Widget technology

Introduction

In this research paper you’ll find the results and conclusion of what technologies will be selected to use when developing the widget app.

Goal

The goal is to understand what technologies are most effective to develop a widget app for Android.

Motivation

The widget app will eventually need to be developed, possibly published and upscaled. Research into what technologies are the most effective solution needs to be done.

Main Question

What technologies are most effective/suited for developing the F1 Android widget application?

Hypothese

My expectation is that Android will be the most effective/suited technology, since native platforms typically offer more support for less significant features compared to hybrid frameworks like Flutter, which are newer technologies and generally have less support for features like widgets.

Sub questions

- What technology allows you to make widgets for Android?
- What are the criteria and what are the positives and negatives for each technology?
- What additional/third party technologies are needed?

Plan of action

Desktop and workshop research will be applied.
A list of the criteria will be made to judge what technologies are the most effective/suited to use for developing the widget app. In addition, POC's will be made in order to get a better sense of what technologies will be most suited. The criteria list is based upon the desktop and workshop research findings.

Results

What technology allows you to make widgets for Android?

After googling ‘how to make widgets for Android’ and ‘what platforms/programming languages allow you to make widgets for Android’, I concluded that the only options to make widgets for Android, are either Android (with Kotlin or Java) itself or using one of the many hybrid frameworks that enable you to develop widgets cross-platform.
Since I have experience with hybrid frameworks, I know which frameworks have the most support, biggest community and newest, up-to-date features. In these aspects, Flutter and React-Native come out on top. For these reasons, I will be looking at these hybrid frameworks only.

What are the criteria and what are the positives and negatives for each technology?

The main criteria are:

- Widget support; the main objective is to create a widget, so features like customization options are important to have
- Design support; the widget will have a design that ideally is implemented as quickly as possible
- Scalability; the widget app must be able to expand by being able to add more widgets

Other advantages and disadvantages will be noted as well.

Technology	**Widget Support**	**Design Support**	**Scalability**	**Other**
Android (Kotlin or Java)	+ Generating widget files (requires Android Studio) + Customizing widget (requires Android Studio)	+ Android Studio has an easy to use drag & drop design interface + Android has Material Design components	+ As a native language Android has a lot of support for expanding your app in a flexible way	+ Managing an Android project is easier compared to hybrid frameworks + Easier to publish app to the Playstore
Flutter (Dart)	+ Generating widget files (requires Android Studio) + Customizing widget (requires Android Studio)	+ Flutter has easy to use front-end components	- As a relatively new language/platform Flutter doesn't have the best scalability yet, although it has a lot better scalability than the older hybrid frameworks like Xamarin. + Google is committed to Flutter so you can expect Flutter to keep up the support	- Managing a Flutter project is more difficult compared to an Android project, because there's a lot of extra files (for iOS for example) + I'm more experienced with Flutter than the other technologies
React-Native (JavaScript)	+ Generating widget files (requires Android Studio) + Customizing widget (requires Android Studio) - Need to create a bridge between React Native and native Android in order to call	+ React-Native has a lot of front-end components based on Material Design	- React-Native has worse scalability as it's only meant to be used to create relatively small apps, this is also noticeable in the speed performances of React-Native which scores very low	- Managing a React-Native project is more difficult compared to an Android project, because there's a lot of extra files (for iOS for example) - Takes a lot of extra steps to set-up dev environment and additional tools

	Android-native modules like ‘SharedPreference’ so that you can control the widget with React-Native instead of Android Kotlin or Java.		compared to Flutter..	

POC

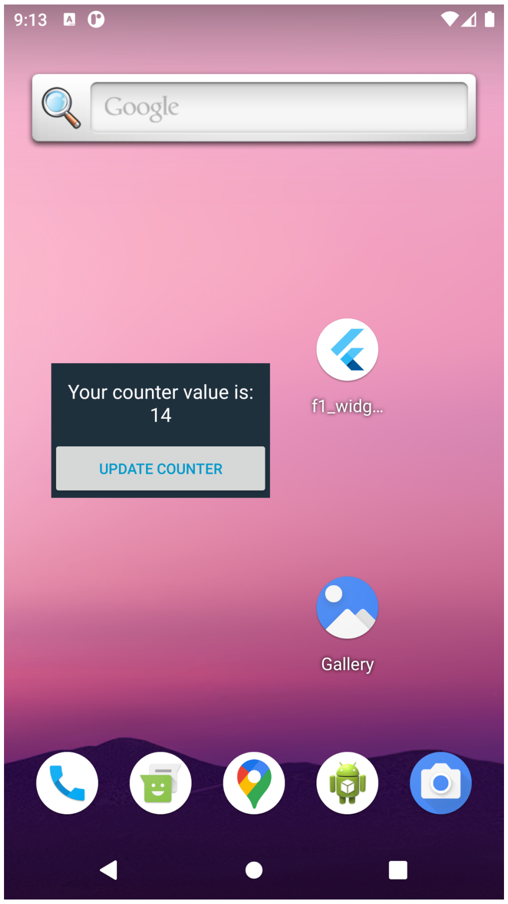
On paper Android looks to be the best option, but I wanted to see how Flutter would do in regard to developing widgets since I have a decent amount of experience with it compared to Android and especially React-Native.

Steps

1. I started the POC by generating a new Flutter project and opening it in Visual Studio Code.
2. For the next step I imported the 'Home Widget' plugin from Pub Dev
3. I open the project in Android Studio to generate the needed widget files
4. Customized the widget in Visual Studio Code; added a button click counter with Dart

Observations

- The 'Home Widget' plugin from Pub Dev is required in order to be able to develop widgets with Flutter, but you still need to write code for the widgets in native code so not with Dart.
- Had to switch a lot between IDE's, because I used Flutter for Visual Studio Code and have a lot of tools in that IDE, but I had to use Android Studio for generating files



What additional/third party technologies are needed?

Database
The widget app will require data to be stored somewhere so that it can be fetched and displayed. There's typically two options when it comes to databases: local or remote database. Remote databases typically get used for data that is sensitive and/or doesn't get used/fetched often. This is because fetching data from a remote database takes longer than when fetching data from a local database.

Since the F1 Widget app doesn't require sensitive data and not a big amount, using a local database will be more benificial.

According to Android's documentation, there are 4 ways to store local data.

- **App-specific storage:** Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
- **Shared storage:** Store files that your app intends to share with other apps, including media, documents, and other files.
- **Preferences:** Store private, primitive data in key-value pairs.
- **Databases:** Store structured data in a private database using the Room persistence library.

The last option 'Databases', seems the best option for the project since this functionality is best used when handling non-trivial amounts of structured data. The most common use case is to cache relevant pieces of data. That way, when the device cannot access the network, the user can still browse that content while they are offline. To use this optimally, the Android 'The Room' SQLite library will be used since it makes working with SQLite-databases easier and is recommended by the Android documentation.

API
The widget will require data in order for it to display the right info e.g. times and dates. Third party API's are often used for small projects like these since developers don't need to spend a lot of time creating a database and/or API themselves.

After doing research I have found quite a few F1 API's, but they did not have all the timings and dates of the race weekend events.
Examples of F1 API's:
- Ergast Developer API
- Postman Formula One API
- Api-Formula-1

Because I couldn’t find a fitting API, I contacted the developers of the iOS F1 widgets app called BoxBox to ask how they get data for their widgets/if they’re using a public API or not.
BoxBox told me they get all the data from F1.com and built their own database. I asked how they update the data when times/dates change. Their response was that one of the developers updates the data manually whenever there is a change.

Since there are no good API options, I should gather my own data and store it somewhere. The data is not sensitive, so could potentially be stored inside an array, JSON list, etc. as an easy solution. This will need to be researched by asking an expert or doing more desktop research.

Conclusion & Recommendations

Now that the research is done, the main question can be answered.
‘What technologies are most effective/suited for developing the F1 Android widget application?’

IDE
During this research I have noticed that both React-Native and Flutter need to generate files with Android Studio and need Android Studio to customize the widgets. This shows Android Studio will be the best IDE to use.

Language
From this research we can conclude that Android’s Kotlin/Java is the most suited language for developing the F1 Android widget application. Kotlin also goes well with the Android Studio IDE since it has a lot of supportive functions. Using Flutter and React-Native would make the development time unnecessarily longer and provide less widget functionalities.

Additional tools/technologies
No third-party API will be used since there’s no good option available. Instead, I will store the needed data somewhere myself.

What’s next?
More POC”s (e.g. POC of the database) need to be made to check if the software architecture will indeed be effective.