Developing Cross-Platform Library Using Flutter

Dilkhaz Y. Mohammed and Siddeeq Y. Ameen

***Abstract* — Third-party libraries are frequently utilized to save implementation time when developing new software. The significance of libraries in the creation of mobile applications cannot be overstated. Others can use the programmer's library created and shared with the rest of the world in their own projects as a result of your efforts. The purpose of this work is to create a taxi service library for developers using both Android and iOS, using Dart Object-Oriented Programming, Dio, and Retrofit. The programmer's creation of an interface for accessing platform-specific functionality from the library and creating Android and iOS apps from its projects needs to speed up software development. Therefore, the best solution is for the programmer to use it. Flutter is an open-source SDK for developing high-performance and more reliable mobile applications for operating systems like iOS and Android, from a single code base. Moreover, Flutter targets the top mobile operating systems like Android and iOS. When developing a Dart open source project, the common conclusion the programmer always ends up with is to share the produced outcomes with the developer community. In the dart world, the latter should be the least objective. This will quickly enable building an app without having to develop everything from scratch. It provides a solution for GPU rendering and UI, powered by native ARM code.**

***Key words* — Cross-platform Library, Dart, Flutter, Dio, and Retrofit.**

I. Introduction

Libraries are well-defined and are designed for reuse throughout implementation. For example, a website may have multiple web pages that implement the same navigation bar or text-field, but none of these objects have a relation to one another. And the mobile application development services have evolved into a higher level with APIs and when developers develop apps for the mobile, they rely heavily on APIs for connectivity. Which allows them to communicate seamlessly with the enterprise. In facts, APIs accelerate mobile development and enable tremendous agility for organizations that are going through their own digital transformation [8].

Developing a library and then sharing it with the rest of the world so that others can utilize it in their projects is not allowing application code to interface directly with native APIs is one of the major issues faced by cross-platform solutions. A naive option would be to use a cross-platform development framework to cover all expected interactions with native APIs. Due to the rapid growth of native APIs, this would necessitate ongoing maintenance. Furthermore, applications would be obliged to include unused wrappers, which would increase the size of the program. Flutter solves the issue by providing a set of services [7].

There are a lot of design patterns that programmers use for Flutter. They are all different ways of managing an app's state. The goal of a design pattern is to provide a clean standard for how our work will be organized, how the components will interact with each other, separate layers so that a change in one is transparent to the others, and most importantly, promote the reuse of blocks of code. Bloc is one of flutter recommendations state management and the core concepts of Bloc are Events and States [3].

This study is mainly aimed at building a taxi service library that other developers might use in their Android and iOS apps. The Retrofit library is a Dio client that makes consuming Rest APIs easier by sending dynamic headers, parameters, requests, and responses in a custom and secured way, as Dio is our HTTP client and handles the connection.

II. Related Work

In general, cross-platform development uses a single code base that can be executed on multiple platforms. Platforms in this sense typically refer to different operating systems provided by software or hardware vendors, such as Android and iOS. The traditional native approach uses native tools; the application communicates with the platform to create widgets or access services as shown in Fig. 1. The widgets are rendered on a screen canvas, and events are passed back to the widgets. However, the problem with this approach is that the programmer has to create separate apps for each platform because the widgets are different [2].

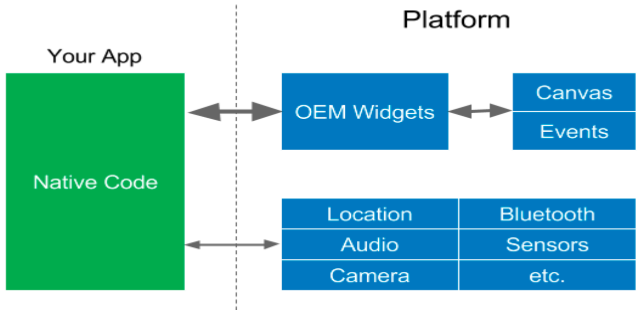


Fig. 1. Native Android/iOS code interacts with the platform.

The React Native approach is a well-known and popular JavaScript framework for cross-platform development. The programmer needs some native code for each platform they support, and then some JavaScript code to bind it all together. React Native, as shown in Fig. 2, uses a so-called bridge to

Submitted on February 08, 2022.
Published on March 04, 2022.

Dilkhaz Y. Mohammed, Scientific Research Center, Duhok Polytechnic University, Iraq.
(e-mail: Dilkhaz.mohammed@dpu.edu.krd)

Siddeeq Y. Ameen, Scientific Research Center, Duhok Polytechnic University, Iraq.
(e-mail: Siddeeq.ameen@dpu.edu.krd)

access the native platform widgets. This is the main reason why React Native can't beat a native app's performance: communication with native components occurs with the help of a JavaScript bridge. An additional layer causes slight delays in app loading. In most cases, this delay is too insignificant to notice, but some performance-critical functionality will make the difference crucial [2].

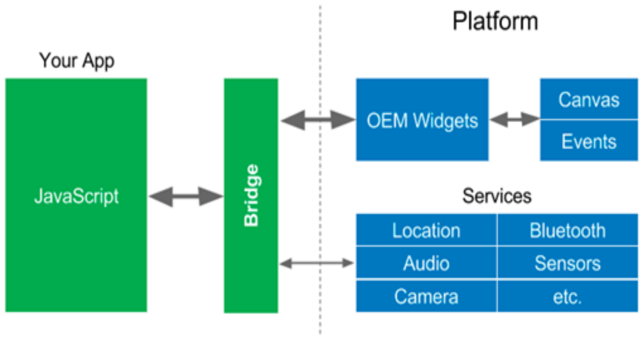


Fig. 2. React Native interacts with the platform.

Another approach across platform development uses the Multi-OS Engine and Java/Kotlin is shown in Figure 3 have only one codebase. Furthermore, the approach is useful due to the fact that, once compiled, such JAR files can be used in different projects on different platforms. Android Studio allows you to easily link frameworks and libraries contained in JAR to XCode projects and also specify all the specified resources necessary for the framework. Libraries will be copied to the final app file. However, the framework lies in the binding generator [4].

However, with the last two approaches, the app code communicates through a bridge, which may have performance implications. On the other hand, Flutter eliminates the bridge and moves the programmer's rendering into his app. Internally, Flutter consists of a framework built with Dart and a rendering engine built mostly in C++.

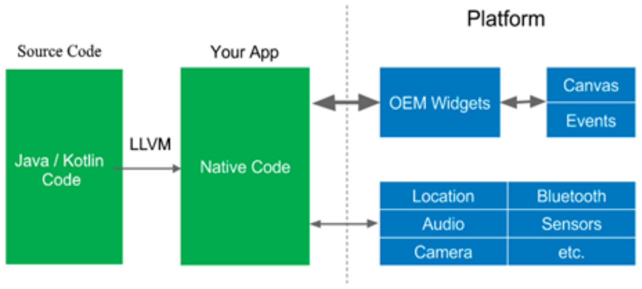


Fig. 3. Multi-OS Engine and Java/ Kotlin Native interacts with the platform.

III. Proposed System Development

These days, almost every mobile app connects to the internet to get and send data. The programmer should definitely learn how to work with responsive web services, as their proper implementation is essential when developing modern apps. Flutter's Retrofit is the easiest way to call rest APIs. In Dart applications, once such a library has been created, managing and deploying it is very convenient. However, to share code across platforms using Dart, to write platform-independent code and share it between Android and iOS with Flutter.

Flutter is an open-source SDK for developing high-performance, high-fidelity mobile apps for iOS and Android devices with the same codebase. Flutter uses the Dart programming language to create components and the Skia 2D graphics engine to bring code to life. A modern, react-style framework is also included in Flutter. The framework's content is depicted in Fig. 4. Skia is used to render the application's UI at the lowest level. Flutter uses a lightweight Dart virtual machine to run the majority of its framework and application code. The rendering engine is written in C++, whereas the framework code is written in Dart. Flutter, creates its own user interface on its own canvas and feeds it to a platform-specific engine.

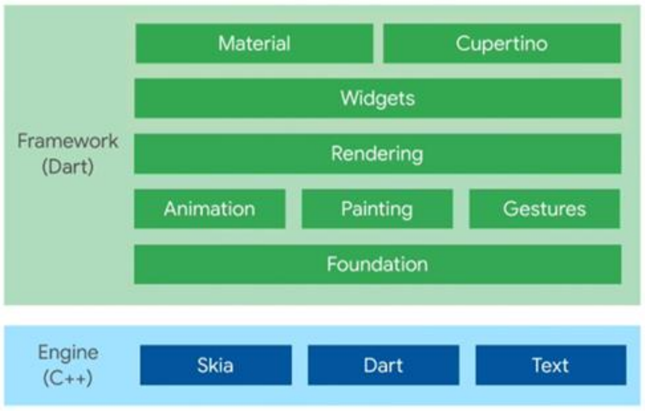
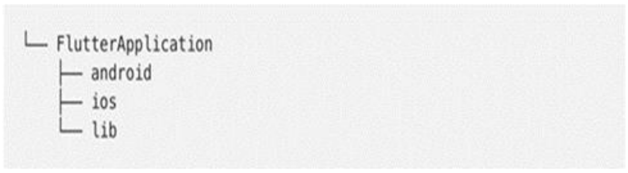


Fig. 4. Flutter framework and engine contents.

It's always been difficult to distribute software on many platforms, such as Android and iOS Mobile, due to the fact that the programmer must maintain a separate codebase for each platform. Flutter addresses this issue by allowing programmers to create mobile apps for both iOS and Android devices. Flutter uses a high-performance rendering engine to render each view component on its own. In terms of architecture, the engine's C or C++ code involves compilation with Android's NDK and LLVM for iOS, respectively, and during the compilation process, the Dart code is compiled into native code. Tim Sneath, group product manager at Google, similarly defines it as "a powerful general-purpose open UI toolkit." The built-in support currently is for the iOS and Android mobile platforms as shown in Fig. 5.



```
└─ FlutterApplication
   ├─ android
   ├─ ios
   └─ lib
```

Fig. 5. Flutter application structure.

Now that the programmer is familiar with all of the basic structures, he or she should be able to grasp how these layers communicate with one another. A new architecture has appeared in the Flutter community. The Bloc pattern is really useful when it comes to separating code into layers. Each layer, or set of classes, is in charge of a particular task. In this project, Data Layer, there is a directory. This data layer is used for the app's model and background communication.

The three main parts of this work are to begin with building a cross-platform library, followed by publishing the library onto the internet using a specific tool that the other developers might use in their Android and iOS apps, and lastly, developing Android and iOS apps using the library. The work organizational chart is shown in Fig. 6.

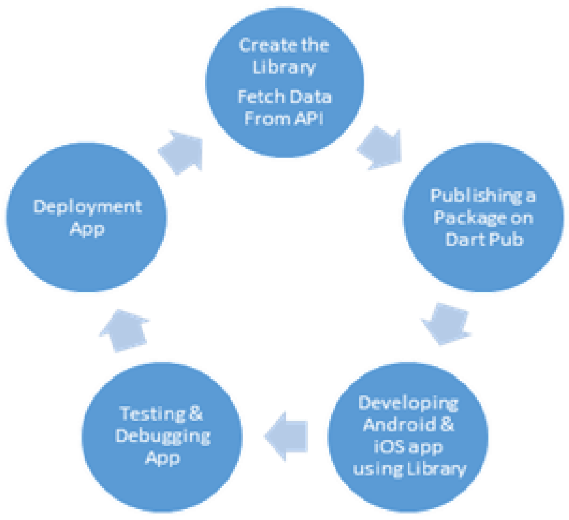

Create the Library
Fetch Data From API
Publishing a Package on Dart Pub
Developing Android & iOS app using Library
Testing & Debugging App
Deployment App

Fig. 6. Work Organizational Chart.

To begin with, creating a library to fetch data from the internet is necessary for most apps. Flutter provides tools, such as the http package, by sending dynamic headers, parameters, requests and responses in a custom and secured way, Retrofit is the best way. Creating a library and sharing it with other developers is the main idea of this work. As soon as the programmer publishes the package, users can depend on it.

Additionally, in flutter, widgets have a different lifespan; they are immutable and exist only until they need to be changed. Whenever widgets or their states change, Flutter’s framework creates a new tree of widget instances. In comparison, an Android view is drawn once and does not redraw until invalidate is called. On iOS, most of what programmers create in the UI is done using view objects. In comparison, an iOS view is not recreated when it changes, but rather it’s a mutable entity that is drawn once and doesn’t redraw until it is invalidated.

Furthermore, Flutter takes a different approach to avoiding performance problems caused by the need for a bridge by using a compiled programming language, namely Dart, Dart is compiled into native code for multiple platforms. In Flutter, almost everything is a widget. A widget is a way to declare and build the user interface to help build that look-alike native platform for Android and iOS, and all that Flutter requires of the platform is a canvas on which to render the widgets so they can appear on the device screen, and access to events and services. As can be seen in Fig. 7, moving the widgets and the renderer into the app does not affect the size of the app, which is similar to minimal apps built with comparable tools.

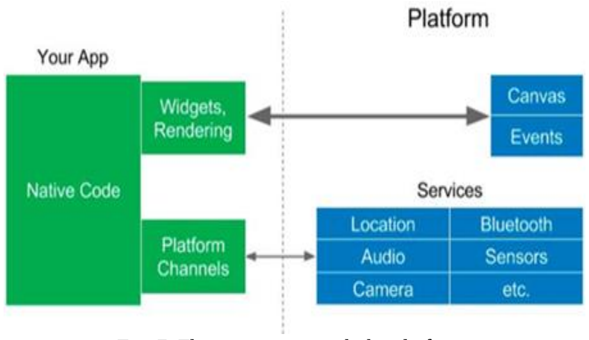


Fig. 7. Flutter interacts with the platform.

IV. Developed System Assessments

Flutter contains networking and JSON serialization for performing basic network tasks, but is pretty daunting to use when handling some advanced features. By comparison, Dio provides an intuitive API for performing advanced network tasks with ease. It implements features like interceptors and default options and is readable. Dio gives simplicity. It has a very intuitive programming API. However, making large Flutter applications can be a pain if the programmer does not implement good design patterns. One of these patterns is the Business Logic Component pattern to solve this problem. In simple terms, Bloc does two things: Connect the source of data to the UI. Update the UI when the state changes. To call Rest API’s by sending dynamic headers, parameters, request and response in a custom and secured way, "Retrofit" is the best way. Due to this serialization approach, Retrofit automatically converts the JSON response into a Dart object. It is better for large projects as the programmer does not need to hand write boilerplate code. Keep in mind that publishing is forever. Once the package is published, users can depend on it. Removing the package would break their dependencies. The programmer can always upload new versions of his package, but the old ones will continue to be available for users that aren’t ready to upgrade yet. As a result, the benefits of these tools are significantly reduced development time, have a simpler codebase, and increase the productivity and efficiency of the app.

V. Conclusion

Flutter networking using these tools feels like a breeze, and it gracefully handles many edge cases. Dio makes it easier to handle multiple simultaneous network requests, all with the safety of an advanced error handling technique. It also allows programmers to avoid boilerplate code. Flutter's retrofit API call allows programmers to call APIs while writing very few lines of code. Once a programmer has implemented a package, he can publish it on the official package repository so that other developers can easily use it. The programmer can always upload new versions of his package, but the old ones will continue to be available for users that are not ready to upgrade yet.

Conflict of Interest

Authors declare that they do not have any conflict of interest.

REFERENCES

[1] Tyagi P. *Pragmatic Flutter: Building Cross-Platform Mobile Apps for Android, iOS, Web, & Desktop*. 1st ed. Boca Raton: CRC Press; 13 August 2021.
[2] Hacernoon.com. what's Revolutionary about Flutter [Internet]. 2017. Available https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514.
[3] Hoang Ly. State Management Analyses of the Flutter Application. BSc. Thesis. Metropolia University of Applied Sciences; 2019.
[4] Fayzullaev J. Native-like Cross-Platform Mobile Development Multi-OS Engine & Kotlin Native vs Flutter. BSc. Thesis. South Eastern Finland University of Applied Sciences; 2018.
[5] Fentaw AE. Cross platform mobile application development: a comparison study of React Native Vs Flutter. MSc. Thesis. University of Jyvaskyla; 2020.
[6] Kuitunen M. CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT WITH REACT NATIVE. BSc. Thesis. Tampere University of Technology; 2018.
[7] Docs.flutter.dev. Developing packages & plugins. [Internet]. Available https://docs.flutter.dev/development/packages-and-plugins/developing-packages.
[8] CompanionLink Blog. The Benefits of Using APIs in Mobile App Development [Internet]. 2021. Available from: https://www.companionlink.com/blog/2021/02/the-benefits-of-using-apis-in-mobile-app-development/.
[9] Mamoun R, Nasor M, Abulikailik SH. Design and Development of Mobile Healthcare Application Prototype Using Flutter. In2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE) 2021 Feb (pp. 1-6). IEEE. Doi:10.1109/ICCCEEE49695.2021.9429595.
[10] Shah K, Sinha H, Mishra P. Analysis of cross-platform mobile app development tools. In2019 IEEE 5th International Conference for Convergence in Technology (I2CT) 2019 Mar 29 (pp. 1-7). IEEE. Doi:10.1109/I2CT45611.2019.9033872.
[11] Flutter's channels - dev. (2020, 12 08). Retrieved from Flutter build release channels: https://github.com/flutter/flutter/wiki/Flutter-build-release-channels#dev.
[12] Flutter Dev. (2020, 12 09). Review Xcode project settings. Retrieved from flutter.dev: https://flutter.dev/docs/deployment/ios#review-xcode-project-settings.
[13] Flutter Dev. (2020, 12 09). Create a keystore. Retrieved from flutter.dev:https://flutter.dev/docs/deployment/android#create-a-keystore.
[14] Flutter Dev. (2020, 12 09). Build and release an iOS app. Retrieved from flutter.dev: https://flutter.dev/docs/deployment/ios.
[15] Flutter Community. (2020, 12 07). flutter_launcher_icons. Retrieved from pub.dev: https://pub.dev/packages/flutter_launcher_icons.

Dilkhaz Y. Mohammed was born in Iraq and educated in both Iraq and Turkey. He has a BSc from the University of Duhok (2004). In 2019, he achieved a Master's Degree in Software Engineering at FIRAT University, Turkey. He is working as an official at the Duhok Polytechnic University scientific research center.

Siddeeq Y. Ameen received BSc in Electrical and Electronics Engineering in 1983 from University of Technology, Baghdad. Next, he was awarded the MSc and Ph.D. degree from Loughborough University, UK, respectively in 1986 and 1990 in the field of Digital Communication Systems and Data Communication. From 1990-2006, Professor Siddeeq worked with the University of Technology in Baghdad with participation in most of Baghdad's universities. From Feb. 2006 to July 2011, he was a Dean of Engineering College at the Gulf University in Bahrain. From Oct. 2011-Sep. 2015 he joined the University of Mosul, College of Electronic Engineering as a Professor of Data Communication and next Dean of Research and Graduate Studies at Applied Science University, Bahrain till Sep. 2017. Presently, he is a quality assurance advisor at Duhok Polytechnic University, Duhok, Iraq. Through his academic life, he published over 100 papers and a patent in the field of data communication, computer. Networking and information security and supervised over 100 Ph.D. and MSc research students. He won the first- and second-best research in Information Security by the Arab Universities Association in 2003.