karma

Design and Implementation of Mobile Application

Design document *

Giovanni Demasi, *10656704*
Tommaso Bucaioni, *10868702*

Prof. Luciano Baresi

A.Y. 2022/23

*Version 1.0

Contents

1 Introduction

1.1 Purpose of this document

This document aims to outline the way in which the karma application has been designed and implemented.

1.2 Document structure

The Design document is made of the following sections:

1. Introduction: it contains all the information needed to understand the following document.
2. Requirements: it contains the entire list of functional requirements that will be implemented in karma application, to have a clear understanding about which features the application will provide
3. Architectural design: it provides details about the system architecture, its components and how they interact.
4. User Interface design: it describes the UI/UX design and it contains mockups of the user interface.
5. Services: it describes the external services used for the development of the application.
6. Implementation and integration: it describes how the application has been implemented, the utilised libraries, and how integration has been managed.
7. Testing: it describes the testing strategies that have been adopted.
8. References: it contains links to resources useful to deepen the strategies and architectures quoted in this document.

1.3 Intended audience

The intended audience of this document is:

1. The Professor of the "Design and Implementation of Mobile Application", Luciano Baresi
2. The actual development team
3. Future possible developers that will work on the project
4. Future "Design and Implementation of Mobile Application" course students

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definition
Swift	Swift is a general-purpose, multi-paradigm, compiled programming language for iOS, iPadOS, macOS, tvOS, and watchOS.
iOS	iOS (formerly iPhone OS) is a mobile operating system created and developed by Apple Inc. exclusively for its hardware.
Framework	A framework is an abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software.
Mockup	In design, a mockup is a scale or full-size model of a design or device, used for demonstration, design evaluation, promotion, and other purposes

1.4.2 Acronyms and abbreviations

Keyword	Definition
DIMA	"Design and Implementation of Mobile Application" course
DD	Design Document
API	Application Programming Interface
UI	User Interface
UX	User Experience
APP	Application
MVVM	Model View View Model
E.g.	Exempli gratia, for instance

1.5 Project goal

The goal of this project is to design and implement a mobile application in Swift called "karma". It consists in a platform where it is possible to create an account to start a fundraising campaigns to receive help through donations or to help someone else by donating to his/her campaign. A non exhaustive list of the karma application features is the following:

1. Registration and login on karma service, using email and password
2. Create a new fundraising campaign, to receive donations from other users
3. Put a bookmark on a campaign to consult them in a second moment in the bookmark section
4. Donation to an existing fundraising campaign

These are some of the main functionalities provided by the karma application, a full list of requirements can be found in section 2.

2 Requirements

In this section is firstly provided a description of the actors involved in the requirements, e.g. who will interact with the application to be. Afterwards, functional requirements will be presented, which will be a detailed descriptions of what karma application will offer to its users.

2.1 Actors involved

In order to have a clear description of requirements, the different actors in the system should be clearly defined.

Actor	**Description**	**Parent**
Visitor	Everyone that launch the application without being authenticated	None
User	Everyone that is authenticated in the karma application	Visitor

2.2 Functional requirements

The most important functional requirements derive directly from the goals of the project. In particular, they describe the functions needed by the system to achieve the desired goal. In the following table, these requirements will be described, and it will be specified their level of priority, from low to high.

#	**Description**	**Priority**
R1	The system shall allow the visitor to register an account in the system, providing an email, a password and profile details	High
R2	The system shall allow the visitor to log in with their credentials	High
R3	The system shall allow the user to edit his/her account details	Medium
R4	The system shall allow the user to create a new fundraising campaign, providing a title, a description, an image and a monetary goal to achieve	High

R5	The system shall allow the user to modify an existing fundraising campaign details, such as title, description and image	Medium
R6	The system shall allow the user to delete (closing it) an existing campaign, having his data deleted from the system	Medium
R7	The system shall allow the user to examine all the existing fundraising campaigns ordered by ascending order of creation time	High
R8	The system shall allow the user to examine in detail a single campaign, where all the campaign's information, donations and some statistics are shown	High
R9	The system shall allow the user to add a bookmark to an existing campaign	Medium
R10	The system shall allow the user to delete a bookmark to an existing campaign	Medium
R11	The system shall allow the user to donate to an existing campaign through Apple Pay service	High
R12	The system shall allow the user to examine all the bookmarked campaigns	Medium
R13	The system shall allow the user to search, by name or username, other users registered to the application	Medium
R14	The system shall allow the user to search, by title, any existing campaign	Medium
R15	The system shall allow the user to examine the profile of another user, where his/her personal information, active campaigns and recent activities are shown	Medium
R16	The system shall allow the user to examine his/her own profile where his/her personal information, active campaigns and recent activities are shown	High
R17	The system shall set the amount of karma points to zero for a new registered user	Medium
R18	The system shall increment the amount of karma points of one for each Euro (€) donated of the user who made a donation	Medium

R19	The system shall decrement the amount of karma points of one for each Euro (€) received of the user who received a donation	Medium
R20	The system shall send a periodic notification to the user to remind him/her to keep track of the published campaigns progress	Medium

3 Architectural Design

3.1 High level architecture

The system is based on a typical client-server architecture. More precisely, the client is a mobile application built in Swift (the main focus of this document) and the back-end functionalities are fully provided by Firebase app development platform.

The logic of the application has been divided into different macro components, according to the MVVM architectural pattern.

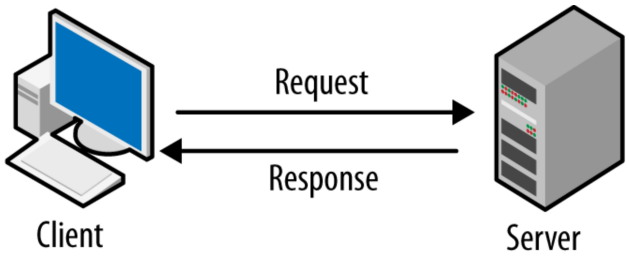

Figure 1: Client server paradigm

3.2 Architectural style and patterns

The karma application architecture can be classified as a RESTful architecture, it is based on the stateless principle, in which the server does not contain any information about the state of client, that is managed directly on client side.

This behavior guarantees less computational load on the server and also a dynamic attitude of the services. The application is then intended to be developed through client side programming, which means that all requests and update of the page are made on client side.

3.3 Model View ViewModel (MVVM)

The karma application has been implemented according to the Model–view–viewmodel (MVVM) pattern. It is an architectural pattern that facilitates the separation of the development of the graphical user interface from the development of the business logic or back-end logic (the model) such that the view is not dependent upon any specific model platform.

The view model of MVVM is a value converter, meaning that it is responsible for exposing the data objects from the model in such a way they can be easily managed and presented. In this respect, the view model is more model than view, and handles most (if not all) of the view's display logic.

More in detail, the component of the MVVM are:

- Model, it is a non-visual class that encapsulates the app's data. Therefore, the model can be thought of as representing the app's domain model, which usually includes a data model along with business and validation logic.
- View, as in the model–view–controller (MVC), it is the structure, layout, and appearance of what a user sees on the screen. It displays a representation of the model and receives the user's interaction with the view (e.g. screen tap gestures), and it forwards the handling of these to the view model via the data binding that is defined to link the view and view model.
- View model, it implements properties and commands to which the view can data bind to, and notifies the view of any state changes through change notification events. The properties and commands that the view model provides define the functionality to be offered by the UI, but the view determines how that functionality is to be displayed.

4 User Interface Design

The aim of this section is to show the design of the main screens of the user application, both of the iOS and iPadOS layout, describing the flow of the main functionalities for which it is intended.

4.1 UX flow diagram

The UX flow chart has been divided, for readability reasons, in two main parts: the login and registration phase and the karma app navigation phase.

4.1.1 Login and registration

When a visitor launch the karma application for the first time he/she has the possibility to perform the login (if he/she already has an account) or to create an account. Firstly the login page is shown. Then, the user can go to the registration page and back. The registration is successful only if it does not already exist a registered account with the same email and only if the password respects some security constraints (it must be made of at lest 6 characters).

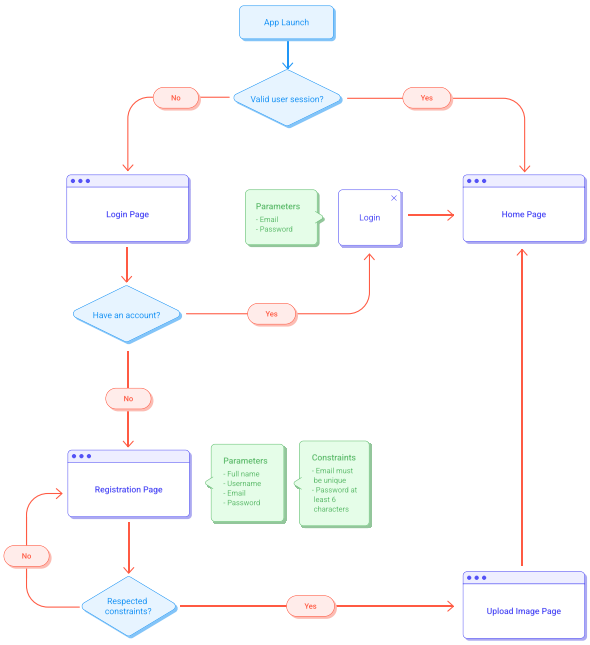

Figure 2: Login and registration UX flowchart

4.1.2 Karma application

Once a visitor has made the login or has successfully created an account, he/she is redirected to the home page. Thanks to a Tab Bar, the user has the possibility to navigate through the four main application pages: home, search, bookmarks and profile.

From the home and bookmark pages the user can examine the campaigns page, while through the search page the user can visit a campaign page or a user profile page. The user can analyse, in the profile page, his account details and his published campaigns. The user has also the ability to edit his account details or the information of a campaign launched by him/her, from the profile page or the campaign page, respectively.

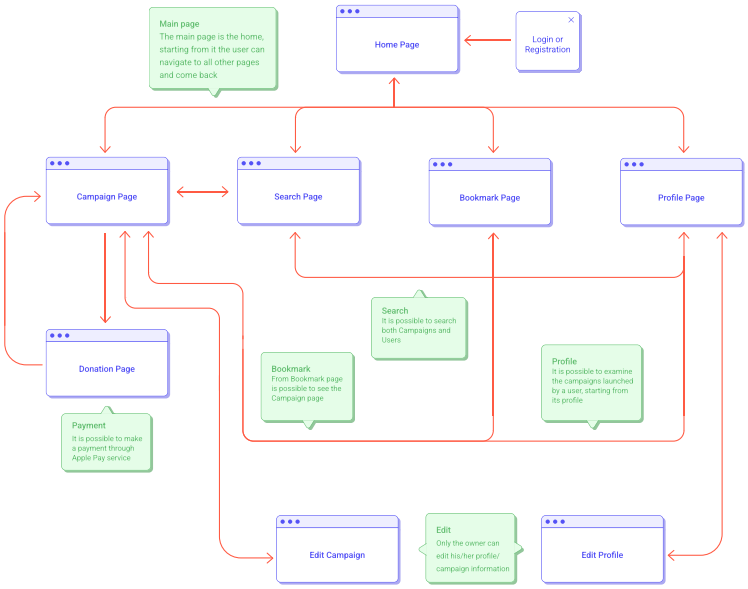

Figure 3: karma application UX flowchart

4.2 iPhone interface

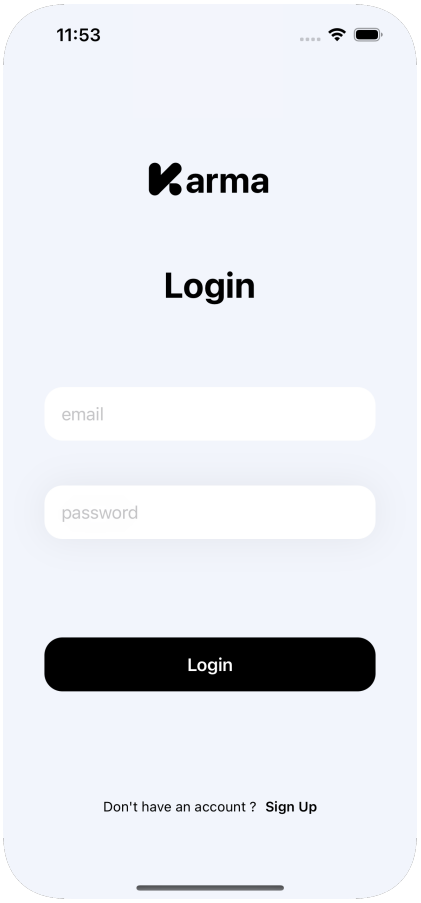

Figure 4: Login page

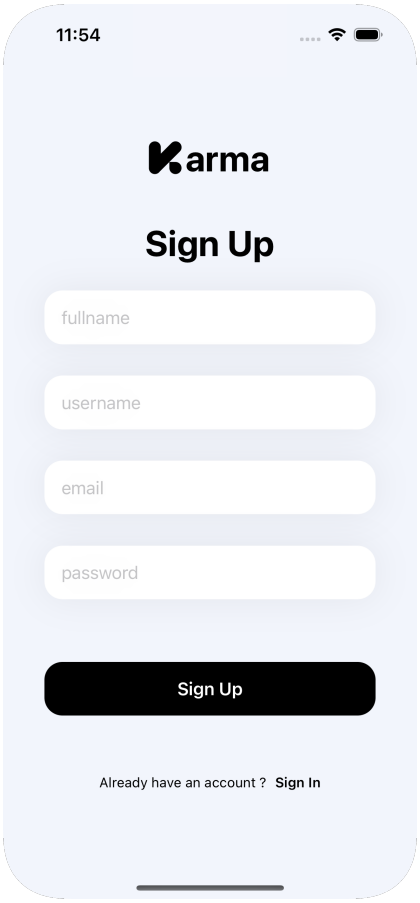

Figure 5: Registration page

The visitor can perform the login or, in case he/she does not have an account, can naviagte to the registration page and create an account.

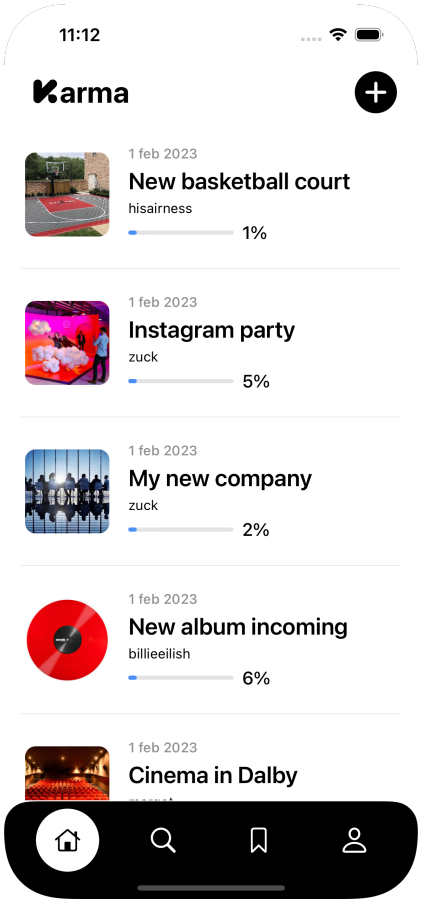

Figure 6: Home page

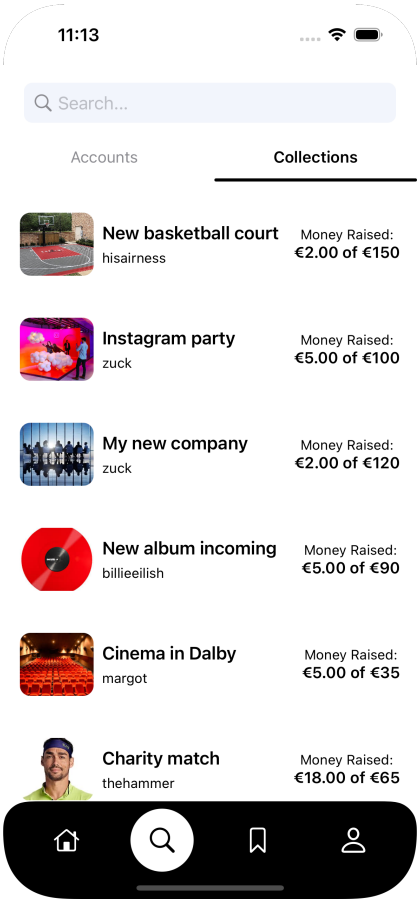

Figure 7: Search page

In the home page the user can see all the published campaigns, sorted by creation time. The user can search another user or a collection in the search page.

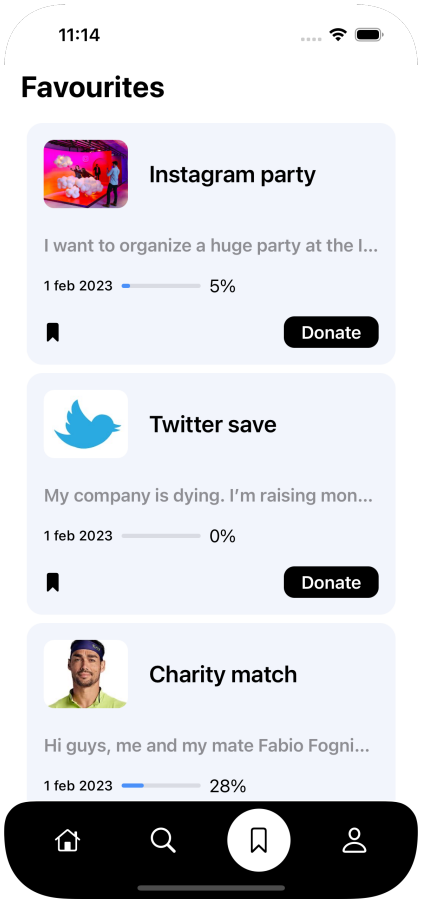

Figure 8: Bookmark page

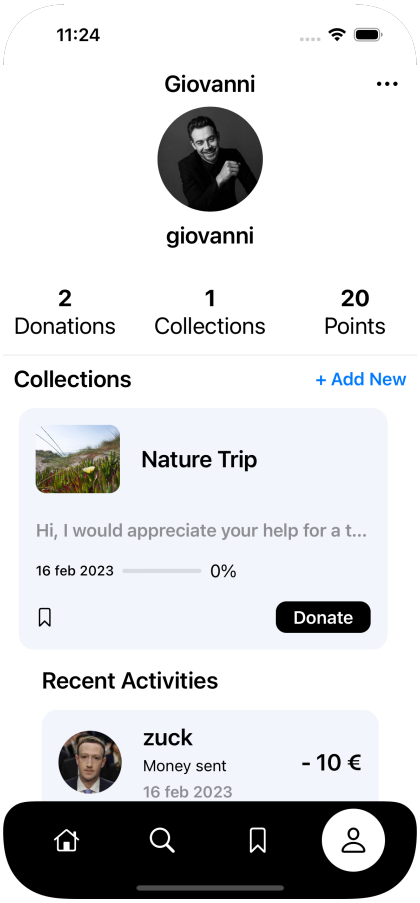

Figure 9: Profile page

The user can examine all his/her bookmarked campaigns in the bookmark page. He/She also has the possibility to examine all the account details, included published campaigns and recent activities.

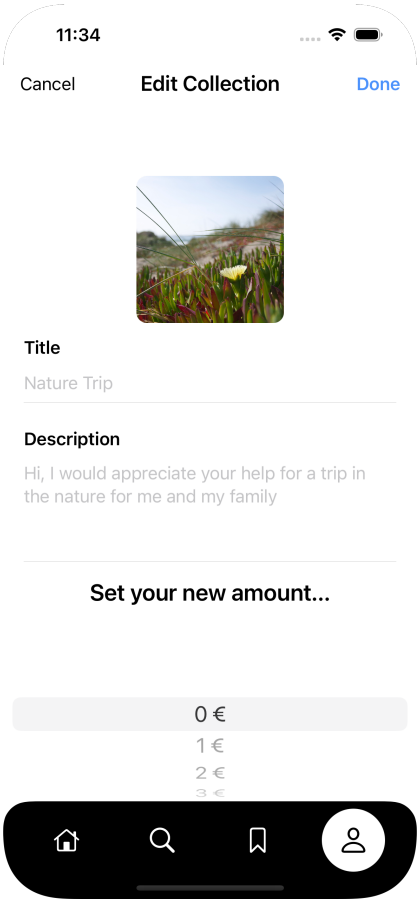

Figure 10: Edit campaign page

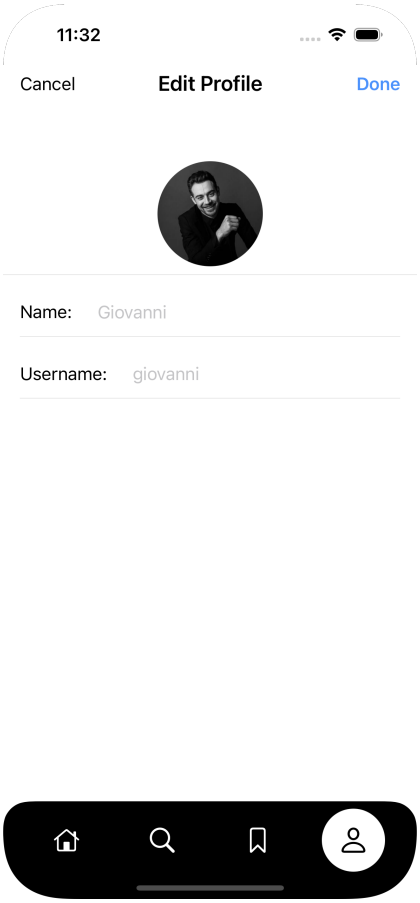

Figure 11: Edit profile page

The user can edit the details of both his profile or a published campaigns, starting from the profile page or from the collection page, respectively.

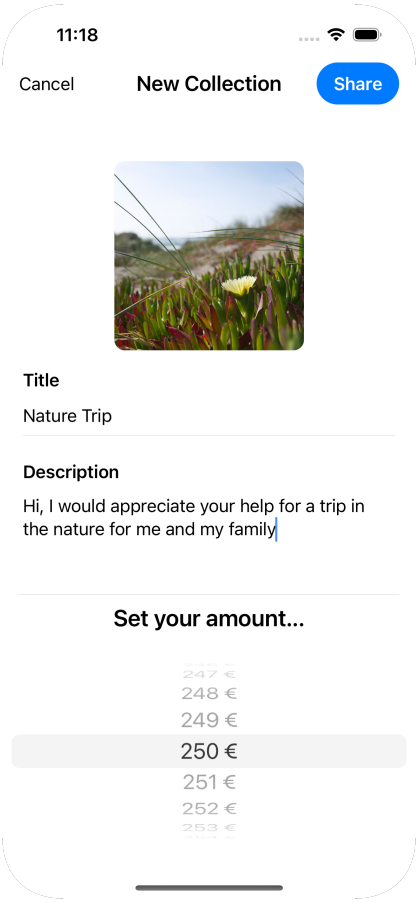

Figure 12: Upload campaign page

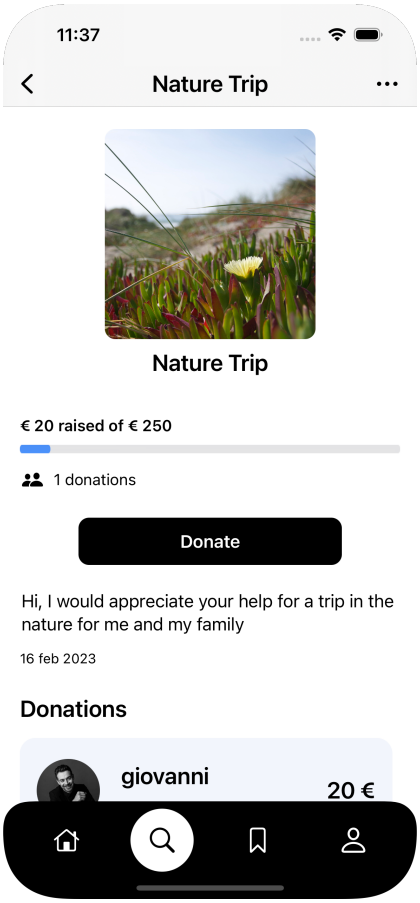

Figure 13: Campaign page

The user can upload a new campaign setting all the required information and he/she can also examine it in detail through the collection page.

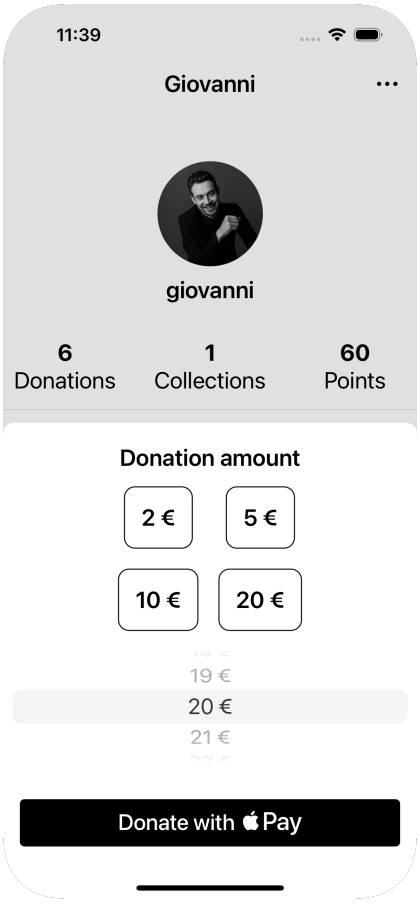

Figure 14: Payment page

The user can make a donation to a campaigns using Apple Pay service.

4.3 iPad interface

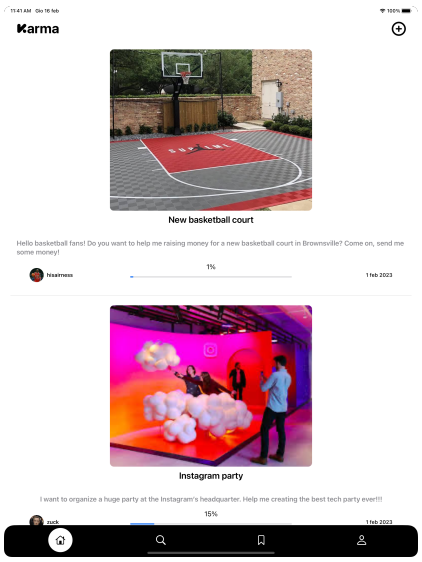

Figure 15: Home page

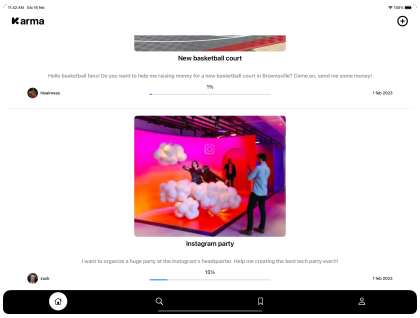

Figure 16: Landscape home page

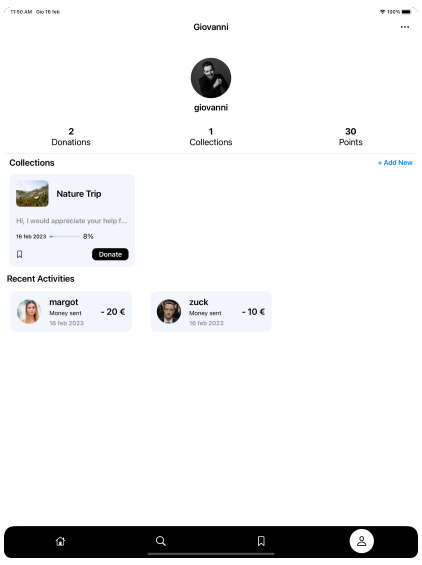

Figure 17: Profile page

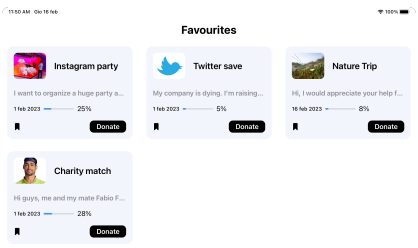

Figure 18: Bookmark page

5 Services

Different services have been used to implement the karma application, a brief description of each will be provided in this section, explaining how it has been used in karma.

5.1 Firebase

Firebase has been used as back-end service to build karma application. In particular, the services that have been utilised are: authentication, firestore and storage.

5.1.1 Authentication

The karma application, to authenticate users, makes a request to Firebase authentication provider, through the Firebase Package.

The authentication handler automatically handles the token expiration, requesting a new token when needed. In addition, the authentication provider performs checks for unique email and password length.

5.1.2 Firestore

Firestore has been used by karma application to save all the users, campaigns and payments data. In particular the collections used are: users, campaigns and payments. Whenever a new user register an account, a new campaign is launched or a donation is made, a new element is created inside the dedicated collection by making a request to Firestore.

5.1.3 Storage

The Firebase storage has been used to save all the images used in karma application. In particular, whenever a user or a campaign is created, the associated image is saved on the Storage to be shown in the views when needed.

5.2 Apple Push notifications

Push notifications have been implemented using both Firebase Cloud Messaging and Apple Push Notifications service.

In particular an APNs certificate is required and each device is identified by an APNs token. These two elements are required by Firebase Cloud Messaging which acts as the backend service and it will send to the device an associated FCM token. All this elements allow FCM to send notifications to the Apple Device.

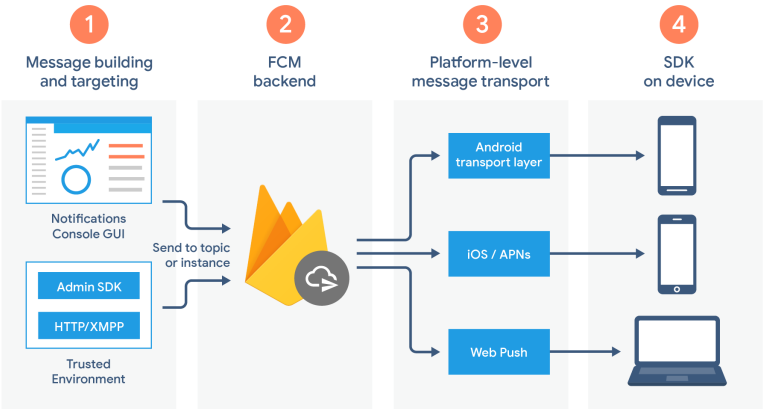

Figure 19: FCM notification architecture

5.2.1 Apple Pay

Apple Pay is a mobile payment technology that provides an easy and secure way for users to pay for real-world goods and services in an iOS and iPadOS.

In particular, Apple Pay service has been integrated into karma application to allow users to make donations to a campaign. To make it possible it is necessary that the user has a saved credit or debit card in the Apple Wallet. Each donation information such as amount, sender, receiver, etc., is saved in Firebase Firestore service.

At the moment of the donation, to initiate a payment, karma app creates a payment request. This request includes the total amount of the donation. Then it passes this request to a payment authorization view controller, which displays the request to the user and prompts for any needed information.

As soon as the user authorizes the payment, Apple Pay encrypts payment information to prevent an unauthorized third party from accessing it. On the device, Apple Pay sends the payment request to the Secure Element, which is a dedicated chip on the user's device. The Secure Element adds the payment data for the specified card and merchant, creating an encrypted payment token. It then passes this token to Apple's servers, where it is re-encrypted using the developers Payment Processing certificate. Finally, the servers pass the token back to karma app for processing.

The payment token is never accessed or stored on Apple's servers. The servers simply re-encrypt the token using your certificate. This process lets karma app securely encrypt the payment information without it having to distribute the Payment Processing certificate as part of the app.

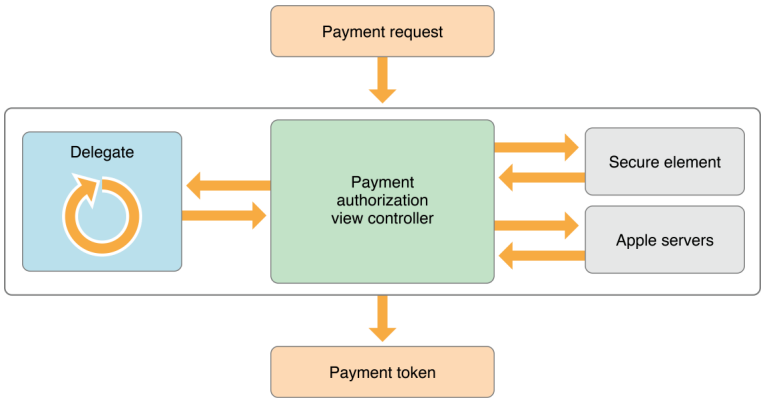

Figure 20: Apple Pay architecture

6 Implementation and integration

6.1 Implementation

Karma application has been developed using Swift and SwfitUI. For this reason, the supported operating systems are iOS and iPadOS. Swift is an open source multi-paradigm programming language developed by Apple. SwiftUI has been used to build all the application views, it uses a declarative syntax, so it is possible to simply state what the user interface should do.

Karma app will provide a slightly different user interface designed for iPads. In fact, the organization of contents in iPad UI must be adapted to a wide screen with a different resolution, which can be exploited to give to the user a better experience. This has been made possible thanks to the ability of Swift to detect the UIDevice characteristic, such as the device class and orientation.

6.2 Libraries

Several libraries have been used for the development of karma application, here a brief description of each will be provided.

6.2.1 Kingfisher

Kingfisher is a powerful, pure-Swift library for downloading and caching images from the web. It provides a chance to use a pure-Swift way to work with remote images. In particular, Kingfisher has been used in karma application to show campaigns and user profile images.

6.2.2 Firebase

As already mentioned, Firebase has been used as a back-end service for the karma application, thus all the required libraries have been used.

6.2.3 View Inspector

ViewInspector is a library for unit testing SwiftUI views. It allows for traversing a view hierarchy at runtime providing direct access to the underlying View structs. It has been used to verify that views showed the correct data to the user.

6.3 Integration process

The application has been developed using git with GitHub. GitHub flow has been used since it is great for small teams. The project has been organized using two type of different branches:

1. master: only the completed features are merged in this branch, it contains the actual state of the application.

2. features: a new branch is created whenever a new feature must be implemented or there is the need of a bug fix. This branch are merged back into the main branch when the work is finished and when it has been properly reviewed.

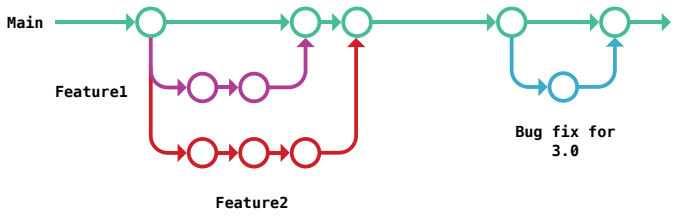

Figure 21: GitHub flow

The working strategy, apart from the branches, consisted in the opening of an issue for each feature or bug fixes. Each issue contained a brief description of the feature or bug, the requirements related to it and the related tests performed. This flow helped us in being always up-to-date within the developing team.

7 Testing

Four types of tests have been performed to verify the correctness of the karma applications

1. Unit testing: it ensures that the end-user (customers) can achieve the goals set in the business requirements, which determines whether the software is acceptable for delivery or not.

2. Integration testing: it ensures that an entire, integrated system meets a set of requirements. It is performed in an integrated hardware and software environment to ensure that the entire system functions properly.

3. UI testing: is ensures the correct behaviour of all the aspects of the application that the user will come into contact with. This usually means testing the visual elements to verify that they are functioning according to requirements.

4. Acceptance testing: it ensures that the mobile app is functional enough to meet the desired criteria set by the business and allows a group of testers to "use" the mobile application to check how it works.

7.1 Unit and Integration testing

Unit and integration testing have been performed using XCTest framework. In particular, to make it possible all the services, which their only purpose is the communication with the back-end, have been mocked. Mocked services have then been used instead of the real services for unit and integration testing, checking that the model correctly handled the data provided, through several assertions.

Different tests have been performed, such as the upload of a new campaign, the donation to a campaign, etc.. Also integration testing have been performed such as the creation of a campaign, then the edit of campaign details and the verification of the correctness of the update.

The tests performed allowed to reach a test coverage of more than 80%.

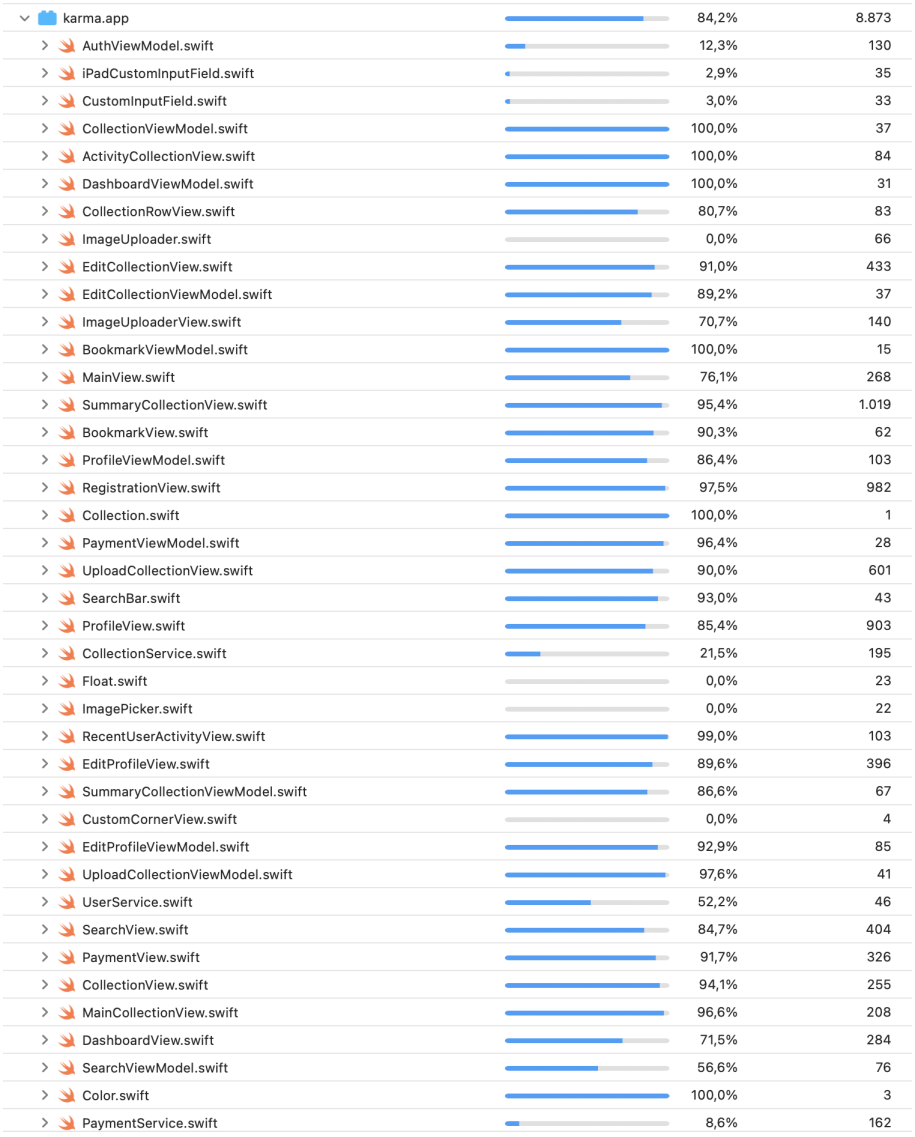

Figure 22: Testing coverage

7.2 UI testing

UI testing have been performed through the simulation of a user interaction with the application. Some of the aspects that have been tested are the registration of a new account, the login, the navigation through the main application pages and so on.

Mocked services have not been used to perform UI testing, because the Firebase

Local Emulators Suite has been used.

The Firebase Local Emulator Suite consists of individual service emulators built to accurately mimic the behavior of Firebase services. This means that it has been possible to connect karma app directly to these emulators to perform integration testing/UI testing without touching production data.

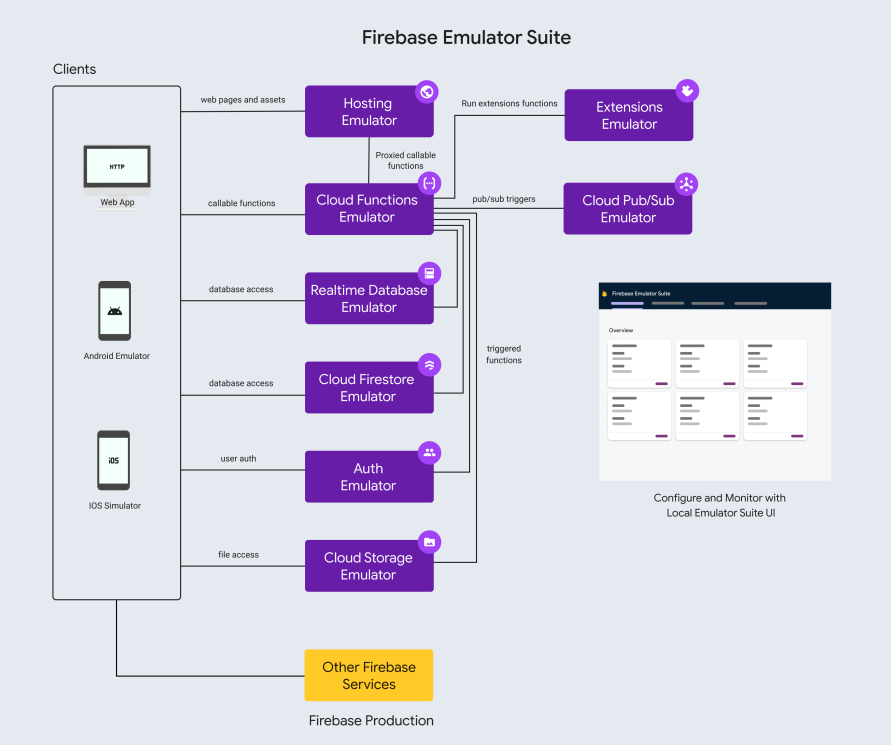

Figure 23: Firebase Local Emulators schema

7.3 Acceptance testing

This represented the last phase of the test plan, which consists in the distribution of the artifacts to a group of selected alpha testers.

The application has not been published in order to achieve this goal, but it has been delivered to developer team's colleagues. Testers performed a list of tests suggested by the developer teams, but not limited to, they have then provided feedback and reported some bugs.

8 References

References

[1] GitHub Docs, *GitHub Flow*

[2] Microsoft Documentation, *The Model-View-ViewModel Pattern*

[3] Apple Deverloper, *Swift Language*

[4] Firebase, *Firebase documentation*

[5] Firebase, *Firebase Local Emulators Suite*

[6] Apple Deverloper, *Apple Pay*

[7] Apple Deverloper, *Apple Notifications*

[8] GitHub, *Kingfisher library*

[9] GitHub, *View Inspector library*