**A Penny for Your Thoughts Ltd.** Movie Match for Android Software Design Document

Matthew Penny

Date: 12/08/2022

1. Introduction

## Purpose

This is the Software Design Document for the capstone project of Matthew Penny, which consists of an Android application that uses an API from Movie of the Night1, a Swipe-able UI, Firebase's Real-time Database as well as it's Storage function. We will be aiming for devices running Android 7.0 and higher to be able to accommodate as many users as possible. We will use Diagrams.net as the wire frame software, as well as to create any diagrams required, including but not limited to: DFD's (Data Flow Diagrams), UML diagrams, and Use Cases. GitHub will be the VCS and there will be public access for any administrators, and instructors who require it.

## Scope

The initial scope of the project was for couples and friends who can never agree on what to watch due to a plethora of choices in the video streaming domain. To start, three services will be offered by the current product, with others being added after the timeline for the original product has passed. There will be the ability to play the movie from the application, as well as social media sharing. The project's head administrator has full access through the VCS and will have final approval of the application's viability. The modeling approach that we have decided on for the project is an Iterative model. This way, the head admin can watch the progress and make any changes or ask questions as to how the project is progressing. The project will also pass through three separate phases: initial planning (including diagrams and database mapping), coding and use case implementation) and complete documentation (including this document). As we have only the one individual working on the project, he will oversee the completion of all tasks within the timeline handed down by the college.

We mentioned above that this was the initial scope of the project. After a midterm meeting with the head administrator, we decided on a small shift away from the original scope. As of that meeting, we moved away from the original targets and broadened the scope. Also, social media sharing and the ability to open the third-party streaming sites was mentioned by the administrator and will be implemented during the project.

## Overview

We will be using this document to detail the functionalities, context, and design of the capstone project. As the sole employee of the company, Matthew Penny is the head designer, programmer, UI/UX developer and he will be referred to as "we" throughout the document. As the project manager, he will also oversee any changes to the document as well as the diagrams that can be found throughout.

1. System Overview

### 2.1 What It Does

Movie Match for Android allows users to browse through multiple streaming services without having to change applications. It also allows users to connect and communicate with each other if they so desire. The application can be used by individuals looking for a one-stop browsing application, couples and friends that have difficulty choosing what to watch and by individuals under lock down or isolation measures to have someone to chat with while watching movies and series.

### Device Requirements

The application is for Android devices only and those devices must be capable of running Android version 7.0 or better. This is a requirement of the streaming services and as such, cannot be altered by us. The programming language is Java, even though Kotlin is the language of Android, as it is the language with which our head programmer is the most comfortable. The devices should also be using the most current version of whichever web browser that they prefer, to allow for compatibility. The application can interact successfully with the different social media platforms, the device's messaging application as well as the different streaming platforms.

1. System Architecture

### User Interface

We originally attempted to use a Card style view to display the movies and series, unfortunately we had issues populating the cards with the API results. As such, after starting the project with a new VCS link, we pivoted to a Recycler View to hold the API results. The older files that will be included with the finished project will show the Card version, which will be reattempted outside of the current scope and timeline. We have used the Navigation Drawer and Contextual menus for the project, allowing for ease of navigation. See figure 3.a for an example.

*Figure 3.a Recycler View example*

Graphical user interface, application

Description automatically generated Shape, rectangle

Description automatically generated with medium confidence

### Database

For storing matches and returning data to the different Recycler Views, we decided on Firebase Realtime Database and Firebase Storage to suit our needs. In the original planning, we decided upon three tables, Users, Movies, and Matches. See Figure 3.b below

*Figure 3.b Original Database*

**Graphical user interface, application, Teams

Description automatically generated**

As shown above, the three tables and their cardinalities were created and implemented in the initial stages of the project. Once we applied normalization to the database, we found that the Matches table was containing redundant information and as such, we dropped the table. Almost all the streaming information is coming from the API call and because of this, we did not have to store information beyond the ID number and title of the movies and series. Due to the structures of the Firebase Database system (JSON tree), figure 3.c shows a clearer picture as to how the remaining tables were built.

*Figure 3.c Firebase JSON Structure*

Graphical user interface

Description automatically generated with medium confidence

The above example is a better representation of how we structured the database. The Users table above shows the Primary Key and Foreign Key used to reference our other table (Movies). We were able to implement all matches and display all “liked” movies without having to implement another table. Figure 3.d will show how the Movies table was structured after the midterm change in scope.

*Figure 3.d Movies Table*

**Graphical user interface, application

Description automatically generated**

* 1. **Architectural Design**

Using a modular program structure, we were able to break down the project into independent pieces to explain and demonstrate functionality more easily. These pieces include the registration, login and logout, account update and deletion as well as the swipe and match functionality. As the application also redirects to both the messaging and streaming platforms already found on the users’ devices, these pieces will be included in the design description.

As shown in the GitHub README file, we started with the Login and Registration for the user. This was decided to be early in the iterations as we required users to be able to start coding on creating Matches. Our focus was on Google and Meta as the third-party sources for registration as we felt that most end users would fall within the three categories. Firebase was also chosen by the team at this point of the project.

Once this was successful, we moved into parsing the JSON response from the API call (*Streaming Availability by Movie of the Night*) and deciding which information was desirable. The second iteration required the swiping mechanism to be implemented as well as matching the users through a positive action. This was a large hurdle for the team as we were able to read and retrieve the information, the difficulty lay with attempting to display the information in the Card view mentioned above. We moved to the Recycler View that is present in the current iteration and the display of the API call was implemented. The matching of users was also a difficult part of the programming section and took a substantial portion of time. Again, the issue was not in seeing the information, it was in linking the users and displaying the information.

The next iteration for the application was used for the “bells and whistles.” The User details card can be used for contacting each other, the Share button for social media was added and the Play button was also successfully implemented. This iteration was the most successful and least time consuming as the steps in gaining knowledge in how to read the API response and adding third-party application functionality aided us in this final push towards the deadline.

### Decomposition Description

As there were no stakeholders to help with gathering the requirements or to dictate the flow of data, we spoke with several people, with and without programming or software development experience and put together a top-level DFD (Data Flow Diagram) as shown in figure 3.e below.

*Figure 3.e Top Level DFD*A picture containing text, row, bunch, different

Description automatically generated

This original plan showed how two Users could meet in a lobby, determined by an OTP (One Time Password) or a hyperlink sent in an SMS or social media message. Once the new direction was decided upon, the top level DFD had to be changed and the following diagram (figure 3.f) was drafted.

*Figure 3.f Top Level DFD(Final)*

Diagram

Description automatically generated

As shown, the application’s new direction allowed for the target groups of users to be extended and allowed for a greater range of usage of the application. From here, we will break down how the application was separated into the different modules and subsystems.

**Module 1. User Registration, Login etc.**

Registration, login, logout etc. are all handled by the Firebase database system. The information saved here is called throughout the application (Welcome page, details card, messaging to mention a few) and the system was built as shown in Figure 3.g below

*Figure 3.g User Registration Module*

Diagram

Description automatically generated

**Module 2. Swiping and Matching**

From the Welcome page, users can start the main phase of the application which is the swipe and the matching. This was an extensive part of the process and required more than a week of reading documentation and logging each variable to see where, when and in what form they were being saved. Figure 3.h below shows how this module was built.

*Figure 3.h Swiping and Marching Module*

Diagram

Description automatically generated

**Module 3. All Third-Party Interactions**

Once a movie or series has been swiped, matched or in some other way interacted with, the third module of the project begins. We have implemented social media and message sharing, contact between the users and the ability to open the different streaming applications, and play the selected item. Figure 3.i demonstrates the flow of this module.

*Figure 3.i Third Party Applications*

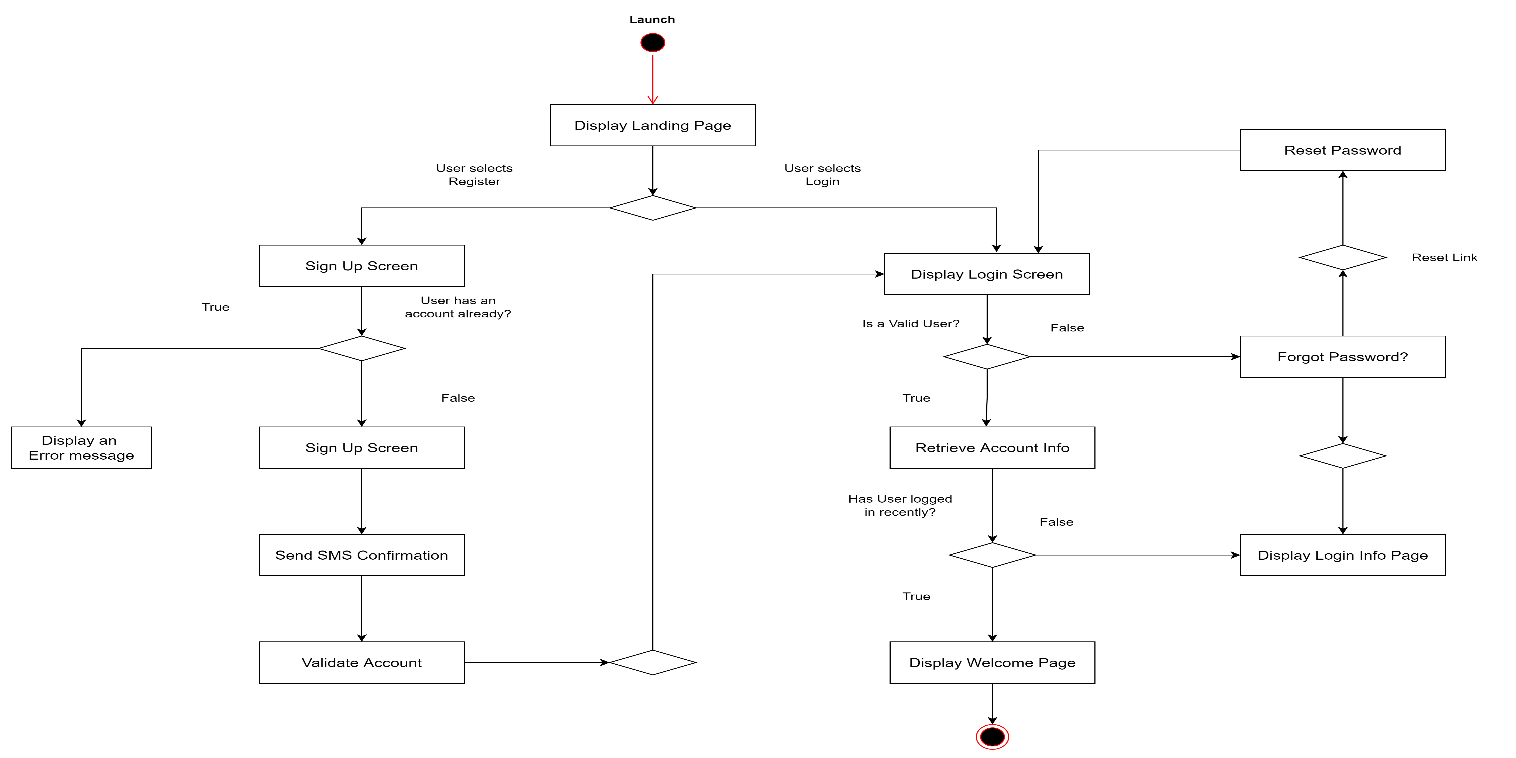
Diagram

Description automatically generated

* 1. **Sequence Diagrams**

Once the above modules were created and merged, there were bugs and fixes that were required to be able to finalize the application. Before the final testing could be implemented, we tested the sequence of actions to ensure that the modules were complete and without crash errors. Figures 3.j, 3.k and 3.l show these sequences.

*Figure 3.j User Registration to Welcome Page*

****

*Figure 3.k Welcome Page to Swipe and Match*

Diagram

Description automatically generated

*Figure 3.l Swipe and Match to Third-Party Applications*

Diagram

Description automatically generated

* 1. **Design Rationale**

Original planning of the application was to have a Card View style of presenting the movies and users, but we had a difficult time with the implementation. As the JSON response to our chosen API call contained 8 items per page, the Card View libraries that we looked at seemed to hold all the responses in one Card. After the second reset of the VCS repository, we moved to a Recycler View, which we were more familiar with and were more easily able to integrate the JSON response. The eight items in the Recycler View can be swiped left or right until the page refreshes itself, or the user can use the Refresh button to generate a new random page and display the new results. We decide to use model classes (using OOP properties) to create Movies and Users, as we would be able to use the accessors to access the information needed. The menus chosen (Navigation Drawer, Spinners and Contextual) were done so as they seemed the most apt to fit in the outline of the application. We hold the user and movie information in a Firebase database and use the Query function to check for matched users. Of course, the application can also be used by individuals, with the available Play button available from the Details page.

4. Data Description

**4.1 Data Description**

The major data and system entities of the application include the Users model, the Movie model, the Array Lists of both models, the tables in the database (again, Users and Movies), the Recycler Views of both models, the API response body (which is returned in JSON format) and Activities for every section. There is a “Swipe” interface that was implemented for choosing, 3 spinner menus for streaming platforms, type of viewing format, and genre, as well as interacting with multiple social media applications and messaging applications. Snack Bar is used to allow Users to change their minds from positive to negative and vice versa. Toast is used often in the application to give Users confirmation or error messages. The Navigation Drawer holds links to Account, Home, and Logout as well as the in-application User’s Manual.

Each User has their details saved under a unique ID that is generated by Firebase upon creation. There is also a connections column that holds the movies’, and series’ particular platform as well as the unique ID’s (also generated by Firebase upon a right swipe). Users can also save an image of their choice in the Storage section of the database. We have not yet implemented the profile photo to the Matches page but hope to do so soon.

The Movies table of the database holds the movies that have been selected to watch by any user but does not save multiple versions of the same film from the same user. We didn’t remove doubles from the API call as users can change their minds. The table holds the movies and series unique ID so the application can show the proper posters and the link to the external application’s platforms or website version.

**4.2 Data Dictionary**

5. Component Design