AlergenEasier: iOS real time Application based on Geofencing and allergens filtering

Mireia Pérez Barros

Abstract—In the avancing world of technology, mobile applications are a rapidly growing segment of the global Mobile market to the extent that nowadays everyone has a phone on his pocket. This paper presents the development process to construct a native iOS application that makes use of Mobile thechnology such as geolocation and a real-time backend to support secure eating in individuals with severe food allergies. This project focuses not only on the application development itself but also on ensuring user privacy and security as well as the application quality by testing it.

Index Terms—Geofencing, native iOS application, Backend-as-a-Service, severe food allergies, allergens filtering

Resum— En el món avançant de la tecnologia les aplicacions mòbils són un segment del mercat global que creix ràpidament, fins a tal punt que avui tothom té un telèfon mòbil a la butxaca. Aquest article presenta el procés de desenvolupament per implementar una aplicació nativa per iOS, que fa ús de tecnologies com la Geolocalització i backend a temps real per donar suport a l'alimentació segura en persones amb al·lèrgies alimentàries greus. Aquest projecte es centra no només en el desenvolupament de l'aplicació, sinó també en garantir la privadesa i seguretat de l'usuari i les seves dades sensibles, així com la qualitat de l'aplicació gràcies al testing.

Paraules clau—Geofencing, aplicació iOS nativa, Backend-as-a-Service, al·lèrgies alimentàries, filtrat d'al·lèrgens

1 Introduction

'That's the whole myth about eating out: you're supposed to relax and someone else worries about your food' -Michael Mezzina, a Brooklyn-based video editor who also has a serious peanut allergy.

Imagine how diferent your life would be if you wake up one day without being able to eat some basic food like Eggs, Gluten, Fish or Milk. And now, imagine travelling to another country and being bound to dinner out everyday. Where would you go? Would you enter to every restaurant you see and ask for their dishes' allergens? Would you just eat some meat and pray there is no cross contamination? Or would you spend hours and hours searching through restaurant websites?

No, unfortunatelly I'm not exageratting. It's hard to overstate how stressful eating out with a severe allergy can be. But the stress isn't just from knowing that if one link in the chain of communication is broken, a trusted dish could actually kill you. Except for Gluten, currently the most controlled allergen, there are still places where you make sure to inform the waiter about your Egg allergy, and you end up with an Egg free Beef steak with a mustard sauce on it, what surprisingly contains Egg.

Not only when travelling, food is bound up in our everyday social lives, family gatherings, even our jobs. It is a fact that we have to be aware, and it is impossible to change that unless you want to end up in a hospital, but what about making our life not only more secure but also easier? That was the main reason for AllergenEasier to be created, a native mobile application made from allergic people to allergic people.

The development of this application stems from a gap in available tools that facilitate secure eating in individuals with severe food allergies. Its design provides to the user information about all the available restaurants adapted to his necessities with only one tap.

This article is structured as follows. Next chapter presents the existing solutions in the market with the same goal than AllergenEasier. Chapter III describes the project objectives. Chapter IV presents the methodology applied during the project development. Chapter V shows the project planning. Based on the results of chapter VI, several conclusions regarding the app development are drawn and presented in the VII conclusion chapter. Finally, future improvements will be explained in chapter VIII.

2 Related work

Before starting the project, a study of the Applications market was made in order to be aware of what kind of help was currently available for allergic users. Even though the proposed app is uniquely focused on iOS users, the study contemplated all available application distribution platforms in the market.

- *Contact e-mail: Mireia.PerezBa@e-campus.uab.cat*
- *Branch: Tecnologies de la Informació i Seguretat*
- *Tutor: Ramon Grau Sala (DACSO)*
- *Course 2018/19*

While there exist several food based applications, they are merely focused on restaurant services that can include diet adapted menus than taking into consideration allergic people necessities.

A website research was also made and this time results where slightly different. Most similar to our approach, restaurantallergens.com and allergyeats.com offer to the user the possibility to see dishes allergens.

Although the objective of the first one is to show the allergens chart in a website format only for a few specific restaurants that are willing to pay for the service, the second one presented big similarities with AllergenEasier. Both software solutions share the same goal: geolocation and allergens filtering.

From a functionality point of view, each of them allows the user to view restaurants from a specific geographical area, to give feedback about his experience and to stand out a restaurant by adding it to favorites or rating it.

However, there are some details that diverge between them. Website solution is only available for United States restaurants and they have to pay if they want to be published. Nevertheless, AllergenEasier has a free specific profile for Restaurants that provide them the possibility to register their dishes and to offer the user a detailed information about their allergens.

The lack of mobile Applications related to food allergies and the inexistance of software solutions not only for allergic Spanish users but also for the rest of the world confirmed the real necessity of AllergenEasier.

3 OBJECTIVES

After the previous analysis and understanding of the problem, the main objective would be, without any doubt, to offer the users with severe food allergies the possibility to filter near restaurants by its dishes allergens.

Therefore, this objective can be expanded into the following:

- Adquire knowledge about Xcode and Swift programming language in order to develop a native[1] iOS application.
- Design an user-friendly application based on a paper prototype study.
- Distinguish between twoo possible user profiles: Consumer and Restaurant. Adapt user experience according to it.
- Develop a real time application that provides data synchronizing between multiple devices.
- Assure user privacy by encrypting sensible data sent to the server as user allergies and passwords and provide a secure data exchange between the application and backend.
- Guarantee application quality by doing not only manual regression testing but also automated UI testing.
- Study Apple Distribution platform and publish the final version of the application on Apple Store.

4 METHODOLOGY

The solution proposed for reaching the goal of this project followed waterfall methodology with Royce's iterative feedback. It is an iterative model that offers the opportunity of going back to previous steps if there is the necessity.

Waterfall methodology separates the project in five different phases: analysis, design, development, testing and delivery[1].

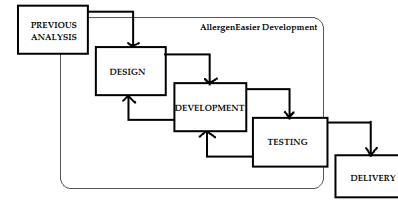

Fig. 1: Steps of waterfall methodolody with Royce's iterative feedback.

Thesese phases are going to be described and its results are going to be presented as follows.

4.1 ANALYSIS

The purpose of the analysis phase is to thoroughly examinate and evaluate the viable options for the project development and to achieve an optimum high level solution that will satisfy the client's requeriments and the project's constraints.

The aim of this project is to develop an application focused on people with severe allergies, therefore their necessities should be analyzed and translated as project requeriments. As I belong to that group of people due to my food allergies and I could be a possible user of the application, it is not necessary to collect requeriments from external resources in order to focus time and efforts to the application development phase.

During this phase a research about all necessary developing tools for the project was also made as well as their configuration and environment preparation. As I have never developed mobile Applications before, this research included getting familiar with Xcode and one of the native programming languages for iOS: Swift.

Once my knowledge in Swift increased, a second study was made in order to collect information about all the available tools for providing AllergenEasier its main functionalities: Geolocation and real time synchronization. This study also contemplated some of the available dabatases for iOS applications: CoreData and Realm, and compared them in order to decide which solution best suits the project.

The results of the second study and the final tools used for developing the project are going to be exposed as follows:

4.1.1 MapKit Framework for Geolocation

MapKit is an Apple framework for iOS which displays map or satellite imagery directly from the app's interface, call out points of interest, and determine

[1] A native implementation means that you are writting the application using the programming Language and programmatic interface exposed by the Mobile operating System of a specific type of Device. For exemple, AllergenEasier was written using the Swift language and the iOS operating system Application Programming Interfaces (APIs) that Apple supplies and suports.

placemark information for map coordinates. MapKit uses the Google Mobile Apps (GMM) service to provide map data.

4.1.2 Realm as the local database

One factor that determines the performance of mobile apps is its responsiveness to search queries. The faster it responses, better is the performance. Successful search results depend on the efficiency of database that is used for storing data of the app.

In the appendix section [A1] at the end of this paper CoreData and Realm databases are going to be analyzed in order to expose the reasons why Realm has been choosen as the best option for the project.

4.1.3 Firebase real time database as a backend solution for multi device Synchonization

Firebase is a Backend-as-a-Service[2] (BaaS) and a Google product that allows the developer to build web and mobile applications without a server side programming language. It offers so many useful features as a Real time database and Authentication, which were the both used in this project.

Realtime Database is a cloud-hosted NoSQL database with SDK support for iOS, Android, and the web. This tool stores data in JSON documents, so everything is either a key or a value [Fig. 2]. Realtime Database has no concepts of data types.

Fig. 2: Extract from AllergenEasier Firebase Real Time Database. Users data is stored as a JSON, with its corresponding keys and values.

Data synchronization uses web sockets, allowing for very snappy transactions. Realtime Database also handles updates when a device is offline or syncing changes when the network reconnects.

It allows the developer to set up a Realtime Database backend and not have to worry about things like deployments, hardware, uptime, and scalability. Thanks to Firebase service, AllergenEasier had a pretty robust backend server running in just a few minutes letting me focus on the fun and unique parts of the app. This was a great advantatge due to the fact that developing a specific backend for AllergenEasier would have taken a lot of time. Another great advantadge that Firebase brings is security, but we will focus on this point during Development step.

Realtime Database has some cons too. It requires to write most of the system's application code on the client. Besides, it is not a Free service so if AllergenEasier popularity increases, the application cost would increase too.

As long as synchronization is concearned, in Firebase you don't just send a request and get back a single response. You're attaching an asynchronous listener to an endpoint in your database. This listener is observing that end point and reporting back anytime the data within the endpoint changes. If it sees any changes in the data it's observing it will call on the closure, which is a function that you've previously defined [Fig. 3].

```
90          ref.observe(.value, with: { snapshot in
91
92              self.tableview.reloadData()
93              self.restaurantsNumber4U.text = String(self.restaurants.count
                    + " RESTAURANTS FOR YOU"
94          })
95
```

Fig. 3: Example of a listener in AllergenEasier code. With these lines od code I'm setting up an observer in firebase, so if I add restaurants to the data base in another view, it is not necessary to call this function again. Firebase observer will see that the data has changed, then call on the closure I defined in the .observe function.

Another Firebase main functionality is cloud storage, used to store and serve user-generated content, such as photos or videos. It is used together with a UIImagePickerController in order to let the user capture an image for the restaurant being registered or select it from his photo album and store it in the cloud for its later synchronization, as it happens with data. Furthermore, uploads and downloads are automatically retried in the case of poor network connections, so it is not necessary to keep track of them. Images are compressed before being uploaded.

4.2 DESIGN

This step objective is to reflect the solution idea, focusing on the best alternative to all the different technical, functional, social and economical aspects.

Three main activities took place during this phase: the paper prototype and navigation map design, define software and data model structure in order to flexibilize, modulate and offer the possibility to reuse the application in a future and time frames planning for each activity of the project, that are going to be explained during the planning section.

4.2.1 Paper prototyping

Paper prototyping is a low fidelity research methodology that helps to visualize and test ideas and concepts in an early stage of a project. Prior to prototyping, you should have a goal in mind and you should have gained knowledge through other research methodologies, what was already done during the analysis step.

[2] Backend-as-a-Service is a model for providing web app and mobile app developers with a way to link their applications to backend cloud storage and APIs exposed by back end applications. These services are provided via the use of custom software development kits (SDKs) and application programing interfaces (APIs).

Fig. 4: Paper prototype of the application main screens: For You, Favorites, Near by and Log in.

4.2.2 Navigation Map

A navigation map shows all the application screens and how the user can navigate through them. It is an easy visual way to see how the application screens relate to one another. Navigation maps are created at design time to help the developer to obtain a general picture of the application magnitude.

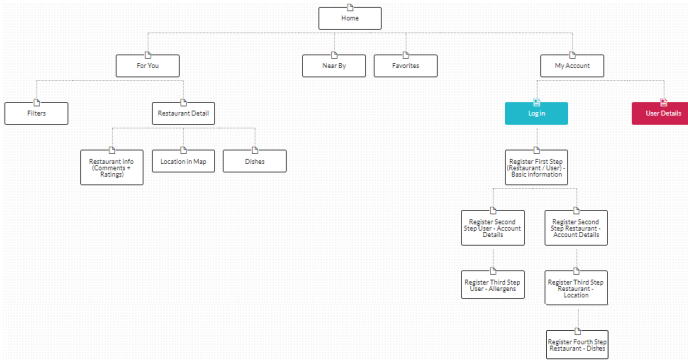

Fig. 5: AllergenEasier navigation map.

4.2.3 Data modeling

The data model refers to the logical interrelationships and data flow between different data elements involved in the information world. It also documents the way data is stored and retrieved in the application and dictates the relationships between database tables, foreign keys and the events involved.

AllergenEasier has two types of databases: the local one using Realm and the cloud real time database using Firebase.

Realm database follows the structure shown in [Fig. 6].

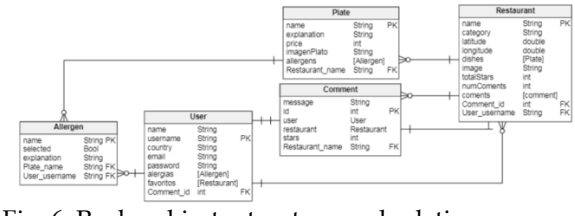

Fig. 6: Realm objects structure and relations.

Firebase real time database, as has been already explained, stores data as JSON objects. It can be imagined as a cloud-hosted JSON tree. Unlike a SQL database, there are no tables or records. When adding data to the JSON tree, it becomes a node in the existing JSON structure with an associated key.

One of the most important practices with Firebase database is to avoid nested data. When you fetch data at a location in your database, you also retrieve all of its child nodes. In addition, when you grant someone read or write access at a node in your database, you also grant them access to all data under that node.

Therefore, in practice, it is best to keep your data structure as flat as possible. If data is instead split into separate paths, also called denormalization, it can be efficiently downloaded in separate calls, as it is needed. For example, in [Fig. 2], favoritos are shown as [*Restaurant primary key]: true* instead of [Restaurant name]: {"attribute": "value", "attribute2": "value"}. Now it is possible to iterate through the list of Favorites by downloading only a few bytes per restaurant.

When data in Firebase is modified, the observers in code call and the callback function. Here is when Realm database is updated with the new values. The main reason for having these two databases connected is to optimize the quantity of data being downloaded from Firebase. If there are no changes in the data stored, application viewControllers can obtain data from Realm every ime they need it without having to download it from irebase. Downloading data from firebase wouldn't be a roblem if it was a free service, but free plan is limited to nly 10GB.

Fig. 7: Structure of the connection between the device storage and Firebase.

4.3 Development

In this step the desing was translated into code. In order to do that, the programming Language Swift was used together with Xcode.

As I had never worked with Swift before, it was difficult for me getting used to it. In order to get familiar with it, a first approach of the application was developed to practice the language before starting to develop the real application. This first contact with the programming Language helped me to adquire experience and to improve code quality in the future.

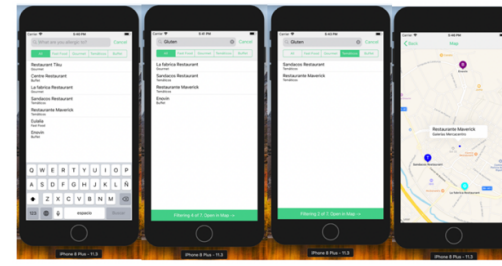

Fig. 8: First approach of an application with similar functionalities than AllergenEasier. This application was developed with the objective to learn Swift. It had no backend, but had a view with a list of restaurants and a filtering button. It also had a map view to practice with mapKit.

Once learned Swift, AllergenEasier development started. In this step all the knowledge obtained from analysis phase was applied in order to implement the application following the explained data model and designs.

4.3.1 Security and privacy

During AllergenEasier development, I realized that user sensible information as passwords and allergens were saved to Firebase as a clear text, so the admin could see all these private data and use it.

Firebase security. With Firebase, the data is protected over the wire using HTTPS. Then, when it lands on the Firebase REST frontend server, HTTPS terminates, and the server has access to the full payload. Then the REST server routes the data to the backend/database, which also has access to the data. When the data is written into disk, it's encrypted at-rest, but the at-rest encryption keys are also available to Google and your administrators will also see the Firestore contents.

In order to improve user privacy and application security, some changes were made to the Application and are going to be explained as follows.

AES encryption using CriptoSwift. CryptoSwift is a growing collection of standard and secure cryptographic algorithms implemented in Swift. Using AES algorithm to encrypt data client side (End-to-End Encryption) prohibits all these participants/roles seeing Firebase data. For example, in [Fig. 2] allergen values have been encrypted.

Generally speaking, there are two kinds of encryption algorithms –– symmetric-key algorithm and asymmetric-key algorithm.

For symmetric-key algorithm like AES, the same cryptographic key is used for both encryptation and decyption. In comparision to asymmetric-key algorithm like RSA, it is usually high speed and low RAM requirements, but because it's the same key for both encryption and decryption. Nevertheless, it's a big problem of key transport from encryption side (sender) to decryption side (receiver).

For asymmetric-key algorithm, it requires two separate keys, one of which is private and one of which is public. Although different, the two parts of this key pair are mathematically linked. The public key is used to encrypt plaintext or to verify a digital signature; whereas the private key is used to decrypt ciphertext or to create a digital signature. Comparing to symmetric-key algorithm, asymmetric-key algorithm does not have the problem of key transport, but it is computationally costly compared with symmetric key algorithm.

In AllergenEasier case, the sender would be the application and the receiver Firebase. Firebase objective is to store the encrypted data sent by the application and send it back to it when is necessary, so it doesn't have to decrypt data. Due this reason, there is no point in using an asymmetric key algorithm.

That was the main reason for using AES symmetric algorithm. Nevertheless, one of AllergenEasier main functionalities is synchronization. Devices must know every key in order to be able to decrypt data sent by Firebase. This supposes the biggest trouble for AllergenEasier data encryption system, because if we had a dedicated server as the backend we could send clear data and pass the encryption/decryption responsibility to it, but Firebase requires to write most of the system's application code on the client side.

One proposed solution was to store encryption keys in Keychain Access because it works in collaboration with iCloud keychain, which lets you share keychains with your other devices. This way, encryption keys could be stored in this shared keychain and every device could decrypt the data sent by Firebase. However, it is necessary to buy an Apple Distribution profile in order to use iCloud Keychain capability, so this solution was discarded.

Finally, the choosen solution for encrypt/decrypt data was to use the same key for every encryption instead of making it random and using a random iv sent to Firebase together with the encrypted data. We know that this isn't the better solution, but it can be solved using a specific dedicated backend for the application as a future improvement.

Firebase Authentication service. Firebase Authentication aims to make building secure authentication systems easy, while improving the sign-in and onboarding experience for end users. It provides an end-to-end identity solution, supporting email and password accounts, phone auth, and Google, Twitter, Facebook, and GitHub login, and more.

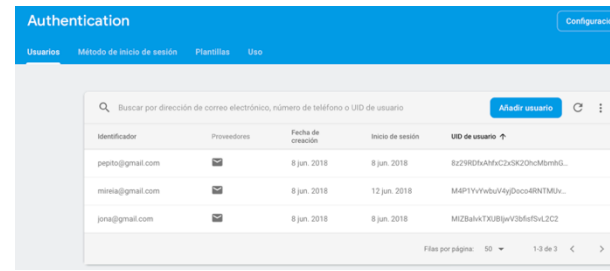

Fig. 9: Firebase authentication service. Example with some users registered using the email.

Firebase authentication service also sends an event to the application every time there is a change in the authentication State. This functionality is so useful in order to check if the user is logged or not.

```
Auth.auth().addStateDidChangeListener() { auth, user in

    if user == nil {
        self.performSegue(withIdentifier: "userHasLogOut", sender: nil)
    }
}
}
```

Fig. 10: Using Firebase authentication service in code. When there is a change in the authentication State, this function will get called and the segue will be performed in case the user has logout.

4.3.2 Using a version Control System: Git

As everyone knows, Git is a version control System. It was used during AllergenEasier development in order to manage an revert changes in the project.

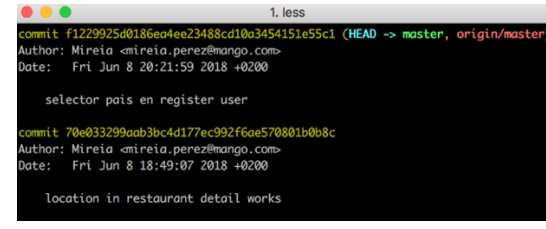

Fig. 11: Some AllergenEasier commits.

Whenever an important change or modification was commited to Git, it was made a Regression testing. This consist of testing existing software applications to make sure that a change or addition hasn't broken any existing functionality. Its purpose is to catch bugs that may have been accidentally introduced into a new build or release candidate, and to ensure that previously eradicated bugs continue to stay dead.

4.3.3 Geofencing and local notifications

The key to being successful in mobile market is about being relevant. People take their mobile devices everywhere –work, holidays, family dinners, Friday night drinks– mobile applications should take advantadge of that fact in order to adapt the user experience to their real necessities.

This was the main reason for Geofencing or region monitoring to be implemented during the project. It allows the application to be notified by the system every time a device crosses a defined region boundary. Using it, AllergenEasier can understand the users' behaviors in the real world and trigger a notification whenever restaurants adapted to their allergies are nearby.

It was implemented using CoreLocation framework for restaurants geolocation together with iOS local notifications. User should be logged in order to obtain its allergies list and compare them with the restaurant allergens.

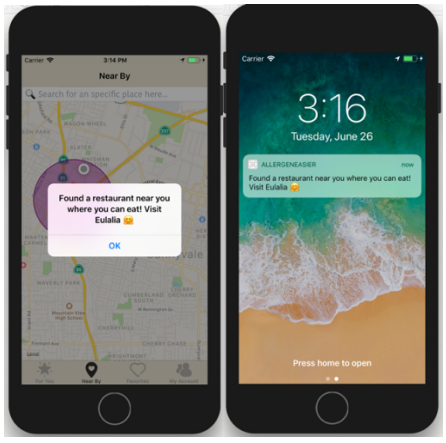

Fig. 12: Notifiying the user upon arriving to a geographic region. This region is related to a restaurant so the user will be notified about the existance of a restaurant adapted to his necessities.

4.4 Testing

Testing phase tests usability -including model performance- and manages generated bugs. It focuses efforts on ensuring the model does what the user expects it to do, as well as predicting how the model will perform in a robust environment. It's important to ensure the application will not result into any failures because it can be very expensive in the future or in the later stages of the development. Considerable tools, time, and resources go into testing.

Once application development was finished, its behaviour was verified in different scenarios and conditions through not only manual but also automated testing.

4.4.1 Manual testing

Ideally, we should get someone else to check our work because another person is more likely to spot the flaws. That was the reason why this step was separated in two types of manual testing: Smoke and beta testing.

Smoke testing. Smoke testing is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The results of this testing are used to decide if a build is stable enough to proceed with further testing.

Once finished the application development, allergenEasier main functionalities were tested to assure they work as expected. As every time an important change in code was commited it was made a regression test, code at this point should be mature enough to check only its main functionalities. This first testing was made in order to assure application quality before showing to some users the final product during exploratory testing.

This testing was made by the developer, and the main functionalities tested during this phase were:

- For You: Restaurant filtering. Real time synchro between devices when adding a new restaurant.
- Restaurant detail: Restaurants rating and commenting functionalities. Dishes and location view also checked.
- Favorites: Add/Remove restaurants from favorites not only as a logged but also as a no logged user. Favorites synchro between devices and accounts.
- Near by: Restaurants shown by user current location. Searcher. Location updates.
- My account: Account synchro between devices. Log in and log out functionalities. Register functionality for restaurants and users.

Beta testing. A beta test is a step in the cycle of a software product release. Previously, the application has always been tested using the built-in simulator and on the developer's own device. Interestingly, he may not be able to uncover some of the bugs, even though he is the app creator.

TestFlight makes it easy to invite users to test an application and collect valuable feedback before it is released on the App Store. To take advantage of TestFlight, a beta build of AllergenEasier was uploaded, and iTunes Connect was used to add the names and email addresses of people invited to test it. Testers will install the TestFlight app for iOS, watchOS, and tvOS so they can use the beta app and quickly provide feedback.

Beta testing is generally opened to a select number of users. They may be some potential app users, colleagues, friends or even family members. The whole point of beta testing is to let a small group of real people get their hands on the application, test it and provide feedback.

In AllergenEasier case, 12 people tested the app through testflight. I would like to point out some of the testers feedback, that can be used in a future iteration for AllergenEasier improvement.

- Enable swipe to delete comments. Some users once published their comment in the restaurant tried to delete through left swipe.
- Country selector in user register. Some of the testers agreed selectors helps improving user experience and increase security.

- Hide keyboard when doing scroll on a view. That kind of testing helped me not only on taking notes of possible future improvements but also identifiying current bugs. Some users noticed that they weren't able to finish the register because the keyboard blocked the screen. This issue is already solved.
- On dishes view, most of the users tapped on the images. One possible improvement is to develop a view in order to show dish details as well as some other images.
- Implementing other filtering options in For You view. Testers agreed on allergens filtering was very useful its combination with location or price filtering would improve its value.

4.4.2 Automated testing

A big advantage of manual testing is the ability to see real user issues. Unlike a robot, when developing software, going through manually allows you to see bugs that a real user could face. Automated testing is best to use when you're working on a large project, and there are many system users. The biggest advantages of automated testing are its relative quickness and effectiveness. AllergenEasier is not a large and complex project, but in a future its code and functionalities will increase so it will save huge effort in regression testing of each release.

Automated testing using Appium

Out of the many options available in the market, it was decided to hold hands with Appium tool. Appium aims to automate any mobile app from any language and any test framework, with full access to back-end APIs and DBs from test code. It is a mobile test automation framework that works for all: native, hybrid and mobile-web apps for iOS and Android.

Appium is a great choice for test automation framework as it can be used for all these different app/web types. Basically, Appium derives its roots from Selenium[3] and it uses JSONWireProtocol internally to interact with iOS and Android apps using Selenium's WebDriver.

In its architecture, Appium is an HTTP server written in Node.js that creates and handles multiple WebDriver sessions. Appium starts tests on the device and listens for commands from the main Appium server. It is basically the same as the Selenium server that gets HTTP requests from Selenium client libraries.

This is how Appium works with iOS:

1. Appium client (Java or Python) connects with Appium Server and communicate via JSON Wire Protocol
2. Appium Server then creates an automation session for the client and also checks the desired capabilities of client and connects with respective vendor-provided framework UIAutomation
3. UIAutomation will then communicate with bootstrap.js which is running in Simulator/Real device for performing client operations
4. Bootstrap.js will perform action on our AUT (Application Under Test)

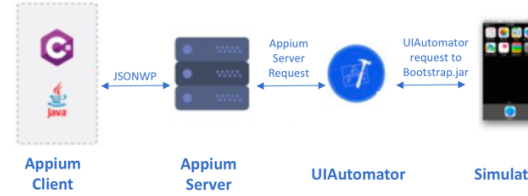

Fig. 13: Schema to show how Appium works with iOS.

Cucumber integration in Appium

Cucumber was integrated with Appium in order to automatize AllergenEasier application under Behaviour Driven Development[4] (BDD). It is a software tool used for testing software that runs automated acceptance tests written in a behavior-driven development (BDD) style. Central to the Cucumber BDD approach is its plain language parser called Gherkin[5]. It allows expected software behaviors to be specified in a logical language that customers can understand.

Cucumber tests are divided into individual Features. These Features are subdivided into Scenarios, which are sequences of Steps. A feature is a Use Case that describes a specific function of the software being tested.

Each Feature is made of a collection of scenarios. A single scenario is a flow of events through the Feature being described and maps 1:1 with an executable test case for the system. The crux of a Scenario is defined by a sequence of Steps outlining the preconditions and flow of events that will take place.

```
Feature: Check favorites synchro feature
    As a ALLERGENEASIER APP user
    I want to be able to synchronize my restaurants in favorites


Scenario: Add a restaurant to favoritos and when the user log in
this restaurant should remain in favoritos
    Given I am on the For You view
    And I click on "Favorites" option
    And I click on My Account in Tabbar
    And I do Login action
    When I click on Favorites in Tabbar
    Then I should see Enovin restaurant in favoritos

  Scenario: Remove a restaurant from favoritos and when the user
  logout and login again this restaurant shouldn't be in favoritos
    Given I am on the For You view
    And I click on My Account in Tabbar
    And I do Login action
    And I click on For You in Tabbar
    And I click on "Favorites" option
    And I click on My Account in Tabbar
    And I do Logout action
    And I do Login action
    When I click on Favorites in Tabbar
    Then I should see Enovin restaurant is not in favoritos
```

Fig. 14: Example feature definition with its scenarios and steps.

[3] Selenium is a portable software-testing framework for web applications. Selenium provides a playback tool for authoring tests without the need to learn a test scripting language (Selenium IDE).

[4] behavior-driven development (BDD) is a software development process that emerged from test-driven development (TDD). It combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software development and management teams with shared tools and a shared process to collaborate on software development

[5] The Gherkin language defines the structure and a basic syntax for the test description that can be understood both by the technical team members as well as by Analysts/PO. This way, while generating tests, live documentation is being generated and it perfectly describes how the system behaves by enriching and maintaining the documentation. The format was introduced by the Cucumber tool.

Test development following Page Object pattern

Selenium automated tests were written in Java following Page Object design pattern. Instead of having each test fetch elements directly and being fragile towards UI changes, the Page Object Pattern introduces a decoupling layer as can be shown in [Fig. 15].

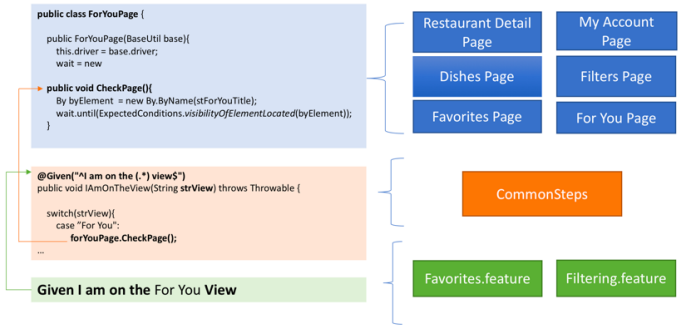

Fig. 15: Example of PageObject pattern structure explained from one of AllergenEasier steps

Test results

Twoo features were developed in total. Each feature had its own scenarios and was focused on one of AllergenEasier main functionalities. The features choosen to be automated were Favorites and Filtering ones.

Due to the fact that AllergenEasier is not a complex application, it has some functionalities that can't be tested automatedly unless the developers have a dedicated database for testing. For instance, Register and Rating functionalities shouldn't be automated without an appropriated database because of the amount of irrelevant data that these tests would generate.

Nevertheless, testing results provided additional value to the project and raised its security. They are going to be shown in the table below.

Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Results**	**Actual Results**	**Pass Fail**
TUI01	Adding to favoritos and synchro after Login	1. In For You view add the first Restaurant to favoritos 2. Do login with the email and password 3. Go to Favoritos to check if the restaurant was added	Email: mireia@gmail.com Password: mango123	Restaurant should be added to favoritos	As Expected	Pass
TUI02	Removing from favoritos as a logged user and synchro after logout	1. Do Login with the email and password 2. Remove the first restaurant from favoritos 3. Do Logout 4. Do Login again 5. Check if the restaurant is not in favoritos	Email: mireia@gmail.com Password: mango123	Restaurant should be removed from favoritos	As expected	Pass
TUI03	Apply a filter and check if the restaurant filtered doesn't contain the allergen: Gluten	1. Filter by Gluten 2. Tap on the first restaurant to see its information 3. Click on Dishes section 4. Check if its dishes don't contain Gluten	Gluten	Restaurant dishes should be Gluten free	As expected	Pass
TUI04	Apply a filter and check if the restaurant filtered doesn't contain the allergen: Egg	1. Filter by Egg 2. Tap on the first restaurant to see its information 3. Click on Dishes section 4. Check if its dishes don't contain Egg	Egg	Restaurant dishes should be Egg free	As expected	Pass

As can be seen in the picture, AllergenEasier passed all the Automated tests.

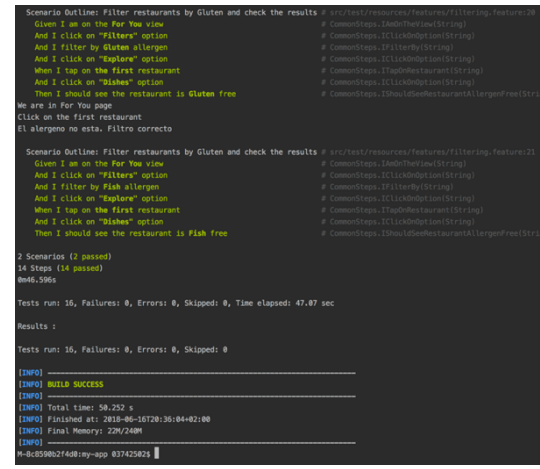

Fig 16. Appium test results for the Filtering feature

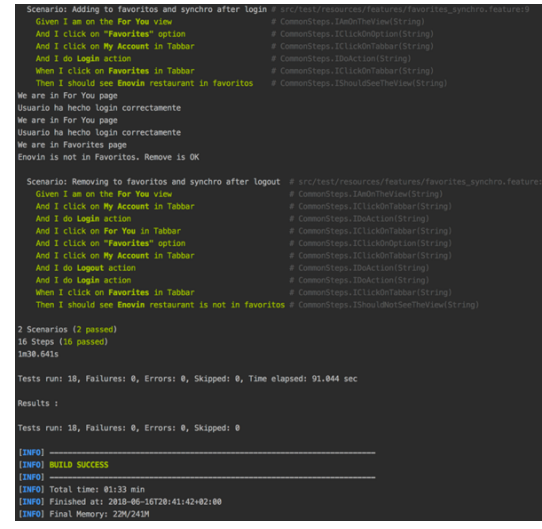

Fig 17. Appium test results for the Favorites feature

4.5 DELIVERY

In this step was made a study of Apple distribution platform. Apple Store allows users to browse and download apps developed with Apple's iOS software development kit. Nevertheless, in order to publish an application in Apple Store it is required to sign up for Apple's iOS developer program, which costs US$99. Once having the account, the application must be submitted to the App Store for approval, which can take some days.

For financtial and time reasons, the last objective proposed for this project could not be reached. However, in a future iteration of the project the app is going to to be published on the App Store.

5 PLANNING

The project was initially divided in five phases and schedule baselines were defined according to the scope of each phase. Nevertheless, during the project development some changes were made to the initial planning.

In this section we are going to present all these changes and to explain how evolved each phase in the course of the project.

Research and environment setting: Weeks 1-5

This phase was programmed from week 1 to 3. Finally it took twoo weeks more due to my ignorance about Swift programming language. Once I started with development stage, every time I had a doubt or I didn't know how to continue I had to go back to this phase and look at the documentation to keep learning.

Application design: Weeks 6-8

This phase was programmed from week 4 to 6. Due to the delay in first phase, it finally took from week 6 to 8.

Application development: Weeks 9-17

This phase was programmed from week 7 to 15. It finally took from week 9 to 17 because of the same reason.

Testing the application: Weeks 18-20

This phase was initially programmed from week 16-19 but for the same reason than the other phases it had to start twoo weeks later. Nevertheless, as delievery phase was finally removed and the application was not published to Application Store, testing gained one week more.

6 Results

The results for this project are not only the development of AllergenEasier iOS application but also its security and quality assurance. I have also learned a new programming language and gained experience with Appium automated testing. Last but not least, I got familiar with Firebase and obtained knowledge about some of its main functionalities as its real time data base and cloud storage.

All AllergenEasier desired functionalities could be developed and are going to be presented as follows.

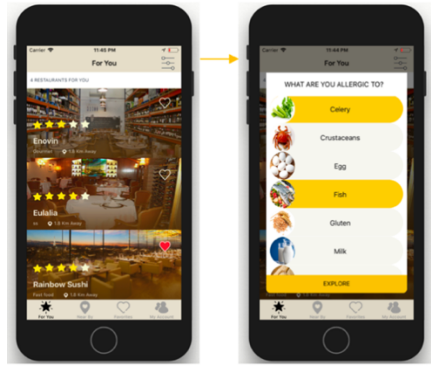

Fig. 18: AllergenEasier main screen and its filtering functionality

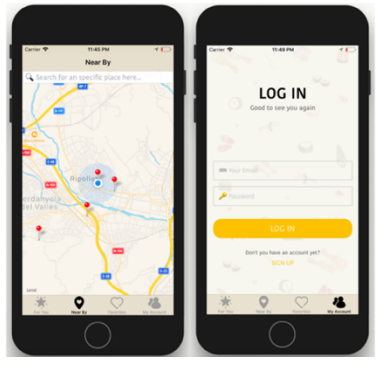

Fig. 19: AllergenEasier Log in and Location screens

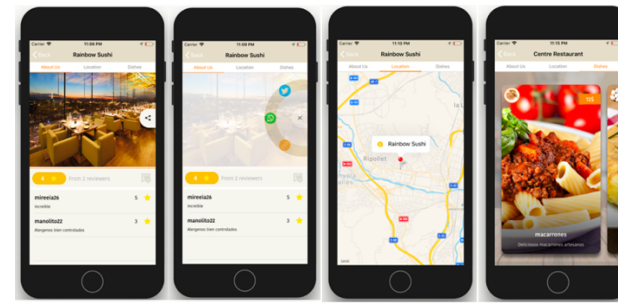

Fig. 20: AllergenEasier restaurant detail. It has three sections: About us, Location and Dishes

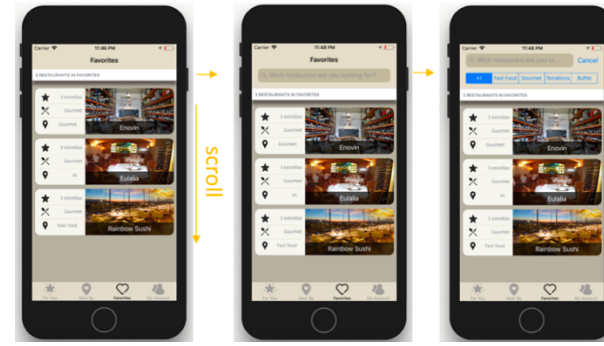

Fig. 21: AllergenEasier favorites screen and its filtering by category and name functionality

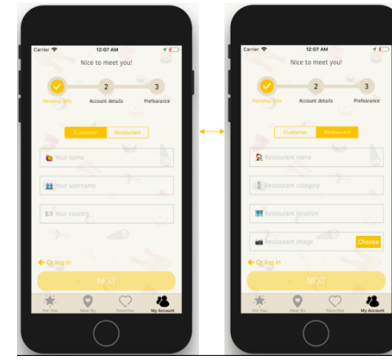

Fig. 22: AllergenEasier register screen. First step. Users have to choose between Restaurant or customer profiles. Button is only enabled when all textfields are filled correctly

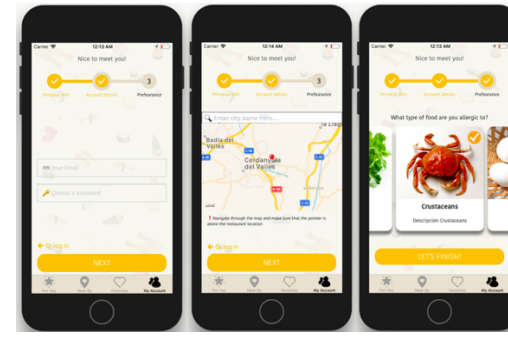

Fig. 23: AllergenEasier register: second and third steps. The first image is similar for both profiles. The second one is only in restaurants register in order to select its location. The last one is only for customers register and allows them to select their allergies.

7 Conclusions

Unless the last one, all objectives in this project were finally achieved. Nevertheless, there are lots of improvement ideas to be done in the application. The process of designing and developing an app has proven to be a longer experience than initially though with many changes and revisions occouring due to technological complications. However, this process brought countless learning opportunities, providing a large area for growth and development.

On the other hand, it is a fact that technology can be so useful and Applications like AllergenEasier can help to make life of people with severe allergies easier. Unfortunatelly, AllergenEasier can't solve the real problem. Even if a restaurant registers after being 100% sure that its dishes doesn't contain some allergen, one little mistake during its preparation can still kill them. AllergenEasier offers to the user the possibility to filter restaurants, but it is the user's responsability to be aware of what they eat.

8 Future improvements

VIP Clean architecture

In an MVC project, code is organized around and grouped by models, views, and controllers. In Clean Swift (VIP), the project structure is built around scenes (or screens). In other words, there are a set of components for each scene that will work for the controller. The components are the model, router, worker, interactor, presenter and configurator.

The communication between the components is done with protocols. Each component will contain Input and Output protocols which will be used for receiving and passing data between them. Worker communicates with Interactor, then Interactor with Presenter and Presenter with ViewController.

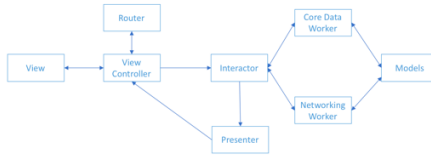

Fig. 24: Clean swift (VIP) flow diagram

Clean Swift contains a bit more coding and few more files than MVC, but that makes it easier for maintenance and writing test cases. The reason why AllergenEasier wasn't developed following VIP architecture was because it was my first project in Swift, so it would be too complicated.

Restaurants pagination

Pagination of iOS Table Views, Android List Views and on the mobile web is a common way of circumventing the technical limitations of power hungry mobile devices and slow mobile networks when dealing with large datasets.

Now there are only a few restaurants in AllergenEasier, but if its number increased in a future, pagination would be a great solution.

More automated tests

Time was an important limitation for the project, so in a future the number of automated tests should increase proportionally with the project growth.

Existing functionalities improvement: Delete user comments and restaurants

These two functionalities weren't implemented because of time limitations. Their implementation would be a priority in a future development.

General Data Protection Regulation (RGPD)

The main objective of the RGPD is to "give back to the citizens the control of their personal data while simplifying the regulatory environment of companies". As AllergenEasier works with user sensible data, a pop up should be implemented in order to inform him about his rights.

Analytics

With analytics help, some relevant information about AllergenEasier could be recollected not only for business purposes but also for improving restaurants service. If restaurants are aware of allergic user's real necessities, they could offer customized facilities adapted to them. For instance, these analytics could be obtained from filtering functionality. Having knowledge of the most commonly choosed allergens would be a good starting point.

Acknowledgments

I would like to extend my thanks to the following people. Manuel Martinez, Francesc Muñoz and Ramon Grau. Without the help of these three individuals my work would have been impossible. Manuel and Francesc, I appreciate the time you both took to share with me your experience and knowledge. Your ideas and suggestions were invaluable in the creation of this project. Finally, Ramon, your dedication to your students and your constant innovation has been pivotal for the success of this project.

References

[1] Slides and notes from Software Design subject. 2016-2017 course.
[2] Slides and notes from Test and Quality subject. 2016-2017 course.
[3] Wang, Wei, and Michael W. Godfrey. "Detecting api usage obstacles: A study of ios and android developer questions." Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, 2013.
[4] RayWenderlich.com. Programming Tutorials. [online] Available at: https://www.raywenderlich.com [Access: 12 Mar. 2018]
[5] Coursera.org. Introduction to Swift Programming. [online] Available at: https://www.coursera.org/learn/swift-programming [Access: 18 Feb. 2018].
[6] Code School. Swift Tutorial. [online]. Available at: https://www.codeschool.com/courses/app-evolution-with-swift. [Access: 18 Feb. 2018].
[7] Udemy.com. iOS 11 & Swift 4 - The Complete iOS App Development Bootcamp. [online] Available at: https://www.udemy.com/ios-11-app-development-bootcamp [Access: 18 Feb. 2018].
[8] Developer.apple.com. The Swift Programming Language (Swift 4.1): A Swift Tour. [online]. Available at: www.developer.apple.com [Access: 16 Feb. 2018].
[9] Youtube. Swift for Absolute Beginners. [online] Available at: https://www.youtube.com/watch?v=t7xUvFs3cPI [Access: 20 Feb. 2018].
[10] Developer.apple.com. MapKit. [online] Available at: https://developer.apple.com/documentation/mapkit [Access: 20 Feb. 2018].
[11] Appium.io. Mobile App Automation Made Awesome. [online] Available at: http://appium.io/docs/en/about-appium/intro/?lang=es [Access: 26 Feb. 2018].
[12] Cucumber.io. Getting started with Cucumber. [online] Available at: https://cucumber.io/docs [Access: 01 Mar. 2018].
[13] Proto.io. Prototypes that feel real. [online] Available at: https://proto.io/ [Access: 01 Mar. 2018].

APPENDIX

A1. COREDATA VS REALM

Ease of use. CoreData is a powerful tool, but it's not so easy to catch up and maintain. On the other side, Realm data models are defined using traditional NSObject-style classes with @properties. Realm objects can be instantiated and used standalone just like regular objects.

That's one of the main reasons for choosing Realm. CoreData would have supposed spending some extra time on its learning. Nevertheless, using Realm was easy due to its similarities with Swift objects.

Speed. Realm is actually faster than other popular mobile data storage solution, especially the queries speed.

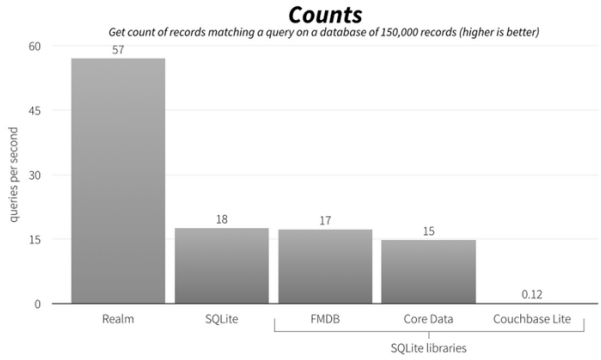

Fig. 25: Graphic showing the number of queries per second of different mobile databases, with CoreData and Realm between them. (Source: *quiita.com*)

Realm browser. There are a few solutions to explore CoreData database independently, but many of those are either expensive or not working. Instead of this, Realm has a utility called Realm Browser, which has been used during AllergenEasier development. It allows to easily explore Realm databases and edit them.

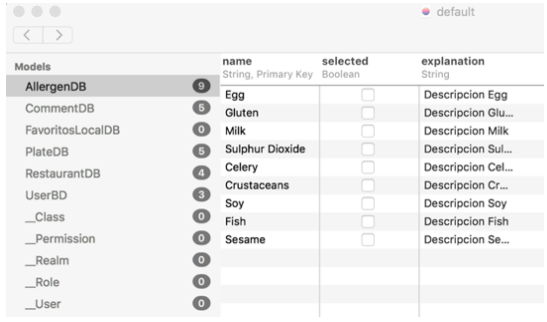

Fig. 26: Extract of Realm Browser. Allergens information stored in Realm database.

Cross-platform. Realm is also available for Android therefore it is possible to share the same data models over iOS and Android. That can be hardly achieved using CoreData. A possible future improvement for AllergenEasier would be its availability in both platforms so using Realm would suppose a huge advantatge.

Migrations. Even though Realm also supports migrations, the developer needs to do the heavy lifting. A powerful feature of Core Data is data model versioning and the framework's support for migrations. Versioning the data model is easy and lightweight migrations are handled by the framework.

Taking into account Realm and CoreData exposed pros and cons, the decision of using one or another depends on the needs of the project. As AllergenEasier requires not only encryptation but also speed and it hasn't got a complex data model that changes frequently, Realm is an obvious choice.