Politecnico di Milano

Design and Implementation of Mobile Applications

Lode

Design Document

Mattia Righetti

September 6, 2020

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the design and prototyping phases used for the realization of the Lode mobile application. In detail, the main components, features and user experience will be discussed.

The main aim of the Lode application is visualizing information and data related to the university world.

This project is the result of the implementation of the knowledge acquired during the course "Design and Implementation of Mobile Applications" provided by the Politecnico di Milano. held by Prof. Baresi Luciano.

1.2 Intended Audience

This document is produced for those who develop, evaluate and use Lode iOS app:

- The engineers who had the idea and developed the application.
- The testers that must verify the effective implementation of all the described components and functions.
- The user who will use the application and take advantage of its functionalities.
- The future contributors who wish to develop new features.

1.3 Definitions, acronyms, abbreviations

Definitions

- **Platform**: The application as a whole.
- **Guest**: A person who can view public contents
- **User**: A guest who already performed the login operation successfully.
- **Framework**: Reusable set of libraries or classes for a software system.

Acronyms

- **MVVM**: Model - View - View Model
- **IDE**: Integrated Development Environment
- **API**: Application Programming Interface
- **UML**: Unified Modelling Language.
- **UI**: User Interface

Abbreviations

- **App**: Mobile Application

1.4 Mobile Application Scope

Lode has been developed for students that need a precise and simple instrument to manage everything concerning the university world.
In particular, the application will be divided into seven views:

- **Home**
- **Course**
- **Exams**
- **Assignments**
- **Stats**
- **Marks**
- **Average Delta Calculator**

In the "Home" view there will be the main page which will be presented to the user. It features a "card stack" that contains quick and useful data to look at, links to the statistics and the marks views and an average calculator that will tell you what your average will be by selecting the CFU of the exam.
In the "Course" view there will be a list of the courses that the user is going to take during his/her entire career, with the expected mark that the user would like to take and a final mark that is going to be visible if the user passed that exam.
In the "Exams" view there will be a list of upcoming and past exams, this is similar to a simple reminder view that will immediately tell you at a glance what are the user's next exams.
In the "Assignments" view there will be a list of assignments that the user will need to complete before a certain date.

In the "Stats" view the user will be presented with data gathered by the application, like the amount of CFUs that the user has taken, his projected average and current average given his expected and final marks, etc.
In the "Marks" view the user will be able to see every course that he/she has added to the application with its final marks, if passed, and its expected mark to better see in which courses he performed better/worse than expected.
In the "Average Delta Calculator" the user will be shown a tool to calculate his/her new average in case he/she takes one of the marks listed in the view. User has to choose how many CFUs a certain course is valid in order to get the list of marks with their new average value.

1.5 Framework

The development of Lode was achieved through the use of the native iOS SDKs, in particular by using the Swift programming language and the brand new SwiftUI framework. This choice made the app feel faster, well integrated and easier to design following Apple's guidelines.

1.6 Functional Requirements

The product provides to users a simple and user-friendly interface to:

1. Add new courses
2. Add upcoming exams
3. Add assignments
4. Set an expected mark to get on a specific course
5. Edit a course
6. Edit an exam
7. Edit an assignment
8. Show statistics based on the data provided on courses (like marks, expected marks, etc.)
9. Provide a tool to quickly calculate the new mean average by course CFUs and mark
10. Show the user expected marks and final marks grouped in a single view

1.7 Non Functional Requirements

The application must be able to:

- Run both on iPad and iPhone.
- Occupy the entire screen available.

1.8 Assumptions, Dependencies and Constraints

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. (for instance iPhone 6S or better).
- **Software limitations**: our application, due to compatibility issues of the SwiftUI framework, will only work on devices that have iOS 13+.

Assumptions and Dependencies

- **No privileged users**: there are no privileged users or administrators with particular functions.
- **OS Permission Granted**: the user will always grant to his OS's device the permission to access to all the needed services.

2 Architecture

2.1 Database

Every data that is inputted by the user or computed at run-time by the application is saved locally using Apple's Core Data framework.

2.2 Model–view–view model

For the implementation of the application I've chosen a MVVM architecture. This choice was made mainly because Apple strongly encourages to do so with the new SwiftUI framework which is made to be reactive to changes.

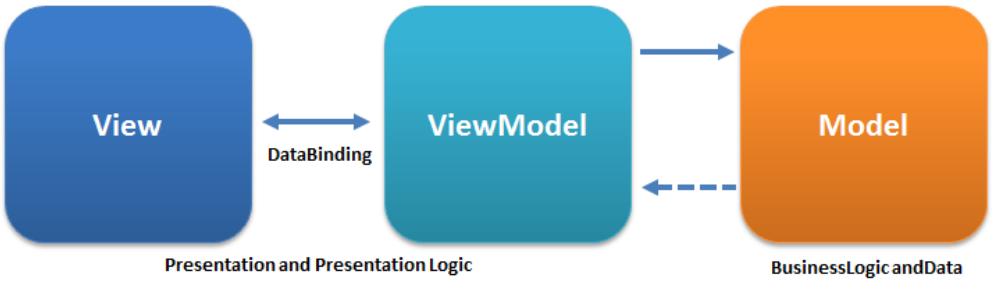


Figure 2.1: MVVM architecture

Traditional MVVM pattern:

- **Model**: Model refers either to a domain model, which represents real state content (an object-oriented approach), or to the data access layer, which represents content (a data-centric approach).
- **View**: As in the model-view-controller (MVC) and model-view-presenter (MVP) patterns, the view is the structure, layout, and appearance of what a user sees on the screen.[6] It displays a representation of the model and receives the user's interaction with the view (clicks, keyboard, gestures, etc.), and it forwards the handling of these to the view model via the data binding (properties, event callbacks, etc.) that is defined to link the view and view model.
- **View Model**: The view model is an abstraction of the view exposing public properties and commands. Instead of the controller of the MVC pattern, or the

presenter of the MVP pattern, MVVM has a binder, which automates communication between the view and its bound properties in the view model. The view model has been described as a state of the data in the model.[7] The main difference between the view model and the Presenter in the MVP pattern, is that the presenter has a reference to a view whereas the view model does not. Instead, a view directly binds to properties on the view model to send and receive updates. To function efficiently, this requires a binding technology or generating boilerplate code to do the binding.[6]

3 Use case functional requirements analysis

This section describes how users can interact with Lode 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.

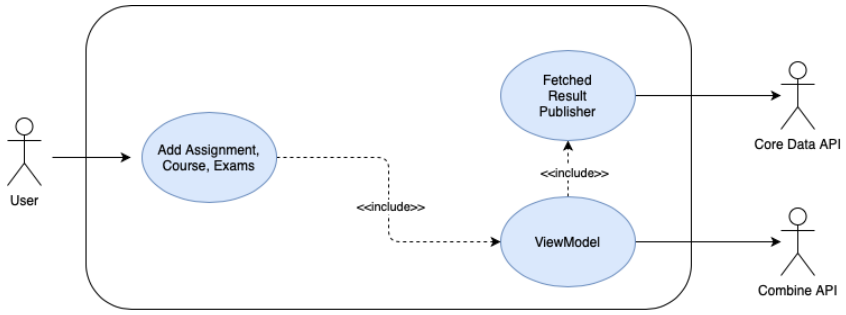


Figure 3.1: Use case related to addition and handling of courses, exams and assignments

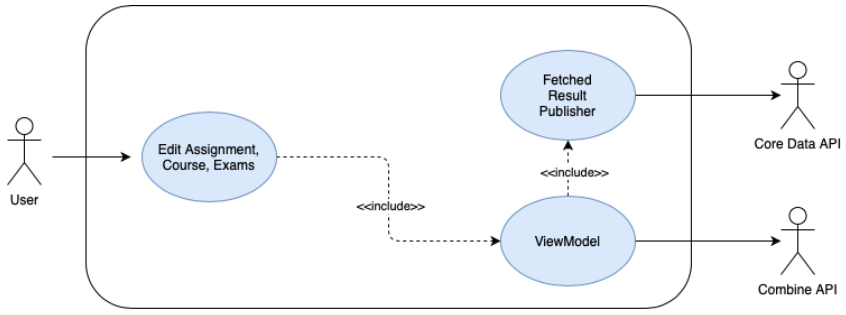


Figure 3.2: Use case related to editing and handling of courses, exams and assignments

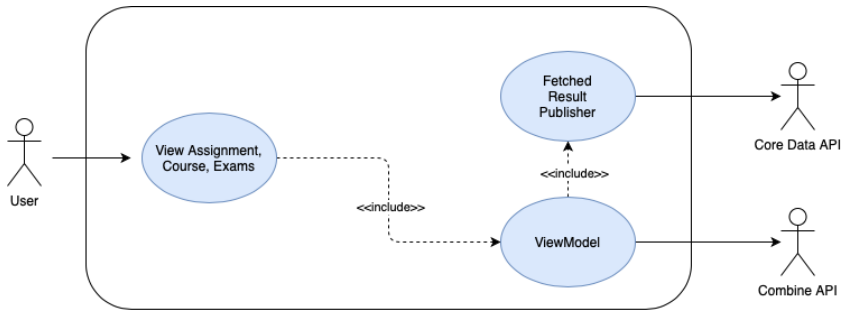


Figure 3.3: Use case related to visualization and handling of courses, exams and assignments

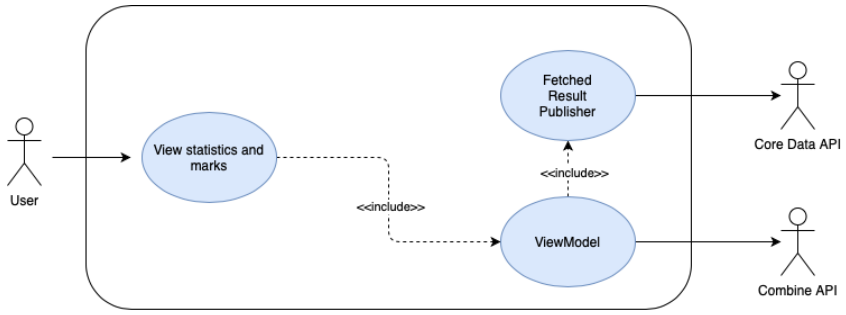


Figure 3.4: Use case related to visualization and handling of statistics and marks

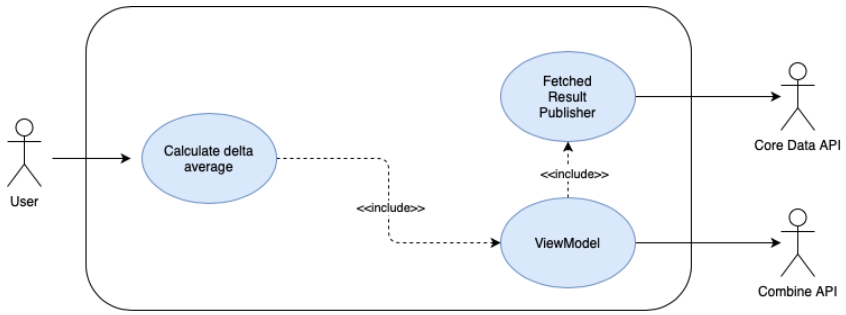


Figure 3.5: Use case related to calculation of the delta average

View Courses

Name	View Courses
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Courses" tab on the bottom of the application • The app will show you a list of the user's courses
Exit condition	User sees courses
Exceptions	The user hasn't added any course previously.

View Exams

Name	View Exams
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Exams" tab on the bottom of the application • The app will show you a list of the user's exams
Exit condition	User sees exams
Exceptions	The user hasn't added any exam previously.

View Assignments

Name	View Assignments
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Assignments" tab on the bottom of the application • The app will show you a list of the user's assignments
Exit condition	User sees assignments
Exceptions	The user hasn't added any assignment previously.

Add Course

Name	Add Course
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Courses" tab on the bottom of the application • The user taps on the plus icon on the top-right section of the navigation bar • The app will push a modal form on top of the courses list to complete • The user fills in all fields of the modal form • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees a newly added course to the courses list and statistics gets updated with its latest data.
Exceptions	None.

Add Exam

Name	Add Exam
Actor	User
Entry Condition	Exam that the user wants to add has to be of a course already present in the courses list.
Goal	14
Event Flow	• The user opens the application • The user presses "Exam" tab on the bottom of the application • The user taps on the plus icon on the top-right section of the navigation bar • The app will push a modal form on top of the exams list to complete • The user fills in all fields of the modal form • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees a newly added exam to the exams list.
Exceptions	No course is present in the courses list.

Add Assignments

Name	Add Assignment
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Assignment" tab on the bottom of the application • The user taps on the plus icon on the top-right section of the navigation bar • The app will push a modal form on top of the assignment list to complete • The user fills in all fields of the modal form • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees a newly added assignment to the assignments list and statistics gets updated with its latest data.
Exceptions	None.

Add Course (alternative)

Name	Add Course
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Home" tab on the bottom of the application • The user taps on the plus icon on the top-right section of the navigation bar • The app will push a modal form on top of the exams list to complete • The user fills in all fields of the modal form • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees a newly added exam to the exams list.
Exceptions	No course is present in the courses list.

Add Exam (alternative)

Name	Add Exam
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Home" tab on the bottom of the application • The user taps on the plus icon on the top-right section of the navigation bar • The app will push a modal form on top of the exams list to complete • The user fills in all fields of the modal form • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees a newly added exam to the exams list.
Exceptions	No course is present in the courses list.

Add Assignment (alternative)

Name	Add Assignment
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Home" tab on the bottom of the application • The user taps on the plus icon on the top-right section of the navigation bar • The app will push a modal form on top of the exams list to complete • The user fills in all fields of the modal form • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees a newly added exam to the exams list.
Exceptions	No course is present in the courses list.

Edit Course

Name	Edit Course
Actor	User
Entry Condition	User wants to change something concerning a course present in the courses list
Goal	10
Event Flow	• The user opens the application • The user presses "Courses" tab on the bottom of the application • The user taps on the "Edit" button on the top-left section of the navigation bar • The user taps on the course that he/she wants to edit • The app will push a modal form on top of the course list with all its current data • The user edits the interested fields • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees an updated course to the courses list and statistics gets updated with its latest data.
Exceptions	None.

Edit Exam

Name	Edit Exam
Actor	User
Entry Condition	User wants to change something concerning an exam present in the courses list
Goal	10
Event Flow	• The user opens the application • The user presses "Exams" tab on the bottom of the application • The user taps on the "Edit" button on the top-left section of the navigation bar • The user taps on the exam that he/she wants to edit • The app will push a modal form on top of the exams list with all its current data • The user edits the interested fields • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees an updated exam to the exams list.
Exceptions	None.

Edit Assignment

Name	Edit Assignment
Actor	User
Entry Condition	User wants to change something concerning an assignment present in the assignments list
Goal	10
Event Flow	• The user opens the application • The user presses "Assignments" tab on the bottom of the application • The user taps on the "Edit" button on the top-left section of the navigation bar • The user taps on the assignment that he/she wants to edit • The app will push a modal form on top of the assignments list with all its current data • The user edits the interested fields • The user presses the "Done" button on the top-right section of the navigation bar
Exit condition	User sees an updated assignment to the assignments list.
Exceptions	None.

Check Statistics

Name	Add Assignment
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Statistics" button in the "Home" tab on the bottom of the application
Exit condition	User sees a list of useful data about his/her student career.
Exceptions	None.

Check Marks

Name	Check marks
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Marks" button in the "Home" tab on the bottom of the application
Exit condition	User sees a list of expected marks side by side with the final mark, if present, in a specific course.
Exceptions	None.

Check Delta Average

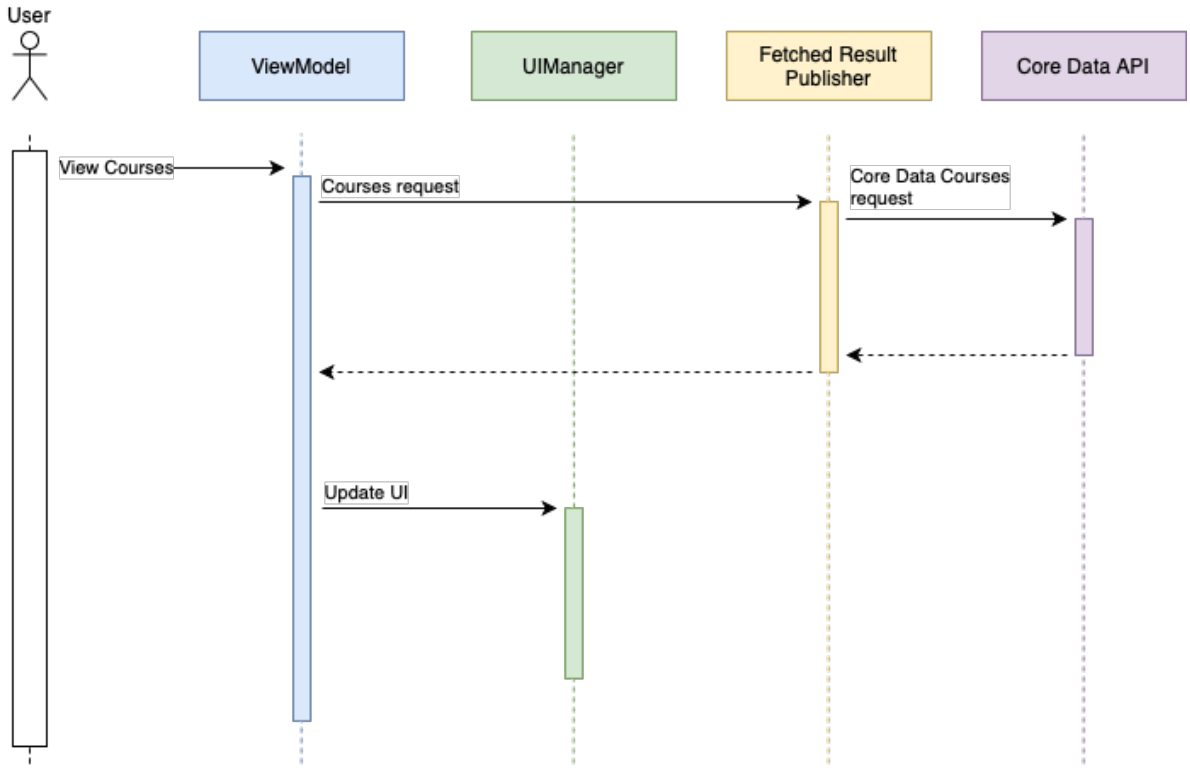
Name	Check delta average
Actor	User
Entry Condition	None
Goal	14
Event Flow	• The user opens the application • The user presses "Delta Average Calculator" button in the "Home" tab on the bottom of the application • The user sets the number of CFUs that he/she prefers
Exit condition	User sees a list of marks going from 18 to 30 and its delta average difference that that mark is going to make.
Exceptions	None.

4 Sequence Diagram

In order to explain our work and facilitate further implementations I've 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.

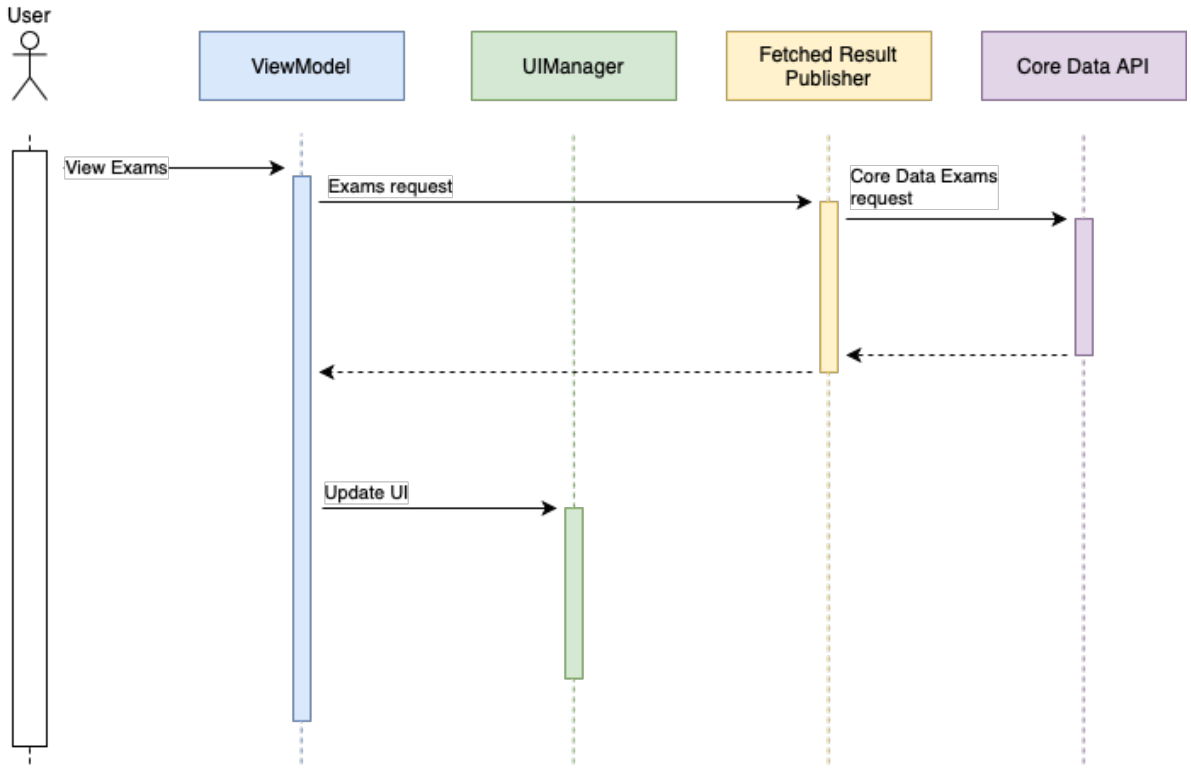
View Courses

The "View Course" procedure starts when the user navigated to the cCurses section using the tab at the bottom. Once pressed, the ViewModel will use his course subscriber to listen for values published by the Core Data API publisher. When every value is retrieved SwiftUI will use the ViewModel course publisher to update the list of Courses.



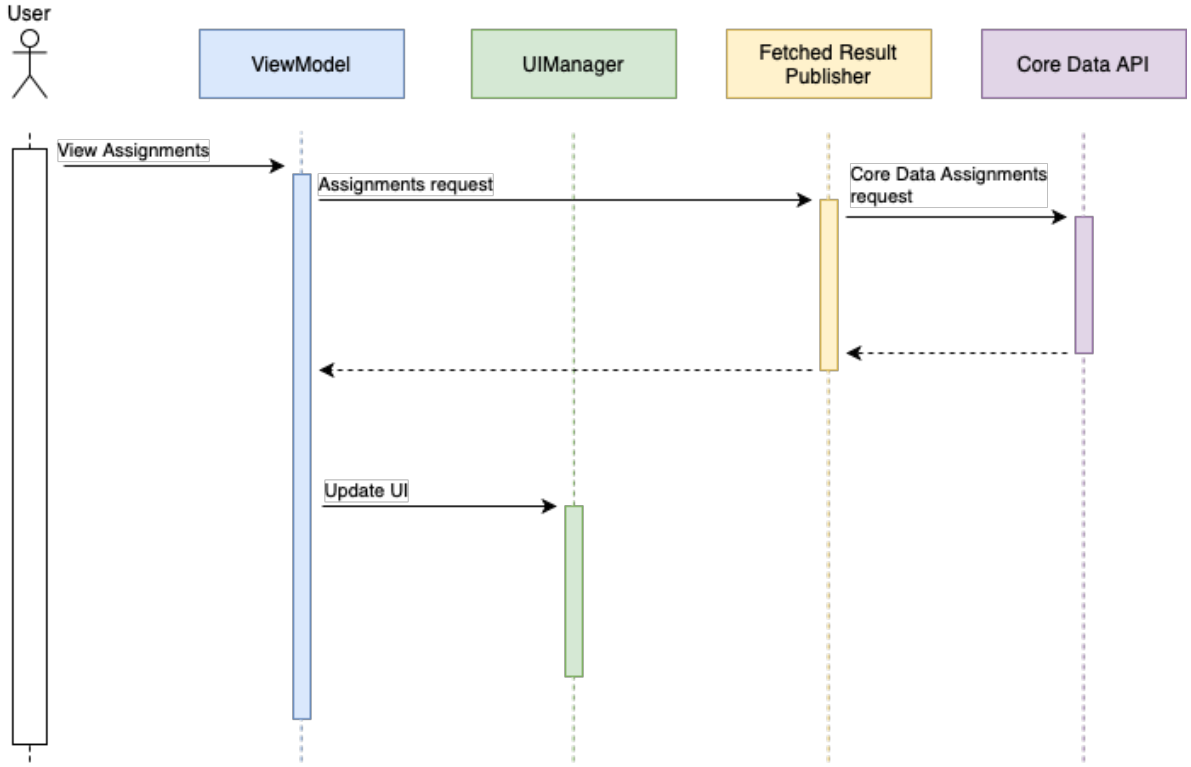
View Exams

The "View Exams" procedure starts when the user navigated to the Exams section using the tab at the bottom. Once pressed, the ViewModel will use his course subscriber to listen for values published by the Core Data API publisher. When every value is retrieved SwiftUI will use the ViewModel course publisher to update the list of Exams.



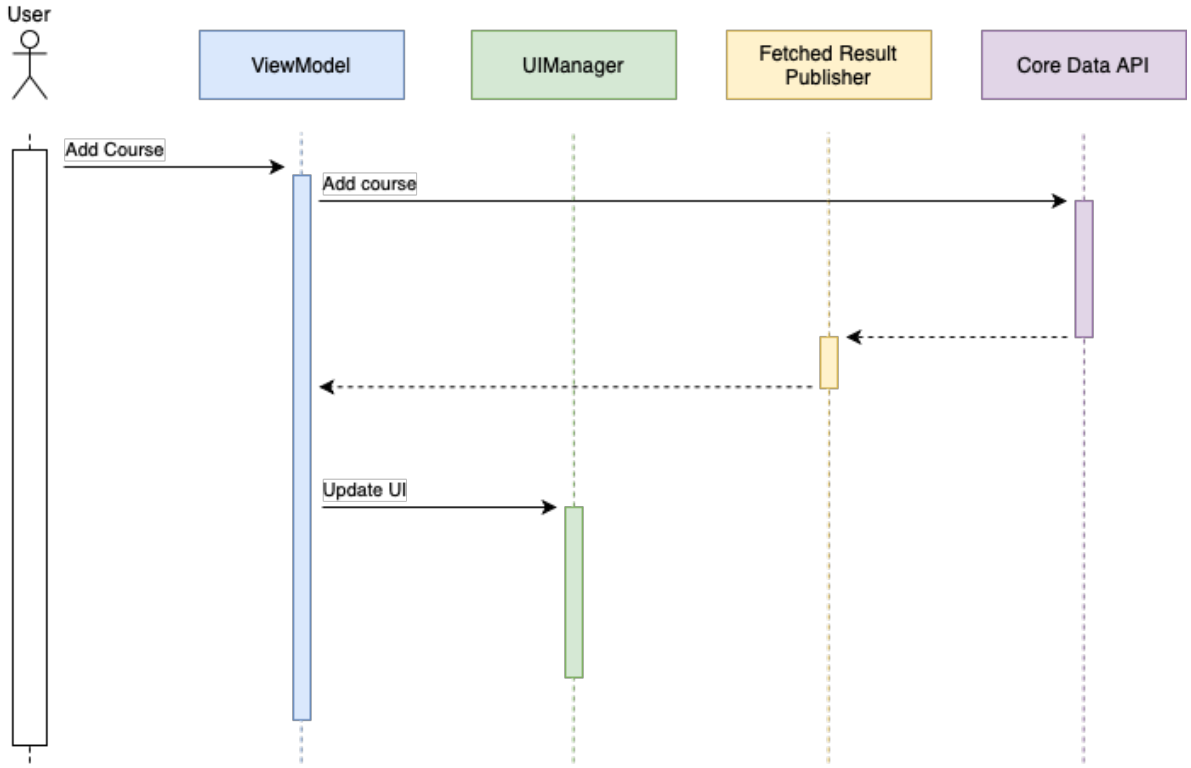
View Assignments

The "View Assignments" procedure starts when the user navigated to the Assignments section using the tab at the bottom. Once pressed, the ViewModel will use his course subscriber to listen for values published by the Core Data API publisher. When every value is retrieved SwiftUI will use the ViewModel course publisher to update the list of Assignments.



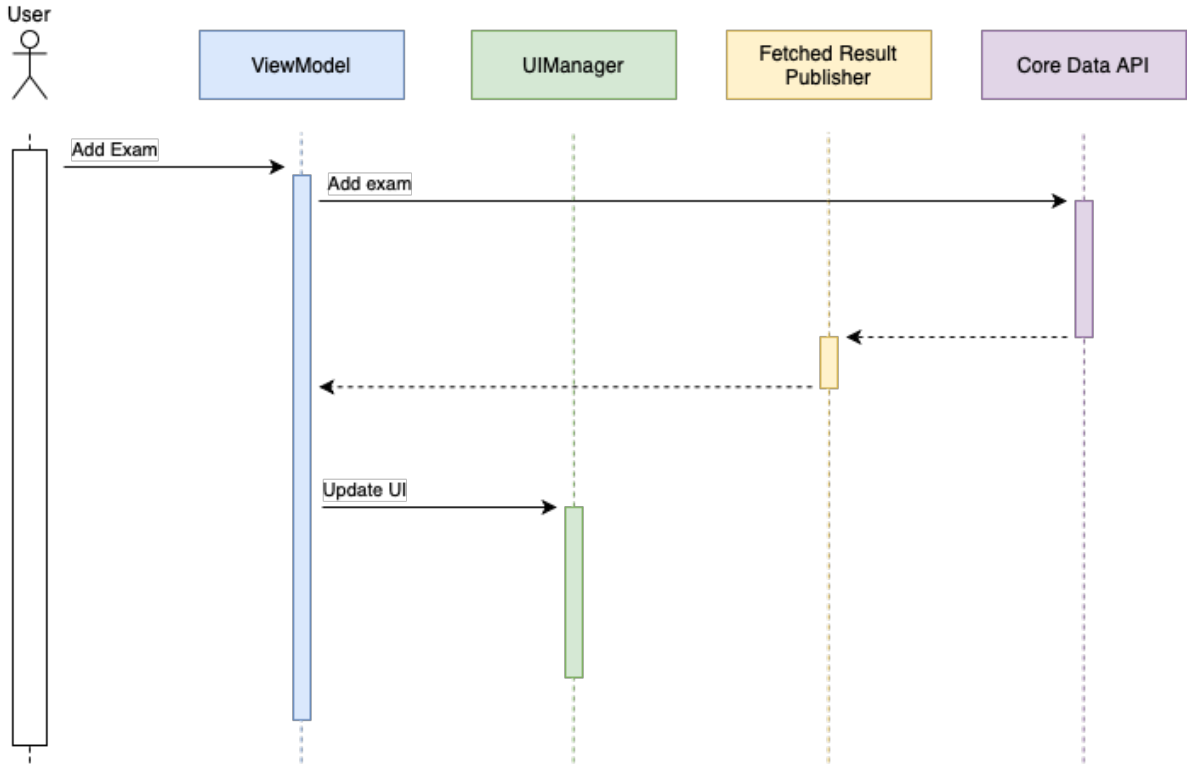
Add Course

The "Add Course" procedure starts when the user presses on the plus button visible in the Courses section of the app. Once pressed, a form will be presented to the user who will fill every required element and the confirm the addition or cancel it. As soon as the user will confirm the addition, data will be written to local storage through Core Data API. This state changing addition will trigger the Core Data course publisher which will send out the new values added. After that, the ViewModel, which is subscribed to the Core Data course publisher, will update its courses and publish them so that SwiftUI can update the list viewed by the user.



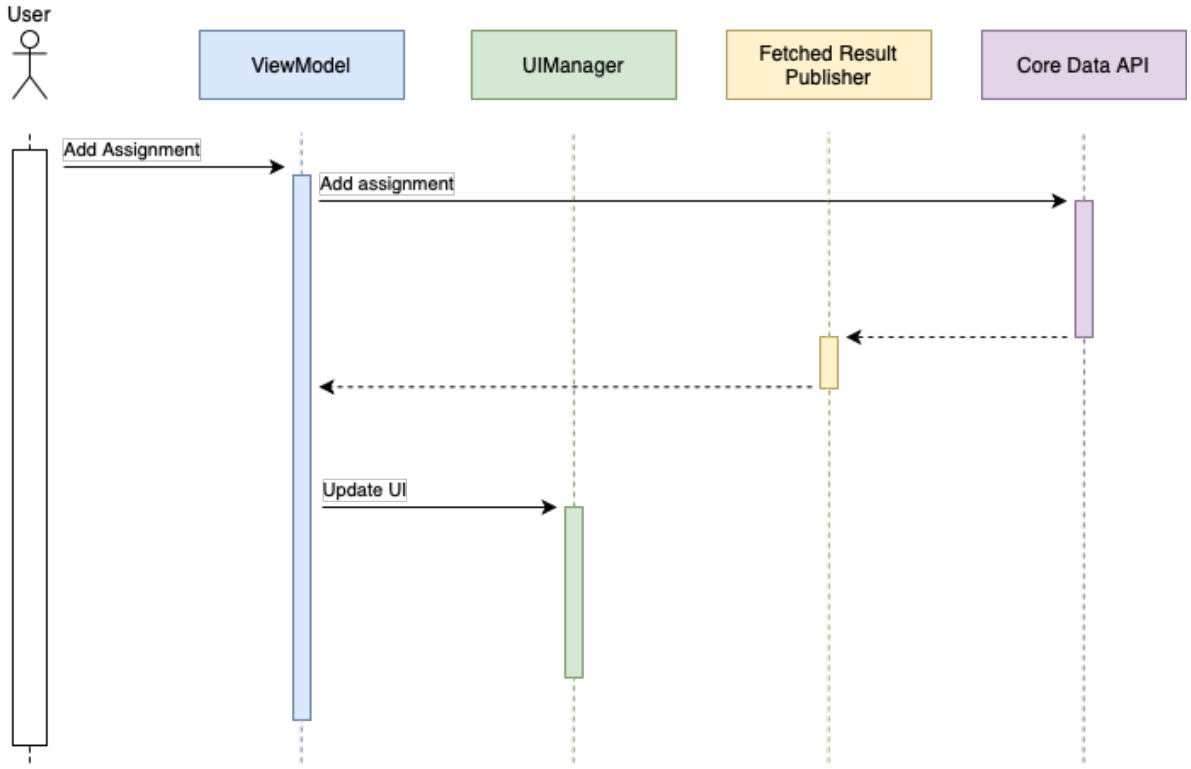
Add Exam

The "Add Exam" procedure starts when the user presses on the plus button visible in the Exam section of the app. Once pressed, a form will be presented to the user who will fill every required element and the confirm the addition or cancel it. As soon as the user will confirm the addition, data will be written to local storage through Core Data API. This state changing addition will trigger the Core Data exam publisher which will send out the new values added. After that, the ViewModel, which is subscribed to the Core Data exam publisher, will update its exams and publish them so that SwiftUI can update the list viewed by the user.



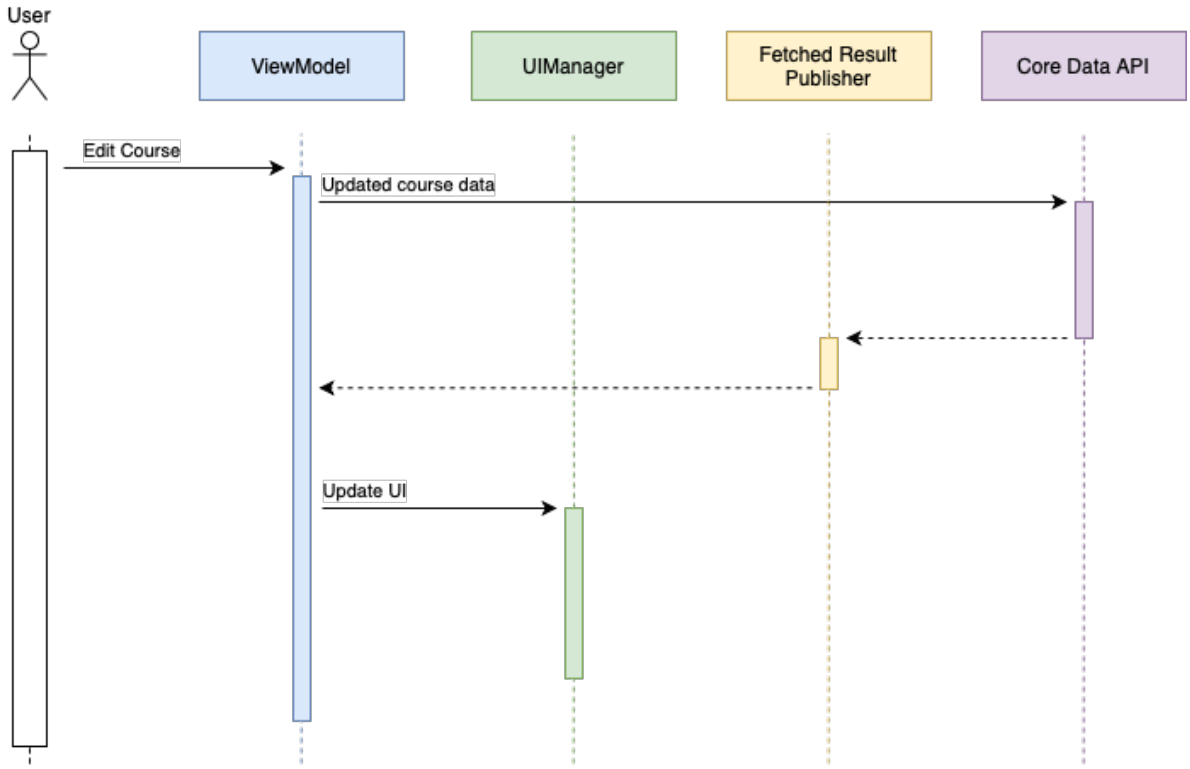
Add Assignment

The "Add Assignment" procedure starts when the user presses on the plus button visible in the Assignments section of the app. Once pressed, a form will be presented to the user who will fill every required element and the confirm the addition or cancel it. As soon as the user will confirm the addition, data will be written to local storage through Core Data API. This state changing addition will trigger the Core Data assignment publisher which will send out the new values added. After that, the ViewModel, which is subscribed to the Core Data assignment publisher, will update its assignment and publish them so that SwiftUI can update the list viewed by the user.



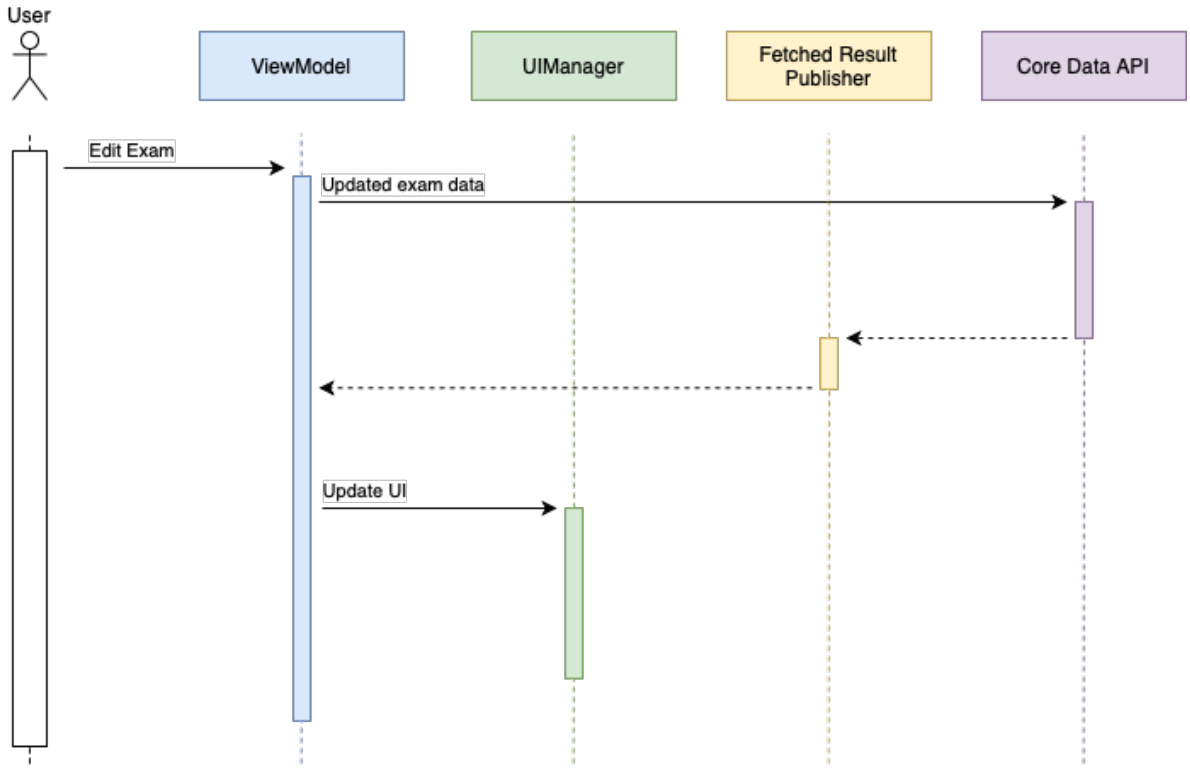
Edit Course

The "Edit Course" procedure starts when the user presses on the "Edit" button visible in the Course section of the app. Once pressed, the user has to press on the the course that he/she wants to change. A new form with fields pre-filled with that particular exam's data will show up and the user can make all the changes that he needs to do. By pressing the "Done" button, the user will submit the new changes to the Core Data API. This state changing addition will trigger the Core Data course publisher which will send out the new values added. After that, the ViewModel, which is subscribed to the Core Data course publisher, will update its courses and publish them so that SwiftUI can update the values of the modified course viewed by the user.



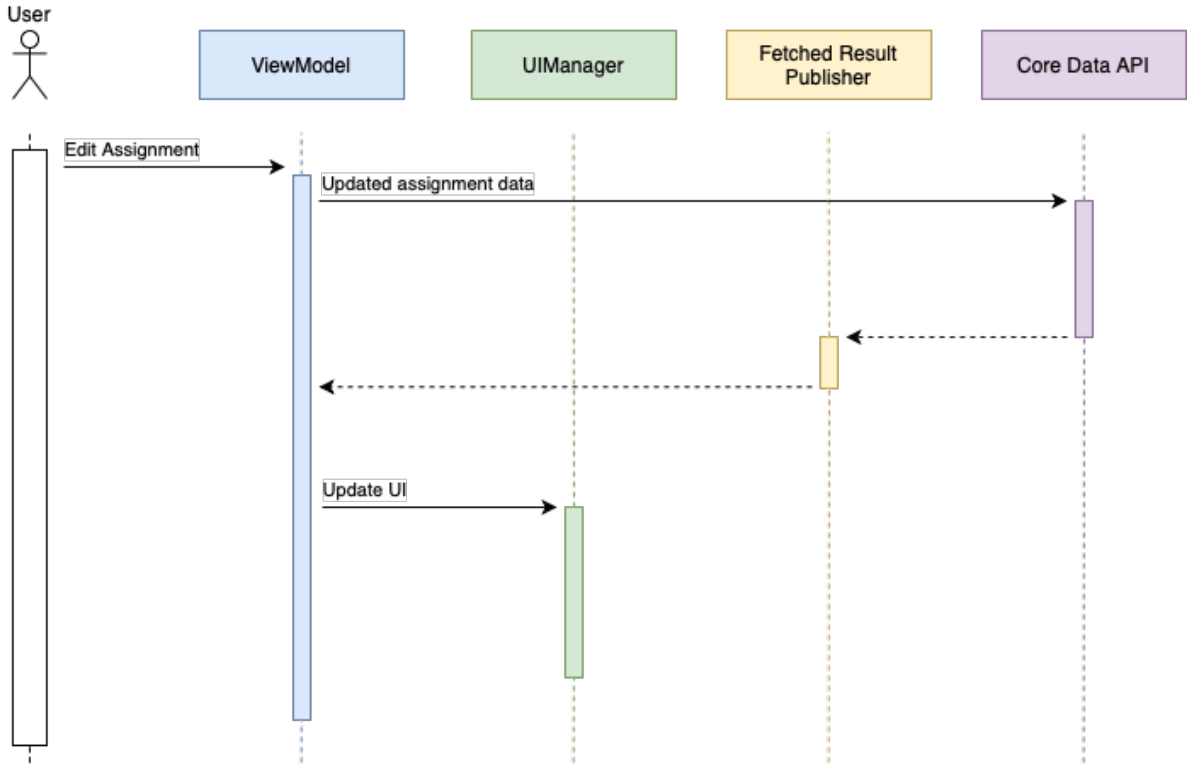
Edit Exam

The "Edit Exam" procedure starts when the user presses on the "Edit" button visible in the Exam section of the app. Once pressed, the user has to press on the the exam that he/she wants to change. A new form with fields pre-filled with that particular exam's data will show up and the user can make all the changes that he needs to do. By pressing the "Done" button, the user will submit the new changes to the Core Data API. This state changing addition will trigger the Core Data exam publisher which will send out the new values added. After that, the ViewModel, which is subscribed to the Core Data exam publisher, will update its exams and publish them so that SwiftUI can update the values of the modified exam viewed by the user.



Edit Assignment

The "Edit Assignment" procedure starts when the user presses on the "Edit" button visible in the Assignment section of the app. Once pressed, the user has to press on the the assignment that he/she wants to change. A new form with fields pre-filled with that particular assignment's data will show up and the user can make all the changes that he needs to do. By pressing the "Done" button, the user will submit the new changes to the Core Data API. This state changing addition will trigger the Core Data assignment publisher which will send out the new values added. After that, the ViewModel, which is subscribed to the Core Data assignment publisher, will update its assignment and publish them so that SwiftUI can update the values of the modified assignment viewed by the user.



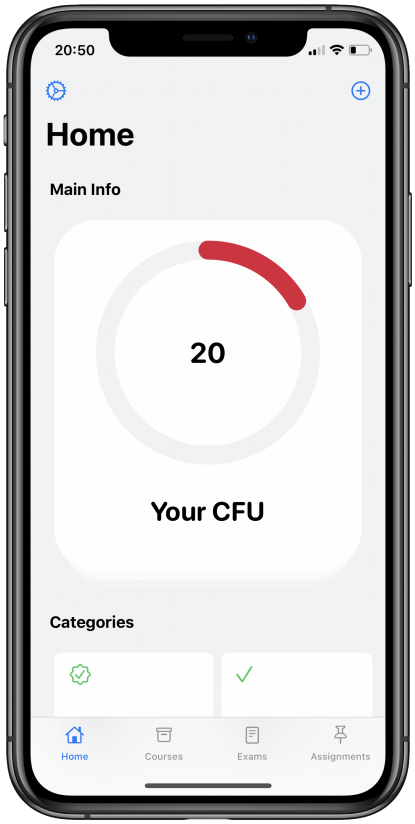
5 User Interfaces

In this chapter the user interface of each page will be discussed and displayed, highlighting the functionalities of the Lode application.

5.1 iOS variant

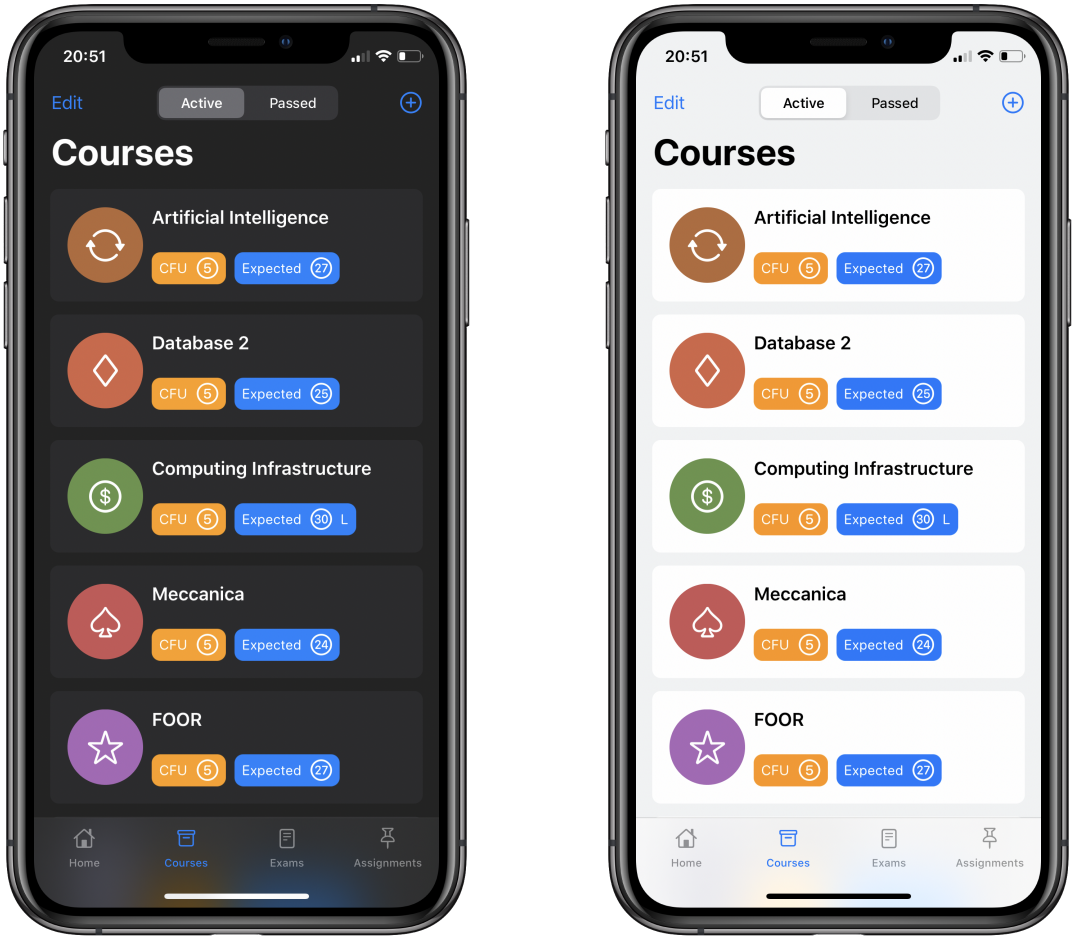
Home Page

The Home page has been designed to make the user able to look at what matter the most in a quick and intuitive way. On the very top you have the navigation bar with a gear and plus icon that will respectively make the user adjust some settings and quickly add assignments, courses and exams. Immediately under that the user will find a deck of cards that are swipable and contains important and concise data. Below that the user can see a "Categories" section that are links to the Marks and Stats views. Under that the user will see a "Tools" section that has a link to the Average Delta Calculator view.



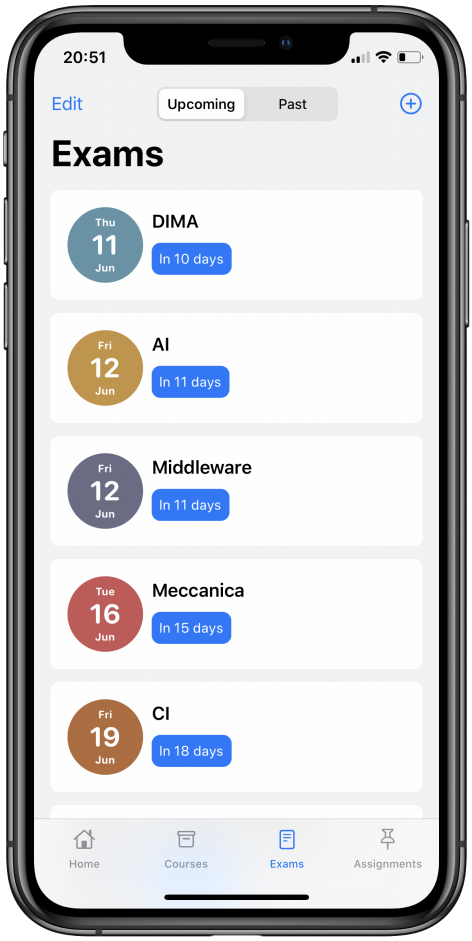
Courses

The Courses tab will present you with your active and passed courses. Active courses are courses that still don't have a final mark associated to them, passed courses are courses that have been passed and that have a final mark associated to them. Each course is contained in a beautifully designed card, both in dark and light mode, that will display the exam name, an icon that can be chosen to give the course a specific meaning and two badges at the bottom. If the user is in the Active section of the list those badge will display the number of CFUs that that course has and its expected mark, if the user is in the Passed section of the list those badges will display the number of CFUs of that course and the final mark that the user got in that specific course.



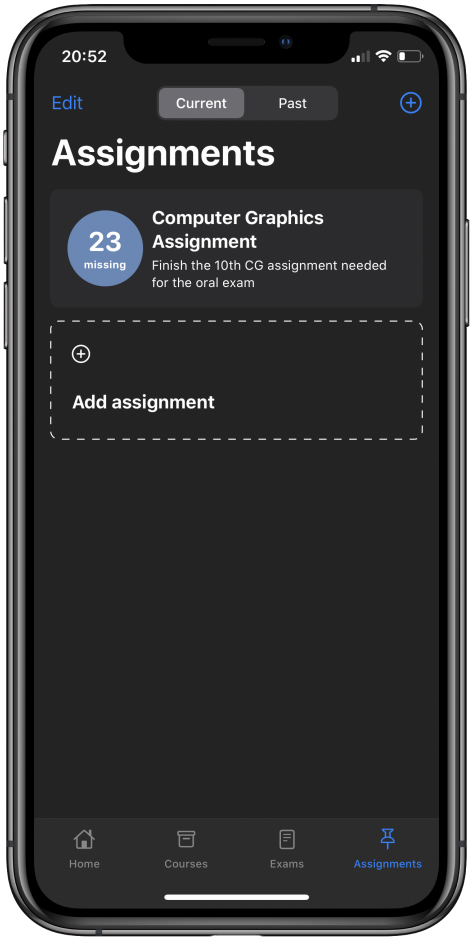
Exams

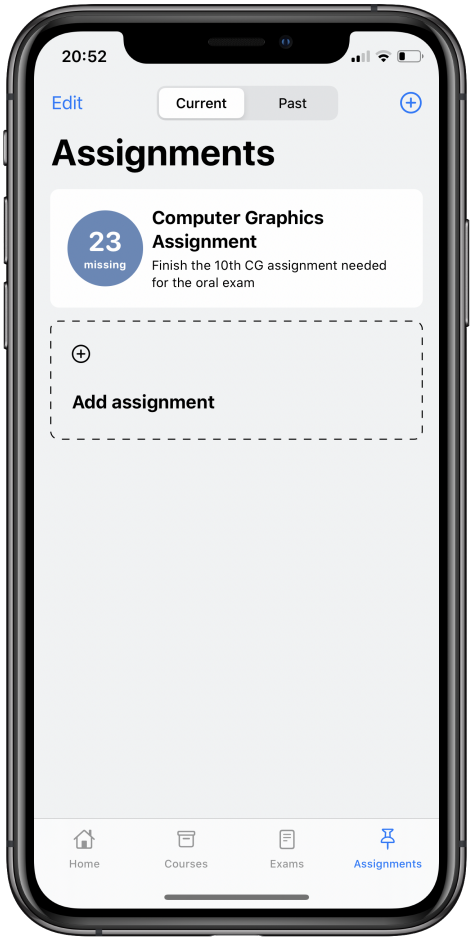
The Exam tab will present you with your upcoming and past exams. Upcoming exams are exams that has to be taken in the future, past exams are exams that has a date that has expired at the time you're looking at the list. Each exam is contained in a beautifully designed card, both in dark and light mode, that will display the exam name, when its going to be and how many days you got left to prepare for it. The capsule that contains the days left is blue when you have more than 5 days to go, it will become red as soon as you reach 4 days.



Assignments

The Assignment tab will present you with your upcoming and past assignments. Upcoming assignments are to-do things that the user needs to complete before due time. The app will show you how many days are missing to the due date so that a user can better organize by counting how many days that task will take.



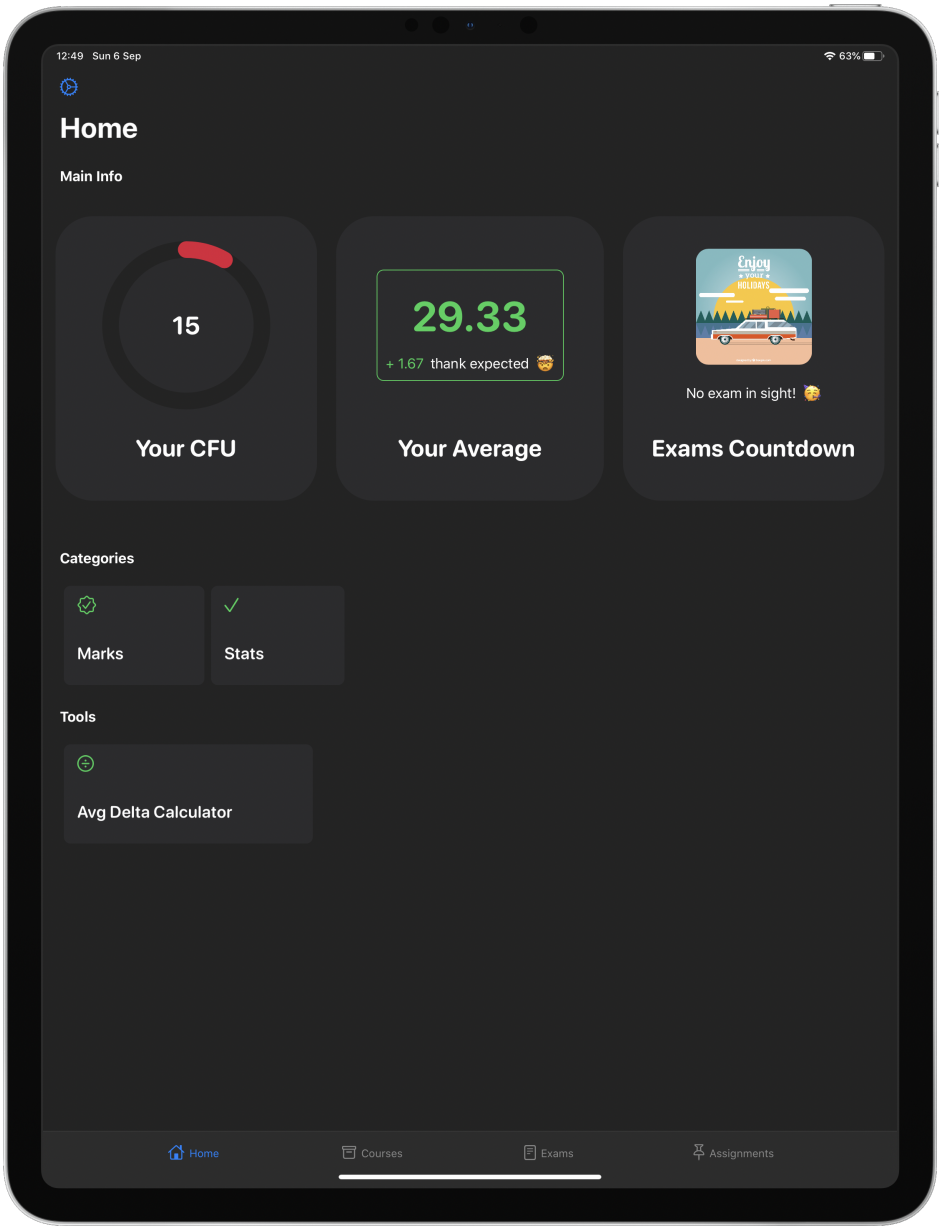


5.2 iPadOS variant

This application has also been designed for iPadOS where you have a lot more screen estate and view components could be accommodated in a much relaxed way.

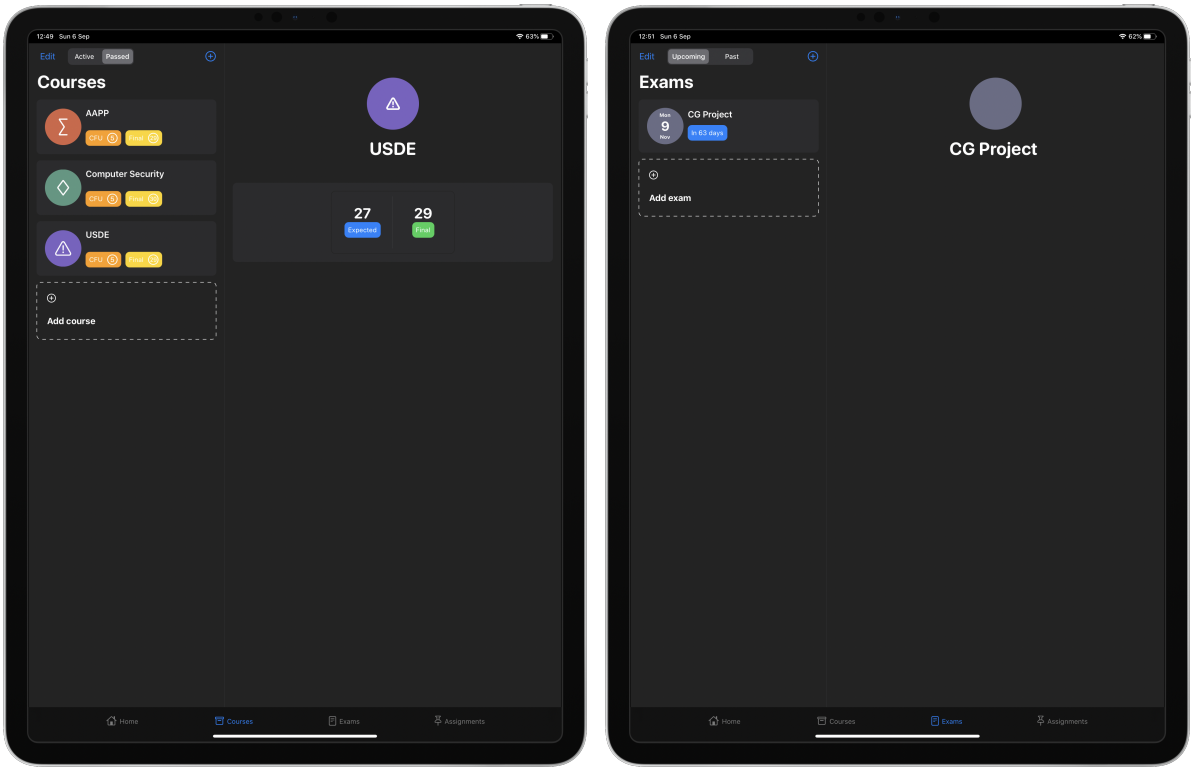
Home

In the iPad variant, you don't have a swipable stack of cards, but you'll see all the cards laid out side by side thanks to the larger screen of the device.



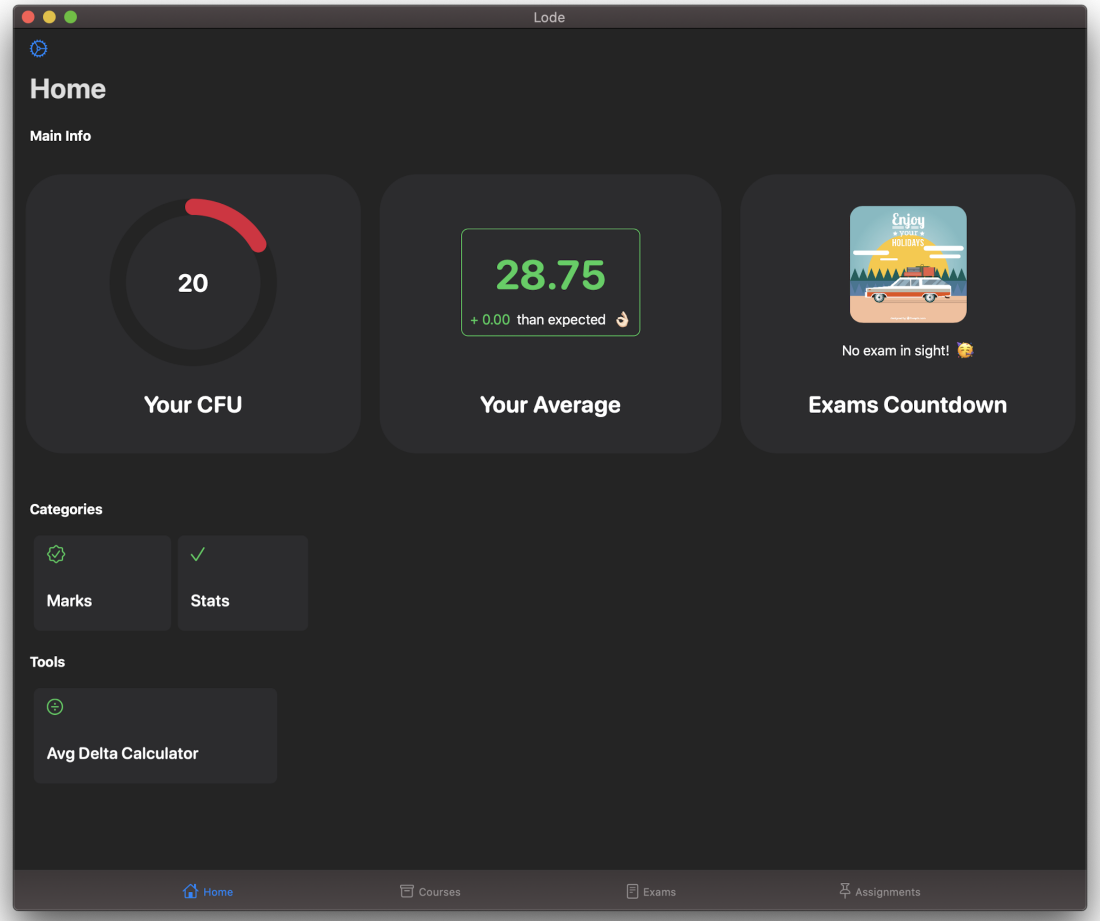
Courses, Exams and Assignments

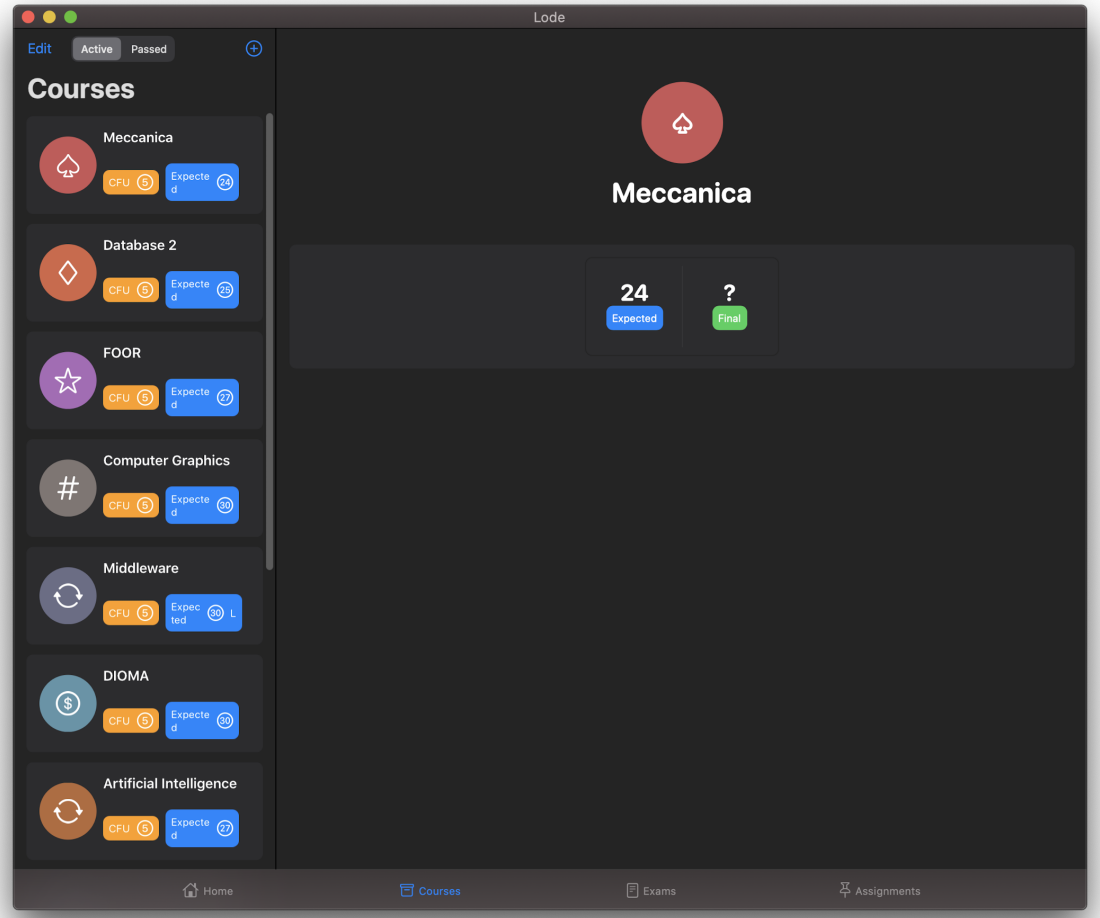
In the iPad variant, courses, exams and assignments dedicated pages will have the usual iPhone list on the left and on the right side you will find a larger view to accommodate some info about the selected element.



5.3 macOS Catalyst variant

With the latest SwiftUI framework, Apple made it really easy to import iPadOS designed apps to the Mac. Hence, this app can easily be launched on macOS.





5.3.1 Issues

Due to the early stage of the SwiftUI framework release, the macOS Catalyst imported app was not as responsive as it was on the iPad and on the iPhone, this made the app not enjoyable which made me leave that feature as a side project that could be explored as soon as SwiftUI grows as a framework and becomes more stable.

6 Test Cases

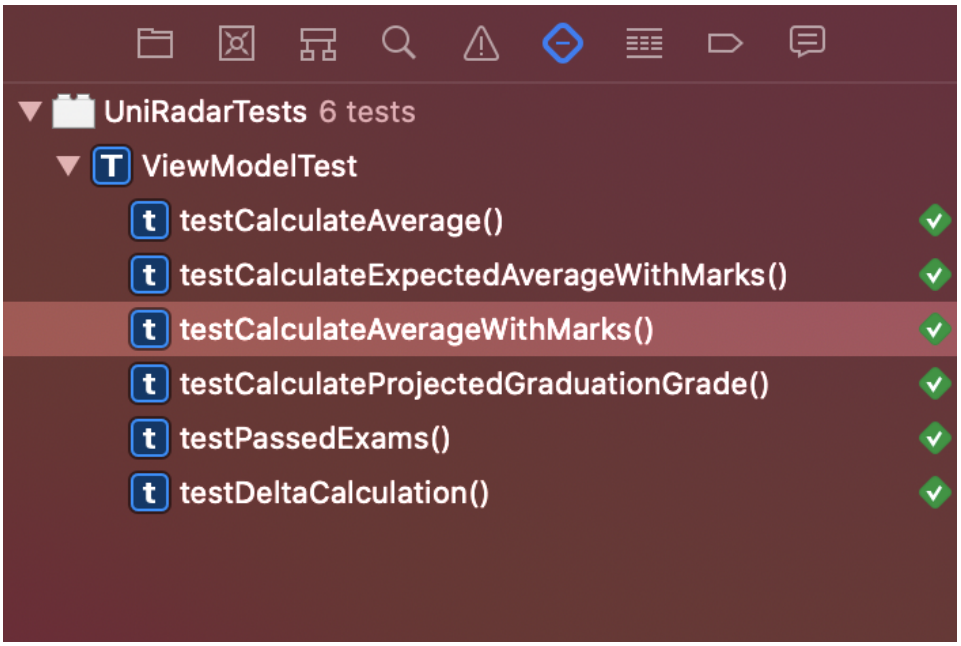
Tests is an application are crucial, especially in Lode were a single element does a lot of math calculations and could potentially return unexpected results. The testing part of the project was relatively smaller and simpler that the general app's implementation, but not less important.

6.1 Difficulties

Since I made use of the new reactive publisher-subscriber framework, Combine from Apple, and due to its early stage of development (Combine when this document was released 6 months ago), Apple didn't give any particular indication on how test that framework properly. Thanks to the Rx-Swift community, from which Combine took a lot of inspiration, I was able to study which were the best practices and approaches to test Combine framework components.

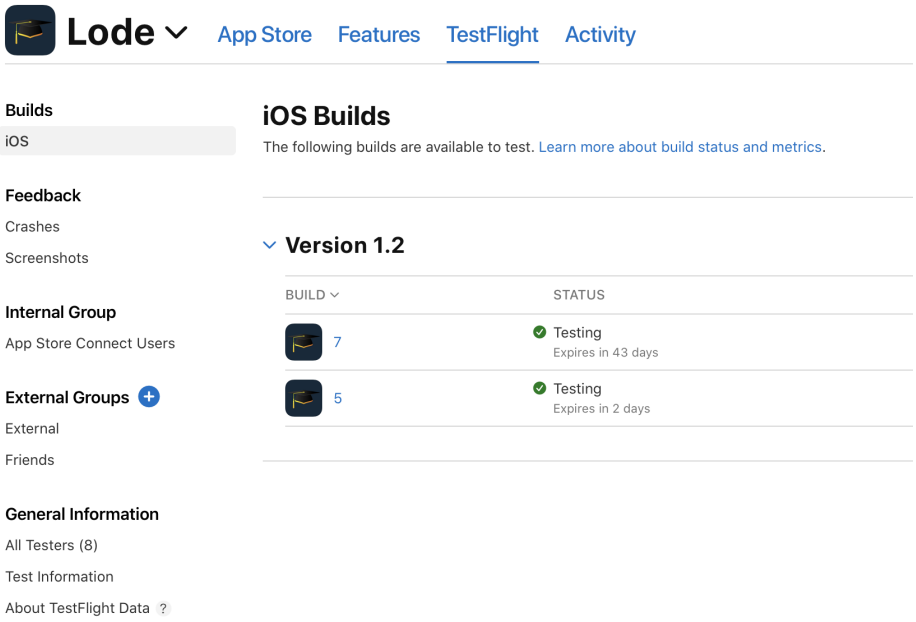
6.2 Results

This is the result of the ViewModelTest suite which is an element that thoroughly tests the ViewModel of the application, which is the core-element and brain of the app.



6.3 TestFlight

The application was further tested by a pool of ten to fifteen different users, which tested thoroughly the application and its features, giving accurate feedback on possible UI improvements and bugs that made those features less effective and, sometimes, could crash the application.



7 Software System Attribute

7.1 Reliability

As the majority of function requires a local storage, the software is reliable as long as there is enough space on the device.

7.2 Availability

No problems of availability were found during both the development phase and public beta testing with users.

7.3 Security

The security of the Lode application was the main concern during the development. All the computations and the data stored is never exposed to the internet and is all stored locally, therefore user security is never put in danger.

7.4 Maintainability

The entire application is very maintainable as the code is very readable, in particular in several critical function. Therefore any developer who wants to improve it or make changes is able to do it without relevant difficulty.

7.5 Usability

The usability and user experience was another big concern during the development phase. In order to improve it, several beta versions of the application was distributed among users who tested it thoroughly. The result of this test highlighted usability issues and bugs that were immediately taken care of, bringing a bug-free and clean product.

8 Tool & Software Used

For the development of this application, the following tools and software has been used:

(a) Xcode 11.6

(b) Draw.io

(a) Itunes Connect

(b) Testflight

(a) Adobe XD

(b) GitHub