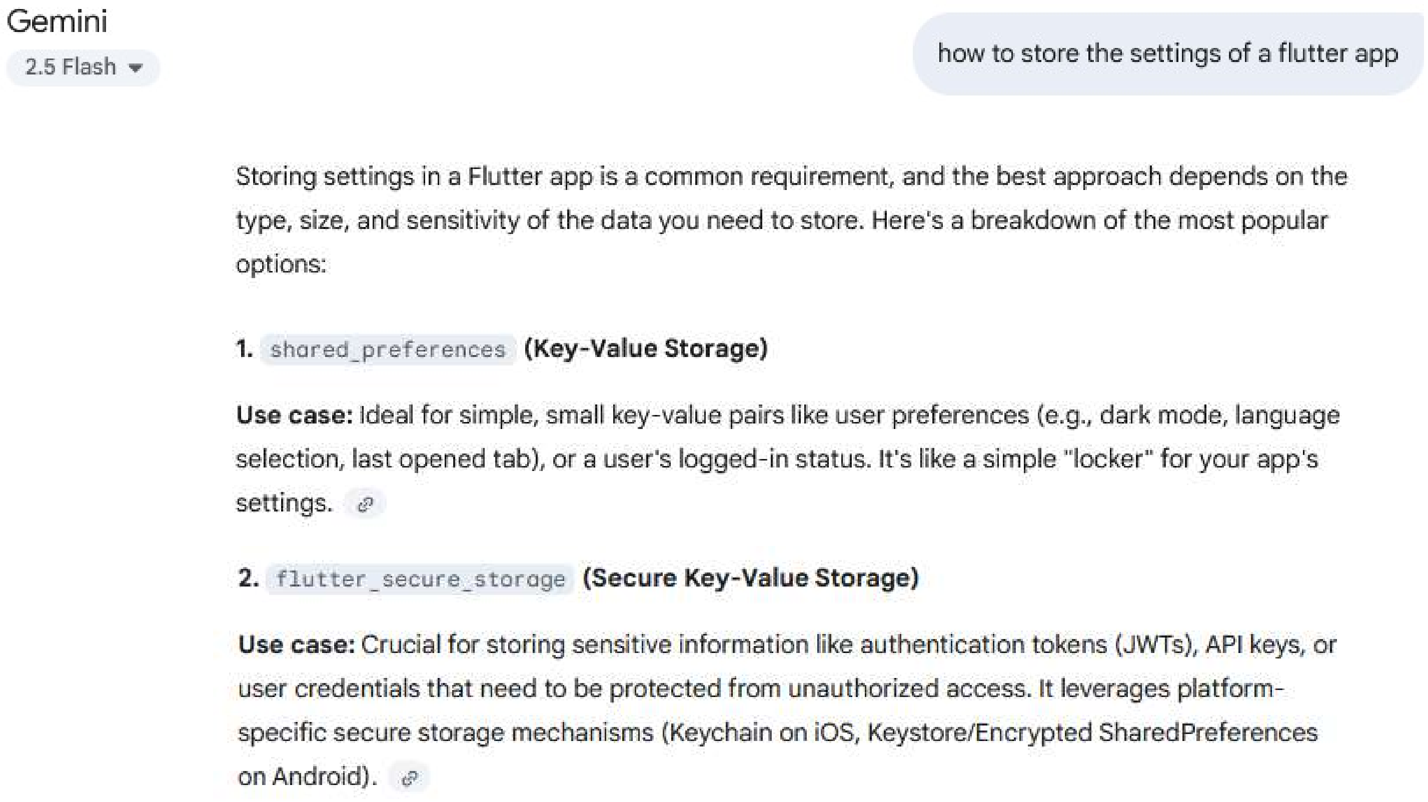
# How to store the settings of a Flutter app



how to store the settings of a flutter app

Storing settings in a Flutter app is a common requirement, and the best approach depends on the type, size, and sensitivity of the data you need to store. Here's a breakdown of the most popular options:

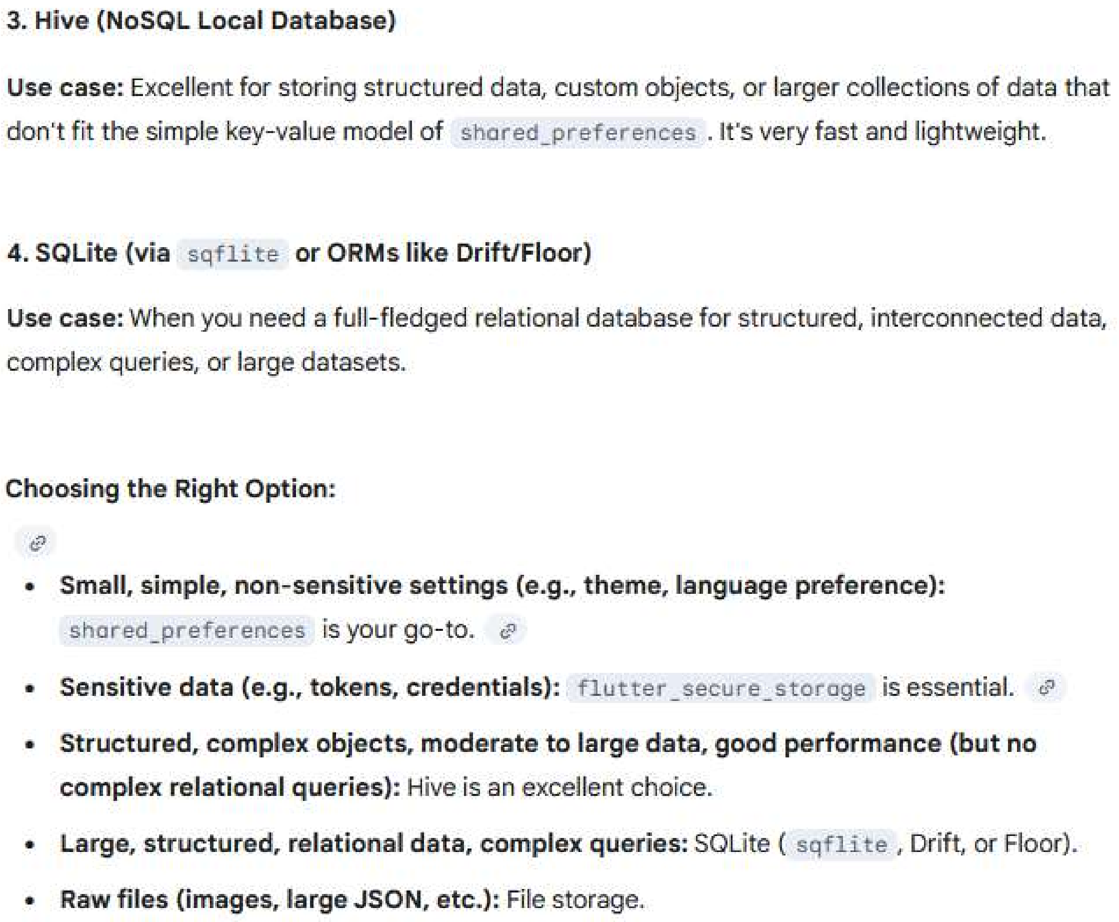
**1. `shared_preferences` (Key-Value Storage)**

**Use case:** Ideal for simple, small key-value pairs like user preferences (e.g., dark mode, language selection, last opened tab), or a user's logged-in status. It's like a simple "locker" for your app's settings.

**2. `flutter_secure_storage` (Secure Key-Value Storage)**

**Use case:** Crucial for storing sensitive information like authentication tokens (JWTs), API keys, or user credentials that need to be protected from unauthorized access. It leverages platform-specific secure storage mechanisms (Keychain on iOS, Keystore/Encrypted SharedPreferences on Android).

# More alternatives to shared_preferences



### 3. Hive (NoSQL Local Database)

**Use case:** Excellent for storing structured data, custom objects, or larger collections of data that don't fit the simple key-value model of `shared_preferences`. It's very fast and lightweight.

### 4. SQLite (via `sqflite` or ORMs like Drift/Floor)

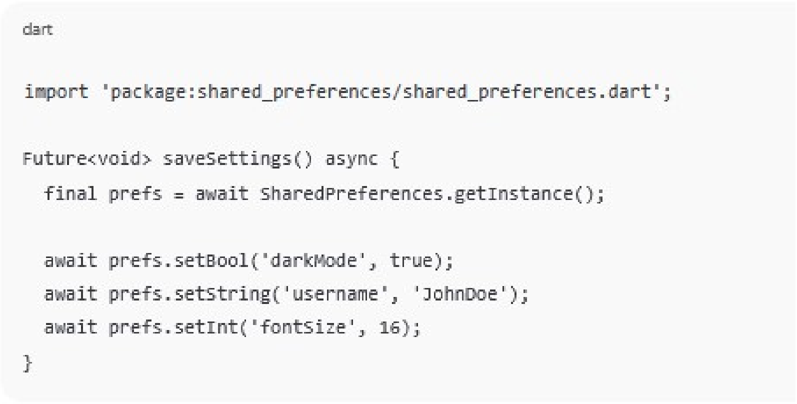
**Use case:** When you need a full-fledged relational database for structured, interconnected data, complex queries, or large datasets.

### Choosing the Right Option:

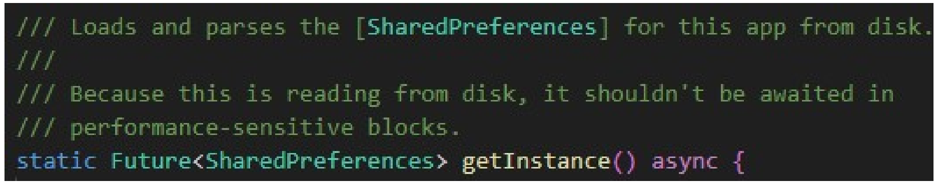
- **Small, simple, non-sensitive settings (e.g., theme, language preference):** `shared_preferences` is your go-to.
- **Sensitive data (e.g., tokens, credentials):** `flutter_secure_storage` is essential.
- **Structured, complex objects, moderate to large data, good performance (but no complex relational queries):** Hive is an excellent choice.
- **Large, structured, relational data, complex queries:** SQLite ( `sqflite` , Drift, or Floor).
- **Raw files (images, large JSON, etc.):** File storage.

# Code samples for storing and retrieving

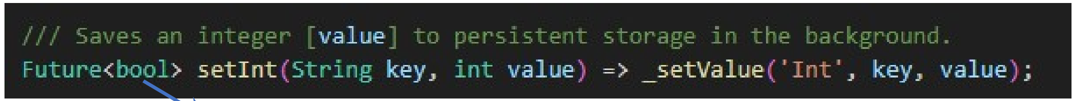
**2. Storing Settings**



```dart
import 'package:shared_preferences/shared_preferences.dart';

Future<void> saveSettings() async {
  final prefs = await SharedPreferences.getInstance();

  await prefs.setBool('darkMode', true);
  await prefs.setString('username', 'JohnDoe');
  await prefs.setInt('fontSize', 16);
}
```



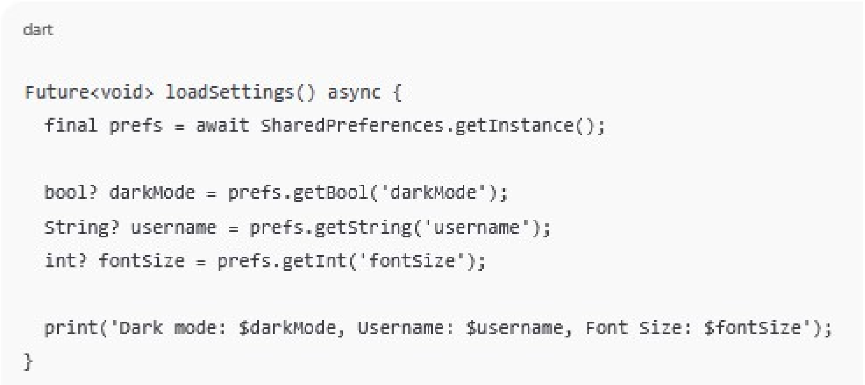
```dart
/// Loads and parses the [SharedPreferences] for this app from disk.
///
/// Because this is reading from disk, it shouldn't be awaited in
/// performance-sensitive blocks.
static Future<SharedPreferences> getInstance() async {
```



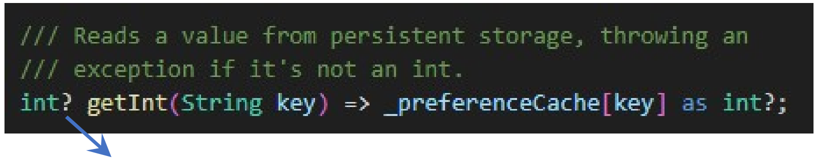
```dart
/// Saves an integer [value] to persistent storage in the background.
Future<bool> setInt(String key, int value) => _setValue('Int', key, value);
```

true if the operation succeeded, otherwise false.

**3. Retrieving Settings**



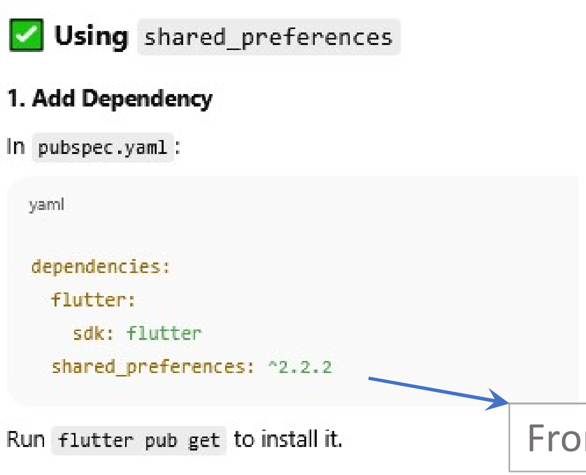
```dart
Future<void> loadSettings() async {
  final prefs = await SharedPreferences.getInstance();

  bool? darkMode = prefs.getBool('darkMode');
  String? username = prefs.getString('username');
  int? fontSize = prefs.getInt('fontSize');

  print('Dark mode: $darkMode, Username: $username, Font Size: $fontSize');
}
```



```dart
/// Reads a value from persistent storage, throwing an
/// exception if it's not an int.
int? getInt(String key) => _preferenceCache[key] as int?;
```

null, if no value had yet been stored under this key.

# Before we can use shared_preferences in our code



### ✅ Using `shared_preferences`

**1. Add Dependency**

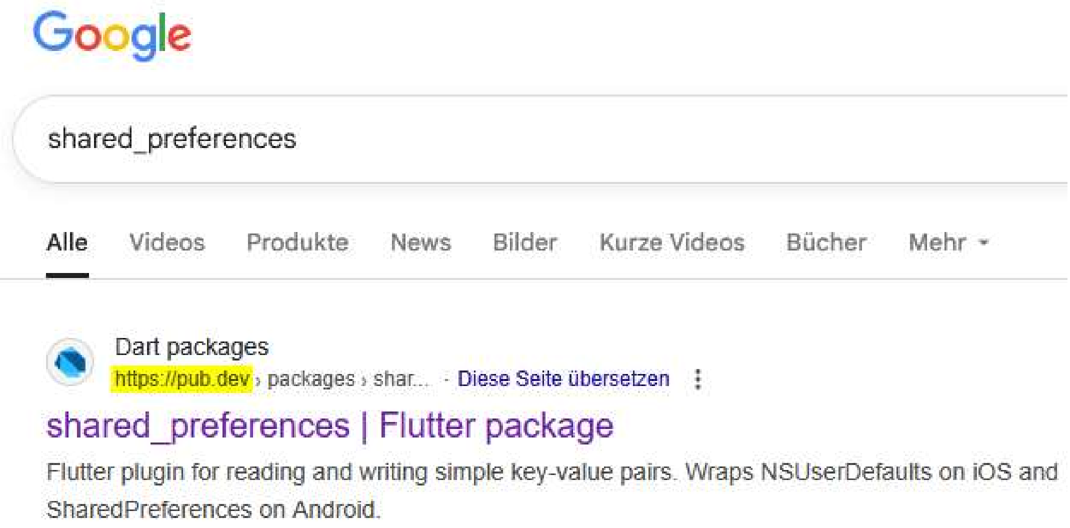
In `pubspec.yaml`:

```yaml
dependencies:
  flutter:
    sdk: flutter
  shared_preferences: ^2.2.2
```

Run `flutter pub get` to install it.

From where to know the version number to put here ?

# All flutter packages are stored in pub.dev

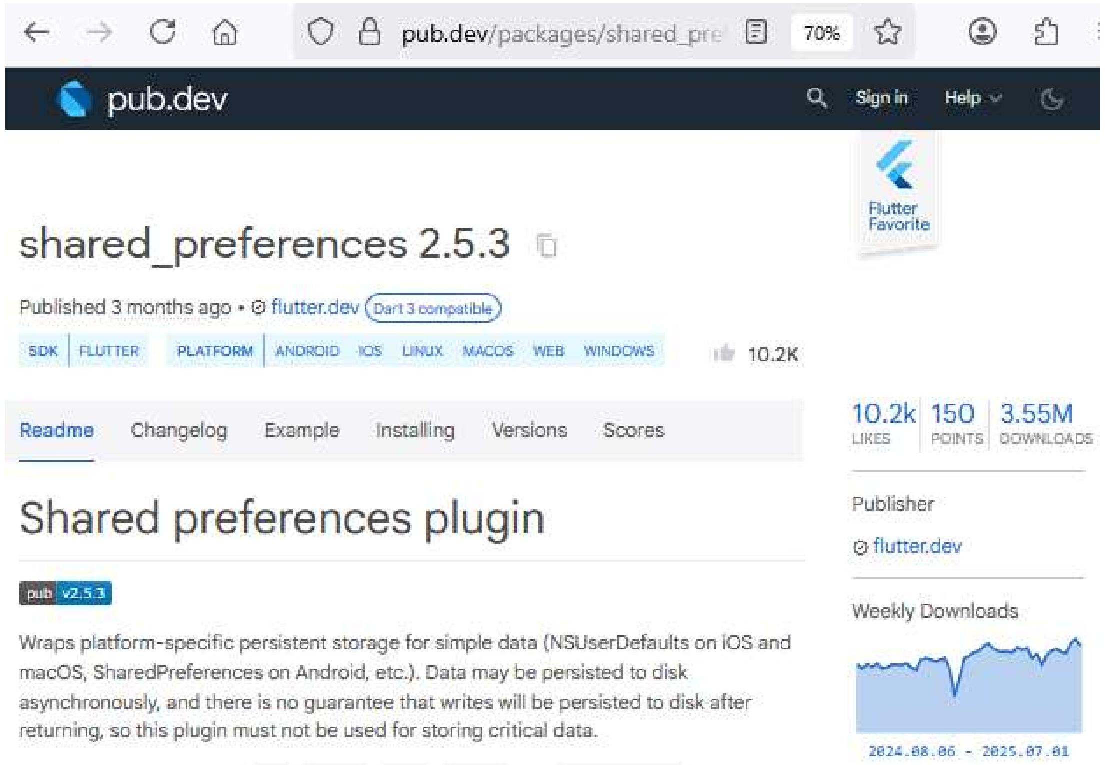


shared_preferences

Dart packages
https://pub.dev › packages › shar... · Diese Seite übersetzen

shared_preferences | Flutter package

Flutter plugin for reading and writing simple key-value pairs. Wraps NSUserDefaults on iOS and SharedPreferences on Android.



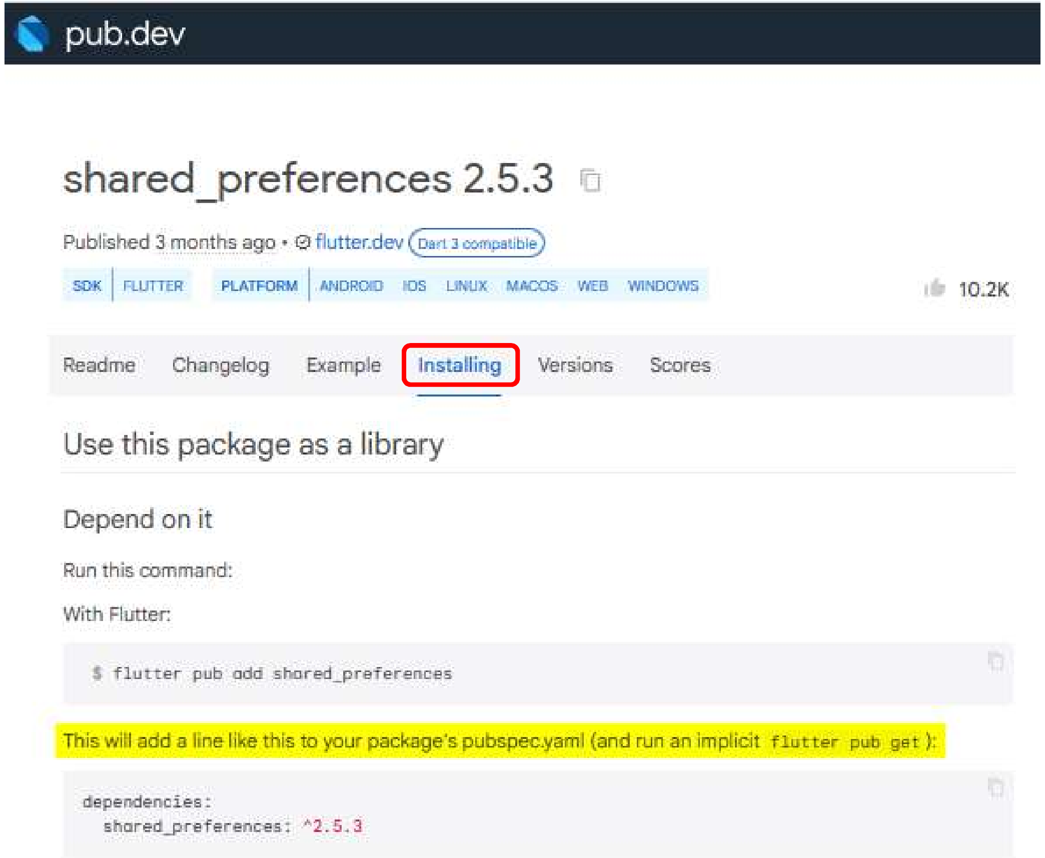
shared_preferences 2.5.3

Published 3 months ago • flutter.dev Dart 3 compatible

Shared preferences plugin

Wraps platform-specific persistent storage for simple data (NSUserDefaults on iOS and macOS, SharedPreferences on Android, etc.). Data may be persisted to disk asynchronously, and there is no guarantee that writes will be persisted to disk after returning, so this plugin must not be used for storing critical data.

# Every package describes its installation



shared_preferences 2.5.3

Published 3 months ago • flutter.dev Dart 3 compatible

Use this package as a library

Depend on it

Run this command:

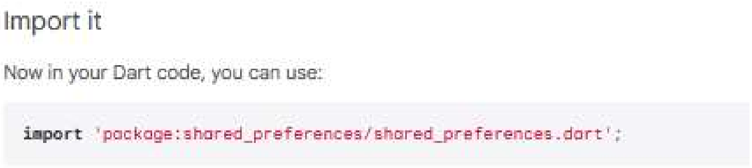
With Flutter:

```
$ flutter pub add shared_preferences
```

This will add a line like this to your package's pubspec.yaml (and run an implicit `flutter pub get`):

```
dependencies:
  shared_preferences: ^2.5.3
```

For copy / paste:
flutter pub add shared_preferences

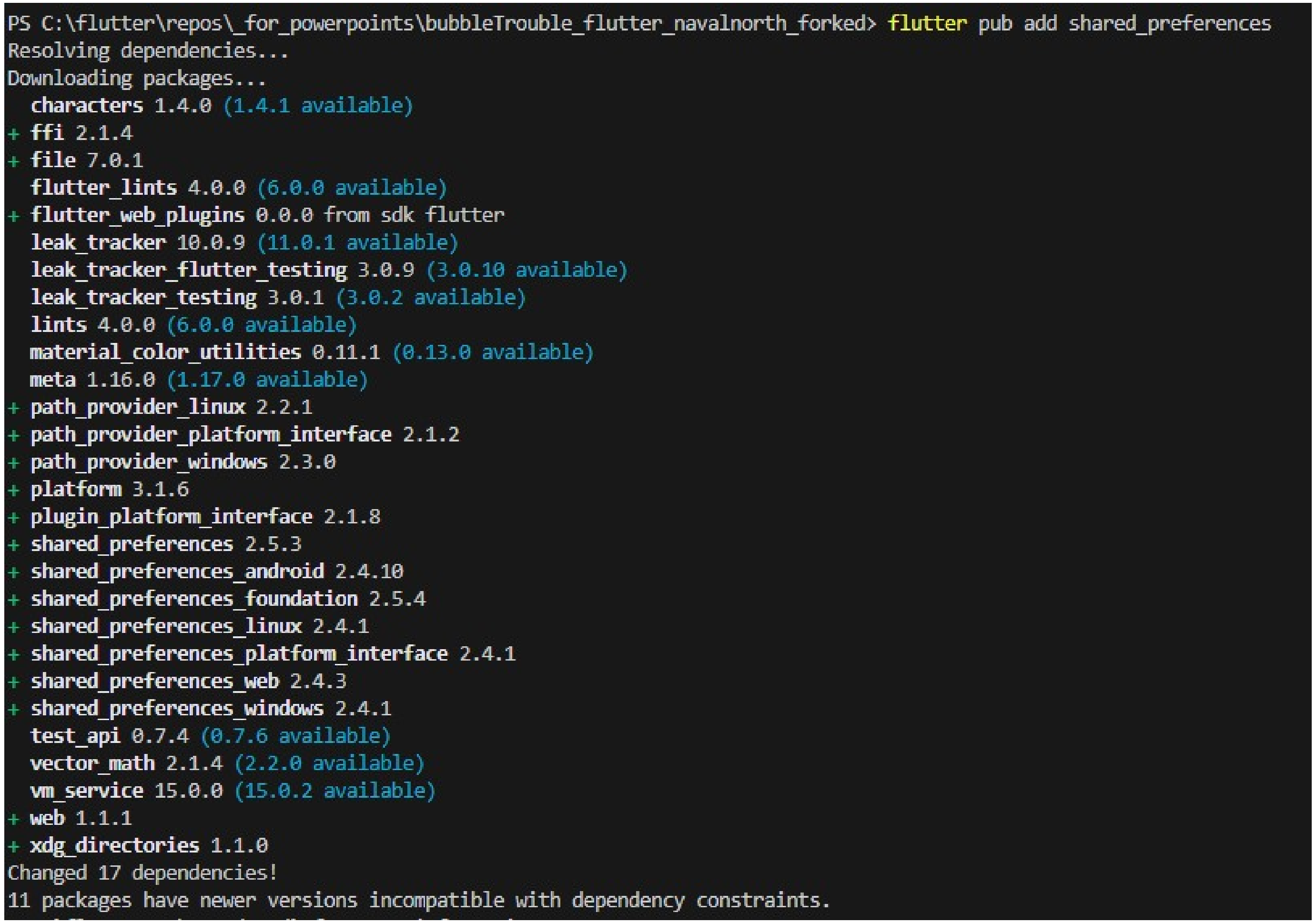


Import it

Now in your Dart code, you can use:

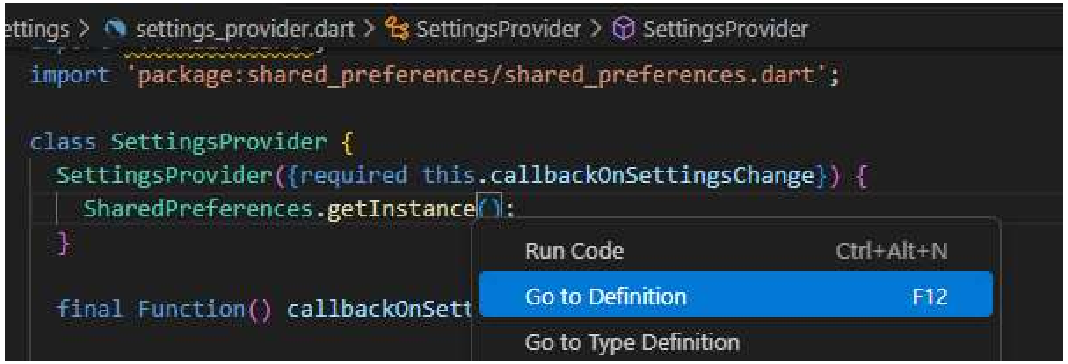
```
import 'package:shared_preferences/shared_preferences.dart';
```

# Install package shared_preferences

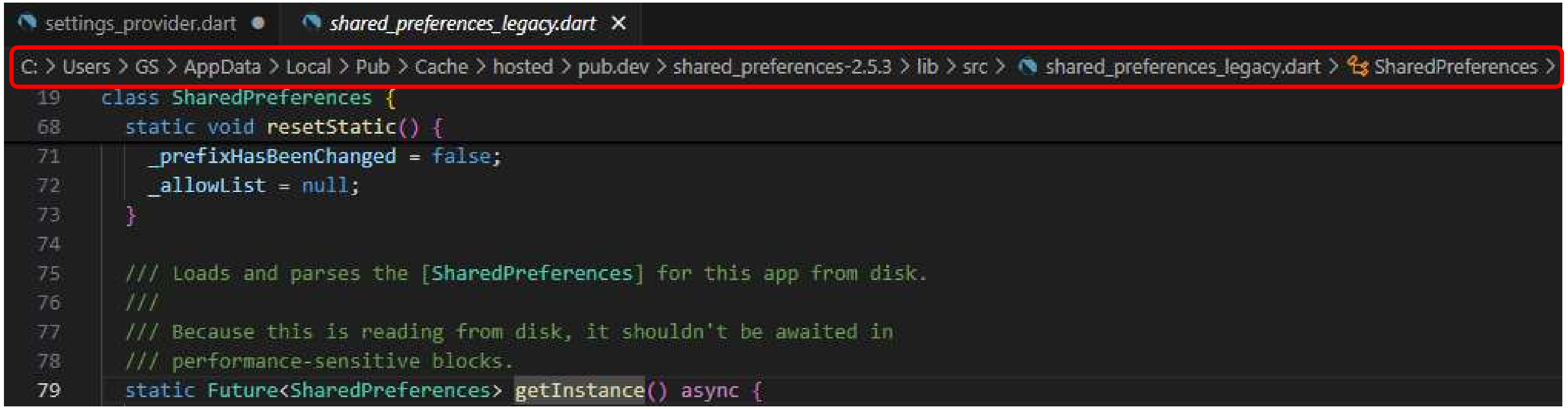


```
PS C:\flutter\repos\_for_powerpoints\bubbleTrouble_flutter_navalnorth_forked> flutter pub add shared_preferences
Resolving dependencies...
Downloading packages...
  characters 1.4.0 (1.4.1 available)
+ ffi 2.1.4
+ file 7.0.1
  flutter_lints 4.0.0 (6.0.0 available)
+ flutter_web_plugins 0.0.0 from sdk flutter
  leak_tracker 10.0.9 (11.0.1 available)
  leak_tracker_flutter_testing 3.0.9 (3.0.10 available)
  leak_tracker_testing 3.0.1 (3.0.2 available)
  lints 4.0.0 (6.0.0 available)
  material_color_utilities 0.11.1 (0.13.0 available)
  meta 1.16.0 (1.17.0 available)
+ path_provider_linux 2.2.1
+ path_provider_platform_interface 2.1.2
+ path_provider_windows 2.3.0
+ platform 3.1.6
+ plugin_platform_interface 2.1.8
+ shared_preferences 2.5.3
+ shared_preferences_android 2.4.10
+ shared_preferences_foundation 2.5.4
+ shared_preferences_linux 2.4.1
+ shared_preferences_platform_interface 2.4.1
+ shared_preferences_web 2.4.3
+ shared_preferences_windows 2.4.1
  test_api 0.7.4 (0.7.6 available)
  vector_math 2.1.4 (2.2.0 available)
  vm_service 15.0.0 (15.0.2 available)
+ web 1.1.1
+ xdg_directories 1.1.0
Changed 17 dependencies!
11 packages have newer versions incompatible with dependency constraints.
```

# Packages are downloaded where ?


ettings > settings_provider.dart > SettingsProvider > SettingsProvider
Run Code Ctrl+Alt+N
Go to Definition F12
Go to Type Definition

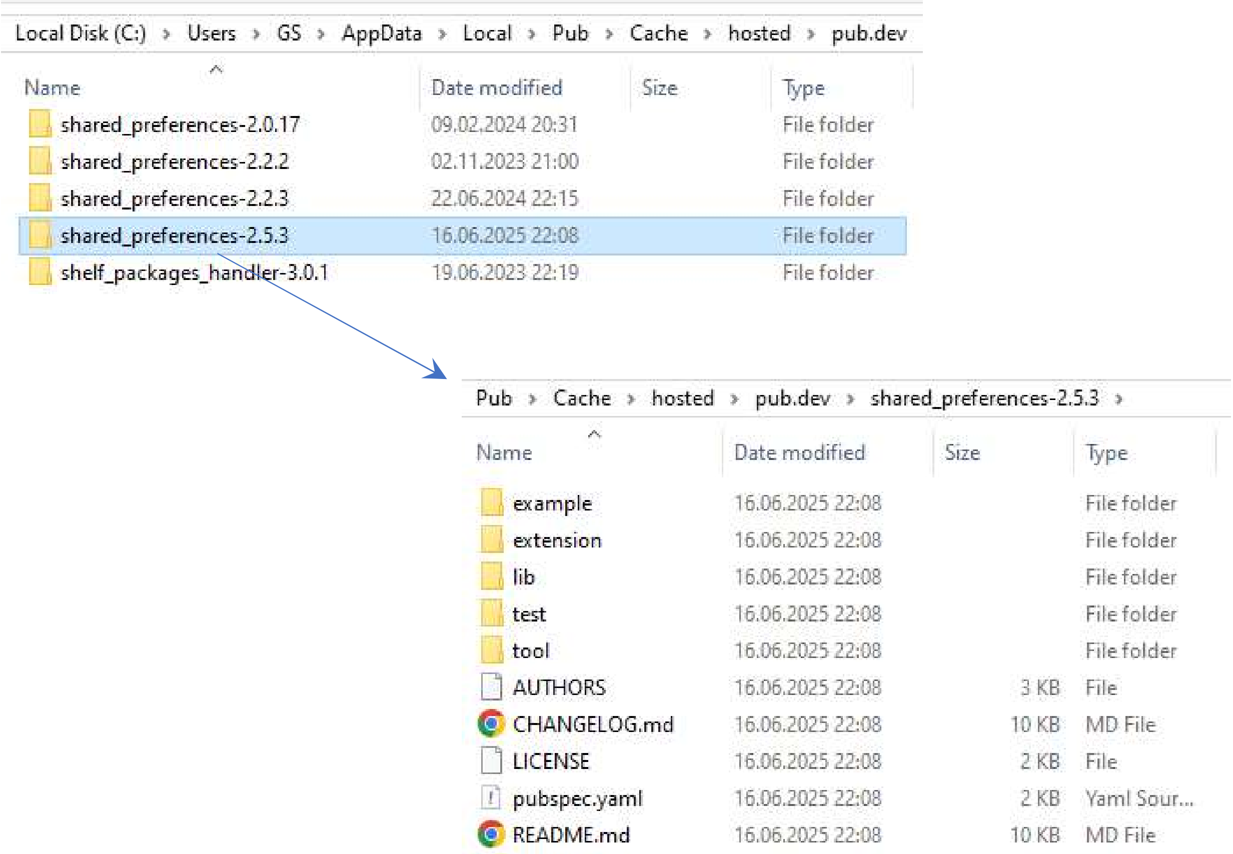
```dart
import 'package:shared_preferences/shared_preferences.dart';

class SettingsProvider {
  SettingsProvider({required this.callbackOnSettingsChange}) {
    SharedPreferences.getInstance();
  }

  final Function() callbackOnSett
```



C: > Users > GS > AppData > Local > Pub > Cache > hosted > pub.dev > shared_preferences-2.5.3 > lib > src > shared_preferences_legacy.dart > SharedPreferences

```dart
class SharedPreferences {
  static void resetStatic() {
    _prefixHasBeenChanged = false;
    _allowList = null;
  }

  /// Loads and parses the [SharedPreferences] for this app from disk.
  ///
  /// Because this is reading from disk, it shouldn't be awaited in
  /// performance-sensitive blocks.
  static Future<SharedPreferences> getInstance() async {
```

# Packages are downloaded where ?



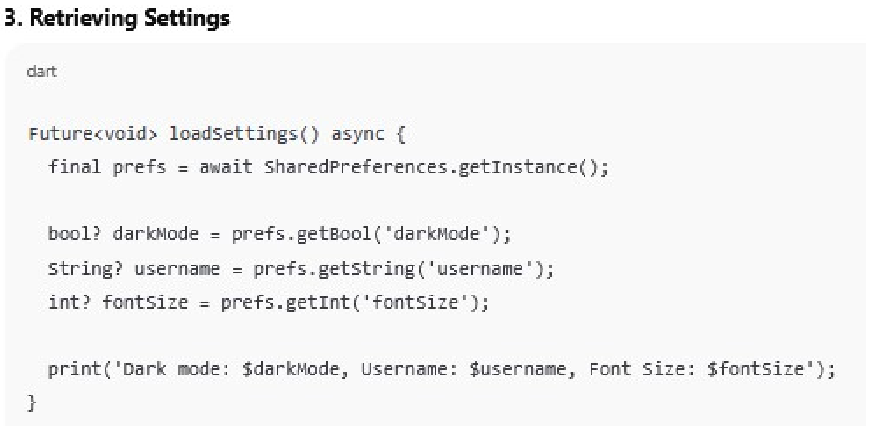
Local Disk (C:) › Users › GS › AppData › Local › Pub › Cache › hosted › pub.dev

| Name | Date modified | Size | Type |
|---|---|---|---|
| shared_preferences-2.0.17 | 09.02.2024 20:31 | | File folder |
| shared_preferences-2.2.2 | 02.11.2023 21:00 | | File folder |
| shared_preferences-2.2.3 | 22.06.2024 22:15 | | File folder |
| shared_preferences-2.5.3 | 16.06.2025 22:08 | | File folder |
| shelf_packages_handler-3.0.1 | 19.06.2023 22:19 | | File folder |

Pub › Cache › hosted › pub.dev › shared_preferences-2.5.3 ›

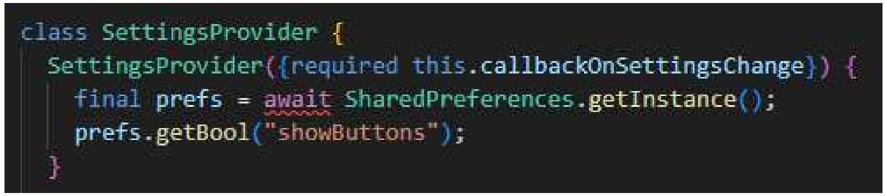
| Name | Date modified | Size | Type |
|---|---|---|---|
| example | 16.06.2025 22:08 | | File folder |
| extension | 16.06.2025 22:08 | | File folder |
| lib | 16.06.2025 22:08 | | File folder |
| test | 16.06.2025 22:08 | | File folder |
| tool | 16.06.2025 22:08 | | File folder |
| AUTHORS | 16.06.2025 22:08 | 3 KB | File |
| CHANGELOG.md | 16.06.2025 22:08 | 10 KB | MD File |
| LICENSE | 16.06.2025 22:08 | 2 KB | File |
| pubspec.yaml | 16.06.2025 22:08 | 2 KB | Yaml Sour... |
| README.md | 16.06.2025 22:08 | 10 KB | MD File |

# Where to call the await for getInstance() ?

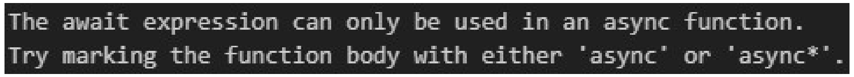


**3. Retrieving Settings**

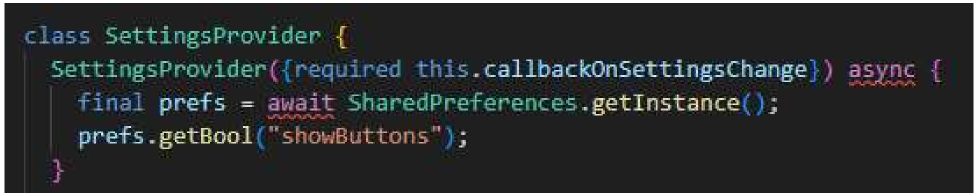
```dart
Future<void> loadSettings() async {
  final prefs = await SharedPreferences.getInstance();

  bool? darkMode = prefs.getBool('darkMode');
  String? username = prefs.getString('username');
  int? fontSize = prefs.getInt('fontSize');

  print('Dark mode: $darkMode, Username: $username, Font Size: $fontSize');
}
```



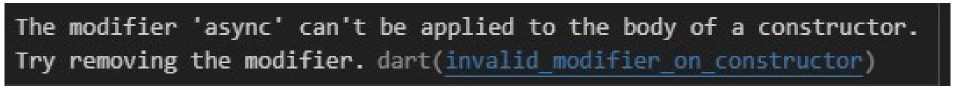
```
class SettingsProvider {
  SettingsProvider({required this.callbackOnSettingsChange}) {
    final prefs = await SharedPreferences.getInstance();
    prefs.getBool("showButtons");
  }
```



```
The await expression can only be used in an async function.
Try marking the function body with either 'async' or 'async*'.
```

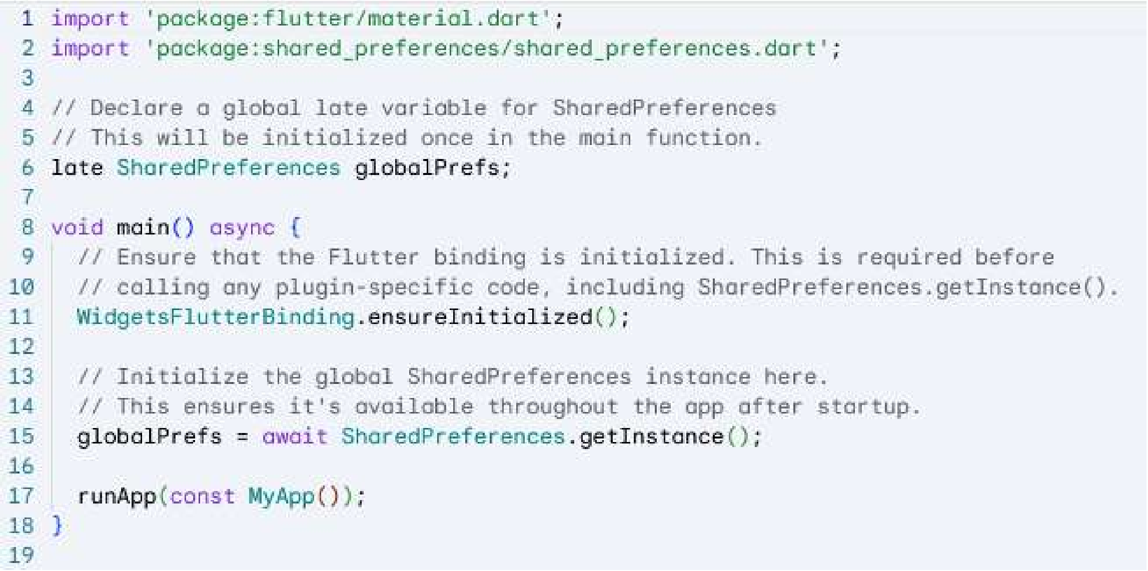


```
class SettingsProvider {
  SettingsProvider({required this.callbackOnSettingsChange}) async {
    final prefs = await SharedPreferences.getInstance();
    prefs.getBool("showButtons");
  }
```



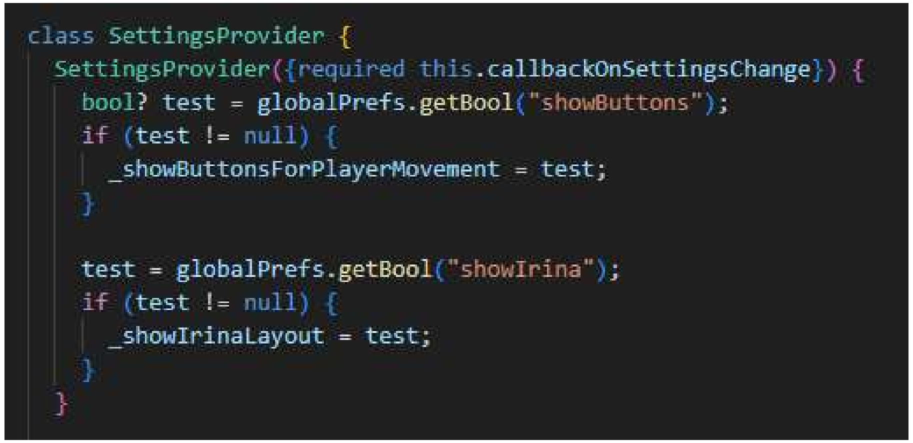
```
The modifier 'async' can't be applied to the body of a constructor.
Try removing the modifier. dart(invalid_modifier_on_constructor)
```

# Code found in Gemini



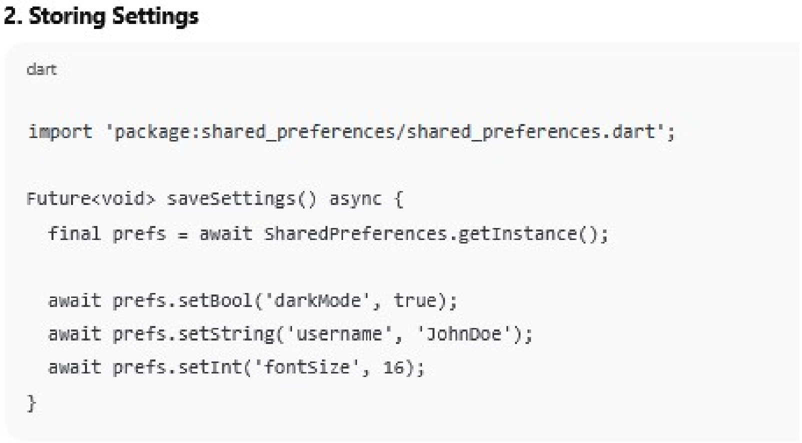
```dart
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';

// Declare a global late variable for SharedPreferences
// This will be initialized once in the main function.
late SharedPreferences globalPrefs;

void main() async {
  // Ensure that the Flutter binding is initialized. This is required before
  // calling any plugin-specific code, including SharedPreferences.getInstance().
  WidgetsFlutterBinding.ensureInitialized();

  // Initialize the global SharedPreferences instance here.
  // This ensures it's available throughout the app after startup.
  globalPrefs = await SharedPreferences.getInstance();

  runApp(const MyApp());
}

```

code.txt



```dart
class SettingsProvider {
  SettingsProvider({required this.callbackOnSettingsChange}) {
    bool? test = globalPrefs.getBool("showButtons");
    if (test != null) {
      _showButtonsForPlayerMovement = test;
    }

    test = globalPrefs.getBool("showIrina");
    if (test != null) {
      _showIrinaLayout = test;
    }
  }
```

# Storing the values in the setters

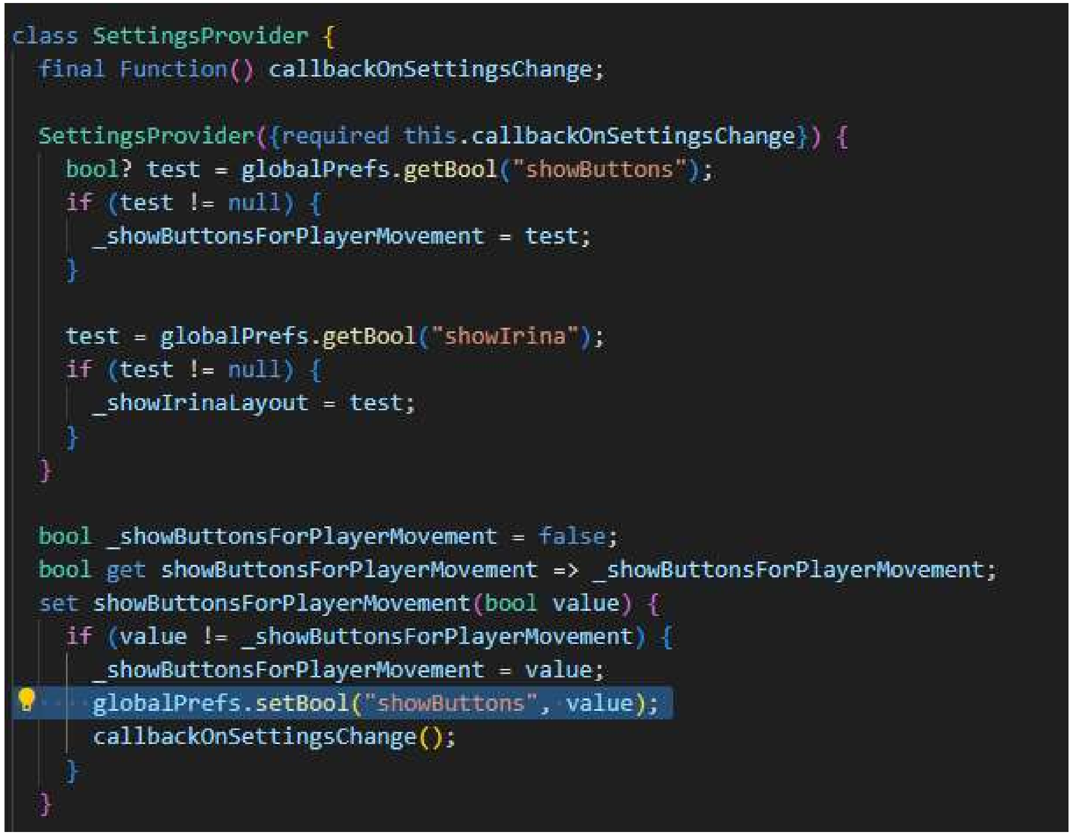


**2. Storing Settings**

```dart
import 'package:shared_preferences/shared_preferences.dart';

Future<void> saveSettings() async {
  final prefs = await SharedPreferences.getInstance();

  await prefs.setBool('darkMode', true);
  await prefs.setString('username', 'JohnDoe');
  await prefs.setInt('fontSize', 16);
}
```

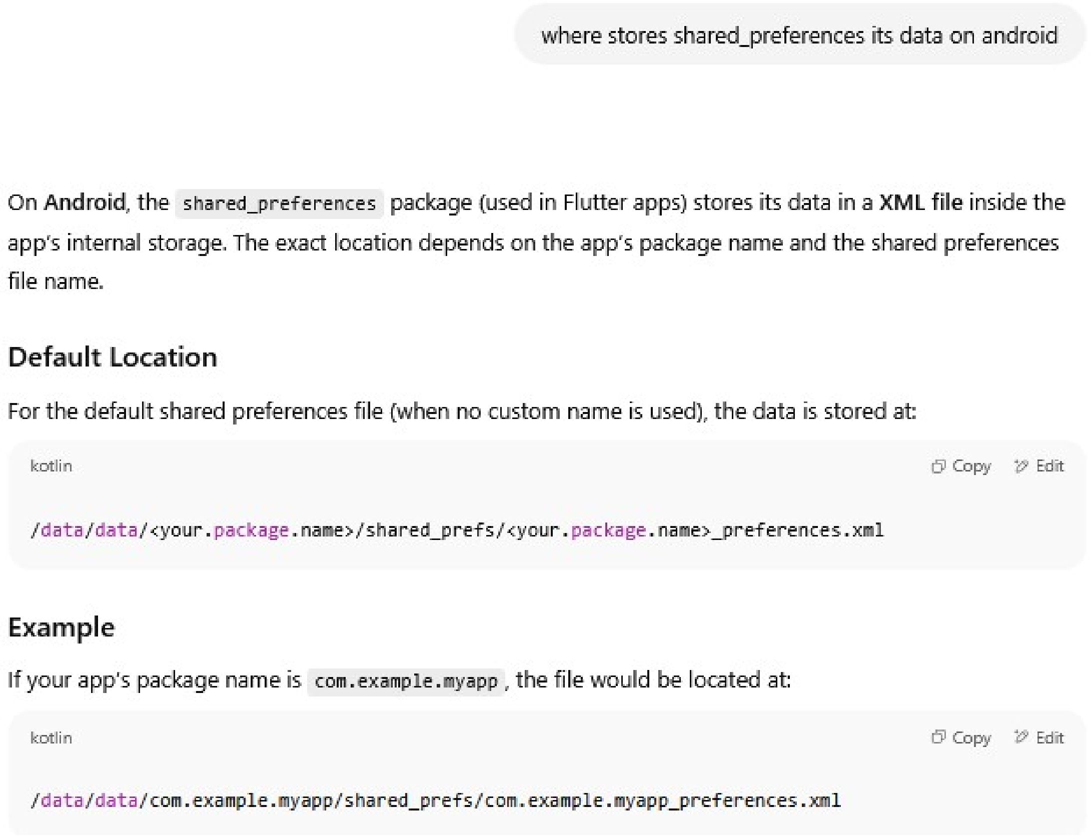
Because a setter cannot be async (same as for c-tor), we do not wait for the result of setBool (true if succeeded, false if not).

That means we “fire & forget”, which is ok, because our settings are not critical.



```dart
class SettingsProvider {
  final Function() callbackOnSettingsChange;

  SettingsProvider({required this.callbackOnSettingsChange}) {
    bool? test = globalPrefs.getBool("showButtons");
    if (test != null) {
      _showButtonsForPlayerMovement = test;
    }

    test = globalPrefs.getBool("showIrina");
    if (test != null) {
      _showIrinaLayout = test;
    }
  }

  bool _showButtonsForPlayerMovement = false;
  bool get showButtonsForPlayerMovement => _showButtonsForPlayerMovement;
  set showButtonsForPlayerMovement(bool value) {
    if (value != _showButtonsForPlayerMovement) {
      _showButtonsForPlayerMovement = value;
      globalPrefs.setBool("showButtons", value);
      callbackOnSettingsChange();
    }
  }
```

# Storage in Android



where stores shared_preferences its data on android

On **Android**, the `shared_preferences` package (used in Flutter apps) stores its data in a **XML file** inside the app's internal storage. The exact location depends on the app's package name and the shared preferences file name.

### Default Location

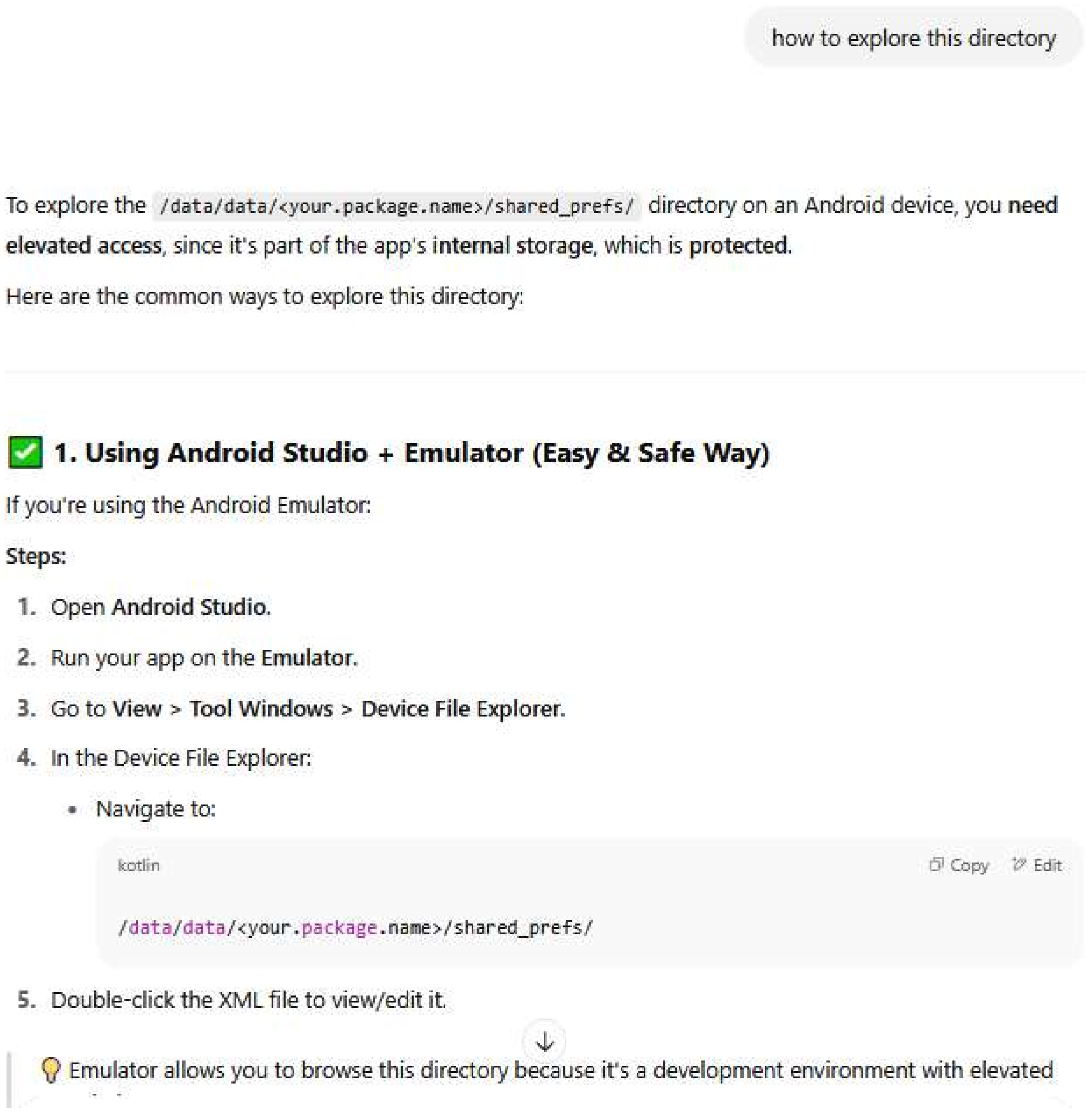
For the default shared preferences file (when no custom name is used), the data is stored at:

```kotlin
/data/data/<your.package.name>/shared_prefs/<your.package.name>_preferences.xml
```

### Example

If your app's package name is `com.example.myapp`, the file would be located at:

```kotlin
/data/data/com.example.myapp/shared_prefs/com.example.myapp_preferences.xml
```



how to explore this directory

To explore the `/data/data/<your.package.name>/shared_prefs/` directory on an Android device, you **need elevated access**, since it's part of the app's **internal storage**, which is **protected**.

Here are the common ways to explore this directory:

---

### ✅ 1. Using Android Studio + Emulator (Easy & Safe Way)

If you're using the Android Emulator:

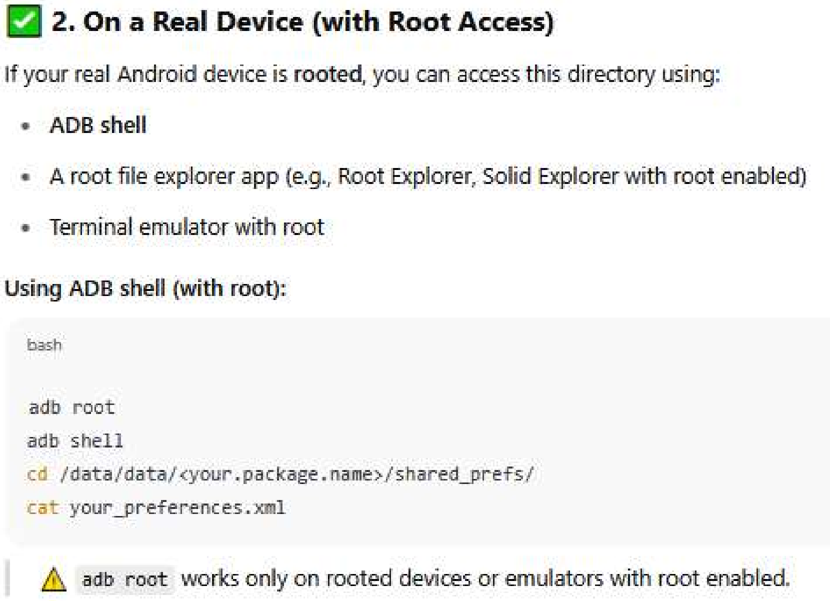
**Steps:**

1. Open **Android Studio**.
2. Run your app on the **Emulator**.
3. Go to **View** > **Tool Windows** > **Device File Explorer**.
4. In the Device File Explorer:
   - Navigate to:

```kotlin
/data/data/<your.package.name>/shared_prefs/
```

5. Double-click the XML file to view/edit it.

> 💡 Emulator allows you to browse this directory because it's a development environment with elevated



### ✅ 2. On a Real Device (with Root Access)

If your real Android device is **rooted**, you can access this directory using:

- **ADB shell**
- A root file explorer app (e.g., Root Explorer, Solid Explorer with root enabled)
- Terminal emulator with root

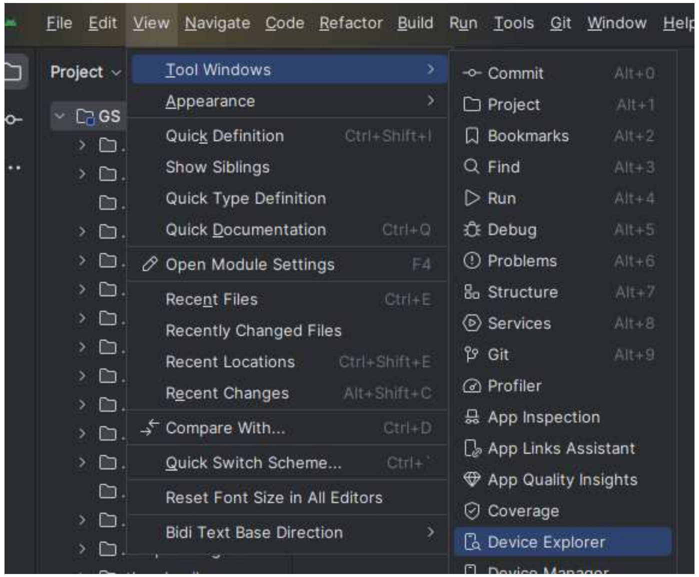
**Using ADB shell (with root):**

```bash
adb root
adb shell
cd /data/data/<your.package.name>/shared_prefs/
cat your_preferences.xml
```

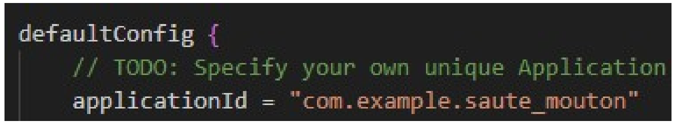
> ⚠️ `adb root` works only on rooted devices or emulators with root enabled.

# Device Explorer in Android Studio

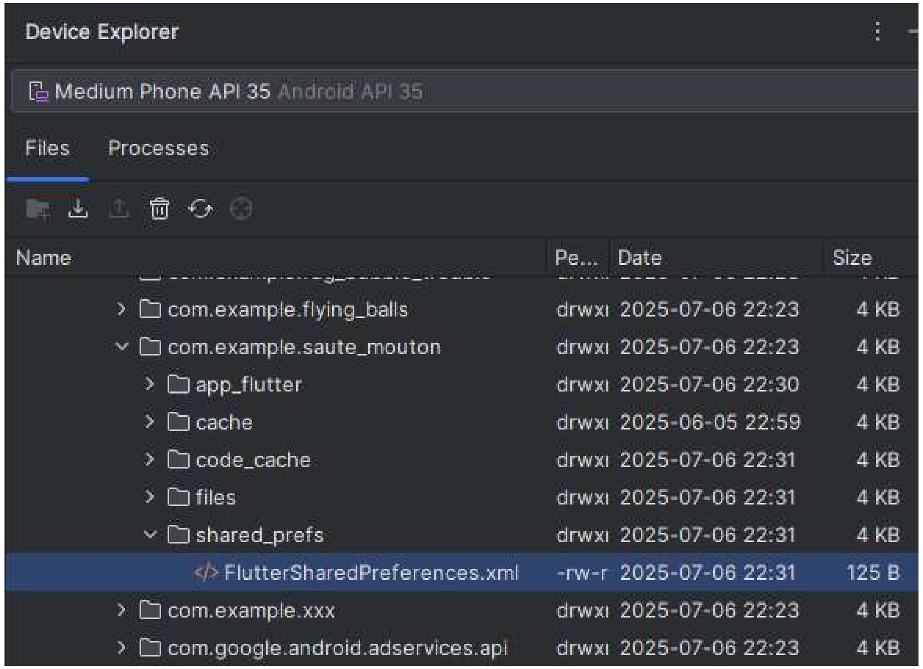
Open a project in Android Studio, then



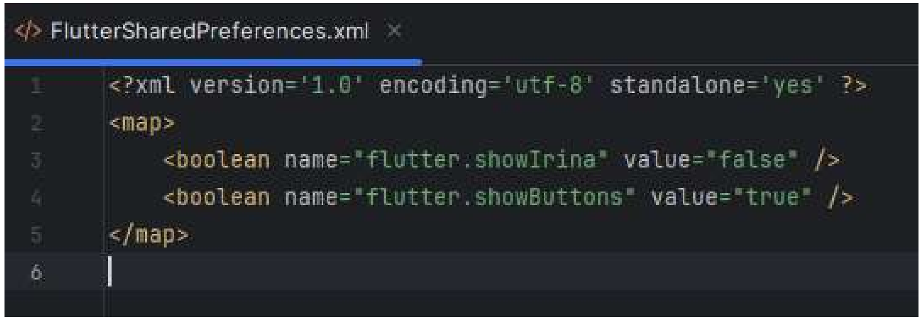
BTW: the application id is defined in file “**android\app\build.gradle**”:



```
defaultConfig {
    // TODO: Specify your own unique Application
    applicationId = "com.example.saute_mouton"
```

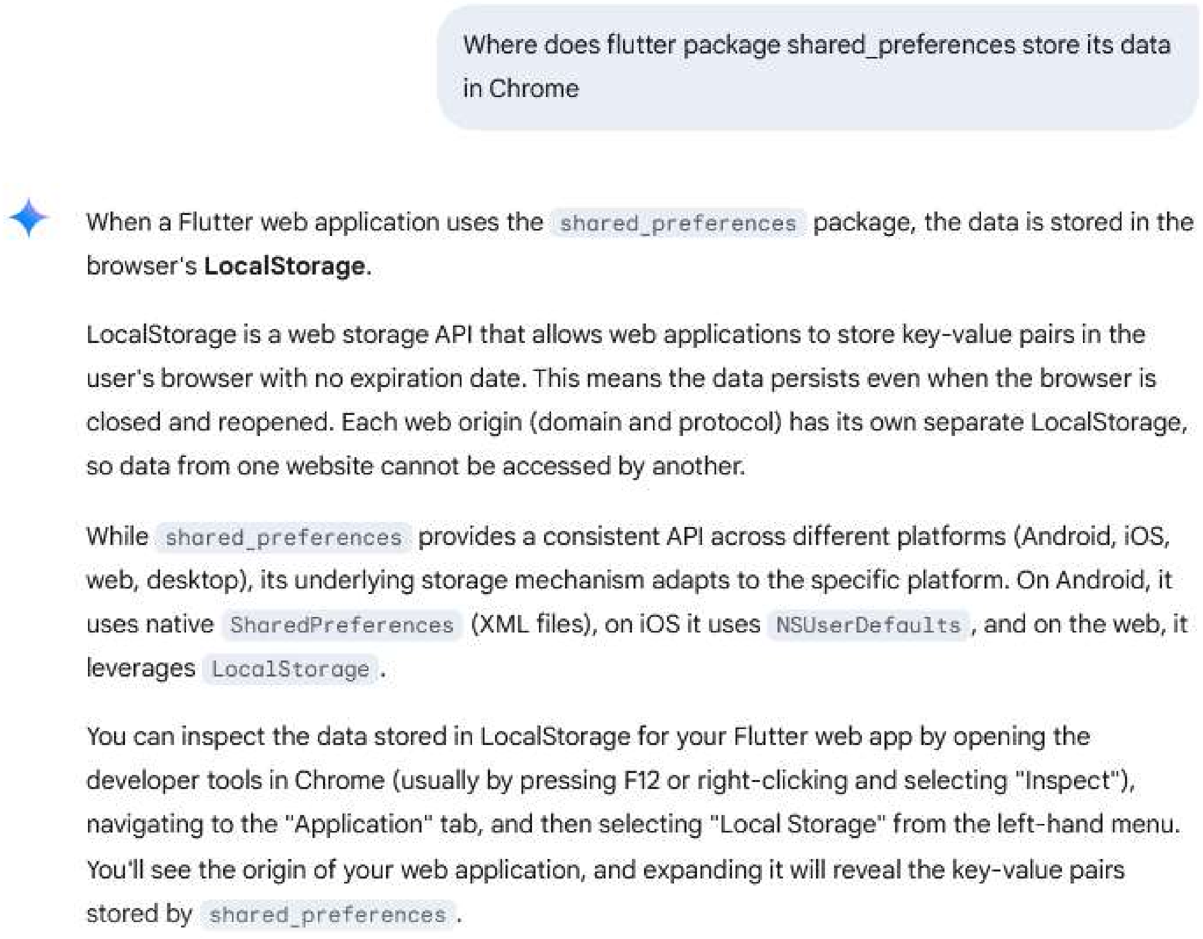


| Name | Pe... | Date | Size |
|---|---|---|---|
| com.example.flying_balls | drwx | 2025-07-06 22:23 | 4 KB |
| com.example.saute_mouton | drwx | 2025-07-06 22:23 | 4 KB |
| app_flutter | drwx | 2025-07-06 22:30 | 4 KB |
| cache | drwx | 2025-06-05 22:59 | 4 KB |
| code_cache | drwx | 2025-07-06 22:31 | 4 KB |
| files | drwx | 2025-07-06 22:31 | 4 KB |
| shared_prefs | drwx | 2025-07-06 22:31 | 4 KB |
| FlutterSharedPreferences.xml | -rw-r | 2025-07-06 22:31 | 125 B |
| com.example.xxx | drwx | 2025-07-06 22:23 | 4 KB |
| com.google.android.adservices.api | drwx | 2025-07-06 22:23 | 4 KB |



```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="flutter.showIrina" value="false" />
    <boolean name="flutter.showButtons" value="true" />
</map>
```

# Where does shared_preferences store its data in Chrome ?



Where does flutter package shared_preferences store its data in Chrome

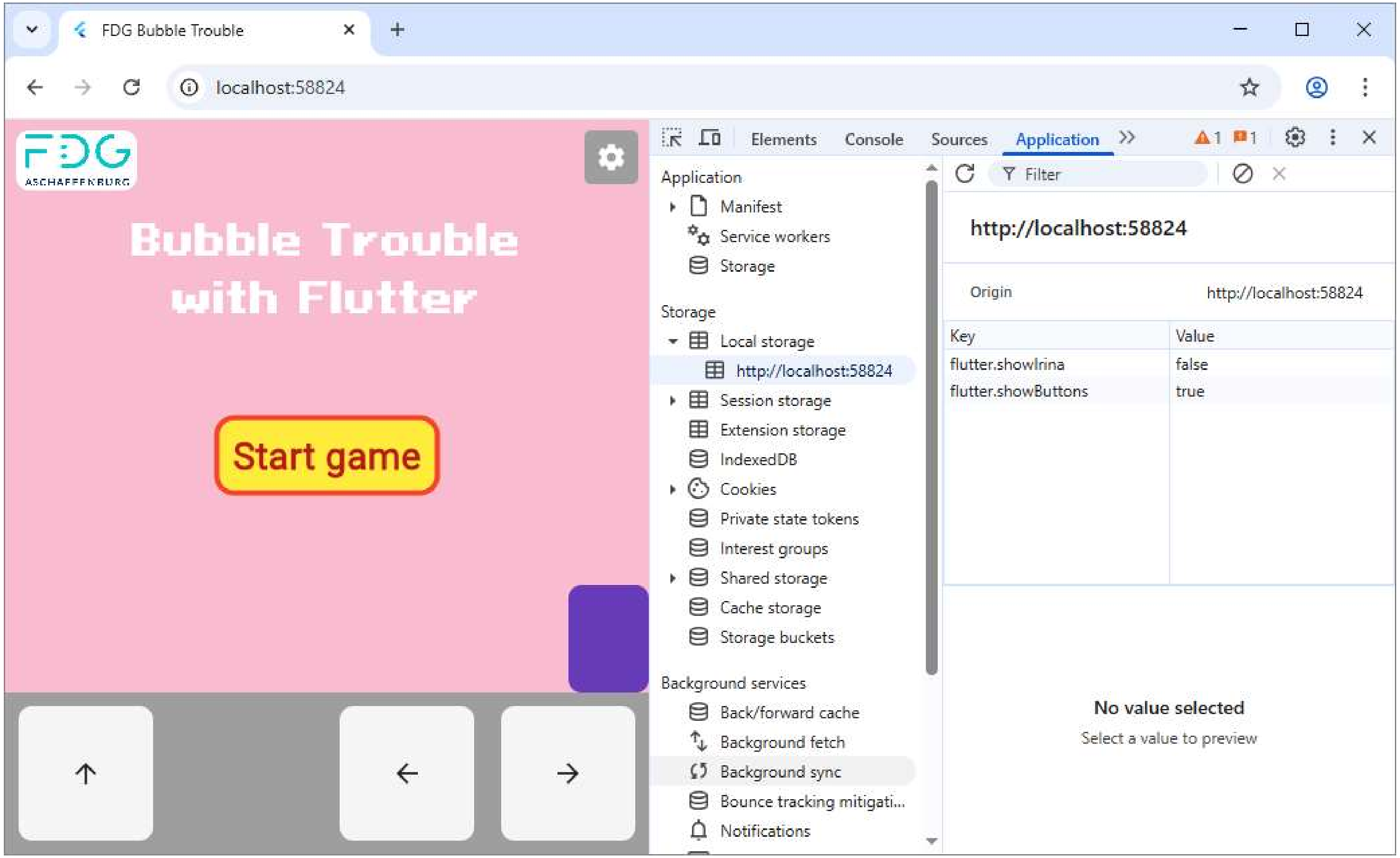
When a Flutter web application uses the `shared_preferences` package, the data is stored in the browser's **LocalStorage**.

LocalStorage is a web storage API that allows web applications to store key-value pairs in the user's browser with no expiration date. This means the data persists even when the browser is closed and reopened. Each web origin (domain and protocol) has its own separate LocalStorage, so data from one website cannot be accessed by another.

While `shared_preferences` provides a consistent API across different platforms (Android, iOS, web, desktop), its underlying storage mechanism adapts to the specific platform. On Android, it uses native `SharedPreferences` (XML files), on iOS it uses `NSUserDefaults`, and on the web, it leverages `LocalStorage`.

You can inspect the data stored in LocalStorage for your Flutter web app by opening the developer tools in Chrome (usually by pressing F12 or right-clicking and selecting "Inspect"), navigating to the "Application" tab, and then selecting "Local Storage" from the left-hand menu. You'll see the origin of your web application, and expanding it will reveal the key-value pairs stored by `shared_preferences`.

# Where does shared_preferences store its data in Chrome ?



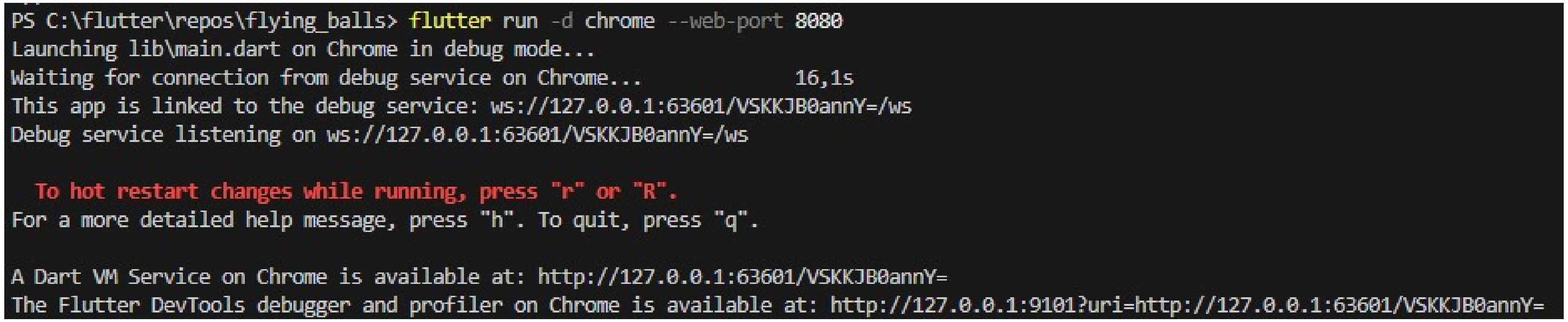
http://localhost:58824

Origin http://localhost:58824

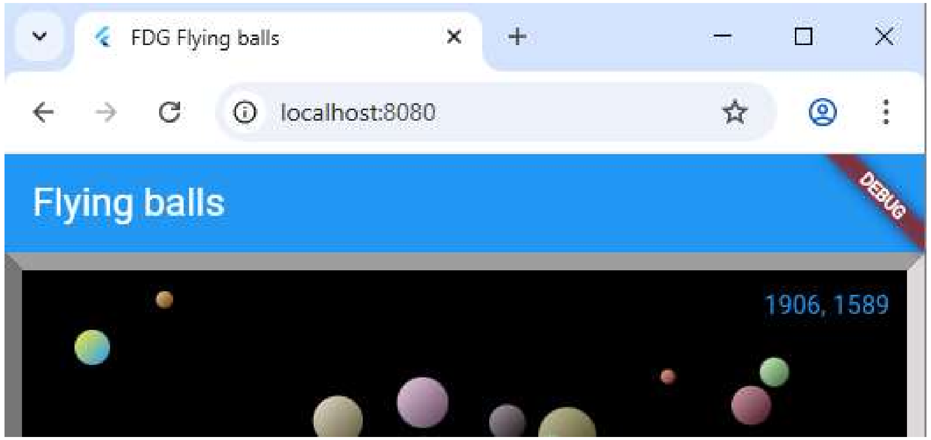
| Key | Value |
|---|---|
| flutter.showIrina | false |
| flutter.showButtons | true |

# Slide 7 in Powerpoint „97 Often used Flutter commands”: Debug web app on a certain port

flutter run -d chrome --web-port *port-number*



```
PS C:\flutter\repos\flying_balls> flutter run -d chrome --web-port 8080
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...             16,1s
This app is linked to the debug service: ws://127.0.0.1:63601/VSKKJB0annY=/ws
Debug service listening on ws://127.0.0.1:63601/VSKKJB0annY=/ws

  To hot restart changes while running, press "r" or "R".
For a more detailed help message, press "h". To quit, press "q".

A Dart VM Service on Chrome is available at: http://127.0.0.1:63601/VSKKJB0annY=
The Flutter DevTools debugger and profiler on Chrome is available at: http://127.0.0.1:9101?uri=http://127.0.0.1:63601/VSKKJB0annY=
```


FDG Flying balls
localhost:8080
Flying balls
DEBUG
1906, 1589