CMSC 23: Mobile Computing

Week 06: Persistence

Objectives

At the end of this Module, the students will be able to accomplish the following:

- Write data to the file
- Read data from the file
- Create an application that uses persistence

Read and Write data in file

In some cases, you need to read and write files to disk. For example, you may need to persist data across app launches, or download data from the internet and save it for later offline use.

To save files to disk, combine the **path_provider** plugin with the **dart:io** library.

To do that, we need first to define our methods inside a class that will access our files. This recipe uses the following steps:

1. Find the correct local path.
2. Create a reference to the file location.
3. Write data to the file.
4. Read data from the file.

Step 1. Find the correct local path

The **`path_provider`** package provides a platform-agnostic way to access commonly used locations on the device's file system. The plugin currently supports access to two file system locations:

1. *Temporary directory*

   A temporary directory (cache) that the system can clear at any time. On iOS, this corresponds to the **`NSCachesDirectory`**. On Android, this is the value that **`getCacheDir()`** returns.

2. *Documents directory*

   A directory for the app to store files that only it can access. The system clears the directory only when the app is deleted. On iOS, this corresponds to the **`NSDocumentDirectory`**. On Android, this is the **`AppData`** directory.

Create a new flutter project named week6_persistence, then inside the lib folder create a new file named file_storage.dart. Lastly define a class for FileStorage, this class will contain all our methods for accessing the file(in this example, storing and retrieving dogs data). All the codes from steps 1-4 must be pasted inside the FileStorage class definition. The necessary libraries/packages have already been included in the import.

```
import 'dart:async';
import 'dart:io';
import 'package:path_provider/path_provider.dart';

class FileStorage{}
```

Don't forget to install the path_provider package by typing the code below in your terminal:

`flutter pub add path_provider`

The example below stores information in the documents directory. You can find the path to the documents directory as follows:

```
//returns the correct local path to store files
Future<String> get _localPath async {
  final directory = await getApplicationDocumentsDirectory();

  return directory.path;
}
```

Step 2. Create a reference to the file location

Once you know where to store the file, create a reference to the file's full location. You can use the **`File`** class from the **`dart:io`** library to achieve this. Copy the code below in the FileStorage class. By doing this, we can reuse this class to save different types of data.

```
...
final String filePath;

FileStorage(this.filePath);
...

//reference the file you will use to store data
Future<File> get _localFile async {
  final path = await _localPath;
  return File('$path/$filePath');
}
```

Step 3. Write data to the file

When saving data anywhere, it is a good practice to create a model for the data that we are going to save. Create a file named dog_model.dart and paste the following code:

```
class Dog {
  String name;
  String age;

  Dog(this.name, this.age);

  get dogData {
    return '${name} ${age}\n';
  }
}
```

Now that you have a **`File`** to work with, use it to read and write data. First, write some data to the file. To write a string to a file, use the **`writeAsString`** method:

```
  //append data to the localfile
  Future<File> writeString(String data) async {
    final file = await _localFile;
```

```
  // Append text to the file
  return file.writeAsString(data, mode: FileMode.append);
}
```

The **file.writeAsString** method can have a parameter named **mode** where you can specify the **FileMode** that you want to use such as append, writeOnly, etc. We can use this method to add individual items to our file.

If you only want to override the data in the file and not append, then you can remove the mode parameter. Above is an example code on how to append data to your file while below is an example of writing data to your file. Add this to your FileHandler as well.

```
//overwrite data to the localfile
Future<File> writeAll(String data) async {
  final file = await _localFile;
  return file.writeAsString(data);
}
```

Step 4. Read data from the file

Now that you have some data on disk, you can read it. Once again, use the **`File`** class. The **`readAsString()`** method reads the entire file contents as a **string**, while **`readAsLines()`** method reads data per line and returns a list of lines of strings. In this example we will use the readAsLines method because we will save every dog data per line.

```
//read data from the file as lines
//this will return a list of string per line
Future<List<String>?> readFile() async {
  try {
    final file = await _localFile;

    // Read the file
    final contents = await file.readAsLines();

    return contents;
  } catch (e) {
    // If encountering an error, return 0
    return null;
  }
}
```

There are also different read methods available for files, you may read about them in this link

Rendering data from file

Now, let us render the file content to our application user interface. To do that, clean up the boilerplate code in main.dart by deleting everything, then copy paste the code below

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

//import necessary packages
import 'show_dogs.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  static const String _title = 'Dogs List';

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: _title,
      theme: ThemeData(
        primarySwatch: Colors.green
      ),
      home: ShowDogsPage(),
    );
  }
}
```

ShowDogsPage is a widget that will render our dog data inside a listview. We have to create an instance of the **FileStorage** class to our homepage(ShowDogsPage) so that we can access our file methods defined in the FileStorage widget inside the ShowDogsPage.

Let's create a new file named **`show_dogs.dart`** then copy and paste the code below. Let's inspect the code and try to predict what the demo code will do when we build it.

```
//import necessary packages
import 'file_storage.dart';
```

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

class ShowDogsPage extends StatefulWidget {
  const ShowDogsPage({Key? key}) : super(key: key);

  @override
  State<ShowDogsPage> createState() => _ShowDogsPageState();
}

class _ShowDogsPageState extends State<ShowDogsPage> {
  //create a future list of string that will store all the dog data
from textfile
  late Future<List<String>?> myDogs;
  FileStorage fileHandler = FileStorage('dogs.txt');

  @override
  void initState() {
    super.initState();
    //initialize myDogs list by getting data from textfile

  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Dogs List"),
      ),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        mainAxisSize: MainAxisSize.min,
        verticalDirection: VerticalDirection.down,
        children: <Widget>[
          myDogsList(),
        ],
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          //navigate to AddDogPage page and automatically calls the
setState()
          //to render newly added dog in textfile when Navigator.pop
is called from AddDogPage page
          Navigator.of(context)
              .push(MaterialPageRoute(
                  builder: (context) =>
                      AddDogPage(title: 'Add Dog', fileHandler:
fileHandler)))
              .then((value) => setState(() {
```

```
                    myDogs = fileHandler.readFile();
                  }));
        },
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }

  //Since readFile method will return a Future list, we need a Future
builder to build our list
  //using listView.builder
  Widget myDogsList() {
    return Expanded(
      child: FutureBuilder(
          future: myDogs,
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              return buildText(snapshot.data as List<String>);
            } else {
              //if textfile has no data
              return const Center(
                child: Text("No dogs found"),
              );
            }
          }),
    );
  }

  void _deleteDog(myDogs, item) {
    setState(() {
      if (myDogs != null) {
        myDogs.remove(item);
      }
    });
  }

  void _writeStrings(myDogs) {
    String str1 = "";

    for (String str in myDogs) {
      str1 = str1 + str + '\n';
    }

    fileHandler.writeAll(str1);
  }

  //widget that returns listview of myDogs data
  Widget buildText(List<String> myDogs) {
```

```
    return ListView.builder(
      physics: const BouncingScrollPhysics(),
      itemCount: myDogs.length,
      itemBuilder: (context, int index) {
        return Card(
          child: ListTile(
            title: Text(myDogs[index].split(' ')[
                0]), //dog name is the first item from returned list
of split()
            subtitle: Text(myDogs[index].split(' ')[
                1]), //dog age is the second item from returned list
of split()
            trailing: const Icon(Icons.delete),
            onTap: () {
              //TODO: insert method to delete data from file
              _deleteDog(myDogs, myDogs[index]);
              _writeStrings(myDogs);
            },
          ),
        );
      },
    );
  }
}
```

The code below is an example of how to read data from a file. We call it in the **`initState`** method so that we can render to the user the content of the textfile when the application is used.

```
  @override
  void initState() {
      super.initState();
      //initialize myDogs list by getting data from textfile
      myDogs = fileHandler.readFile();
    }
```

Writing data to file

Now that we can read data from the file and render it to our app, let's create a page to add new dog data in the file using the writeString method. To do that, we need first to create a new file named add_dog.dart and paste the code below. Let's inspect the code and try to predict what the demo code will do when we build it.

```
import 'package:flutter/material.dart';
import 'dog_model.dart';
import 'file_storage.dart';

class AddDogPage extends StatefulWidget {
  const AddDogPage({Key? key, required this.title, required
this.fileHandler})
      : super(key: key);
  final String title;
  final FileStorage fileHandler;
  @override
  State<AddDogPage> createState() => _AddDogPageState();
}

class _AddDogPageState extends State<AddDogPage> {
  final TextEditingController _name =
      TextEditingController(); //controller for getting dog name
  final TextEditingController _age =
      TextEditingController(); //controller for dog age
//data to be inserted to file

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          buildTextField('Enter name', _name),
          buildTextField('Enter age', _age),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              buildSaveButton(),
              buildViewButton(),
            ],
          )
        ],
      ),
    );
  }

  //function for creating a textfield widget which accepts the label
and controller as parameter
  Widget buildTextField(String label, TextEditingController
_controller) {
    return TextField(
```

```
      controller: _controller,
      decoration: InputDecoration(
        border: const OutlineInputBorder(),
        labelText: label,
      ),
    );
  }

  //function for creating save button
  Widget buildSaveButton() {
    return ElevatedButton(
        child: const Text('Save'),
        onPressed: () {
          //create an instance of the data you want to save

          //write formatted data to textfile

          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(content: Text('Processing Data')),
          );
          //clear text on controllers after successful insert of data
to database
          _name.clear();
          _age.clear();
        });
  }

  Widget buildViewButton() {
    return ElevatedButton(
      onPressed: () {
        //go back to list of dogs page
        Navigator.pop(context);
      },
      child: const Text("View"),
    );
  }
}
```

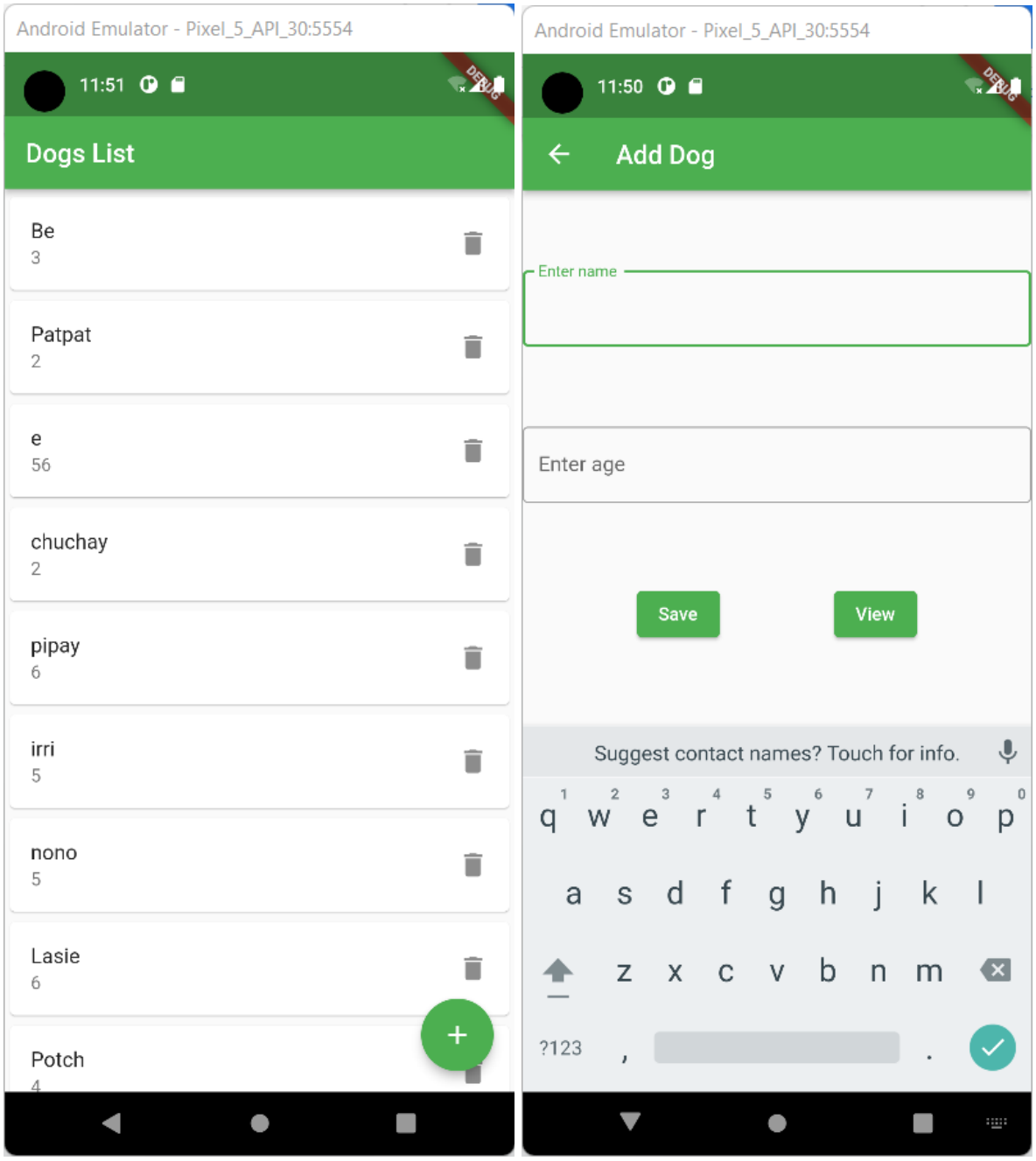
Before we call the **`writeString`** method of FileStorage class, we need first to format the data we will save in our text file. Because our read data uses readAsLines method, we need to add a newline at the end of every dog entry in our text file. In the example below we formatted the dog data with a space between the name and age so that we can easily split the data when rendering a list of dogs in the app (see the getter method for dogData in the dog model). Add the lines of code inside the onpressed method of the buildSaveButton.

```
//create an instance of the data you want to save
Dog dog = Dog(_name.text, _age.text);
```

Next, let's call the **`writeString`** method of FileStorage class so that we can write the dog data to our file every time the saved button is pressed. Update onpressed method and add the code below.

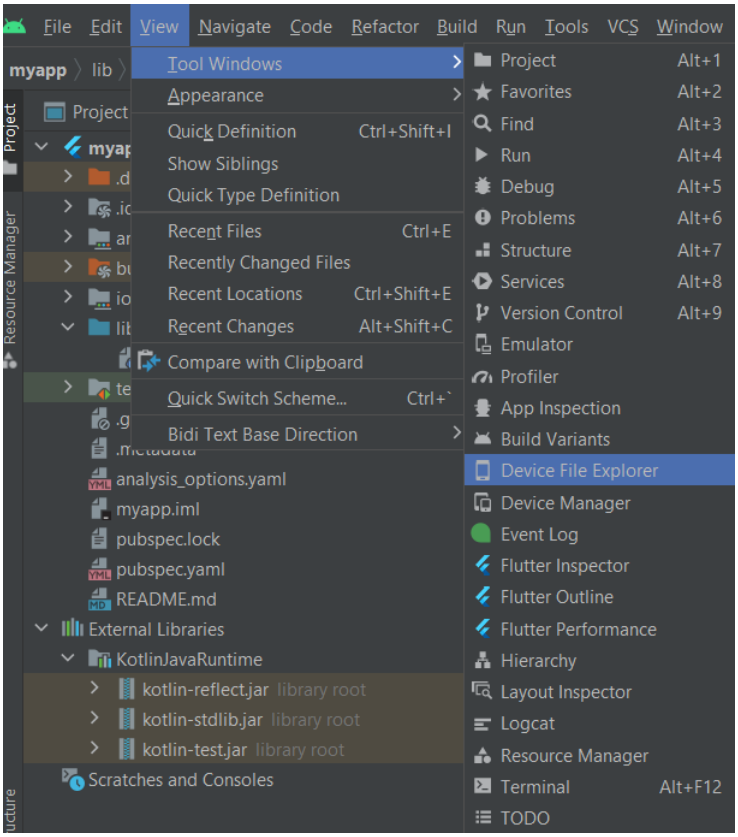
```
//write formatted data to textfile
widget.fileHandler.writeString(dog.dogData);
```

Run the main dart file and voila! We can now persist data using files! Close the app and run again, the data saved must still be there. Below are the sample screenshots of the fully working app.

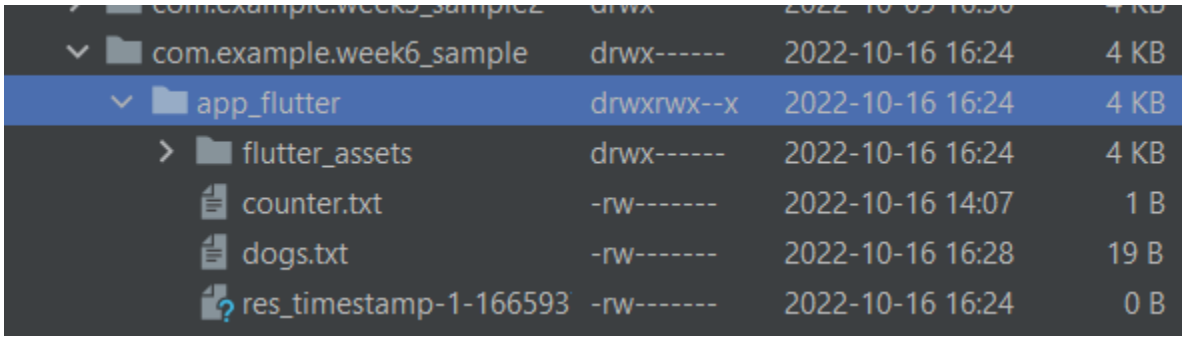

Android Emulator - Pixel_5_API_30:5554
11:51
Dogs List
Be
3
Patpat
2
e
56
chuchay
2
pipay
6
irri
5
nono
5
Lasie
6
Potch
4
Android Emulator - Pixel_5_API_30:5554
11:50
Add Dog
Enter name
Enter age
Save
View
Suggest contact names? Touch for info.

Checking your created data

To view the created files, you need to open the Android Studio application, then go to View > Tools Window > Device File Explorer as seen in image below.



Then after, you click the Device File Explorer, you will see a list of files under your choice device. Note that the emulator must be open to view the list. Then on the Device File Exploer list go to data > data > then find the name of your app (e.g com.example.week6_sample) > app_flutter



You will now see the created text file.

References:

- Biessek, A. (2019). *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2*. Packt Publishing Ltd.
- Iversen, J., & Eierman, M. (2014). *Learning mobile app development: A hands-on guide to building apps with iOS and Android*. Pearson Education.
- Panhale, M. (2016). *Beginning hybrid mobile application development*. New York: Apress.
- Payne, R. (2019). *Beginning App Development with Flutter: Create Cross-Platform Mobile Apps*. Apress.
- Read and write files https://docs.flutter.dev/cookbook/persistence/reading-writing-files