Cinemigos: Adventures in App Development

Jack Bodine
University of Puget Sound
Tacoma, Washington, USA
me@jackbodine.com

Clark Brace
University of Puget Sound
Tacoma, Washington, USA
clarkbrace@gmail.com

ABSTRACT

This paper details the design, development, and distribution of the Cinemigos mobile app, which streamlines the process of finding and selecting movies to watch with friends and groups. During the 2023 spring semester at the University of Puget Sound we created a movie swiping app that allows users to discover new movies, see interest overlap with friends and family and, by extension make the process of choosing a movie to watch simple. Over the course of this project we learned the processes of iOS development, UI design, database management, use of API as an information source as well as learning many languages including as Swift and Dart. The results from our efforts rendered a fully fledged application called Cinemigos. As of writing this paper we have our iOS distribution available on the App Store and have begun development of an Android distribution. We plan on continuing to work on this project in the future.

KEYWORDS

Mobile Development, Swift, iOS, Android

1 INTRODUCTION

Cinemigos is a mobile app for finding the best movie to watch with friends. It seeks to remove long movie-related discussions, make it easy to choose a movie to watch, and recommend new movies to users and groups. Users mainly interact with Cinemigos through a swiping interface. A stack of movies is presented to the user who can vote on whether or not they want to watch the movie by swiping right or left for yes and no respectively. Users can then add friends, to see a list of all the movies that they both want to watch. Users can also create groups with multiple friends which show all the liked movies ranked from most likes to least (see figure 1).

There is a large number of auxiliary features we had to implement to finish our vision. This includes account creation, reporting/blocking users, friend requests, a tutorial, and more. We currently support account creation and authentication with e-mail or with "Sign in with Apple." Cinemigos requires users to be signed in to access social features of the app, including adding friends and creating groups. Otherwise, users can use Cinemigos without creating an account. They will still be able to vote on movies that are stored on their device until they create an account when the data is moved to our database.

The goal with Cinemigos was to remove the friction when deciding on what movie to watch both with friends and with large groups. Software is especially suited to solve these types of problems through storing, sharing, and analyzing data. We decided that mobile development was the best platform to solve this problem. Since smartphones have become ubiquitous, mobile applications have become a primary way to distribute software to a large number of people, especially those who are not technologically oriented.

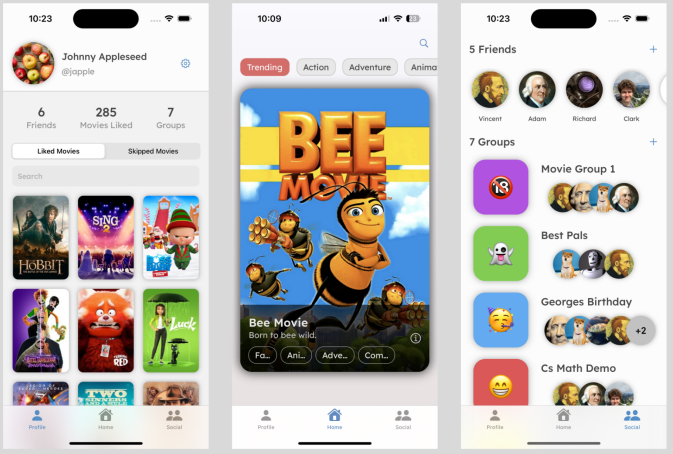

Figure 1: Screenshots from Cinemigos showing the profile, home and social menus.

Cinemigos is currently available on the iOS app store. As of writing, we have 174 total downloads and 90 registered accounts. We decided to focus on the iOS version of the app since we are both iPhone users. Most of the content of this paper specifically refers to the iOS version of the app with only specific sections relating to the Android version. We began development of the Android version only once the iOS version was nearing completion to ensure we would have at least one published product as we likely wouldn't have finished either product if we had worked on both concurrently. Nonetheless, we are still developing both versions and intend to release the Android version on the Google Play Store by the end of summer 2023.

2 BACKGROUND AND RELATED WORK

Many apps and programs exist to analyze movie data, however, we have found no existing solutions that take up the same space as Cinemigos. There are similar apps available that also use a swipe-based interface on movies to keep track of a user's movies and even show-liked movies between a single friend. But these apps are missing group features and are often reported as buggy. Additionally, they lack a decent level of UI design that would make them intuitive to use. Lacking these core features and design, these apps are all relatively unknown and are filled with negative reviews.

To address the shortcomings of similar apps and not make the same mistakes, we spent a lot of time designing the app before we started development to ensure the app would end up easy to use. Additionally, we added beta testers before the official release to help us find bugs and crashes, as those were common causes for negative reviews on similar apps.

Since the movie space is filled with many other apps, projects and websites like IMDB, there is plenty of support and resources for acquiring movie data. We found The Movie Database (TMDB) and its API extremely helpful in our app. Their API is thoroughly documented with how to get and parse any data one might need relating to a specific movie. They also have API requests for getting currently trending movies as well as movies by genre selection[4].

On the development side, app development is a vast field in software, with a total revenue of over $543 billion[10]. There is no shortage of resources and support for app development. Our main resources were the official Swift Programming Language Documentation[1] and the course *CS193p: Developing Applications for iOS using SwiftUI*[8]. CS193p is a course taught at Stanford by Professor Paul Hegarty. He was generous enough to upload the lecture recordings he took from when the course was taught remotely in Spring 2021. The course covers iOS development for people who are already well-versed in programming, making it a well-suited resource. Among other things, the course covers the Swift programming language, how to build user interfaces using the SwiftUI library, and good software architecture.

The UI is influenced by the "swipe" interface common in many dating apps. We figured that it is the best way to get through a lot of movies quickly, so that users have the data to use the other features of the app. After all, we want Cinemigos to make movie decisions faster and less confusing. Neither of us have any training in graphic or UI design but we wanted the app to be intuitive to use and feel professional. We studied Apple's *Human Interface Guidelines* extensively when designing the app.

Before beginning, we also needed to decide on how we would store user data and handle accounts. There are many solutions to the problem of data storage that already exist and we already had some experience in writing and hosting our own back-end from previous projects and classes. However, this is a time-consuming process and would have taken a lot of time away from feature development. We also wanted to experiment with preexisting software and services as we hadn't previously had much experience working with other development tools. Two well-documented back-end services used for many mobile apps are AWS Lambda and Firebase. Ultimately we decided on Firebase, due to having headaches with Amazon Web Servers in the past.

3 IMPLEMENTATION

3.1 Design

Development of Cinemigos began not in an IDE but in Adobe XD. Adobe XD is a tool for designing user interfaces (see figure 2). It takes significant time to write code for a view in SwiftUI and to ensure it works on both iPhone and iPad. We knew it would be wasteful to try to design the app as we went and would be much easier to have a pre-designed reference. This also ended up helping the UI stay consistent with two people working on it.

Designing the UI early forced us to think through all features that we would need to add. This helped us in creating our schedule since we had most features figured out from an early point. This means that there were few surprises throughout development and we were able to stick to our original schedule.

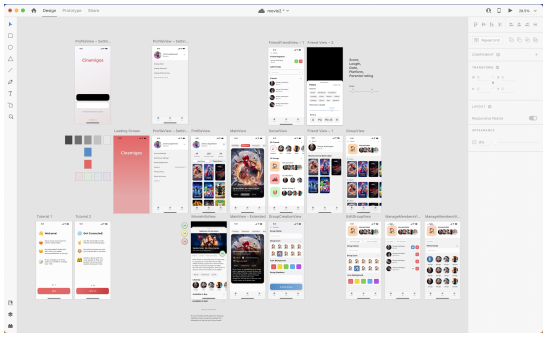

Figure 2: A screenshot from Adobe XD showing the design of Cinemigos.

Some features that were overlooked or unplanned are not in our design and are noticeably less polished in the app. For example, creating groups was planned from the start and it has a satisfactory UI. However editing groups after they have been created was not in the original design, thus the feature looks slightly out of place in the actual app. (In fact, it is actually just the create group UI reused inside of a group page. A menu specifically designed just for changing the group name and icon will look much better.)

Reporting and blocking users was another unplanned feature that Apple insisted we implement and they are simply stock buttons nudged up into the corner of the app.

3.2 Development Environment

iOS development takes place in Xcode (see figure 3). This is essentially the only way that Apple allows developers to write, test and distribute native apps. Xcode is an integrated development environment (IDE) specifically designed for writing code in Swift for Apple products. It has a built-in canvas for quickly testing code and more extensive simulators for testing on all sorts of Apple devices.

Xcode integrates with App Store Connect which is Apple's solution to distributing and testing apps. We were able to use a service called TestFlight to test Cinemigos on our physical devices throughout the semester and later to distribute to beta testers.

When making Cinemigos we frequently tested it on different iPhones and iPads in Xcode to make sure the interface didn't break. What we hadn't considered is that Apple now allows users to install iOS apps on their new ARM-based MacBooks. By default, when run on MacBooks, users can resize the app to any aspect ratio or resolution. However, we had to disable this since it would often cause issues with overlapping UI elements.

The ability to add package dependencies is another important feature that Xcode enables. Swift packages allow us to import components of Swift written by other developers. These packages are added as dependencies to the project which means they are bundled along with our code when running. In Cinemigos we use various packages in order to gain access to high-quality and complicated libraries that already exist[3]. One such package we use is called SwiftUI Shimmer. Shimmer serves as a placeholder animation while images or text are loaded. Another package we use is a popup package that displays error notifications to the users when information is not entered correctly or as an alert to the user when something

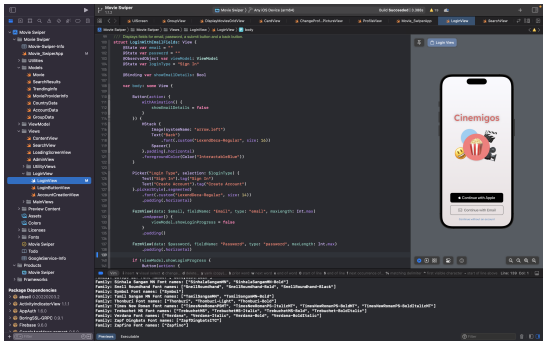

Figure 3: A screenshot of Xcode showing an iPhone 14 Pro simulator on the right.

has been successfully completed. We also utilize the Firebase package which is created by the Firebase team to facilitate data exchange between Cinemigos and the Firebase database.

Cinemigos was coded using the Swift programming language which is a functional and imperative language. Swift was used to write all of the logic behind the app including the data pathways between our various APIs, and the database.

3.3 Features Overview

Movies are presented on the home page of the app. Each movie is given a card that users and swipe right or left on. Or they can tap the card to see more information about a movie like its description. The stack of movie cards is constantly refreshing itself so that the user never runs out of movies to swipe on. At the top of the screen is a list of genres that a user can select to only see movies of that genre. By default, we show the currently trending movies. The magnifying glass in the top right corner brings the user to a search menu where they can find details on any movie. They can also like or dislike any movie directly from the search results.

The profile button on the navigation bar brings users to a page where they can see the list of all their liked or disliked movies. Tapping on a movie brings up all the movie's information including length, genre, and release date. If any of that user's friends have also liked the movie they will show up here (see figure 4). We also display what platforms each movie is available on to stream, rent or buy. Because this information is country-specific we allow users to change their country in settings. From the movie info screen, users can also change their vote on a movie or remove it entirely.

Also in the profile menu users can access their settings. The settings menu lets users change their profile picture, either to a default option or upload their own. If a user is using email for their authentication method, they can also change their email or password from this menu. There is an option to delete your account as well, in which case the user is removed from all groups and friend lists and their account data is entirely deleted.

The social tab is where users can add and interact with other users. They can send friend requests which once accepted lets you see the intersection of your movie preferences with that of your friends. It also shows how similar your interests are to your friend

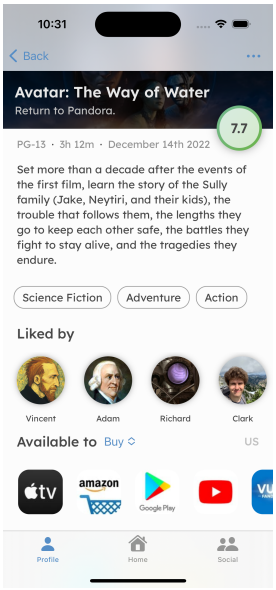

Figure 4: An example movie info page showing which friends have liked it and where it is available for purchase.

via a similarity score. This is calculated using the Jaccard index of both sets of liked movies[5].

Once a user has multiple friends they can create groups. A group runs through every movie liked by each member in the group, tallying up the number of likes as it goes. Then it displays them from most to least likes. Any member of a group can edit the group, add or remove members, or send friend requests to any other member of the group who might not already be on their friend list. Groups also have a section called "Cinemigos Picks" these are movies found in the favorite genre of the group that none of the members have voted on. It serves as a way to expose groups to more niche movies that they would possibly like.

Cinemigos is account optional for the most part. The login screen has a button labeled "Continue without creating an account." If a user chooses this option they can still swipe on movies and use the search function, however, they cannot access any features of the social tab and are prompted to create an account if they attempt to. When a user is signed out, all votes are saved to the user's device, we do not create a document for them on our database. When they create an account, we check if there are any movies saved. If there are we move that data from the device into the newly created account.

3.4 MVVM

The design pattern that we used in order to structure Cinemigos is the MVVM (model view view-model) pattern (see figure 5). It essentially splits up the project into three parts, the model, the view, and the ViewModel. The model is in charge of representing how data is stored and is stateful. Additionally, the model covers all data sources such as databases and APIs. The view is responsible for how the information is shown to the user as well as for detecting user input and it is stateless. The ViewModel is in charge of relaying

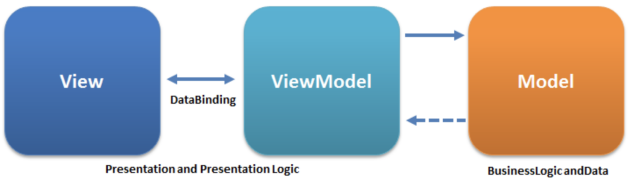

Figure 5: The Model-view-viewmodel pattern.[6]

information between the model and the view as the user interacts with the view[8][6].

The models that we use for Cinemigos are used as classes to store relevant information. The movie model stores information about each movie such as genre, rating, and URLs to movie poster images. The group model holds the properties of each group including members, the group name, and the group's emoji. The account data model holds information about users including their first and last names, the URL to their profile image, and their unique user id. Another part of our model is the APIs that we utilize to get our information for the application along with the database we use to store user data. Cinemigos trending information model uses The Movie Database API in order to load in the movie data for users to swipe on and interact with. It also allows for genre selection to enable users to customize the movies being curated in the movie swiping stack.

The ViewModel for Cinemigos has a lot of responsibilities including keeping track of what view should be displayed to users, storing user data while the app is running, requesting data from the model, passing information to the views to be displayed, and receiving information from the views about user intent and adjusting the model and state of the application in response. Essentially on start up the ViewModel checks to see if the user is signed in and depending on the result displays the correct view to the user. Cinemigos allows users to use the application by signing in or through a limited non-social mode that has no access to social functions and that stores all data locally rather than on our database.

The ViewModel also holds lots of additional data while running in order to keep the program as responsive as possible. One way this is accomplished is by cashing some of the data locally. While the ViewModel does hold references for where to access information from our database and The Movie Database through the use of queries to specific ids, those requests take time which reduces the responsiveness of the app and is computationally expensive. In order to reduce these requests recently selected movies and friend profile information are cashed locally in their entirety until they are replaced with more recently used content. This drastically reduces the number of outgoing requests as often, users interact with the same data numerous times.

SwiftUI was used to create the user interface for the application. SwiftUI is an entirely declarative tool kit. This means that it stores no state and simply reflects what it is programmatically defined to show. Information can be requested from the non-declarative side by way of the ViewModel in order to display data dynamically. SwiftUI's its central design paradigm centers around the idea of creating views that are essentially defined user interface regions. Views can be added together in order to create other views and through this workflow, the UI is constructed. Another important aspect of SwiftUI is user intent detection. Making use of functions that detect user input, functions from the ViewModel can be called to change the state of the app which is then reflected by changes in the UI.

In Cinemigos we have one main content view which consists of the three main screens users use, the profile view, the swiping view, and the social view. Each of these three views is composed of countless sub-views which help to contain different parts of the UI and also decrease code duplication as once a view is written it can be reused as many times as needed. Cinemigos has upwards of 50 unique views that comprise the user interface of our application.

3.5 Firebase

Firebase is a cloud service created by Google which serves as our back-end database for Cinemigos (see figure 6). It controls everything related to storing user-generated data, retrieval, and user authentication among other functions. The Firestore database allows for the creation of documents which are essentially containers that store user data fields. These fields are what Firebase calls collections. The Firestore database has the additional capability to store other data such as images in a separate part of the database. Firebase also has a functions service that allows custom functions to be written to automate tasks by detecting changes in the database. Firebase's messaging service can also be used to send data to the user's devices even when the associated application is not running. Firebase is also a service that works across many platforms. This means that our iOS and Android distributions of Cinemigos use the same Firebase project and database to store data and connect users.

Our implementation for Firebase consists of four main parts: user authentication, user data, group data, and user profile images. When users sign up or log in to Cinemigos, Firebase uses their credentials to confirm the user's identity using industry-standard authentication protocols including OAuth 2.0. This allows for other services such as login with Apple and Google to be added as forms of authentication. Once an account is created it is given a unique user id that is used to associate with that user's account and data[7].

When it comes to storing user data each user has access to their document's collections. Each user has collections to store information such as their name and username, lists of liked and disliked movie IDs, the user's list of friend Ids, incoming friend requests, the group id of the groups the user is in, and more. The other main data document stored in the database are groups. Each group document stores collections of information about which users are in the group, the group name, emoji, and group color. We also used the Firebase storage service in order to store user-uploaded profile images. When a user uploads an image they wish to use as their profile image, it is down-scaled and then stored in the database. The URL to their image is then added to their user document for future reference.

Data deletion is also an important service that Cinemigos provides. Should a user decide they would like to delete their account and all associated data, first requests are sent to the database to scrub the deleted users' data. This involves removing the deleted user from all of their friend's documents and the groups the deleted

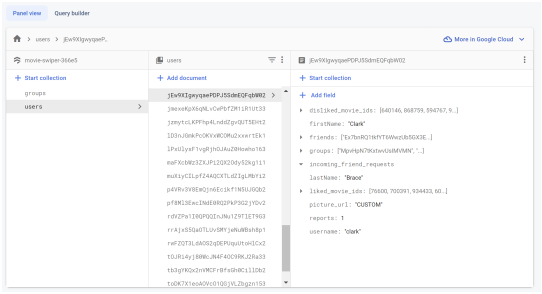

Figure 6: Firestore Database.

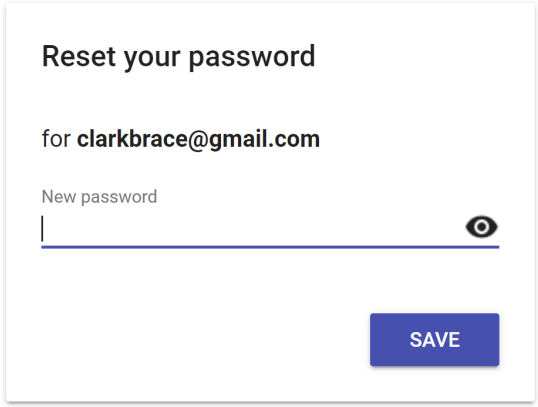

Figure 7: Reset password dialog.

user was in and deleting the user's profile image. Next, the user's account authentication is deleted meaning that if the user wanted to they could potentially create a new account using the same credentials as all references to the old account no longer exists. Finally, all personal data on the user's device is deleted and the user is sent back to the login screen.

3.6 Reset Password

Another big consideration when creating an account-based platform is what to do when users forget their passwords. This problem is a direct result of Cinemigos having the ability to sign in and create accounts with email and password. The tricky thing about this problem is that it cannot be solved within the confines of the application and necessitates an auxiliary system in order to help users recover their accounts.

In order to solve this issue we first use a Firebase authentication feature that emails users if they press a button on the login screen that says they forgot their password. From the Email, users are able to click a link to our recovery page which helps the user with resetting their password (see figure 7). This system allows users to quickly and easily reset their passwords should they forget them. Luckily this was the only other system that needed to be created for account recovery as other forms of user authenticating such as sign in with Apple or sign in with Google are built into their respective systems and have their own recovery process which we do not have any control over.

3.7 APIs

We have two movie APIs we use for Cinemigos: The Movie Database (TMDB) and JustWatch. TMDB provides requests like "get movie details" which takes a movie ID and returns information about the movie, including the name, image URLs, genre, rating, etc. They also have a "movie discover" request that returns the trending movies and can optionally take a specific genre or range. We use this request to get the ids of movies that build the stack on the main page of the app.

JustWatch only has a get watch providers request. This gives the streaming, rent and buy providers for a movie given its TMDB movie ID. The request returns separate providers for every country, because what might be available to stream on Netflix in the US might only be available on Hulu in the UK. Instead of just using the US data, we have a setting for users to change which country they want to get movie info from.

Initially, we had difficulty getting API requests to work. The examples provided by Apple were helpful but we quickly noticed that while the app was getting movie info to add movies to the stack, the rest of the app wouldn't function. This led us to learn about concurrency in Swift. We learned that the best way to fix this was with *dispatch queues*[2]. In Swift, dispatch queues are FIFO queues that automatically handle concurrency by taking a function, designated by a Task block, and executing it on a pool of threads managed by the system. We applied this to our code for API requests and our code started crashing immediately. What had happened was that the API request would be made on a new thread but the main thread would continue and immediately try to access the data before the API request would complete. We solved this using a *dispatch semaphore* which takes one argument denoting how many threads we want to be able to access the data[9]. We set the value to one, so that only the child thread making the request could access the returned movie object until we dismiss the semaphore, giving the main thread access to the movie data. In the meantime the main thread or main dispatch queue continues to run, allowing the app to remain responsive.

It was our intention to allow users to search through their liked or disliked movies in Cinemigos. This proved to be more difficult than expected. Lists of movies are stored by movie ID number in Cinemigos. Only when a movie poster appears on screen, do we make an API request to TMDB to get the rest of the movie's information including the link to the poster. This way, we don't need to make potentially hundreds of requests to TMDB on launch to get the data for every movie. However, this method doesn't allow for users to search for movies by name since that would require getting the movie data for every movie in the list they are searching. Luckily, TMDB has an API request specifically for searching by name. To search in the app, we just make the single request to TMDB to search by the users search term, then compare the resulting IDS with the ids in the list being searched. To reduce requests even further, each search happens on a new thread. The thread immediately sleeps for a quarter of a second upon being created. Then it checks if the search term in the main thread matches the search term it was given. If they don't, probably because the user typed in another character, the thread stops before making

the API request. This prevents a different request from being made each time the user types a character into the search field.

We wanted to further reduce the number of API requests we made to both TMDB and to Firebase. To do this we implemented caching for movies and users. When Cinemigos realizes it needs data on a movie it first checks the movie cache. This is a dictionary in Swift with a movie id as a key and the movie's data as the corresponding value. If it exists in the dictionary, all the movie data is already on the device. Otherwise, it will make the API request and store the movie in the cache for next time. So if a user is frequently visiting the same group or friend. We do not make API requests each time it's opened. This also reduces load time for the user.

3.8 iOS Review Process

The iOS App Store review process took place entirely on the previously mentioned App Store Connect website. One of the features of this website is editing the app's app store page and communicating with the app review team.

There was a lot of required information before we could submit to the app store. In addition to the data that users see like a description, app screenshots, change log, etc, we also needed to provide information on what data we collected, if we were using any encryption methods, how we store user's data, and who we share it with, different app ratings for different countries and more. Once this was all done, we were able to submit our first build to the reviewers.

A few days later we received a response detailing what we would need to change before we could be accepted onto the store. This included the following issues:

- Require that users agree to terms (EULA) and these terms must make it clear that there is no tolerance for objectionable content or abusive users.
- A mechanism for users to block and to flag abusive users.
- Fix an error caused when trying to add friends.
- Document what happens to user accounts when they are deleted and any user data associated with their account.
- Sign-in with Apple should automatically fill in a users first and last name and not require them to type it in when creating their profile.

This first response was by far the most extensive and included screenshots of Cinemigos, specifically highlighting what things should be changed. We addressed the issues promptly and resubmitted. Following this, there were a couple of weeks of back and forth with Apple. From this point on they would point out only one issue at a time and occasionally even repeated instructions that we had already completed for them.

Eventually, they accepted one of our builds and we were available on the App Store later that day. Since Cinemigos has been available we have released three additional updates, adding some features but mostly fixing bugs that were reported by our users along with bugs we discovered soon after launch. We need to submit each individual update for approval but none of these have been rejected and are usually accepted within a couple of hours.

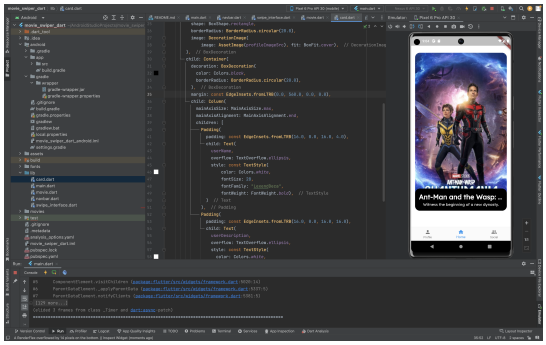

Figure 8: The Android Studio IDE with a Pixel 6XL Simulator.

3.9 Android Development

As previously mentioned we began Android development once we were confident we could release the iOS version before the end of the semester.

Android development is done in Android Studio (see figure 8), the Android equivalent of Xcode. It also features a digital simulator to test the app. Overall the two IDEs are very similar and it is easy to switch between them.

Flutter is a UI framework similar to SwiftUI. Unlike how SwiftUI is specific to iOS, Flutter can be used to design Android, Web or even iOS interfaces. Flutter has similar components to SwiftUI called "containers." Although Flutter seems just as powerful as SwiftUI, if not more, we were not able to get as far with using it. The previously mentioned CS193p course taught us everything we needed to know to make Cinemigos with SwiftUI but we have not been able to find a similar resource for Flutter.

While our iOS app is written entirely in Swift. Our Android version uses both Kotlin and Dart. Flutter requires all components to be written in Dart. Kotlin is similar to both Swift and Java and replaced Java as the primary language for Android development a couple of years ago.

4 RESULTS

We were able to complete our vision with the release of Cinemigos on the iOS App Store. All the features we set out to include are implemented with many extra included. Since the release of the app we have received lots of feedback including bugs and feature requests which we have implemented and plan to continue to do so into the future. We have used Cinemigos along with our friends to plan movie nights and pick our movies for us, it is gratifying to be able to use our app to solve the problem we set out to achieve.

Since release we have reached 174 total installs (see figure 9). This is a good metric to show that our app is circulating and that people are at the least testing it out. 90 of those users found the app interesting enough to create an account. We like to see this since it tells us people are trying to access the social features of the app which is where much of the app's utility comes from.

We began this semester with a schedule and were able to follow it well enough to finish every task we had specifically laid out. In addition to the scheduled features, we had a list of possible "extensions" to implement. From this list, we were able to implement

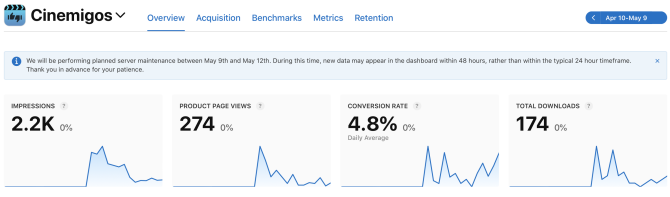

Figure 9: A screenshot of App Store Connect sales analytics for Cinemigos.

a tutorial and add dark mode. Other things from this list that we have made progress on are notifications, an Android version, and adding extra filter options for movies.

While we were in development we also had ideas for new features which we had not initially thought of. These include movie search, group movie recommendations, friend similarity scores, custom profile pictures, and blocking/unblocking users.

All of these features come together in an app that to the users we've talked to feels polished and professional. We are happy with the state of Cinemigos we were able to complete within the time frame of the semester and are excited to continue with the project into the future.

5 DISCUSSION

We believe that Cinemigos is a great way to quickly discover new movies and connect with friends. Our app is very user-friendly allowing users to quickly begin learning about movies and swiping to establish their preferences. We found that often times deciding on a movie to watch can be a very long process, especially in a large group. On top of that often people forget about the movies they have been meaning to watch without some sort of movie list which in and of itself can be a clunky solution. We really wanted to create a product that allowed for easier access to information about movies and then go one step further and allow users to see their overlap in movie interest with their friends and family to make the movie selection process simple.

Our application succeeds in allowing users to quickly discover movies by utilizing our quick yet comprehensive user interface. Allowing users to make snap decisions or look deeper and read the description of movies creates a terrific environment for learning about different films. Our platform also allows users to get as specific as they want with what movies appear as the genre can be changed so that only movies that fit those specifications appear. On top of that Cinemigos also allows for users to search for movies manually which is another great way to add movies rather than waiting for them to appear in the stack. We believe that our simple yet sophisticated movie discovery interface is what sets us apart from other platforms.

Our main selling feature that our application boasts is the ability to add friends and create groups. After researching other similar platforms which curate movie information to the users we found that no other platform allows users to see what movies they have liked in common with others. We think that this makes our product stand out as it drives higher engagement with not only the material but also with others. The ability to create custom groups is also a big selling feature as it allows large groups to easily come to decisions surrounding what they would like to watch as well as see the overlap in their friends' interests.

Our application is also set up such that the database is cross-platform compatible which helps makes our product more flexible and by extension. Currently, one of the major issues when it comes to mobile devices and platforms is the split between Apple and Android users. It is very rare to find a group of people who all use a single mobile operating system. The fact that our application is cross-platform compatible and has a heavy social component not only adds credibility to our product but also increases the ease with which groups of users are able to use our platform.

6 FUTURE WORK

We have enjoyed making and using Cinemigos and now that we have a growing user base we plan to continue development into the future. In the weeks since releasing Cinemigos, we have received a number of feature requests from our users.

People have asked for ways to bookmark the movies they come across either when swiping or looking at group selections. We plan on expanding on this bookmarking idea by creating private user-defined lists like "Favorite Movies", "Want to watch comedy", "Watched movies", etc. that users can add movies too and revisit. This opens the way for more list-related features. Such as making some lists publicly available on your profile or creating shared lists that multiple people can contribute to.

Additionally, people have asked for ways to send movies to certain people. We are planning on adding a feature where you can 'forward' a movie to a friend or group. This would give the forwarded-to user or group a notification that you believe they would like this movie. It would also add it to the friend or group profile in a special section for specifically selected movies.

We also want to expand the amount of movie information provided to include actors, studios, writers, and essentially any information one might be able to find on the movie's IMDb page. People have asked for a way to watch trailers directly from the app. We agree that this would be a great way of exposing users to more information about a movie before they swipe on it.

Another set of feathers that we would like to implement going forward are other forms of authentication. Currently the iOS distribution only allows users to sign in with Apple or with Email and Password. Going forward we would like to expand these options to include other authentication providers such as google, Facebook and possibly even GitHub. This would increase the ease of access users would have to our app and be a good way of giving users more choice in how they interact with Cinemigos.

Throughout the semester we have also considered adding advertisements to the app as an experiment. Building Cinemigos into an ad-supported model could allow us to purchase more advanced movie APIs and repay Apple's developer license cost. By adding advertisements and using the revenue to buy our own ads on other platforms could also help us grow Cinemigos into a larger app.

By the end of summer, we hope to have the Android version of the app completed and released on the Google Play Store. As previously mentioned, we made the decision to start with the iOS app and development for the Android app has lagged behind throughout the

semester. In our experiments with Android, we have been impressed by the tools made available to developers and believe we can finish the Android app in less time than it took for us to finish the iOS version. It helps that we have many of the development design decisions made and since Google owns both our back-end provider, Firebase, and Android they are much more integrated together than Firebase and iOS.

The largest challenge with the Android app right now is that we are very enthusiastic about the future features of Cinemigos. But every feature we add to the iOS version must be mirrored onto the Android version, this means that every feature we add delays Android deployment.

7 CONCLUSION

Working on this project has been a very long but rewarding process that has taught us many new skills and produced a product that we are proud of. Over the course of this project, we learned a few entirely new programming languages, learned to use and navigate IDEs that were new to us, and experiment with and implement a handful of new technologies. Capstone being a term-long project also taught us a lot of useful lessons. Through this process, we learned how to properly maintain a large code base. In the beginning, when we were just starting out on our project we did not have a main design paradigm for the project and soon found that our code was disorganized and hard to maintain. We recognized this issue early and took steps to remedy it by taking a week to refactor our code to use the MVVM standard. We also learned how to conduct research for ourselves in order to learn the skills necessary to keep the project progressing smoothly. Unlike almost every course we have had in college, Capstone forced us to stay self-directed and to plan the path our project would take. This taught us to get creative when things were not working and to become better at problem-solving.

Our main driving goal over all else this semester was to try our best to produce a tangible product that we were both proud to put our names on. We are excited to say that we believe we have. Over the course of the semester, we managed to achieve most if not all of the goals we set initially as well as a bunch that we came up with as the semester progressed. Cinemigos definitely turned out to be a project that we both found deeply rewarding to work on and one that definitely helped develop our skills as computer scientists. We look forward to continuing our progress as we work further to develop Cinemigos.

ACKNOWLEDGMENTS

We would like to acknowledge Professor Paul Hegarty of Stanford University for recording and publishing his course: *Developing Applications for iOS using SwiftUI*. His lectures were the foundation that taught us how to develop for iOS and we are very grateful. We would also like to acknowledge our friends and department members at The University of Puget Sound who have provided us with feedback on the app throughout the semester. Additionally, we would like to thank Adam A. Smith for serving as our capstone instructor.

REFERENCES

[1] 2023. The Swift Programming Language. https://docs.swift.org/swift-book/documentation/the-swift-programming-language/
[2] Apple. [n. d.]. Dispatchqueue. https://developer.apple.com/documentation/dispatch/dispatchqueue/
[3] Apple. 2023. Package Manager. https://developer.apple.com/documentation/xcode/swift-packages
[4] The Movie Database. [n. d.]. https://developers.themoviedb.org/3/getting-started/introduction
[5] Wikimedia Foundation. 2023. Jaccard index. https://en.wikipedia.org/wiki/Jaccard_index
[6] Wikimedia Foundation. 2023. Model–view–viewmodel. https://en.wikipedia.org/wiki/ModelâĂŞviewâĂŞviewmodel
[7] Google. 2023. Firebase Authentication. https://firebase.google.com/docs/auth/
[8] Paul Hegarty. 2021. CS1930 - Developing Apps for iOS. https://cs193p.sites.stanford.edu
[9] Roy Kronenfeld. 2018. The Beauty of Semaphores in Swift. https://medium.com/@roykronenfeld/semaphores-in-swift-e296ea80f860
[10] Statista. 2023. https://www.statista.com/outlook/dmo/app/worldwide