National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

EC-303: Mobile Application Development

Class: BESE 8AB + BSCS 7AB

Lab 12: Storing data to local persistent storage and restoring from local storage.

Date: 26th May 2021

Time: 9:00 AM - 12:00 PM

Instructor: Dr. Hassan Ali Khattak

Lab Engineer: Mr. Aftab Farooq

Lab 12: Storing data to local persistent storage and restoring from local storage

Objectives

The objective of this lab is helping students to familiarize themselves that how to store data to local persistent storage and restore from local storage.

Description

Most mobile apps need to store persistence data locally. There are many reasons to do so, from restoring the app state at the last session, to improve app performance. Therefore, Flutter offers us several ways to accomplish this feature. In this lab, I will introduce to you 3 different ways to store persistence data in Flutter:q

- Store key-value to shared_preference
- Saving data to a local database
- Writing and reading a file

Tools/Software Requirement

Flutter, Android Studio, XCode, Any Editor

Helping Material:
- Flutter official Documentation
- https://www.solutelabs.com/blog/flutter-tutorial-for-beginners-step-by-step-guide.
- Stack overflow
- Flutter Community

Helping Link for Icons & Logo:
- https://icons8.com/icons/set/mobile-app

Prerequisites

I'm assuming that you:

- Have the Flutter development environment set up (This tutorial was tested with Flutter.
- Know the way around your IDE (I'm using Android Studio, but VSCode and IntelliJ are fine, too.)
- Have experience creating a basic Flutter app

Setup

Start a new Flutter project. I'm calling mine **flutter_saving_data.**

Lab Task

Saving data to a local database

For large amounts of data SharedPreferences is not a good option. Here are some examples of the kind of data I am talking about:

- Names and addresses
- A word frequency list
- High scores

To develop an app that needs to persist a large amount of data, structure data, you should consider using a local database. Basically, we can query and write easier.

In Android, developer can use SQLite to serve as a local database. With iOS, it is Core Data. However, we're still able to use the SQLite database in iOS to avoid the complicated of handling Core Data. In Flutter, we simply use a plugin named sqflite for both iOS and Android app.

Installing

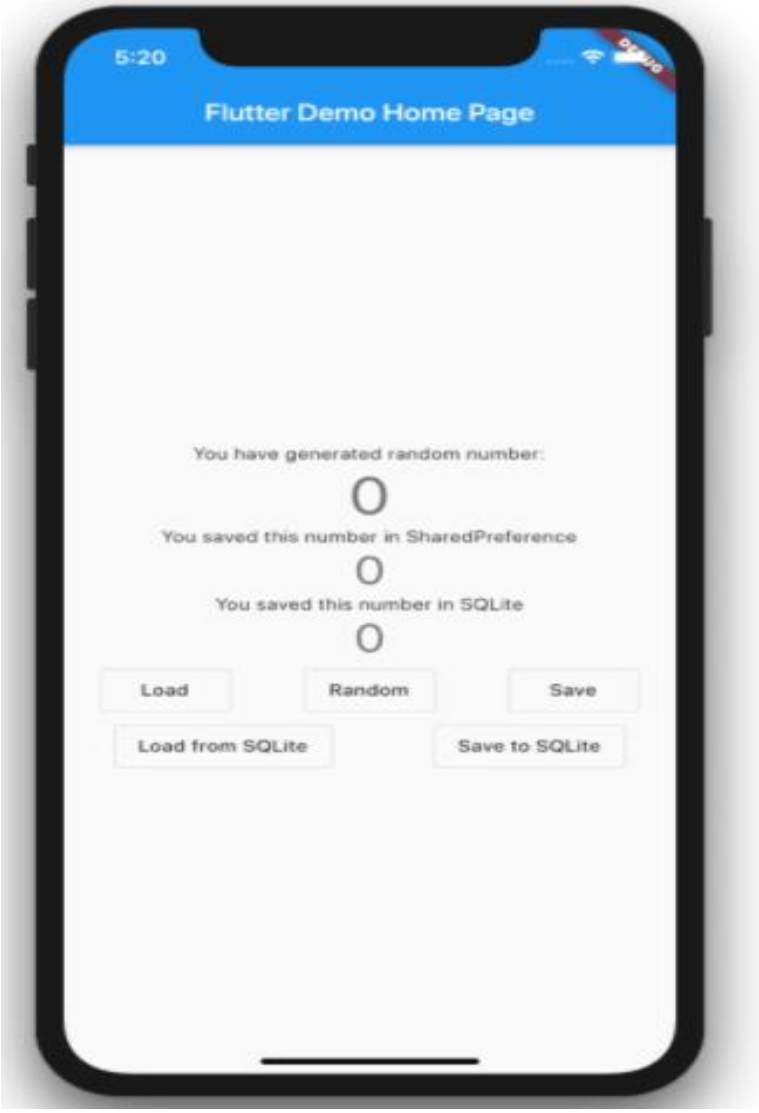
Add **sqflite** and path dependencies in **pubspec.yaml**

```
dependencies:
  flutter:
    sdk: flutter
  sqflite: ^1.3.0
  path: ^1.6.4
```

I usually use sqflite with path plugin. While sqflite interacts with SQLite database, path helps us get exactly directory where the database is.

Build an Example

Continue with the previous example app, we add two more buttons to insert and load data from database.



Implementation

Now we implement a simple example to show how sqflite works. When working with SQLite database, we should follow these points:

- You should create a Database Service to interact with the database. Keep it separated with other files for clear and maintain it.
- It's highly recommended to make Database Service as a singleton object. It means we will open only a connection to the database. Avoid too many connections can reduce your app performance.

Firstly, we create a model class to reflect the data structure of a table. Although, it's not required to work with SQLite, we can use data easier within the model. It is also great to define some helper function to verify data or converting data to Map before insert to database.

Now we define a model class RandomNumber, named the file as **random_number.dart**

```dart
class RandomNumber {
  int value;
  DateTime createdTime;
  RandomNumber(this.value, this.createdTime);

  RandomNumber.fromMap(Map<String, dynamic> map)
      : assert(map["value"] != null),
        assert(map["createdTime"] != null),
        value = map["value"],
        createdTime = map["createdTime"] is String
            ? DateTime.parse(map["createdTime"])
            : map["createdTime"];

  Map<String, dynamic> toMap() {
    return {
      "value": this.value,
      "createdTime": this.createdTime.toString(),
    };
  }
}
```

- And then, we will create a Service to do database action such as open connection, insert, update, delete, or close the connection.
- So why we define a separate service to do these jobs. We define a database service because of two below points:
- Firstly, every business code that related together should be placed in one file. So it makes a clean, clear, and readable code. Furthermore, if we change plugin from sqflite to another one day, we just need to update this service.
- And then, we will make this service as a singleton class. It means we only open one connection to the database and we can use this service everywhere around your app.

And the last one is adding an insert and query button to **main.dart**. When user clicks to button, it calls database service to get saved number.

Writing and reading a file

In some cases, database can't fit well your data. You may need to write to a local file and read it later. I can list down here the cases:

- Save data across app launches.
- Save log file.
- Export database as a CSV file

To write and read a file, we use path_provider plugin to get the directory to save file and write file by using dart:io built-in library.

We should note that there are two types of local directory:

- **Temporary directory**: it usually use for cache. It can be clear anytime. On iOS, it called NSCachesDirectory. And cache directory in Android.
- **Document directory**: A directory that app can store files that only it can read. We should use this type to persist data. It's named AppData on Android, and NSDocumentDirectory on iOS.

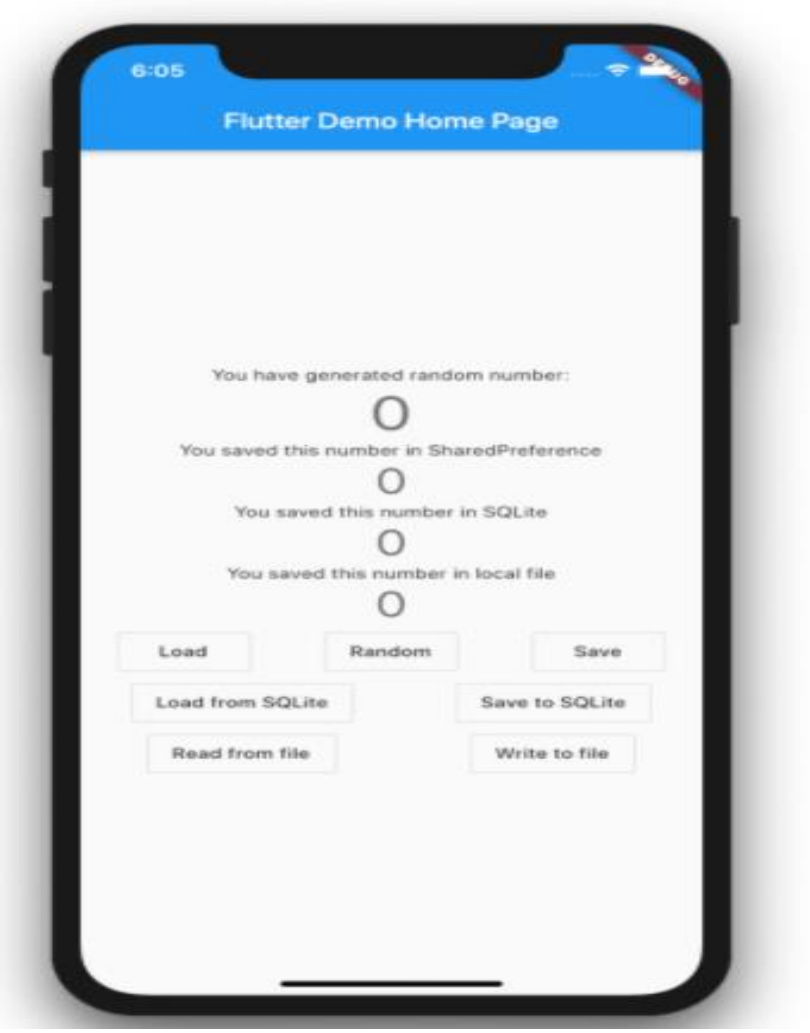
Installing

We add **path_provider** as project dependencies:

```
dependencies:
  path_provider: ^1.6.7
```

Build an Example

Now continue to add the other two buttons to this example which allow us to write and read data. Below is an example image



Implementation

We add below function to **main.dart** to get the document directory and the specific file to read and write.

```dart
Future<String> get _localPath async {
  final directory = await getApplicationDocumentsDirectory();
  return directory.path;
}

Future<File> get _localFile async {
  final path = await _localPath;
  return File('$path/randoms.txt');
}
```

In the code, we use **getApplicationDocumentsDirectory()** function to get the document directory which was discussed above. You can get temporary directory by using **getTemporaryDirectory().**

We also hard-coded file path as $path/randoms.txt, you can change to whatever you want. Now create two buttons to read and write data to file.

Conclusion

After a long post, we know three different methods to persist data in Flutter. Each method shows its pros and cons in a specific circumstance. Therefore, understand it well can help you choose your best option and make your app performance better.

Deliverable

Compile a single word document by filling in the solution part and submit this Word file on LMS. This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva/quiz related to the tasks. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to Mr. Aftab Farooq : aftab.farooq@seecs.edu.pk.