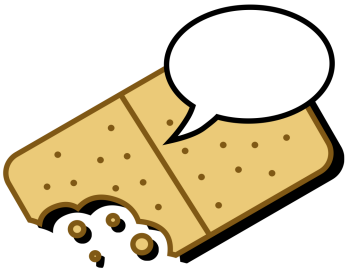
POLITECNICO DI MILANO

DESIGN AND IMPLEMENTATION OF MOBILE APPLICATIONS



Cracker

Design Document

by

Leonardo Febbo

Person code: 10492637
Matricola: 905180

Elisabetta Ferreri

Person code: 10487482
Matricola: 899082

February 17, 2020

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the design phases of the realization of the "Cracker" mobile application, with particular attention to the architecture and the user experience.
The main aim of "Cracker" is to help users who read Marvel comics by letting them mark the issues they have read and keeping them up to date with the newest releases by Marvel.
This project is the result of the implementation of the knowledge acquired during the course ”Design and Implementation of Mobile Applications” provided by Politecnico di Milano.

1.2 Definitions and acronyms

1.2.1 Definitions

- **Framework**: reusable set of libraries or classes for a software system
- **Issue**: a single, periodical, magazine-like publication
- **REST**: a way of providing interoperability between computer systems on the Internet
- **Series**: a title collecting a variable amount of issues
- **User**: a person who has performed the login operation successfully

1.2.2 Acronymns

- **API**: Application Programming Interface
- **HTTPS**: HyperText Transfer Protocol Secure
- **JSON**: JavaScript Object Notation
- **MVC**: Model - View - Controller
- **URL**: Uniform Resource Locator

1.3 Scope

Cracker has been developed for users who regularly read Marvel comics and want to keep track of the series they are reading and the new issues that are released every week by the American publishing house. Similar apps already exist for other types of media, such as tv shows and movies, but none of them have comics as their focus. Keeping the possible needs of the users in mind, six main screens have been found of interest for our application:

- **Issue**: screen displaying information about a selected issue
- **Series**: screen displaying information about a selected series
- **Up Next**: list of issues to be published in the current week, in the next week and in the current month
- **To Read**: list of the next issues to read for each of the series the user follows
- **Search**: search for a series given its name
- **Profile**: personal information about the user

1.4 Framework

The development of Cracker was achieved through the use of native iOS SDKs, in particular by using the Swift programming language. This choice allowed greater control of system resources and access to system services, otherwise not possible if using cross-platform frameworks. The purpose is to implement different functionalities and integration with other sites.

1.5 Functional requirements

Cracker provides a simple, intuitive and user-friendly interface that allows the users to:

1. register with a personal e-mail or login with Facebook and Google
2. see information about an issue
3. see information about a series
4. mark series they are reading as *following*
5. select which issues of a series they have read
6. see the next issue not yet read for all the series they are following
7. see the series they are following
8. see statistics on the series they are following
9. see, on a weekly basis, the latest issues of all the series Marvel is publishing
10. search for a specific series, given its name

1.6 Non-functional requirements

The application must be able to:

- run on both iPhone and iPad
- adapt its written content to the language selected in the device settings (available languages: English, Italian)
- adapt to different screen sizes
- keep preferences and status at every start

1.7 Assumptions, dependencies, constraints

1.7.1 Assumptions and dependencies

- **Internet connection**: the device used by the user disposes of an Internet connection and a sufficient bandwidth to use the application
- **API availability**: the API provided by third part's services are always available
- **No privileged users**: there are no privileged users or administrators with particular functions
- **No user connections**: every user is independent from the others

1.7.2 Constraints

- **Hardware limitations**: our application runs on every mobile device like smartphones and tablets; therefore, as the app consumes a low amount of RAM, the only hardware constraint for the users is to have a mid-range device
- **Parallel operations**: the application must be able to handle multiple parallel requests with high reactivity

2 Architecture

2.1 Overview

In this section we describe the architectural design of our system, the main components and their interactions.
The system will be described starting with high-level components. Then a more detailed description is provided for the client and the database structure, including the architectural patterns applied.

2.2 High level components

The main high level components of our system are:

- **Client**: this component is responsible for the visualization of data (front-end) and for the management of requests and replies to and from the services providing data to the system.
- **Firebase**: this component is responsible for the storage of the permanent data of the users and for the registration and login of users; we use the services provided by Firebase to manage these data.
- **Marvel APIs**: we use the APIs provided by Marvel in order to always have complete and up-to-date data on comics.
- **Facebook**: used for the login procedure.
- **Google**: used for the login procedure.

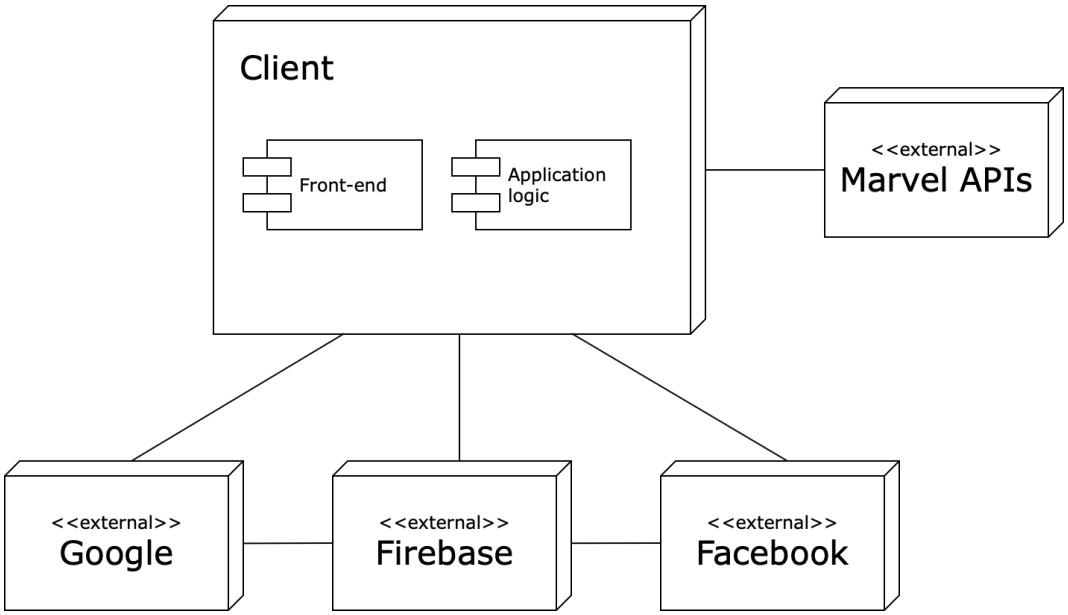


Figure 1: High Level Component View

2.3 Client

For the implementation of the application we have chosen a mobile back-end, that is a client architecture. This choice was made mainly because the application does not interface with other users and because for various services it uses third-party APIs. Communication with third-party services is based on HTTPS REST requests, in particular through GET requests. The client uses the traditional MVC pattern:

- Model: this package contains all the classes representing data to be shown to the single user, taken by the Controller and published by the View.
- View: this package contains all the components that display data to the user and interact with him.
- Controller: this package contains all the objects in charge to interact between one or more view objects of the application and one or more model objects.

2.4 Database

Since the application receives all the necessary data from external services through the API, the only data that has to be saved is the information about the user, the series that he follows and the comics he has read. Moreover, since there is no interaction between the different users, this data is saved on the Cloud Firestore on Firebase Platform to guarantee ACID properties. The database interacts only with the application layer. The security restriction will be implemented to guarantee the user privacy from unauthorized users. The communication between the database and application tier has to be encrypted. The access to the data has to be guaranteed only to the authorized user for that data.

Figure 2: Database Cloud FIrestore

The E / R model of the data collected through the API together with the user data saved on the firebase is as follows:

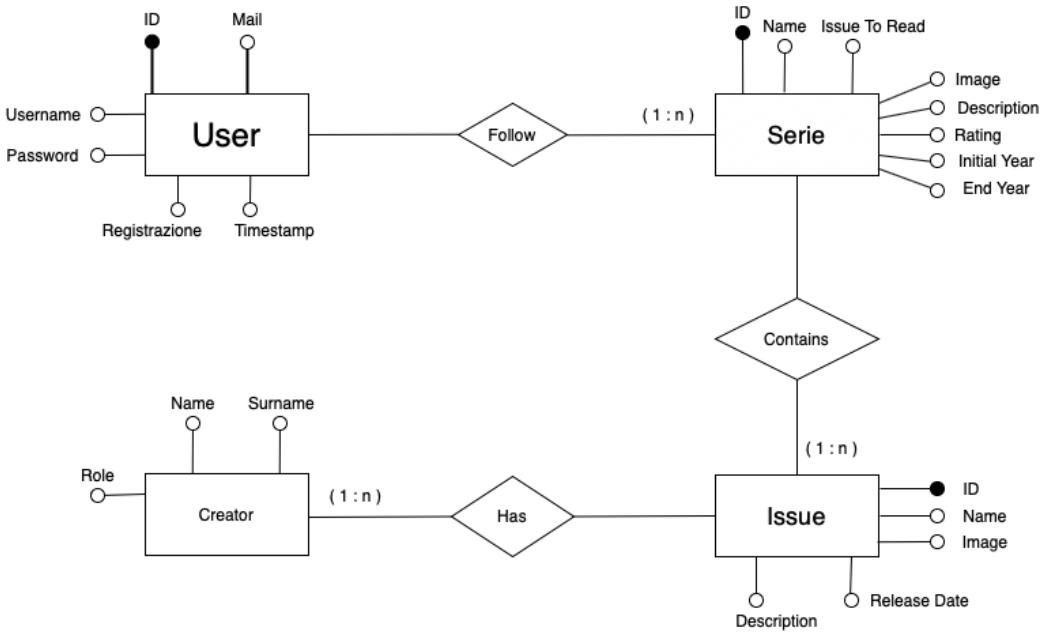


Figure 3: E/R Diagram

3 External Services and Libraries

The "Cracker" application uses Marvel Developer API in order to provide the user all the services. The SwiftyJSON library was used to parse the JSON data.

3.1 Facebook SDK

The Facebook SDK was used to manage the login through Facebook account. When a user wants to authenticate, the application makes the call to the Facebook service through:

```
func loginButton(_ loginButton: FBLoginButton, didCompleteWith result:
    LoginManagerLoginResult?, error: Error?) {
   if let error = error {
      print(error.localizedDescription)
      return
   } else {
      let credential = FacebookAuthProvider.credential(withAccessToken:
          AccessToken.current!.tokenString)
      // then sign in through firebase with this credentials
   }
}
```

This method is used to send the user on Facebook for the authentication; once authenticated, the user redirected to ”Cracker” which will store the token provided into Firebase. The following pods were included to use Facebook:

```
pod 'FacebookCore'
pod 'FacebookLogin'
```

3.2 Google SDK

The Google SDK was used to manage the login through Google account. When a user wants to authenticate, the application makes the call to the Google service through:

```
func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!,
    withError error: Error!) {
   if let error = error {
      if (error as NSError).code ==
          GIDSignInErrorCode.hasNoAuthInKeychain.rawValue {
         print("The user has not signed in before or they have since signed
             out.")
      } else {
         print("\(error.localizedDescription)")
      }
      return
   } else {
      guard let authentication = user.authentication else { return }
      let credential = GoogleAuthProvider.credential(withIDToken:
          authentication.idToken, accessToken: authentication.accessToken)
      // then sign in through firebase with this credentials
   }
}
```

This method is used to send the user on Google for the authentication; once authenticated, the user redirected to ”Cracker” which will store the token provided into Firebase. The following pods were included to use Google:

```
pod 'GoogleSignIn'
```

3.3 Firebase SDK

Firebase SDKs are used to access the database, to login with email and password and to save other authentication through Facebook and Google.

```
Auth.auth().signIn(with: credential) { (authResult, error) in
   if error == error{
      print("Error Log In Firebase \(error)")
      return
   } else {
      print("Logged In Firebase")
   }
}
```

For database access, the components included in the SDKs was used:

```
let user =
    Firestore.firestore().collection("Users").document("\((Auth.auth().currentUser?.uid)!)")
User.collection("Series").getDocuments() { (querySnapshot, err) in
   if let err = err {
      print("Error getting documents: \(err)")
   } else {
      print("Collection got")
      ...
   }
}
```

The following pods were included to use Firebase:

```
pod 'Firebase/Auth'
pod 'Firebase/Firestore'
pod 'Firebase/Analytics'
pod 'FirebaseFirestoreSwift'
```

3.4 Alamofire

Alamofire is used to send GET requests to the endpoint supplied by Marvel APIs. If the answer is valid, it returns a JSON file that can be parsed.

```
Alamofire.request(url, method: .get, parameters: parameters).responseJSON
    { response in
   if response.result.isSuccess {
      print("Success! Got the data")
      let json : JSON = JSON(response.result.value!)
   } else {
      print("Error \(String(describing: response.result.error))")
   }
}
```

The following pods were included to use Alamofire:

```
pod 'Alamofire'
```

3.5 Marvel Comics API

The Marvel Comics API is a RESTful service which provides methods for accessing specific resources at canonical URLs and for searching and filtering sets of resources by various criteria. All representations are encoded as JSON objects. All documentation can be found here: https://developer.marvel.com/

3.5.1 Service Endpoint

The Marvel Comics API's base endpoint is http(s)://gateway.marvel.com/

3.5.2 Resources

You can access six resource types using the API:

- **Comics**: individual print and digital comic issues, collections and graphic novels.
- **Comics series**: sequentially numbered (well, mostly sequentially numbered) groups comics with the same title.
- **Comics stories**: indivisible, reusable components of comics. For example, the cover from Amazing Fantasy 15 or the origin of Spider-Man story from that comic.
- **Comics events and crossover**: big, universe-altering storylines.
- **Creators**: women, men and organizations who create comics.
- **Characters**: the women, men, organizations, alien species, deities, animals, non- corporeal entities, trans-dimensional manifestations, abstract personifications, and green amorphous blobs which occupy the Marvel Universe (and various alternate universes, timelines and altered realities therein).

Results returned by the API endpoints have the same general format, no matter which entity type the endpoint returns. Every successful call will return a wrapper object, which contains metadata about the call and a container object, which displays pagination information and an array of the results returned by this call. This pattern is consistent even if you are requesting a single object.

Putting it together, a typical result will look like this:

```
{
   "code": 200,
   "status": "Ok",
   "etag": "f0fbae65eb2f8f28bdeea0a29be8749a4e67acb3",
   "data": {
      "offset": 0,
      "limit": 20,
      "total": 30920,
      "count": 20,
      "results": [{array of objects}}]
   }
}
```

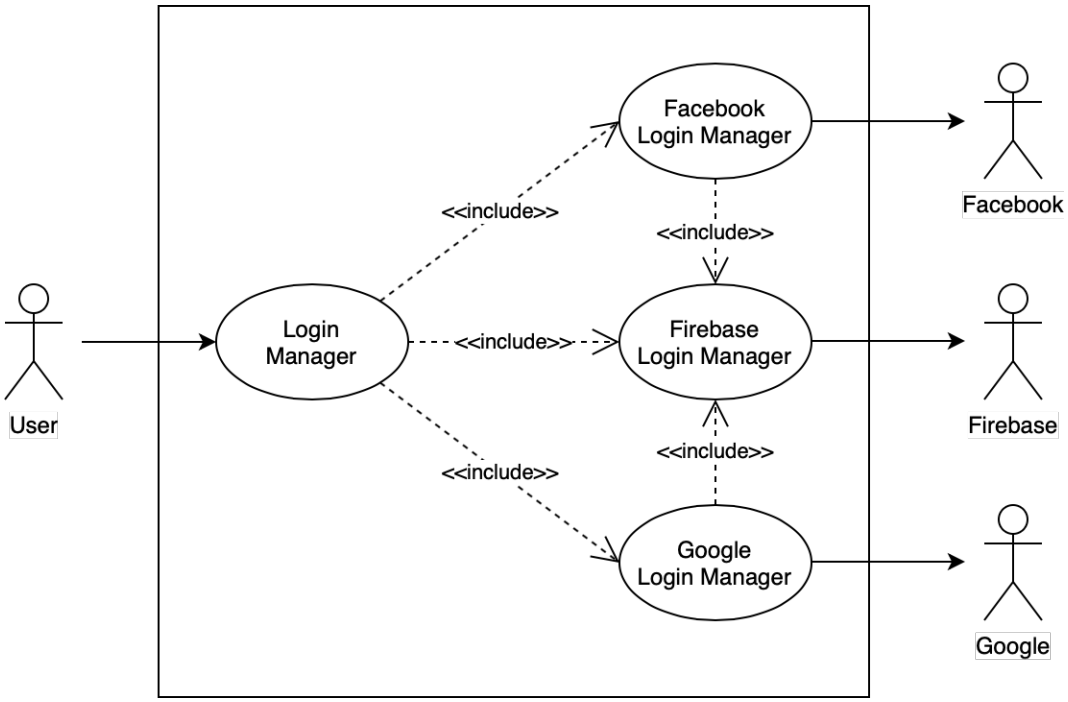
3.5.3 Authentication

All requests to the APIs must be authenticated using the methods outlined in the request signing and authentication guidelines. Requests which fail authentication generally pass a 401 HTTP code and an error describing the reason for rejection.

4 Use Cases

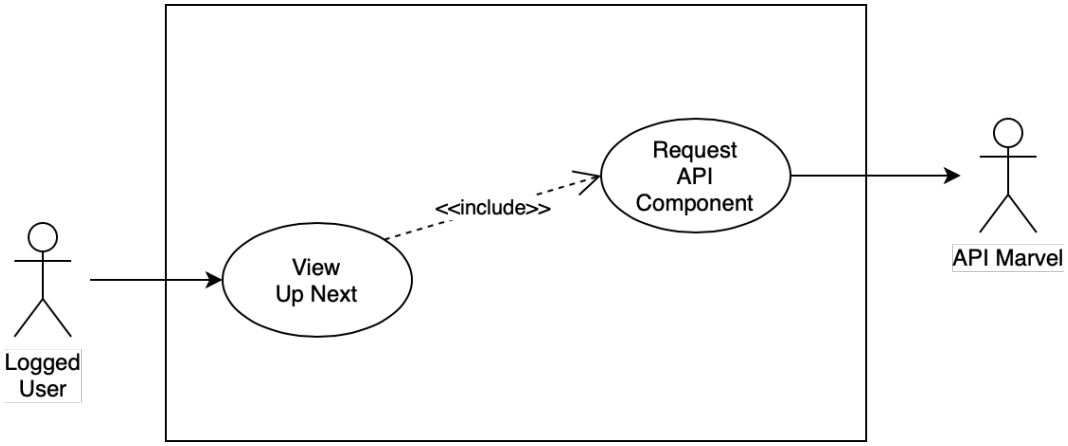
This section describes how actors can interact with "Cracker" in order to use all the features implemented in the app. The focus of this part is on the front-end and we show the operations that can be performed by the actors without taking care of the system architecture behind the app.

4.1 Login



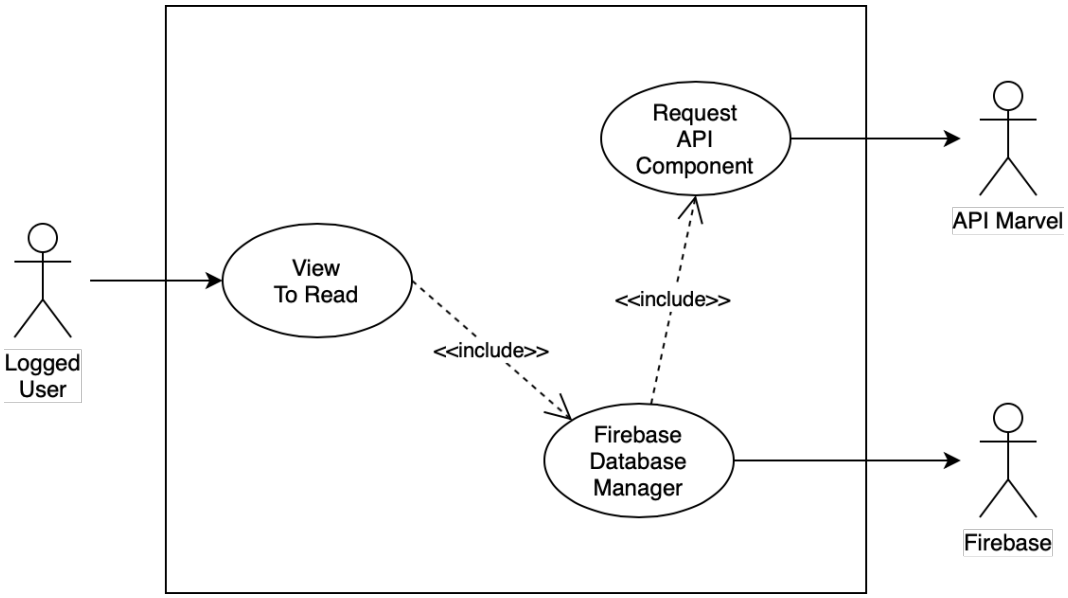
Name	Login
Actor	User
Entry Condition	Guest with login credentials
Goal	1
Event flow	• The user opens the application • The user presses ”Login with Facebook” (or "Login with Google") located in the welcome screen • The user is redirected on Facebook (or Google) to enter credentials • The app logs in through Facebook (or Google) • The app login into Firebase using Facebook (or Google) account or • The user opens the application • The user presses ”Login with e-mail" located in the welcome screen • The user is asked to input his e-mail and password • The app login into Firebase using e-mail and password
Exit condition	Actor becomes Logged User
Exceptions	The user is not connected to the network or he hasn’t a Facebook or Google account

4.2 View Up Next



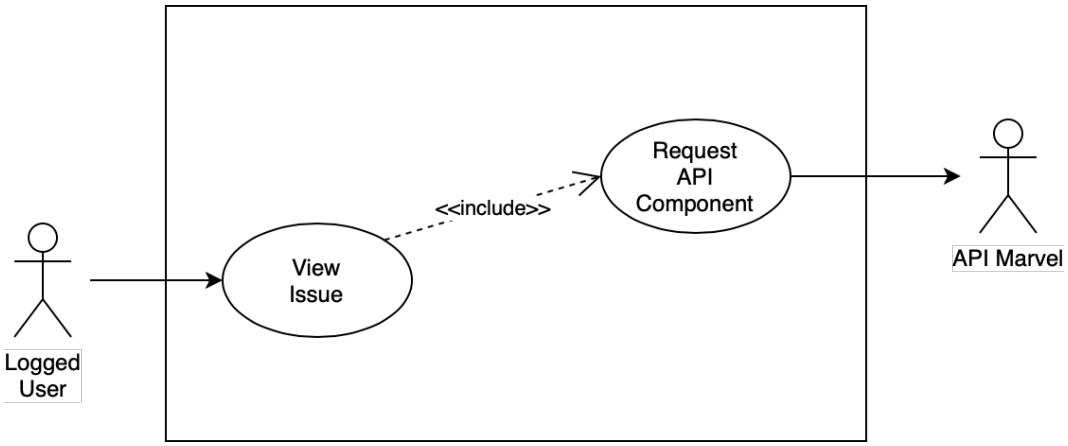
Name	View Up Next
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	9
Event flow	• The user opens the application • The user presses on the "Up Next" tab located on the Tab Bar • The app shows the upcoming issues divided by being published this week, next week, this month
Exit condition	The user reads the titles of the upcoming issues
Exceptions	The user is not connected to the network so he cannot send a request to read the titles of the upcoming issues

4.3 View To Read



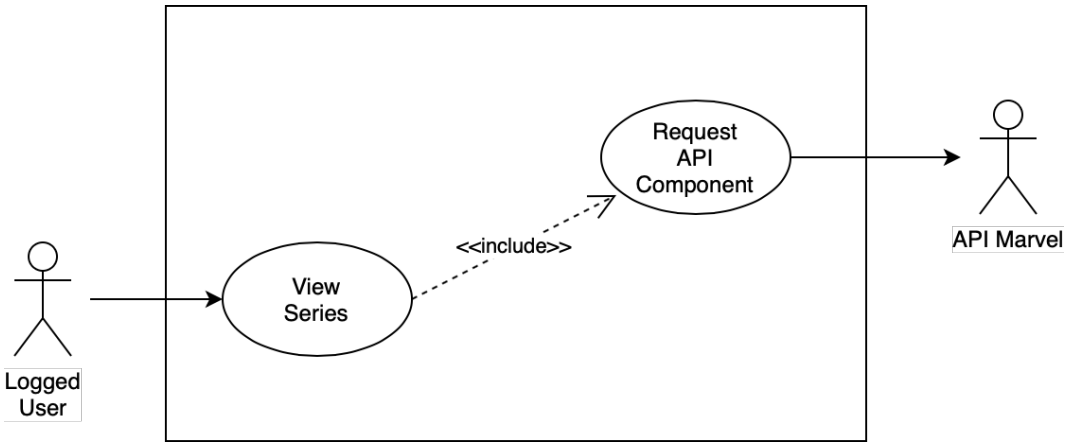
Name	View To Read
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	6
Event flow	• The user opens the application • The user presses on the "To Read" tab located on the Tab Bar • The app shows the next issue to read for all the series that the user is following
Exit condition	The user sees the covers of the next issues he has to read
Exceptions	The user is not connected to the network so he cannot send a request to get the next issues he has to read

4.4 View Issue



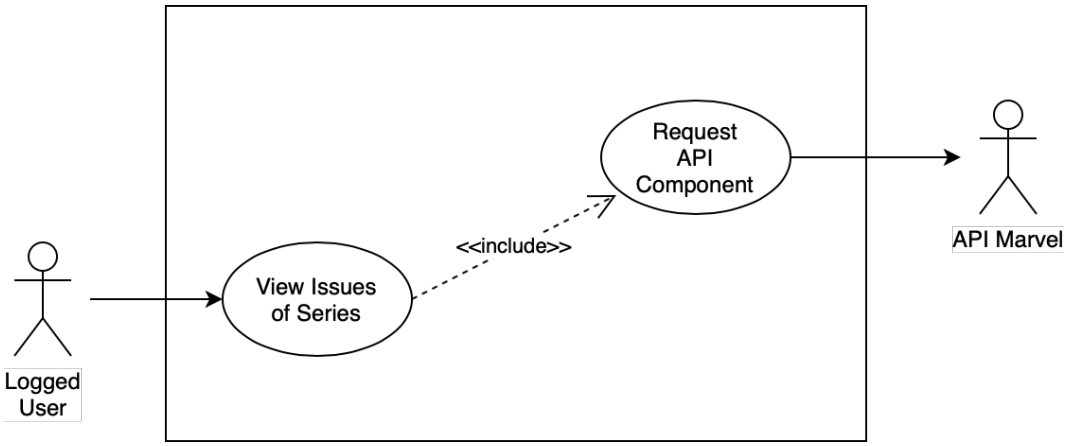
Name	View Issue
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	2
Event flow	• The user opens the application and navigates through some screens • The user presses on the button representing an issue in one of the screen where such a button is present • The app shows the most important information about the issue
Exit condition	The user reads the information about the issue
Exceptions	The user is not connected to the network so he cannot send a request to get the information about the issue

4.5 View Series



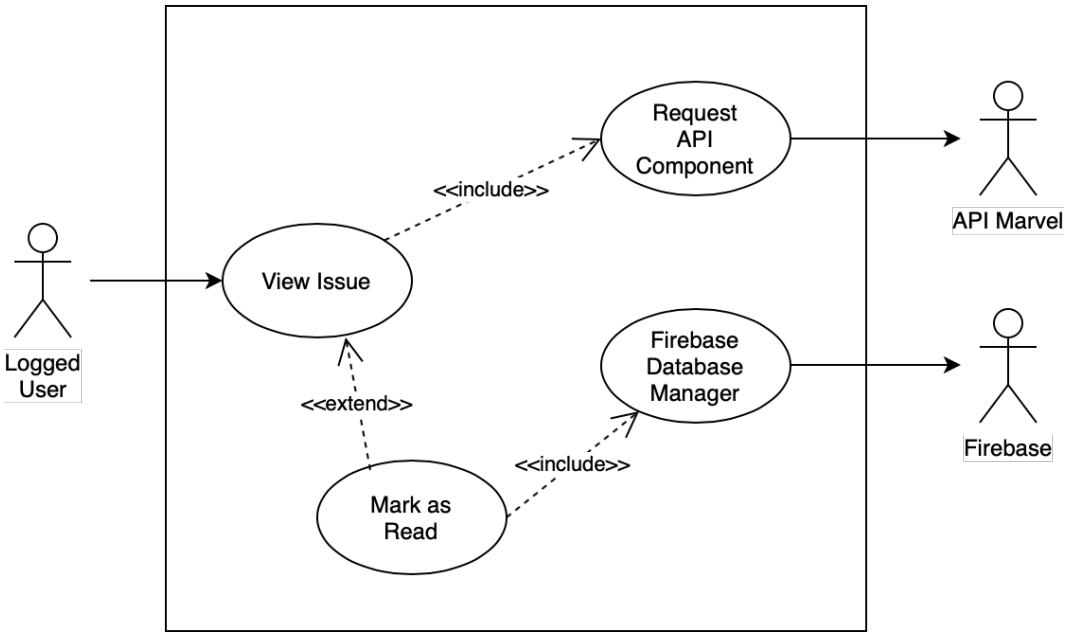
Name	View Series
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	3
Event flow	• The user opens the application and navigates through some screens • The user presses on the button representing a series in one of the screen where such a button is present • The app shows the most important information about the series
Exit condition	The user reads the information about the series
Exceptions	The user is not connected to the network so he cannot send a request to get the information about the series

4.6 View Issues of Series



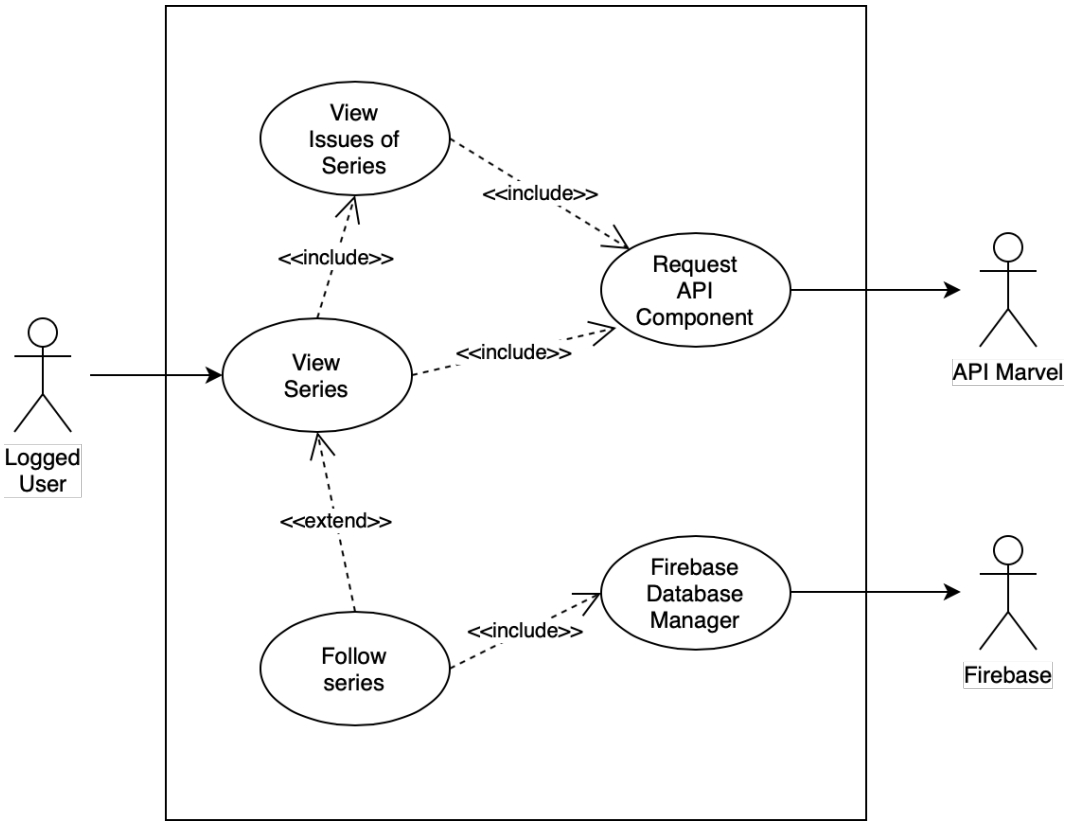
Name	View Issues of Series
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	3
Event flow	• The user opens the application and navigates through some screens • The user reaches a screen displaying the information about a series • The user presses on the "See all issues" button • The app shows all the issues belonging to that series grouped in sections of 10
Exit condition	The user sees all the issues belonging to a series
Exceptions	The user is not connected to the network so he cannot send a request to get the issues belonging to a series

4.7 Mark Issue as Read



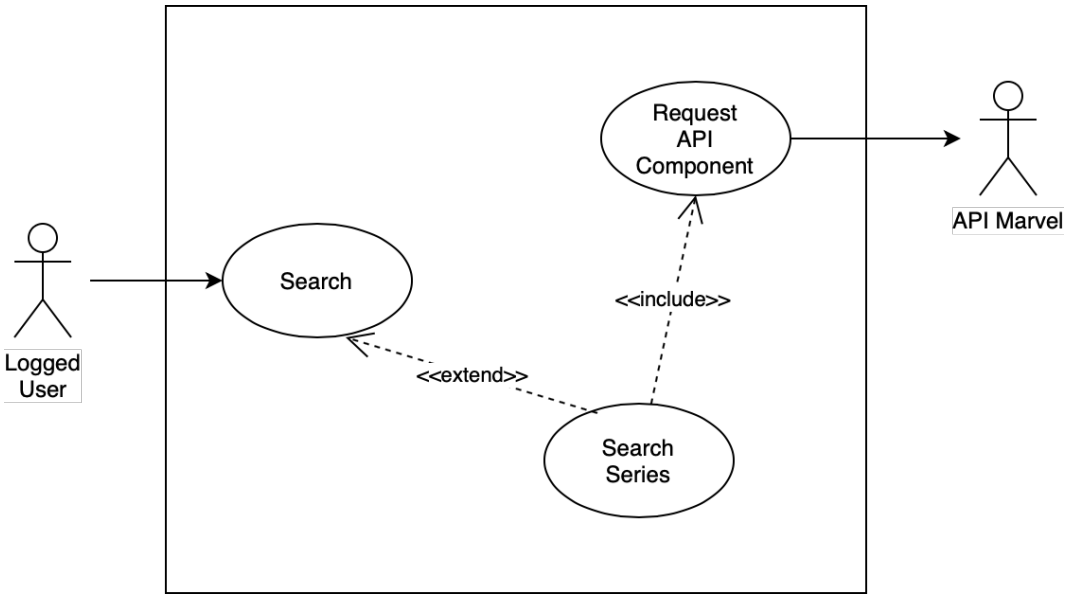
Name	Mark Issue as Read
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	5
Event flow	• The user opens the application and navigates through some screens, reaching one with information about an issue • The user presses on the "Mark as read" button • The app records the action and updates the text of the button to "Mark as unread"
Exit condition	The user marked an issue as read
Exceptions	The user is not connected to the network so he cannot send a request to mark an issue as read

4.8 Follow a Series



Name	Follow a Series
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	4
Event flow	• The user opens the application and navigates through some screens, reaching one with information about a series • The user presses on the "Follow this series" button • The app records the action and updates the text of the button to "Unfollow this series"
Exit condition	The user followed a series
Exceptions	The user is not connected to the network so he cannot send a request to follow the series

4.9 Search for a Series



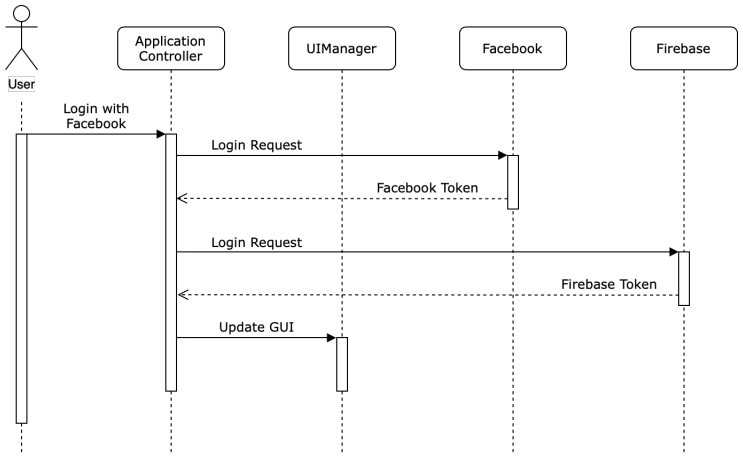
Name	Search for a Series
Actor	Logged User
Entry Condition	The user logged in correctly
Goal	10
Event flow	• The user opens the application • The user presses on the "Search" tab located on the Tab Bar • The app show a search bar • The user presses on the search bar and types the name of a series • The app shows the series whose name matches the one written by the user
Exit condition	The user sees a list of series that match his input
Exceptions	The user is not connected to the network so he cannot send a request to search for a series

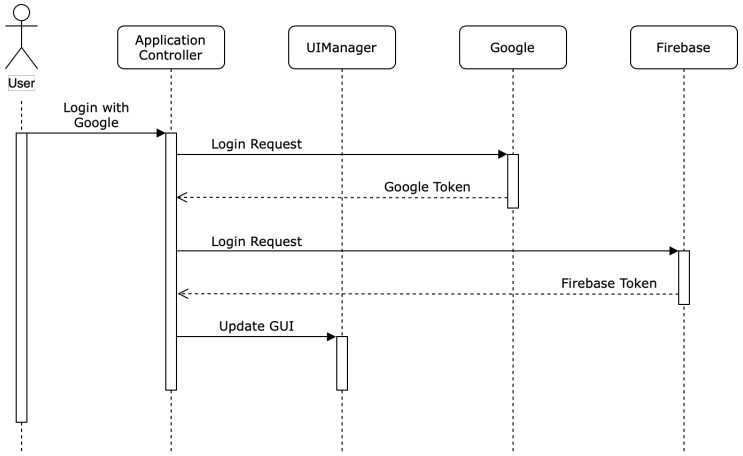
5 Sequence Diagrams

In order to explain our work and facilitate further implementations we have decided to show the logical flows aimed to create some features. The sequence diagrams will describe the interactions between the different parts of the system and the user.

5.1 Login

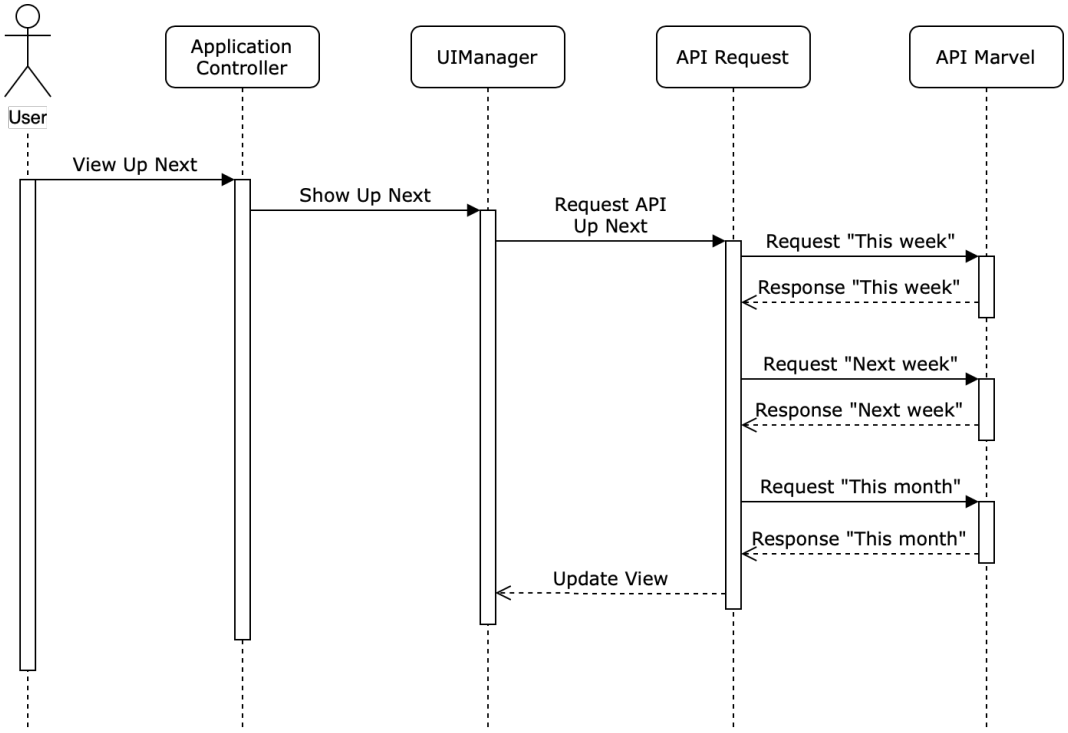
The ”Login” procedure starts when the user opens the app for the first time or whenever he logs out. The user can choose to be redirected to either Facebook or Google to complete the procedure. Immediately after logging on Facebook or Google the application will log into the Firebase Database.
Once this procedure is completed, the system will update the GUI by enabling the available sections. In the event of an error, the system will show an error message.





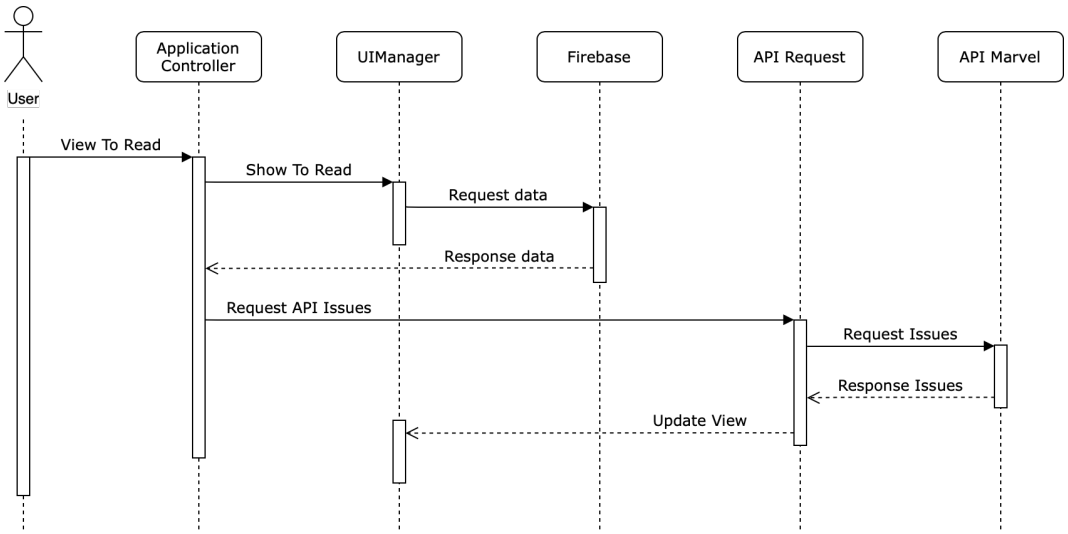
5.2 View Up Next

The ”View Up Next” procedure starts when the user opens the application when he’s already performed the login or when he presses the ”Up Next” tab in the Tab Bar. The sequence diagram shows the normal procedure.
After the user activates the "Up Next" tab, the application will send 3 requests to the ”MarvelAPI” service, one for each of the sections that will be displayed (comics released in the current week, comics released in the following week, comics released in the current month), in order to retrieve the titles of the issues belonging in each of the sections.
Once this information is obtained, the Controller will create a UITableViewCell for each issue and insert them into the correct section in a TableView.



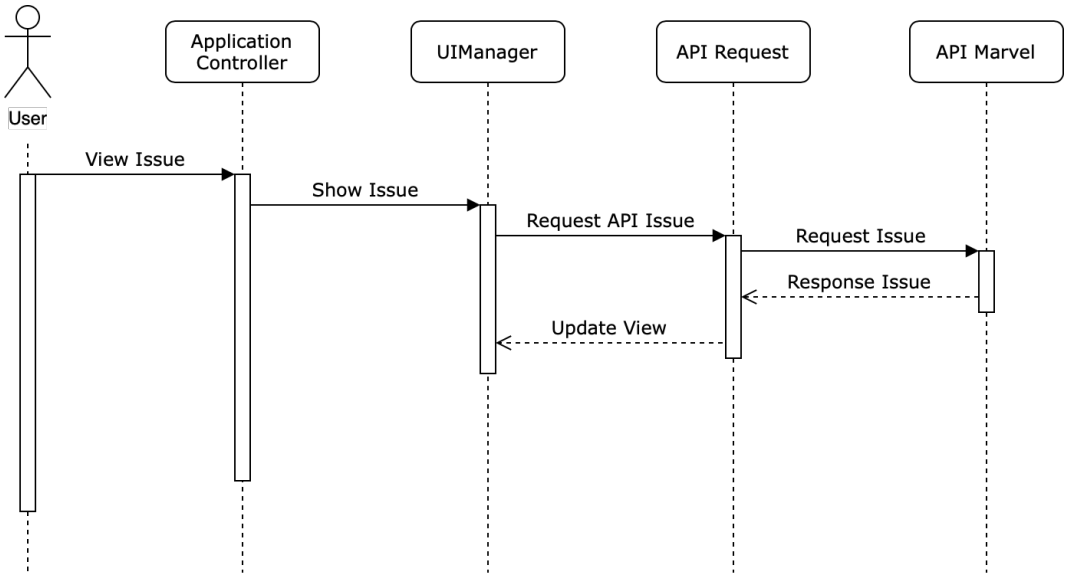
5.3 View To Read

The ”View To Read" procedure starts when the user presses the ”To Read” tab in the Tab Bar. The sequence diagram shows the normal procedure.
After the user activates the "To Read" tab, the application will send a request to Firebase in order to retrieve the identifiers of the next issues the user has to read.
Once these identifiers are obtained, the application will send a request to the "MarvelAPI" service in order to retrieve information about the issues.
Once this information is obtained, the Controller will create a custom UICollectionViewCell for each issue and insert them into a UICollectionView.
Furthermore and asynchronously, the application will download the cover of each issue to display to the user.



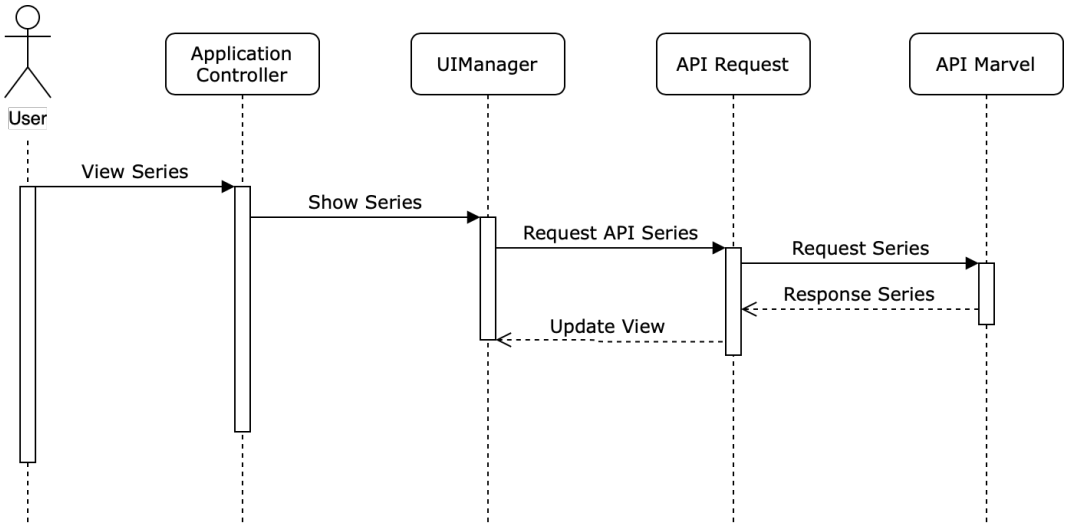
5.4 View Issue

The ”View Issue" procedure starts when the user presses a button representing an issue in one of the screens where such buttons are displayed. The sequence diagram shows the normal procedure.
After the user activates the "View Issue" button, the application will send a request to the "MarvelAPI" service in order to retrieve information about the issue.
Once this information is obtained, the Controller will create a set of UILabels, UITextViews and a UIImage to display the relevant information of the issue.



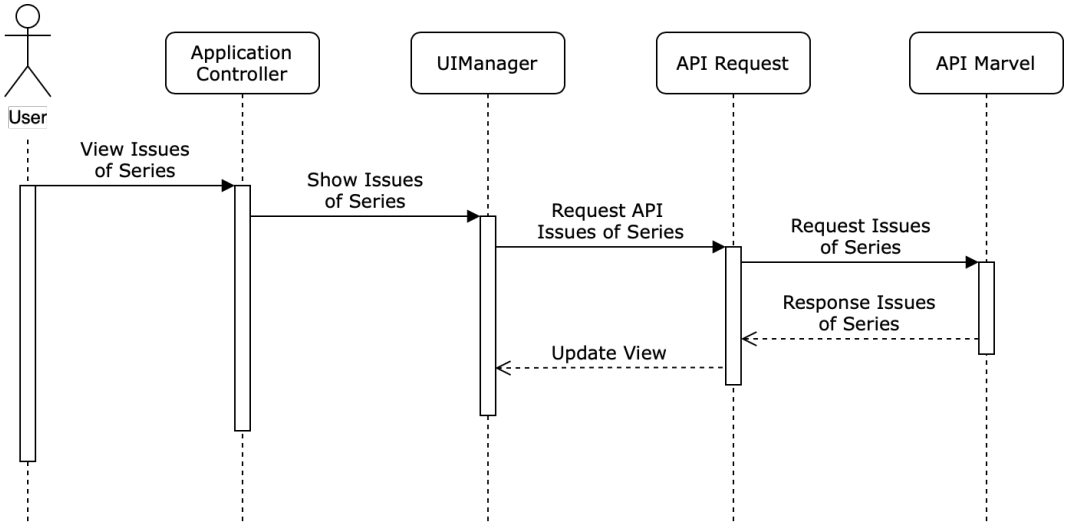
5.5 View Series

The ”View Series" procedure starts when the user presses a button representing a series in one of the screens where such buttons are displayed. The sequence diagram shows the normal procedure.
After the user activates the "View Series" button, the application will send a request to the "MarvelAPI" service in order to retrieve information about the series.
Once all this information is obtained, the Controller will create a set of UILabels, UITextViews and a UIImage to display the relevant information of the series.



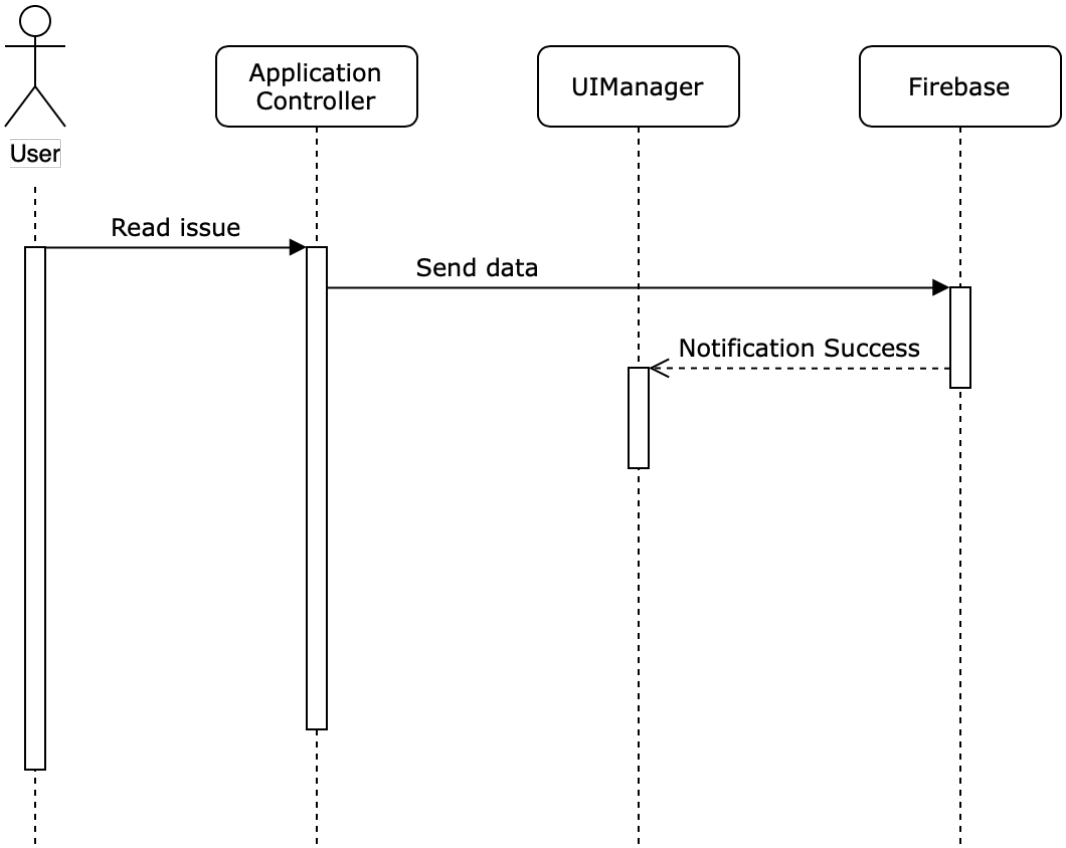
5.6 View Issues of Series

The ”View Issues of Series" procedure starts when the user presses the "See all issues" button when he’s viewing a series. The sequence diagram shows the normal procedure.
After the user activates the "See all issues" button, the application will send a request to the "MarvelAPI" service in order to retrieve all the issues belonging to that series.
Once all this information is obtained, the Controller will create a UITableViewCell for each issue and insert them into a UITableView, dividing them in expandable sections with 10 maximum cells for section.



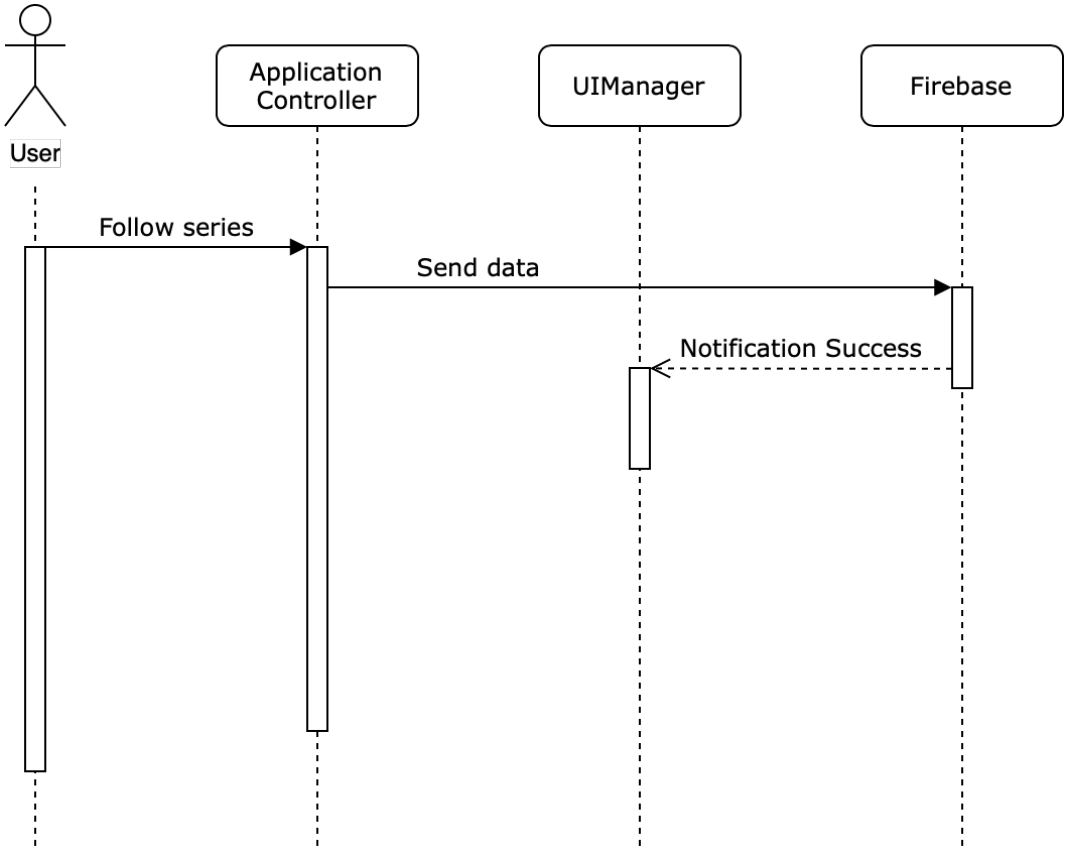
5.7 Mark Issue as Read

The ”Mark Issue as Read" procedure starts when the user presses the corresponding button while he’s either viewing an issue or the "To Read" section. The sequence diagram shows the normal procedure.
After the user activates the "Mark Issue as Read" button, the application will send a request to Firebase in order to register the issue as read in the database.
After receiving a notification of success, the Controller will update the GUI to show the success of the procedure.



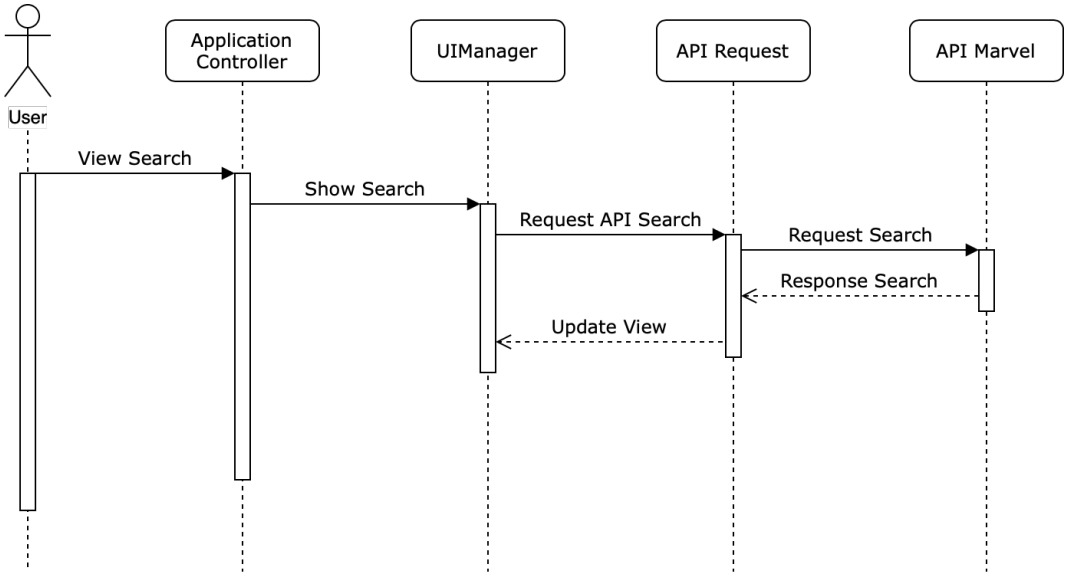
5.8 Follow a Series

The ”Follow a Series" procedure starts when the user presses the corresponding button while he’s viewing a series. The sequence diagram shows the normal procedure.
After the user activates the "Follow a Series" button, the application will send a request to Firebase in order to register the series as followed in the database.
After receiving a notification of success, the Controller will update the GUI to show the success of the procedure.



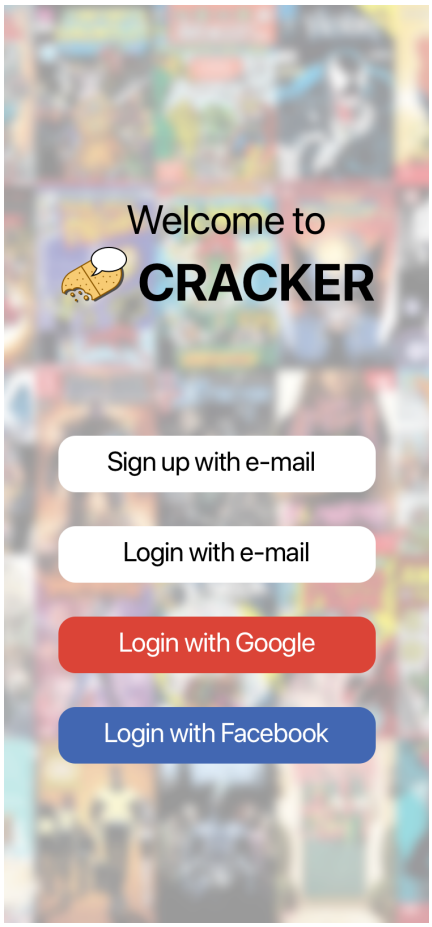
5.9 Search for a Series

The ”Search for a Series” procedure starts when the user presses the ”Search" tab in the Tab Bar. The sequence diagram shows the normal procedure.
After the user activates the "Search" tab, the application will send a request to the ”MarvelAPI” service in order to retrieve the series whose name matches the string written by the user.
Once this information is obtained, the Controller will create a UITableViewCell for each issue and insert them in a TableView.

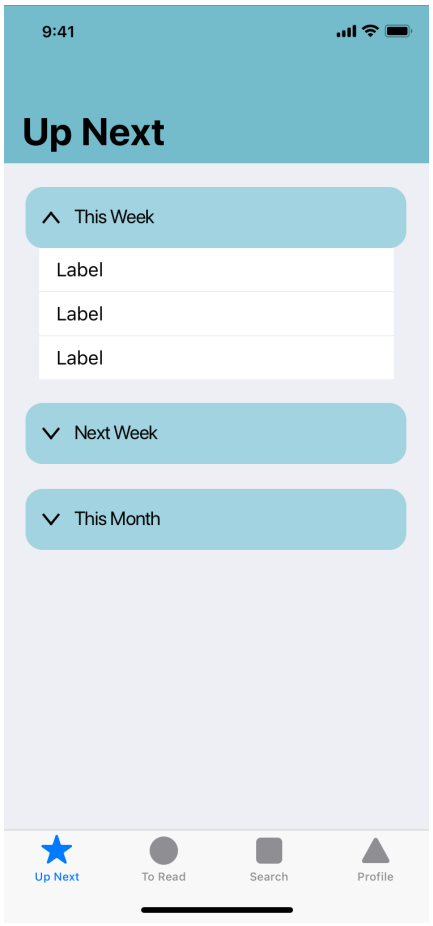


6 User Interface Design

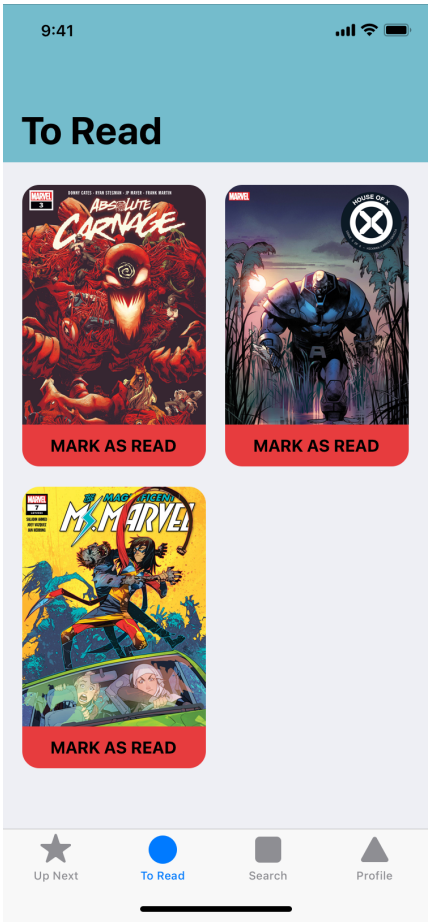
In this section we show the user interfaces of all the screens of the application.



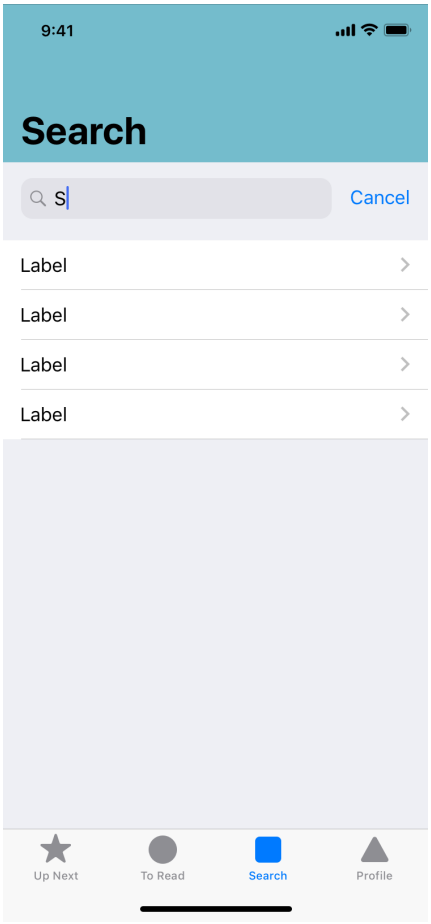
(a) Login



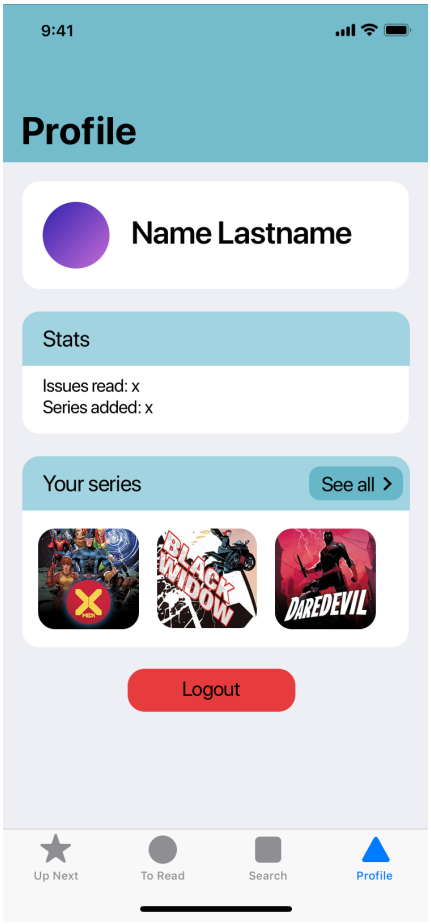
(b) Up Next



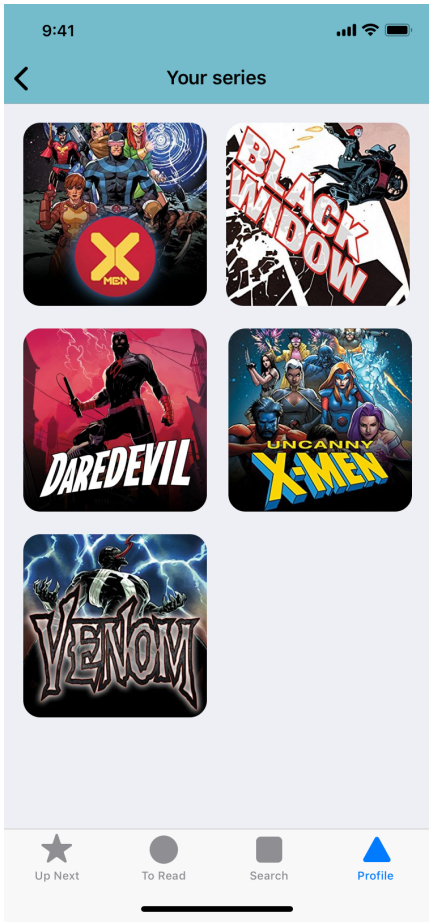
(a) To Read



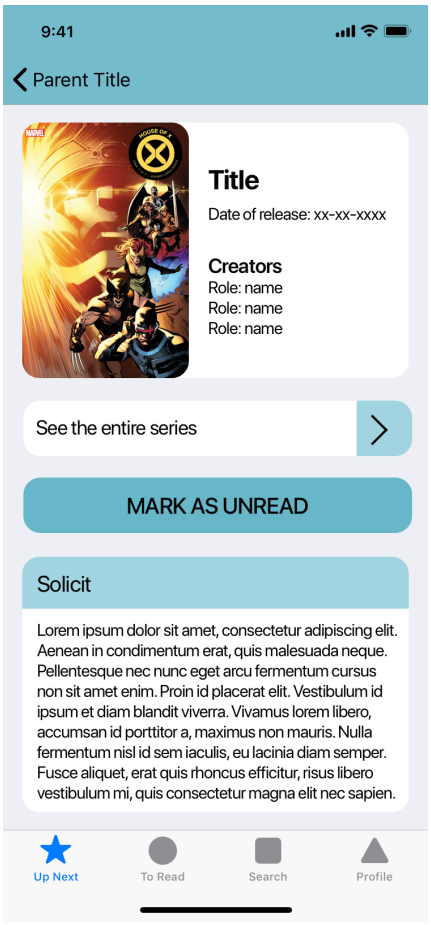
(b) Search



(a) Profile



(b) Your series



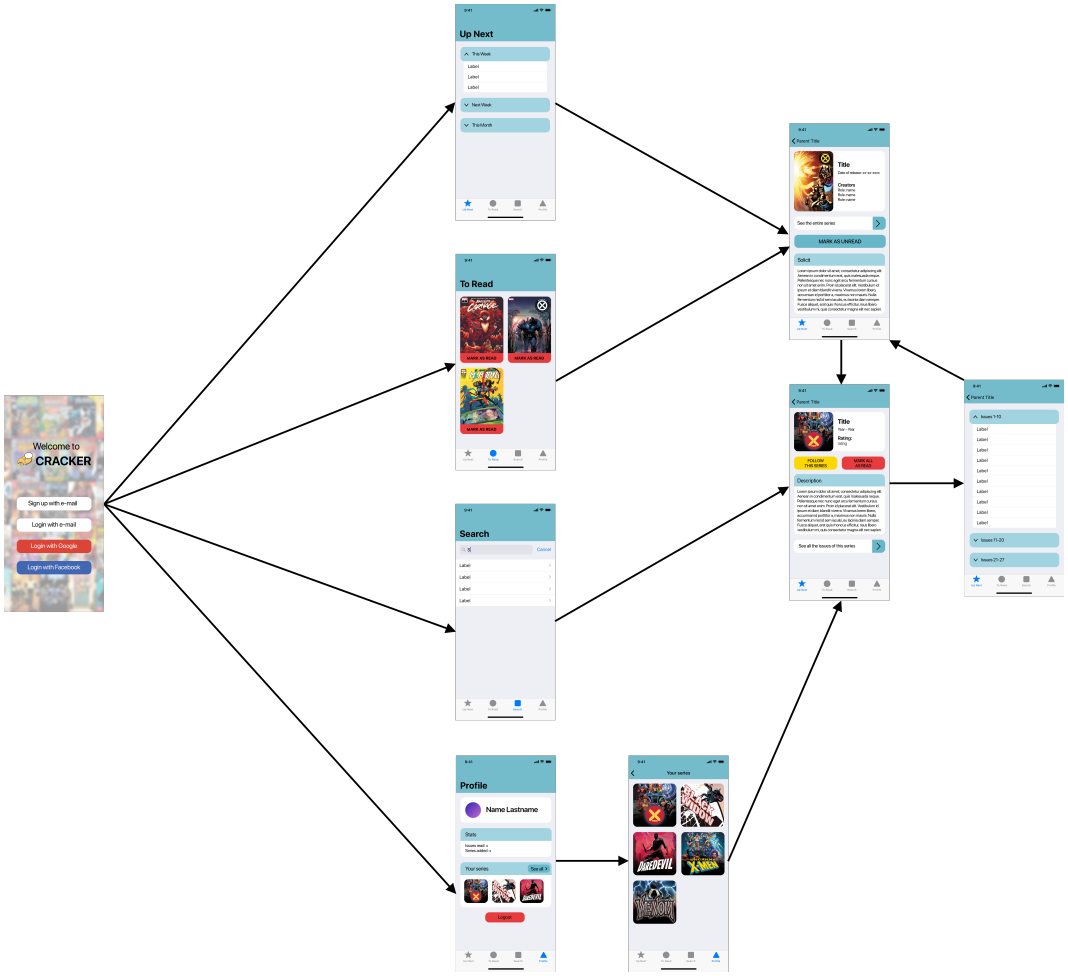
(a) Issue

(b) Series

(a) Series: light mode

(b) Series: dark mode

We close this section by showing the interaction between the different screens.



7 Software System Attributes

7.1 Reliability

As the majority of function requires an internet connection, the software is reliable as long as there are no connectivity problems.

7.2 Availability

The availability parameter also relies on the internet connection signal and on the responses provided by Marvel APIs. No problems of availability were found during the development phase.

7.3 Security

As all the information provided are retrieved from the web, there are different checks to perform in order to keep the user information safe. Furthermore, for the communication with external services, the HTTPS protocol was chosen to guarantee greater security.

8 Used Tools

The following softwares, tools and services have been used for the development of the application:

- XCode 11.3
- Sketch
- Github
- Firebase
- draw.io