**ĐẠI HỌC ĐÀ NẴNG**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ - VIỄN THỐNG**

  



**BÁO CÁO TUẦN**

**LẬP TRÌNH ĐA NỀN TẢNG**

**CHỦ ĐỀ:**

**SHARED PREFERENCES TRONG FLUTTER – LƯU TRỮ DỮ LIỆU ĐƠN GIẢN**

|  |  |
| --- | --- |
| **Sinh viên thực hiện:** | **Trần Quang Khánh (106220220)**  **Hồ Nguyên Tâm (106220233)** |
| **Lớp học phần:** | **22.44** |
| **Lớp sinh hoạt:** | **22KTMT1** |
| **GVHD:** | **TS. Nguyễn Duy Nhật Viễn** |

  

**TP. Đà Nẵng, 14/10/2025**

**BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Họ và tên sinh viên | Số thẻ sinh viên | Lớp sinh hoạt | Phân công nhiệm vụ | Phần trăm đóng góp |
| Trần Quang Khánh | 106220220 | 22KTMT1 | ​Viết phần Giải thích nguyên lý hoạt động của SharedPreferences, viết phần Demo ứng dụng lưu trữ cài đặt dark mode, font size | 50% |
| Hồ Nguyên Tâm | 106220233 | ​Viết phần So sánh với các phương thức lưu trữ khác, viết phần Xử lý lỗi và best practices | 50% |
| Tổng | | | | 100% |

Link code github: <https://github.com/Tam472022/Demo_SharedPreferences_7>

**MỤC LỤC**

[**1. Nguyên lí hoạt động của Shared Preferences: 3**](#_Toc212406719)

[**1.1. Khái niệm: 3**](#_Toc212406720)

[**1.2. Cơ chế hoạt động: 3**](#_Toc212406721)

[**1.3. Tổng kết: 5**](#_Toc212406722)

[**2. Demo ứng dụng lưu trữ cài đặt dark mode, font size: 5**](#_Toc212406723)

[**2.1. Code cho phần chính (Đọc dữ liệu từ SharedPreferences): 5**](#_Toc212406724)

[**2.2. Cây Widget thể hiện giao diện của các trang phục vụ demo: 6**](#_Toc212406725)

[**2.3. Kết quả đạt được: 8**](#_Toc212406726)

[**2.4. Tổng kết: 9**](#_Toc212406727)

[**3. So sánh với các phương thức lưu trữ: 10**](#_Toc212406728)

[**3.1. File Storage: 10**](#_Toc212406729)

[**3.2. SQLite / Moor / Drift: 11**](#_Toc212406730)

[**3.3. Hive / ObjectBox: 12**](#_Toc212406731)

[**3.4. Secure Storage: 13**](#_Toc212406732)

[**3.5. Cloud Firestore / Firebase: 14**](#_Toc212406733)

[**3.6. Đề xuất lựa chọn phương thức lưu trữ trong Flutter: 15**](#_Toc212406734)

[**3.7. Tổng kết: 16**](#_Toc212406735)

[**4. Xử lý lỗi và Best Practices: 17**](#_Toc212406736)

[**4.1. Không dùng MODE\_PRIVATE khi tạo SharedPreferences: 17**](#_Toc212406737)

[**4.2. Lưu tất cả dữ liệu vào một file duy nhất: 17**](#_Toc212406738)

[**4.3. Sử dụng apply() cho mọi trường hợp: 18**](#_Toc212406739)

[**4.4. Lưu dữ liệu quá lớn hoặc phức tạp: 18**](#_Toc212406740)

[**4.5. Không xử lý khi cập nhật phiên bản ứng dụng (migration): 19**](#_Toc212406741)

[**4.6. Gọi SharedPreferences trực tiếp ở nhiều nơi trong ứng dụng: 19**](#_Toc212406742)

[**4.7. Không hủy listener khi Activity bị hủy: 20**](#_Toc212406743)

[**4.8. Tổng kết: 21**](#_Toc212406744)

[**5. Tóm lược nội dung chính: 21**](#_Toc212406745)

[**Tài liệu tham khảo: 22**](#_Toc212406746)

# **1. Nguyên lí hoạt động của Shared Preferences:**

## **1.1. Khái niệm:**

SharedPreferences là một cơ chế lưu trữ dữ liệu cục bộ trong Flutter (và Android nói chung), cho phép ứng dụng lưu trữ dữ liệu dưới dạng các cặp khóa – giá trị (key – value).

Ngoài ra, SharedPreferences còn cho phép lưu và truy xuất các thông tin nhỏ, có tính cài đặt hoặc trạng thái.

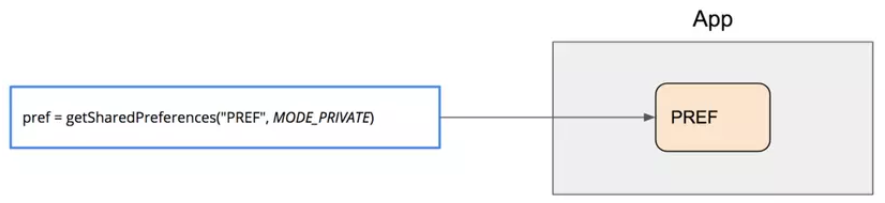
Ứng dụng: chế độ hiển thị (dark/light mode), ngôn ngữ, tên người dùng đã đăng nhập gần nhất... [1]

## **1.2. Cơ chế hoạt động:**

Cơ chế hoạt động của SharedPreferences gồm 3 bước chính:

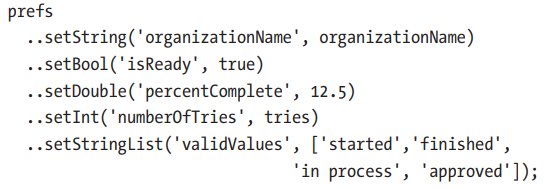
* Bước 1: Truy cập vào bộ nhớ thiết bị: Dùng hàm getInstance() để kiểm tra xem bộ nhớ đã có file dữ liệu SharedPreferences hay chưa. Nếu chưa có, nó sẽ tạo mới file mặc định. Cú pháp sử dụng được minh họa qua Hình 1:



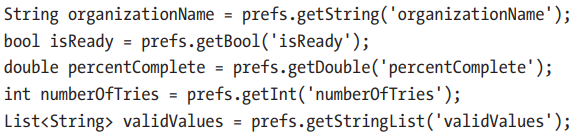


*Hình 1. Cú pháp sử dụng hàm getInstance() để kiểm tra bộ nhớ file dữ liệu SharedPreferences và hình ảnh minh họa cách truy cập dữ liệu SharedPreferences.*

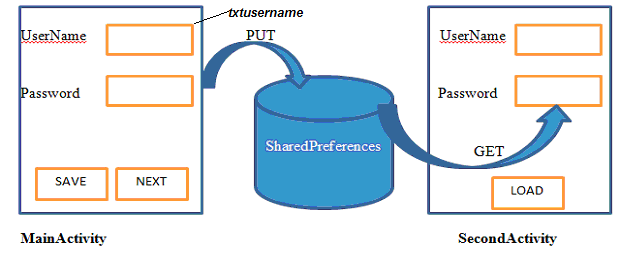
* Bước 2: Truy xuất dữ liệu [2]: Dữ liệu được truy xuất theo cặp key-value. Mặt khác, các thao tác này được thực hiện bất đồng bộ. Tức là, giao diện chính vẫn chạy như thường lệ, nhưng thao tác đọc/ghi vẫn diễn ra trên thread phụ (dùng Future) nhằm tránh gây đơ chương trình làm ảnh hưởng đến độ mượt mà của chương trình. Cú pháp khai báo được minh họa qua Hình 2, Hình 3 [2] và hoạt động truy xuất dữ liệu từ SharedPreferences được minh họa qua Hình 4:



*Hình 2. Hình ảnh ví dụ minh họa khai báo cú pháp khai báo việc lưu dữ liệu của cặp key-value ra SharedPreferences.*



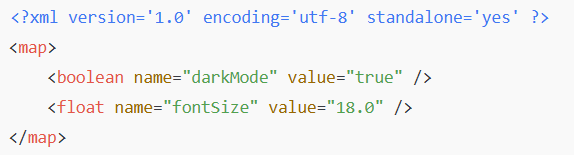
*Hình 3. Hình ảnh ví dụ minh họa khai báo cú pháp khai báo việc đọc dữ liệu của cặp key-value từ SharedPreferences.*



*Hình 4. Hình ảnh ví dụ minh họa hoạt động truy xuất dữ liệu giữa ứng dụng với dữ liệu SharedPreferences.*

* Bước 3: Lưu xuống hệ thống tệp: Flutter sẽ gửi dữ liệu xuống nền tảng gốc:
  + Android: Ghi trong file có đuôi là .xml
  + IOS: lưu vào UserDefaults.

Khi gọi hàm getInstance sau này, lớp native sẽ đọc file XML/UserDefaults và lưu dưới dạng key-value cho ứng dụng đang chạy. Các ví dụ minh họa về cú pháp lưu trữ dữ liệu dưới dạng key-value qua file cho hệ điều hành Android hoặc iOS được minh họa qua Hình 5 và Hình 6 [5] dưới đây:



*Hình 5. Hình ảnh ví dụ minh họa về cú pháp lưu trữ dữ liệu dưới dạng key-value thông qua file có đuôi là .xml để sử dụng cho Android.*



*Hình 6. Hình ảnh ví dụ minh họa về cú pháp lưu trữ dữ liệu dưới dạng key-value thông qua UserDefaults để sử dụng cho iOS [5].*

## **1.3. Tổng kết:**

Phần này đã trình bày cơ chế hoạt động về lưu trữ dữ liệu cục bộ của Flutter bằng SharedPreferences, bao gồm: nguyên lí hoạt động, cách truy xuất, cách nó lưu trữ ra file đuôi .xml đối với Android và iOS. Phần này giúp hiểu rõ cách SharedPreferences vận hành, từ khâu khởi tạo, đọc/ghi cho đến việc lưu xuống dữ liệu hệ thống tệp. Đồng thời, minh họa bằng cú pháp và hình ảnh thực tế.

# **2.** **Demo ứng dụng** **lưu trữ cài đặt dark mode, font size:**

## **2.1. Code cho phần chính (Đọc dữ liệu từ SharedPreferences):**

Các hàm cần thiết:

* Hàm initState(): Dùng để gọi \_loadSettings() khi mở trang ứng dụng và đọc dữ liệu đã lưu ra từ trước.
* Hàm \_loadSettings(): Tuân theo các bước một cách tuần tự, cụ thể: Truy cập instance trong SharedPreferences. Sau đó, thực hiện việc đọc các giá trị key darkMode, fontSize từ bộ nhớ.
* Hàm setState() được gọi ra dùng để cập nhật UI sau khi có được dữ liệu key-value.
* Hàm \_saveSettings(): Ghi lại giá trị \_isDarkMode, \_fontSize xuống bộ nhớ.

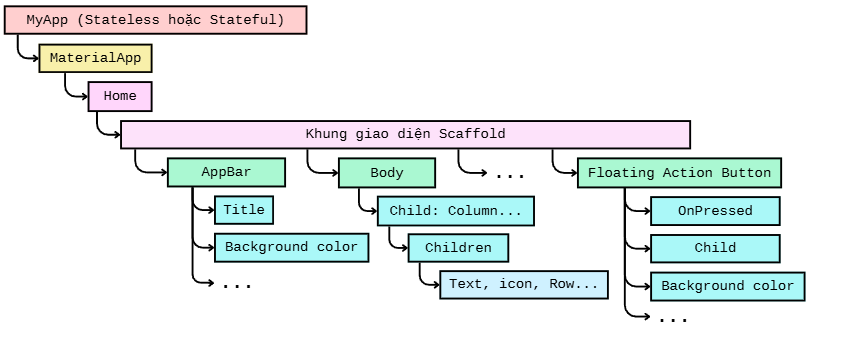
Đoạn code cho việc tương tác giữa ứng dụng với SharedPreferences được minh họa qua hình 7 dưới đây:



*Hình 7. Đoạn code cho việc tương tác giữa ứng dụng với SharedPreferences.*

## **2.2. Cây Widget thể hiện giao diện của các trang phục vụ demo:**

Nhắc lại cây Widget: cây widget tổng quát được vẽ theo hình dưới đây (phát triển thêm từ tài liệu [1]):



*Hình 8. Cây widget tổng quát dùng để xây giao diện.*

Dựa vào Hình 8, ta thấy cấu trúc cây widget tổng quát của một ứng dụng Flutter khi sử dụng Scaffold. Cây widget thể hiện mối quan hệ cha – con giữa các thành phần giao diện, bắt đầu từ widget gốc của ứng dụng đến các phần tử con. Cụ thể:

* Widget MyApp (Stateless hoặc Stateful) là điểm khởi đầu của ứng dụng, bao gồm MaterialApp – nơi quản lý chủ đề, điều hướng và trang Home. Bên trong Home là Scaffold, đóng vai trò là khung bố cục chính cho giao diện Material Design.
* Scaffold bao gồm các phần chính như:
  + AppBar: thanh tiêu đề, chứa các thành phần như title hoặc nút thao tác.
  + Body: vùng nội dung chính, có thể chứa các widget bố cục như Column, Row hoặc ListView,....
  + FloatingActionButton: nút hành động nổi, có các thuộc tính như onPressed, child, backgroundColor,...

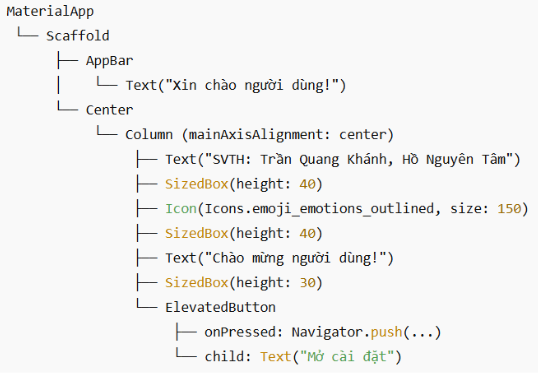
Áp dụng đối với ứng dụng cần demo: ứng dụng được chia ra làm 2 giao diện chính: 1 là giao diện cài đặt hiển thị, 2 là giao diện của màn hình chính.

Cây Widget sử dụng dùng để xây dựng giao diện cài đặt ứng dụng được minh họa qua Hình 9 dưới đây:



*Hình 9. Cây Widget sử dụng dùng để xây dựng giao diện cài đặt ứng dụng.*

Cây Widget sử dụng dùng để xây dựng giao diện màn hình chính được minh họa qua Hình 10 dưới đây:



*Hình 10. Cây Widget sử dụng dùng để xây dựng giao diện màn hình trang chủ (màn hình chính).*

Giới thiệu sơ lược về Navigation:

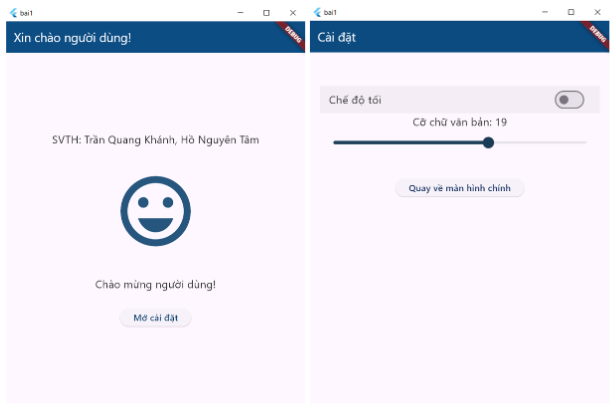
* Navigation dùng để chuyển giữa các màn hình (Page/Screen).
* Flutter quản lý các màn hình bằng Navigator Stack (ngăn xếp):
  + Navigator.push() → chuyển sang trang mới, đẩy trang hiện tại vào stack.
  + Navigator.pop() → quay lại trang trước, lấy trang ra khỏi stack.
* Cú pháp sử dụng Navigation giữa 2 màn hình được mô tả qua Hình 11 dưới đây:



*Hình 11. Cú pháp sử dụng Navigation để chuyển đổi qua lại giữa 2 màn hình.*

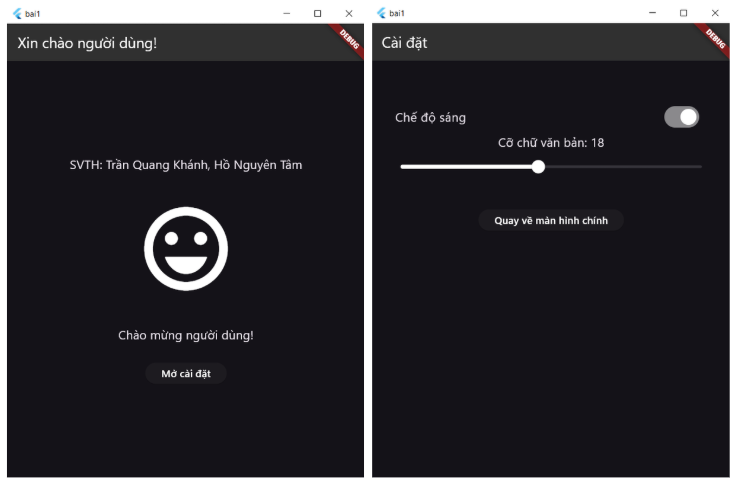
## **2.3. Kết quả đạt được:**

* Ban đầu, ứng dụng chạy ở chế độ sáng, cỡ chữ là 19.
* Giao diện chính hiển thị thông tin sinh viên thực hiện và nút “Mở cài đặt”. Khi người dùng chọn “Mở cài đặt”, người dùng có thể: bật chế độ tối, thay đổi cỡ chữ văn bản.
* Sau khi quay lại, giao diện sẽ cập nhật ngay theo những gì mà người dùng đã lưu ở phần cài đặt.
* Hình ảnh minh họa về kết quả cho chế độ sáng được minh họa qua Hình 12 dưới đây:



*Hình 12. Hình ảnh kết quả minh họa về chế độ sáng cũng như thông tin cài đặt về cỡ chữ là 19 mà người dùng tinh chỉnh.*

* Còn khi người dùng bật chế độ tối, giao diện màn hình sẽ chuyển sang màu nền tối – chữ sáng.
* Mặt khác, khi người dùng chỉnh cỡ chữ xuống 18, toàn bộ giao diện màn hình sẽ được giảm xuống còn 18, áp dụng cho toàn bộ văn bản.
* Khi quay lại màn hình chính, các thay đổi được lưu trữ và hiển thị ngay nhờ cơ chế SharedPreferences.
* Hình ảnh minh họa về kết quả cho chế độ tối được minh họa qua Hình 13 dưới đây:



*Hình 13. Hình ảnh kết quả minh họa về chế độ tối cũng như thông tin cài đặt về cỡ chữ là 18 mà người dùng tinh chỉnh.*

## **2.4. Tổng kết:**

Phần demo đã minh họa rõ cách ứng dụng Flutter sử dụng SharedPreferences để lưu trữ và truy xuất các thông tin cài đặt giao diện.

Khi người dùng thay đổi chế độ hiển thị (dark/light) hoặc cỡ chữ, các giá trị này được ghi lại vào bộ nhớ cục bộ thông qua SharedPreferences, sau đó được đọc lại và áp dụng tự động khi người dùng quay lại màn hình chính hoặc khởi động lại ứng dụng.

Kết quả cho thấy SharedPreferences hoạt động ổn định, nhanh và hiệu quả trong việc lưu trữ dữ liệu nhỏ.

Qua đó, có thể khẳng định rằng SharedPreferences là giải pháp phù hợp cho các ứng dụng Flutter cần lưu cài đặt cá nhân hóa hoặc trạng thái người dùng, giúp cải thiện trải nghiệm và tính liên tục của ứng dụng.

# **3.** **So sánh với các phương thức lưu trữ:**

SharedPreferences trong Flutter là phương thức lưu trữ dữ liệu dạng key–value đơn giản, thường dùng cho các thông tin nhỏ như cài đặt người dùng, chế độ giao diện, ngôn ngữ hay trạng thái đăng nhập. Cơ chế của nó là ghi dữ liệu vào bộ nhớ trong dưới dạng file XML, cho phép truy xuất nhanh và dễ dàng giữa các lần mở ứng dụng. Tuy nhiên, SharedPreferences chỉ phù hợp cho dữ liệu nhỏ, không bảo mật và không thể lưu trữ danh sách hay đối tượng phức tạp. Vì vậy, ngoài SharedPreferences, Flutter còn cung cấp nhiều phương thức lưu trữ khác như Secure Storage, File Storage, SQLite, Hive hay Firebase — mỗi phương thức lại có ưu điểm và phạm vi ứng dụng riêng, tùy theo loại dữ liệu và yêu cầu của ứng dụng.

## **3.1. File Storage:**

A screenshot of a phone

AI-generated content may be incorrect.

*Hình 14. Hình ảnh minh họa về lưu trữ bằng File Storage.*

**Dạng dữ liệu phù hợp:**

File Storage dùng để lưu trữ dữ liệu dạng chuỗi, JSON, text hoặc file nhị phân. Phù hợp trong các trường hợp cần lưu trữ dữ liệu linh hoạt, nhiều cấu hình hoặc dữ liệu không theo cấu trúc cố định. [2]

**Kích thước:**  
Dung lượng trung bình (từ vài trăm KB đến vài MB).

**Ưu điểm:**

* Linh hoạt, có thể lưu bất kỳ kiểu dữ liệu nào như JSON, text hoặc binary.
* Dễ dàng tùy chỉnh định dạng file theo nhu cầu.
* Không giới hạn cấu trúc dữ liệu, thuận tiện cho việc export hoặc import.

**Nhược điểm:**

* Cần tự xử lý quá trình đọc, ghi và parse dữ liệu JSON thủ công.
* Dễ xảy ra lỗi định dạng nếu quá trình ghi file không đúng.
* Không có sẵn cơ chế truy vấn hoặc kiểm tra dữ liệu.

**Khi nên dùng:**

Thích hợp cho các trường hợp lưu file cấu hình, cache tạm thời hoặc dữ liệu JSON không quá phức tạp, ví dụ: lưu danh sách tạm, file log, hoặc dữ liệu text nội bộ.

## **3.2. SQLite / Moor / Drift:**

A blue screen with a blue background

AI-generated content may be incorrect.

*Hình 15. Hình ảnh minh họa về lưu trữ bằng SQLite.*

**Dạng dữ liệu phù hợp:**

Lưu trữ dữ liệu có cấu trúc dạng bảng, danh sách hoặc dữ liệu có mối quan hệ giữa các bảng. Phù hợp với các ứng dụng quản lý thông tin như sách, sinh viên, sản phẩm, v.v [6].

**Kích thước:**

Dung lượng lớn, có thể lên đến hàng trăm MB dữ liệu.

**Ưu điểm [1]:**

* Hỗ trợ truy vấn mạnh mẽ bằng SQL hoặc thông qua ORM.
* Có khả năng lọc, sắp xếp, thống kê và phân trang dữ liệu.
* Tổ chức dữ liệu có cấu trúc rõ ràng, dễ mở rộng.
* Flutter hỗ trợ tốt qua các thư viện như sqflite hoặc drift (trước đây là Moor).

**Nhược điểm:**

* Cần thiết lập cấu trúc bảng, cột và kiểu dữ liệu phức tạp.
* Yêu cầu hiểu biết về cú pháp SQL hoặc thư viện ORM.
* Việc thay đổi cấu trúc dữ liệu (migration) khá phức tạp.

**Khi nên dùng [1]:**

Phù hợp cho các ứng dụng quản lý dữ liệu lớn như Library Management, To-do List, Danh bạ hoặc các ứng dụng bán hàng.

## **3.3. Hive / ObjectBox:**

A logo with blue and black text

AI-generated content may be incorrect.

*Hình 16. Hình ảnh minh họa về lưu trữ bằng Hive.*

**Dạng dữ liệu phù hợp:**

Lưu trữ dữ liệu dạng object theo hướng lập trình hướng đối tượng (OOP), có thể lưu trực tiếp các class hoặc model mà không cần chuyển đổi sang JSON [1].

**Kích thước:**

Dung lượng lớn, hiệu suất cao, phù hợp cho ứng dụng offline có nhiều dữ liệu.

**Ưu điểm:**

* Lưu object nhanh, không cần định nghĩa SQL hoặc schema phức tạp.
* Truy cập nhanh hơn SQLite, đặc biệt với dữ liệu phi cấu trúc.
* Hoạt động ổn định trong môi trường offline.
* Dễ sử dụng, có thư viện hỗ trợ tốt trong Flutter (như Hive).

**Nhược điểm:**

* Cần học cách cài đặt và sử dụng thư viện riêng.
* Không hỗ trợ truy vấn phức tạp như SQL.
* Một số kiểu dữ liệu yêu cầu adapter để chuyển đổi trước khi lưu.

**Khi nên dùng:**

Phù hợp với các ứng dụng cần lưu dữ liệu offline dạng object, ví dụ như ứng dụng ghi chú, danh sách công việc hoặc ứng dụng không sử dụng server.

## **3.4. Secure Storage:**

A screen shot of a computer

AI-generated content may be incorrect.

*Hình 16. Hình ảnh minh họa về lưu trữ bằng Secure Storage.*

**Dạng dữ liệu phù hợp:**  
Dành cho dữ liệu nhạy cảm như token, mật khẩu, thông tin tài khoản hoặc khóa mã hóa [1].

**Kích thước:**  
Dung lượng rất nhỏ (chỉ vài KB).

**Ưu điểm:**

* Dữ liệu được mã hóa an toàn, lưu trong keychain (iOS) hoặc keystore (Android).
* Dữ liệu chỉ có thể truy cập bởi chính ứng dụng.
* Đảm bảo mức độ bảo mật cao.

**Nhược điểm:**

* Cài đặt phức tạp hơn SharedPreferences.
* Tốc độ truy cập chậm hơn do cần giải mã dữ liệu.
* Chỉ nên dùng cho dữ liệu có dung lượng nhỏ.

**Khi nên dùng:**  
Thích hợp để lưu trữ token đăng nhập, mật khẩu, khóa bảo mật hoặc mã xác thực. Có thể kết hợp với SharedPreferences để đảm bảo vừa tiện lợi vừa an toàn [1].

## **3.5. Cloud Firestore / Firebase:**

A computer diagram of folders

AI-generated content may be incorrect.

*Hình 16. Hình ảnh minh họa về lưu trữ bằng Cloud Firestore.*

**Dạng dữ liệu phù hợp:**

Lưu trữ dữ liệu online, có thể đồng bộ giữa nhiều thiết bị hoặc người dùng. Hỗ trợ lưu object, danh sách hoặc dữ liệu thời gian thực [2] [3].

**Kích thước:**

Linh hoạt, phụ thuộc vào tài khoản Firebase (có thể lên đến hàng GB).

**Ưu điểm [3]:**

* Đồng bộ dữ liệu real-time giữa các thiết bị.
* Tích hợp sẵn các tính năng xác thực và bảo mật (Firebase Auth).
* Dễ dàng mở rộng sang Cloud Storage để lưu file lớn.
* Có thể xem và chỉnh sửa dữ liệu trực tiếp trên Firebase Console.

**Nhược điểm [2]:**

* Yêu cầu kết nối Internet liên tục.
* Giới hạn dung lượng miễn phí, nếu vượt sẽ bị tính phí.
* Cần cấu hình Firebase và quản lý API key cẩn thận.

**Khi nên dùng:**

Phù hợp cho các ứng dụng cần chia sẻ dữ liệu giữa nhiều người dùng, ví dụ như ứng dụng chat, ghi chú online, bán hàng hoặc quản lý tài khoản trực tuyến [3].

## **3.6. Đề xuất lựa chọn phương thức lưu trữ trong Flutter:**

Từ những định nghĩa và ứng dụng của SharedPreferences và các phương thức lưu trữ dữ liệu trong Flutter ở trên, ta sẽ có bảng đề xuất lựa chọn phương thức lưu trữ phù hợp với từng hình huống như Bảng 3.1 sau:

*Bảng 1: Bảng đề xuất lựa chọn phương thức lưu trữ trong Flutter*

|  |  |  |
| --- | --- | --- |
| **Tình huống / Nhu cầu lưu trữ** | **Phương thức đề xuất** | **Giải thích lý do chọn** |
| Lưu cài đặt người dùng (chế độ dark mode, ngôn ngữ, kích thước font) | SharedPreferences | Dữ liệu nhỏ, dạng key–value, đọc ghi nhanh, không cần bảo mật cao. |
| Lưu token đăng nhập, mật khẩu, thông tin nhạy cảm | Secure Storage | Dữ liệu nhỏ, nhưng yêu cầu bảo mật cao — được mã hóa bằng keychain/keystore. |
| Lưu file cấu hình, dữ liệu tạm dạng JSON/text | File Storage (Local) | Dữ liệu vừa, không có cấu trúc phức tạp, có thể định dạng tùy ý (JSON, text). |
| Lưu danh sách lớn, dữ liệu có cấu trúc (sách, user, sản phẩm) | SQLite / Moor / Drift | Quản lý tốt dữ liệu dạng bảng, hỗ trợ truy vấn SQL, lọc, sắp xếp, cập nhật. |
| Lưu object hoặc model phức tạp theo hướng OOP | Hive / ObjectBox | Hiệu suất cao, lưu trực tiếp object mà không cần SQL hoặc JSON parsing. |
| Lưu dữ liệu cần đồng bộ giữa nhiều thiết bị hoặc người dùng | Cloud Firestore / Firebase | Dữ liệu online, tự động đồng bộ real-time, dễ chia sẻ và mở rộng. |
| Lưu cache hoặc dữ liệu tạm để tăng hiệu năng | File Storage / Hive | Dễ ghi/xóa nhanh, không yêu cầu truy vấn hoặc bảo mật. |
| Ứng dụng yêu cầu vừa bảo mật vừa nhanh (token + cài đặt) | Kết hợp Secure Storage + SharedPreferences | Token lưu bằng Secure Storage, cài đặt lưu SharedPreferences để tối ưu. |

## **3.7. Tổng kết:**

*Bảng 2: Bảng tổng hợp so sánh các phương thức lưu trữ trong Flutter*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Phương thức** | **Dữ liệu phù hợp** | **Kích thước** | **Ưu điểm** | **Nhược điểm** | **Khi nên dùng** |
| **SharedPreferences** | Key–Value nhỏ (bool, string, int, double) | Nhỏ (<1MB) | Dễ dùng, nhanh, lưu tự động | Không lưu object, không bảo mật | Cài đặt người dùng, token, trạng thái |
| **File (local)** | Chuỗi, JSON, text | Trung bình (MB) | Linh hoạt, tùy chỉnh định dạng | Dễ lỗi định dạng, phải parse thủ công | Lưu text, cache tạm |
| **SQLite / Moor / Drift** | Bảng dữ liệu, danh sách | Lớn | Truy vấn mạnh, có cấu trúc | Cài đặt phức tạp | App quản lý dữ liệu (sách, danh bạ) |
| **Hive / ObjectBox** | Object (OOP) | Lớn | Lưu object nhanh, không cần SQL | Cần học thư viện riêng | App offline nhiều dữ liệu |
| **Secure Storage** | Dữ liệu nhạy cảm (token, password) | Nhỏ | Mã hóa an toàn | Cài đặt phức tạp hơn | Lưu thông tin đăng nhập |
| **Cloud Firestore / Firebase** | Dữ liệu online | Linh hoạt | Đồng bộ real-time, đa thiết bị | Cần mạng, quota hạn chế | Dữ liệu chia sẻ, đa người dùng |

Trong Flutter, mỗi phương thức lưu trữ dữ liệu có ưu và nhược điểm riêng, phù hợp với từng loại dữ liệu và mục đích sử dụng khác nhau. **SharedPreferences** thường được dùng để lưu trữ dữ liệu nhỏ dạng key–value như cài đặt người dùng, chế độ hiển thị hay token đăng nhập vì đơn giản và truy cập nhanh. Với dữ liệu nhạy cảm cần bảo mật cao hơn, **Secure Storage** là lựa chọn phù hợp nhờ cơ chế mã hóa. Nếu cần lưu trữ nội dung dạng văn bản, JSON hoặc file nhỏ, **File Storage** mang lại sự linh hoạt và dễ tùy chỉnh. **SQLite** hoặc **Drift** phù hợp cho các ứng dụng có dữ liệu lớn, có cấu trúc và cần truy vấn phức tạp như quản lý danh bạ hoặc sản phẩm. Trong khi đó, **Hive** hoặc **ObjectBox** thích hợp với ứng dụng offline lưu trữ nhiều object vì tốc độ cao và dễ sử dụng. Cuối cùng, với ứng dụng cần đồng bộ dữ liệu giữa nhiều thiết bị, **Cloud Firestore / Firebase** là giải pháp tối ưu.

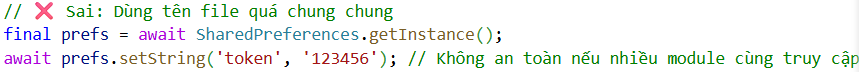
Tóm lại, không có phương thức lưu trữ nào là tốt nhất cho mọi trường hợp; việc lựa chọn phụ thuộc vào nhu cầu: dữ liệu nhỏ nên dùng SharedPreferences, dữ liệu nhạy cảm dùng Secure Storage, dữ liệu có cấu trúc dùng SQLite hoặc Hive, còn dữ liệu cần đồng bộ online nên dùng Firebase.

# **4. Xử lý lỗi và Best Practices:**

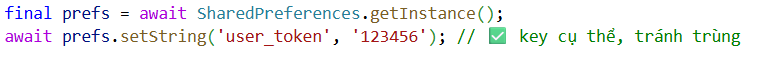
Trong quá trình sử dụng SharedPreferences trong Flutter (hoặc Android), nếu không tuân thủ đúng nguyên tắc, ứng dụng có thể gặp lỗi liên quan đến bảo mật, hiệu năng hoặc quản lý dữ liệu. Dưới đây là các lỗi thường gặp cùng hướng khắc phục và thực hành tốt nhất:

## **4.1. Không dùng MODE\_PRIVATE khi tạo SharedPreferences:**

**Vấn đề:** Nếu không dùng Context.MODE\_PRIVATE, dữ liệu có thể bị ứng dụng khác truy cập hoặc chỉnh sửa, gây mất an toàn.



**Cách khắc phục:** Luôn khởi tạo SharedPreferences với Context.MODE\_PRIVATE (đây là chế độ mặc định, chỉ app hiện tại được phép truy cập dữ liệu).



## **4.2. Lưu tất cả dữ liệu vào một file duy nhất:**

**Vấn đề:** File SharedPreferences có thể trở nên quá lớn, làm chậm truy cập và dễ xảy ra lỗi ghi đè key.

A computer code with text

AI-generated content may be incorrect.

**Cách khắc phục:** Chia nhỏ dữ liệu theo chức năng, tạo nhiều file SharedPreferences khác nhau, ví dụ:

* user\_prefs để lưu thông tin người dùng
* settings\_prefs để lưu cài đặt giao diện
* cache\_prefs để lưu dữ liệu tạm thời

A screen shot of a computer code

AI-generated content may be incorrect.

## **4.3. Sử dụng apply() cho mọi trường hợp:**

**Vấn đề:** apply() ghi dữ liệu bất đồng bộ (asynchronous). Nếu app tắt ngay sau khi gọi, dữ liệu có thể chưa được ghi xong → dẫn đến mất dữ liệu.

A close up of a text

AI-generated content may be incorrect.

**Cách khắc phục:**

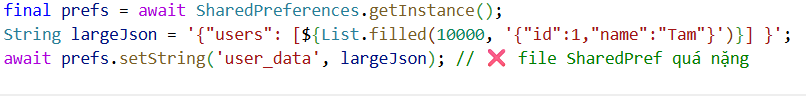
* Dùng commit() khi cần chắc chắn dữ liệu được ghi ngay lập tức (đồng bộ).
* Dùng apply() khi không quá quan trọng về thời gian ghi, ưu tiên tốc độ.

A close up of a text

AI-generated content may be incorrect.

## **4.4. Lưu dữ liệu quá lớn hoặc phức tạp:**

**Vấn đề:** SharedPreferences chỉ thích hợp cho dữ liệu nhỏ (kiểu int, String, bool). Nếu lưu danh sách lớn, JSON hoặc hình ảnh, ứng dụng dễ bị lag hoặc crash.



**Cách khắc phục:**

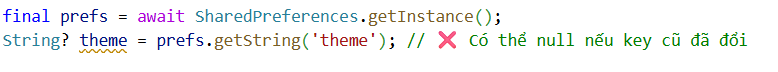
* Chỉ lưu các dữ liệu nhỏ, đơn giản (boolean, int, string).
* Với dữ liệu phức tạp hoặc lớn, nên dùng SQLite, Hive hoặc File Storage.

A close-up of a computer screen

AI-generated content may be incorrect.

## **4.5. Không xử lý khi cập nhật phiên bản ứng dụng (migration):**

**Vấn đề:** Khi app cập nhật version mới, key hoặc cấu trúc dữ liệu có thể thay đổi → gây lỗi đọc dữ liệu cũ.



**Cách khắc phục:**

* Lưu version\_code trong SharedPreferences
* Khi phát hiện version thay đổi, thực hiện cập nhật hoặc xóa dữ liệu cũ để tránh lỗi không tương thích.

A screen shot of a computer code

AI-generated content may be incorrect.

## **4.6. Gọi SharedPreferences trực tiếp ở nhiều nơi trong ứng dụng:**

**Vấn đề:** Gây trùng key, khó bảo trì và kiểm soát luồng dữ liệu.

A screen shot of a computer code

AI-generated content may be incorrect.

**Cách khắc phục:**

* Tạo lớp quản lý tập trung (Singleton), ví dụ PreferencesManager, để đọc/ghi dữ liệu thống nhất.
* Việc này giúp dễ mở rộng, sửa lỗi và tái sử dụng code.

A computer code with colorful text

AI-generated content may be incorrect.

## **4.7. Không hủy listener khi Activity bị hủy:**

**Vấn đề:** Nếu đăng ký OnSharedPreferenceChangeListener nhưng không hủy khi Activity bị destroy, có thể gây memory leak.

A close up of a text

AI-generated content may be incorrect.

**Cách khắc phục:** Gọi unregisterOnSharedPreferenceChangeListener() trong phương thức onDestroy() để giải phóng tài nguyên.

A screen shot of a computer code

AI-generated content may be incorrect.

## **4.8. Tổng kết:**

*Bảng 3: Bảng các sai lầm thường gặp và cách khắc phục*

|  |  |
| --- | --- |
| **Sai lầm thường gặp** | **Cách khắc phục / Best Practice** |
| Không dùng MODE\_PRIVATE → Dữ liệu có thể bị truy cập bởi app khác | Luôn tạo với Context.MODE\_PRIVATE để bảo mật dữ liệu |
| Lưu tất cả dữ liệu vào một file duy nhất → File lớn, truy cập chậm, dễ ghi đè key | Tách nhiều file SharedPreferences theo chức năng (user, settings, cache, …) |
| Dùng apply() cho mọi trường hợp → Có thể mất dữ liệu khi app tắt đột ngột | Dùng commit() khi cần chắc chắn ghi dữ liệu, apply() khi ưu tiên tốc độ |
| Lưu dữ liệu lớn (list, JSON, hình ảnh) → App lag, crash | Chỉ lưu dữ liệu nhỏ (boolean, int, string…), còn lại dùng SQLite / Room |
| Không có cơ chế migration khi cập nhật app → Dữ liệu cũ không tương thích, lỗi đọc key | Lưu và kiểm tra version trong SharedPreferences, xóa hoặc cập nhật dữ liệu khi cần |
| Gọi SharedPreferences trực tiếp ở nhiều nơi → Khó quản lý, trùng key, khó bảo trì | Tạo lớp Singleton PreferencesManager để quản lý truy cập tập trung |
| Không hủy listener khi Activity bị destroy → Gây memory leak | Dùng unregisterOnSharedPreferenceChangeListener() trong onDestroy() |

Khi sử dụng SharedPreferences, các lỗi thường gặp chủ yếu liên quan đến bảo mật, hiệu năng và bảo trì như: không dùng MODE\_PRIVATE, lưu dữ liệu quá lớn, dùng apply() sai cách hoặc quản lý key không thống nhất.

Để khắc phục, cần tuân thủ best practices: luôn dùng MODE\_PRIVATE, chia nhỏ file theo chức năng, chọn commit() hoặc apply() phù hợp, chỉ lưu dữ liệu nhỏ, thêm cơ chế cập nhật khi đổi version, quản lý tập trung qua PreferencesManager và hủy listener khi không dùng.

Tuân thủ những nguyên tắc này giúp SharedPreferences an toàn, ổn định và dễ bảo trì hơn trong ứng dụng Flutter.

# **5.** **Tóm lược nội dung chính:**

SharedPreferences là một giải pháp nhẹ và tiện lợi để lưu trữ dữ liệu nhỏ dạng key–value trong Flutter, thường được dùng cho các thông tin như cài đặt người dùng, token đăng nhập hoặc trạng thái ứng dụng. Cơ chế hoạt động của nó dựa trên việc lưu dữ liệu vào bộ nhớ trong của thiết bị dưới dạng XML hoặc JSON, giúp truy xuất nhanh và dễ dàng.

Qua ví dụ ứng dụng lưu dark mode và font size, ta thấy SharedPreferences giúp người dùng giữ nguyên tùy chỉnh cá nhân ngay cả khi tắt app. Khi so sánh với các phương thức lưu trữ khác, SharedPreferences phù hợp nhất cho dữ liệu nhỏ, không cần cấu trúc phức tạp; trong khi các công cụ khác như SQLite, Hive hay Firebase phù hợp hơn cho dữ liệu lớn, bảo mật cao hoặc cần đồng bộ.

Cuối cùng, việc tuân thủ best practices như dùng MODE\_PRIVATE, tách file theo chức năng, chỉ lưu dữ liệu nhỏ và quản lý tập trung qua lớp PreferencesManager sẽ giúp ứng dụng ổn định, bảo mật và dễ bảo trì hơn.

# **Tài liệu tham khảo:**

[1] Nguyễn Duy Nhật Viễn, Slide bài giảng – Môn Lập trình đa nền tảng, Trường Đại học Bách Khoa Đà Nẵng, 2025.

[2] R. Payne, Beginning App Development with Flutter: Create Cross-Platform Mobile Apps, Apress, 2021.

[3] M. Napoli, Beginning Flutter: A Hands-On Guide to App Development, Wiley, 2020.

[4] Flutter Team, Flutter: Architectural Overview, Flutter Documentation, 2025.

[5] W. Byczkowski, “Local data storage on iOS – When and how to use it? Keychain, Core Data, and other options,” Holdapp Blog, 25 Jul. 2023.

[6] J. Horton, Android Programming for Beginners, Second Edition, Packt Publishing, 2018.