ASSIGNMENT 1 REPORT

COSC2659 - IOS DEVELOPMENT

SUBMITTED BY

Hua Nam Huy- s3881103

Table of Contents

1. Introduction

For the first assignment, I have chosen to develop an iOS app about movies called MovieHub. The fundamental idea behind this application is that it will use the TMDB API to retrieve lists of movies from the site and then display them in a dashboard from which the user can choose to view the details of a movie that they are interested in. The inspiration for MovieHub stems from my hobby of viewing movies in my leisure time. While not exatcly a cinephile, I do enjoy searching for films or series with interesting plot to watch, particularly those in the sci-fi, historical, or fantasy genres. Some of my favorites movies are Avatar, Inception, Edge of Tomorrow and the Lord of the Rings series.

While exploring various movie streaming platforms, I often found myself intrigued not only by the content but also by the user interface designs. The seamless and visually appealing interfaces of these platforms greatly influenced my vision for MovieHub. These systems' smooth and visually appealing interfaces greatly influenced my concept for MovieHub. Futhermore, seeing how these platforms managed to displayed a large variety of movies in an aesthetically beautiful and user-friendly manner encouraged me to develop a basic but similar experience within my app.

2. Project Description

MovieHub is a simplified implementation of apps such as IMDb, TMDB, Letterboxd and Moviebase. Developed using Swift, the app aims to provide user with up-to-date information with the latest movies in the world of cinema. When the app is launched, the user is presented with a welcome view that displays the app logo and an info button with information about the developer. Following that, the app will launch on the homescreen, with a tab view offering three major views. The first one that is shown to the user is the dashboard view which contain a list of carousel views for movies from the now playing, popular, upcoming and top rated lists. Second is the list view which display a simple list of most recent movies and the third and final one is the search view which allows the user to look up the name of their favorite movie from the TMDB database. In all of these displays, the viewer can click on the card of a specific movie to view details such as release date, runtime, description, casts, and so on. Because we need to flip between lists and movie information frequently, the design of these views makes extensive use of scroll views and navigation links. Furthermore, there is a dark mode button that allows the user to convert between light and dark mode displays based on their preferences.

Figure 1: App icon of MovieHub

A demo of the application can be viewed through this link: [MovieHub Demo.mp4]

3. Implementation Details

a) Welcome Screen

The first view shown to the user when they launches the app is the welcome screen, this view includes a basic Vstack consisting of the app's logo, name and slogan with a button to move to other views in the app at the bottom. The only notable feature in this view is the info button which utilize the Toggle button to call on action a sheet view which will display information about the developer.

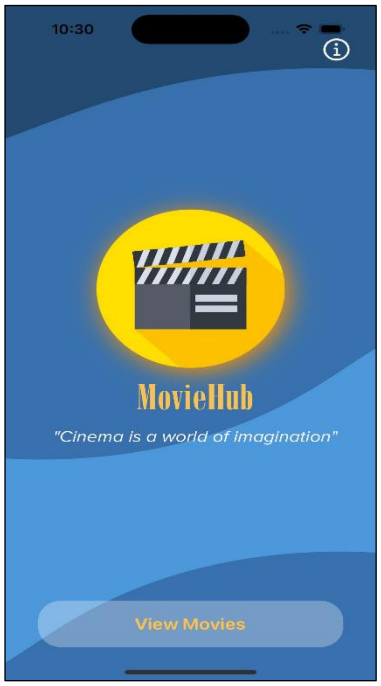
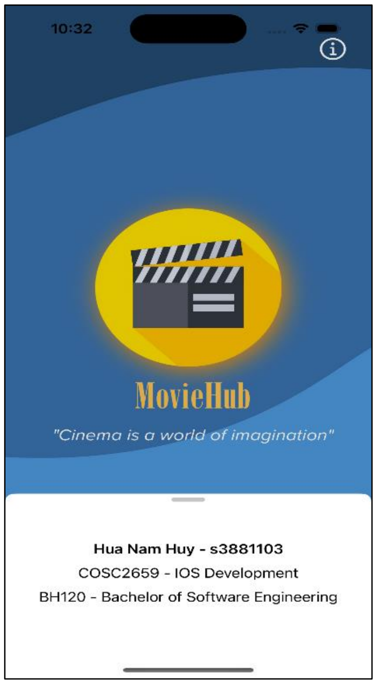
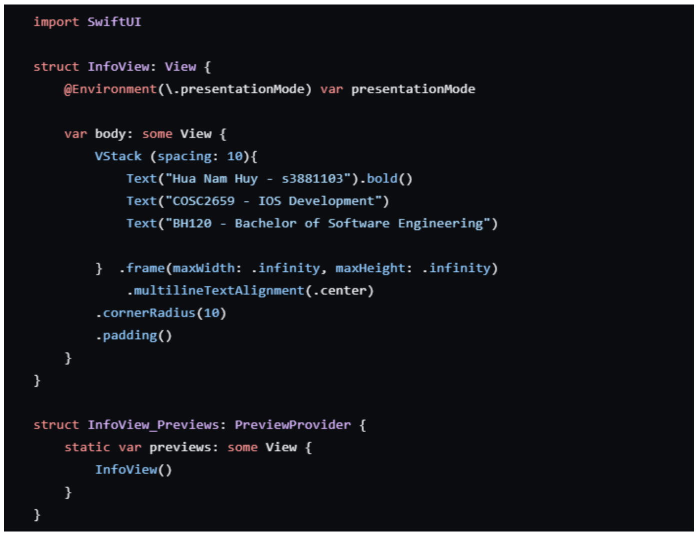


Figure 3: Welcome screen



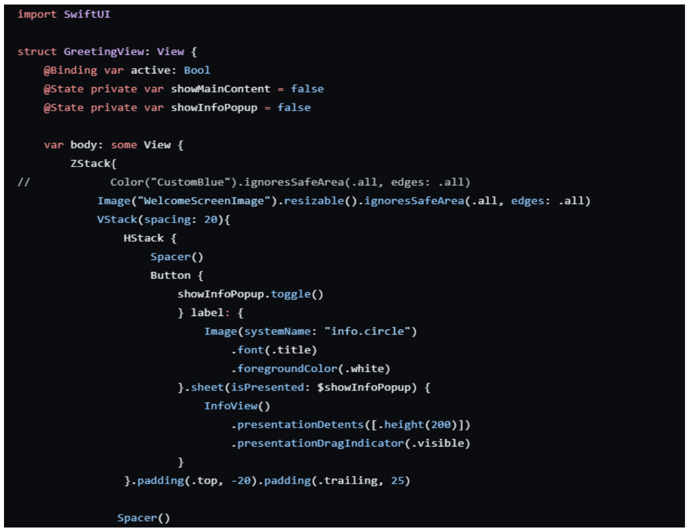
Hua Nam Huy - s3881103
COSC2659 - IOS Development
BH120 - Bachelor of Software Engineering

Figure 2: Welcome screen with info pop up



```swift
import SwiftUI

struct InfoView: View {
    @Environment(\.presentationMode) var presentationMode

    var body: some View {
        VStack (spacing: 10){
            Text("Hua Nam Huy - s3881103").bold()
            Text("COSC2659 - IOS Development")
            Text("BH120 - Bachelor of Software Engineering")

        }  .frame(maxWidth: .infinity, maxHeight: .infinity)
            .multilineTextAlignment(.center)
        .cornerRadius(10)
        .padding()
    }
}

struct InfoView_Previews: PreviewProvider {
    static var previews: some View {
        InfoView()
    }
}
```

Figure 5: Info View code



```swift
import SwiftUI

struct GreetingView: View {
    @Binding var active: Bool
    @State private var showMainContent = false
    @State private var showInfoPopup = false

    var body: some View {
        ZStack{
//            Color("CustomBlue").ignoresSafeArea(.all, edges: .all)
            Image("WelcomeScreenImage").resizable().ignoresSafeArea(.all, edges: .all)
            VStack(spacing: 20){
                HStack {
                    Spacer()
                    Button {
                        showInfoPopup.toggle()
                        } label: {
                            Image(systemName: "info.circle")
                                .font(.title)
                                .foregroundColor(.white)
                        }.sheet(isPresented: $showInfoPopup) {
                            InfoView()
                                .presentationDetents([.height(200)])
                                .presentationDragIndicator(.visible)
                        }
                }.padding(.top, -20).padding(.trailing, 25)

               Spacer()
```

Figure 6: Info button implementation

b) Dashboard View

The main feature of the application, this MovieDashboardView displays a list of carousel views containing movie information within a navigation view. The purpose of this view is to showcase different categories of movies, such as "Now Playing," "Popular," "Upcoming," and "Top Rated." The data for these categories is managed using separate MovieListState objects (nowPlayingState, upcomingState, topRatedState, and popularState), which are observed to update the view's content dynamically. The reference for this design was from the youtube video: https://youtu.be/cTNhMgNV53s.

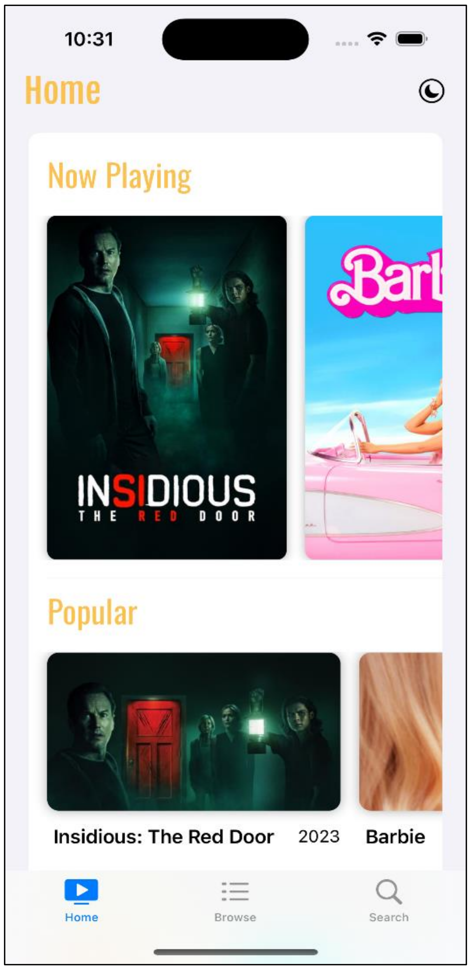
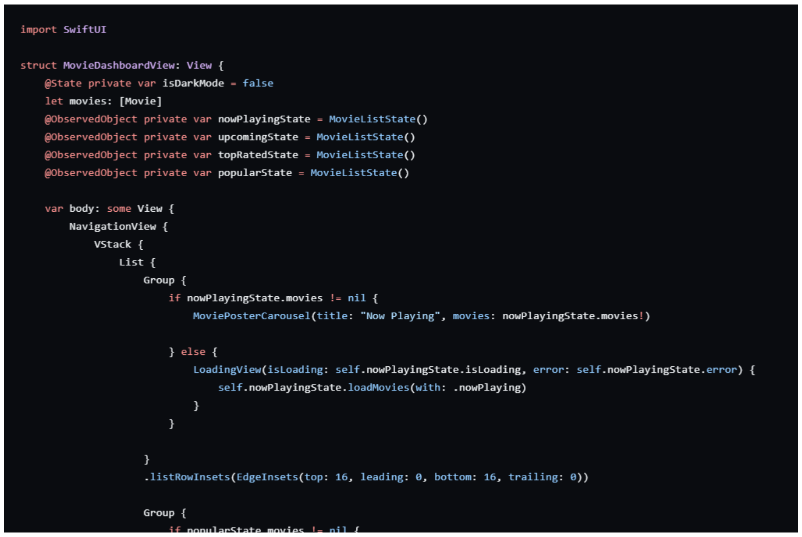


Figure 6: Dashboard UI

Inside the body property of the MovieDashboardView, the main layout consists of a VStack containing a List of grouped elements. Each group represents a different category of movies, and within each group, the code checks whether the movie data for that category is available or not. If the movie data is available, it is displayed using a specific type of carousel view (MovieBackdropCarousel for "Popular," "Upcoming," and "Top Rated," and MoviePosterCarousel for "Now Playing"). If the movie data is not available, a loading view is shown while the data is being fetched.



```swift
import SwiftUI

struct MovieDashboardView: View {
    @State private var isDarkMode = false
    let movies: [Movie]
    @ObservedObject private var nowPlayingState = MovieListState()
    @ObservedObject private var upcomingState = MovieListState()
    @ObservedObject private var topRatedState = MovieListState()
    @ObservedObject private var popularState = MovieListState()

    var body: some View {
        NavigationView {
            VStack {
                List {
                    Group {
                        if nowPlayingState.movies != nil {
                            MoviePosterCarousel(title: "Now Playing", movies: nowPlayingState.movies!)

                        } else {
                            LoadingView(isLoading: self.nowPlayingState.isLoading, error: self.nowPlayingState.error) {
                                self.nowPlayingState.loadMovies(with: .nowPlaying)
                            }
                        }

                    }
                    .listRowInsets(EdgeInsets(top: 16, leading: 0, bottom: 16, trailing: 0))

                    Group {
                        if popularState.movies != nil {
```

Figure 7: Code snippet of MovieDashboardView

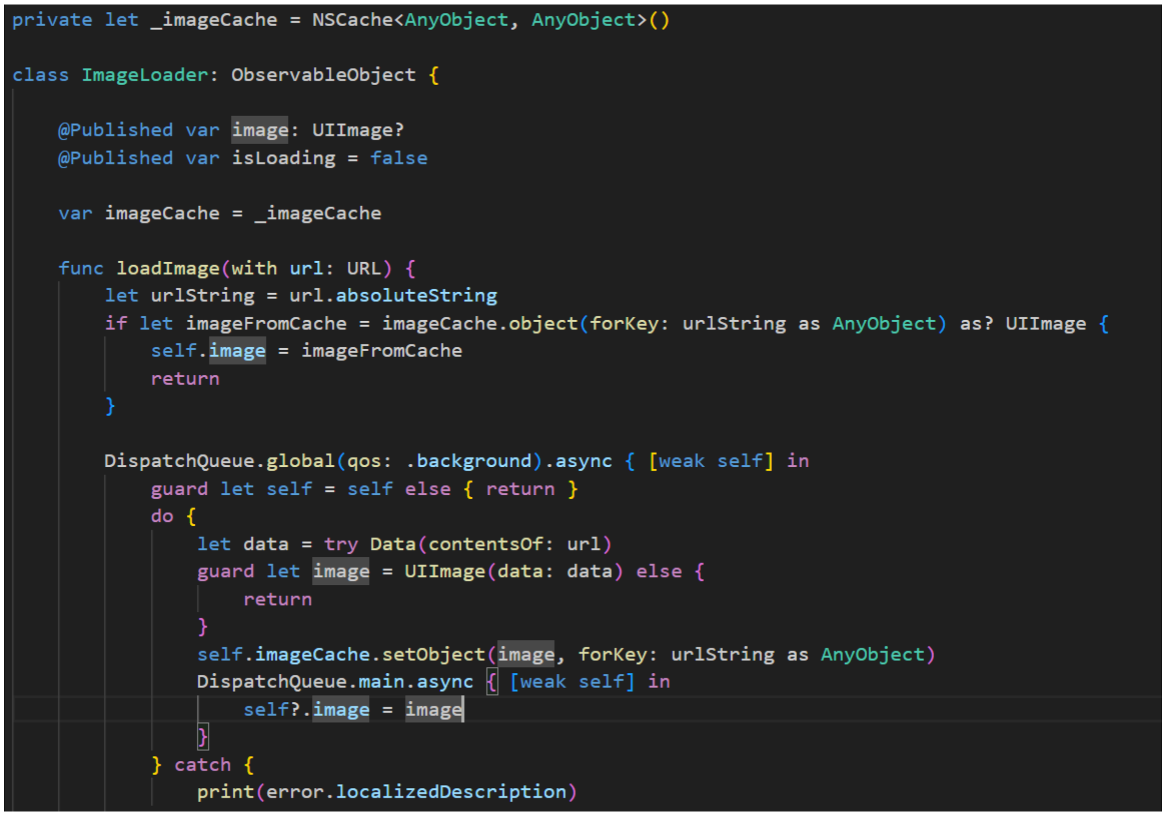
Selecting a movie card will automatically switch the user to the movie details view that display all the information about the film the user picked. Additionally, there's a tab view below that display all the available views that the user can select to switcth to and a toggle button in the navigation bar's trailing position that allows users to switch between light and dark modes. The overall color scheme of the view is then set based on the selected mode.

- **ImageLoader() class**

An important component for displaying the images inside the app. The class is designed to handle the loading and caching of images from URLs, which can be used within SwiftUI views for displaying visual content. SwiftUI can watch and react to changes in the image and loading states because the class uses the ObservableObject interface.

Upon initialization, the ImageLoader class defines two published properties: **image**, which holds the loaded image, and **isLoading**, which indicates whether an image is currently being fetched. The class also contains a private instance of an image cache (imageCache) backed by an NSCache object, designed to store and retrieve images based on their corresponding URL strings.

The core functionality of the class lies in the **loadImage(with url: URL)** method. This function accepts a URL input, attempts to obtain the relevant picture from the cache, then assigns it to the image property if successful. If the picture is not found in the cache, the procedure loads the image data from the supplied URL asynchronously. After obtaining the picture data, it is transformed into a UIImage object, cached, and assigned to the image property. To update the SwiftUI view, the **DispatchQueue.main.async** section ensures that the image assignment is performed on the main thread.



```swift
private let _imageCache = NSCache<AnyObject, AnyObject>()

class ImageLoader: ObservableObject {

    @Published var image: UIImage?
    @Published var isLoading = false

    var imageCache = _imageCache

    func loadImage(with url: URL) {
        let urlString = url.absoluteString
        if let imageFromCache = imageCache.object(forKey: urlString as AnyObject) as? UIImage {
            self.image = imageFromCache
            return
        }

        DispatchQueue.global(qos: .background).async { [weak self] in
            guard let self = self else { return }
            do {
                let data = try Data(contentsOf: url)
                guard let image = UIImage(data: data) else {
                    return
                }
                self.imageCache.setObject(image, forKey: urlString as AnyObject)
                DispatchQueue.main.async { [weak self] in
                    self?.image = image
                }
            } catch {
                print(error.localizedDescription)
```

Figure 8: ImageLoader() code snippet

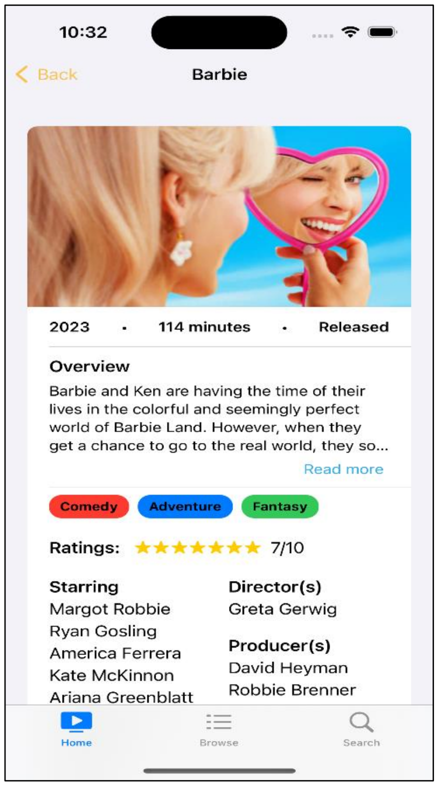
c) Movie Details View

As previously mentioned, the movie details view is what appears after the user click on a card in the carousel on the dashbaord view. This view is designed to display comprehensive information about a movie, including its title, release date, runtime, status, overview, genres, ratings, cast, and crew.

The main code structure, **MovieDetailView** (See Appendices), receives a movieId as a parameter and utilizes the movieDetailState observed object to manage the loading of detailed movie information. The view's body is structured using a ZStack, where a LoadingView is displayed while the movie details are being fetched. Once the details are available, a MovieDetailListView is shown, displaying various attributes of the movie.

MovieDetailListView is responsible for presenting detailed information. It starts with an image (backdrop) of the movie, followed by metadata including release date, runtime, and status. The overview of the movie is shown, with the option to expand or collapse it by tapping on it. The genres are displayed as colored badges. Ratings and the top cast and crew members are also presented. The **MovieDetailImage** structure is used to display the movie's backdrop image. It leverages the ImageLoader class to fetch and display the image from the provided URL, showing a placeholder rectangle until the image is loaded.

Initial design for the details view would have included image carousels of the cast and crew right at the bottom section followed by images of scenes from the movie itself. However, time constraints and the complexity of adding more views and models to showcase those additional images prevented there inclusion in the final submmited version of the app.


10:32

Back Barbie

2023 · 114 minutes · Released

Overview

Barbie and Ken are having the time of their lives in the colorful and seemingly perfect world of Barbie Land. However, when they get a chance to go to the real world, they so...

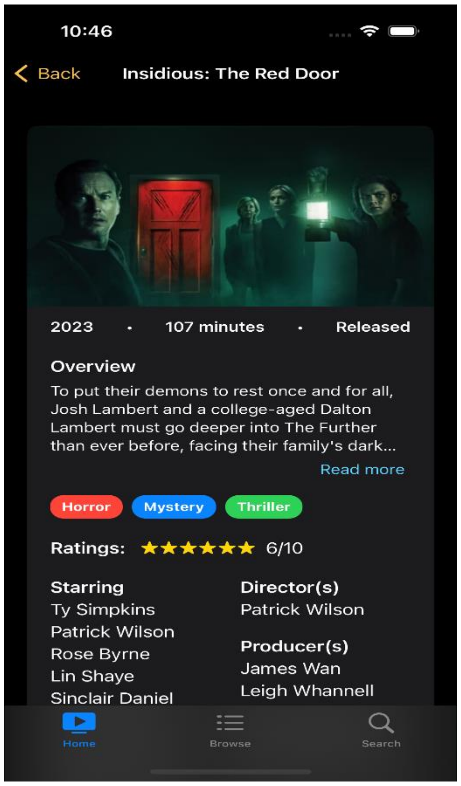
Read more

Comedy Adventure Fantasy

Ratings: ★★★★★★★ 7/10

Starring	Director(s)
Margot Robbie	Greta Gerwig
Ryan Gosling	
America Ferrera	Producer(s)
Kate McKinnon	David Heyman
Ariana Greenblatt	Robbie Brenner

Figure 9: Movie Details UI


10:46

Back Insidious: The Red Door

2023 · 107 minutes · Released

Overview

To put their demons to rest once and for all, Josh Lambert and a college-aged Dalton Lambert must go deeper into The Further than ever before, facing their family's dark...

Read more

Horror Mystery Thriller

Ratings: ★★★★★★ 6/10

Starring	Director(s)
Ty Simpkins	Patrick Wilson
Patrick Wilson	
Rose Byrne	Producer(s)
Lin Shaye	James Wan
Sinclair Daniel	Leigh Whannell

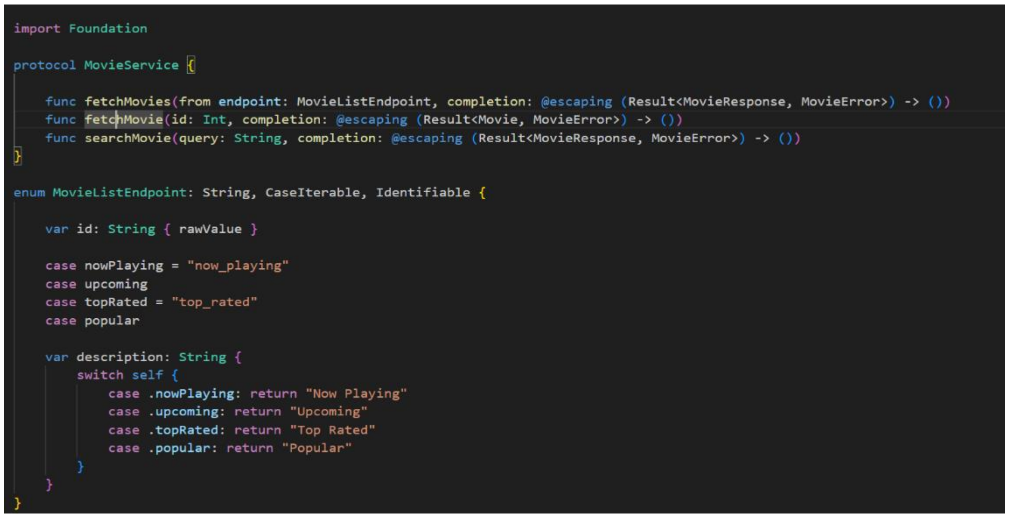
Figure 10: Movie Details UI Dark Mode

- **MovieService and TMDB Service**

These two files handle the task of fetching the data from the TMDB database using an API key to authorize and handle API requests for data then store it in .json files. These data then will be decoded into objects that SwiftUI can use to display the information about a movie onto the details screen.

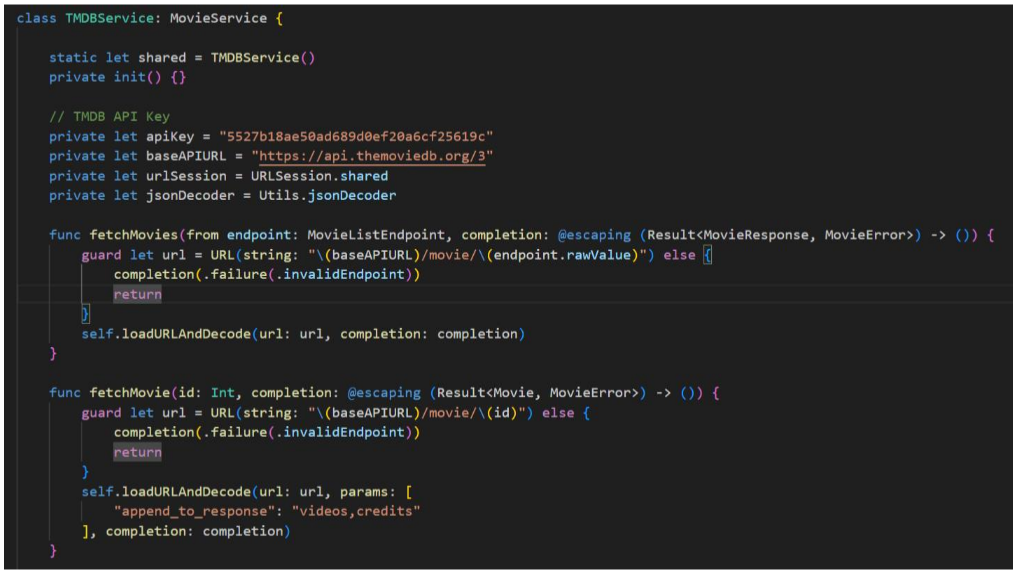
In the **MovieService** file, the MovieService protocol outlines the required methods to fetch movie data, with the associated MovieListEndpoint enumeration specifying the different categories of movies available, such as "Now Playing," "Upcoming," "Top Rated," and "Popular." The MovieError enumeration defines possible errors related to API requests and responses.

TMDBService contains the TMDBService class, which is compliant with the MovieService protocol. It fetches movie listings, comprehensive movie information, and search results from the TMDB API. This class contains the logic for creating API URLs, processing API requests and responses, and decoding JSON data into Swift model objects using the JSONDecoder.



```swift
import Foundation

protocol MovieService {

    func fetchMovies(from endpoint: MovieListEndpoint, completion: @escaping (Result<MovieResponse, MovieError>) -> ())
    func fetchMovie(id: Int, completion: @escaping (Result<Movie, MovieError>) -> ())
    func searchMovie(query: String, completion: @escaping (Result<MovieResponse, MovieError>) -> ())
}

enum MovieListEndpoint: String, CaseIterable, Identifiable {

    var id: String { rawValue }

    case nowPlaying = "now_playing"
    case upcoming
    case topRated = "top_rated"
    case popular

    var description: String {
        switch self {
            case .nowPlaying: return "Now Playing"
            case .upcoming: return "Upcoming"
            case .topRated: return "Top Rated"
            case .popular: return "Popular"
        }
    }
}
```

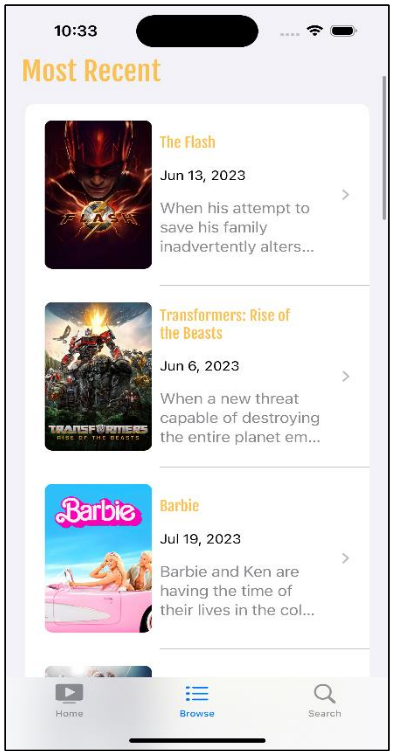
Figure 4: Code snippet for MovieService class



```swift
class TMDBService: MovieService {

    static let shared = TMDBService()
    private init() {}

    // TMDB API Key
    private let apiKey = "5527b18ae50ad689d0ef20a6cf25619c"
    private let baseAPIURL = "https://api.themoviedb.org/3"
    private let urlSession = URLSession.shared
    private let jsonDecoder = Utils.jsonDecoder

    func fetchMovies(from endpoint: MovieListEndpoint, completion: @escaping (Result<MovieResponse, MovieError>) -> ()) {
        guard let url = URL(string: "\(baseAPIURL)/movie/\(endpoint.rawValue)") else {
            completion(.failure(.invalidEndpoint))
            return
        }
        self.loadURLAndDecode(url: url, completion: completion)
    }

    func fetchMovie(id: Int, completion: @escaping (Result<Movie, MovieError>) -> ()) {
        guard let url = URL(string: "\(baseAPIURL)/movie/\(id)") else {
            completion(.failure(.invalidEndpoint))
            return
        }
        self.loadURLAndDecode(url: url, params: [
            "append_to_response": "videos,credits"
        ], completion: completion)
    }
```

Figure 5: Code snippet for TMDBService class

d) List View



Most Recent

The Flash
Jun 13, 2023
When his attempt to save his family inadvertently alters...

Transformers: Rise of the Beasts
Jun 6, 2023
When a new threat capable of destroying the entire planet em...

Barbie
Jul 19, 2023
Barbie and Ken are having the time of their lives in the col...

The MovieListView is a straightforward view that presents movies in a vertical list consisting of "cards" which display the movie's poster, name, release date, and preview text in a Hstack. When the user press on a card, they will be directed to the Movie Details to see more information about the selected movie.

This view reads the.json file and displays all of the movies in it on the screen. While this is good, I had hoped that it would directly query the TMDB database and present all movies in the list rather than only a subset of them. Furthermore, the list lacks a sort tool, which would have allowed the user to sort by movie release date or genre. Although this is relatively simple, I had difficulty incorporating the feature into my application because it was unable to sort the videos during testing. An extra feature that I would like to introduce in the future is the ability for users to click on an icon to pin and add their favorite movie to a personalized list.

e) Search View

The final feature of the app is the Search feature which does what it says – allowing the user to search for a movie by entering its name onto the search bar. As they type, the view will then display a list of movie names and their release dates that matches the user input from which the user can choose to view the movie details.

The **MovieSearchView** defines the MovieSearchView structure, which serves as the main view for searching and displaying movie results. The view is wrapped in a NavigationView and contains a List that presents various components. It includes a custom SearchBarView to input search queries, a loading view (LoadingView) that appears while search results are being fetched, and a section that lists the retrieved movies. The movies are represented by ForEach loops, and each movie is linked to a MovieDetailView upon selection.

The **SearchBarView** defines the SearchBarView structure, which is a UIViewRepresentable wrapper around the UISearchBar UIKit component. This custom wrapper makes it possible to use the UIKit-based search bar within the SwiftUI view hierarchy. The search bar is styled as minimal and allows for text input. The Coordinator class manages the search bar's interactions, including updating the binding text as the user types and handling the search button press.

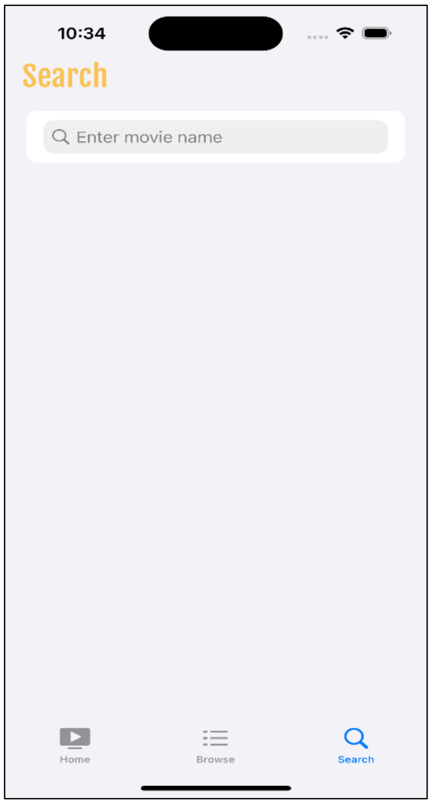
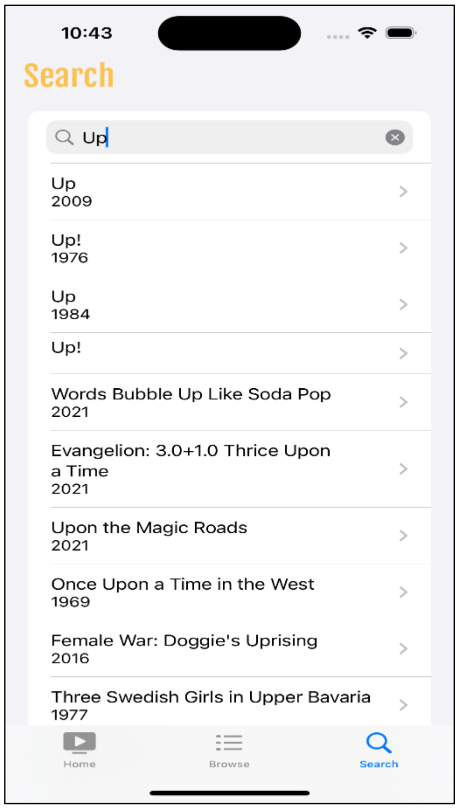


Figure 7: Search Bar



Up
2009

Up!
1976

Up
1984

Up!

Words Bubble Up Like Soda Pop
2021

Evangelion: 3.0+1.0 Thrice Upon a Time
2021

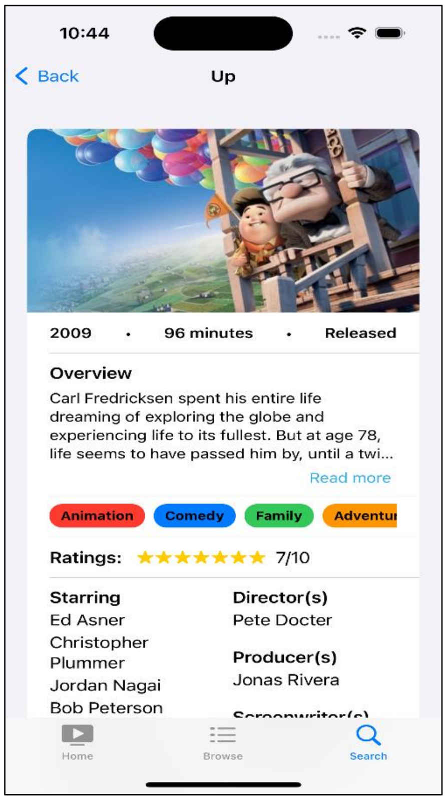
Upon the Magic Roads
2021

Once Upon a Time in the West
1969

Female War: Doggie's Uprising
2016

Three Swedish Girls in Upper Bavaria
1977

Figure 6: Search results based on user input



2009 • 96 minutes • Released

Overview

Carl Fredricksen spent his entire life dreaming of exploring the globe and experiencing life to its fullest. But at age 78, life seems to have passed him by, until a twi...

Read more

Ratings: ★★★★★★★ 7/10

Starring
Ed Asner
Christopher Plummer
Jordan Nagai
Bob Peterson

Director(s)
Pete Docter

Producer(s)
Jonas Rivera

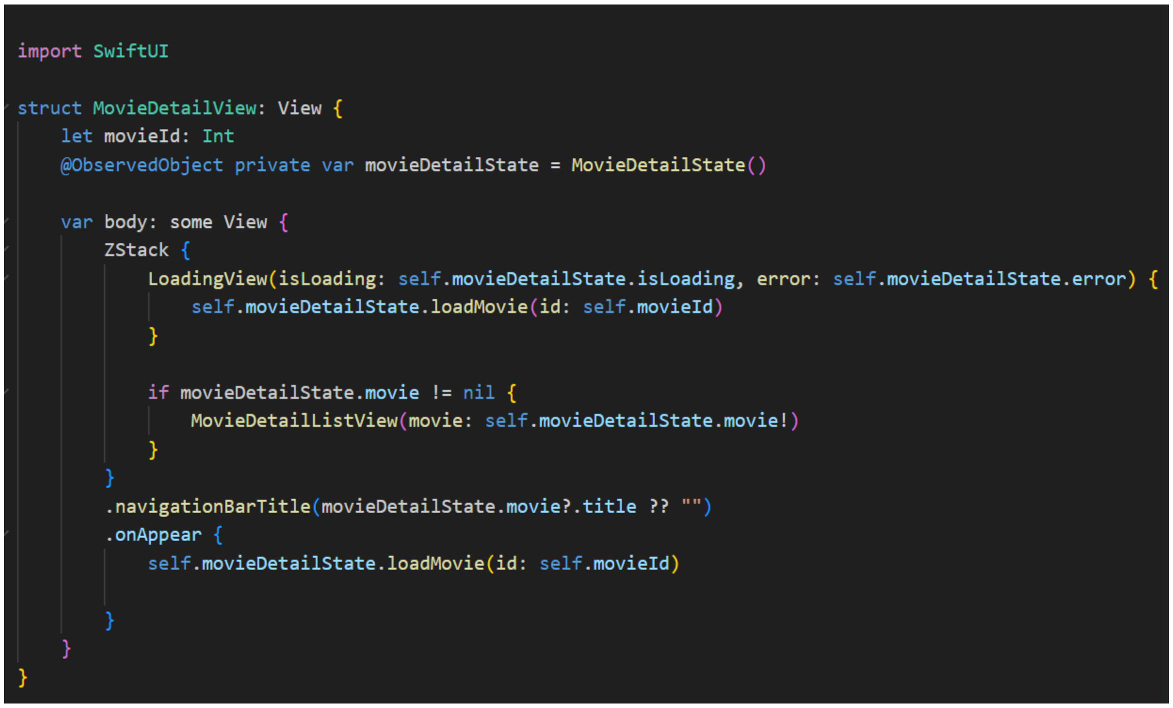
Figure 8: Details of movie from search

4. Conclusion

Overall, I felt that my MovieHub application for the first assignment has met most of the requirements with the exception of certain missing functions such as sort and custom lists. While I am pleased with the functionality achieved, I also want to further improve and develop the app tp enhance the user experience. One significant area for development might be the addition of a thorough sorting option that allows users to organize movie listings based on criteria such as release date, rating, and genre. Furthermore, the ability to create custom watchlists would allow users to curate their own collections, allowing for a more customized and engaging connection with the app. Despite my lack of experience, I felt that I have learned a lot about developing iOS apps using the Swift language. As the app evolves, I am committed to overcoming these limitations and developing MovieHub to become a comprehensive and important tool for all movie enthusiasts.

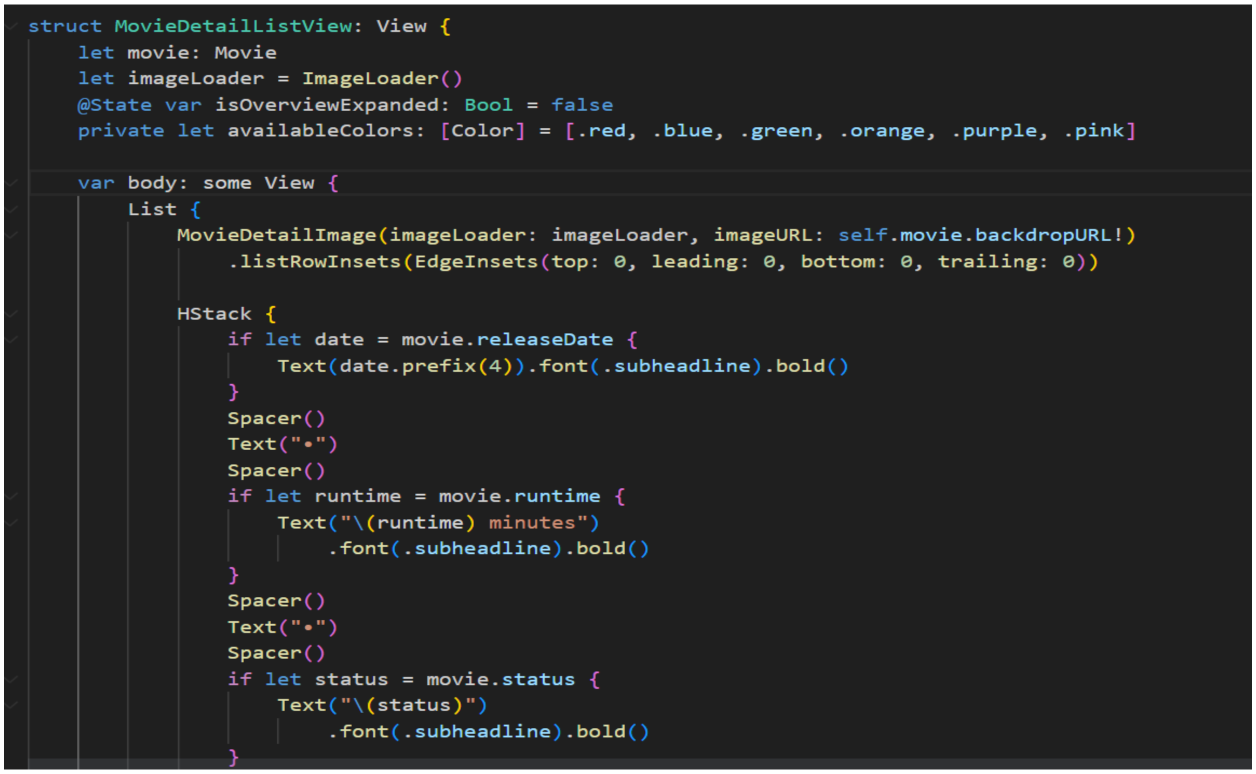
5. References

6. Appendices



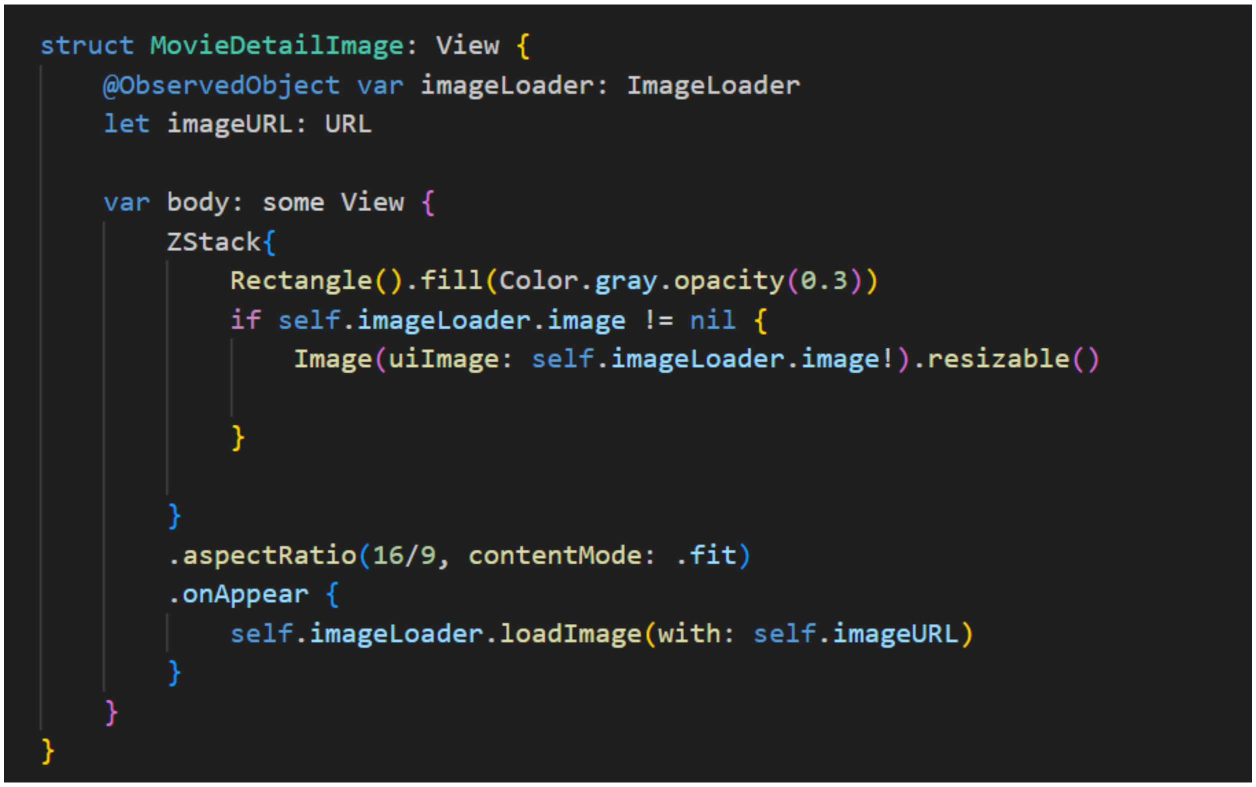
```swift
import SwiftUI

struct MovieDetailView: View {
    let movieId: Int
    @ObservedObject private var movieDetailState = MovieDetailState()

    var body: some View {
        ZStack {
            LoadingView(isLoading: self.movieDetailState.isLoading, error: self.movieDetailState.error) {
                self.movieDetailState.loadMovie(id: self.movieId)
            }

            if movieDetailState.movie != nil {
                MovieDetailListView(movie: self.movieDetailState.movie!)
            }
        }
        .navigationBarTitle(movieDetailState.movie?.title ?? "")
        .onAppear {
            self.movieDetailState.loadMovie(id: self.movieId)

        }
    }
}
```

Figure 9: MovieDetailView code snippet



```swift
struct MovieDetailListView: View {
    let movie: Movie
    let imageLoader = ImageLoader()
    @State var isOverviewExpanded: Bool = false
    private let availableColors: [Color] = [.red, .blue, .green, .orange, .purple, .pink]

    var body: some View {
        List {
            MovieDetailImage(imageLoader: imageLoader, imageURL: self.movie.backdropURL!)
                .listRowInsets(EdgeInsets(top: 0, leading: 0, bottom: 0, trailing: 0))

            HStack {
                if let date = movie.releaseDate {
                    Text(date.prefix(4)).font(.subheadline).bold()
                }
                Spacer()
                Text("•")
                Spacer()
                if let runtime = movie.runtime {
                    Text("\(runtime) minutes")
                        .font(.subheadline).bold()
                }
                Spacer()
                Text("•")
                Spacer()
                if let status = movie.status {
                    Text("\(status)")
                        .font(.subheadline).bold()
                }
```

Figure 10: MovieDetailListView code snippet



```swift
struct MovieDetailImage: View {
    @ObservedObject var imageLoader: ImageLoader
    let imageURL: URL

    var body: some View {
        ZStack{
            Rectangle().fill(Color.gray.opacity(0.3))
            if self.imageLoader.image != nil {
                Image(uiImage: self.imageLoader.image!).resizable()

            }

        }
        .aspectRatio(16/9, contentMode: .fit)
        .onAppear {
            self.imageLoader.loadImage(with: self.imageURL)
        }
    }
}
```

Figure 11: MovieDetailImage code snippet