triển phần mềm sử dụng Git để quản lý phiên bản mã nguồn.
* SVN (Subversion): Là một hệ thống quản lý phiên bản tập trung khác, nhưng hiện tại ít được sử dụng hơn so với Git.
* **Firebase**

Firebase trong Flutter là một bộ công cụ và dịch vụ được tích hợp sâu vào nền tảng Flutter để hỗ trợ nhà phát triển xây dựng ứng dụng di động một cách nhanh chóng và hiệu quả. Một số dịch vụ Firebase quan trọng trong Flutter bao gồm:

* Firebase Authentication: Cung cấp cơ chế xác thực người dùng thông qua email, số điện thoại, tài khoản Google, tài khoản Facebook và các cách khác.
* Cloud Firestore và Realtime Database: Các dịch vụ lưu trữ dữ liệu thời gian thực, cho phép bạn lưu trữ và đồng bộ dữ liệu giữa các thiết bị và người dùng.
* Firebase Storage: Dịch vụ lưu trữ tệp tin (hình ảnh, video, v.v.) trên đám mây, giúp quản lý và tải xuống các tệp tin trong ứng dụng.
* Firebase Cloud Messaging (FCM): Dịch vụ thông báo đẩy cho phép bạn gửi thông báo đến người dùng trên các thiết bị di động.
* Firebase Analytics: Cung cấp phân tích về việc sử dụng ứng dụng, giúp bạn hiểu rõ hơn về hành vi người dùng.
* Firebase Performance Monitoring: Giúp theo dõi hiệu suất ứng dụng và phát hiện vấn đề về tốc độ và tải trang.
* Firebase Remote Config: Cho phép bạn cập nhật cấu hình ứng dụng từ xa mà không cần phải phát hành lại ứng dụng.
* Firebase AdMob: Cho phép tích hợp quảng cáo vào ứng dụng để kiếm thu nhập.
* **Native, Integration**

1. **Native (Ứng dụng Native):**

Ứng dụng Native là những ứng dụng được phát triển bằng ngôn ngữ lập trình và công cụ phát triển chính của nền tảng cụ thể. Ví dụ, ứng dụng Android được viết bằng Java hoặc Kotlin, và ứng dụng iOS được viết bằng Swift hoặc Objective-C. Ứng dụng Native có thể sử dụng tối ưu hóa đối với các tính năng và chức năng cụ thể của nền tảng mà chúng chạy trên.

1. **Integration (Tích hợp):**

Tích hợp trong Flutter đề cập đến việc kết hợp hoặc sử dụng các thành phần, chức năng, hoặc tính năng từ hệ thống hoặc nền tảng gốc (native) vào ứng dụng Flutter. Điều này giúp bạn sử dụng những gì mà Flutter không cung cấp sẵn hoặc cần tích hợp với các tài nguyên native.

Ví dụ về tích hợp và sử dụng các thành phần native trong Flutter:

* Native Modules: Bạn có thể viết các mô-đun native (như plugin) để tích hợp các tính năng hoặc thư viện native vào ứng dụng Flutter của mình. Điều này cho phép bạn sử dụng các API và chức năng của hệ thống hoặc nền tảng.
* Platform Channels: Flutter cung cấp khả năng sử dụng Platform Channels để giao tiếp giữa mã Dart (Flutter) và mã native (Android hoặc iOS). Điều này giúp bạn truyền dữ liệu và lời gọi hàm giữa hai môi trường.
* Embedding Views: Bạn có thể nhúng các thành phần native như bản đồ hoặc trình phát video vào giao diện Flutter sử dụng các thư viện tương tác với nền tảng.
* Using Native SDKs: Bạn cũng có thể sử dụng các SDK native để tích hợp các dịch vụ như xác thực, quảng cáo, thanh toán, v.v. vào ứng dụng Flutter của bạn.