Movies app

==========

Features

1. List of movies

List of movies API

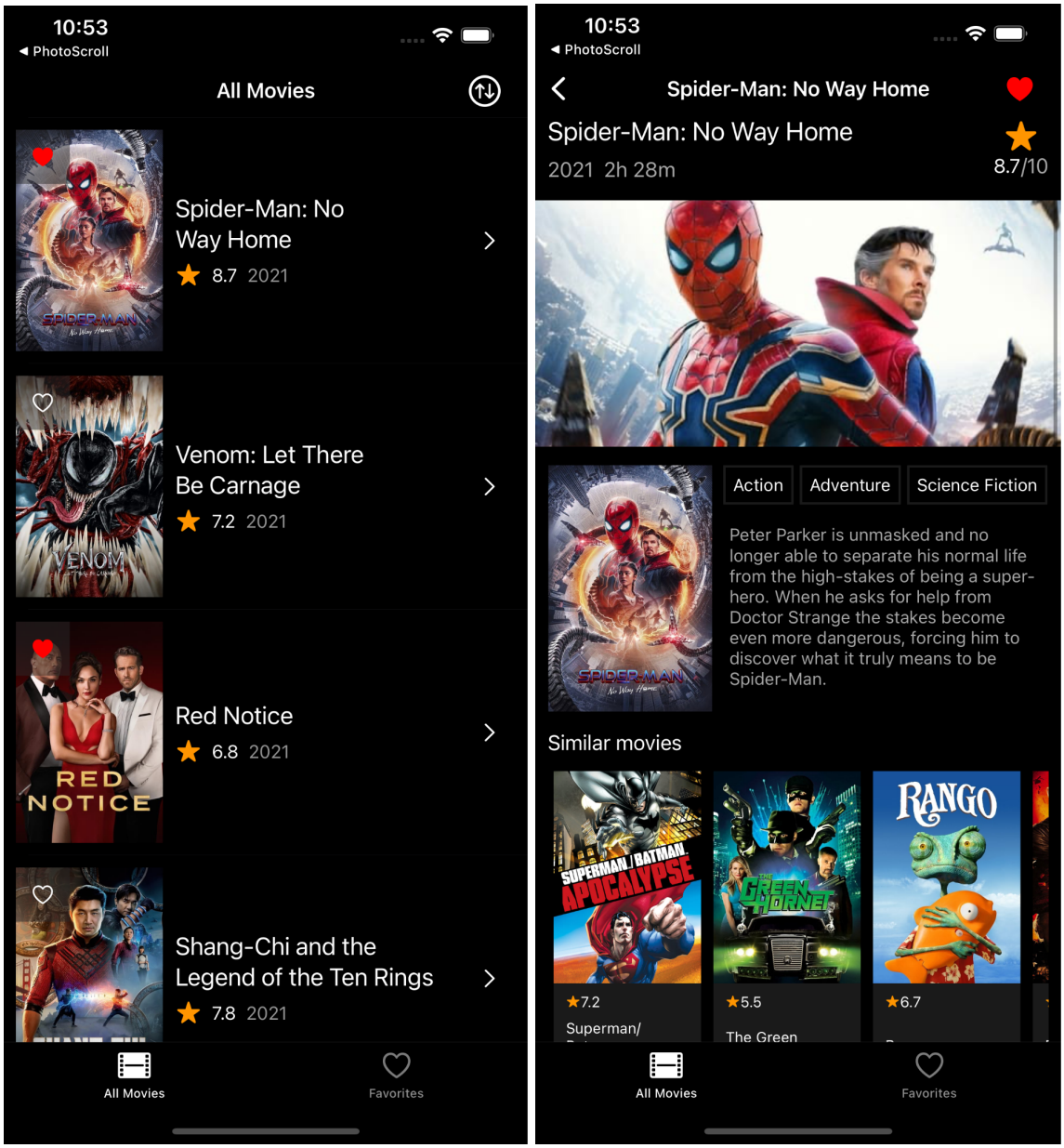
Images API

Title of list: “Movies”

The app must display a **list of movies** inside a UITableView. When the user taps on a movie row you need to open a new screen which will show the movie’s details.

Each row must contain:

- Title of the movie (*original_title*). The title should take as many lines it needs, but to fit into the cell. If it does not fit, truncate the end.
- a small thumbnail with the movie poster (*poster_path*)
- the movie overview
    - Star icon + Movie rating (*vote_average*)
    - Release year (*release_date*)



2. Movie details

Movie details API

Images API

Title: movie title (*original_title*)

- the title of the movie (*original_title*)
- Release year (*release_date*)
- Duration (*runtime*)
- Movie cover photo (*backdrop_path*)
- a small thumbnail with the movie poster (*poster_path*)
- Movie genres (*genres*)
- Star icon + Movie rating (*vote_average*)

List of genres must show all genres returned by the API. Can be implemented as a CollectionView.

3. Movie details + similar movies

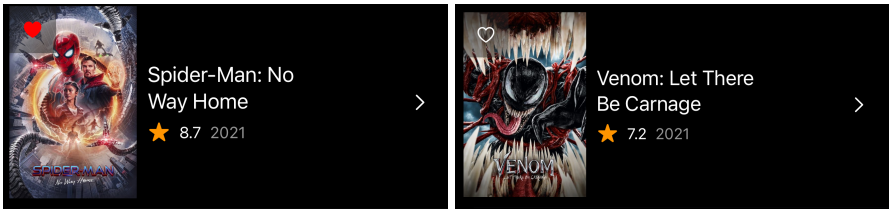
Similar movies API

Same as Movie Details, but adding Similar Movies section. Selecting a movie will open another screen with that movie details.
Similar movies section will be implemented as a horizontal Collection View. Each cell will contain:

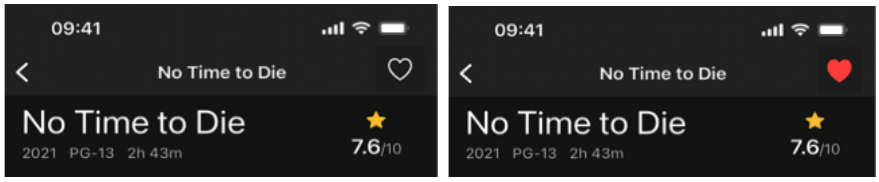
- Movie thumbnail ((*poster_path*) - size should be similar to the one in Movies List
- Star icon + Movie rating (*vote_average*)
- the title of the movie (*original_title*)

4. Favorites

- Add heart icon for each movie
    - Movies List: on each cell, on top left corner



    - Movie Details: on nav bar, top right corner



    - Similar movies: on each cell, on top left corner (not actionable)
- Heart icon has clear background color and it is filled red if the movie is favorite, not filled if it's not a favorite
- A movie can be marked / unmarked as favorite:
    - By tapping on the heart icon on the Movie cell in Movies List
    - By tapping on the heart icon in Movie Details
    - Unmarked by swipe action on cells in Favorites list (see below)
    - Change (new value of the flag) should be reflected everywhere in the app !
- Add bottom tab bar with two bar items: (tabbar looks very similar with navbar - same color)
    - All Movies
        - Icon: film (deselected state) / film.fill (selected state)
        - Contains ALL movies, as the Movies List right now
        - Title of tab bar item AND view controller is "All Movies"
    - Favorites
        - Icon: heart (deselected state) / heart.fill (selected state)
        - Contains just FAVORITE movies. UI / UX same as Movies List
        - **NOTE:** here we don't need a heart icon on each cell !
        - Add Swipe action to "Remove" from Favorites list. This will remove the movie from this list and will update the heart icon on the movie.
        - Title of tab bar item AND view controller is "Favorites"
- Favorite flag will be persisted between app launches. It's up to you where you save this information.

5. List of movies + pagination

Paginated list of movies API

Update list of movies to fetch more movies as you scroll down. Scrolling should be smooth, but even so, if the user gets to the end of the list and the app is still fetching the next page of movies, show a spinner at the bottom of the table view.

==========

6. Image caching

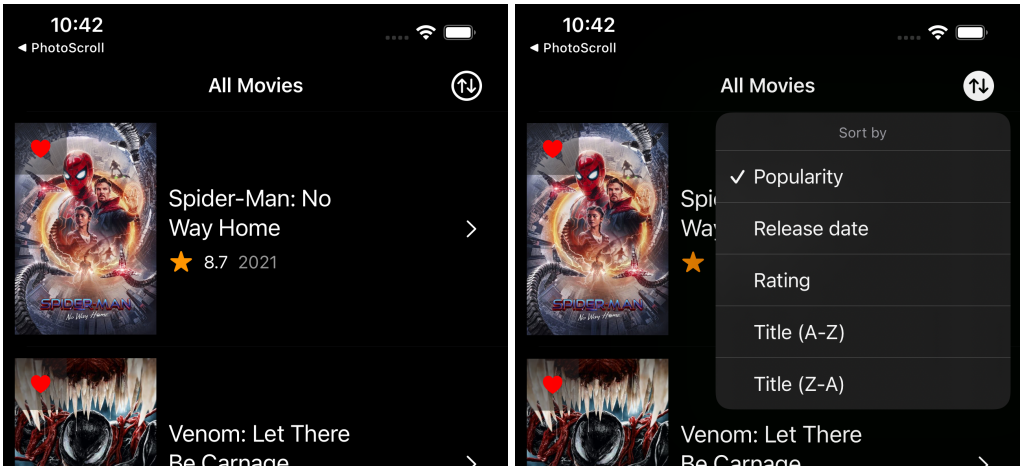
- Cache images that are loaded from API, and next time the image is shown, it is taken from Cache and not downloaded again

7. Movies caching

- Cache movies that are received from API, including those from pagination.
- Next time the app is opened, the cached movies will be displayed until the API request is ready.
    i. If the API request fails, or there is no network connection, the cached movies will remain displayed on screen.
    ii. If the API request succeeds, the new movie data will be cached instead of the old cache.

8. Sort all movies

- In all Movies, add a sort button on top right corner. You can use any of the arrow.up.arrow.down icons from SFSymbols   with fill on selected state.
- Sort button will open up a modal menu, with title “Sort by” and the following options. One single option can be selected, and the UI can either be a checkmark or a radio button.
    i. Popularity (default) : *popularity.desc*
    ii. Release date : *release_date.desc*
    iii. Rating : *vote_average.desc*
    iv. Title (A-Z) : *original_title.asc*
    v. Title (Z-A) : *original_title.desc*
- When selecting a sort option, a new API call to the MovieDB will be made, with *sort_by* parameter set to the proper value (see above in italic). When the response is received, the movies will be displayed on screen. The pagination should work on any sort option.
- Sort option that is selected remains as a user setting, and will remain selected until the user chooses another option for sorting.
- Inspiration: custom view or context menu https://developer.apple.com/documentation/uikit/uicontrol/adding_context_menus_in_your_app



9. Sort favorites

Add Sort functionality to Favorite Movies screen, with same sort options. Sorting will be done “manually”, meaning all sorting algorithms must be implemented in the app, because Favorites do not use an API to fetch all movies from TheMovieDB.

10. Open full screen image Movie Details + animation

Sample video

Open Cover and Poster images in full screen mode, by tapping on that image in Movie Details. Full screen image will support:

- Animation when opening full screen mode
- Image zoomed to fit (aspect fit) in the screen, and the rest of the screen will be blurred
- Device rotation (landscape + portrait mode)
- Pinch to zoom in / out
- Max zoom in 3x
- Min zoom out 1x, if zoomed more than original image size, the image will snap back to fit the screen.
- Double tap to 2x zoom in/out (by 100%) , and the location where the user tapped will become centered in the screen
- Drag in any direction to exit full screen, animated in the direction the drag happened. Dismiss will be done only if the image is in original size. If the image is zoomed in, dragging will just scroll the image.
- If the image is zoomed in, do not let the user drag the image outside of the visible area, leaving only blurred background on the screen. If image margins are dragged inside of the screen, when the user lifts up the finger snap the image back to the margins of the screen.
- See Photos, Facebook etc. apps for reference

- https://www.raywenderlich.com/5758454-uiscrollview-tutorial-getting-started
- https://www.raywenderlich.com/6747815-uigesturerecognizer-tutorial-getting-started
- https://www.raywenderlich.com/18411703-uikit-animation
- https://www.raywenderlich.com/5304228-ios-animation-tutorial-getting-started
- https://www.raywenderlich.com/2925473-ios-animation-tutorial-custom-view-controller-presentation-transitions (intermediate)

11. Similar movies + pagination

Paginated list of similar movies API
Update list of similar movies to fetch more movies as you scroll right. Scrolling should be smooth, but even so, if the user gets to the end of the list and the app is still fetching the next page of movies, show a spinner at the end of the collection view.

12. Search

Search API
Search Keywords API

- ❖ Add Search bar on top of All Movies and Favorites, either on the nav bar, or below the navbar and above the table view
- ❖ https://developer.apple.com/design/human-interface-guidelines/ios/bars/search-bars/
- ❖ When tapping and typing in the search bar use the Search Keywords API to get the most popular keywords

- ❖ Selecting a keyword, or tapping on Search/Done button in the keyboard will execute a search after that string, using the Search API
- ❖ Search bar should have Clear and Cancel buttons enabled when in search mode (see documentation)

13. Authentication

Authentication API

- Create a Startup page that will let users authenticate themselves . It will be shown only if there isn't any user signed in, no session ID saved.
  - Background color: similar to other pages (black or very dark gray)
  - Title: Find a suitable title for the page
  - "Sign in" button, rounded rect border, white filled, black text
  - "Continue without signing in" button, no border, underlined, white text
- "Continue without signing in" button will dismiss the Startup page and let the user into the app (like the app behaves right now)
- "Sign in" button will follow instructions: https://developers.themoviedb.org/3/authentication/how-do-i-generate-a-session-id
  - **Step 1:** Create a request token
  - **Step 2:** Validate the token
    - A. Open the TheMovieDB web page where the user will authenticate and approve the permissions. Web page can be opened with SFSafariViewController or UIApplication.shared.openURL()
    - B. Use our own Login page with username and password inputs https://developers.themoviedb.org/3/authentication/validate-request-token
  - **Step 3:** Create a session ID
  - Keep the sessionID secret and safe, and use it for those APIs that need a session ID
  - SessionID must be refreshed (go through authentication steps 1-3) when:
    - SessionID expires in the meantime
    - Session is wrong, malformed
    - Any of the POST APIs that requires a sessionID, throws an error.
    - Error case: sessionID is not set or is wrong on API

```
401 Unauthorized
{  "success": false,
   "status_code": 3,
   "status_message": "Authentication failed: You do not have
permissions to access the service."
}
```

- Keep sensitive information in a secured place https://www.raywenderlich.com/9240-keychain-services-api-tutorial-for-passwords-in-swift
- Optional: Biometrics

- ➢ https://www.hackingwithswift.com/read/28/4/touch-to-activate-touch-id-face-id-and-localauthentication
- ➢ For Step 2. B “Use our own Login page with username and password inputs” You can use biometrics (touch ID, face ID) in order to access saved credentials.
- ➢ Otherwise you can use biometrics to access the sessionID when the user opens the app and you don’t have the sessionID already. This is just for the purpose of trying biometrics, in real life you don’t need to use biometrics for the session ID or the access token.

14. Settings Page

- ❖ Create a new Settings page and a tabbar item for this page, with “gearshape” icon (”gearshape.fill” for selected state).
- ❖ Settings page will be a table view that contains:
  - ➢ Username: <<username>> taken from Get Details API
  - ➢ Sign out button >> that signes out the user, clears all user data and sessionID and refreshes the UI (where we displayed users data).
    OR
  - ➢ Sign in button << if there is no user signed in >> that directly executes Step 1 from Authentication.

Get Account Details API

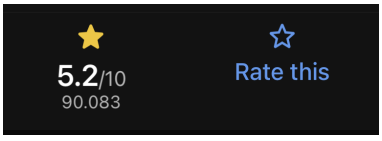
15. Movie details + Rate [authentication needed]

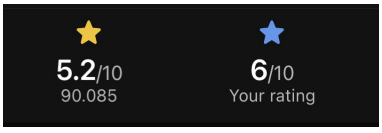
Rate movie API

Add a rate button on Movie Details screen. Tapping on that button will open a modal popover with a slider in it and the title “Rate this movie”. The slider will have [min, max] = [1, 10].
The modal will also have two buttons: Cancel and Submit. Cancel button will dismiss the popover without rating, and Submit button will call the Rating API
You can search for a third party widget for this component and add it to the project using Cocoapods.

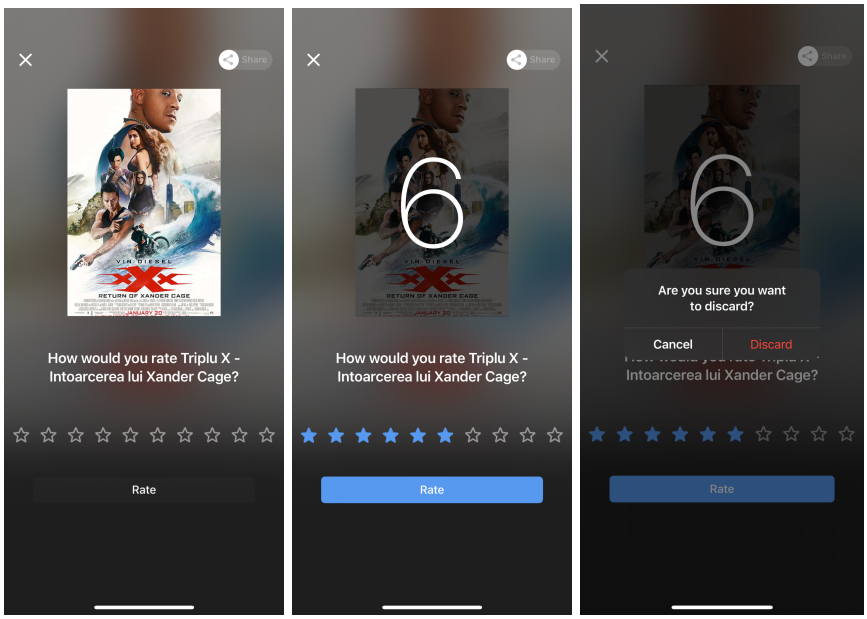
Inspiration:

- You will need to move the existing rating from the top
- Rating and Rate this should be clickable
- There will be a section for this below movie description (above Similar Movies)
- Tapping on Rate this will open either a new full screen (see below), either just a modal popover.





- Rating score will be refreshed on Movie Details page.

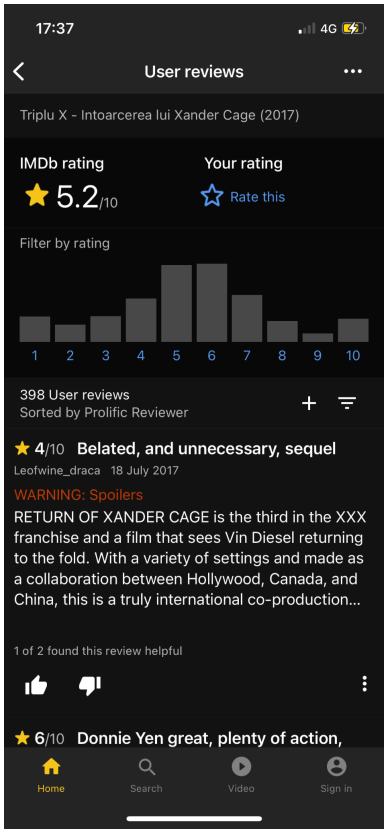


16. Movie Details + Reviews

Reviews API

On tapping on the rating star a new screen will be presented with user reviews
Screen will include (se inspiration in screenshot):

- Movie title
- Rating section (general rating, my rating)
- A list with all user ratings , paginated!
- Each rating cell will contain
    - Rating score (*author_details* > *rating*)
    - Author (*author*)
    - Date (*created_at*)
    - Review content, limited to 5 rows. On tapping on the cell, the cell content will expand to fully display the review. (*content*)



17. Movie details + videos

Movie videos API

Same as Movie Details, but adding Videos section. Selecting a video will open a modal webview with the video URL playing. Use WKWebView or SFSafariViewController.

Section title: “Videos <<number_of_videos>>”
Videos section will be implemented as a horizontal Collection View. Each cell will contain:

- Video thumbnail (first frame of the video)
- Play button
- Title of the video (*name*)
- Type of video (*type*)
- Date (*published_at*)

18. Localization

Add localization to the Movies app. Include at least a new language (ex. Romanian) for all strings in the app.
Set the language on all the APIs request as well, as a query param or as the API documentation requires.

&language=ro-RO
&language=en-US
&etc.

19. Genre filtering

Genre filtering API

Make genres tappable in Movie Details. When tapping a genre, a new screen will be presented, very similar to Movies List.
Title: <<genre name>>

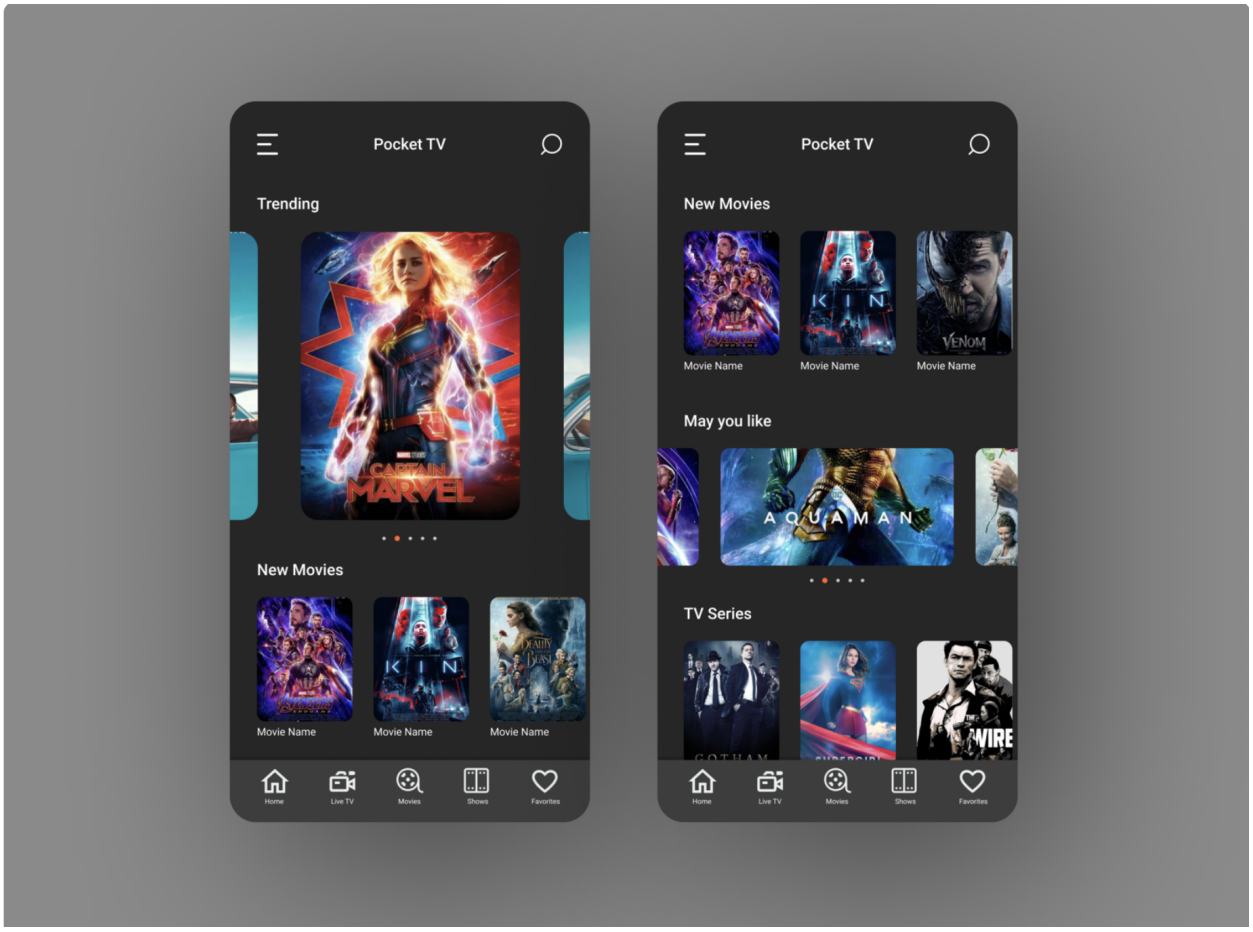
20. Explore Movies

Make a new tab in tabbar, named “Explore”, icon “sparkle.magnifyingglass”. Tabs order will be: All Movies, Explore, Favorites, Settings.
This new screen will contain several sections:

- Latest **Latest movies API**
- Now Playing **Now Playing movies API**
- Popular **Popular movies API**
- Top Rated **Top Rated movies API**
- Upcoming **Upcoming movies API**

Sections will have different sizes, as shown below. The collection view should be reusable, only the section name and the item size differs.
Sections should have infinite scrolling, meaning it will request the next page/batch of movies as we scroll.

Inspiration:



21. Favorites + Core Data

https://www.raywenderlich.com/10794954-beginning-core-data
https://www.raywenderlich.com/3815-intermediate-core-data

Add Core Data into the Movies App. Create entities for all the model objects that are going to be saved as Favorites (ex. the Movie).
Change Favorites screen ‘s datasource to use CoreData instead of the existing caching mechanism you are currently using (json files, etc). CoreData will be used also for sorting and searching through Favorites.

22. Unit testing

Implement unit tests for:

- Data sources
- Utility classes
- Extensions
- Managers
- Services
- Other business logic classes

23. Cocoapods

https://www.raywenderlich.com/7076593-cocoapods-tutorial-for-swift-getting-started

Choose a third party component to install via Cocoapods.

- Static code analysis
  - SwiftLint: install the tool and solve the warnings.
- Networking layer
  - Alamofire
  - Moya
- Downloading images
  - SDWebImage
  - AlamofireImage
- Star rating widget
  - Choose one from Github or search on https://cocoapods.org
  - Pay attention to how the widget looks like, how easy it is to implement and integrate into this app, if it has recent updates (code changes), if it supports Swift 5+ and iOS 14+, if it has too many issues (bugs) opened, etc.
- etc.

24.

==========

Design Patterns

1. MVC

Use MVC for building Movies app's All Movies, Movie Details and Favorites screens.

2. Delegate

Use Delegate pattern for indirect communication from views or cells to parent view controllers.

3. Notifications

Use NotificationCenter to notify that a favorite movie has changed its status.

4. Singleton

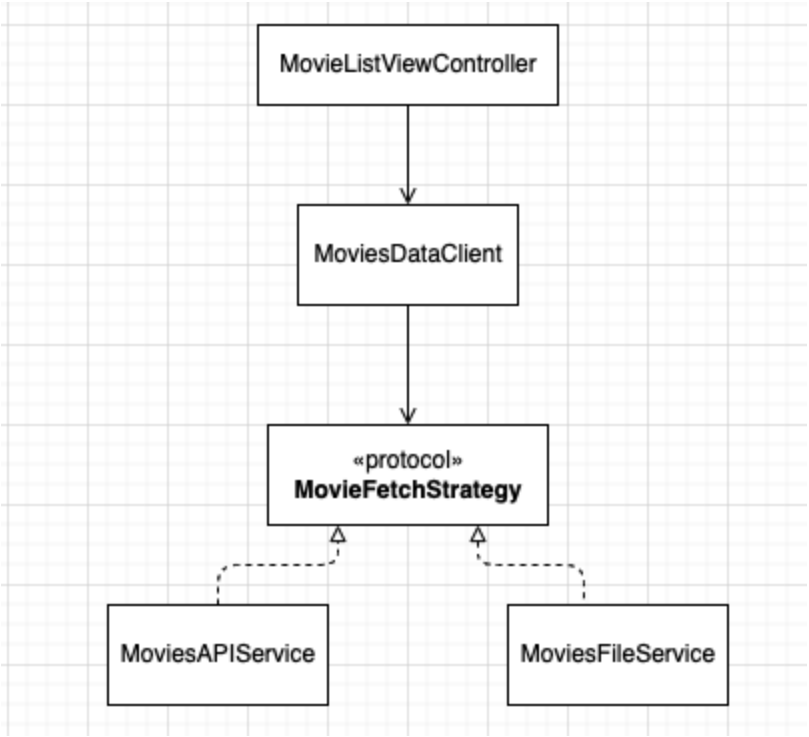
Identify where you would need a Singleton in the app.

5. Strategy Pattern

Use Strategy pattern in MoviesDataClient. **MoviesDataClient** will use a **MovieFetchStrategy** (protocol) that will help fetch movies.
MovieFetchStrategy will be implemented by

- **MoviesAPIService** - uses themoviedb API to fetch movies from server
- **MoviesFileService** - uses FileManager to fetch movies from jsons saved on disk

MovieFetchStrategy definition:
- func that fetches movies, uses pagination, and are sorted by an attribute (like it is already implemented, no changes here)



Notes:

- MoviesAPIService and MoviesFileService can make other things besides implementing MovieFetchStrategy. ex. MoviesFileService will have ability to save data to jsons.
- Most probably you cannot keep the "caching functionality" in the new implementation. It is ok, but you can still try to think of a way to keep it, maybe after you've implemented the Strategy pattern.
- You can still think about having a singleton MoviesManager that uses the two clients, and handles other details (for ex. caching)

AllMovies VC --- <uses a> ---> MoviesManager --- <uses a> ---> MoviesDataClient --- <has a> ---> MoviesAPIService strategy.
Favorites VC --- <uses a> ---> MoviesManager --- <uses a> ---> MoviesDataClient --- <has a> ---> MoviesFileService strategy.

6. Observer

Based on the Observer pattern, use Publishers to update the heart icon on Movie Details when a movie has changed its favorite status. You can do this for updating the heart icon on the movie cells as well.

7. MVVM

Instead of MVC, try to migrate existing views to using MVVM pattern

8. Memento

Implement Memento pattern for saving Favorites on the device.

9. Builder

- https://www.raywenderlich.com/books/design-patterns-by-tutorials/v3.0/chapters/9-builder-pattern

10. Factory

- https://www.raywenderlich.com/books/design-patterns-by-tutorials/v3.0/chapters/11-factory-pattern

11. Adapter

- https://www.raywenderlich.com/books/design-patterns-by-tutorials/v3.0/chapters/12-adapter-pattern

12. ff

==========

Notes

General notes:

- For icons use: **https://sfsymbols.com**
- For fonts use native System Font (default)
- App layout should be compatible with all screen sizes
- Support for landscape mode
- Minimum supported version: iOS 14.0

User interactions:

When the user taps on a movie row you need to open a new screen which will show the movie's details:

- the movie poster
- the rating
- the movie description
- other relevant info that comes from the API.

API specs:

Api documentation: https://developers.themoviedb.org/3/getting-started

API key: 626d05abf324b3be1c089c695497d49c

1. List of movies API
    a. API spec: https://developers.themoviedb.org/3/discover/movie-discover
    b. Sample: **GET** https://api.themoviedb.org/3/discover/movie?api_key=626d05abf324b3be1c089c695497d49c
2. Movie details API
    a. API spec: https://developers.themoviedb.org/3/movies/get-movie-details
    b. Sample: **GET** https://api.themoviedb.org/3/movie/211672?api_key=626d05abf324b3be1c089c695497d49c
    c. Format: https://api.themoviedb.org/3/movie/ <movie_id> ?api_key=626d05abf324b3be1c089c695497d49c
3. Similar movies API
    a. API spec: https://developers.themoviedb.org/3/movies/get-similar-movies
    b. Sample: **GET** https://api.themoviedb.org/3/movie/211672/similar?api_key=626d05abf324b3be1c089c695497d49c
    c. Format: https://api.themoviedb.org/3/movie/ <movie_id> / similar?api_key=626d05abf324b3be1c089c695497d49c

4. Paginated list of movies API
    a. API spec: https://developers.themoviedb.org/3/discover/movie-discover
    b. Sample: **GET** https://api.themoviedb.org/3/discover/movie?api_key=626d05abf324b3be1c089c695497d49c&page=2
5. Paginated list of similar movies API
    a. API spec: https://developers.themoviedb.org/3/movies/get-similar-movies
    b. Sample: **GET** https://api.themoviedb.org/3/movie/211672/similar?api_key=626d05abf324b3be1c089c695497d49c&page=2
6. Movie videos API
    a. API spec: https://developers.themoviedb.org/3/movies/get-movie-videos
    b. Sample: **GET** https://api.themoviedb.org/3/movie/460458/videos?api_key=626d05abf324b3be1c089c695497d49c
    c. Check if *site* == “YouTube”
    d. Format: https://www.youtube.com/watch?v=<key_from_response> OR https://youtu.be/<key_from_response>
    e. Sample: https://www.youtube.com/watch?v=4q6UGCyHZCI OR https://youtu.be/IQqqAWMIIAQ
7. Images API
    a. API spec: https://developers.themoviedb.org/3/getting-started/images
    b. Sample: **GET** https://image.tmdb.org/t/p/w500/kqjL17yufvn9OVLyXYpvtyrFfak.jpg
    c. Format: https://image.tmdb.org/t/p/ <resolution> / <poster_path_from_response>
8. Search API
    a. API spec: https://developers.themoviedb.org/3/search/search-movies
    b. Format: https://api.themoviedb.org/3/search/movie?api_key=626d05abf324b3be1c089c695497d49c&language=en-US&query=<search_string>&page=<next_page_number>&include_adult=false
    c. Sample: **GET** https://api.themoviedb.org/3/search/movie?api_key=626d05abf324b3be1c089c695497d49c&language=en-US&query=spider&page=1&include_adult=false
9. Search Keywords API
    a. API spec: https://developers.themoviedb.org/3/search/search-keywords
    b. Sample: **GET** https://api.themoviedb.org/3/search/keyword?api_key=626d05abf324b3be1c089c695497d49c&query=wid&page=1

10. Authentication API

a. API spec: https://developers.themoviedb.org/3/authentication/how-do-i-generate-a-session-id

b. **Step 1:** Create a request token

i. API spec: https://developers.themoviedb.org/3/authentication/create-request-token

ii. Example: `GET` https://api.themoviedb.org/3/authentication/token/new?api_key=626d05abf324b3be1c089c695497d49c

iii. Success:

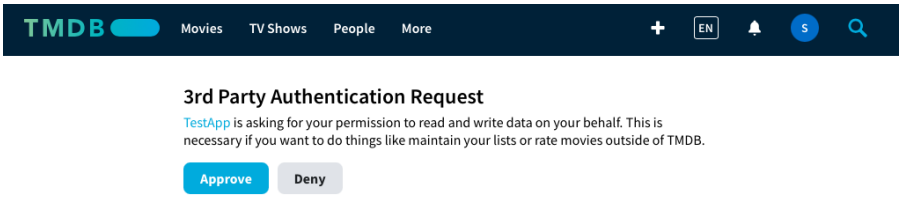
`200 OK`

```
{
   "success": true,
   "expires_at": "2022-01-26 17:04:57 UTC",
   "request_token": "11c06a6071f0ce910001552497d06d4a1da05465"
}
```

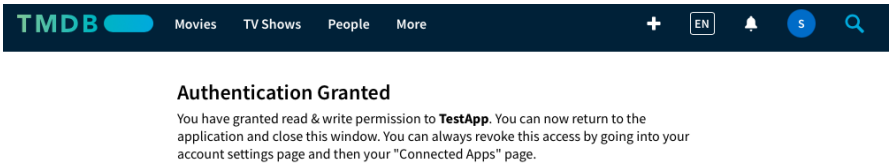
c. **Step 2:** Validate the token (2 options!!)

i. Open the TheMovieDB web page where the user will authenticate and approve the permissions. After permission is granted user will manually go back to the app, and the app must check if it needs to do Step 3 (it has an unexpired request token, but it does not have yet a session ID). More advanced option would be to implement **deeplinks** in the app and set a `redirect_to` query parameter in the URL , that will open when the user approves the permissions in the webpage.

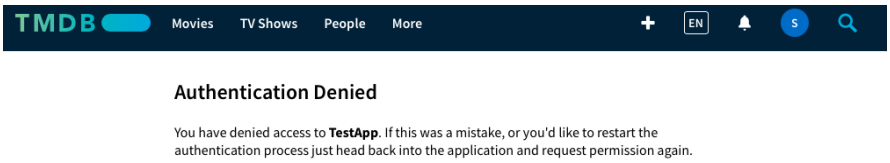
1. https://www.themoviedb.org/authenticate/{REQUEST_TOKEN}
2. Sample: https://www.themoviedb.org/authenticate/8a25d70e344fc6057e5fff395a826beaab715579



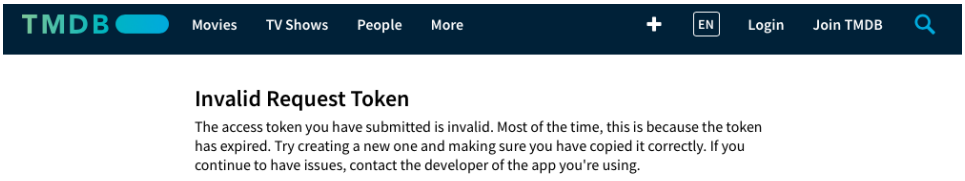
3. Approve: In same webpage user is automatically redirected to https://www.themoviedb.org/authenticate/11c06a6071f0ce910001552497d06d4a1da05465/allow



4. Deny: In same webpage user is automatically redirected to https://www.themoviedb.org/authenticate/f6624ed6267f680f64ef3f6eba5dc69ab95ca956/deny



5. Error:



ii. Use your own Login page with username and password inputs https://developers.themoviedb.org/3/authentication/validate-request-token

1. Sample: POST
   https://api.themoviedb.org/3/authentication/token/validate_with_login?api_key=626d05abf324b3be1c089c695497d49c

```
{ "username": "silvia.cret",
  "password": "TheMovieDB",
  "request_token":
"d55387394cf3c0ab5635d75307a13cce712e20d0" }
```

2. Success:

```
200 OK
{  "success": true,
   "expires_at": "2022-01-26 16:43:48 UTC",
   "request_token":
"d55387394cf3c0ab5635d75307a13cce712e20d0" }
```

3. Error case: Wrong credentials

```
401 Unauthorized
{  "success": false,
```

```
"status_code": 30,
"status_message": "Invalid username and/or password: You did not provide a valid login." }
```


d. **Step 3**: Create a session ID
   i. API spec: https://developers.themoviedb.org/3/authentication/create-session
   ii. Sample: **POST** https://api.themoviedb.org/3/authentication/session/new?api_key=626d05abf324b3be1c089c695497d49c

```
{"request_token": "6bc047b88f669d1fb86574f06381005d93d3517a"}
```

   iii. Success:

```
200 OK
{ "success": true,
  "session_id": "4591a98392bdd6dfec331913cca6aee6a2a9f855" }
```

   iv. Error case: User DENIED the permissions in Step 2, but you still try to do Step 3

```
401 Unauthorized
{ "success": false,
  "failure": true,
  "status_code": 17,
  "status_message": "Session denied."
}
```

   v. Error case: Invalid token

```
200 OK
{ "success": false,
  "status_code": 6,
  "status_message": "Invalid id: The pre-requisite id is invalid or not found." }
```


11. Get Account Details API

a. API spec: https://developers.themoviedb.org/3/account/get-account-details
b. Format: **GET** https://api.themoviedb.org/3/account?api_key=626d05abf324b3be1c089c695497d49c&session_id=<session_id>

12. Rate movie API

a. API spec: https://developers.themoviedb.org/3/movies/rate-movie
b. Sample: **POST** https://api.themoviedb.org/3/movie/211672/rating?api_key=626d05abf324b3be1c089c695497d49c&session_id=4591a98392bdd6dfec331913cca6aee6a2a9f855
c. Success:

201 Created


```
{  "success": true,
   "status_code": 1,
   "status_message": "Success." }
```


d. Repeating the action, Success:

201 Created


```
{ "success": true,
   "status_code": 12,
   "status_message": "The item/record was updated successfully." }
```


e. Error case: No session ID / Wrong session ID

401 Unauthorized


```
{ "success": false,
   "status_code": 3,
   "status_message": "Authentication failed: You do not have permissions
to access the service." }
```


13. Reviews API
    a. API spec: https://developers.themoviedb.org/3/movies/get-movie-reviews
14. Genre filtering API
    a. API spec: https://developers.themoviedb.org/3/discover/movie-discover
    b. Sample: **GET** https://api.themoviedb.org/3/discover/movie?api_key=626d05abf324b3be1c089c695497d49c&language=en-US&sort_by=popularity.desc&page=1&with_genres=878
    c. Format: https://api.themoviedb.org/3/discover/movie?api_key=626d05abf324b3be1c089c695497d49c&with_genres=<comma_separated_genre_ids>
15. Latest movies API
    a. API spec: https://developers.themoviedb.org/3/movies/get-latest-movie
    b. Sample: **GET** https://api.themoviedb.org/3/movie/latest?api_key=626d05abf324b3be1c089c695497d49c&language=en-US
16. Now Playing movies API
    a. API spec: https://developers.themoviedb.org/3/movies/get-now-playing
    b. Sample: **GET** https://api.themoviedb.org/3/movie/now_playing?api_key=626d05abf324b3be1c089c695497d49c&language=en-US&page=1
17. Popular movies API
    a. API spec: https://developers.themoviedb.org/3/movies/get-popular-movies

b. Sample: **GET** https://api.themoviedb.org/3/movie/popular?api_key=626d05abf324b3be1c089c695497d49c&language=en-US&page=1

18. Top Rated movies API
    a. API spec: https://developers.themoviedb.org/3/movies/get-top-rated-movies
    b. Sample: **GET** https://api.themoviedb.org/3/movie/top_rated?api_key=626d05abf324b3be1c089c695497d49c&language=en-US&page=1
19. Upcoming movies API
    a. API spec: https://developers.themoviedb.org/3/movies/get-upcoming
    b. Sample: **GET** https://api.themoviedb.org/3/movie/upcoming?api_key=626d05abf324b3be1c089c695497d49c&language=en-US&page=1