Developing Cross-Platform Library Using Intel Multi-OS Engine

Dilkhaz Y. Mohammed and Peter A. Cooper

***Abstract* — Mobile application development services have reached a higher level with APIs. Developers create and develop applications for mobile devices, and they often rely on APIs for connectivity. An API is the functions and methods in a library that a programmer can call to ask it to do things for you; it's the interface to the library. A library is a set of classes that a programmer can use to solve a certain problem, but it doesn't change your code at a structural or architectural level. 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 his efforts. In this paper, the programmer uses Java Object-Oriented Programming to provide a way to share code across platforms and gives the possibility to develop native cross-platform mobile applications. The purpose of this work is to create a taxi service library for developers using both Android and iOS using Java programming with the help of Intel's Multi-OS Engine Framework, Retrofit, and GSON utilities, which were also used in this project. In developing a Java open-source project, the common conclusion the programmer always ends up with is to share the produced outcomes with the developer community, which should be the least objective in the Java world.**

***Keywords* — Android Development, Cross-Platform Library, IOS Development, Java, Multi-OS Engine.**

I. Introduction

Libraries are a great way to create modular code that can be easily shared. Libraries are created and distributed as packages. In Java applications, once such a library has been created, managing, and deploying it is very convenient. To share code across platforms using Java, write platform-independent code and share it between Android and iOS with the Multi-OS Engine framework, proposed by MIGERAN and developed by Intel. The Multi-OS Engine is an open source cross-platform framework that enables the programmer to develop native mobile apps with only Java expertise, without sacrificing native look-and-feel performance. The programmer can reuse as much common Java code as feasible while also add platform-specific UI code for each platform [1].

Developing a library and sharing it with the world later on so that others can make use of it in their projects is a good idea. The programmer just needs to make sure to only use APIs that are available on both Android and iOS. The programmer must create an interface to access platform-specific functionality from the library and then create Android and iOS implementations of that interface in their projects, thereby speeding development [2], [3].

This research is mainly aimed at creating a taxi service library that other developers might use in their Android and iOS apps. Retrofit is responsible for handling responsive web services. This powerful library makes it relatively easy to fetch and upload JSON via a REST-based web service. Retrofit uses the OKHTTP for making HTTP requests.

II. Multi-OS Engine Framework

Multi-OS Engine Framework allows developers to use Java to create native mobile apps for Apple iOS and Android devices that have the same appearance, feel, and performance as native apps. The most intriguing aspect is that the developed programs are compiled to native code and make use of Java common code. It saves a lot of time for Android developers who wish to create an iOS app as well. Its runtime is based on Android's current ART, which is the runtime component that executes Java programs on Android. ART offers a range of characteristics that ensure that apps on iOS devices run at their best. These following components are included in a compiled Multi-OS Engine program (Fig. 1), such as NatJGen, which allows you to generate Java code.

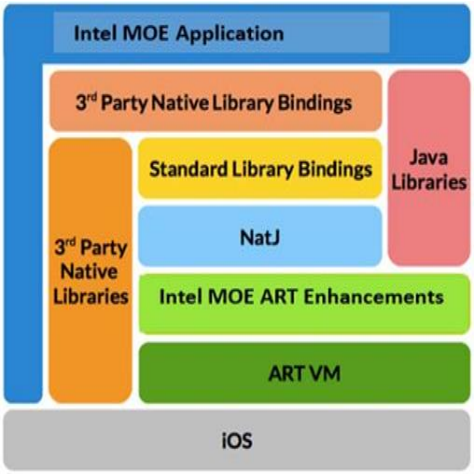

Fig. 1. Multi-OS Engine app components.

III. Java Object Oriented Programming

Every application in the Multi-OS Engine is written in

Submitted on February 24, 2022.
Published on March 17, 2022.
Dilkhaz Y. Mohammed, Software Engineering, Firat University, Elazig, Turkey; Scientific Research Center, Duhok Polytechnic University, Duhok, Iraq.
(e-mail: Dilkhaz.mohammed@dpu.edu.krd).

Peter A. Cooper, Computer Science, Sam Houston State University, Texas, USA.
(e-mail: cooper@shsu.edu).

Java. Java is a general purpose language that derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. Java is the language of choice for creating Android apps. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM), regardless of computer architecture.

IV. Android Studio IDE

Android Studio is Google's official integrated development environment; as the name implies, its primary use is Android app development. It is based on Jet Brains' IntelliJ IDEA IDE. Even though it's designed for Android development, it's also a good fit for Multi-OS Engine because it's written in Java and built with the Gradle system. Furthermore, it allows Android developers to write both Android and iOS code in a single IDE. Only an additional MOE plugin is required to enable Multi-OS Engine.

V. Proposed Methodology

A. Overview

Almost every mobile app nowadays connects to the internet to receive and send data. A programmer should learn how to use RESTful web services because their proper implementation is essential when developing modern apps. Retrofit is a Java REST client. Once such a library has been created, it is very easy to manage and deploy. However, to share code across platforms using Java, to write platform-independent code and share it between Android and iOS with a Multi-OS Engine, Multi-OS Engine apps should have three modules. The views of each platform should be separately placed in two of these modules, and the third module is used for including the shared views of the platforms. Both views of Android and iOS do not have direct access to each other. Essentially, these views should include only methods that are in control of the UI, and minimizing these methods is highly recommended. To write clean code and produce a separate view, it is recommended to use the Model-View-Presenter architecture see Fig. 3.

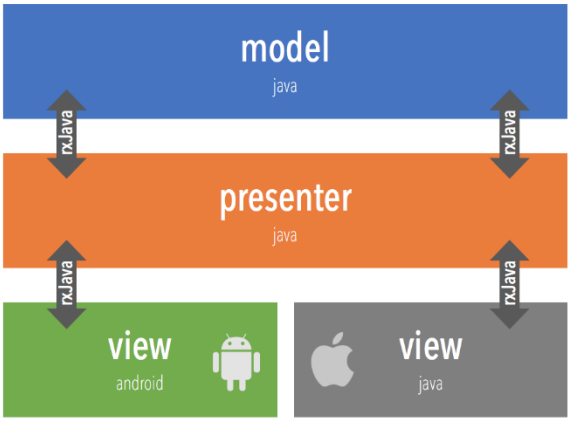

Fig. 2. The architecture of a Multi-OS Engine app.

In Android Studio, an application begins as an Android project. The Multi-OS Engine configures the project to build and run as an iOS app on the iOS simulator, which can be accessed through Android Studio or a real device. Whenever an iOS app is launched, the ART VM is started, and the precompiled code is executed. It is completely integrated with Android Studio, is hosted on Mac OS or Windows, and contains all of the development tools required to create an iOS app; the development process is illustrated in Fig. 3.

Fig. 3. Development process with multi-OS engine.

To begin with, the Android code is simple; it simply obtains data and binds it to the UI. However, it truly requests data asynchronously from an API via common module methods and then displays it in the native UI. The second module contains all of the shared logic for the application. It is the main part of the project and contains all of the business logic. This is the code that communicates with the REST API. The final step in project setup is to create an iOS module that contains all of the iOS code written in Java as well as the entire Xcode project. The Android Studio Multi-OS Engine plugin is required to bind the resulting UI to the rest of the Java-coded app. However, the Multi-OS Engine and Java communicate with the platform in a similar native tools manner, the difference being only in the additional process of transferring source code to native one, and everything else is the same. That's why it's still required to have different code for handling UI see Fig. 4.

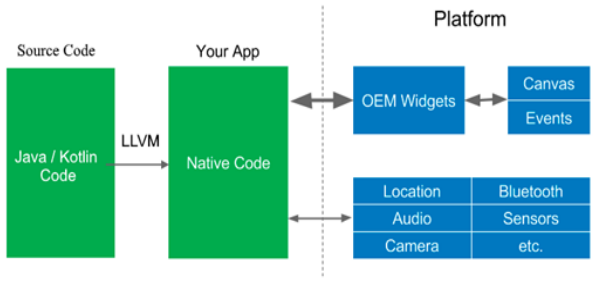

Fig. 4. Multi-OS Engine interact with the platform.

Furthermore, it is critical to understand that Multi-OS Engine and Java are not intended to replace existing native development tools, but rather to extend them and enable better code sharing and reuse. It is still necessary to use these tools to develop apps, and native development for both platforms is now available. These tools simply allow for faster prototyping, which results in improved app behavior. For example, if business logic changes, it is only changed once and is reflected in both apps. Because both platforms should behave similarly, testing becomes easier and faster. As a result, logical bugs will not be platform-specific, and finding and fixing them on one platform will be reflected on another.

B. Proposed System

The app is available on an Android or iOS-based mobile phone.

C. Workflow Diagram

The programmer presents a simplified diagram of the state paths of an Android app. To request a ride for a passenger or to view a passenger's trip history, the passenger needs to authenticate with Uber and authorize the app (have the rider sign in and authorize the app to access their Uber account) after authorization is completed. Users can make requests to the Uber web service.

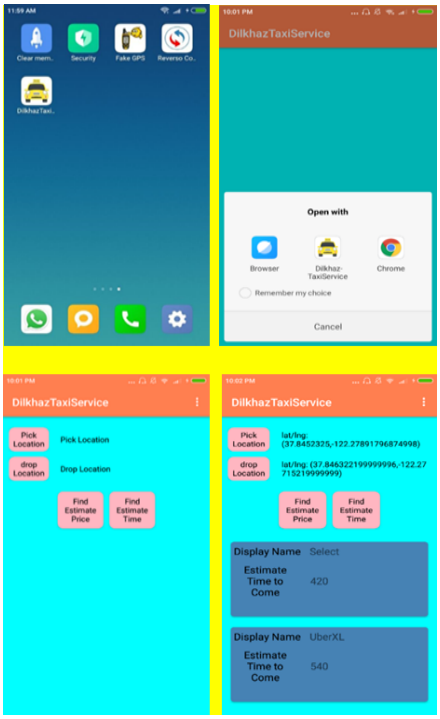

Fig. 5. Taxi service app interface.

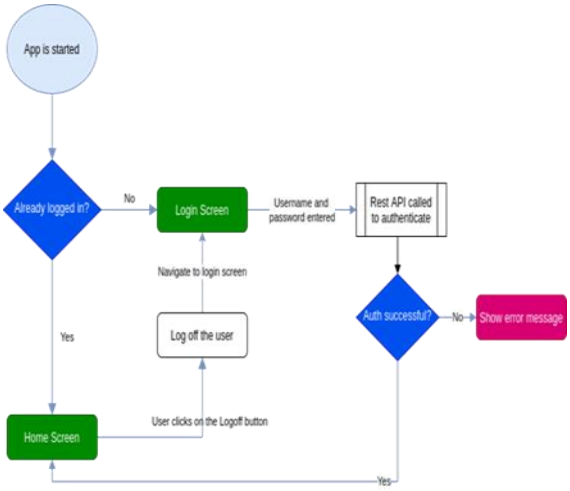

Fig. 6. Workflow Diagram.

VI. Conclusion

Getting data from a RESTful API using these tools feels like a breeze, and it gracefully handles many edge cases. Retrofit 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. Java'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 come by and upload new versions of his package, but the old ones will remain available for users who are not yet due for an upgrade.

Acknowledgment

I would want to express my heartfelt gratitude to everyone who has helped and encouraged me during the production of this report; I couldn't have done it without their support and direction. Prof. Dr. Peter Cooper, my supervisor, is to be commended for giving me the opportunity to work on this research study under his guidance.

Conflict of Interest

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

References

[1] Milaqi I. Cross platform library (Doctoral dissertation, Saxion).
[2] Salza P, Palomba F, Di Nucci D, De Lucia A, Ferrucci F. Third-party libraries in mobile apps. *Empirical Software Engineering*, 2020 May; 25(3):2341-77.
[3] Salza P, Palomba F, Di Nucci D, D'Uva C, De Lucia A, Ferrucci F. Do developers update third-party libraries in mobile apps? In *Proceedings of the 26th Conference on Program Comprehension*, 2018, May 28 (pp. 255-265).
[4] Shah K, Sinha H, Mishra P. Analysis of cross-platform mobile app development tools. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, 2019, Mar 29 (pp. 1-7). IEEE.
[5] You D, Hu M. A Comparative Study of Cross-platform Mobile Application Development. Wellington, New Zealand.66.
[6] Davis AL. Modern programming made easy: using Java, Scala, Groovy, and JavaScript. Apress; 2020 Jan 17.
[7] Zhan X, Fan L, Chen S, Wu F, Liu T, Luo X, Liu Y. At hunter: Reliable version detection of third-party libraries for vulnerability identification in android applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, May 22 (pp. 1695-1707). IEEE.
[8] Zhan X, Fan L, Liu T, Chen S, Li L, Wang H, Xu Y, and Luo X, Liu Y. Automated third-party library detection for android applications: Are we there yet? In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, Sep 21 (pp. 919-930). IEEE.

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 Degree in Software Engineering at the University of FIRAT University.

Peter A. Cooper was born in the United Kingdom and educated in both the UK and the US, Dr. Cooper has an M.A. and Ph.D. from the University of Missouri-Columbia (1993) and has taught 13 years in secondary education and 18 years in higher education. Dr. Cooper teaches Networking, Network Security, Programming in Java and C++.