

Biomedical Wearable Technologies
for Healthcare and Wellbeing

Shared Preferences

A.Y. 2023-2024

Giacomo Cappon



Outline

- **Persist key-value data**
- shared_preferences
- Counter app with persistence
- Exercise
- Homework
- Resources

Persist key-value data

- Today we will start to understand how to implement persistence: as such, it will be possible to “save” things on the disk and “resume” the state of the application when it is restarted
- The simplest type of persistence (and commonly used by many applications)? Storing primitive (`String`, `bool`, `int`, `float`, `List<String>`) key-value data. Some examples:
 - `"id" : 3`
 - `"user" : "mario"`
 - `"isAdmin" : false`
 - ...

key : value

Persist key-value data

➤ Some use cases of key-value data persistence:

- Remember the preferences of a certain user
 - App theme
 - App “general” parameters
- Restore previous session status
 - Was the user logged in during the last session?
 - Decide which screen to visualize when the app restarts

key : value

➤ How to implement key-value data persistence

- Fortunately, in Flutter implementing this functionality is very easy thanks to the `shared_preferences` package
 - https://pub.dev/packages/shared_preferences

Outline

- Persist key-value data
- **shared_preferences**
- Counter app with persistence
- Exercise
- Homework
- Resources

shared_preferences

- `shared_preferences` encloses a singleton of class `SharedPreferences` that can be accessed via the `getInstance()` asynchronous static method:

```
final sp = await SharedPreferences.getInstance();
```

- What is a “**singleton**”?
 - Singleton is a design pattern (1 of the 23) of object-oriented programming that guarantees that for a given class, only one instance of that class can be created.
 - This means that only one instance of the `SharedPreferences` will be present in our app. So, in practice, the `getInstance()` method will return always the same object.

shared_preferences – write

➤ Here's the methods of shared_preferences to **write** data:

```
// Save an integer value to 'counter' key.  
await sp.setInt('counter', 10);  
// Save an boolean value to 'repeat' key.  
await sp.setBool('repeat', true);  
// Save an double value to 'decimal' key.  
await sp.setDouble('decimal', 1.5);  
// Save an String value to 'action' key.  
await sp.setString('action', 'Start');  
// Save an list of strings to 'items' key.  
await sp.setStringList('items', <String>['Earth', 'Moon',  
'Sun']);
```

shared_preferences – read

➤ Here's the methods of `shared_preferences` to **read** data:

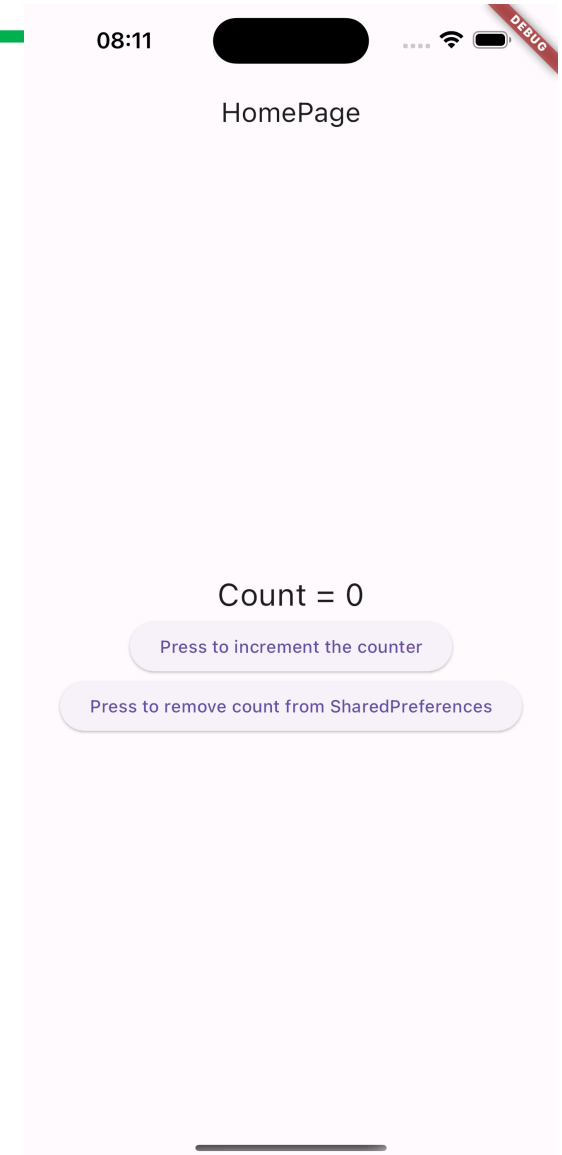
```
// Try reading data from the 'counter' key. If it doesn't exist,  
returns null.  
final int? counter = sp.getInt('counter');  
// Try reading data from the 'repeat' key. If it doesn't exist,  
returns null.  
final bool? repeat = sp.getBool('repeat');  
// Try reading data from the 'decimal' key. If it doesn't exist,  
returns null.  
final double? decimal = sp.getDouble('decimal');  
// Try reading data from the 'action' key. If it doesn't exist,  
returns null.  
final String? action = sp.getString('action');  
// Try reading data from the 'items' key. If it doesn't exist,  
returns null.  
final List<String>? items = sp.getStringList('items');
```


Outline

- Persist key-value data
- shared_preferences
- **Counter app with persistence**
- Exercise
- Homework
- Resources

Counter app with persistence

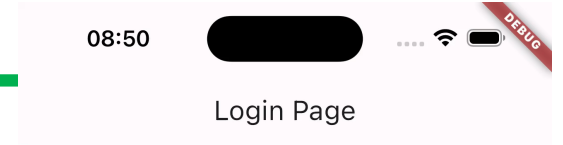
- As a case of study, today we will implement a simple app called “counter” that persists an integer to offer the following functionalities:
 - If a user tap a button, an integer counter (starting from 0) is incremented by 1 and the result is displayed by a `Text` widget. The new value of the counter is then persisted using `shared_preferences`
 - If the user restarts the app, the app “resumes” the value of the counter by loading it from the disk
 - If another button is tapped, the integer is removed from `SharedPreferences`. So, if the app is restarted, the counter will start back from 0.



Outline

- Recap
- Persist key-value data
- `shared_preferences`
- Counter app with persistence
- **Exercise**
- Homework
- Resources

Exercise



- Exercise 08.01 (easy-medium)
- Start from the code of Exercise 06.02 (the login_flow app)
 - Right now, every time you restart the app, the LoginPage is showed to user
 - Use the shared_preferences package to change this behaviour and let the app “remember” the last user session. As such:
 - If a user opens the app for the first time, the LoginPage route is showed;
 - If the user logs in, the HomePage is showed;
 - If a user restarts the app and he/she did not log out in the last session, the HomePage is showed;
 - If a user restarts the app and he/she logged out in the last session, the LoginPage is showed.

A login form with two input fields: 'Email' and 'Password'. Below the fields is a rounded rectangular button labeled 'Login'. The form is styled with light gray borders and a clean, modern aesthetic.

Outline

- Persist key-value data
- shared_preferences
- Counter app with persistence
- Exercise
- **Homework**
- Resources

Homework

- Get familiar with `shared_preferences`
- Take look to a safer alternative: `flutter_secure_storage`
 - https://pub.dev/packages/flutter_secure_storage

Outline

- Recap
- Persist key-value data
- `shared_preferences`
- Counter app with persistence
- Exercise
- Homework
- **Resources**

Resources

- shared_preferences package
 - https://pub.dev/packages/shared_preferences
- Flutter community cookbook on SharedPreferences
 - <https://docs.flutter.dev/cookbook/persistence/key-value>