<Hongda Lin, Qin Sun>
CSE 5236
Professor Champion
11/29/2022

Buckeye Schedule Final Report

Section 1 - Introduction

According to statistics[9], most universities have two semesters per year and students are expected to obtain their degree in four years, which means that there will be 15 credit hours of workload per semester. Breaking it down further, courses in most universities with a semester system are worth three credit hours. On average, university students will take 5 courses every semester. For each class, the teacher might hand out assignments and students are fully responsible to keep track of the due dates of multiple assignments from different classes in order to finish them in time. With the increase of the number of tasks, it is more likely for students to miss the due dates or mix up the due dates of tasks from different courses. Therefore, our team created a mobile application called Buckeye Schedule, which is a schedule application that assists students to record the details of assignments given out in each class.

Buckeye Schedule provides students with a convenient tool which could be used to manage and schedule the progress of course assignments. Our application comprises three main modules. Firstly, students register/login the application via the login/register module. Their account information will be stored in Cloud Firestore[2] and all additional tasks will be associated with the corresponding account so that the tasks created by that user could be retrieved from the database and displayed in the dashboard module when the user logs in the application next time. In addition to the register/login page, the register/login module also contains a settings page which mainly reuses the register page layout, and users could modify their account information on this setting page. After the user logs into the application, he will be redirected to the dashboard page, which displays the skeleton of all the tasks already created by the current user. There is a search bar on the top of the dashboard page which could be used to filter which tasks to display according to the keyword typed in. By clicking on the skeleton of tasks already created or clicking on the add-new-task button, the user will be redirected to the third module, which is the task editing module. On the task editing page, the user could edit the title, description, type of the task or even take a photo and attach it to the task. After the editing, the application will create a new task if the task does not exist or save the modified information if the task already exists. On the due date of the task, the application will prompt a notification to remind the user of the upcoming event. With all these features, Buckeye Schedule helps students to manage their assignments and avoid missing or mixing up dues.

Section 2 - Design Process

The application is meant to be a schedule app that helps college students keep schedules more efficiently. Students can create schedules not only with plain text, but also with photos, urls, colors, and reminders. Once the application's idea is clear, we aim for an object-oriented design process to translate basic ideas into more concrete work products in Android Studio[1]. The first step of the design process is to define the nouns and verbs for our app that could be translated into classes or other structures. Nouns like user, schedule (title, subtitle, description), miscellaneous (color, image, url, reminder) are the most important things that we think would make up the app. Verbs are also important when considering different functionalities of the app. Some verbs that we think that are most related to our app are sign in, sign up, sign out, view, add, delete, update, create, search, select, cancel, etc. We combined these nouns and verbs and came up with the first UML class diagram in checkpoint2 to help us clear the responsibilities of different classes and architect the needed database schema. Below is the UML diagram of our app.

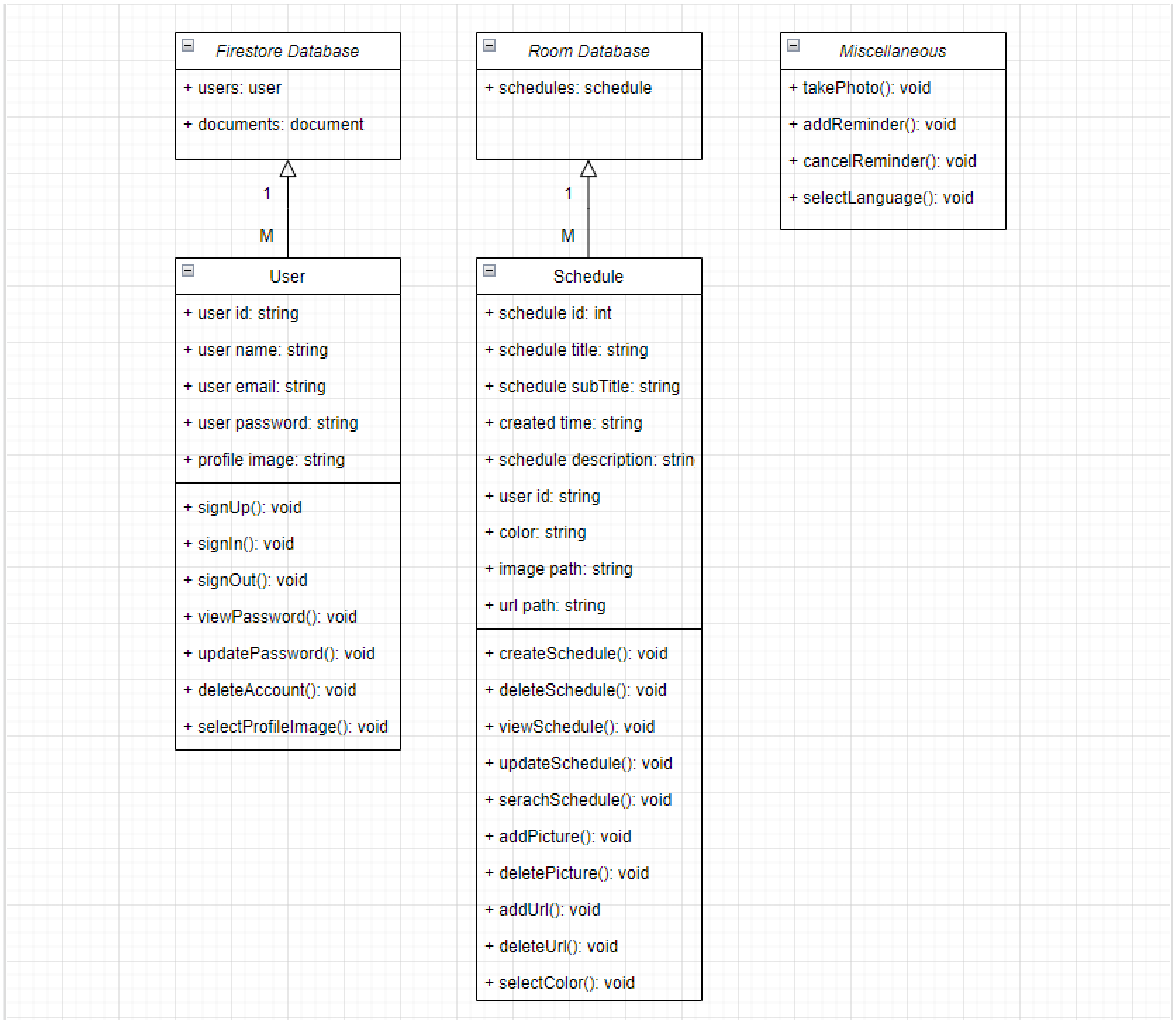

Figure 1: UML Class Diagram

After identifying the classes and responsibilities, a use case diagram would be helpful for visualizing what the app could do. For the basic functionalities, the student should be able to sign up a new account and use it to sign in the app. The student will then be directed to the schedule page and start creating schedules. By clicking the create schedule button, the student is directed to the create schedule page, where the student will be prompted to enter the schedule title, subtitle and description. Finally, the student clicks the confirm button and the schedule will be created and displayed in the schedule page. There is also an account page where the students could view their account and sign out. In order to make our app stand out from other scheduling apps in the market, we target schedule as the core of our app and start imaging some advanced functionalities that could make a better schedule experience. The first thing that comes to our mind is images, since students these days like to combine images with text for better memorization. If the students are allowed to add images when creating schedules, it would make scheduling more effective. Url[3] is also something that we think the student might find useful. The student could be directed from our schedule app to any websites they schedule to visit. What's more, the ability to add reminders is what makes a scheduling app useful. The student could pick the date and time for the app to remind them that there is a schedule to complete. With so many things that could aid the scheduling experience, we decided to put them in miscellaneous. Apart from these, some quick actions for creating schedules are also desirable. One of the quick actions that we believe will make our app stand out is the photo taking quick action. With this, the student could use the device camera to take a picture and the app generates a schedule using the picture just taken. Below is the use case diagram of our app.

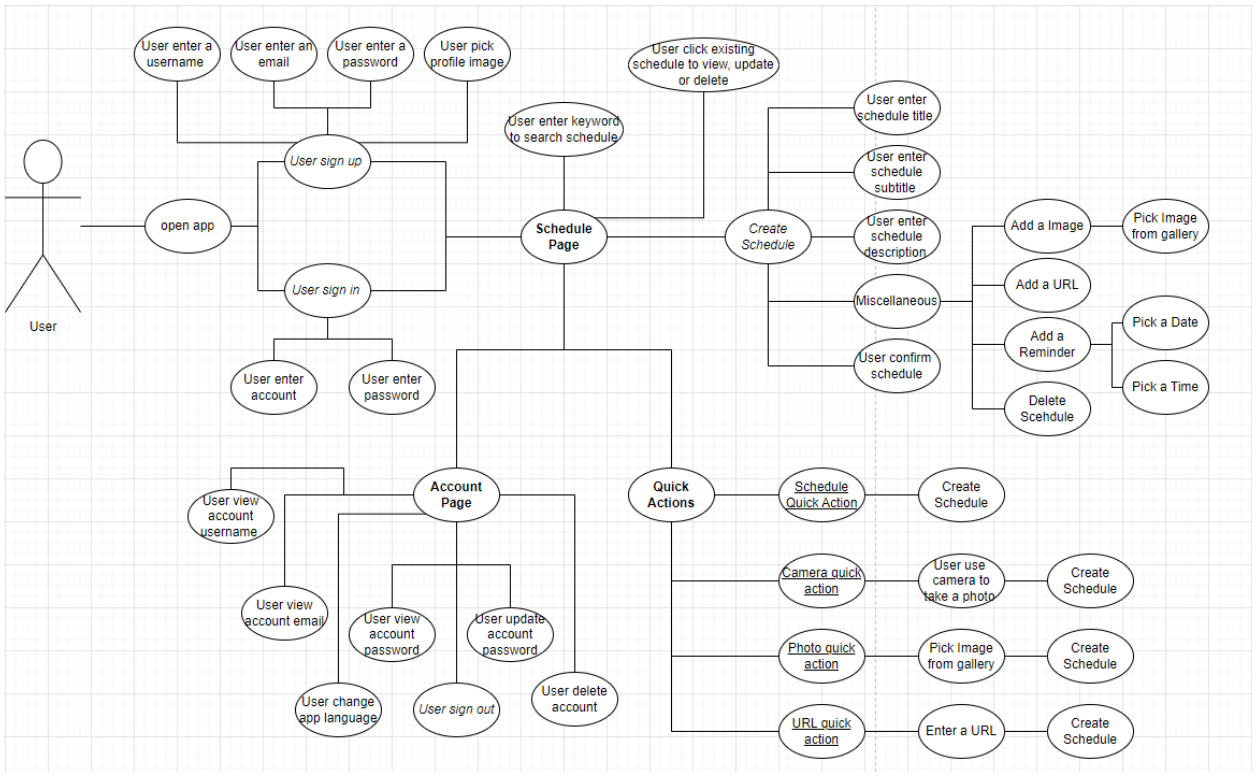

Figure 2: Use Case Diagram

With a solid understanding of the classes and use cases, it's time to start thinking about the potential services that could be employed in our app. Our initial thoughts about the database schema was to have two separate databases, one for user and another one for schedule (refer to UML diagram). For the user database, we decided to use an external service, since it's safer to store the user data on cloud than local. Firestore[2] is the database we decided to use. It's a cloud-hosted NoSQL database with fast query and more intuitive (JSON liked) data model. The data stored in the Firestore database consists of user id, user name, user email (foreign key), user password, and profile image (decoded image bitmap). For the schedule database, we decided to use a local Room[4] database combined with a LiveData[5] observer, in short, MVVM[6] design pattern. The reason why we use MVVM is because it separates the view from business logic and enhances the reusability of code. The properties that we think a schedule should have are schedule id (foreign key), schedule title, schedule subtitle, created datetime, schedule description, user id (dependency injection), color, image path, and url path. Since each schedule has a user id, for different users who use this app, we could query the corresponding schedules from the room database. So different users won't see each other's schedule when using the same device, which introduces the ViewModel Factory[7]. When considering CRUD[8] operations, besides sign up (create), users could view (read) their account information (email, username, password) in the account page. Also, we want them to be able to update the password and delete the account. Similarly, users could create, read, update, and delete schedules.

It should be mentioned here that we are constantly updating the UML class diagram and the user cases diagram to help us stay in the right direction. We will explain them in detail later on in the discussion of going from design to implementation.

<u>**Section 3 - Translating Design to Implementation**</u>

After the design process is completed, we begin working on the implementation of Buckeye Schedule using Android Studio[1]. We decided to use Java[10] to write all the functionalities. The implementation process starts with finishing the basic UI then working on different business logics. It's the most efficient way since it eliminates the unnecessaries and helps to visualize the needed implementations. Since the project is divided based on checkpoints, we will introduce our implementation process in this order.

First, in checkpoint 3, we implemented the UI for sign in page and sign up page based on the screen mockup from checkpoint 2. We learned to use different ViewGroups such as ScrollerView and RelativeLayout with View[13] components such as TextView, EditText and Button. Along with the UI implementation, we discovered View Binding[11], an instance of a binding class containing direct references to all views that have an id in the corresponding layout. This allows the implementation of business logic quicker, such as a button press event. After UI finished, we began working on integrating Firestore[2] to sign in and sign up activities. With the help of official documentation, we successfully set up the connection between our app and the firestore database. In SignUpActivity, the student username, email, password, and profile image (bitmap decoded) will be validated and collected in a HashMap object (user) when the student clicks the sign up button. Then we reference the firestore database instance and add the student object to the database collection. Whereas in the sign in SignInActivity, the entered email and password will be matched with existing users in the firestore database. And if matching is successful, the student is signed in to the app. Students now can sign up for an account or sign in the app with different accounts on the same device. Besides sign in and sign up, we also finished our splash screen using Android animation components[12] and a basic schedule page (main page). Inside the schedule page, we use a FrameLayout to hold different fragments that we want to implement. Fragments including schedule fragment, create schedule fragment, and account fragment are first introduced to the project. With most UI and activities finished, we start practicing different tools Android Studio provides. For example, debugging, profiling, and logging. We utilized the Logcat to display the activity and fragment lifecycle and Profiler to view the app performance. Things went well and we became familiar with Android development.

Then, in checkpoint 4, we improved the UI from the previous checkpoint and implemented the account page. The main focus is on data persistence. In our account page, the student could view its account username, email, password and profile image. The student could also update the account password or delete the account. The implementation is simple. We keep a document reference of the current user from Firestore database at the MainActivity and perform the CRUD[8] operations in the AccountFragment by referencing it. We also implemented the sign out button in the account page so the user could sign our account and sign in with a different account. Apart from implementing different firestore functionalities, we also look into the Room database and MVVM design pattern for checkpoint 5. We first define the Schedule Entity[14]. Each schedule should have a schedule id, schedule title, schedule subtitle, schedule datetime, schedule description, user id, color, image path, and url path. Each schedule property also has its getter and setter method. After this, we define abstract methods in Schedule DAO[15] to access (CRUD) our Schedule Database, which is also implemented for building and referencing the real Room[16] database. With the Entity, DAO[15], and Database finished, we move on to the next checkpoint.

Checkpoint 5 is where different gaps start to appear. As usual, we start with the implementation of several UIs. The schedule page consists of a button to create a schedule, a search bar for searching schedule, a quick action bar for different quick actions and a RecyclerView[17] for listing all the schedules user created. The create schedule page allows the student to enter schedule title, schedule subtitle and schedule description. The miscellaneous is also a part in the create schedule page, where the user could add color, image, url, and reminder to the schedule. Creating these UIs is a massive work in designing layouts and drawables. After UI finished, we continued working on implementing the database for schedules. The first thing we did is to implement the Schedule Repository[18]. The repository is important since it isolates data sources and provides a clean API for data access to the rest of the app. What's more, the repository is the place where we define the LiveData[19] (the list of schedules) by fetching all the schedules from the database using the query method in Schedule DAO. The Schedule ViewModel[20] is the last thing to implement for completing the Schedule Database. The implementation of Schedule ViewModel is very similar to implementing the Schedule Repository. We just need to create a repository instance and call the methods in it. Though with everything implemented, the hard part is to combine them together and put them into use. To utilize the advantages of LiveData and RecyclerView, we realize that we need to create a Schedule Adapter[21] to help do the bridging between UI component and data source that helps to fill data in UI component. The data source in this case is the LiveData, which is the list of schedules inside the Schedule Database. What we did is register an observer in the schedule fragment for the changes in the Schedule ViewModel. When the observer notices that the LiveData, which is the list of schedules ViewModel returned, has changed, it will notify the View to make updates, which update the data for RecyclerView. This part also takes a massive part to implement since the implementation of MVVM design pattern is more abstract and difficult than typical MVC or MVP. The schedule database is finished, but several problems appear. The first problem is that in order to implement methods that involve internal services like image picking, camera accessing or picture saving, we need to let the app launch other activities (ex: device gallery) and while asking permissions from the user. However, several methods involved with it are deprecated, for example, OnActivityResult. It takes us time to come up with the alternative solution using ActivityResultLauncher to registerForActivityResult, like RequestPermission and StartActivityForResult. The second problem we encountered is different users who signed in the same device could view each other's schedules. Although we have the user id stored in each user, we did not add the user id to the query schedule method in DAO and we did not have the user id injected in the ViewModel. However, the ViewModelProviders doesn't allow us to simply pass the user id to the ViewModel. What we figured out is we need to add a Schedule ViewModelFactory[7] to create a ViewModel with the ability to inject the user id, which is like the constructor of the ViewModel. The last problem is the implementation of the photo taking functionality and reminder functionality. We want the photo taken to be saved in the device gallery. With the help of the official document, we successfully implemented it with getExternalStoragePublicDirectory and FileProvider. For the reminder, we want the user to be able to pick the exact date and time for the app to send an alarm to the user to notify them there is a schedule to complete. The implementation consists of a DatePickerDialog and a TimePickerDialog for datetime picking and a BroadcastReceiver combined with AlarmManager to construct a PendingIntent to let the alarm happen at the exact date time the user picked. After everything is resolved, the app is fully functionally working as we expected. We are really grateful for what we have accomplished for now.

Finally, in checkpoint 6, no more UI needs to be implemented. We put our focus on bug fixings and different NFRs. One bug that we find is that the image is rotated right 90 degrees when we use the real device for photo taking. What we did to fix this is to implement a BitmapHelper using the ExifInterface to determine the rotation of the image and modify the bitmap before using. The NFRs we implemented for this app consist of internet connection checking, OS termination warning, screen rotation resistance. We also implement the functionality for switching the app language between English and Chinese. After everything is finished, we then write several test cases (unit testing) for the email, phone, password, url, datetime validation and testing about password encryption and decryption with two different algorithms, Base64 and AESCrypt. We also work on the app performance improvement by code refractory. The app is finally ready at this point and we complete the translation from design to implementation

Section 4 - Suggested Changes and Improvements

Though we did have the unit testing finished, we still could implement some other NFRs, for example, ui testing using Espresso or accessibility support for users with disabilities. The non-functional requirements are the things that make the app stand out from others. However, due to the time limit, we are only able to finish a part of the NFR implementation, as section 3 mentione, it would be better if we could add more NFRs.

After finishing the implementation of the application, there are some design issues that we found in checkpoint 5. The biggest design issue we have is referencing the methods in MainActivity from the several fragment activities that live within it. We have many public methods in the MainActivity such as replaceFragment, setTitle, and signOut implemented in the MainActivity, and they are called in different fragments. This is not a good design since we should make the activity low coupling and high cohesion. However, on the other hand, this is the inevitable issue in Android studio when you have too many fragments. When we implement the app, our first priority is to make the app running and meet all the functional requirements we need. And we did make several big code refractory and performance optimization by removing the unnecessary initializations and separating the code group into several methods. As there are still many things to learn in Android studio, we think we could still do better in code design and improvements.

Apart from those functionalities that we have already implemented in our Buckeye Schedule application, we could further realize some more modules which can help students to get better control over their assignments. For example, we can incorporate Google Maps[22] into our application to display the location where each task is created. The location will be marked on the map, so users could easily tell different assignments apart. In this way, it’s much harder for users to mix up the due dates of different assignments. In addition, the recording function could also be added to our application as it could make it much easier for users to note down complicated descriptions.

References

[1] Android Open Source Project, Android, http://developers.android.com
[2] Firebase Cloud Firestore, https://firebase.google.com/
[3] URL - Android Developers, https://developer.android.com/reference/java/net/URL
[4] Room - Android Developers, https://developer.android.com/jetpack/androidx/releases/room
[5] LiveData overview, https://developer.android.com/topic/libraries/architecture/livedata
[6] Android MVVM Design Pattern, https://www.digitalocean.com/community/tutorials/android-mvvm-design-pattern
[7] Create ViewModels with dependencies - Android Developers, https://developer.android.com/topic/libraries/architecture/viewmodel/viewmodel-factories
[8] What is CRUD? - Codecademy, https://www.codecademy.com/article/what-is-crud
[9] Jacob Imm, "How Many Classes Should I Take A Semester?", https://www.northcentralcollege.edu/news/2020/10/01/how-many-classes-should-i-take-semester#:~:text=Since%20most%20schools%20have%20two,take%20five%20classes%20a%20semester
[10] Overview (Java Platform SE 7) - Oracle Help Center, https://docs.oracle.com/javase/7/docs/api/
[11] View Binding - Android Developers, https://developer.android.com/topic/libraries/view-binding
[12] Introduction to animations - Android Developers, https://developer.android.com/develop/ui/views/animations/overview
[13] View - Android Developers, https://developer.android.com/reference/android/view/View
[14] Entity - Android Developers, https://developer.android.com/reference/android/arch/persistence/room/Entity
[15] Dao - Android Developers, https://developer.android.com/reference/android/arch/persistence/room/Dao
[16] Save data in a local database using Room, https://developer.android.com/training/data-storage/room
[17] Android Recyclerview - Explore available libraries, https://developer.android.com/develop/ui/views/layout/recyclerview?gclid=Cj0KCQiA4aacBhCUARIsAI55maHrC3GjkTYl1k9MRDeglz5aGNuDZPDNDRKnRIMhT8geKY45MhbqiQIaAjI-EALw_wcB&gclsrc=aw.ds
[18] Repository Pattern - Android Developers, https://developer.android.com/codelabs/basic-android-kotlin-training-repository-pattern#3
[19] LiveData overview - Android Developers, https://developer.android.com/topic/libraries/architecture/livedata
[20] Android Jetpack ViewModel - Manage UI related data, https://developer.android.com/topic/libraries/architecture/viewmodel?gclid=CjwKCAiAhKycBhAQEiwAgf19ev75Truvs62v7-Brzo9MS7Boksr5OVnRsDpJkH7pDPI67DxN67vJYBoCxQoQAvD_BwE&gclsrc=aw.ds

[21] Adapter | Android Developers,
https://developer.android.com/reference/android/widget/Adapter
[22] Maps Platform - Google Developers, https://developers.google.com/maps