**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**< SharedPrerences trong Flutter**

**Lưu trữ dữ liệu đơn giản >**

**Sinh viên thực hiện:**

**01. Trương Văn Tú Lớp: 22KTMT1**  **MSSV:**  106220240

**02. Trần Đình Quang Nhật** **Lớp: 22KTMT2**  **MSSV:**  106220265

Người hướng dẫn:

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

**THUYẾT MINH**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**< SharedPrerences trong Flutter**

**Lưu trữ dữ liệu đơn giản >**

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | Trương Văn Tú | Tìm hiểu lí thuyết, code nội dung | 50% |
| 02 | Trần Đình Quang Nhật | Tìm hiểu lí thuyết, code nội dung | 50% |

Link code github: <https://github.com/Truongvantu06042004/topic2_dnt/blob/master/lib/main.dart>

**Mục lục**

**NỘI DUNG** 1

1. SharedPreference 1

1.1. Giới thiệu 1

1.2. Nguyên lí hoạt động 1

1.3. Cách sử dụng 1

2. Demo Dark Mode và Font Size 2

2.1. Sơ bộ về demo code 2

2.2. Kết quả demo 4

3. Các phương pháp lưu trữ khác 5

3.1. SQFLITE 5

3.1.1. Giới thiệu 5

3.1.2. Cách sử dụng 5

3.2. FIREBASE 7

3.2.1. Giới thiệu 7

3.2.2. Firebase Realtime Database & Firestore 8

3.2.2.1. Firebase Realtime Database 8

3.2.2.2. Firebase Cloud Firestore 10

3.3. Flutter Secure Storage: 11

3.3.1. Giới thiệu 11

3.3.2. Cách sử dụng 12

4. So sánh SharedPreferences với các phương thức lưu trữ khác: 12

5. Xử lý lỗi và Best practice: 13

5.1. Xử lý lỗi 13

5.2. Best practices 14

**TÀI LIỆU THAM KHẢO** 16

**NỘI DUNG**

**1. Shared Preferences**

**1.1. Giới thiệu**

- Shared Preferences là một plugin trong Flutter dùng để lưu trữ dữ liệu đơn giản dưới dạng key-value trên các nền tảng khác nhau.

- Đây là phương pháp lưu trữ dữ liệu không đồng bộ, giúp lưu trữ các dữ liệu nhỏ như cấu hình ứng dụng, trạng thái đăng nhập của người dùng.

- Shared Preferences không được khuyến khích sử dụng để lưu trữ các dữ liệu quan trọng do cơ chế ghi dữ liệu vào bộ nhớ máy. Nếu gỡ ứng dụng sẽ mất toàn bộ dữ liệu.

- Các kiểu dữ liệu được hỗ trợ bao gồm: int, double, bool, String và List<String>.

**1.2. Nguyên lí hoạt động**

SharedPreferences là cơ chế lưu trữ nhỏ, dạng key-value trên thiết bị, thích hợp cho dữ liệu như cấu hình, flag hoặc token tạm thời. Nguyên lý hoạt động:

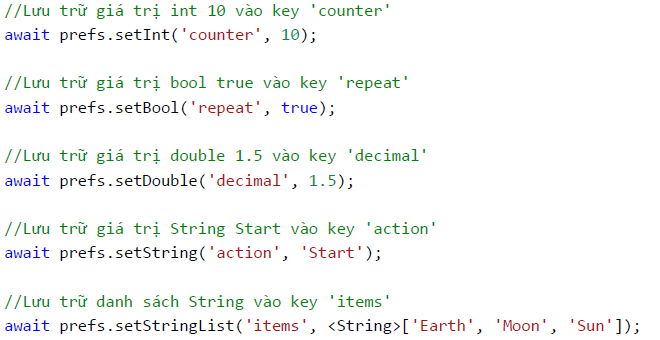
1. Lưu trữ theo key-value
   * Mỗi giá trị được gắn với một khóa duy nhất để truy xuất.
2. Lưu trên thiết bị
   * Android: lưu trong XML file tại /data/data/<package\_name>/shared\_prefs/.
   * iOS: lưu trong UserDefaults.
3. Không mã hóa dữ liệu
   * Dữ liệu plaintext, không nên lưu thông tin nhạy cảm.
4. Truy xuất dữ liệu
   * Plugin SharedPreferences cache dữ liệu trong bộ nhớ để truy xuất nhanh, đồng thời ghi vào file lưu trữ.
5. Đặc tính
   * Persistent: tồn tại khi ứng dụng tắt hoặc thiết bị khởi động lại.
   * Nhẹ và đơn giản, không thích hợp dữ liệu lớn hay phức tạp.

**1.3. Cách sử dụng**

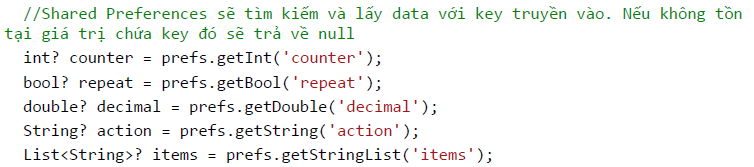
- Khởi tạo đối tượng Shared Preferences:



- Ghi dữ liệu (set):



- Đọc dữ liệu (get):



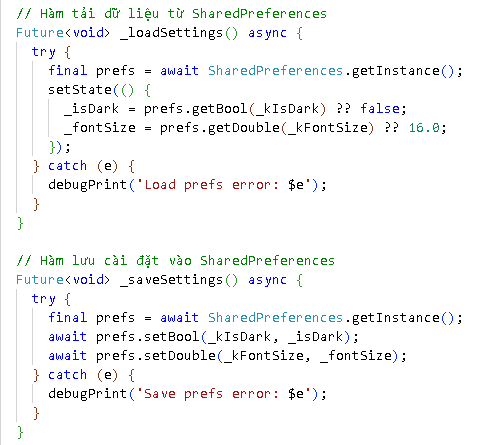
- Xóa dữ liệu (remove):



**2. Demo DarkMode và Font Size**

**2.1. Sơ bộ về demo code**

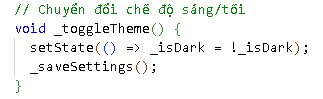
- Lưu và tải dữ liệu SharedPreferences



+ Khi app khởi động (initState), \_loadSettings() đọc dữ liệu đã lưu.

+ Khi thay đổi theme hoặc font, \_saveSettings() cập nhật giá trị lưu lại trong bộ nhớ cục bộ.

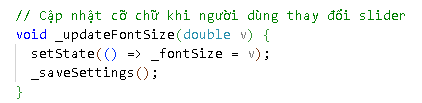
- Chuyển đổi theme Darkmode



+ Khi người dùng bật tắt **“Chế độ tối”** ở SwitchListTile, hàm này đảo trạng thái \_isDark

+ Gọi \_saveSettings() để ghi lại trạng thái mới

- Thay đổi Fontsize

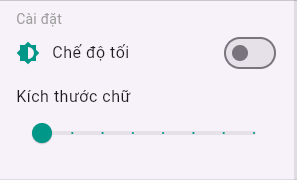


+ Khi trượt slider, giá trị font size cập nhật tức thời trong UI

+ Gọi \_saveSettings() để ghi lại trạng thái mới

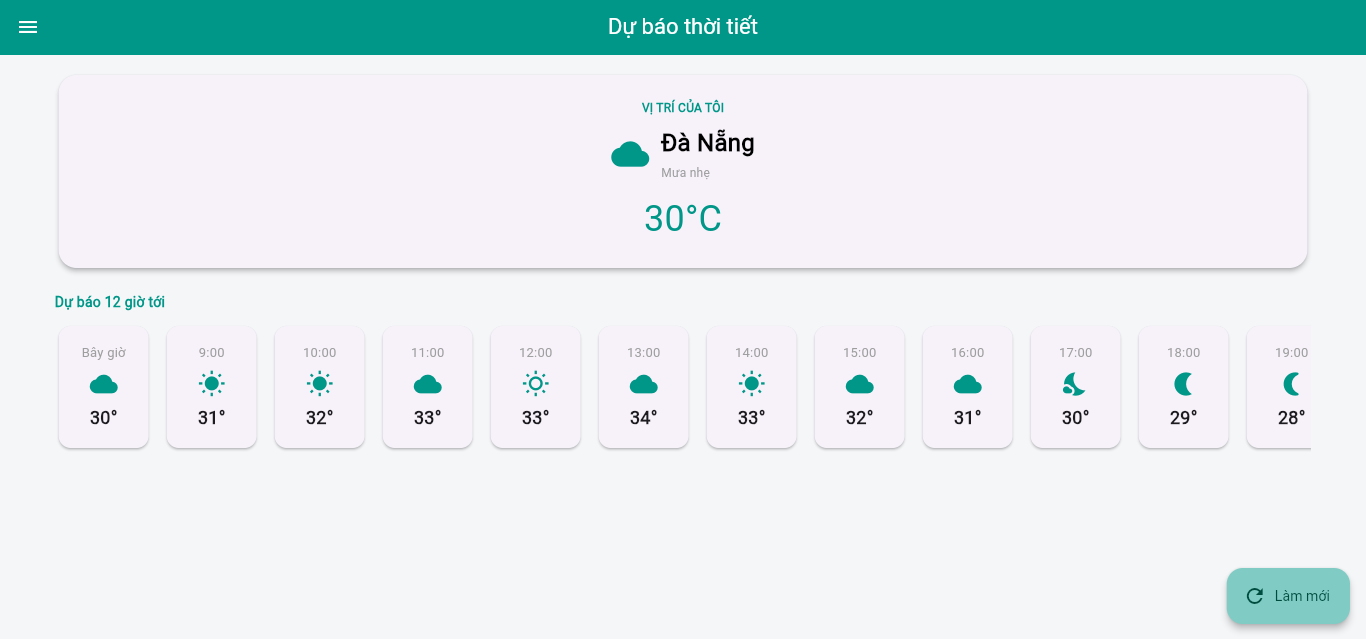
**2.2. Kết quả demo:**

- Giao diện cài đặt Darkmode và Fontsize



***Hình 1***. Giao diện cài đặt Darkmode và Fontsize

- Giao diện ứng dụng ở chế độ sáng và tối (Darkmode)

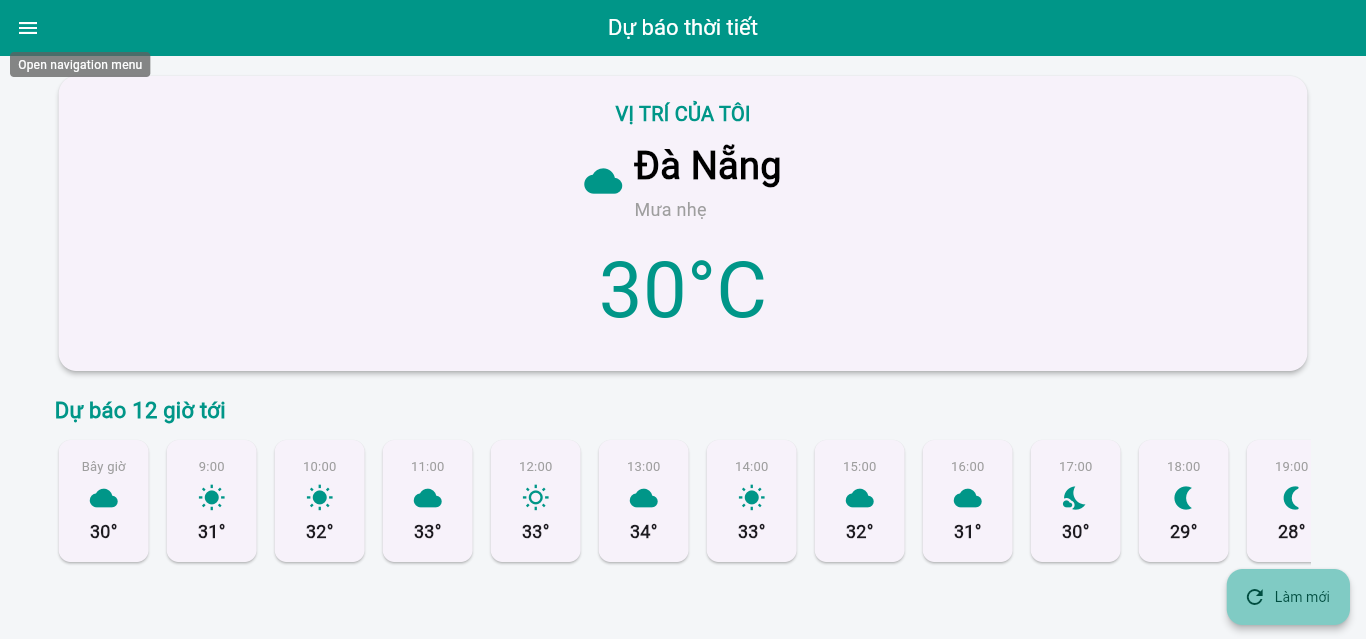
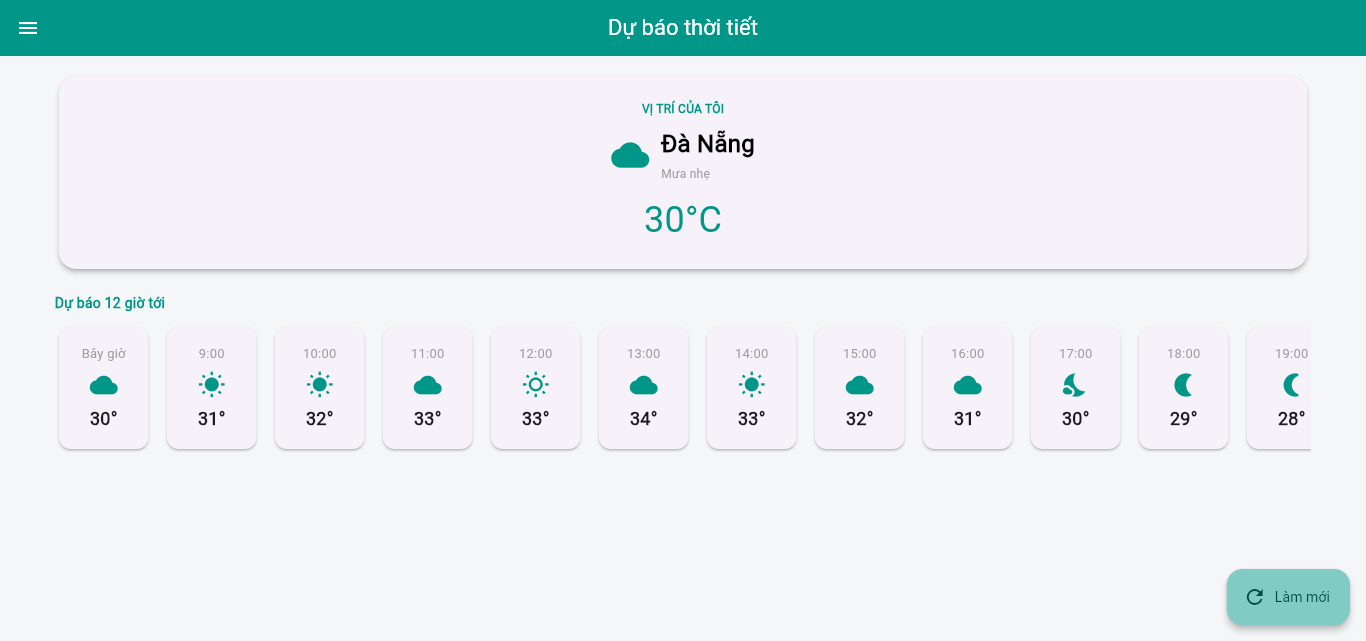


***Hình 2.*** Giao diện ứng dụng ở chế độ sáng



***Hình 3.*** Giao diện ứng dụng ở chế độ tối

- Giao diện ứng dụng khi thau đổi Fontsize



***Hình 4***. Giao diện ứng dụng khi thay đổi kích cỡ chữ (trái: 12 – phải: 26)

**3. Các phương pháp lưu trữ khác:**

**3.1. SQFLIFE:**

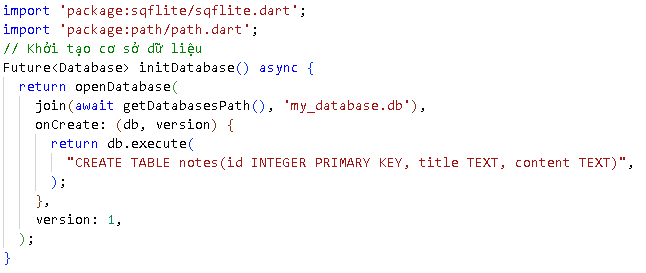
**3.1.1 Giới thiệu**

SQLite là một hệ quản trị cơ sở dữ liệu quan hệ nhúng, nhỏ gọn và dễ sử dụng, không cần thiết lập server riêng. Trong các ứng dụng di động, SQLite thường được dùng để lưu trữ dữ liệu cục bộ như thông tin người dùng, cài đặt ứng dụng hoặc các dữ liệu có cấu trúc khác. Trong Flutter, plugin phổ biến sqflite cho phép tương tác với SQLite một cách thuận tiện, cung cấp các phương thức để thực hiện các thao tác cơ bản như thêm, truy vấn, cập nhật và xóa dữ liệu

**3.1.2 Cách sử dụng:**

* Mở cơ sở dữ liệu

Cơ sở dữ liệu SQLite được lưu dưới dạng một tệp trên hệ thống tệp, với vị trí xác định bởi một đường dẫn. Nếu đường dẫn là tương đối, nó sẽ được hiểu là tương đối so với thư mục mặc định mà hàm getDatabasesPath() cung cấp.



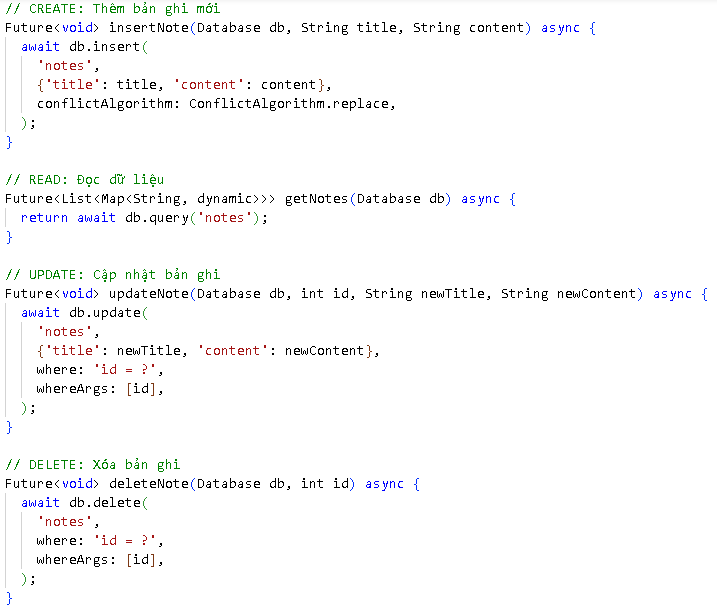
* Thực hiện các thao tác CRUD:

+ Create (INSERT): Thêm bản ghi mới vào bảng.

+ Read (SELECT): Truy vấn dữ liệu theo điều kiện.

+ Update (UPDATE): Cập nhật thông tin bản ghi.

+ Delete (DELETE): Xóa bản ghi không còn cần thiết.



**3.2 FIREBASE**

**3.2.1 Giới thiệu:**

Firebase là một nền tảng phát triển ứng dụng của Google, cung cấp nhiều dịch vụ hỗ trợ phát triển ứng dụng di động và web như xác thực người dùng, cơ sở dữ liệu thời gian thực, lưu trữ tệp tin, thông báo đẩy và nhiều dịch vụ khác. Firebase giúp các nhà phát triển dễ dàng xây dựng và mở rộng ứng dụng mà không cần quản lý cơ sở hạ tầng phía sau. Các dịch vụ chính của Firebase bao gồm:

− Firebase Authentication: Xác thực người dùng qua email, số điện thoại và các nhà cung cấp xác thực bên thứ ba như Google, Facebook, GitHub.

− Firebase Realtime Database: Cơ sở dữ liệu NoSQL lưu trữ và đồng bộ hóa dữ liệu giữa người dùng theo thời gian thực.

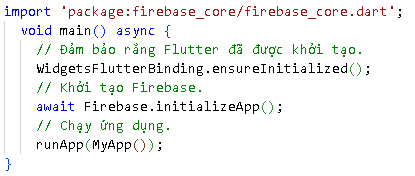
− Firebase Firestore: Cơ sở dữ liệu NoSQL hiện đại hơn, hỗ trợ truy vấn mạnh mẽ và mở rộng tốt hơn.

− Firebase Storage: Lưu trữ và chia sẻ tệp tin.

− Firebase Cloud Messaging: Dịch vụ thông báo đẩy.

− Firebase Analytics: Công cụ phân tích ứng dụng.

Cách khởi tạo Firebase trong hàm main



**3.2.2 Firebase Realtime Database & Firestore**

**3.2.2.1 Firebase Realtime Database**

Firebase Realtime Database là cơ sở dữ liệu NoSQL cho phép lưu trữ và đồng bộ hóa dữ liệu theo thời gian thực giữa các thiết bị, phù hợp cho các ứng dụng yêu cầu cập nhật dữ liệu ngay lập tức như chat, trò chuyện hoặc các ứng dụng cộng tác trực tuyến.

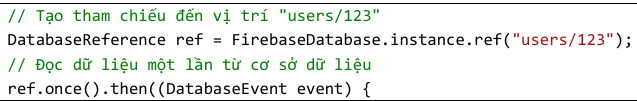
Các khái niệm chính

− DatabaseReference: Tham chiếu đến vị trí của dữ liệu trong cơ sở dữ liệu. Mỗi DatabaseReference có thể được sử dụng để thực hiện các thao tác đọc và ghi dữ liệu.

− DatabaseEvent: Sự kiện được phát ra khi dữ liệu tại vị trí tham chiếu thay đổi.

Các cú pháp thường được sử dụng:

- Đọc dữ liệu





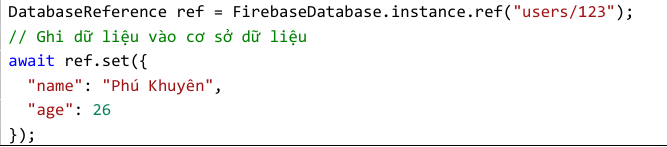
Trong đó:

+ DatabaseReference là tham chiếu đến vị trí của dữ liệu trong cơ sở dữ liệu.

+ once() được sử dụng để lấy dữ liệu một lần từ cơ sở dữ liệu và không lắng nghe các thay đổi sau đó.

+ event.snapshot.value chứa dữ liệu từ cơ sở dữ liệu.

- Ghi dữ liệu



Hàm set() được sử dụng để lưu trữ dữ liệu tại vị trí chỉ định, thay thế bất kỳ dữ liệu nào đã tồn tại tại vị trí đó.

- Xoá dữ liệu



Hàm remove() được sử dụng để xoá dữ liệu tại vị trí chỉ định.

**3.2.2.2 Firebase Cloud Firestore**

Firestore là cơ sở dữ liệu NoSQL hiện đại hơn, cung cấp khả năng truy vấn mạnh mẽ và mở rộng tốt hơn so với Realtime Database. Nó phù hợp cho các ứng dụng cần quản lý dữ liệu phức tạp và yêu cầu truy vấn linh hoạt, ví dụ như quản lý dữ liệu người dùng, các ứng dụng thương mại điện tử...

Các khái niệm chính

− Collection: Một tập hợp các tài liệu (documents). Mỗi Collection có thể chứa nhiều tài liệu.

− Document: Một tài liệu chứa dữ liệu dưới dạng cặp khóa-giá trị (key-value). Mỗi Document nằm trong một Collection.

Các cú pháp thường được sử dụng:

- Đọc dữ liệu

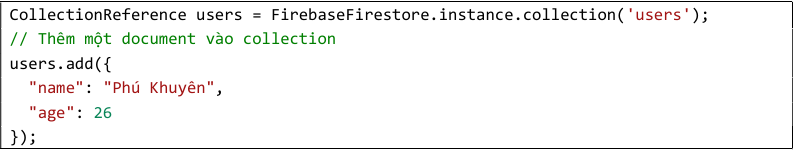


Trong đó:

+ CollectionReference là tham chiếu đến một tập hợp các document trong Firestore.

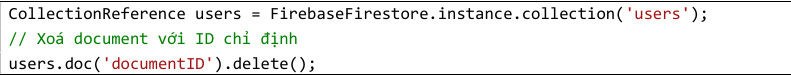
+ get() được sử dụng để lấy tất cả các document trong collection một lần.

- Ghi dữ liệu



Hàm add() được sử dụng để thêm một document mới với một ID tự động tạo vào collection.

- Xoá dữ liệu:



Hàm delete() được sử dụng để xoá document với ID chỉ định.

**3.3. Flutter Secure Storage:**

**3.3.1. Giới thiệu**

Flutter Secure Storage là một thư viện của Flutter cho phép lưu trữ dữ liệu nhạy cảm (như mật khẩu, token, API key) một cách an toàn, bằng cách mã hóa dữ liệu trước khi lưu trên thiết bị. Đây là giải pháp thay thế an toàn cho SharedPreferences khi cần bảo vệ thông tin quan trọng của người dùng.

## Đặc điểm

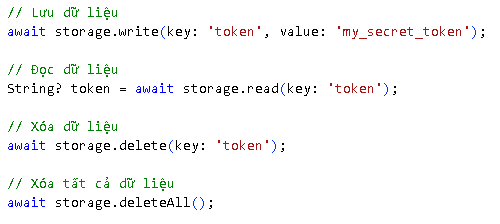
* **Lưu trữ dạng key-value:** Dữ liệu được lưu dưới dạng cặp khóa – giá trị.
* **Mã hóa tự động:** Dữ liệu được mã hóa trước khi lưu, đảm bảo an toàn.
* **Hỗ trợ đa nền tảng:**
  + iOS: lưu trên **Keychain**
  + Android: lưu trên **Keystore**
* **Kích thước dữ liệu nhỏ:** Thích hợp lưu trữ dữ liệu nhạy cảm nhưng không phù hợp với dữ liệu lớn.
* **Dễ sử dụng:** API đơn giản, tương tự SharedPreferences.

**3.3.2. Cách sử dụng**

Để thao tác với dữ liệu, bạn cần khởi tạo một đối tượng FlutterSecureStorage. Đối tượng này sẽ cung cấp các phương thức đọc, ghi và xóa dữ liệu một cách an toàn.



Dữ liệu được lưu dưới dạng **key-value**, đồng thời được **mã hóa tự động**. Ví dụ:



Như vậy, với một đối tượng storage, bạn có thể **quản lý dữ liệu nhạy cảm một cách toàn diện,** từ lưu, truy xuất đến xóa bỏ khi không cần thiết nữa.

**4. So sánh SharedPreferences với các phương thức lưu trữ khác:**

***Bảng 1.*** Bảng so sánh các phương thức lưu trữ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Phương thức** | **Loại dữ liệu** | **Lưu offline** | **Lưu online** | **Đặc điểm** | **Ưu điểm** | **Nhược điểm** |
| **Shared**  **Preferences** | Key-value, dữ liệu nhỏ (int, double, bool, String, List<String>) | ✅ | ❌ | Lưu cấu hình, flag, token | Dễ sử dụng, đơn giản, nhanh | Không lưu dữ liệu lớn, không phức tạp, không bảo mật cao |
| **SQLite / SQFlite** | Bảng, hàng, dữ liệu quan hệ | ✅ | ❌ | Cơ sở dữ liệu quan hệ nhúng, truy vấn SQL | Lưu dữ liệu phức tạp, hỗ trợ truy vấn | Cần thiết kế bảng, không tự đồng bộ online |
| **Firebase Realtime Database** | JSON key-value, dữ liệu real-time | ✅ (cache offline) | ✅ | Đồng bộ dữ liệu real-time, hỗ trợ multi-user | Dữ liệu đồng bộ ngay lập tức, dễ dùng cho app online | Cấu trúc dữ liệu phức tạp khi lớn, phụ thuộc internet |
| **Firebase Cloud Firestore** | Document, collection, JSON | ✅ (cache offline) | ✅ | Dữ liệu dạng document, hỗ trợ truy vấn mạnh hơn Realtime DB | Hỗ trợ query, filter, offline, đồng bộ | Chi phí cao hơn, phức tạp hơn Realtime DB |
| **Flutter Secure Storage** | Key-value, dữ liệu nhạy cảm (token, mật khẩu, API key) | ✅ | ❌ | Dữ liệu được mã hóa tự động, lưu trữ an toàn trên Keychain (iOS) / Keystore (Android) | Bảo mật cao, dễ sử dụng, quản lý dữ liệu nhạy cảm | Chỉ lưu dữ liệu nhỏ, không thích hợp dữ liệu lớn hoặc truy vấn phức tạp |

**5. Xử lý lỗi và Best practice:**

**5.1. Xử lý lỗi:**

Khái niệm: Xử lý lỗi là quá trình phát hiện và kiểm soát các tình huống bất thường có thể xảy ra khi ứng dụng đang chạy, nhằm đảm bảo ứng dụng không bị dừng đột ngột và người dùng vẫn có trải nghiệm mượt mà. Trong Flutter, điều này thường được thực hiện bằng **try-catch** hoặc **Future.catchError()** khi làm việc với các thao tác bất đồng bộ.

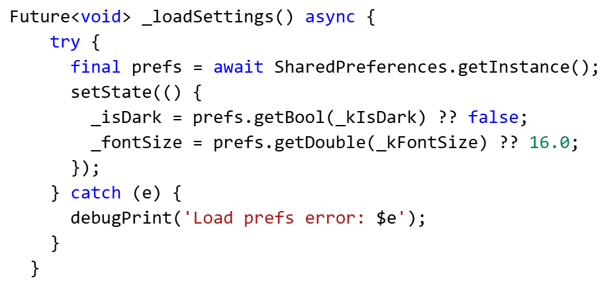
Mục đích:

+ Giúp ứng dụng hoạt động ổn định ngay cả khi gặp lỗi.

+ Hỗ trợ lập trình viên dễ dàng tìm và khắc phục lỗi trong quá trình phát triển.

+ Tăng độ tin cậy và trải nghiệm người dùng.

Ví dụ về xử lí lỗi:



Trong đó:

+ try: chứa các đoạn mã có khả năng gây lỗi (ví dụ như truy cập dữ liệu).

+ catch (e): bắt lại lỗi nếu có, giúp chương trình không bị dừng.

+ debugPrint(): in ra thông tin lỗi trong quá trình debug để dễ dàng kiểm tra.

**5.2. Best practices:**

Khái niệm: “Best Practices” là những quy tắc hoặc phương pháp được cộng đồng lập trình công nhận là hiệu quả và tối ưu nhất trong việc phát triển phần mềm.  
Chúng giúp code dễ đọc, dễ bảo trì, giảm lỗi và tăng hiệu suất của ứng dụng.

Một số Best Practices trong ứng dụng Flutter sử dụng Shared Preferences được nêu trong bảng sau:

***Bảng 2.*** Best Practices khi làm việc với Shared Preferences

|  |  |
| --- | --- |
| Thực hành tốt | Giải thích |
| 1. Sử dụng async/await | Khi làm việc với dữ liệu bất đồng bộ (như Shared Preferences), dùng await giúp mã lệnh thực hiện tuần tự và dễ đọc hơn. |
| 2. Phân tách chức năng rõ ràng | Tách riêng các hàm như \_loadSettings() và \_saveSettings() giúp dễ bảo trì, kiểm thử và mở rộng. |
| 3. Xử lý lỗi bằng try-catch | Giúp đảm bảo an toàn cho ứng dụng, tránh crash khi gặp sự cố. |
| 4. Gọi setState() hợp lý | Cập nhật lại UI khi dữ liệu thay đổi để hiển thị chính xác. |
| 5. Khai báo hằng số cho key | Dùng biến const như \_kIsDark, \_kFontSize giúp tránh gõ sai key và dễ quản lý khi mở rộng ứng dụng. |
| 6. Không lưu dữ liệu nhạy cảm | Vì Shared Preferences không mã hóa, nên không dùng để lưu mật khẩu hoặc thông tin cá nhân quan trọng. |
| 7. Tối ưu hóa việc truy cập dữ liệu | Không nên gọi SharedPreferences.getInstance() quá thường xuyên; nên lưu đối tượng prefs tạm thời để dùng lại. |

**TÀI LIỆU THAM KHẢO**

[1] ThS. Bùi Phú Khuyên, Giáo trình “LẬP TRÌNH TRÊN THIẾT BỊ DI ĐỘNG”, Trường Đại học Công nghệ TP.HCM (HUTECH), 2024

[2] Flutter Community, "shared\_preferences package," Pub.dev. [Online]. Available: <https://pub.dev/packages/shared_preferences>

[3] QuickCoder, "Choose the best data storage solution for your Flutter app," QuickCoder, Mar. 12, 2024. [Online]. Available: [https://quickcoder.org/choose-the-best-data-storage-solution-for-your-flutter-app/](https://quickcoder.org/choose-the-best-data-storage-solution-for-your-flutter-app/?utm_source=chatgpt.com)

[4] Flutter, "Best practices for Flutter performance," Flutter.dev. [Online]. Available: <https://docs.flutter.dev/perf/best-practices>