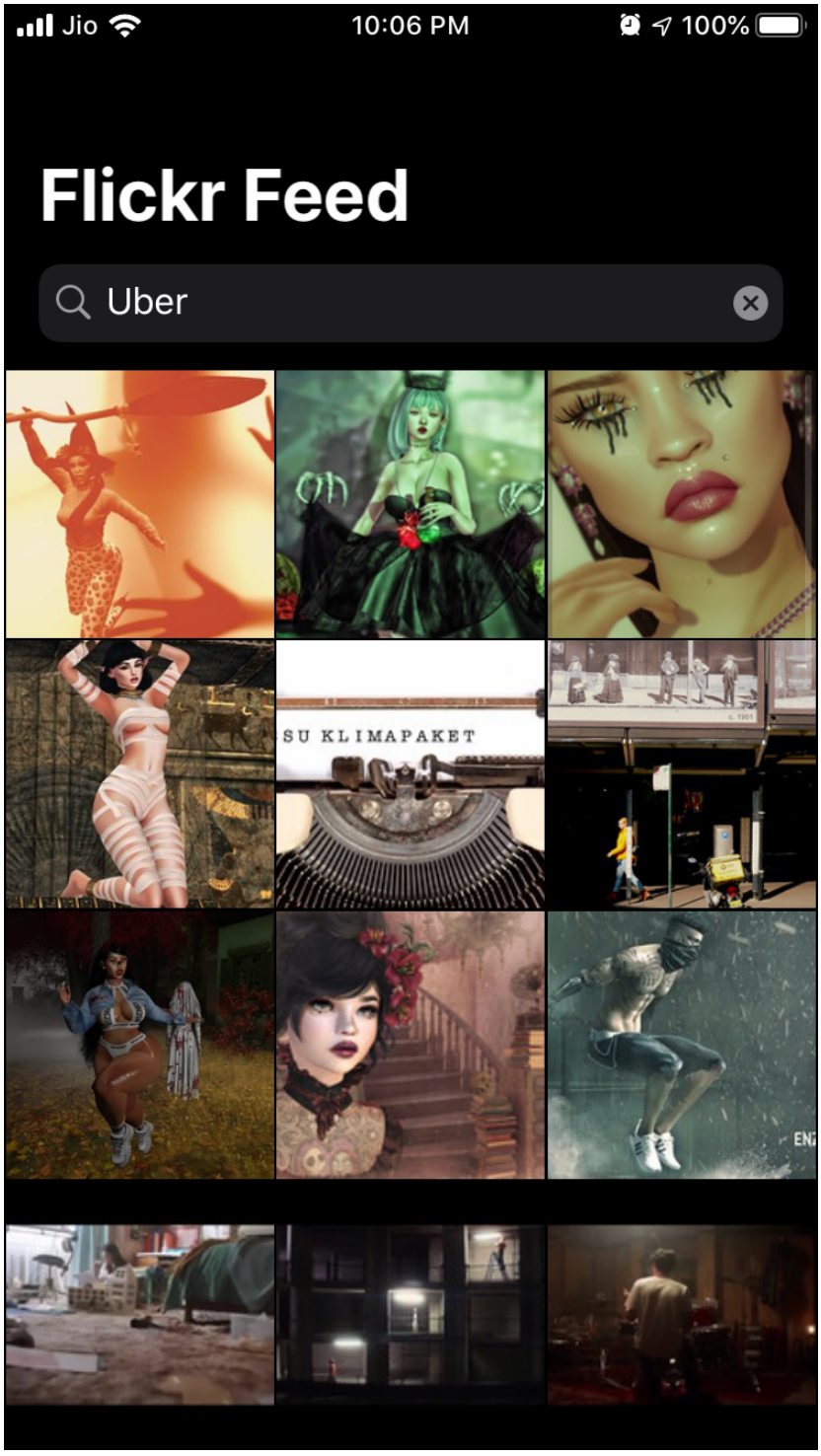
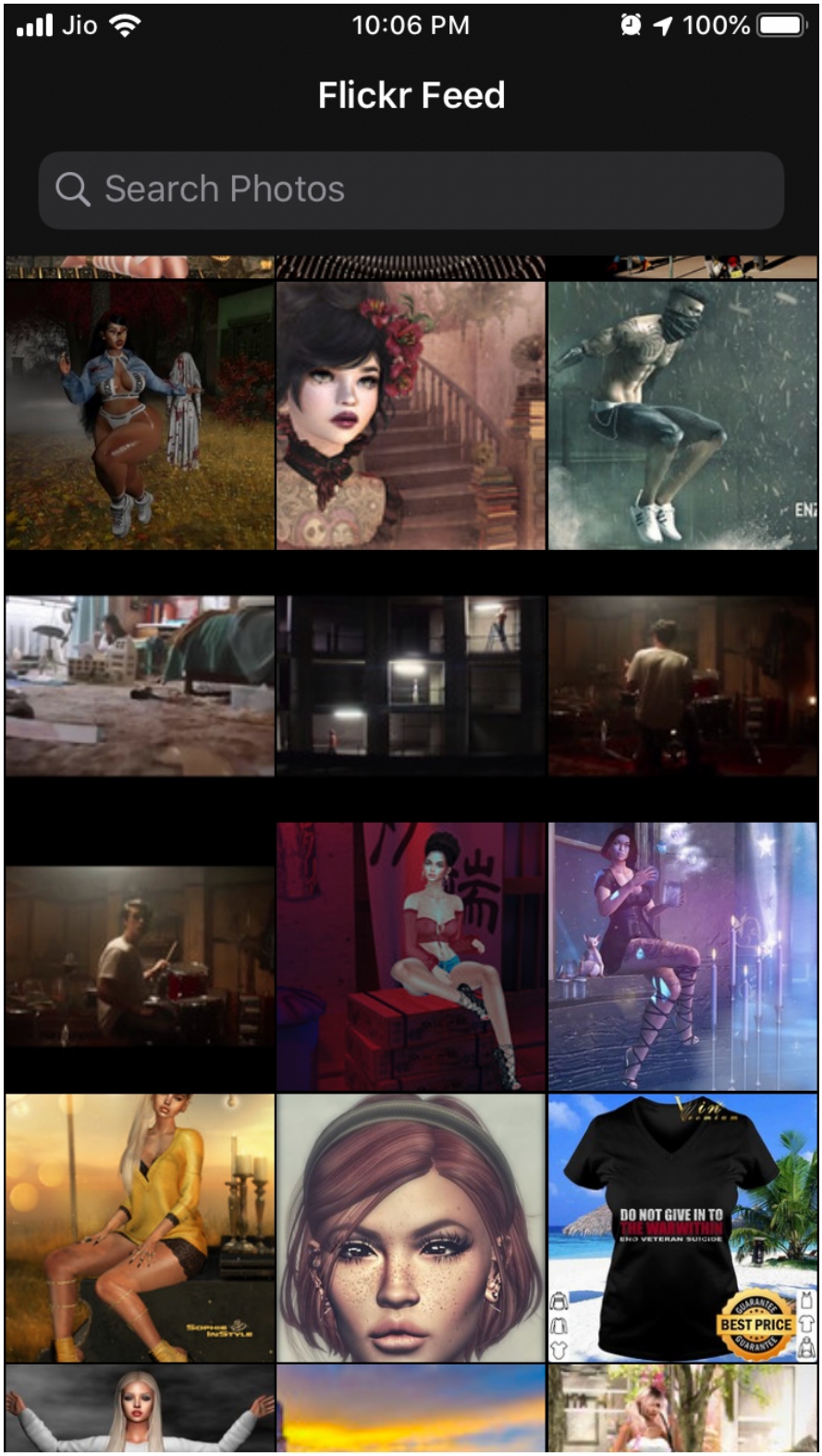
Flickr Feed App

Uber iOS Assignment Submission by Sudipta Sahoo

Phone: 8080842428
Email: me@sudiptasahoo.in





Project Details:

- Project Name: **Flickr Feed**
- Developed with ❤️ and dedication using Swift 5.1 and XCode 11

Project Instructions:

- Open ***FlickrFeed.xcodeproj*** file
- Scheme to be used to run the project: **FlickrFeed**
- Code run and tested on iPhone 7 device running **iOS13** and iPad running **iOS12.**

Project Dependencies:

- Programming Language*:* ***Swift 5.1***
- IDE*: XCode 11*

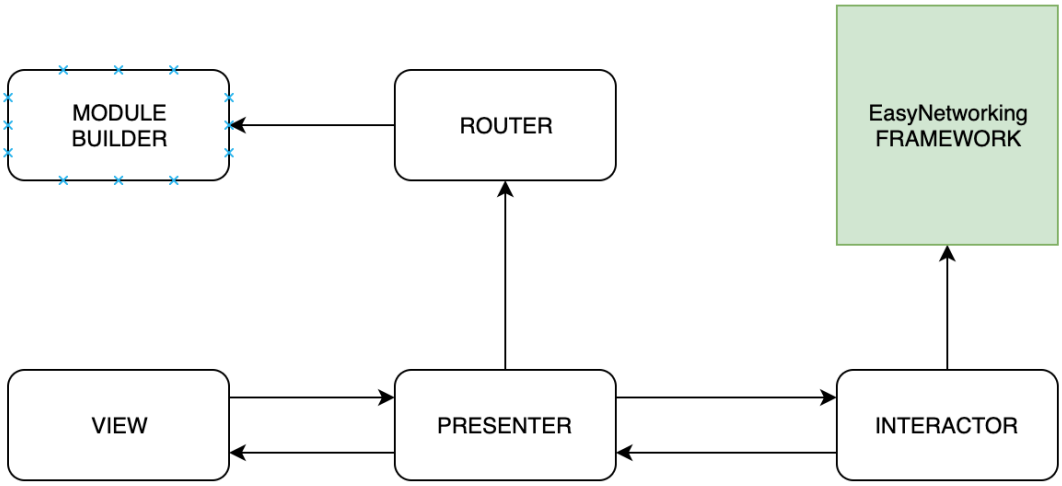
Project Structure:

- Architecture used: **VIPER**
- High Level Workspace Modules:
    - **FlickrFeed** – The Main iOS Application
    - **EasyNetworking** – The lightweight Network Library which is capable of serving any kind of HTTP request through its different layers of abstractions.
    - **ImageCache –** A tiny module for image caching.
- Reasons for creating different modules: Modules are free from any app specific use cases and can be used in any app. This enables re-usability of the code across multiple apps inside an organization.
- How modules are injected in the main app? - through pre-compiled *framework* file

Project Sanitizations done:

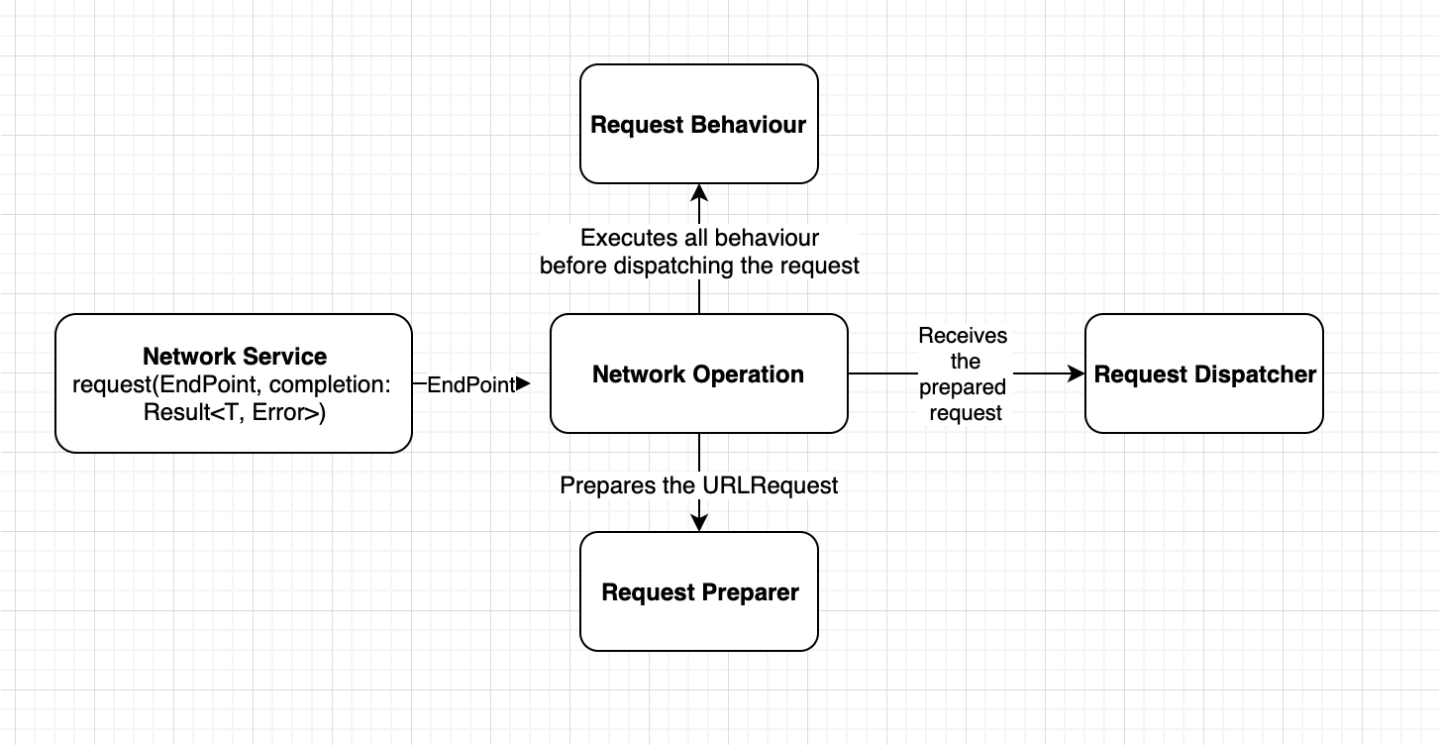
- The following sanitizations have been run and the project has been found clean of such issues:
    - Main thread sanitizer

Project UML Diagram:



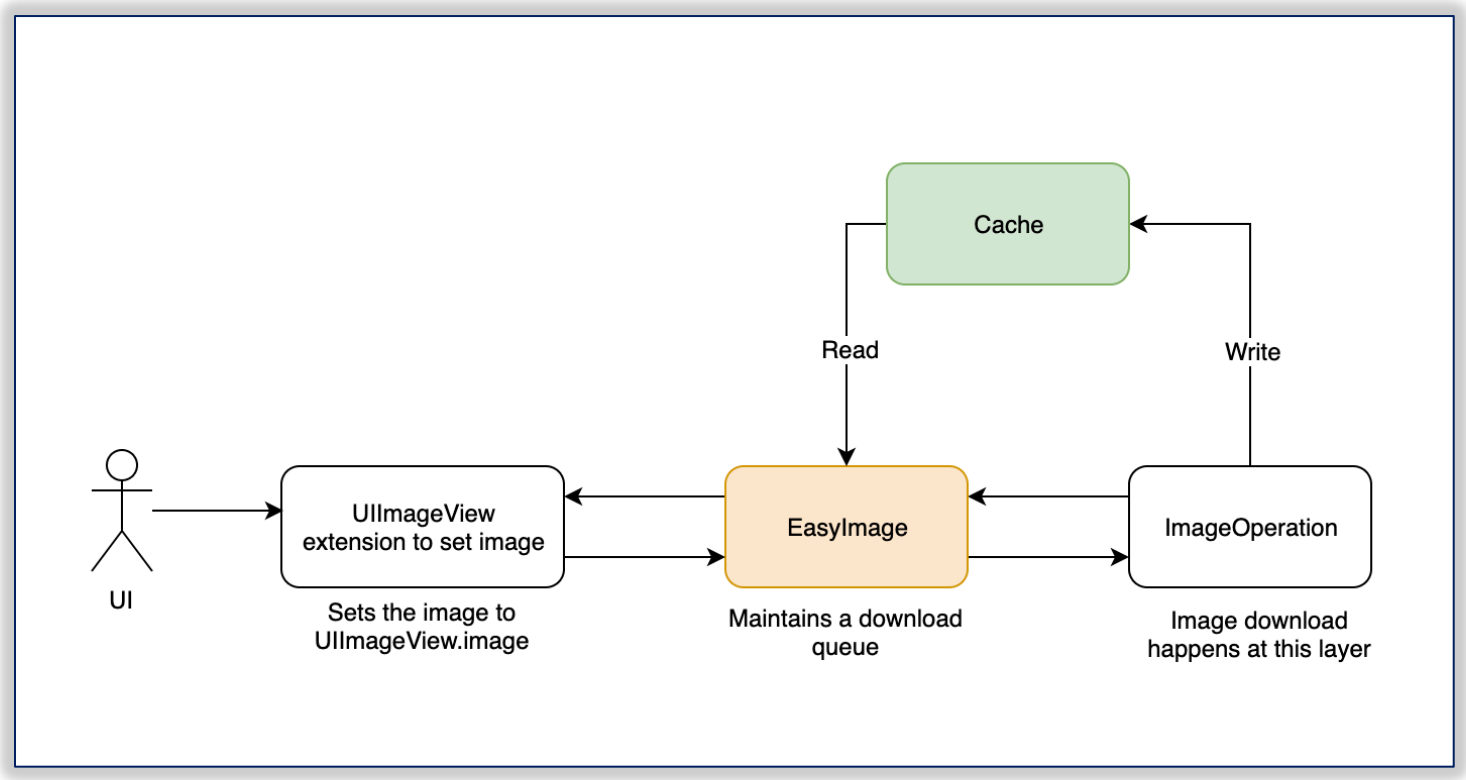
- **VIEW**: This is dumb and takes all data from the PRESENTER.
- **PRESENTER*:*** This layer always has the latest data and communicates with VIEW and ROUTER. Data with the PRESENTER is the only source of truth at any point in the execution time.
- **INTERACTOR:** This layer only has the business logic like form validations, getting data from networking layer and then modify it as required.
- **ROUTER:** This layer communicates with the BUILDER and navigates across the app.
- **BUILDER:** This layer creates modules resolving and injecting all the dependencies and returns the module VC. This can be modified to return the whole dependency container, if required in a bigger project.
- **ENTITY:** The FlickrFeed, FlickrPhoto, etc model confirms to Decodable protocol making it easily understandable by other layers.
- **NETWORKING:** This is the framework for all HTTP type tasks.

Networking Framework component Diagram:

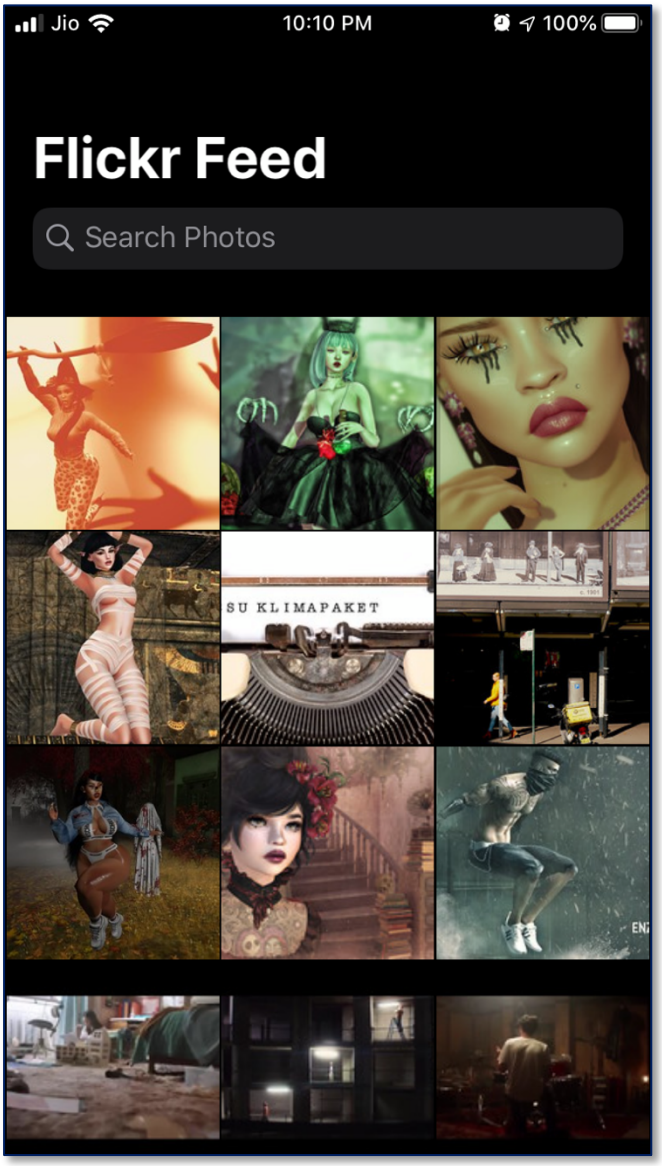


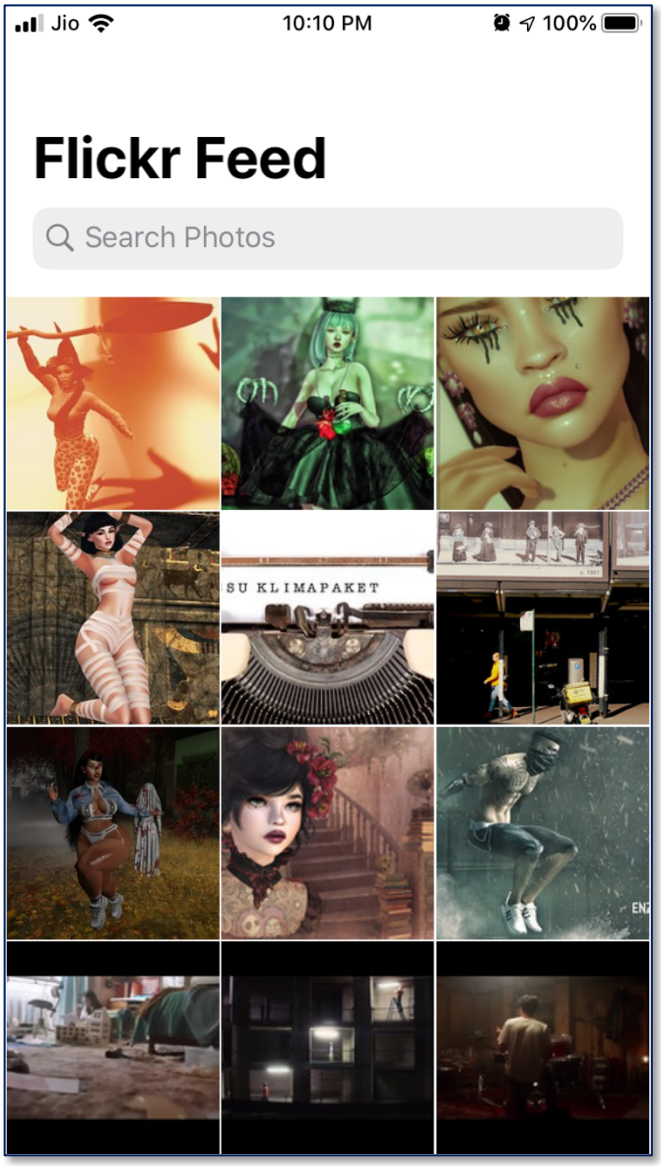
- **Request Behaviour**:
  - **modify()*:*** It modifies the request and return it. OAuth and authentication module can confirm this protocol and do the needful
  - ***willSend():*** Network Logger implements this method to log the request on the console.
  - **didReceive()**: NetworkLogger implements this to log the received response.
- This complete module is loosely coupled and can be useful while mocking and testing.
- Refer the code for more insight. ☺

ImageCache Framework component diagram:



Dark Mode support





Practices followed:

- Regular commit after each independent module/feature is developed.
- Followed TDD approach of Software Programming practice.
- Followed CLEAN Architecture.
- Written XCode documentation for functions.
- Followed SOLID principles.
- Code Reusability:
    - Kept each *function* across the project as less no of lines as possible.
    - Added ImageCache and EasyNetworking code as target to maximize its reusability across different other apps in the organization.

Why EasyNetworking and ImageCache is added as target?

- Code Modularity.
- Dynamic frameworks make the app launch faster.
- Making the code reusable and importable in other apps.

Known bugs:

UI/UX :

- Jerk in the Navigation Bar when search bar is transitioning to becomeFirstResponder. This seems like an iOS 13 issue.
- As soon as the NEW/FRESH search result arrives for the first time, the SCREEN loader disappears, and CollectionView Footer loader starts. This creates a bad experience.

Logical :

- Async image download is not allowed to set the image to the wrong cell through the usage of **imageUrl** as key**.** A faulty logic around this is not allowing the Image Downloader code to set the image to the correct cell sometimes. Once in a 100 times.
- Very fast scrolling leaves the cell blank without image for some time while displaying cached images.
    - This is happening due to asynchronous access of cached images and slow performance of URLCache.
    - Using NSCache will solve the problem but NSCache provides in-memory cache and does not provide in-disk storage. Custom logic have to be written around NSCache to have both in-memory and in-disk caching. Due to shortage of time, I picked up URLCache which provides both in-memory and in-disk caching out of the box.

Note:

- While XCUITest runs, the app does not automatically launch and the test case keeps on waiting for the component to appear on the screen and eventually fails.
    - Solution: Manually launch the app while XCUITest starts printing *waiting for element.*
    - This one looks like a XCode 11 bug.

Scopes of improvement:

- Overall code coverage should reach 80% +.
- UI:
  - Better way to display error on screens instead of just Error popups.
- Code:
  - Better and detailed documentation for the Networking Framework.
  - Cache used for Image caching could have been made generic to store any kind of Data.
  - ImageCache framework code can be written more clean and optimized using SOLID principles.
- Is there a scope of use of Reactive Programming?
  - Yes, binding of INTERACTOR -> PRESENTER and PRESENTER -> VIEW can easily be achieved through RxSwift and this will make the codebase more clean and robust.
- Address Sanitizer and other XCode profilers are yet to be run on the project.
- **Imp**: Custom collection view layout can be written to display each image in their actual aspect ratio if height and width of the image is already known from the server. Calculating it on the iOS side will be performance hit. Collection View needs these parameters before rendering the cell layout.
- I would love to discuss more with your team if given the opportunity.

Thank you

- Sudipta Sahoo
I can be reached at +91 8080842428 or me@sudiptasahoo.in