Reflections on the creation of my mobile application

Andrew Goodwin

WGU

Reflections on the creation of my mobile application

Western Governors University’s (WGU) Bachelor of Science in Software Development program requires completion of the creation of a mobile application and a summary document which addresses “mobile application development through the context of the architecture involved, including hardware and software capabilities and limitations,” identifies “the version of the operating system” my “application was developed under and is compatible with,” describes challenges I experienced during development and how I overcame them, what I “would do differently” if I restarted the project,” and discuss “how emulators are used and the pros and cons of using an emulator vs. using a development device” (ABM1 – ABM1 Task 1 Mobile Application Development. (n.d.)).

# Mobile application development through the context of the architecture involved

Development for some other platforms requires nothing more than a word processor, however development for Android requires special considerations be given to hardware and software requirements.

## Software

Android development, while not strictly requiring the use of Google’s integrated development environment (IDE) Android Studio, is improved by its use. Android Studio provides ready access to plugins and libraries specifically designed for Android development. Similarly, Android Studio provides debugging tools specifically for android development as opposed to just the underlying programming language. For example, at one point during development I included some layout properties and Android Studio informed me that those layout properties were not supported for my minimum software development kit (SDK) and suggested others to use.

In addition, Android Studio has an integrated emulator which allows application testing on many different devices. Emulators and development devices will be discussed in greater detail later.

Android Studio is compatible with recent operating systems, including Windows 10, the operating system I utilized during production. Windows 10 was not my first choice as it is somewhat resource intensive for an OS. Android Studio is also notorious for being resource intensive.

I initially attempted to use Ubuntu Linux. Android Studio installed and ran well but failed when I attempted to utilize the emulator due to hardware incompatibility. This issue and other hardware considerations are discussed next.

**Hardware**

I knew from a previous android development course that Android Studio was resource intensive and that my current development device, and old Dell laptop with an i3 processor and 4 GB of RAM would not be adequate. In preparation for the course I purchased a desktop computer with an i7 processor that supported virtualization (necessary for running the Android emulator) and 8 GB of RAM. This computer was adequate to the task, however, it had an Nvidia GPU. Unfortunately, Linux OS requires special configuration to utilize Nvidia devices.

After some time engaged in configuration and troubleshooting I began to suspect that any time I saved by using Linux instead of Windows 10 as my development OS would be less than the amount of time spent on configuration. While I could have purchased a GPU from another manufacturer, in every other respect my existing Nvidia card was adequate for my needs. Ultimately, I could not justify the expense of a new GPU for what might be only a marginal improvement in development pace.

**Android Architecture**

Android application architecture is flexible. I utilized Model-View-ViewModel methodology (MVVM). Under this approach persistent data concerns (Model) are separated from temporary data concerns (ViewModel) and the user interface (View). My previous course in Android programming did not include separation of concerns so, while the concept was not new to me, the execution involved a learning curve. I also utilized the Room library to manage my database which saved a great deal of code to manage operations.

**Version**

Android applications are developed for a minimum SDK and target SDK. Put simply the minimum SDK is the oldest version of the Android OS with which the application is compatible. The target SDK is the most recent version of the Android OS for which the application is designed. The minimum SDK for my application is 14 which means that it is compatible with Android 4.0 (Ice Cream Sandwich). The target SDK (and the version it was developed under) is 29, compatible with Android 9 (Q).

**Challenges**

**The challenge**

The greatest technical challenge I experienced was learning how to use MVVM architecture with LiveData. As previously stated, MVVM architecture is a methodology emphasizing separation of concerns, that is, the database, business logic, and user interface are all kept separate.

LiveData objects are a way of streamlining database operations to maintain data persistence throughout the activity lifecycle. This is necessary because the activity lifecycle does not necessarily meet user intuitive expectations of when the activity would be destroyed and recreated, and data lost and restored. As an example, an ArrayList might be loaded from the database at the creation of the activity. Then the user makes changes which are saved to the arraylist but not the database. Then the user switches from portrait to landscape mode at which point the activity reloads the ArrayList from the database and all user changes are lost. By placing the ArrayList inside a LiveData object, as soon as changes are made to the ArrayList they are reflected in the database and reloaded back to the UI.

I utilized LiveData objects for ArrayLists representing each table in my database. However, while the ArrayLists were in the LiveData object they were not accessible for operations except in specific locations, such as an OnChanged method in the ViewModel. While I am sure there are ways to access the ArrayList, I was not able to identify them in the course of my work.

In addition, my understanding of the MVVM model limited my ability to access the database to working through the ViewModel and the RecyclerView adapter. While I could have worked directly between the activity and the database, this violated the principle of separation of concerns and meant that I would lose the benefit of using a LiveData object in the first place. This left me with limited access to my database for operations. Thus, where normally I would write separate queries to get specific information from the database, I was limited to working from the one ArrayList contained in the LiveData object.

**Overcoming the challenge**

Since I could not find the right way to access my data, I had to improvise. The ArrayLists were accessible via the RecyclerView adapters used to display the ArrayLists themselves, so, where normally I would have used a query from the related class, I used the adapter to access that data.

To give an example, one the requirements of the project was that it contain progress tracking elements; yet, there were few actual progress tracking elements in the project as described. I utilized a progress bar in the main activity to represent student progress. The main activity, however, had no access to the database. In order to access the data I needed, the start and end dates of the first and last terms, I instantiated the terms RecyclerView adapter and the terms ViewModel. Then I accessed the terms adapter from within the onChanged method of the terms ViewModel. In essence, I utilized the code needed to get the terms list from the database and load it to the RecyclerView without actually loading it to the RecyclerView.

**What I would do differently**

Were I to do this project over again I would have invested in a recent book on android development so I could learn how to properly utilize the MVVM model. I used a lot of tutorials that told me what to do, but not why. Many others utilized code that the instructor stated would not be suitable for actual development, but were adequate for the tutorial. Unfortunately, they never explained why it should not be used for development or, more importantly, the proper way to accomplish said task.

I would have liked a book that demonstrated different ways the Room library could be used and exactly how the Data Access Objects, data repositories, ViewModels, and adapters interacted, and the correct way to structure them. In particular, while I understand that responses to queries coming from the database are supposed to be delivered by way of the data repositories and viewmodels, I’m very unclear on if updates to the database are supposed to travel by way of the same route or if they are supposed to be updated by direct calls to the data access objects.

Ultimately, I would have liked to have had clean examples of how an experienced developer would code the model and ViewModel layers, and a clear explanation of the rationale behind each decision. If I had something like that, and I were to do the project over again, I would like my code to meet professional standards, and I would like to know how and why it met those standards.

**Emulators vs development devices**

An emulator is a program that emulates an operating system, hence the name. Emulators run on top of other OS by way of virtualization. In essence, a portion of system resources are devoted to a separate operating system that is accessible to, but not a part of the underlying operating system. Android Studio has its own emulator that allows emulation of many different versions of Android OS on many different devices. This is extremely useful for testing an application on multiple devices including phones and tablets.

An alternative to emulators are development devices. Development devices are actual physical devices upon which the application is installed and run. Using development devices provides the advantage of being able to see how an application runs on the physical device. For example, while any Android phone with an adequate operating system may be able to run a compatible application, the processer on a particular device may be inadequate to run an application at a reasonable speed, or the graphics processor may not be able to display the application. The disadvantage to using development devices is that a physical device must actually be purchased, maintained and stored. This can represent a significant investment in devices that will likely be obsolete within 2-3 years.

Emulators, on the other hand, provide the ability to test an application on many different types of devices without needing those devices on hand. Emulators are by far much more affordable than an investment in dozens of devices. That having been said, Emulators also demand significant system resources on the host computer. Those resources are then unavailable for other applications. Also, the underlying hardware of the emulated device is the computer running the emulator and the resources designated as available to the emulator. Thus, it is possible that the emulator may represent an application as working better or worse on an emulated device than would the actual physical device. This is particularly true if the computer running the emulator has significantly different system resources than the physical device being emulated. Ideally, developers have a variety of physical and emulated devices available for testing.

References

ABM1 – ABM1 Task 1 Mobile Application Development. (n.d.). Retrieved 8/18/19 from <https://tasks.wgu.edu/student/000772620/course/13980005/task/609/overview>