Project Work Documentation
in
iOS Development

Dramady

Jönköping University
Course: iOS Development 2021
Teacher: Garrit Schaap
Students: Anestis Cheimonettos
Rasmus Andersson Kolmodin
Emails: chan20uf@student.ju.se
anra20xq@student.ju.se

Table of Contents

Chapter 1: Introduction

This project documentation is about an app called Dramady. The project name is a reference to the Greek word Dramaturgy which is associated with stage play.
The idea behind creating Dramady was to learn about the iOS programming language Swift by doing something funny and interesting - a movie app.
Watching movies is a part of many people's everyday life and searching the right one can be quite difficult and time consuming. Dramady solves that problem by letting its app users look up for movies in an app with a minimalistic *Graphical User Interface* (GUI) and without the need of registering somewhere. Simply said - Dramady allows users to search, browse and save movies to a watch or favorite lists for easy managing. The *Figure 1* below shows one of the Dramady logos.

Figure 1. The Dramady Logo

Chapter 2: Architecture and Planning

Dramady is meant to be working on iPhone smartphones. The app requires connection to the internet for retrieving different movie's data from an *Application Programming Interface* (API) via the internet which operates with a movie database. In the early stages the planned App Architecture pattern to use was the Model-View-ViewModel, which we tried to follow.

The app was planned to have the following functionality:

- Search a movie by its title
- Browse the all time most watched movies
- Browse the most popular movies at the moment
- Ability to mark movies as favorites or to watch later with offline access to them.

The *Figure 2* below visualizes the very first prototype that was created with Figma during the planning stage of Dramady.

Figure 2. Prototype screenshots of Dramady

The Dramady project will have a multitude of different pages, as listed below in *Figure 3*:

- Main page (The screen a user is taken to upon starting the app)
- Favourites Page
- Popular Page
- Results Page
- Trending Page
- Watchlist Page
- Movie Page(the file is named MovieView in project)

The MovieView will be provided with a title id that correlates with the id from imdb, and then checks the CoreData model to see if the user has previously saved the movie by either adding it to *favourites list* or *watchlist* and therefore storing it locally. If no movie with that id is detected in the CoreData model the API model will be used to fetch it with API calls.

The API model returns different result types(E.g. one for PopularMovies, one for TopMovies etc…) which can be found in the directory *ResultTypes* as shown in *Figure 4* below. When a movie is chosen to be displayed, the MovieView tries fetching the movie by its id from the LocalMovie list(CoreData model). And if the movie was not present in the CoreData model then it gets fetched from the API model.

The *watchlist* and *favourites list* pages are different from the others since we want the data in those pages to be stored on the device to be available for offline use. When having this architecture, the two named pages will make use of the LocalMovie list, which is a CoreData class and prevents us from having conflicts and different types of information on each movie. All other pages in the application will use different calls on the API model to fetch the movie from the API instead of the local storage.

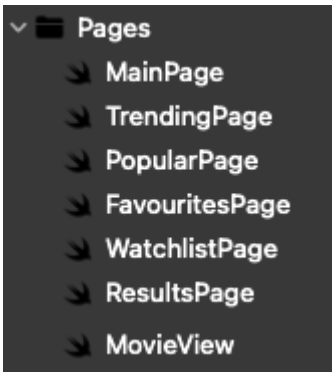

Figure 3. Dramady Screen View pages

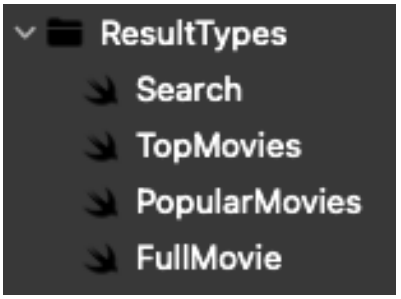

Figure 4. The Result Types

The *Figure 4* below shows the App Architecture and an example of steps a user can take to find a movie called Harry Potter. The figure also shows which screens fetch data from the Api model and which from the CoreData model.

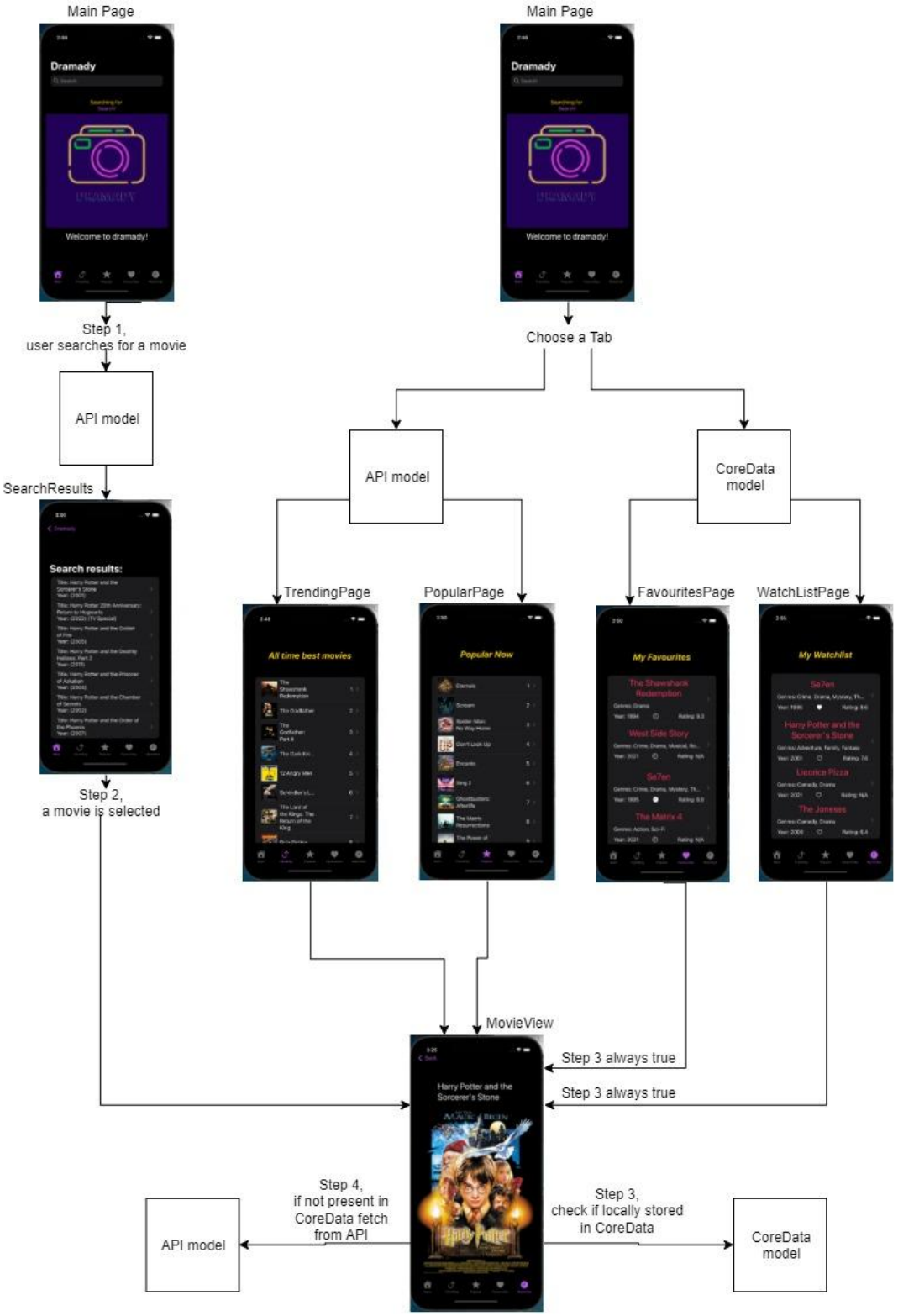

Figure 5. Diagram of App Architecture and App use

Chapter 3: Implementation

Graphical User Interface implementation

The GUI part of the app was implemented with Apple's framework SwiftUI. It was used to create all the views and the visual decorations of the app.
The app has 7 different Views. Bellow is a list of them together with their description:

- **MainPage** - the start point of the app with a brief app description and a search bar for searching for movies. When entering a search text the ResultsPage will be opened.
- **ResultsPage** - shows the search result from the MainPage. If the search process was successful a result in the form of a list with movies that correspond to the search request will be shown. A click on any movie will send the user to the MoviePage.
- **MoviePage** - shows the detailed information about a movie. Has buttons that can add the movie to the FavouritesPage or WatchListPage.
- **TrendingPage** - contains a list of the most popular movies of all time. If any movie gets selected it will be opened in the MoviePage.
- **PopularPage** - same as TrendingPage with the only difference that it shows the most popular movies at this moment.
- **FavouritesPage** - is also available for offline view/interaction. It is implemented in a form of a list that except for movies name it also shows if the movie is also present in the watchlist, its genres, year and imDb's rating. By clicking on any movie the MoviePage will be opened.
- **WatchListPage** - same as the FavouritesPage. Only difference is that it is showing if the movie is present in the FavouritesPage.

Challenges:

Using this framework in this particular project was very difficult compared to creating small projects like a TaskList app because of how complicated it is to plan the hierarchy of the Views. It was surprisingly unbelievable to discover that one View can behave itself differently depending on the structure of its parent View, or even more surprising, parents parent View. The complicity and restrictivness of SwiftUI were causing the most of the difficulties in this project. Of course all these difficulties were also connected to the lack of experience working with Swift language and SwiftUI and after some practice, experiments, creativity and most important - teamwork, the problems were behind.

Logic part implementation

This part of the code consists mainly of the following things:

- The structs that are used to manage the data that the API calls return.
- The API calls - retrieving information about the movies.
- CoreData - provides the functionality to save the needed data on the device.
- The tests - to provide a stable working app.

Chapter 4: Learning

During this project we learned a lot of new things about the Swift programming language and the SwiftUI framework. We improved our skills in using Xcode and especially Git. The skills in using Git is quite often a requirement when looking for a job in our field, so using it in this project was great. The app and GUI design processes were something we trained well by doing proper studies of the existing apps on the market that are similar to the one we did in this project.
The most important is that we as a team both agreed that we learned a lot about working\cooperating with others. The ability to adapt\be flexible and understand colleagues is crucial and can only be trained.
We definitely feel that this project will have an impact in our feature professional and non professional aspects of life.

Chapter 5: Outlook

The future plans with Dramady are:

- Rework completely from scratch the GUI part. We feel we can do it better now.
- Add more features, like deleting the movie record completely from the device via a new page called “History”.
- Add support for browsing series and most important - Anime!
- We will do our best so we can upload Dramady to the AppStore. That will look amazing in the achievement bullet list in our resumes.