Shared Preferences in Flutter and CRUD Operations (Tutorial)

Introduction

Shared Preferences is a lightweight key-value store in Flutter that allows you to persistently store and retrieve small amounts of data. It is commonly used to store application settings, user preferences, and other similar data. In this document, we will explore how to use Shared Preferences in Flutter and perform **CRUD** (Create, Read, Update, Delete) operations.

Installation

To use Shared Preferences in your Flutter project, you need to add the shared_preferences package to your pubspec.yaml file:

```
dependencies:
  flutter:
    sdk: flutter
  shared_preferences: ^2.0.8
```

After adding the package, run flutter **pub get** to fetch and install it.

Getting Started

To start using Shared Preferences, you need to import the package in your Dart file:

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

CRUD Operations

Create

To create or save a value in Shared Preferences, you need to use an instance of SharedPreferences and call the set method with the desired key-value pair. Here's an example:

```
void saveData(String key, String value) async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setString(key, value);
}
```

In the above example, we create a method saveData that takes a key and a value as parameters. We obtain an instance of SharedPreferences using the getInstance method and save the value using the setString method.

Read

To read a value from Shared Preferences, you can use the getString method, providing the key corresponding to the value you want to retrieve. Here's an example:

```
Future<String?> loadData(String key) async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  return prefs.getString(key);
}
```

In the above example, we create a method loadData that takes a key as a parameter. We obtain an instance of SharedPreferences using the getInstance method and retrieve the value using the getString method. The method returns a **Future<String?>** because the value can be null if it doesn't exist.

Future in flutter

Futures represent the result of an asynchronous operation that may complete at some point in the future. They are used to handle operations that might take time to complete, such as network requests, file I/O, or any other asynchronous tasks.When you make an asynchronous call in Flutter, such as an API request, it typically returns a Future object immediately. This Future object represents the result of the operation, which may be available now or in the future. You can use this Future object to listen for the completion of the operation and handle the result accordingly.

Futures in Flutter allow you to handle asynchronous operations in a non-blocking manner, ensuring that your app remains responsive and doesn't freeze while waiting for long-running tasks to complete.

Update

To update an existing value in Shared Preferences, you can simply call the set method with the new value for the same key. Here's an example:

```
void updateData(String key, String value) async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setString(key, value);
}
```

In the above example, we create a method updateData that takes a key and a value as parameters. We obtain an instance of SharedPreferences using the getInstance method and update the value using the setString method.

Delete

To delete a value from Shared Preferences, you can use the remove method, providing the key of the value you want to remove. Here's an example:

```
void deleteData(String key) async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.remove(key);
}
```

In the above example, we create a method deleteData that takes a key as a parameter. We obtain an instance of SharedPreferences using the getInstance method and remove the value using the remove method.

Some Main Widgets Used in the tutorial

1. Column and Row

Column and Row are two fundamental layout widgets in Flutter that help you organize and arrange other widgets in a vertical or horizontal direction, respectively. Column arranges its children vertically, whereas Row arranges its children horizontally. These widgets are often used to create responsive UIs and control the placement and alignment of multiple widgets within a parent container.

More about column and row and main axis alignments: https://medium.com/flutter-community/flutter-layout-cheat-sheet-5363348d037e

2. SizedBox

SizedBox is a widget in Flutter that allows you to specify a fixed width and/or height for its child widget. It is commonly used to add empty space or provide a specific size constraint to other widgets. SizedBox is useful when you need to add padding, margins, or create fixed-size gaps between widgets.

3. TextFormField

TextFormField is a widget in Flutter that combines the functionality of a TextField widget with additional features specifically designed for handling forms and user input. It provides built-in validation, auto-complete suggestions, input formatting, and error handling capabilities. TextFormField is commonly used to collect user input, such as email addresses, passwords, or any other textual data.

4. Container

Container is a versatile widget in Flutter that allows you to create a rectangular visual element with customizable properties, such as color, padding, margin, border, and alignment. It can contain other widgets and is often used as a layout component or to provide styling and visual effects to its child widgets.

5. ListView.builder

ListView.builder is a widget in Flutter that enables you to efficiently create scrollable lists with a large number of items. It builds only the visible items on the screen dynamically, which helps conserve memory and improves performance. ListView.builder is commonly used when you have a dynamic list of items that may exceed the screen size.

6. Card

Card is a material design widget in Flutter that provides a stylized container with rounded corners, a shadow, and padding. It is used to present related pieces of information as a single cohesive unit. Cards can be used to display images, text, or a combination of both, and they are often used in lists, grids, or as standalone elements in an application's user interface.

7. Visibility

Visibility is a widget in Flutter that allows you to control the visibility of its child widget based on a boolean value or an animation. It can be used to show or hide widgets dynamically in response to user interactions or certain conditions. Visibility is commonly used when you want to conditionally display or hide a widget based on certain criteria.

8. SingleChildScrollView

SingleChildScrollView is a widget in Flutter that provides a scrolling container for a single child widget. It allows the child widget to exceed the screen size vertically or horizontally and enables the user to scroll and view the content. SingleChildScrollView is commonly used when you have a widget that exceeds the available screen space and needs to be scrollable.

9. initState and setState

initState and setState are methods provided by the StatefulWidget class in Flutter. initState is called when the widget is inserted into the widget tree for the first time, and it is typically used to initialize variables or perform one-time setup operations. setState is used to update the state of a widget, triggering a rebuild of the user interface with the updated data. It is commonly used in response to user interactions or when data changes need to be reflected in