Research Notebook

Joshua Markle

Contents

Timesheet

Date	**Time Completed**
3 October 2023	2h 30m
4 October 2023	1h 30m
5 October 2023	2h
14 October 2023	5h
15 October 2023	1h 30m
22 October 2023	1h 30m

App Setup and Learning Process

In this section, I will research different methods of app development with the goal of deciding how to build my app and what I will use to make it.

3 October 2023 - App Frameworks

In order to make an app, some sort of framework that provides a standard way to build and deploy applications.

What I am looking for in a framework:

- Cross platform: Can build to both iOS and Android
- Established community: Pre-existing libraries and a place to go if I run into errors

In my search for suitable framworks, I have found that the two most popular ones are probably going to give me the best success in this project. These two frameworks are called React and Flutter, and dominate most of the app building options.

	React Native	Flutter
General	Made by facebook, React uses JavaScript to make it's components. It offers a variety of UI options and more native experience	Flutter was made by Google and has it's own take on the app development process. It uses Dart to render widgets (display components)
Pros	• Easy to learn and master (JavaScript) • Native UI unique to iOS and Android • Best for complex animations	• Dart is similar to other languages I know • Excelent documentation • Standard UI across both iOS and Android • Highly performant
Cons	• Poor documentation • Constant updates (many things get deprecated) • Overwelhming number of options/solutions	• Harder to master • Limited libraries by 3rd parties • Large app sizes

Sources

- React Native vs Flutter, cross-platform mobile application frameworks (Wenhao Wu)
- Flutter vs. React Native: Which One to Choose in 2023? (Chinmayee Deshpande)

- Flutter vs React: A Comparison

4 October 2023 - Project Setup

After conducting the research on these two popular frameworks, I have decided to go with the Flutter framework ultimately because of the development process. It has better documentation that is simple and straight to the point and can make working on apps quicker and cause less suffering.

I have setup my development environment using Neovim, a popular text editor that I have been using for a while. It is highly configurable and I plan of leveraging exactally that to hasten development.

Neovim packages installed:

- flutter-tools.nvim by akinsho
- vim-lsc and vim-lsc-dart by natebosch

(Packages can be found on GitHub)

Both of these packages allowed me to setup my project and provide helpful code completion. I also installed the Android operating system onto my laptop inorder for me to run a virtual phone on my computer to test out my app in realtime.

Currently, I have created a blank project and am fully preped for development.

5 October 2023 - Common Knowledge & Best Practices

Before work can be started on any large project, it is important that there is a plan to guide development and ensure that time is not wasted on extra tasks.

Notes pertaining to building a flutter app:

- Widgets: Building blocks of UI
- Stateless Widgets:
  - Don't store mutable state
  - Rebuild only when parent changes
- Stateful Widgets:
  - Maintain mutable state
  - Can rebuild when internal state changes
- Packages & Plugins:
  - Reusable components
  - Find at `https://pub.dev/flutter`
- Basic Flutter App Structure
  - `main.dart` as the entry point
  - Define `main()` function
  - Define at least one `Widget`
- Creating a New App
  - Use terminal or command prompt
  - `flutter create [app_name]`
  - `cd [app_name]`
  - `flutter run` to start the app
- Additional Resources
  - Flutter Documentation: `https://flutter.dev/docs`
  - Flutter Community: `https://flutter.dev/community`

Below is a sample project that builds necessary components for the app and centers text within the center of the screen.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Stateless Widget Example'),
        ),
        body: Center(
          child: Text('Hello, Flutter!'),
        ),
      ),
    );
  }
}
```

This code demonstrates a stateless widget and how Flutter structures their app components. Each class can contain Widget that returns a page to the user's screen when called. Stateful widgets on the other hand update in realtime on the user's screen unlike a stateless widget. This is the core of flutter, everything is a widget either stateless or stateful.

Sources Reviewed:

- Test Drive (Flutter Documentation)
- How To Build Mobile Apps With Flutter (FreeCodeCamp)
- GeeksForGeeks Flutter Documentation

14 October 2023 - Sample App

The purpose of this session is to make a sample app that does something and learn more about flutter's UI system.

After messing around with the UI system, I have learned about the most common features and widgets (A widget is any object within the app that displays, aligns, or alters the screen):

- **Text**: This widget displays text and the `style` characteristic allows for different text styles.
- **Row**: This aligns all children widgets in a row
- **Container**: A convient way of surrounding oter widgets and specifying various characterstics like: `width, height, background color, and aligning`. I will probably use this a lot when making modular objects.
- **FlatButton**: (also TextButton and RaisedButton) Used to create a button with a material design (the app design default to flutter apps)
- **Scaffold**: A widget that typically holds EVERY other widget. It provides general framework and structure to the the app like adding an `AppBar` or `Body` or `NavigationBar`.
- **Image**: A way for flutter to render in images
- **Icon**: A super awesome way to adding icons into any app. Flutter has thousands of builtin icons so I will not have to pay for icons in any app I make.
- **ListView**: A scrollable list that works well for long lists of items. Each item is also configuable but that looks complicated.
- **Stack**: A way to stack widgets ontop of each other. Like stacking squares ontop of each other to make a neat image.
- **Card**: A short and easy way to make a box with rounded corners and a drop shadow. The same exact effect can also be achieved with only containers.

Using a variety of these basic widgets, I was able to tweak the sample app (that is in place when you make a new projects) to change the title and add a container that contains a counter that increments one when a button on the bottom is pressed!

15 October 2023 - Combining Widgets

The essense of flutter is the ability to combine widgets to make a sleek and functional UI that users will want to interact with. The purpose of this entry is to learn more about combining widgets together in order to make a pretty and user friendly design.

- **Scaffold with AppBar and FloatingActionButton**:
  The `Scaffold` widget provides a canvas where we can place other widgets like `AppBar` and `FloatingActionButton` to quickly establish the basic structure of an app.

```
Scaffold(
  appBar: AppBar(title: Text('My App')),
  body: Center(child: Text('Hello World')),
  floatingActionButton: FloatingActionButton(
    onPressed: () {},
    child: Icon(Icons.add),
  ),
)
```

- **ListView with ListTile**:
  Combining `ListView` and `ListTile` provides a quick way to present lists with additional properties like leading icons and trailing actions.

```
ListView(
  children: [
    ListTile(
      leading: Icon(Icons.mail),
      title: Text('Email'),
      trailing: Icon(Icons.navigate_next),
    ),
    ListTile(
      leading: Icon(Icons.message),
      title: Text('Messages'),
      trailing: Icon(Icons.navigate_next),
    ),
  ],
)
```

- **Stack with Positioned**:
  This combination is powerful for overlaying widgets on top of each other. With the `Positioned` widget, we can specify the exact position of a child inside a `Stack`.

```
Stack(
  children: [
    Container(color: Colors.yellow, child: Text('
Background Text')),
    Positioned(
      top: 50,
      left: 50,
      child: Text('Overlaid Text'),
        ),
      ],
    )
```

The cool thing about flutter is that everything is modular and adding these widgets is easy and straight forward. Together, they can make something beautiful.

22 October 2023 - The Flutter Package System

The purpose of this entry is to learn about Flutter's packaging system and how I can use other people's code instead of building large features from the groud up.

Popular Flutter Packages

- **provider**: A state management solution that's recommended by the Flutter team. It simplifies data flow in applications and provides easy access to shared state.
- **http**: A comprehensive package for sending HTTP requests. It provides both low-level classes for sending requests directly and high-level functions for fetching and uploading data.
- **sqflite**: A SQLite plugin for Flutter, allowing for local database storage, querying, and other operations.
- **shared_preferences**: A package for storing simple data persistently. It wraps around native capabilities to store key-value pairs.
- **flutter_bloC**: A predictable state management library that helps implement the BLoC (Business Logic Component) design pattern.

Installing Flutter Packages The process of installing and using a Flutter package is remarkably streamlined.

1. Navigate to `pub.dev` and search for the desired package.
2. On the package's page (typically at pub.dev), locate the `Installing` tab. Here, find the package's latest version string.
3. In the Flutter project's `pubspec.yaml` file, add the package under `dependencies` with the correct version. For instance:

   ```
   dependencies:
     flutter:
       sdk: flutter
     provider: ^latest_version
   ```

4. Run `flutter pub get` in the terminal. This will fetch and install the package.
5. Once installed, the package can import the package in the Dart code with:

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

The existence of these packages significantly accelerates the development process, as reinventing the wheel becomes unnecessary. The community-driven nature of the ecosystem means developers can use solutions from experts.