Last edited by **Luis Eduardo Perez** 6 months ago

Sprint 4 – Deliverable

Value Proposition

- The app considers the user's interest, viewing, and search information, making movies and tv shows recommendations related to the user's search. This implies a more targeted experience in a more seamless manner, without having to depend on a second human opinion for a movie or tv show.
- The app's catalog of movies is dynamic, depending on the user's watched and liked movies.
- Common preferences in movies and TV shows may be found within a group of people.
- The app suggests movies or TV shows that he or she might like but that is not related to what he or she has seen.
- The app will collect information of how many time the users spend on each activity so in this way we can know were to use ads.
- The app register the countries where people access the app so this will work if we want to create personalized ads depending on each country.
- Users could search upcoming movies in our application.
- Users can know which movies they had liked so that the could select a movie to watch in the future.

Proposed revenue model

For the Movies revenue model we will have two principal ways for obtaining money. In the first place we will do advertisements in the app, we will use Google AdMob and it's estimated that if the app has around millions of downloads we could earn an estimated of $ 2.000 USD Daily. Another way to monetize the app would be to offer the app to different video streaming companies such as Netflix, Amazon Prime, Disney plus, and more. So in this way the users can search movies provided by each streaming company and for every person that goes to watch a movie to each streaming video platform we will charge a fee for taking clients to their page.

The google Play store charges a single payment of $25 USD for publishing an app on their store. On another hand the App Store from Apple charges $99 USD for publishing an app on their store an this payment has to be renewed annually.

In Colombia the price for developing an app varies from different ranges.

- Basic App Between $20.000.000 COP and $50.000.000 COP
- Moderate App Between $50.000.000 COP and $120.000.000 COP
- complex App Between $120.000.000 COP and $250.000.000 COP

You also have to take in mind that cross platform apps are less expensive than native apps and because our team will be working on Flutter and Kotlin and both frameworks are for cross-platform mobile apps this can reduce the costs of the project.

Regarding the costs of maintenance there's a lot of factors that can affect these costs so we will describe some of these costs that we will have to keep in mind.

- Costs of the servers where the app is allocated 20-60 USD monthly.
- Costs of Databases where the information is storage (Depend on the amount of data that will be storage)
- Code Modifications due to bugs, new updates or updates on the IOS or Android software. (Depending on how many modifications will the app have).
- Charges due to App analytics.
- Changes on the technologies used, for example if we are using mongo DB and they do a change in their platform maybe we have to do changes as well.
- Unpredictable additional events.

Micro-optimization Strategies

Flutter

Performance Before micro-optimization implementation

Micro-optimization implementation

Performance After micro-optimization implementation

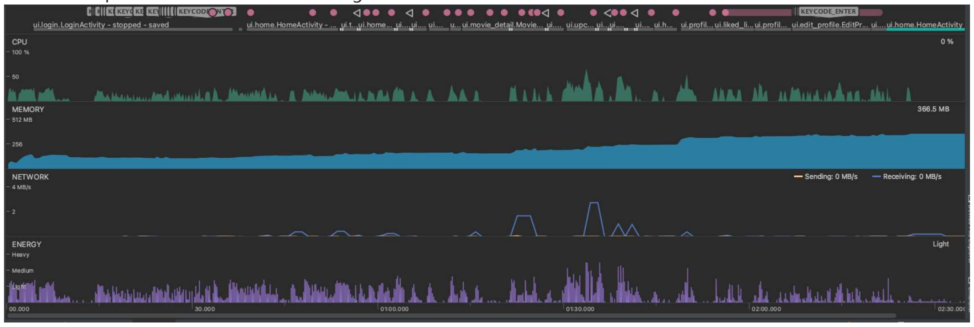
Kotlin

Performance Before micro-optimization implementation

Video Before

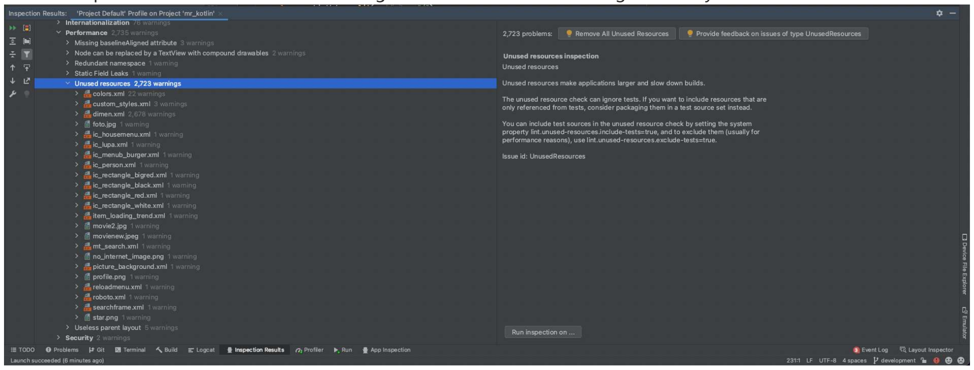
first we make a video of the actions that where executed which is this one link:

the android profiler tool show the following answers

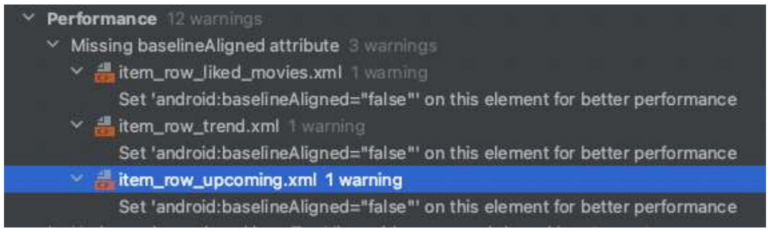


Micro-optimization implementation

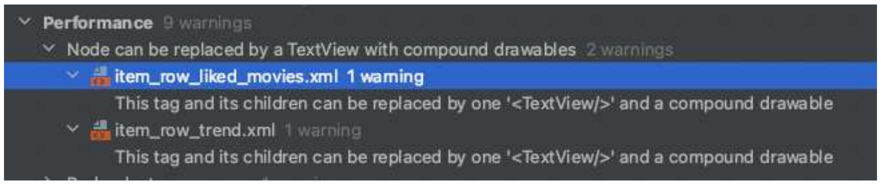
For microoptimizations we based on removing unused resources and change some layouts



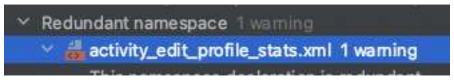
```
Inspection Results:   'Project Default' Profile on Project 'mr_kotlin'
  > Internationalization 76 warnings
  v Performance 2,735 warnings
    > Missing baselineAligned attribute 3 warnings
    > Node can be replaced by a TextView with compound drawables 2 warnings
    > Redundant namespace 1 warning
    > Static Field Leaks 1 warning
    v Unused resources 2,723 warnings
      > colors.xml 22 warnings
      > custom_styles.xml 3 warnings
      > dimen.xml 2,678 warnings
      > foto.jpg 1 warning
      > ic_housemenu.xml 1 warning
      > ic_lupa.xml 1 warning
      > ic_menub_burger.xml 1 warning
      > ic_person.xml 1 warning
      > ic_rectangle_bigred.xml 1 warning
      > ic_rectangle_black.xml 1 warning
      > ic_rectangle_red.xml 1 warning
      > ic_rectangle_white.xml 1 warning
      > item_loading_trend.xml 1 warning
      > movie2.jpg 1 warning
      > movienew.jpeg 1 warning
      > mt_search.xml 1 warning
      > no_internet_image.png 1 warning
      > picture_background.xml 1 warning
      > profile.png 1 warning
      > reloadmenu.xml 1 warning
      > roboto.xml 1 warning
      > searchframe.xml 1 warning
      > star.png 1 warning
    > Useless parent layout 5 warnings
  > Security 2 warnings

2,723 problems:  Remove All Unused Resources  Provide feedback on issues of type UnusedResources

Unused resources inspection
Unused resources

Unused resources make applications larger and slow down builds.

The unused resource check can ignore tests. If you want to include resources that are
only referenced from tests, consider packaging them in a test source set instead.

You can include test sources in the unused resource check by setting the system
property lint.unused-resources.include-tests=true, and to exclude them (usually for
performance reasons), use lint.unused-resources.exclude-tests=true.

Issue id: UnusedResources

Run inspection on ...

TODO  Problems  Git  Terminal  Build  Logcat  Inspection Results  Profiler  Run  App Inspection      Event Log  Layout Inspector
Launch succeeded (6 minutes ago)                      231:1  LF  UTF-8  4 spaces  development
```



```
v Performance 12 warnings
  v Missing baselineAligned attribute 3 warnings
    v item_row_liked_movies.xml 1 warning
        Set 'android:baselineAligned="false"' on this element for better performance
    v item_row_trend.xml 1 warning
        Set 'android:baselineAligned="false"' on this element for better performance
    v item_row_upcoming.xml 1 warning
        Set 'android:baselineAligned="false"' on this element for better performance
```

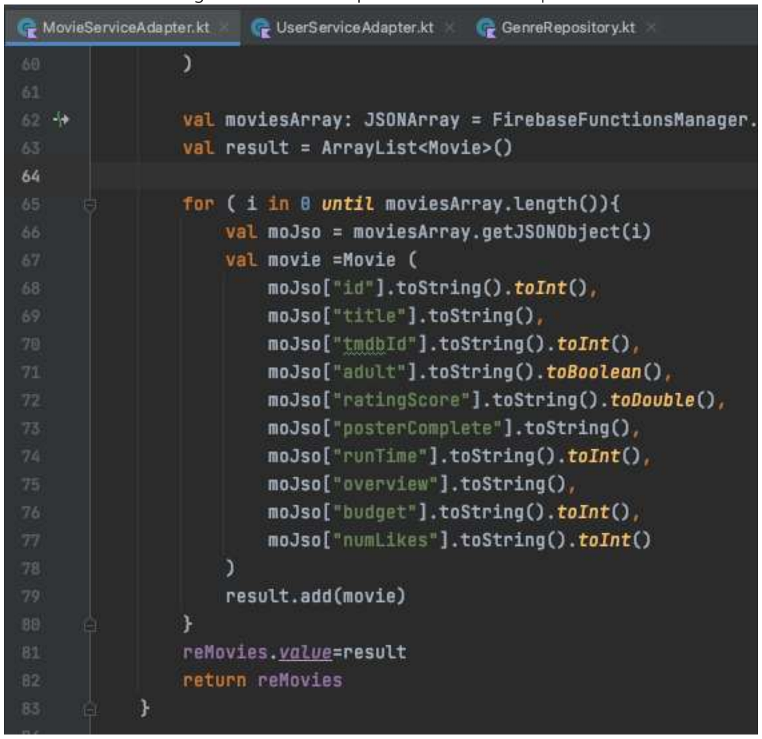


```
v Performance 9 warnings
  v Node can be replaced by a TextView with compound drawables 2 warnings
    v item_row_liked_movies.xml 1 warning
        This tag and its children can be replaced by one '<TextView/>' and a compound drawable
    v item_row_trend.xml 1 warning
        This tag and its children can be replaced by one '<TextView/>' and a compound drawable
```

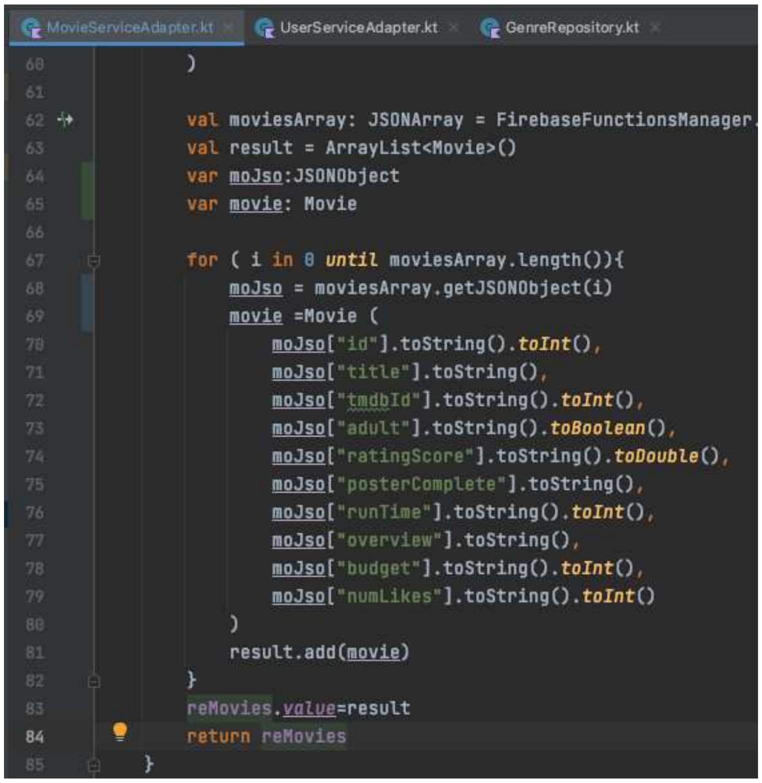


```
v Redundant namespace 1 warning
  v activity_edit_profile_stats.xml 1 warning
```

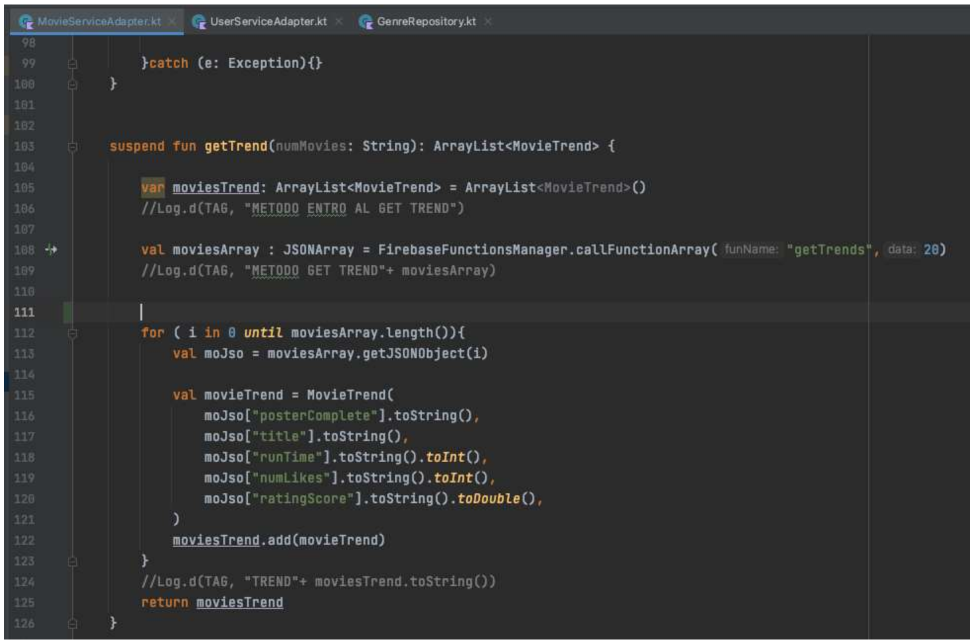
The methods that change to make micro-optimizations in the loop



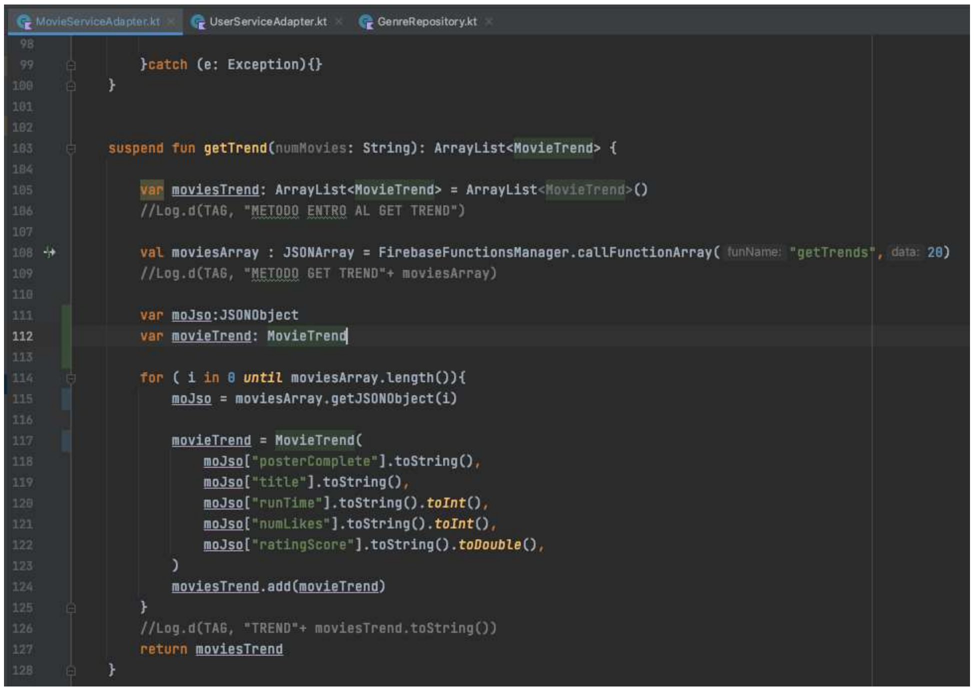
```kotlin
        )

        val moviesArray: JSONArray = FirebaseFunctionsManager.
        val result = ArrayList<Movie>()

        for ( i in 0 until moviesArray.length()){
            val moJso = moviesArray.getJSONObject(i)
            val movie =Movie (
                moJso["id"].toString().toInt(),
                moJso["title"].toString(),
                moJso["tmdbId"].toString().toInt(),
                moJso["adult"].toString().toBoolean(),
                moJso["ratingScore"].toString().toDouble(),
                moJso["posterComplete"].toString(),
                moJso["runTime"].toString().toInt(),
                moJso["overview"].toString(),
                moJso["budget"].toString().toInt(),
                moJso["numLikes"].toString().toInt()
            )
            result.add(movie)
        }
        reMovies.value=result
        return reMovies
    }
```



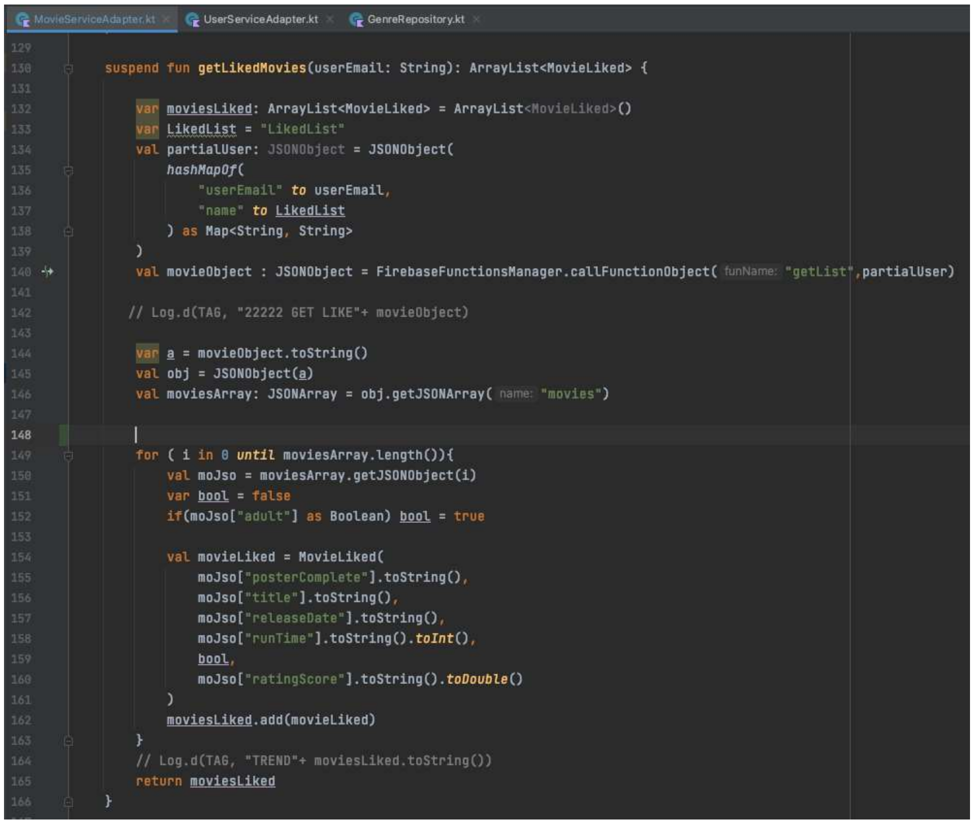
```kotlin
        )

        val moviesArray: JSONArray = FirebaseFunctionsManager.
        val result = ArrayList<Movie>()
        var moJso:JSONObject
        var movie: Movie

        for ( i in 0 until moviesArray.length()){
            moJso = moviesArray.getJSONObject(i)
            movie =Movie (
                moJso["id"].toString().toInt(),
                moJso["title"].toString(),
                moJso["tmdbId"].toString().toInt(),
                moJso["adult"].toString().toBoolean(),
                moJso["ratingScore"].toString().toDouble(),
                moJso["posterComplete"].toString(),
                moJso["runTime"].toString().toInt(),
                moJso["overview"].toString(),
                moJso["budget"].toString().toInt(),
                moJso["numLikes"].toString().toInt()
            )
            result.add(movie)
        }
        reMovies.value=result
        return reMovies
    }
```



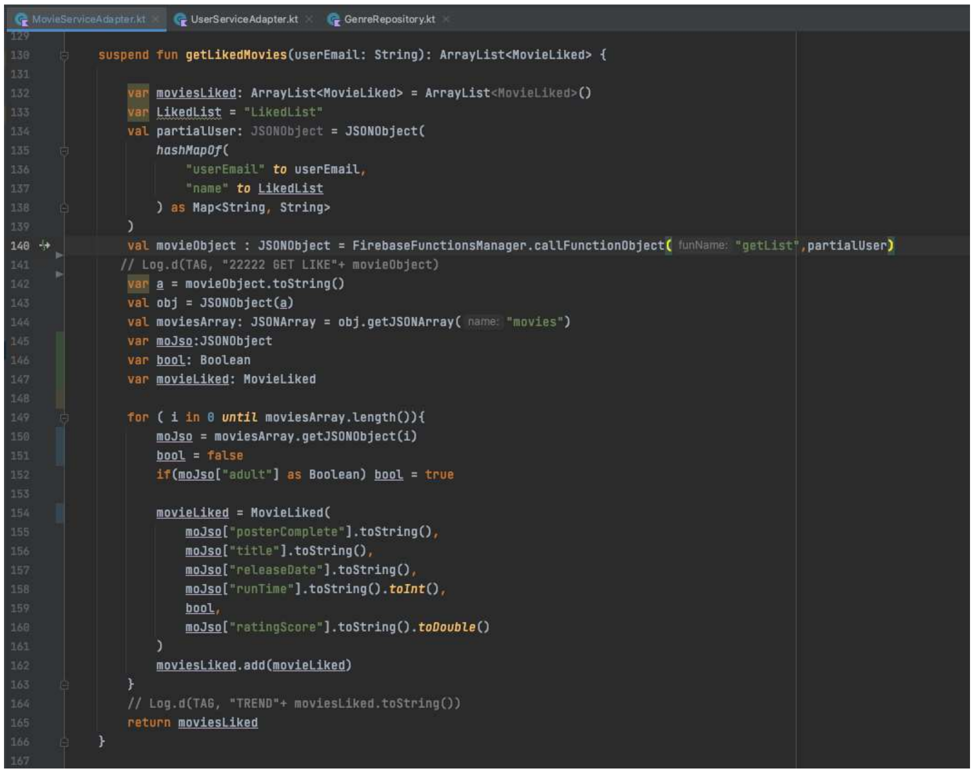
```kotlin

        }catch (e: Exception){}
    }


    suspend fun getTrend(numMovies: String): ArrayList<MovieTrend> {

        var moviesTrend: ArrayList<MovieTrend> = ArrayList<MovieTrend>()
        //Log.d(TAG, "METODO ENTRO AL GET TREND")

        val moviesArray : JSONArray = FirebaseFunctionsManager.callFunctionArray( funName: "getTrends", data: 20)
        //Log.d(TAG, "METODO GET TREND"+ moviesArray)


        for ( i in 0 until moviesArray.length()){
            val moJso = moviesArray.getJSONObject(i)

            val movieTrend = MovieTrend(
                moJso["posterComplete"].toString(),
                moJso["title"].toString(),
                moJso["runTime"].toString().toInt(),
                moJso["numLikes"].toString().toInt(),
                moJso["ratingScore"].toString().toDouble(),
            )
            moviesTrend.add(movieTrend)
        }
        //Log.d(TAG, "TREND"+ moviesTrend.toString())
        return moviesTrend
    }
```



```kotlin

        }catch (e: Exception){}
    }


    suspend fun getTrend(numMovies: String): ArrayList<MovieTrend> {

        var moviesTrend: ArrayList<MovieTrend> = ArrayList<MovieTrend>()
        //Log.d(TAG, "METODO ENTRO AL GET TREND")

        val moviesArray : JSONArray = FirebaseFunctionsManager.callFunctionArray( funName: "getTrends", data: 20)
        //Log.d(TAG, "METODO GET TREND"+ moviesArray)

        var moJso:JSONObject
        var movieTrend: MovieTrend

        for ( i in 0 until moviesArray.length()){
            moJso = moviesArray.getJSONObject(i)

            movieTrend = MovieTrend(
                moJso["posterComplete"].toString(),
                moJso["title"].toString(),
                moJso["runTime"].toString().toInt(),
                moJso["numLikes"].toString().toInt(),
                moJso["ratingScore"].toString().toDouble(),
            )
            moviesTrend.add(movieTrend)
        }
        //Log.d(TAG, "TREND"+ moviesTrend.toString())
        return moviesTrend
    }
```



```
    suspend fun getLikedMovies(userEmail: String): ArrayList<MovieLiked> {

        var moviesLiked: ArrayList<MovieLiked> = ArrayList<MovieLiked>()
        var LikedList = "LikedList"
        val partialUser: JSONObject = JSONObject(
            hashMapOf(
                "userEmail" to userEmail,
                "name" to LikedList
            ) as Map<String, String>
        )
        val movieObject : JSONObject = FirebaseFunctionsManager.callFunctionObject( funName: "getList",partialUser)

       // Log.d(TAG, "22222 GET LIKE"+ movieObject)

        var a = movieObject.toString()
        val obj = JSONObject(a)
        val moviesArray: JSONArray = obj.getJSONArray( name: "movies")


        for ( i in 0 until moviesArray.length()){
            val moJso = moviesArray.getJSONObject(i)
            var bool = false
            if(moJso["adult"] as Boolean) bool = true

            val movieLiked = MovieLiked(
                moJso["posterComplete"].toString(),
                moJso["title"].toString(),
                moJso["releaseDate"].toString(),
                moJso["runTime"].toString().toInt(),
                bool,
                moJso["ratingScore"].toString().toDouble()
            )
            moviesLiked.add(movieLiked)
        }
        // Log.d(TAG, "TREND"+ moviesLiked.toString())
        return moviesLiked
    }
```

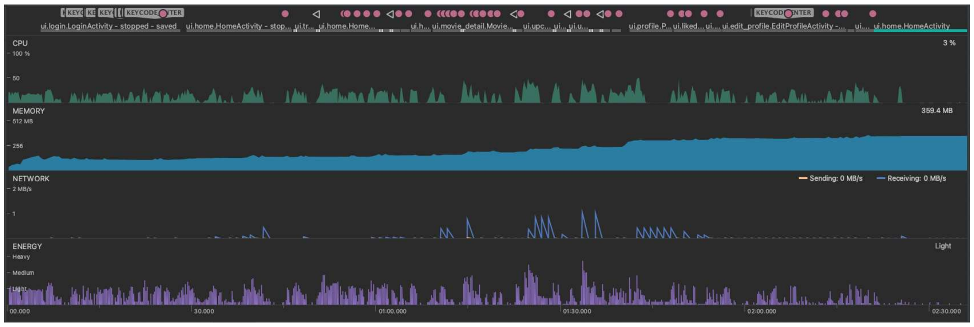


```
    suspend fun getLikedMovies(userEmail: String): ArrayList<MovieLiked> {

        var moviesLiked: ArrayList<MovieLiked> = ArrayList<MovieLiked>()
        var LikedList = "LikedList"
        val partialUser: JSONObject = JSONObject(
            hashMapOf(
                "userEmail" to userEmail,
                "name" to LikedList
            ) as Map<String, String>
        )
        val movieObject : JSONObject = FirebaseFunctionsManager.callFunctionObject( funName: "getList",partialUser)
       // Log.d(TAG, "22222 GET LIKE"+ movieObject)
        var a = movieObject.toString()
        val obj = JSONObject(a)
        val moviesArray: JSONArray = obj.getJSONArray( name: "movies")
        var moJso:JSONObject
        var bool: Boolean
        var movieLiked: MovieLiked

        for ( i in 0 until moviesArray.length()){
            moJso = moviesArray.getJSONObject(i)
            bool = false
            if(moJso["adult"] as Boolean) bool = true

            movieLiked = MovieLiked(
                moJso["posterComplete"].toString(),
                moJso["title"].toString(),
                moJso["releaseDate"].toString(),
                moJso["runTime"].toString().toInt(),
                bool,
                moJso["ratingScore"].toString().toDouble()
            )
            moviesLiked.add(movieLiked)
        }
        // Log.d(TAG, "TREND"+ moviesLiked.toString())
        return moviesLiked
    }
```

Performance After micro-optimization implementation

we show the video and the android profiler results after the micro-optimizations. while executing this methods we find that the app does its work faster.

Video After



Implemented features

Implemented features Sprint 2

- Smart Feature: A movie recommendation system has been partially implemented. This smart feature takes into account the movies that a specific user likes. This is visible by clicking the random button on the navbar.
- Authentication: The authentication system has been implemented using firebase. The app checks if a user is logged in the session, if not it will show the login and register screen, otherwise, the home screen will be shown
- External Service: The app communicates with firebase storage and function services. The storage is used to save lists, users, and likes. On the other hand, the functions are used to integrate the data stored in firebase storage with the recommendation system, which is hosted in an AWS EC2.

Implemented features Sprint 3

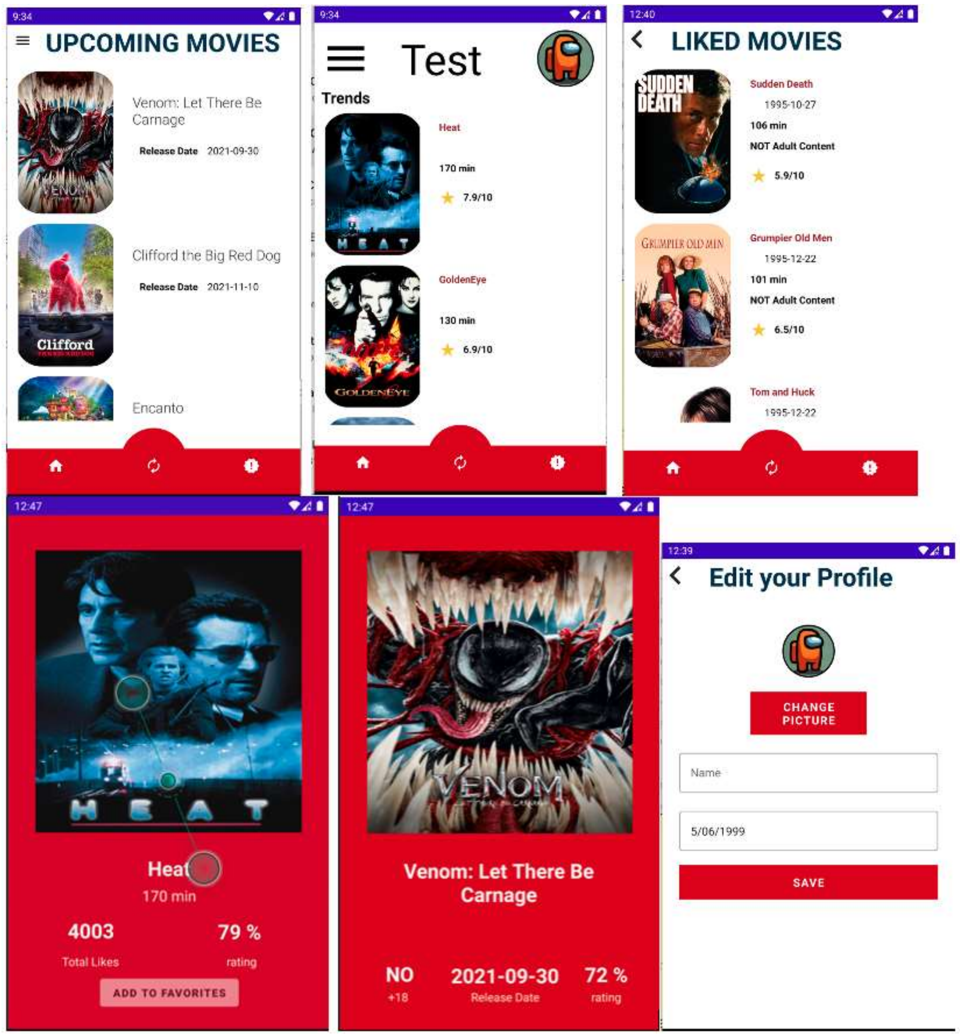
Implemented features Sprint 4

- Trends Feature: This functionality will show the movies that have more likes from the user.
- Upcoming feature: This feature will show in a scroll view some upcoming movies that that will come out soon.
- Notification on events: This functionality will send a notification every day to recommend a movie to user for him to watch.
- Delete Account: This feature will delete from the database the user credentials an all the information.
- Edit User: This feature will enable the user to change some certain attributes such as the name, profile picture and birthday.
- Liked List: This feature will show to the user all the movies he have give a like so that he could check this liked movies to watch.

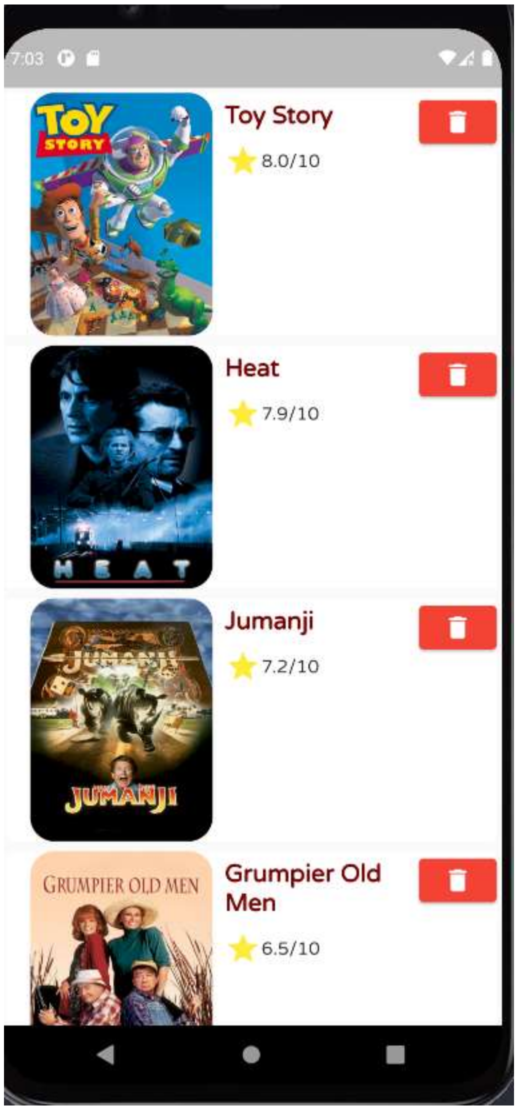
Implemented Views

- Trend movies view: The trend list is a list of movies trending on the app. This is based on the total number of likes the app's users have given the movies. The information shown to the user of each movie is Poster of the movies, title of the movie and the rating of the movie.
- Upcoming movies view: The upcoming movie list is a list of movie that are going to be in movie theaters. Therefore, the information given to the user is the movie's title, the poster and the release date of the movie.
- Trend Detail: The trend movie detail shows more information about the trend movie. It show the poster of the movie, the title, the duration of the movie, the amount of likes the movie has and the rating of the movie. On the bottom you can see a button that will add the movie to the favorites of the user.
- Upcoming Detail: The upcoming movie detail view shows more informacion about the soon to be released movie. The poster of the movie, the title, if the movie is suitable for minors, the release date, and the popularity of the movie.
- Edit Profile
- Notification Event
- Liked Movies

KOTLIN


UPCOMING MOVIES
Venom: Let There Be Carnage
Release Date 2021-09-30
Clifford the Big Red Dog
Release Date 2021-11-10
Encanto
Test
Trends
Heat
170 min
7.9/10
GoldenEye
130 min
6.9/10
LIKED MOVIES
Sudden Death
1995-10-27
106 min
NOT Adult Content
5.9/10
Grumpier Old Men
1995-12-22
101 min
NOT Adult Content
6.5/10
Tom and Huck
1995-12-22
Heat
170 min
4003
Total Likes
79 %
rating
ADD TO FAVORITES
Venom: Let There Be Carnage
NO
+18
2021-09-30
Release Date
72 %
rating
Edit your Profile
CHANGE PICTURE
Name
5/06/1999
SAVE

FLUTTER

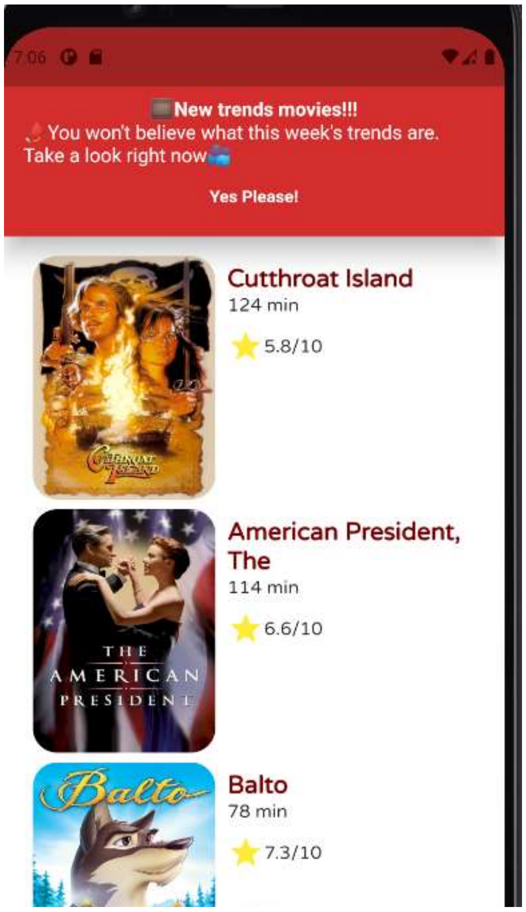

7:03

Toy Story
8.0/10

Heat
7.9/10

Jumanji
7.2/10

Grumpier Old Men
6.5/10


7:06

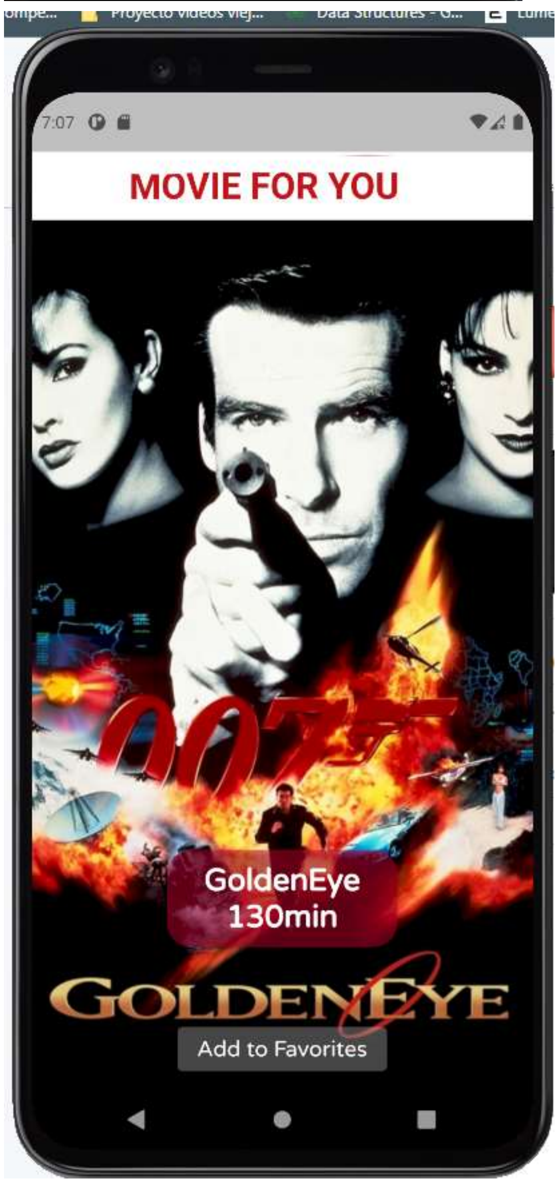
New trends movies!!!
You won't believe what this week's trends are. Take a look right now

Yes Please!

Cutthroat Island
124 min
5.8/10

American President, The
114 min
6.6/10

Balto
78 min
7.3/10



Business Questions

Business Questions Sprint 2

1. What is the percentage of the recommended movies liked by the user? [Type 3] (Juan Camilo Villamarin)
2. How long do users stay on the app?[Type 4] (Julian Oliveros Forero)
3. What is the distribution of genres liked by the user ? This will be presented on a Pie chart [type 2] (Luis Perez)
4. What are the top 5 genres mostly liked by the app's users? [Type 4] (Sebastian Baquero)
5. How many suggestions does the user pass before the first like on average? [type 3] (Kevin Blanco)
6. How much time does the system take to generate a recommendation, This will be presented in a scale by time [type 1] (Tobia Gasparoni)

Business Questions Sprint 3

1. Which is the most used feature by the users? [Type 3] (Juan Camilo Villamarin)
2. Which part of the code where the app constantly crashes? [Type 1] (Julian Oliveros Forero)
3. What is the distribution of genres liked by the user ? This will be presented on a Pie chart [type 2] (Luis Perez)
4. What are the top 5 genres of the most liked movies on the app? [Type 4] (Sebastian Baquero)
5. How many suggestions does the user pass before the first like on average? [type 3] (Kevin Blanco)

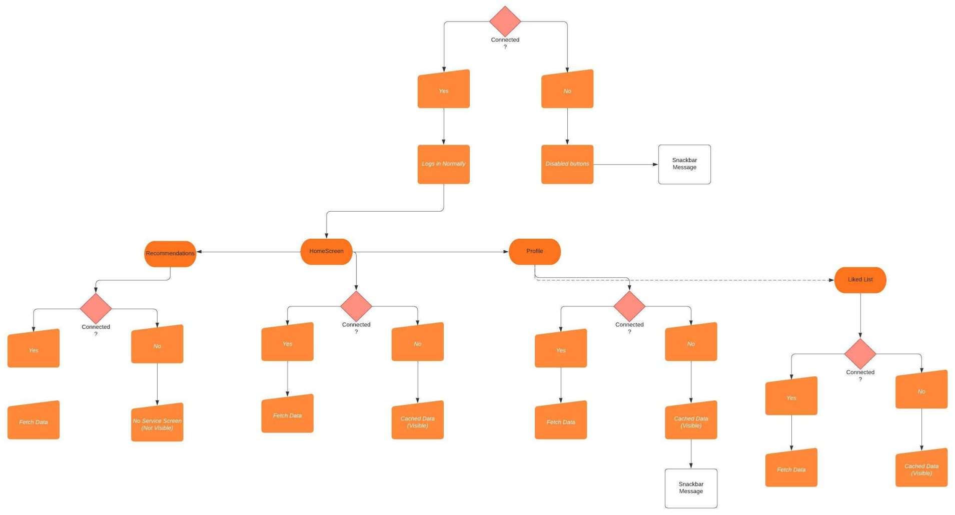
6. How much time does the system take to generate a recommendation, This will be presented in a scale by time [type 1] (Tobia Gasparoni)

Business Questions Sprint 4

1. How many notifications are opened by the user [Type 3] (Juan Camilo Villamarin)
2. Is the average time of startup the app greater than our goal (1s)? [Type 1] (Julian Oliveros Forero)
3. Views by Page title and screen class [type 2] (Luis Perez)
4. [Type 4] (Sebastian Baquero)
5. How many days have passed since the user last used the suggested movie feature? [type 2] (Kevin Blanco)
6. What is the weekly retention rate of the users in the app? [type 5] (Tobia Gasparoni)

Eventual connectivity strategy(ies).

Flutter



Local Storage Strategies

Flutter The User is Saved to local storage through the library cache_manager. It is saved as a json file and can be accessed with a key. The data persists even on activity deletion

Kotlin The User is Saved to local storage through sharedPreferenses.

Multi-threading Strategies

Flutter

1. Isolates are used to fetch the movies from the database in order to store them into a SQLITE database locally. Also asynchronous code, Futures are implemented to deliver all the information that is not local to the app, Users, Movies and images.

Kotlin

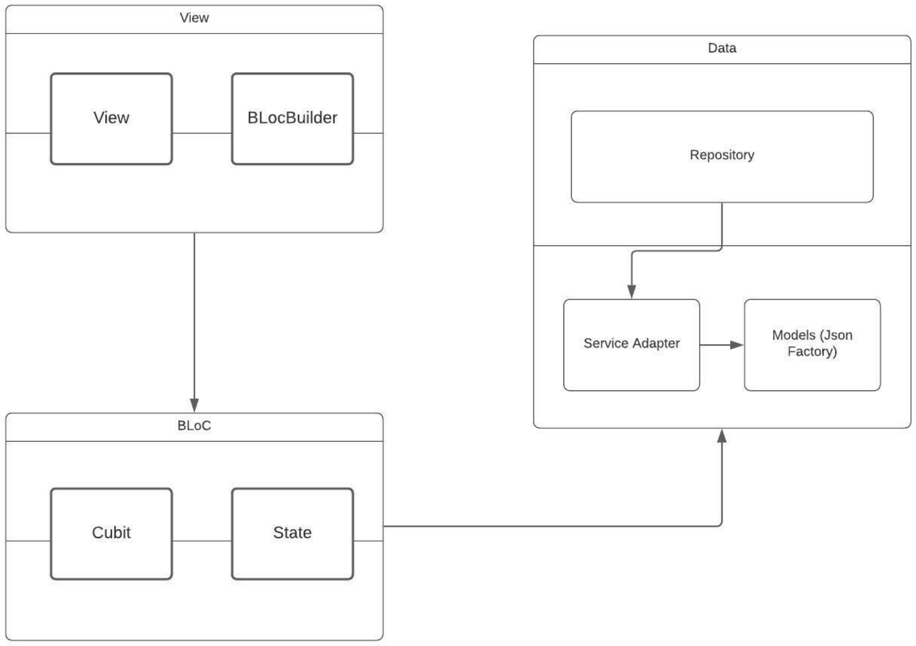
1. Implent of lifecyclescope to call all the methods that need to retrive information from firebase functions.
2. Most of our multi-threading strategies are implemented when the app has to do I/O operations, in our case these are implemented when the app needs information from the remote database which is requested through Firebase. We implement such functionality through the coroutines, a Kotlin solution to allow the execution of asynchronous code. The way it is used is with the async/await pattern, which allows other functions of the app to keep running while executing a potentially long-running task.

Caching Strategies

Kotlin

In order to cache the movie posters we are displaying on the app, we decided to use the image loading library Glide.

Flutter Streaming Platform Images are saved on cache while displaying movies, also the profile image, user, liked list and piechart data are stored and checked. The data is not fetched unless it is pulled by the user.


View
View
BLocBuilder
BLoC
Cubit
State
Data
Repository
Service Adapter
Models (Json Factory)