Cardiff University

Final Year Project

“Pocket Waiter”- iOS Application in SWIFT. Integrated Development Environment: Xcode

---

Author:
Athanasios Gkavalis

Supervisor:
Dr Martin Chorley

Moderator:
Dr Yu-Kun Lai

“I Always did something I was a little not ready to do. I think that’s how you grow. When there’s that moment of ‘Wow I’m not sure I can do this’ and you push through those moments, that’s when you have a breakthrough”

-Marissa Mayer
CEO of Yahoo

Abstract

This project is about an IOS application, which I programmed in SWIFT language and its main aim is to help people order when they sit at a restaurant. The idea of the application came up when I was sitting with my friends at a restaurant and we had been waiting too long to be served. So what I did, was to take my phone out of the pocket, call the restaurant and place my order. In the end, when they asked me where to deliver the food I told them, at table number 16. From that day on, I was thinking how easier life would be, if everyone of us could order from something almost everyone has; his mobile phone.

Since today, I haven't seen any restaurant use technology like that to help with customer's satisfaction and the restaurant's productivity. I strongly believe that the main reason of that, is because small restaurants can't afford a system like that, with screens to all tables that will allow people to order directly from them. So my main idea here was to build a cheap "system" for ordering and menu displaying, in a portable device that almost everyone has, like mobile phones. That would save a lot of money from restaurants which couldn't afford expensive systems like EPOS. So an application, where anyone can download from their smart phone and order directly from it, is a very revolutionary and low-budget idea, which can change the way people order today. It is as if almost everyone has a Waiter in his Pocket, who explains to them the menu of the restaurant and informs them of the price of each item, at the same time.

Acknowledgements

I would like to thank my supervisor, Dr Martin Chorley, for his unwavering support throughout this whole project. It has been marvelous to have guidance from someone close to my generation who listened to my concerns and problems patiently in every meeting and helped me calm down and overcome some stressful situations during the building of the application. His help has contributed through the three months of working on this application.

I would also like to thank my parents (Penelope Antoniadou and Stamatis Gkavalis) for their financial and emotional support during my whole "Undergraduate Career" as a computer scientist in Cardiff University.

Also, I want to thank Mr. Iheanyi Ibe from the Cardiff University Enterprise Centre for Skills, for his guidance and support with the Santander bank awards application.

Lastly, I want to thank all of my friends who always supported me and evaluated the application after the final implementation and they gave me valuable feedback on how to improve it.

Table of Contents

1 Introduction 8
1.1 Outline 8
1.2 SUMMARISED POCKET WAITER APPLICATION FUNCTIONS 9

2 The Background and Implementation 10
2.1 Market Research 10
2.2 Background Learning and Problems 12
2.3 Application Construction & Problems 14
2.3.1 Cloud Based Service 18
2.3.1.1 Introduction 18
2.3.1.2 MySQL Database 19
2.3.2 Problems & Solutions to App Functions 22
2.3.2.1 Restaurant Table View 22
2.3.2.2 QR Code Reader 28
2.3.2.3 Restaurants Nearby 31
2.3.2.4 Share Us 34
2.3.2.5 Quizzy 36
2.4 iOS App Development Risks 40
2.5 Competitors 42

3 The "Specification & Design" 44
3.1 User Interface 44
3.1.1 Buttons 45
3.1.2 App Logo 45
3.2 User Requirements 46
3.3 System Design 47
3.3.1 Table View 47
3.3.2 QR Code Scanner 49
3.3.3 Restaurants Nearby 50
3.3.4 Share Us & Quizzy 50
3.3.5 Bug Report 51
3.3.6 About Us 52
3.4 Database Design 52
3.5 API Details & Database Components 54

4 Results & Evaluation 55
4.1 Testing 55
4.1.1 Xcode Simulator VS iOS Device Testing 56
4.1.2 Differences Between Simulator and Device Testing 57
4.1.3 How testing is Performed 57
4.1.4 TestFlight, the Professional way of App testing 59
4.1.5 Speed And Runtime Testing 59
4.2 SWOT Analysis 59
4.3 Products, Services and Benefits 60
4.3.1 Product and Services 60
4.3.2 Benefits 61

5 Future Work 61

6 Conclusions 62

7 Reflection and Learning 62

8 Glossary & Referencing **64**
8.1 Glossary 64
8.2 Referencing 65

9 Appendix 70
9.1 UI Evaluations Testing 70
9.2 QR Code Scan Example 74
9.3 Business Plan 75

Table of Figures

1. Figure # 1. The time Users spend on Mobile Apps. 11
2. Figure # 2. Swifty, an iOS app which helps you practice with Swift 12
3. Figure # 3. General App's Xcode Page 14
4. Figure # 4. Project's Storyboard with a button item selected 15
5. Figure # 5. Swifty, an iOS app which helps you practice with Swift 16
6. Figure # 6. Outlet Connections between storyboard & code 16
7. Figure # 7. Identity Inspector 17
8. Figure # 8. Connection Inspector 17
9. Figure # 9. Backendless Database 18
10. Figure #10, PHP script for database connection 19
11. Figure #11, "GET" request for the PHP script from the server 20
12. Figure #12, the menu items before the JSON format 21
13. Figure #13, the menu items in JSON format 21
14. Figure #14, NSObject type declaration 21
15. Figure # 15. Scroll Down menu – color change upon restaurant selection 22
16. Figure # 16. NSObject Types, very important for JSON 23
17. Figure # 17. Dynamic table cell filling 23
18. Figure # 18. Dynamic restaurants table 23
19. Figure # 19. Restaurants table 24
20. Figure # 20. iPhone Settings Menu Table 24
21. Figure # 21. Tables Structure 25
22. Figure # 22. Table rows and sections declaration 25
23. Figure # 23. Table details in a more dynamic way 25
24. Figure # 24. Menu items type 26
25. Figure # 25. Navigation between tables 26
26. Figure # 26. Identity inspector 27
27. Figure # 27. Detailed view of items 27
28. Figure # 28. Scene Dock 27
29. Figure # 29. Line of code, responsible for table repetition 28
30. Figure # 30. iPhone Camera opens upon user permission 29
31. Figure # 31. QR code reader class 29
32. Figure # 32. The two versions of code- the first is outdated 30
33. Figure # 33. Capturing & Decoding of QR code 30
34. Figure # 34. Yellow QR box Identifier Specifications 30
35. Figure # 35. QR reader, when user scans a QR code 31
36. Figure # 36. Map Kit View 32
37. Figure # 37. iMapKit location settings 32
38. Figure # 38. Zoom at user's current location 33
39. Figure #39. Mapping spherical data to flat surface 33
40. Figure # 40. Positioning the red pins according to restaurants Geo-position 34
41. Figure # 41. 'Playroom' 35
42. Figure # 42. Share to Facebook service 35
43. Figure # 43. Pocket Waiter logo, embedded inside the app code 35
44. Figure # 44. Add a prefixed message to 'share us' function 36

45. Figure # 45. The figure declares all types of services that social framework provides 36
46. Figure # 46. Quizzy Game UI interface 36
47. Figure # 47. The Quizzy view controller at my storyboard 37
48. Figure # 48. Pick questions function 37
49. Figure # 49. String of questions & Answers 38
50. Figure # 50. NSLog developer reports 38
51. Figure # 51. iAd Network diagram 39
52. Figure #52. Application Upload to iTunes Connect Procedure 39
53. Figure # 53. iAd Network Implementation Guide 40
54. Figure # 54. iAd banner functions 40
55. Figure # 55. McDonald's ordering kiosks 42
56. Figure # 56. Different sizes of Pocket Waiter Logo 45
57. Figure # 57. First logo VS the updated logo 46
58. Figure # 58. App Name space differences 46
59. Figure # 59. The Restaurants-->the Menu & the detail view 47
60. Figure # 60. Dynamic table in storyboard & on device 48
61. Figure # 61. Email form with restaurant's email prefixed 48
62. Figure # 62. No Internet Connection Alert 49
63. Figure # 63. Algorithm for removing the status bar from an iOS device 49
64. Figure # 64. QR code reader locates the QR code 49
65. Figure # 65. Requests authorization from user, for his location monitoring 50
66. Figure # 66. Bug feedback 51
67. Figure # 67. About us 52
68. Figure #68. Database ERD (Entity Relationship Diagram) 53
69. Figure # 69. Restaurants Table 53
70. Figure # 70. Menu Items Table 53
71. Figure # 71. UML Diagram of the app components & interaction between them 54
72. Figure # 72. Test Case 56
73. Figure # 73. App performance while running on the simulator 57
74. Figure # 74. Different simulators & Device 58
75. Figure # 75. Project's General Settings 58

1 Introduction

1.1 Outline

Applications nowadays have “invaded” people’s lives and they have become very necessary. If you woke up on time this morning, it was thanks to the alarm app on your smartphone. Applications known as “Apps”, are programs on your computer or your smart phone that allow you to do something and make people’s lives more convenient. For example, if you left your coffee at home you don’t have to worry. A mobile app like Maps will determine your current location and suggest you espresso bars close to your area. Apps are about productivity, communication, entertainment and more.

There are more than three million applications out there, that have been designed to help people’s lives but also make them addicted to technology. My Project is called “Pocket Waiter”. I built it in Xcode on a Mac system, using SWIFT language and it is available for all type of iOS devices. Swift language, is a very powerful programming language for all kinds of apple products, which is interactive, expressive and the most important is that with that language, apps run lighting fast. Basically Swift language is a very simple combination of C and Objective-C language with low-level primitives like operators which provides developers with the power and performance they demand for their project.

Now, my project Application main use, is to let people order and view their food menu, when they sit at a restaurant, directly from their mobile devices. I think everyone has sat at a restaurant during a busy day, waiting for hours to be served. ‘Pocket Waiter’ is an iOS based Mobile Application aimed at the restaurant sector. While it is aimed at this area, emphasis is on small to medium- sized restaurants with limited resources and budget restrictions.

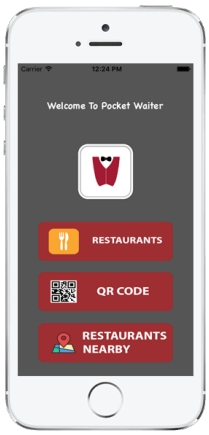
This application will provide ordering services to restaurants at an affordable or minimal cost. The application uses email services, which are provided for free from providers like Google or Yahoo, to process orders. In that way, restaurant owners with limited funds can have the same ordering system standard used by bigger food outlets like McDonald’s, at no considerable cost. The Pocket Waiter Application will reduce the time waiters invest in ‘waiting’ allowing their time to be spent on other areas and improve productivity.

The Application will be very beneficial for low-budget restaurants, because it is designed to use systems that are mostly provided for free to the society and will be available to restaurant owners at a minimal/competitive cost compared to similar existing products. It will allow our target audience to attract more customers, more business and invariably get more revenue.

My mission is to develop an affordable and efficient product to help our users operate their businesses in a more competitive way. The idea for the product came from personal experience and the ongoing acquisition of skills and knowledge from my time as a computer scientist at Cardiff University. However, I had to learn the Swift language and get familiar with the Xcode programming platform from scratch and without any academic guidance, so as to be able and create this iOS application.

Pocket Waiter, will load all the restaurants dynamically from a server and upon user restaurant selection, the respectively menu will appear. After costumer decides what he wants to order, he can

then place the order via email. The procedure of placing the order and sending it to the kitchen is designed to be free for both. After the user sees the main menu, he can then click on the “waiter” icon-button which is positioned on the top right of the menu of each restaurant and a special form will come up, requesting users to type their order and their table number. In the form, the email of the company is prefixed.



This new way of ordering, is meant to replace in the future the already expensive EPOS systems which require very high maintenance fees from restaurant owners. There are applications on the market who offer ordering services, but yet none of them is multifunctional and using email for processing the order. The reason is because large software companies promote centralised systems on the market and they try to sell as much equipment as possible, like PDA’s, so restaurant holders to depend exclusively on them. A couple lines above, I used the term multifunctional because apart from an ordering system it provides users with other extra features like Restaurants nearby, QR reader, social media integration, a quiz game and much more, so the user to has everything he needs, “wrapped” into one only application.

1.2 Summarised Pocket Waiter Application Functions

Restaurants

- It loads all restaurants directly from the Database (phpMyAdmin) and after that when the restaurant is selected, its main menu pops up and after user clicking on an item, it gives him more details about the item he selected. This function requires internet connection. Without internet connection, restaurants and items cannot be loaded and an error message is displayed, informing the used to activate his mobile data.

QR Code Reader

- It has a QR Code viewer button. When user touches this button, the iPhone camera loads and reads all types of unlocked QR codes and displays the message (i.e. Table number) on the screen of the phone, to inform the user about the table number he is sitting at. The great thing about this function is that it does not require any internet connection.

Restaurants Nearby

- Another function I built for the purpose of the project was the Restaurant Nearby. When user selects this function, a map pops up and zooms into his current location. Near the user location it shows the restaurants that operate with “Pocket Waiter” with red pins. When the user taps on the pin, more details come up like the name of the restaurant and its address

Share Us

- Share Us on Facebook, is another function which allows user to post something on his Facebook account. This function is designed to help with advertisement of Pocket Waiter, because each post a user will make will be followed by the app logo which is embedded into the app. The user, in order to be able to use this amazing feature, has to pre enter his Facebook account information in his iPhone settings. After that, the ID and the password will automatically be collected from the application and it will allow him to post.

Quizzy Game

- Quizzy is a small game that I created for users who are bored just to refresh their Facebook or Twitter page again and again, while they wait for their food. This is a question game, asks questions at random order and it only goes to the next question if the user has previously answered correctly. Also, this is a great opportunity, for the maintenance of my application. To be more specific, I have added an advert banner which will show adverts to users as long as the internet connection is enabled, so to be able to pay for the server maintenance fees.

Bug Report

- On the main App page, on the top left corner there is a small 'bug' icon. Obviously, this is for users to report to me any bugs or problems they found using the application. When user selects "Report a bug" it opens up his email account with Pocket Waiter email pre-entered and allows him to send his complaint to the developers of the Application (in the current situation to my personal Cardiff University mail account).

About Us

- On the top right corner of the application there is an info button. This is the "About Pocket Waiter" page, where users can find the name of the developer, the version of the application and some general information about the developer.

- To sum up, I was trying to build the application according to Apple suggestions, to provide best user experience. The environment of an application is not in command line, because the user is not necessarily a programmer. So I had to connect, the back-end with the front-end of the application and make it look very user friendly, in order to provide users with best experience.

2 The Background and Implementation

2.1 Market Research

For my final year project, I wanted to create something that was really missing from the market. Something that will "sail" side by side with the new generation requirements and provide comfort to low budget restaurants and bars. I was always wondering why there is not any App like "Just Eat", in restaurants and instead of waiting for someone to explain or bring the menu to you, to have it directly from your mobile device. So I strongly believe that an application like the one I have created, is really missing from the market. The concept of 'restaurant Apps' is not entirely new. Existing 'restaurant Apps' are predominantly used to explore local restaurants, find establishments near you, view menus and order takeaway deliveries. This is great if you are exploring what is available near you.

Pocket Waiter is ideal if you already know which restaurant you want to dine at or your favourite restaurant and also combines previous technologies which I mentioned above and many more, like social media integration or a quiz game and apart from that, it is designed to provide ordering services, mostly for free, to both, customers and restaurant owners. I focused on building an app that will easily be adopted on the part of the society in a sector that is constantly evolving, foodservices.

According to the UK Restaurant Market Report 2014 published by Allegra Foodservice, it was estimated that the value of the UK restaurant market will reach £48.2 billion in 2014 and rise to £52 billion by 2017. Branded eateries are forecast to grow by 6.5% over the next three years to reach a value of £17.6 billion by 2017. In particular, fast food outlet sales are forecast to grow by 12.4% and numbers of outlets are expected to increase by 7.6% between 2013 and 2014, with McDonald's and KFC leading the growth. The report also revealed that the most active consumers in the restaurant sector are aged 18 to 24, indicating opportunities for restaurateurs to engage better with older consumers [0]. This study shows that the sector of foodservices is evolving and that the most active consumers are young people, something which will facilitate the faster adoption of my application. Young generations always like changes and new technological innovations which will help them in their daily lives.

Furthermore, according to a 2012 report by Deloitte entitled 'Taste of the Nation', generational divisions have emerged in terms of the frequency with which individuals go out to eat and drink. Despite financial pressures caused by the economic slump, 18 to 34-year-olds are driving the market by eating out more - on average 31 times a month, up from 25 times a month in 2011. This is nearly double the rate among 35 to 54-year-olds and more than three times that of people aged 55 and over, who eat out on average just 11 times per month [55]. This survey clearly shows that young people are responsible for the growth of the market, and my app targets mostly this age group.

So it is clear that my “Pocket Waiter” application could be adopted quickly because we can observe that nowadays the world is “connected” via mobile devices and people are addicted to new applications. A research shows that people spend the 85% [1] of their time on their smartphone using an App while 40% of them have already placed orders online via their mobiles. Mobile orders can become a significant percentage of restaurant business.

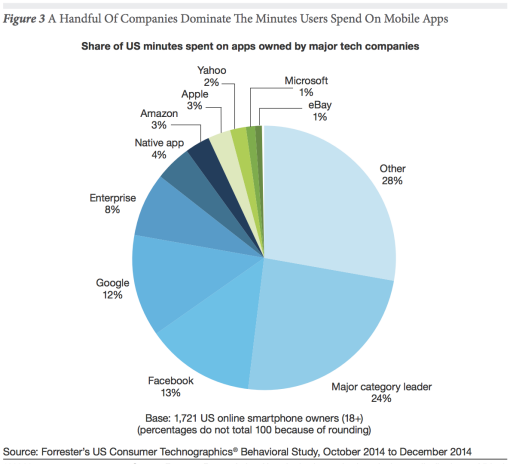


Figure # 1. The time Users spend on Mobile Apps.

2.2 Background Learning & Problems

Before starting searching and implementing the application, I had to contact my supervisor to my project idea. I visited Dr Chorley, I explained him my idea and we agreed on that project. The big problem for me was that I did not have any experience with application development and I did not know if in the end I would be able to demonstrate an application. I had to choose between creating an android application or an iOS application. I had read a statistic analysis, which clearly showed that iOS applications are more popular with 100bn downloads whereas the android downloads were at about 1,600,000m downloads [2]. So I decided to develop an iOS application, despite the big obstacle and difficulty of learning a new language, Swift, that Apple requires for iOS application development, within a very limited period of time. An android application maybe would be easier for me, because for its development, I had to use a language that I had been taught from my first year at University, Java.

Learning a new language by yourself is not an easy process because you do not have an anchor point to start and it is much more difficult when time is pressing you. I did not have someone who is an expert, to guide me through the correct steps of learning something new and protecting me from useless and inaccurate pieces of information that exist on the internet. The language I needed for the development of this project is called SWIFT. I started reading and learning this new language in the same way, Cardiff University introduced us to a new language called python, via its official book that was called “Learn Python the hard way”. So I downloaded the official Apple SwiftBook [3] and started learning a couple of basic variables names like “var” or “let”, some basic functions like print “Hello World” but the most important outcome I got from that book, was that If someone can understand the “Rules” of a new language, then it is much easier for him to start developing small apps from the beginning. To test what I learned from the SwiftBook, I downloaded a very well developed application called SWIFTY (figure #2). That application asks you very simple questions and as you move on, it levels up and gets more difficult. It really helped me understand the basics of Swift and allowed me to go deeper into that language and start building a more complicated code.

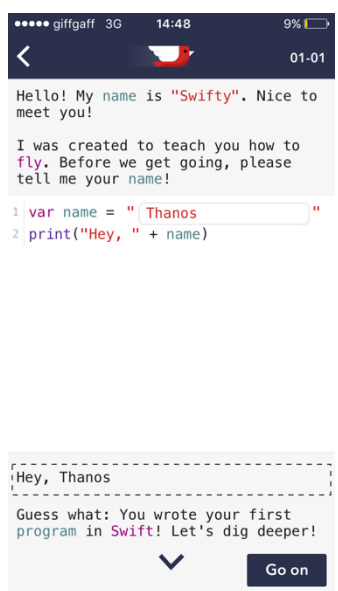


Figure # 2. Swifty, an iOS app which helps you practice with Swift.

Apart from the Swift language had to learn, in order to start building an application, I also had to learn how to use the basic app developing tool, Xcode. The combination of Swift language and Xcode IDE is the “Rules” I have mentioned above and they are both dependent on each other in order an application to be built.

After installing Xcode on my Mac computer, I checked some Apple’s existing projects so to introduce myself to the iOS app development, but the truth is that it was a really big shock for me at first, as I had no idea what each item and attribute, at Xcode, meant. Xcode is a very powerful tool but you have to spend a great amount of time in order to learn and understand how it is working. So in order to learn the basics of Xcode, I had to read an online book [4], which guided me to make my very first simple steps. The great difficulty with Xcode and Swift is that they are both continuously updated and most of the things I read on that book, were quite outdated and most of them did not work properly. Apart from reading that book, I watched many online videos on YouTube on how to make your first steps with Xcode and create a very simple application, but “YouTubers” are people mostly without any teaching experience and for me it was very difficult to understand how they got from one step to the next without any explanation. What I did was to visit a website called “Lynda.com” where I found videos galore on how to start building something from scratch, like a simple “Hello World” app. That website helped me create 2-3 very small and not so useful applications, but after that I got a general idea on how some items are constructed and how Xcode “reacts”.

After making a concept interface on a paper and started realising how Xcode works, I made my first UI Buttons. I had to use Photoshop in order to make buttons look nice. That is another factor that makes the app creation more challenging than making a program running via a terminal. I had to care about the front-end in order to make it user friendly. I had to learn about fixing all the constraints of the items I added to my application, in order to fit to all iOS devices and also follow the official Apple iOS Human Interface Guidelines [5] in order, this application, to pass Apple Design tests and get permitted from the Apple’s team, to be uploaded to the apple store and be available for download. To be more specific, at first I had created a very basic interface with a simple ‘back’ button, without using any navigation bar (Apple Human Interface Guideline). Obviously because of my lack of experience with Xcode IDE and Swift, I did not know that I had to use a navigation bar to “go back” instead of putting a simple “< back” button on the top left corner of my view controller of the application and connect it manually with the storyboard. Without a navigation bar, my application would not meet the “Apple Standards” so Apple would not allow me to publish it on the App Store.

Another problem which I have listed above and I had to overcome, was the constraints of some of my buttons. After placing the buttons to my storyboard and running the app, all the buttons lost their position. I found some guidance online on how to fix the constraints but it was quite confusing on the Xcode platform. Xcode does not give you the mobile device real size interface in order just to place all the buttons you want at the position you want. It gives you a square interface, and we all know that there is no iOS device (iPhone) with a square screen, and for each button individually, I had to add their vertical and horizontal position manually.

What constraints do is to describe positioning relationships between buttons and the view controller. After getting a feeling of how to build and design an iOS application, I created a new project in order to start building my Final Year Project application. With iOS development I ended up with more

questions than when we agreed with my supervisor to take this application construction as my final year project.

2.3 Application Construction & Problems

After a few small app samples, I started creating my Application for the project. Xcode asked me to provide a bundle Identifier and select a team. After some research I did online [6] about the meaning of them, I found out that I should have an Apple developer account and a certificate in order to be able to run the application on my phone and upload it to the app store. Thankfully, Cardiff University gave me an account so I was able to proceed with my project without having to pay 79 pounds, for the Apple developer account. So for the bundle identifier, I had to add a personal ID, in order my app to be unique and for the "team" I added the Cardiff University certificate, in order to be able to install the application to my iPhone and also make it available to the app store. (figure #3)

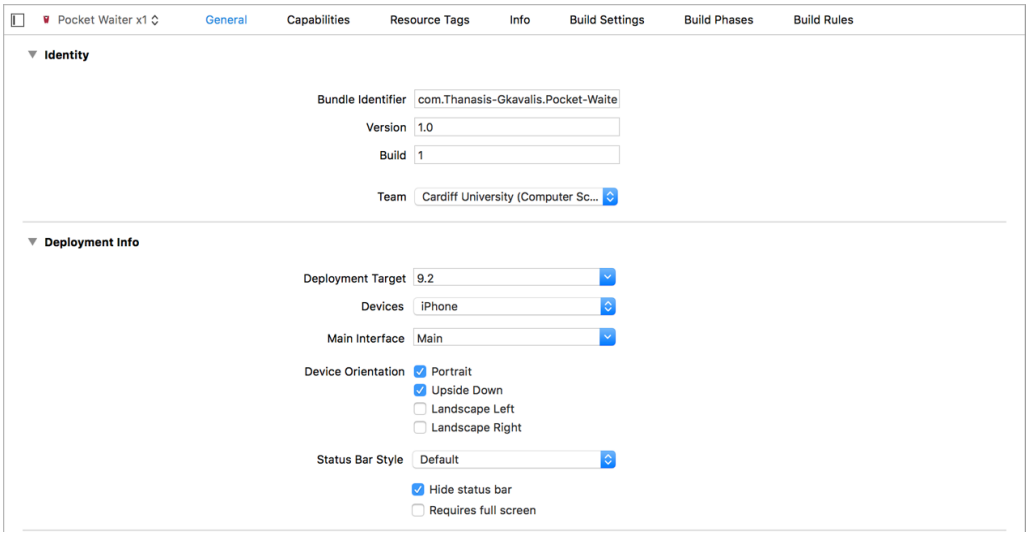


Figure # 3. General App's Xcode Page.

As I mentioned a couple of times before, constraints were a big headache for someone like me, with no previous background in Xcode. When I was trying to make my application look similar to my handmade design mock-ups, I faced a problem of the positioning of the buttons. There were some errors in the button constraints, that prevented my application from running and giving me an error message in the app delegate. The error did not mention where the problem was and it was just taking me back to the app delegate of my application every time I was running it. The app delegate is the application delegate. It is a helper object that takes responsibility for everything that happens to the application while is running and also it is normally used to perform tasks on application start-up and shutdown. Think of the app delegate representing the app and the view controller representing each individual screen. So, after a lot of experimentation with my application, by running it again and again, I found, that the missing constraints were the reason why my application was not running. What I did was to disable the automatic constraints option and add them by myself, manually.
I think Apple should make the Xcode IDE, more helpful to programmers with less experience and provide better guidance with errors, instead of giving them a single "There is somewhere an error we can't define", message. For me, these types of errors were very annoying and time consuming, because I had to go over my code many times in order to identify and fix them. After that, I wanted to follow my mock-ups and make my first list of restaurants in order to move on with my application. At first, I wanted the restaurants to be shown as a scroll down menu and change colour upon user selection using Picker View. I tried to make it in my main application but something was not working

and my whole code was messed up. To overcome that problem, I decided for everything new I wanted to implement, I would make it in a separate app and if it was working properly, then I would transfer it to my main app. That move, saved me a lot of time, because every time something was not working, I could fix it separately, without messing up my whole application. After a conversation with my supervisor, he suggested to me that the restaurants have to be connected to a database and loaded directly from the server, otherwise, every time I wanted to add a restaurant I had to release an update to the App store. Something that would be very annoying to customers who have downloaded and were using the app. The problem here for me was obvious, how to connect my application to a database. I did research [7] on that and I found out that I firstly had to create tables to my application, so in the future, to be able to connect it with a server and load data dynamically from it. Therefore, I had to convert the scroll down menu, into a table. At first, I used static cells because that was what I read from apple Xcode tutorial page [7]. To be more specific, for each restaurant, I added a cell into the table manually and to each cell I added the name and a photo of the restaurant, again manually. It was working and it looked quite nice, but afterwards, it would be impossible for me to connect it to a database and load data from the server but that was the anchor point for me, to start creating something. Afterwards, I transformed my static table cells into dynamic, in order later to be able to make them load from a server's database. Before talking about the tables, I want first to explain how I used the Xcode object library in order to be able to add buttons, labels and images to the application.

Xcode provides you with a library which allows developers to create items like buttons, faster. The object library which is positioned in the lower-right corner of the workspace window in the utilities area, contains the visual and auditory elements which you build your application's user interface.

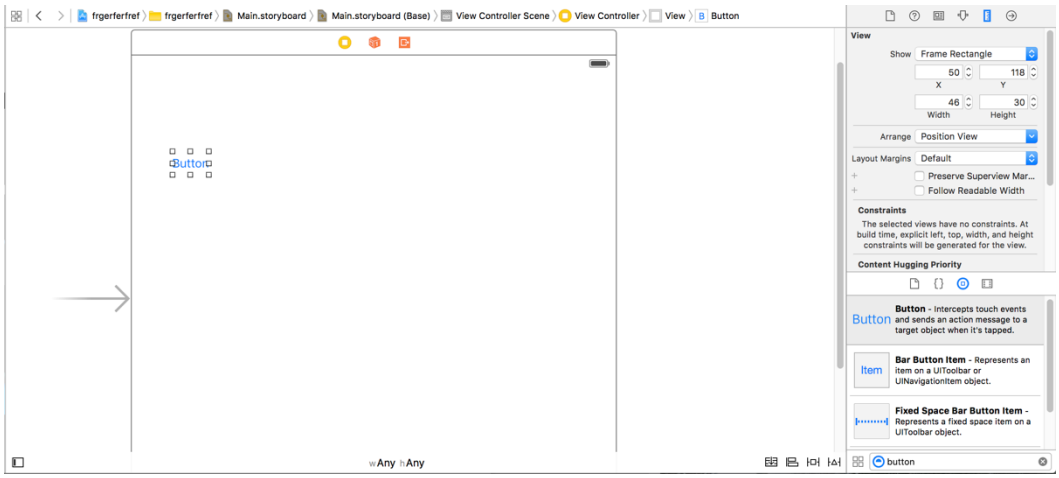


Figure # 4. Project's Storyboard with a button item selected.

After adding some elements from the object library like "Image View" or "Label" to my app's storyboard, I had to connect them to the appropriate piece of code from my project so to respond upon user request. The code communicates with the user interface objects via **Action** and **Outlet** connections [8]. When we need to send a message from a control to our code, we first need to create an Action connection. A control is an object of the user interface that makes instant actions when user uses the object. For example, when a user touches a button, the button sends an action message to the code in order to execute the proper action. The easiest way a developer can create a connection between a control (i.e. button) and the code, is by Control-dragging (figure #5) the control, which is at our storyboard, to the object's implementation file (code).

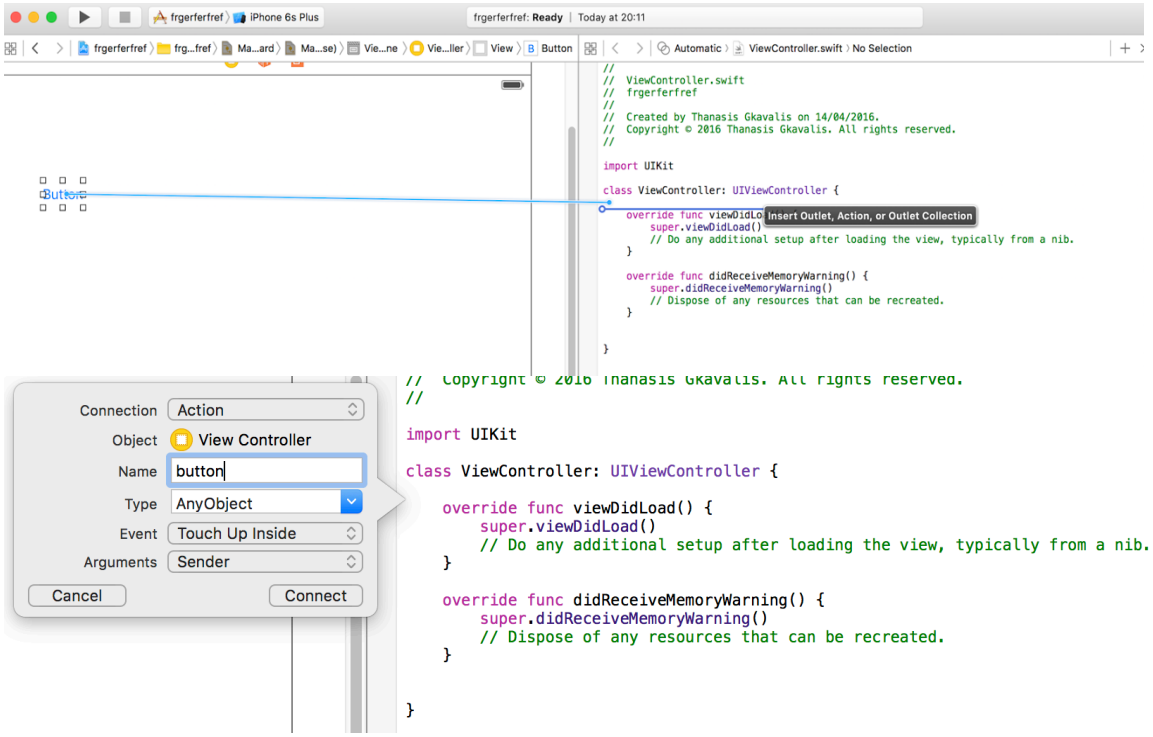


Figure # 5. Swifty, an iOS app which helps you practice with Swift.

On the other hand, if we need to send a message from our code to a user interface object, we need to create an Outlet connection. (figure #6)

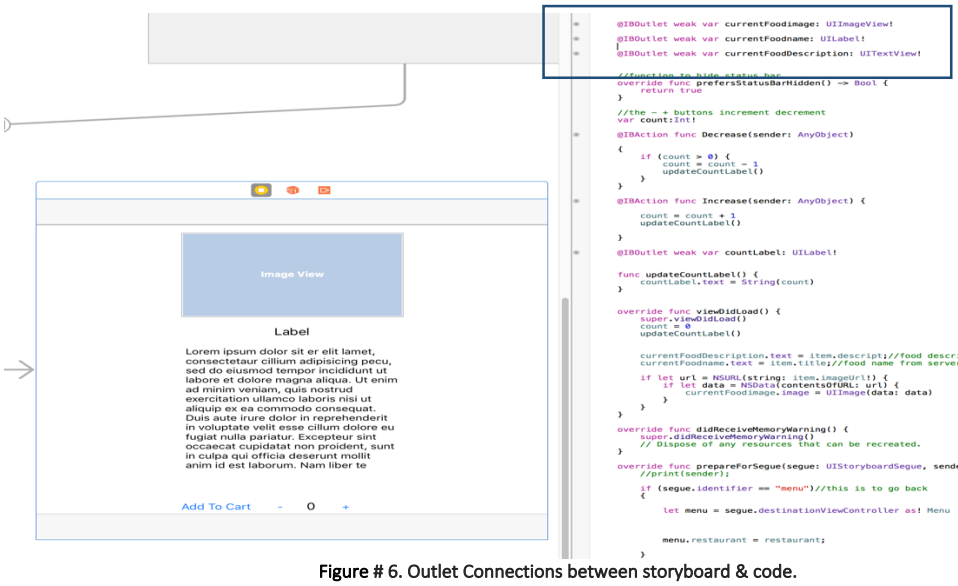


Figure # 6. Outlet Connections between storyboard & code.

A small problem I encountered with Action and Outlet connections was that when I added to my project more View Controllers, Xcode could not recognize how to match the view controllers from the storyboard with the correct code files. So it did not allow me to connect anything in the beginning when I added more than one view controller file. The problem with a situation like this is that you only learn these small details only by experimenting with the construction of your app. No one online who is not a teacher, will explain these small details to you, because they obviously lack of teaching experience and they also take for granted that beginners in Xcode, like me, should know these steps. After experimenting with my project, I found out that if you click at the view controller you want to

connect the code with, on the top-right corner there is a small square attribute called “show the identity inspector”, which allows you to select a class (figure #7), meaning the file you want to connect with your controller’s object. After selecting a class, Xcode allows you to connect your objects with Action or Outlet connections.

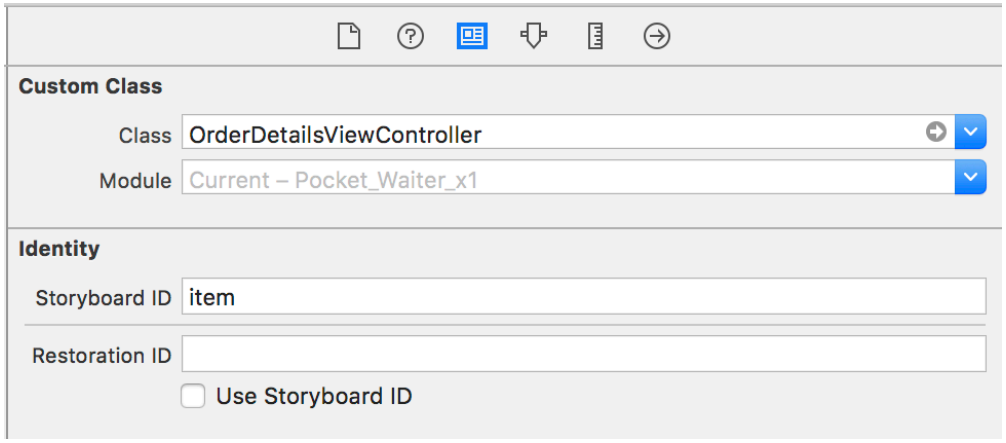


Figure # 7. Identity Inspector.

Now, if you have done a wrong connection you can change it by Right-Clicking at the controller’s object (figure #8) and click the X to the left of the connection name in order to remove the connection.

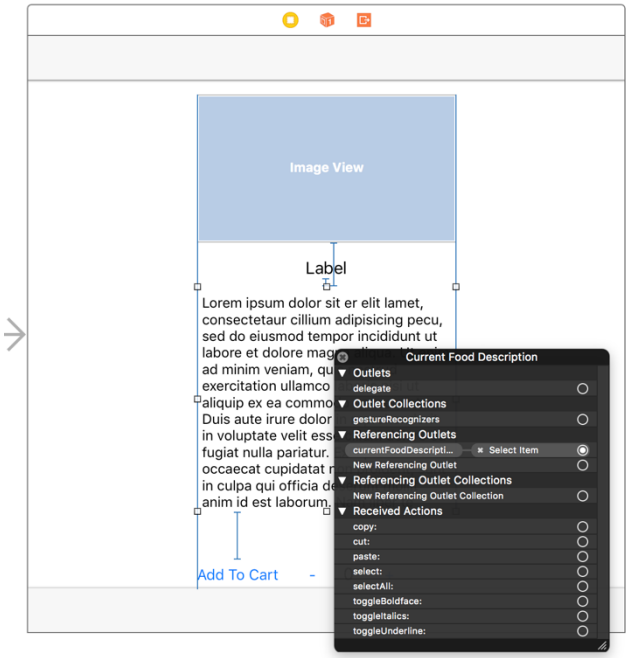


Figure # 8. Connection Inspector.

2.3.1 Cloud Based Service

2.3.1.1 Introduction

After fixing some small problems I had with the constraints, object connections and cell design, I started searching for a backend service to host my application, in order to reach my project requirements. I had read that there was an amazing backend service called Parse that was very fast and the installation was quite straightforward. It is a Backend as a service [9], which was acquired by Facebook in 2013. Parse allowed app developers to focus more to the frontend and enrich user experience, as the backend was taken care by the BaaS provider. Unfortunately, Parse announced that it was closing and new users like me, did not have the opportunity to create a new account. Parse suggested users should go and create an account to a cloud based system called backendless. So I created an account there and filled my tables but the big drawback with that service was that they weren't any tutorials on how to connect using Swift language. Also, for the connection with this backend service, I found out that I had to code in objective-c in order to be able to connect it with that server. My whole project was based in Swift language so I could not use even that service as a backend for my application. A couple of weeks before the deadline, after having the biggest part of my application working, I started searching for a database to store all my data and make them load dynamically to my application and finally, I successfully achieved to store and retrieve data to my application dynamically, from MySQL server via phpMyAdmin.

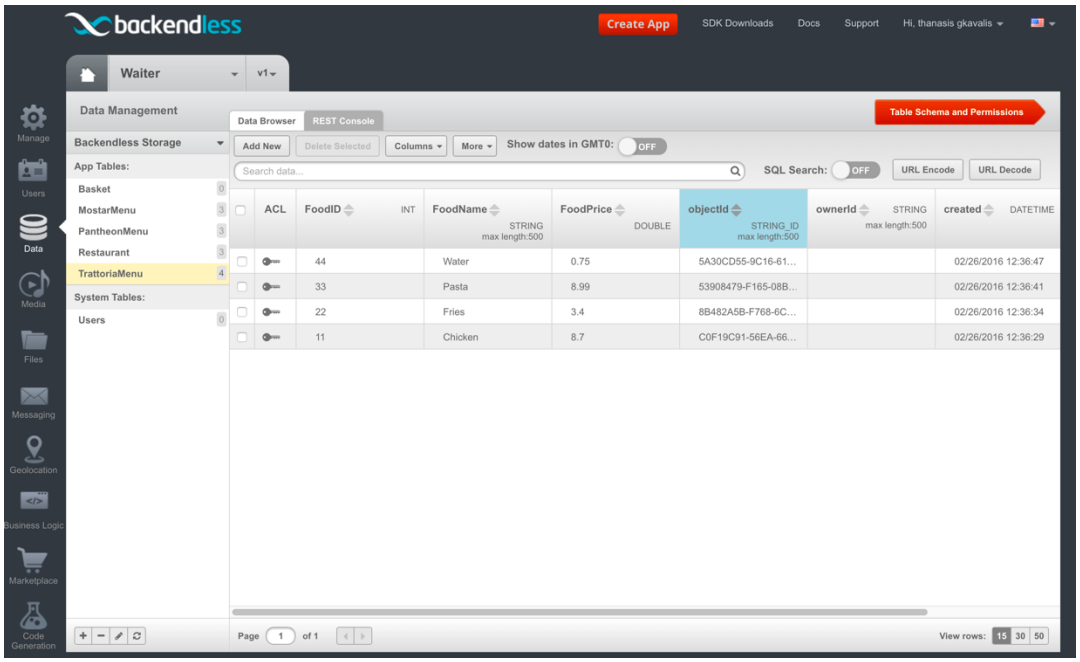


Figure # 9. Backendless Database.

2.3.1.2 MySQL Database

Background & Implementation

The connection of the application to load dynamically all the data from an online server, was by far one of the most difficult parts of my project. For that reason, I left it for the end. Before that I have converted all the tables, from static, to load data dynamically from a file, within my Xcode project. In that way it would be much easier for me to connect it afterwards, with the server.

The reason why I selected phpMyAdmin for the MySQL database connection, is because Cardiff university had already provided me with a MySQL account, from the first year of my degree. Just to specify, phpMyAdmin, is an open source tool written in PHP and its purpose is to take control of the administration of MySQL. Via phpMyAdmin, programmers can create, delete, update database tables or rows.

First and foremost, I created two tables, one for the restaurants and one for the menu items and afterwards I filled them with data. I found guidance from an online source [48], which helped me with the filling of the database tables.

After the successfully filling of the database, I had to create a php script so my application to be able to connect with the database and load the data dynamically. This php script includes the server login details, so the app to get the right permission for accessing the database and also what type of data to "GET" from the database. This script contains credentials for database connection and queries.

```php
<?php
$servername = "csmysql.cs.cf.ac.uk"; //Cardiff University Server
$username = "c1332970";//My academic student number as server username
$password = "Thanasis2016";//Server password. Different from my university password
$database = "c1332970";

      try
      {
      $conn = new PDO("mysql:host=$servername;dbname=$database", $username, $password);
      // set the PDO error mode to exception
      $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $restaurants = array();
        // get all restaurants
        $sth = $conn->prepare('SELECT * FROM Restaurant');
        $sth->execute();
        $result = $sth->fetchAll();

        // loop through restaurants
        // get menu items for each restaurant and add to restaurant array
        // add the restaurant array to restaurants array
        // r_id is very important, connects two tables, restaurants with menu items.
        foreach ($result as $row) {
                $sth2 = $conn->prepare('SELECT * FROM MenuItem WHERE r_id = ' . $row["r_id"]);
                $sth2->execute();
                $result2 = $sth2->fetchAll();
                $row['MenuItems'] = $result2;
                array_push($restaurants, $row);
        }

        echo json_encode($restaurants);
    }
      catch(PDOException $e)
    {
    echo "Connection failed: " . $e->getMessage();
    }
// close the connection - important!
$conn = null;
?>
```

Figure #10, PHP script for database connection.

As I previously stated, before we can gain access to data from MySQL database, we need firstly to be able to connect with the server and this can be performed only via the php script I have created (figure #10). This php script is a PDO (PHP Data Objects). PDO, is a database access layer which provides a uniform interface for accessing multiple databases. In the design section, I explain the reason why I preferred a PDO over MySQL and MySQLi, as database accessing layer. For the creation of this php script, my main and most accurate guide was the W3schools.com website at the php section, under the PDO example [49].

The script from the figure #10, was written in php language. The language I use for my project, is Swift and obviously I couldn't simply copy and paste that php script inside my application code. In order to be able and make my app read this script file, I uploaded the php file to an online server and afterwards I used a "GET" request within my main code to allow my application, read the php script. This file basically, is the connector between the app and the database.

```
Alamofire.request(.GET, "http://users.cs.cf.ac.uk/A.Gkavalis/waiter/").responseJSON {
            response in switch response.result {
            case .Success(let data):
                let json = JSON(data)
```

Figure #11, "GET" request for the PHP script from the server.

For the hosting of the php script, I used the Cardiff's University servers. In order to upload the php file to the university servers, I used the WinSCP program, on a windows computer. WinSCP is an open source FTP client which allows you to transfer files securely between a local (University Server) and a remote computer (my computer). I have previously used that FTP client in my first year in university but I had to refresh my memory for the installation and the access gaining. Fortunately, Cardiff University had provided us with the right guidance on how to gain access to school's servers remotely and also it has a fully documented website about accessing files remotely [50].

The PDO file I needed for the database and the app connection, is now available online and can be found in the above URL. (http://users.cs.cf.ac.uk/A.Gkavalis/waiter/). The big challenge for me now, was to search and find a way on how to make my code to access the php file from that URL. Luckily, after very intense searching on the web, I found the fastest, newest and the only library, written in Swift language, which allowed my code to read the php file over the URL. This library is called Alamofire. It is basically an HTTP networking library written in Swift. Xcode don't provide developers with this library, so in order to be able to import it to my code, I firstly downloaded the Alamofire library from the official creator's GitHub account [51] and afterwards I dragged and dropped it to my main project. The problem with the Alamofire when I was trying to install it to my project, was that in required from me to update my Xcode version and because I was running out of time I didn't want the last fifteen days before my project submission, an IDE update, to bring new bugs to my application. The Xcode update, fifteen days before the deadline, was a giant risk for me, but it was the only way that I could make Alamofire work. Fortunately, after the update I only had only a couple of alerts, which I fixed them, without any bugs.

Last but not least, the app couldn't read the database as it was. It had to be in JSON form. JSON stands for JavaScript Object Notation and it is a syntax for storing and exchanging data. In order to parse JSON I had to use SwiftyJSON. SwiftyJSON is a very simple parsing library, which gives a better and clearer syntax, than the build-in Xcode libraries. I downloaded the **SwiftyJSON. Swift** file from the creator's GitHub Account [52] and I dragged and dropped it, into my Xcode app project.

In order the data to start loading dynamically from the server to my application, I had to make some changes within my **RestaurantViewController.swift** file. I converted my previous dynamic table

(figure #12) into a JSON form (figure #13), so my application to be able and communicate with the server for “Getting” the appropriate request results.

```
let trattoria = Restaurant();

        trattoria.name = "Trattoria"
        trattoria.image = UIImage(named:"trattoria.jpg");
        trattoria.imageName = "trattoria.jpg";
        trattoria.menuItems = ["Chicken","Chips"];
        trattoria.itemPrice = ["8,50 £","2.99 £"];
        trattoria.itemImage = ["chicken.jpg","chips.png"];
```

Figure #12, the menu items before the JSON format.

```
menuItem.descript = json[inc!]["MenuItems"][i!]["descript"].stringValue
                        menuItem.title = json[inc!]["MenuItems"][i!]["title"].stringValue
                        menuItem.imageUrl = json[inc!]["MenuItems"][i!]["imageUrl"].stringValue
                        menuItem.price = Float(json[inc!]["MenuItems"][i!]["price"].stringValue)
```

Figure #13, the menu items in JSON format.

Swift is a very strict language with the data types, so before converting the NSObject class objects into JSON in swift, I had first to declare each data type within the **Restaurant.swift** file, located inside the “Models” folder in my Xcode project.

```
class Restaurant: NSObject {
    var id: String = ""
    var name: String = ""
    var image: UIImage?
    var imageName: String?
var MenuItems: [MenuItem] = [MenuItem]()
}
```

Figure #14, NSObject type declaration.

Languages I Used:

- Swift
- PHP
- JSON

2.3.2 Problems & Solutions to App functions

2.3.2.1 Restaurant Table View

As I said in my introduction page, creating a list of restaurants was quite challenging for me. At first I entered to my application a couple restaurants manually in form of simple names in order to create something to look like table cells and upon selection I made them change colour (figure #15).

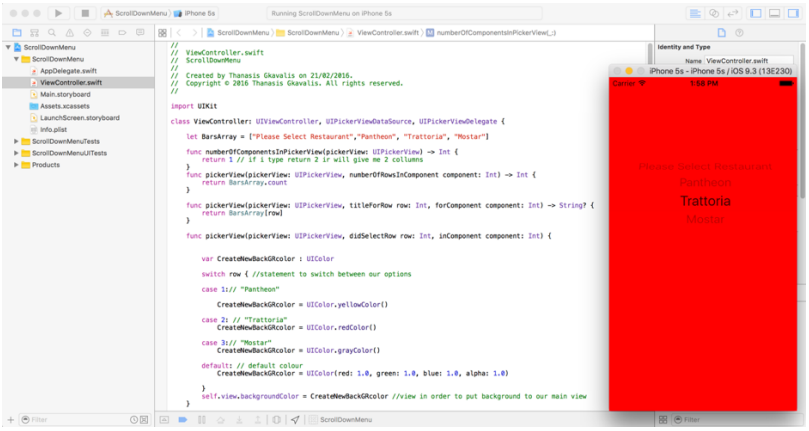


Figure # 15. Scroll Down menu – color change upon restaurant selection.

When I made my first attempt, it looked quite nice but I was sure that afterwards it would be impossible for me to connect it with a database, because it was completely static and all the restaurant names were inserted manually by myself. By looking at another application for online food ordering, which is called "Just Eat" I found out that that application used tables with cells in order their items to be displayed. After that, I started searching, from online resources, how to create tables. Again, Apple's official developer help website [10], helped me understand how tables work, how to display data into table cells and how to pass data from one table to another. But theory is always easier than actually developing and building it. I wanted to give another form to my restaurants, in order the display to look more professional. So I went to my main storyboard, I deleted my previous version of the restaurants list and I dragged and dropped a table view controller, to my main view controller. Afterwards, I entered manually again from the object library a couple of table view cells into my table view controller and I added a label and an image for each restaurant. It looked more professional now, but I had to find a more dynamic way to display my restaurants for future data retrieval from a database. As I have mentioned before, YouTube is not the best tool for a "rookie" developer, because people there help you build something with Xcode and they take some simple steps for granted by assuming that you know them. After subscribing to Lynda.com I found a very helpful video [11] that helped me find out how to make my table view cells more dynamic. What I did was to create a new view controller for restaurants and I added on it, a single table view with a single image view (an item from object library to display images) and a Label. After that, I had to connect these two items with my code. I created a new file that I named it CustomCell.swift and I did all my Outlet connections there, one for the image view and one for the label. After that, I created another file with an **NSObject** class. With an NSObject class (figure #16) you can specify the type of

your object. In my case, the name of the restaurant is specified it as a String and the image as a UIImage.

```
import Foundation
import UIKit

class Restaurant: NSObject {
var id: String = ""
var name: String = ""
var image: UIImage?
var imageName: String?
var MenuItems: [MenuItem] = [MenuItem]()
}
```

Figure # 16. NSObject Types, very important for JSON.

The procedure of converting something static into something more dynamic, in order in the future to allow me to connect the app with a server was very challenging. Later, I had to fill the cells with images and labels. At that point, I created my main RestaurantViewController.swift and I started filling my empty cells dynamic (figure #17).

Figure # 17. Dynamic table cell filling (code has been commented out in my code).

I cannot say that now my table is absolutely dynamic, because all the details had to be entered by me manually again, but in my storyboard now I had only one table view controller with only one image and only one label (figure #18), where they were both filled dynamically from another file I had for details (the file above). It was something like having a server embedded to my code.

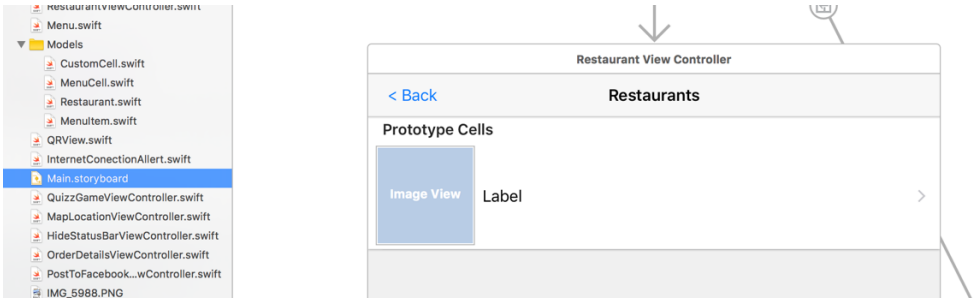


Figure # 18. Dynamic restaurants table.

After creating the list of restaurants to look like a table, I faced a small problem with my images. They were not displayed and I was not sure for the reason. After some research [12] I found out that the images had to be stored inside the Xcode project in a specific file called “Assets.xcassets”. What this file does, is to resize all the images in order to fit to table cells and avoid having problems with the constraints. After fixing this small bug, my first table was working absolutely fine and started looking more professional (figure #19). I also added some arrows in the end called “Disclosure Indicator” in order to inform the user that there is another page after clicking on a restaurant.

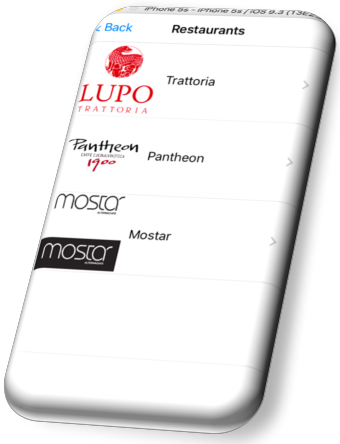


Figure # 19. Restaurants table.

Before moving on to my second table view and explaining how I reached the point to pass data between the two tables, I wanted first to explain a little more about my first table view and give some more information about it and why table views are so important to iOS development.

If you have used an iOS device like iPhone or iPad, for more than 10 minutes you have already used table views. These are one of the most usual iOS interface options for both third party developers and Apple's own applications. Whenever you see numerous rows of data being presented like this, is almost always a table view. A great example for my assumption is the main settings app (figure #20) of all iOS devices. It uses grouping rows into different sections and in each row there is a different control, where when a user clicks it, it takes him to another view controller.

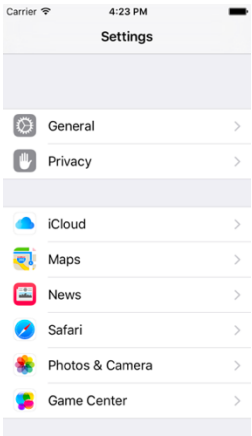


Figure # 20. iPhone Settings Menu Table.

Also the mail application or the music application, uses table views, in order to display the data like mail or songs in the form of cells, which provides a better user experience. Table views are one column wide. They do not store data and that is very important! Table views only show data that is stored somewhere else like an array or property list. So here, I had to understand this big difference between a table view and a database table. Now, each piece of data that is displayed in a table view is considered to be one row and each row contains precisely only one cell. In my app, in my first table view with restaurants, I have three rows and each cell has an image and a label. In order to be able to display images dynamically and data to my table view controller, I had to hook it up to a data source.

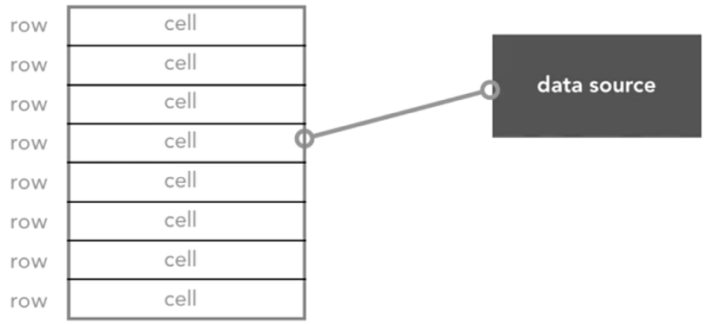


Figure # 21. Tables Structure.

Now for the data source, we should have an object defined as a data source for this table view and that data source needs to provide methods that will say at its most basic, how many sections the table has and in each section how many rows exist. We can hard-code these commands or when we create a new view controller, Xcode gives us the opportunity to select its property from the beginning i.e. **tableViewController.swift** and Xcode will create a file for us with some basic and mandatory functions we have to fill in order to create a table view.

The following two lines of code, give the program to understand how many sections and how many rows we want to our our table to have. In the second function, which refers to the number of rows of the table (figure #22), I could have typed "return 3" like the number of restaurants, but in the future if I wanted to connect it to a database, every time I wanted to add a restaurant I had to go back here and increment the number of rows in order my new restaurant to be displayed.

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }
    override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return restaurants.count
    }
```

Figure # 22. Table rows and sections declaration.

By returning **myrestaurants.count**, every time I want a new restaurant to be added, the only thing I have to do is to fill my RestaurantViewController.swift, with a restaurant's details (figure #23).

```
trattoria.name = "Trattoria"
    trattoria.image = UIImage(named:"trattoria.jpg");
    trattoria.imageName = "trattoria.jpg";
    trattoria.menuItems = ["Chicken","Chips"];
    trattoria.itemPrice = ["8,50 £","2.99 £"];
    trattoria.itemImage = ["chicken.jpg","chips.png"];


    myrestaurants.append(trattoria)
```

Figure # 23. Table details in a more dynamic way.

For my second table, I used the same tactic. Again I have created another view controlled and I have control-dragged from the object library another table view, in order the menu to start displaying items for each restaurant respectively. I have connected the table the same way as before with Outlet connections in order to interact with my code. After that, I created another file with objects (NSObject) (figure #24) and I defined each type of the items.

```
import Foundation

class MenuItem: NSObject {
    var title: String?
    var descript: String?
    var imageUrl: String?
    var price: Float?
}
```

Figure # 24. Menu items type.

With that table view I faced three problems and the two of them, took me a huge amount of time to solve them.

Firstly, I could not pass data between tables. I had done all the connections on the storyboard but when I was compiling the application, it was constantly taking me back to the app delegate, which means that there was an error. My worst nightmare, an error that Xcode couldn't specify and I had to locate and solve it. The problem with tables, is that they do not cope with simple connections from the storyboard in order to transfer the user from the one view to another. The navigation with table views had to be hard-coded and use a function called "PrepareForSegue". I was unaware of the fact that it was impossible to connect tables from the storyboard with a single control-drag line. After doing a lot of research online [13,14,15,16] about table views, I observed that all of them had a piece of code at the end, that seemed to be a navigator between tables. After my research in the aforementioned resources, I finally understood how this piece of code works and why is that important when you work with tables.

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        //print(sender);

        if (segue.identifier == "menu")//this is to go back
        {
            let menu = segue.destinationViewController as! Menu

            if let indexPath = self.tableView.indexPathForSelectedRow{
                menu.restaurant = restaurants[indexPath.row];
            }
```

Figure # 25. Navigation between tables.

The figure above, is from my project's code when I finally achieved to pass data from the restaurants to the menu. The "menu" which is positioned next to performSegueWithIdentifier, is the ID I gave to my menu view controller at the storyboard. When a developer uses more than one tables in his application, he has to go to the storyboard, select the table and give it, a unique ID, in order Xcode understand in which view controller to "take" the app user.

Adding an ID to a table view is a very easy procedure. The only thing you have to do is to select the table you want from the storyboard, click a square button on your Xcode IDE, called "identity inspector", located on the top-right of it and give an ID at the **storyboard ID** which falls under the identity (figure #26). But with iOS development and Xcode, if you have not understood or you are not familiar with the "Rules", sooner or later you will find obstacles that will be very time consuming for your project, especially when you have a short period of time.

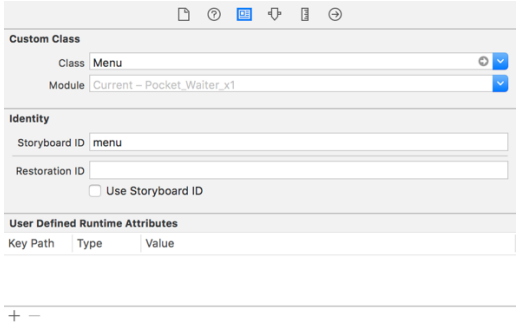


Figure # 26. Identity inspector.

After creating my second table with the menu items, I had the idea of creating a detailed view for each item separately, in order to give user more feedback about the item he selected. So now, after user selecting a restaurant and his preferable item from the “Menu Items” the application takes him to another view controller with more details of the item he selected. For example, when user selects an item from “Menu Items” a new page appears with a bigger image of the item, an item description and a label (figure #27).

Figure # 27. Detailed view of items.

Unfortunately, this nice feature of detailed view, has created a new bug to my application. The bug was quite annoying for the progress of my app development. The problem here was that if the user wanted to go from the detailed view of the item, back to the main menu, all the items were whipped out. I was very sure that the bug was not, more than a line of code. It was very tricky, because I used the same piece of code when I guided from one table to the other, without having any problem. The problem appeared when I added the detail view, so at first I thought it was something wrong with that file of code. But the solution, was much simpler. I did research and I found out that, because of Table View complexity, I had to embed in the table, a navigation controller [18]. To fix that bug, I clicked on the scene dock (figure #28) of my table view and selected “Embed in Navigation Controller”. What navigation controller did was to connect all the table views and detail view together.

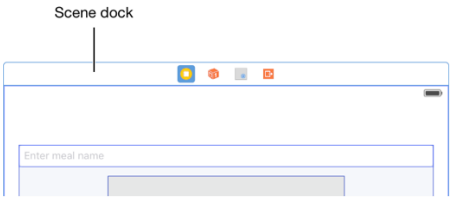


Figure # 28. Scene Dock.

After that I had two navigation controllers, the one that I have hard-coded (prepare for segue) and the one, that Xcode embedded into my project. After that, a second bug made its appearance and every time I was clicking on a menu item or a restaurant, it navigated there twice. To fix that issue, I got back to my code and commented out, a single line of code, which was responsible for the repetition of the table view display.

```
//the line below with performsegue is responsible for the error with whiping out the table
        //performSegueWithIdentifier("menu", sender: selectedRestaurant) //storyboard ID
```

Figure # 29. Line of code, responsible for table repetition.

2.3.2.2 QR Code Reader

QR code, is a machine-readable code, consisting of an array of black and white squares, which is mostly used for storing small pieces of information, like a short text or a URL, which afterwards can be read by the camera of the smartphone device [19]. The first letters, QR, mean Quick Response, something that is very useful and necessary to our modern and very fast in rhythms as well as growth of our society. The big difference of QR code over an ordinary barcode, is that the QR code can be scanned from a mobile device with a camera, vertically or horizontally with an instant response. Also, QR code stores data inside these square boxes that can be retrieved without internet connection.

I thought that a QR code reader will be a useful function for the “Pocket Waiter” users, in order to help them scan and find out their table number. Making a QR code reader from scratch would not be a wise idea, because I could have wasted my limited time there, and end up with a QR reader instead of an ordering system. I asked my supervisor whether I could get help from the internet, from a project similar to the QR reader I wanted to develop, and he strongly advised me that there was no reason for “reinventing the wheel” and he was absolutely right. The evolution of applications did not happen within a day, from a single developer. Daily, application developers around the world, have conversations via forum websites like stackoverflow, read about all the new updates that Apple releases and most important, they take a piece of code or an application and they try to improve it.

My first aim in order to make this function work, was to make the iPhone camera load. For a QR reader, camera is the most necessary hardware mechanism, a devise should have. I found a very helpful website [20] which helped me understand what kind of methods I had to implement in order to make a simple app, which just asks from an iOS mobile device to load the camera (figure #30).

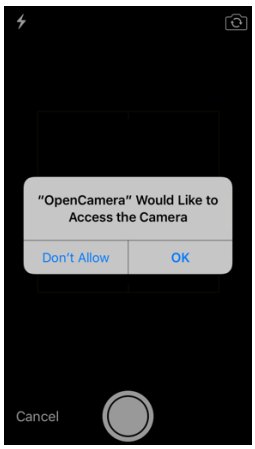


Figure # 30. iPhone Camera opens upon user permission.

As I have declared in my introduction page, every time I wanted to implement something new to my application, my strategy was to take everything apart. In that situation, I have created a new application where the only thing it did, was to make iPhone load the camera via the application. Afterwards, I started searching on how to build a QR reader, using Swift language. I found an interesting project on GitHub [21] which helped me understand what kind of methods an app like that should have. First and foremost, I realised what kind of frameworks I had to include to my QR reader application, what type of connections I had to do, in order everything to be displayed to the user and also this project guided me to search some protocols like **AVCaptureMetadataOutputObjectsDelegate [22],** which are necessary when you build an app that uses the camera of the device. His whole project, gave me a larger image of what I had to search and what I had to import to my code. In my case I had to import **AVFoundation** and at the **QRView** class, to add A**VCaptureMetadataOutputObjectsDelegate (**figure#31**).**

```
import UIKit
import AVFoundation

class QRView: UIViewController, AVCaptureMetadataOutputObjectsDelegate {
```
Figure # 31. QR code reader class.

It was a small step for developing this QR reader app but the problem here was that the instructions I found online about building a QR code were outdated and even my Xcode simulator could not run his QR code project. Here comes the big difficulty in developing iOS applications. Apple continuously releases new updates and developers have to be daily informed about Apple future updates, in order to make their applications run smoothly on all iOS devices. I got a general idea on how a QR reader works but I had to find a newer version of it, in order to be able to build something similar to the one I wanted to. My first attempt to make a QR reader was by importing a framework from the Xcode. I found that method online [23] but it was not as simple as it looked. The problem here was that when I was trying to import the Framework to the app, Xcode did not recognise that framework. After a couple of attempts, I discovered that this framework had to be created by me, by installing the cocoapods of this QR code reader, from my Mac terminal.
CocoaPods [26], is a dependency manager for Swift and Objective-C. It has thousands of libraries that can help you scale iOS projects elegantly. These are some amazing tools that experienced app developers, who know the Apple "Rules" use, in order to progress with their projects faster and in a more elegant way. These libraries, are daily tested on the part of developers and they are continuously updated, in order to "sail" side by side with Apple latest requirements.

After understanding what I had to import to my code and how some objects like AVCaptureMetadataOutputObjectsDelegate work, I found a great tutorial online [27] which helped me build a QR reader. The problem I faced here was that the tutorial was based to an older Xcode version and when I tried to build something similar to that, my Xcode was giving me errors. The QR reader application was working perfectly when I first made as separate app, but when I tried to implement it to my code, which was based on a newer version of Xcode, I had connectivity issues. What I did here, was to make some changes-updates to the QR reader code. Some of them were small changes, for example at the end I had to add (;) or (!) or in some cases I had to replace **var** with **let** but some others changes were more challenging. A great example here is that I had to update a line of code from the VideoSnapReview function in order to be compatible with iOS 9 and later versions. In the figure #32, I have commented out the line which game me the error and prevented my app from running properly. Under the first line of code from figure #32, is the newer version of it.

```
do {
            //PhoneDeviceInput = try AVCaptureDeviceInput.deviceInputWithDevice(PhoneDevice)
            PhoneDeviceInput = try AVCaptureDeviceInput(device: PhoneDevice) as AVCaptureDeviceInput
                                                                                    }
```

Figure # 32. The two versions of code- the first is outdated.

The exciting thing about QR Readers, is that they can read instantly any QR code which is "unlocked" without internet connection. By the term **unlocked**, I mean that some companies use QR codes to hide important messages, like the passenger details on the airplane tickets, which these can only be read from special QR readers, which require internet connection for verification. In my QR code app the following code from figure #33, is responsible for capturing the QR code and decoding the information which is embedded in it.

```
// AVCaptureOutput Delegate Standard Function for QR code reading
    func captureOutput(captureOutput: AVCaptureOutput!, didOutputMetadataObjects metadataObjects: [AnyObject]!,
fromConnection connection: AVCaptureConnection!) {

        if metadataObjects == nil || metadataObjects.count == 0 {
            QRCodeMain?.frame = CGRectZero// again apple data types for geometry CGRectZero
            lblQRCodeResult.text = "No QR Code Found"// it will show that message as long as our phone camera
doesnt detects a QR code

            return;
        }
        let QRMachineReaderStandard = metadataObjects[0] as! AVMetadataMachineReadableCodeObject

        if QRMachineReaderStandard.type == AVMetadataObjectTypeQRCode {
            let objBarCode = SnapCaptureVideo?.transformedMetadataObjectForMetadataObject(QRMachineReaderStandard
as AVMetadataMachineReadableCodeObject) as! AVMetadataMachineReadableCodeObject
            QRCodeMain?.frame = objBarCode.bounds;
            if QRMachineReaderStandard.stringValue != nil {
                lblQRCodeResult.text = QRMachineReaderStandard.stringValue
                                                                                    }
```

Figure # 33. Capturing & Decoding of QR code.

Once completing this piece of code and after a little debugging, I wanted to initialize a yellow box in order to highlight the QR code. To achieve that, I had to create an UIView object which was already embedded in the Xcode library, and add the square frame details, like its colour and its width.

```
//this shows the yellow square...without it, it still shows QR code but without square
    func PrepareQRCodeSquarePreview() {
        QRCodeMain = UIView()
        QRCodeMain?.layer.borderColor = UIColor.yellowColor().CGColor //QR Code Square colour = yellow
        QRCodeMain?.layer.borderWidth = 3 //border width 3..we can clearly see it.
        self.view.addSubview(QRCodeMain!) // add the subview
        self.view.bringSubviewToFront(QRCodeMain!) //show to preview in the front +  send to the front and animate
to the center of the view.
                                                                                    }
```

Figure # 34. Yellow QR box Identifier Specifications.

So now every time the user wants to scan the QR code from its table, the only thing he has to do is to click on the “QR CODE” button which is positioned between the “RESTAURANTS” & “RESTAURANTS NEARBY” buttons and when he scans the QR code, a message will appear at the navigation bar with the number of the table (figure #35).

Figure # 35. QR reader, when user scans a QR code.

2.3.2.3 RESTAURANTS NEARBY

Nearby Restaurants and the QR code reader, were the two extra functions that I really wanted to add to my application, in order to make it look professional but most importantly I wanted it to be unique and multifunctional. Applications nowadays, in order to “Survive” they have to look like something similar to a Swiss Knife.

To be more specific here, everyday new applications are being created and the ones which do not follow user preferences, they become less downloadable and at a certain point, users stop using them. By the term “Swiss Knife” I mean that applications nowadays, should provide users with many other functionalities apart from the main one that they have been designed for. A great assumption to support my example is that more and more social applications like Instagram or Snapchat, apart from their main functionality which is photo uploading, they have embedded in a chat function, in order to keep their users. Imagine how many users, the Facebook would have lost if it did not provide their users with the “messenger” app.

My ambition for this application, is when a user clicks on the “Restaurants Nearby” button, a map to be loaded and zoom into the user current location and show the restaurants in form of red pin annotations, so the user finds places to eat near him, compatible with the “Pocket Waiter” app. I also designed the code in a way, so when the user clicks on a red pin, more details to be displayed for the restaurant, i.e. its post code. It was a great idea to have this function inside my main application, but I knew that it would be challenging to build it plus the fact that time was pressuring me, to finish with the main functions of my project like, connecting my app with a server so, all restaurants and items to be loaded directly from it. My main motivation to try and insist on building this function, was my supervisor, who from the time I told him my idea, he liked it and he strongly advised me to find a way and build it in the end, because as he said, it would be very interesting to have a ‘location app’ inside my “Pocket Waiter” app.

The tough part, when I started searching and building this function was the fact that in parallel, I was trying to make my app load data dynamically from a server plus that I was writing my dissertation report because time was getting eliminated from day to day.

Before doing any research on how to add a map to my application, I went to the object library of Xcode to see if Xcode provided me with a map library or I had to search online for a map library. Luckily, Xcode provides iOS developers with the “Map Kit View” library (figure #36).

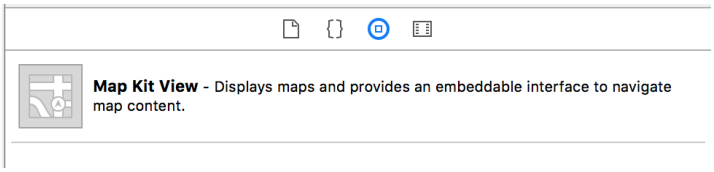


Figure # 36. Map Kit View.

In order to learn how to use this map kit, I went to the Apple’s official website for developers [28]. Firstly, I wanted to create a simple app which just locates, with a blue dot, the current position of the user. Unfortunately, all of the apple suggestions and help books were in Objective-C and for my project I was using Swift language. I could have combined these two languages together, but I did not want to just copy & paste code from the internet without having any idea on how to use it. So I started searching for other resources online to get helped and understand the **Map Kit View deeply.** Before starting hard coding, in a new view controller I added the Map Kit from the Xcode object library, after that, I created a new Swift file at my project, which I named **MapLocationViewController.swift** and in the end I made an Outlet connection between them, in order the storyboard to respond to the code. After importing the Map Kit and the CoreLocation from Xcode library, I started building the current location finder app. The **Map kit** is a framework, is based on the Apple Maps and APIs and provides iOS developers with a simple mechanism for integrating detailed mapping capabilities into any application. On the other hand, the **CoreLocation** is another framework, created by Apple, which gives developers the opportunity to get user location. My first aim for this app was to find user’s current location, with the user permission. To make this locator app, work at its best, I made it updating user current location every second and I set its accuracy to the highest level (figure #37).

```
        self.UserlocationManager.desiredAccuracy = kCLLocationAccuracyBest //user current location with best
results, user exact location

        self.UserlocationManager.requestAlwaysAuthorization() //use location services only after user approval

        self.UserlocationManager.startUpdatingLocation() //turn on location..start searching for user location
```

Figure # 37. iMapKit location settings.

Apple provides this feature [30] of location accuracy to all iOS devices with a newer firmware that 2.0. For example, I could have replaced the first line of code from the figure #37, with **kCLLocationAccuracyNearestTemMeters**. With that simple change, the user's current location would not be precise, but it would be accurate to within ten meters of the desired user location. I also set the map to be two dimensional (2D) and additionally I made the app zoom into current user's location. I found this structure (figure #38) on the official Apple's help for developer's page [31], under the Data types.

```
let regionZoom = MKCoordinateRegion(center: centerCircle, span: MKCoordinateSpan(latitudeDelta: 1, longitudeDelta:
1)) //circle were map zoom to and spam numbers are how to zoom


        self.mapView.setRegion(regionZoom, animated: true) //go to that region and zoom in, animation is for zoom
```

Figure # 38. Zoom at user's current location.

The **latitudeDelta** is the amount of north-to-south distance to display the map.
The **longitudeDelta** is the amount of east-to-west distance to display the map.

The **Delta values** in my code (1,1) indicate the level of the desired zoom on the map. The smaller the Delta values are, the higher the zoom level.

To understand the coordinates systems used by Map Kit, we have first to understand how the three-dimensional (3D) surface of the Earth, is mapped to a two-dimensional (2D) map (figure #39).

Figure #39. Mapping spherical data to flat surface.

Map Kit uses a cylindrical map projection [33] like the one in the figure above. In a Mercator Map projection (cylindrical map), the coordinates of a sphere are positioned onto the surface of the cylinder, which is then unwrapped to generate a flat map. This type of map became the standard map projection for nautical purpose [34], because of its ability to connect object's consecutive positions. The advantage of a Mercator map projection is that content of the map, is scaled in a way that benefits the general navigation. The map projection which is used by Map Kit, uses the Prime Meridian as its central meridian.

Once I had managed to locate current user location, I wanted to enter three annotations to display the location of each restaurant that is listed inside my "Pocket Waiter" app to the map I had previously created. To get this done, firstly, I needed the longitude and the latitude of each restaurant. For the purpose of the project I went to an online latitude & longitude generator [32] and I created three coordinates near my University. After that, I added a title and a subtitle for each annotation in order to give further details to the user about the restaurant (figure #40).

```
let userlocation = CLLocationCoordinate2DMake(51.484478, -3.1702539)// i added these coordinates to be close to my
possition to help the presentation, so i made them close to cardif university
        let userlocation2 = CLLocationCoordinate2DMake(51.5030142, -3.1941843)
        let userlocation3 = CLLocationCoordinate2DMake(51.48865188, -3.26517105)

        let annotationPin: MKPointAnnotation = MKPointAnnotation()
        annotationPin.coordinate = userlocation
        annotationPin.title = "Trattoria"
        annotationPin.subtitle = "Cardiff Newport Road 12"


        let annotationPin2: MKPointAnnotation = MKPointAnnotation()
        annotationPin2.coordinate = userlocation2
        annotationPin2.title = "Pantheon"
        annotationPin2.subtitle = "Cardiff Quentin Street"

        let annotationPin3: MKPointAnnotation = MKPointAnnotation()
        annotationPin3.coordinate = userlocation3
        annotationPin3.title = "Mostar"
        annotationPin3.subtitle = "St.Fagans street,Cardiff"

        self.mapView.addAnnotation(annotationPin)
        self.mapView.addAnnotation(annotationPin2)
        self.mapView.addAnnotation(annotationPin3)
```

Figure # 40. Positioning the red pins according to restaurants Geo-position.

The big difficulty I had when I was constructing this app, was the combination of all this information from different resources. For example, in order to find the user location and understand the iMapKit object meanings, I had firstly to read the Apple's developers page and after realising how it should be done in order to work properly, I 'decoded' some tutorials (that I have referenced in my code) which used older versions of Xcode and sometimes Objective-C. For the annotations, I read the Apple official page, and I realised that I had to use a title, a subtitle and their coordinate position, when I wanted to create one annotation. The last part with the annotations, I built it exclusively, by trying numerous of times to combine the information, I found in the Apple developers' website.

2.3.2.4 Share Us

After 2004, when Facebook made its first appearance, the way people used to communicate in the past, has dramatically changed. Worldwide, there are over 1,59 billion [35] active Facebook users. What this means for new generation developers is that Facebook is too big to ignore. Because of the fact that every minute on Facebook there are posted more than 135k photos and almost 290k statuses are uploaded [35], I wanted to give both, to my application and to the app users, the opportunity to share their ideas and their thoughts to this enormous social network, via my app. Inside my application, when the user clicks on the "Pocket Waiter" logo in the main menu, a new view controller appears which is more fancy that the rest. I have named this space "Playroom" (figure #41). It is a place inside my app, where the user can spend his time, while waiting for his order.

Figure # 41. 'Playroom'.

When the user clicks on the **Share us** button, the application automatically pops up a form to the user's device display, with the "Pocket Waiter" logo pre-installed to that form. After that, the user can share his preferable status on his Facebook account. This way of status sharing, is beneficial for both, my application and user. The reason why this add on is so important for my application is because users cannot post a status without the pocket waiter logo on their status. This is a basic marketing strategy I have formed, in order to advertise my application via user shares and statuses. The construction of this function was quite straightforward because Xcode provides developers with an already made social sharing framework.

Primarily, I added the **Social.Framework** to my application, in order to be able to import afterwards, the class "Social", to my **PostToFacebookViewController.swift** file where I was building the code for this sharing function. Before starting coding, I created a new view controller with a 'share us' button on it and I linked it with the swift file, via an Action connection. After that, I created a variable and I called it "ShareToFacebook" in order to set this function to be posted directly to Facebook, by automatically taking the username and password details of the user from his main iPhone device settings. The user is required to have previously entered his Facebook account information to his iPhone settings menu.

```
var ShareToFacebook : SLComposeViewController =
            SLComposeViewController(forServiceType: SLServiceTypeFacebook)
```

Figure # 42. Share to Facebook service.

With the above figure #42, by declaring the type of service to be: **SLServiceTypeFacebook,** I navigate this app to find user's Facebook account information. At the moment Xcode supports services for Facebook and Twitter. For example, if I wanted instead of Facebook to make my application share to twitter I only had to convert a single line of code and from **SLServiceTypeFacebook** to change it to **SLServiceTypeTwitter**. In order to make the app users, unable to delete the "Pocket Waiter" logo from their shares, I firstly added the Logo to my Assets.xcassets file of my project and then I 'called' it in the ShareToFacebook line of code (figure #43).

```
ShareToFacebook.addImage(UIImage(named: "PWlogo.png"))
```

Figure # 43. Pocket Waiter logo, embedded inside the app code.

The **ShareToFacebook.addImage** line of code, is in order to make the device, recognise the service type of the composing posts. For example, if I wanted to add a prefixed text, I had to add the service type called: **.setInitialText(_:)** and the message I wanted. A great example is the following figure from my code:

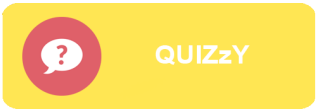
```
ShareToFacebook.setInitialText ("Hey, I just used Pocket Waiter For my order and everything was done, as fast as my laundry besket gets refilled xD")
```

Figure # 44. Add a prefixed message to 'share us' function.

In order to be able to learn about all these service types that the Social statement provides, I had again to visit my main guide, the official Apple developers page [36]. It is the only source that has the latest updates directly from Apple and it is the main website, where iOS developers around the world, got educated from.

Figure # 45. The figure above is from Apple website, which declares all types of services that Social framework provides

2.3.2.5 Quizzy



The Quizzy, is my last extra function I added to my application, to make it more attractive to users. Quizzy is a Question game, which tests user's knowledge, by displaying random questions and user has to choose among four possible answers. By the time he makes the right choice, the Quizzy game displays to him another question at random order. If user closed the application and opened it again the Quizzy game, the question would be different from the previous one.

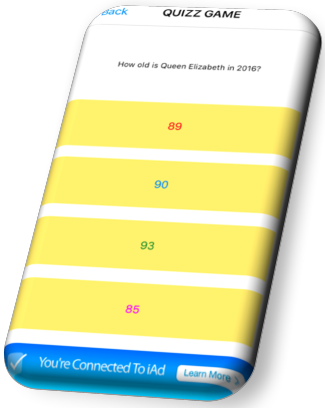


Figure # 46. Quizzy Game UI interface.

Again for the development of this application I used dynamic table cells instead of having hundreds of view controllers with static data on them. The figure below, shows one view controller with a label and four buttons which will load data randomly, from the file where I have placed my questions and the output will be like the above figure #46. The questions haven't been inspired by me, but from a

book called “10000 general knowledge questions and answers” I found online [37]. I got some really good pieces of advice, from some online tutorials on how to make buttons react among user touch & click, that I have referenced at my code.

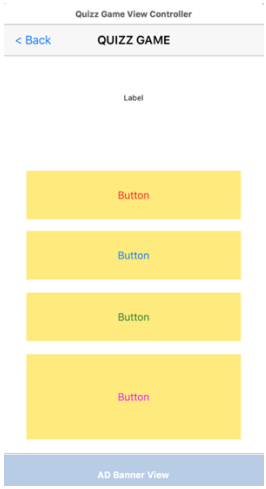


Figure # 47. The Quizzy view controller at my storyboard.

For this task, I created a new view controller in my main storyboard and I added four buttons and one label. I connected the four buttons with Action connections, to my new swift file I created (QuizzGameViewController.swift), in order to store the code, which will run in the background of the app and I connected the label with an Outlet connection in order to show the questions that are stored to my swift file, randomly. After that, I gave an ID to each button (i.e. for the first button I gave the number 0 because it is swift and almost in all programming languages, they start counting from 0). Next, I created a function in order the app to pick a question from my “Questions” array (figure #48).

```
func PickQuestion(){

        if QuestionsGenerator.count > 0 //meaning that if there is sommething inside my "Question" array, ther run!

        {

            //QNumber = 0 that was for questions in a row-static, not random

            QuestionNumber = random() % QuestionsGenerator.count //it will choose between zero and how many
questions I have

            QuestionLabel.text = QuestionsGenerator[QuestionNumber].QuestionSTRING//this is the question we are
on,it will take the question from code and place it to our app


            NumberOfAnswer = QuestionsGenerator[QuestionNumber].AnswerNumber

            for i in 0..<Buttons.count

            {
                Buttons[i].setTitle(QuestionsGenerator[QuestionNumber].AnswersSTRING[i], forState:
UIControlState.Normal)

QuestionsGenerator.removeAtIndex(QuestionNumber)
```

Figure # 48. Pick questions function.

The figure #48 above, shows my main function (func), which selects a random question from the array where my questions are stored in. The “for loop” here is very important here because it guides the program to do something if the buttons are more than zero. Inside the for loop, what this line of code does, is to grab each of the buttons and set a title. I put the [i] inside the Answers array in order to make the program, each time, take all the answers and place them inside the Buttons[i]. the last line of code from the figure above, will remove the questions, the user has answered correctly, so to

avoid repetitions. If I comment out this last line of code from the figure #48 [**QuestionsGenerator.removeAtIndex(QNumber)],** the Quizzy app will never run out of questions, but the same questions will be repeated again and again.

Now, the following figure #49, shows how the questions are structured inside the application. The text, next to "Question:" will be displayed to the label on top of the app which uses an Outlet connection, whereas the "Answers:" array, will display each of its text, at the buttons. These four items inside the array are represented with the numbers 0,1,2,3 respectively. So at the end of each question there is an "Answer:" item. This makes it clear to the compiler which answer is the correct one in order fill the user with more questions.

```
//start writing questions
       //we start cointing from zero so 4 questions...0,1,2,3
        QuestionsGenerator = [CreateQuestion(QuestionSTRING: "Who was the President of the United States in 2015?",
AnswersSTRING: ["Biil Clinton","Barack Obama","Abraham Lincoln","John F.Kennedy"], AnswerNumber: 1),
CreateQuestion(QuestionSTRING: "When Cardiff University was founded?", AnswersSTRING: ["1780","1946","1783","1883"],
AnswerNumber: 3),
CreateQuestion(QuestionSTRING: "Who was the Main founder of Apple Inc?", AnswersSTRING: ["Steve Jobs","Bill Gates","Steve
Wozniak","John Terry"], AnswerNumber: 0),
CreateQuestion(QuestionSTRING: "What is converted into alcohol during brewing?", AnswersSTRING:
["Water","Sugar","Lemon","Coffee"], AnswerNumber: 1),
CreateQuestion(QuestionSTRING: "In which bay is Alcatraz?", AnswersSTRING: ["Greece","Brasil","San Fransisco","New York"],
AnswerNumber: 2),
CreateQuestion(QuestionSTRING: "When was Ferrari founded?", AnswersSTRING: ["1999","1945","1899","1939"], AnswerNumber: 3),
CreateQuestion(QuestionSTRING: "How tall a Giraffe can be?", AnswersSTRING: ["18ft","17ft","20ft","10ft"], AnswerNumber: 0),
CreateQuestion(QuestionSTRING: "What colour is Facebook logo?", AnswersSTRING: ["Pink","Green","Red","Blue"], AnswerNumber:
3),
CreateQuestion(QuestionSTRING: "How Long Do Sea Turtles Live?", AnswersSTRING: ["300 years","30 years","50 years","100
years"], AnswerNumber: 2),
CreateQuestion(QuestionSTRING: "How many eyes does a tiger has?", AnswersSTRING: ["Five","Three","One","Two"], AnswerNumber:
3),
CreateQuestion(QuestionSTRING: "When World War II started?", AnswersSTRING: ["1932","1930","1939","1945"], AnswerNumber: 2),
CreateQuestion(QuestionSTRING: "What is the capital of Greece?", AnswersSTRING:
["Athens","Chalkis","Thessaloniki","Santorini"], AnswerNumber: 0),
CreateQuestion(QuestionSTRING: "What is the weight of an elephant approximately?", AnswersSTRING:
["7.000kg","2.000kg","25.000kg","4.000kg"], AnswerNumber: 0),
CreateQuestion(QuestionSTRING: "How old is Queen Elizabeth in 2016?", AnswersSTRING: ["89","90","93","85"], AnswerNumber:
1),
CreateQuestion(QuestionSTRING: "Who is the creator of Emails?", AnswersSTRING: ["John Applesed","Bill Gates","Ray
Tomlinson","Thanos Gkavalis"], AnswerNumber: 2)]

PickQuestion()
```

Figure # 49. String of questions & Answers.

Because I had four buttons and I did not remember all the answers from my Quizzy app, I created a few NSLog("") reports-alerts (figure #50), which are only visible to the developer via Xcode and not to users, in order complexity and confusion when developing the app, to get eliminated. For example, when I was building this application and I had not added that kind of report, I had every time to run the application to my simulator and try all the possible answers, in order to be sure that everything was working fine.

```
// NSLog("Wrong!")this shows to our *Xcode IDE* a message when the answer is wrong
```

Figure # 50. NSLog developer reports.

As I have previously declared, my application is designed for low budget restaurants, that do not have the financial liberty to supplement their businesses with expensive ordering systems like the one McDonald's uses, but I will refer to them later. The importance of this Quiz game, is divided into two sections. Firstly, because it will raise the interest of the restaurant customers, while they wait for their order and it will prevent them from getting bored, doing something more interesting than refreshing numerous of times their Facebook "Wall" for a remarkable photo or status to come up. Secondly, this Quiz game will provide me, with a small amount of income, in order to pay for the server maintenance, via an advert banner that I have implemented.

I came up with this idea, of adding an advert, because I had to find a way to raise funds for the server maintenance after leaving university. As a student, Cardiff University provides me with a free server hosting, but afterwards I have to pay for that, myself. After iOS 4.0, Apple has created a framework, called "iAd" [38] where developers can use, in order to add adverts to their application.

The iAd, is a platform for advertising, which provides developers with new opportunities to raise their funds, plus promote their applications. The iAd framework, does the necessary work to download adverts from the iAd app network. The only thing that developers should care about is to design the app in a way, to accommodate space for the advert banners.

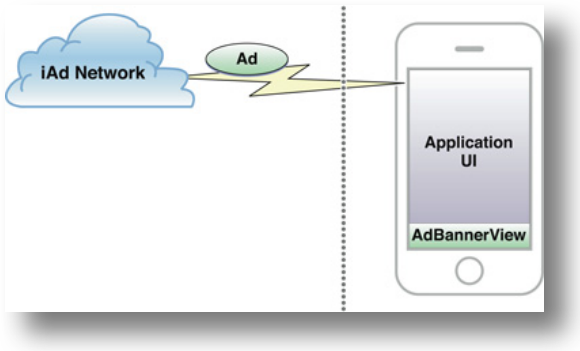


Figure # 51. iAd Network diagram.

In order for developers to be able to add adverts to their app, after all the coding implementations and user interface design, they have to agree with Apple's contract for the iAd network and complete related tax and finance information. After that, developers have to upload their application with the banner embedded in the code and once it has been approved from iAd team, the app will start earning money per advert click.

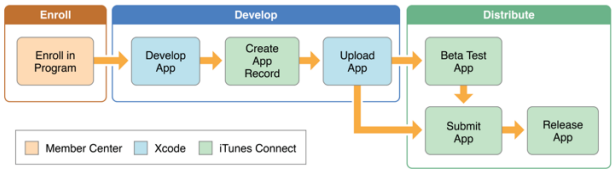


Figure #52. Application Upload to iTunes Connect Procedure

The iAd, gives developers the opportunity to decide the portion of the user interface screen, to display the banner ad. In order to have a banner view advert in the application user interface, we have to use the class called **ADBannerView** whereas if the developer wants to have a full screen ad, he has to use the **ADInterstitialAd** class. A full screen app most of the times is little annoying for users but the chances are, that more users will tap on a full screen advert rather that a small banner on the bottom of the user interface. Despite the fact that full screen banners are more visible to users, I preferred to have a small ad banner in my application just because I can understand that sometimes, adverts, can become very annoying to users and prevent the app from being user friendly. IAd network implementation guide [39] helped me select the right Ad format for a standard banner.

Ad Format	Device	Dimensions
Standard Banner	**iPhone 6 Plus**	414 W x 50 H points (portrait) 736 W x 32 H points (landscape)
	iPhone 6	375 W x 50 H points (portrait) 667 W x 32 H points (landscape)
	iPhone 5 iPhone 5, iPhone 5c, iPhone 5s	320 W x 50 H points (portrait) 568 W x 32 H points (landscape)
	iPhone 4 iPhone 4, iPhone 4s	320 W x 50 H points (portrait) 480 W x 32 H points (landscape)
	iPad iPad Air 2, iPad Air, iPad mini 3, iPad mini 2, iPad mini	768 W x 66 H points (portrait) 1024 W x 66 H points (landscape)

Figure # 53. iAd Network Implementation Guide.

Now, for the implementation of the advert banner, I dragged and dropped from the Xcode object library, the iAd banner view and afterwards I connected it with my code, through an Outlet connection. After that, I imported the iAd and I coded the three basic functions, (they were already embedded in the Xcode, figure #54) which are standard and necessary, every time a developer wants to add an advert to his application.

```
        //functions for the advert baner

    }

    func bannerView(banner: ADBannerView!, didFailToReceiveAdWithError error: NSError!) {
        AdvertBanner.hidden = false
    }


    func bannerViewDidLoadAd(banner: ADBannerView!) {
        AdvertBanner.hidden = false
    }

    func bannerViewActionShouldBegin(banner: ADBannerView!, willLeaveApplication willLeave: Bool) -> Bool {
        return true

    }
```

Figure # 54. iAd banner functions.

2.4 iOS App Development Risks

Before making my proposal to my supervisor, Dr Chorley, and both agreeing to take this application development, as my final year project, I have met with some other teachers, to talk about some final year projects proposals and about their difficulty. All of the projects were quite interesting and the most important was that the language that I needed for almost all of them was python. It was the language I learned from the first year in Computer Science and it was an easy human readable language, that made me like programming from the beginning. All of my teachers and my tutor advised me, that dissertation is not something easy and that I had to choose something that interested me and something beneficial for the society. I always was very curious how all these phone applications were created and I was very interested in creating something which involved user interface, rather than creating something which would run on the terminal. So I came up with the idea of making an ordering application for iOS devices.

The great risk I took for this project was that I had not any previous experience with application development and the most important was the fact that I did not know Swift and Xcode. I also was a windows user all these years and for the purpose of the project I bought a Mac computer, three months before the project deadline because the Xcode IDE is only available to Apple computers. As I said, the risk was enormous for me, because I had to learn how to use Xcode IDE and Swift language in a very limited amount of time, from online resources and not from experienced teachers who can guide me step by step on how to build an application from scratch. The first two months of learning how to use Xcode and Swift, I was continuously filtering everything I saw at online tutorials and I also tried to follow the Apple instructions, from the Apple help for developers' website, because it was the only verified material I found online. Unfortunately, it took me a long time to understand this website (almost five weeks before the dissertation submission), because Apple assumes that its website, is read by experienced developers and it used expressions that were not readable by new developers. Another problem I had with the Swift language, was that it was a very new language and there were not many resources online for new developers.

I do not know if my proposal was a wise choice, for doing something for first time to the most important coursework of my three years at university, but the sure thing is that I "looked at the tree and I overlooked the forest". The only thing that motivated me, was the fact that I was doing something new. Since I was a kid, when the first iPhone was released, I have wanted to create apps and I have been very excited about it. The first month of searching and exploring the iOS application world, I was surrounded by the fear of not handing in anything. In fact, I was discouraged and I suffered severe depression. I realised how difficult it was to do something completely new, without any qualified guidance and the most important, to develop something new completely alone. In the second year of my computer science degree, I had to make a website for movie fans with a group of other seven people. We had a lot of obstacles and difficulties, but firstly the project was about something we were familiar with, from the first year, when we had to make a website for an online shop and secondly we were a team. If someone was unfamiliar with something, the rest of the team helped him move forward and in the end we had a great result. The remarkable fact to mention here is that for that group project, we had two semesters to complete it, whereas for this new project I chose to be my final year project, I only had about three months. This project was not my supervisor's proposal, so to be able to help me, if I had a problem during the development, but it was my proposal and my supervisor had made clear, from our first meeting that he was not familiar with iOS application development and that I had to fend for myself.

Another big risk I had to deal with, was the Apple updates. When the new iOS was released and Xcode asked from me to update it, I had never thought that Apple in its updates also changed Swift code commands. After the update, part of my application wasn't working and until I realised that this was due to the update and fix it, it took me valuable time. For a beginner like me, that was really stressful and also in every obstacle I had, the risk of not overcoming it was ten times higher because I was alone and if part of the code was not working I could not move forward, neither had I someone to turn to for help.

In short, the risk of this project was in an enormous level because I could either develop something or not to hand in anything at all. This was because I decided to make something completely new for me, away from step by step guidance from verified resources, like the University guidance which helped me learn my first programming language, python.

The last risk I took for this project, is that ten days before the dissertation deadline, I entered an application competition and I was chosen to represent Cardiff University in the yearly Santander Bank app competition. I said risk because in parallel with my dissertation and app development, I had to write a business plan of my application, so to be able to enter that competition.

I like taking risks and I do not regret it now, because I strongly believe that the higher the risk is, the highest the outcome, and in my situation the outcome was that I learned a very fast growing language in a short period of time, that will help me build future applications and just within three months of self-learning about application development, I entered for the first time in my life, an application idea-development competition which is organised by Santander Bank in cooperation with Cardiff University, having as my main app idea proposal, my final year project, "Pocket Waiter".

2.5 Competitors

The idea of ordering from your mobile device is not something new. We have previously seen applications like "just eat", which help people select an item from the food menu and at a touch of a button to have it delivered to their door. But still, I have not met a single application which gives the restaurant customers to order from their mobile devices instead of waiting for the waiter to be served via email as its main service. We can find an application close to the one I have developed, if we visit a McDonald's restaurant.

Figure # 55. McDonald's ordering kiosks.

At the entrance of every McDonald's restaurant, there are these large ordering screens (figure #55) which are called "McDonald's ordering kiosks". From there, people can order and pay for their food with the aid of credit card terminals. But a ginormous company like McDonald's, which is worldwide spread can afford for systems like the one I just described. I couldn't find the cost of each kiosk online, so I asked the manager of McDonald's in Cardiff Queen street directly and he told me that each of these screens cost at about 2,500£, but they are profitable because they have replaced almost two workers from each McDonald's restaurant and over a year they "save" almost 15,000£/kiosk, to the company. It is obvious that low budget restaurants cannot afford systems like these, because they are very expensive and also these systems have to be paid by the owner of the restaurant and not by the central authority for a restaurant like McDonald's.

My system is much more friendly and affordable for low budget restaurants, because I have embedded these "kiosks" to my application. In that way, all the costumers have the opportunity to have a "kiosk" embedded in their smart phones if they downloaded the app and the owner of the restaurant will not have to care about installing expensive EPOS systems for the same ordering result.

I strongly believe that restaurants who cannot afford to pay for new technologies, like ordering systems, fail to follow the evolution and as a result they close. This phenomenon is called by economists "creative destruction" [40]. I truly believe that this kind of antagonism, is really unfair and it is something that low budget restaurants struggle to follow. With this application, I give every restaurant owner, the opportunity to evolve and stand among the competition with colosseum companies which use latest technology systems. This is the main reason why I tried and make it cost nothing to restaurant owners. As I have mentioned before, the funds for the server maintenance will be raised by the adverts I have added to this application in the quiz game. The only thing that the restaurant owner is required to have, is an email account, which is provided free, thanks to companies like Google and Yahoo, so that to receive orders.

I am coming from a country, Greece, where the "ordering systems" that most of the restaurants use, is to take the order by hand on a paper. This old-fashioned system, is very time consuming and it is also very outdated. The main reason why this ordering system is so popular in my country, is because restaurant owners, cannot afford to pay for large ordering systems with PDA's and many times in the summer period, I have observed tourists, preferring restaurants with systems like these because they found them more attractive and they do not care that much for the food quality. That is why I insisted on making an application which would try to restore the balance in the restaurant market.

My main strategy was to make it cheap, so in the future to replace the already large and expensive systems. Pocket Waiter, will intergrade into existing EPOS systems where it already exists, to help restaurants engage better with their existing and potential customers. The app will allow customers to view the menu, order from them and many more other features. The reason why I intensely believe that my application is revolutionary, is for the simple reason that it can provide customers and the restaurant managers, with more functionality than the already systems and it will cost nothing to both of them. In my application I have added some other extra features like find restaurants nearby, QR code scanner or a quiz game, because I want to make people like it and use it as their main ordering system, when they visit a restaurant.

If this application be released to the market and attract a giant number of users and reach the target amount of downloads within a short period of time, then the competitor companies will try to eliminate its existence. By competitor companies, I mean all these colosseum software companies, which have created Centralised systems and charge ginormous amounts for "server maintenance". An example for a company like that is EPOSCOMPANY. Epos Company, supplies various PDA equipment to restaurants mainly, for remote order taking. If EposCompany feels that it is extinction is being threatened by my application, it will try in various ways to minimize the "power" of "Pocket Waiter" and try to push my application out of the market.

This application is unique and after a lot of research I did not find any project close to mine, designed to be that cheap. Also, my application combines many technologies that many different "competitor" apps have them as their basic functions.
<u>For example:</u>

- **Toptable**
  This app allows users to book tables online but it is limited only to London.
- **Poynt**
  This app allows users to find restaurant near their current location
- **Square meal**
  This app allows user to find information about restaurants and book tables.

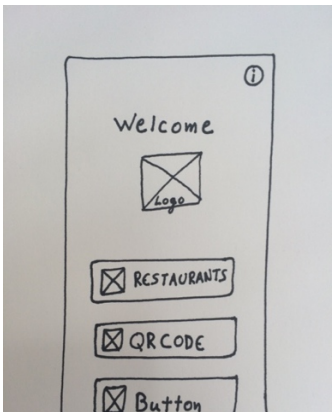
- **Just Eat**
  This app is very close to my app's functionality, but Just eat is only for delivery and also it doesn't provide users with any extra features apart from ordering.

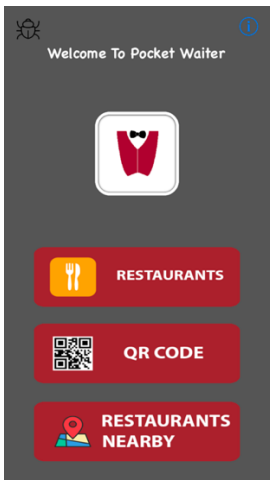
Pocket Waiter will be more than just a comparison or search app for restaurants. Being able to view the menu, find restaurants near you, play a game and order your entire meal using the app, while at the restaurant takes it one step above some of the apps mentioned above

3 The "Specification & Design"

3.1 User Interface

Before starting doing my research about developing an iOS application, firstly, I had to design the UI of the application on a piece of paper, in order to have a general picture of what it had to be look like in the future. After drawing a prototype on a piece of paper, then it was easier for developers to follow the pattern and create the user interface, similar to the handmade prototype. The following figures, show how my application looked like when I made my first attempt, drawing the user interface by hand and how it looks like now, after converting a drawing into a real application, with full functionality.





The Xcode IDE helped me position all the buttons and add the right constraints to each of them, so every time user launches the application from an iOS device, all items on the interface to be symmetric. This section of designing the application and make it look more fancy and user friendly, is in the same level of importance for me with the code that will run in the background. Even if a developer has created an app which provides user, with amazing functionality, if the front-end of the application is outdated or it is very difficult for user to use, then is more likely his app to be used from less and less users and as a result, not to reach the preferable amount of downloads and placed on the Apple store "shelf". Xcode did not provide me with any tools to design the buttons but it allowed me to replace buttons with icons.

3.1.1 Buttons

For the button design, I used Adobe Photoshop CS6. It is a marvellous tool, which provides developers with all the tools they need in order to give that user friendly feeling to their applications. I am one of those developers who emphasise a lot on the design of the application and believe that a small detail can make a huge difference to the app user. For example, tablets are not something new. Bill Gates launched Microsoft's touch input tablet computer a decade before Steve Jobs released the iPad to the market [41]. The reason why iPad achieved such fast popularity, was because of their very user friendly design and because of Apple's great marketing strategy.
This example makes my assumption clear that even if a developer makes an application with perfect functionality, if he does not use a modern and user friendly interface, his application will not attract people's attention and there is a chance, that another developer will upload a similar app to the market, with similar or even less functionality, but with much better UI design, and his application to be preferred by users. In order to achieve a great result and make my app user friendly, I followed the Apple human interface for iOS devices, guidelines [42]. Also, some buttons in my application, like the info button or the bug button, because I wanted to make them look professional, I downloaded these icons from an Apple suggested webpage [43] and I replaced, the buttons Xcode provided me, with them. On that website, I found all the new iOS (iOS 9) updated icons in order to represent some common tasks and types of content like the information button. I insisted on using these built-in icons for my app because users already know what they mean.

3.1.2 App Logo

The logo of the app, is another detail that developers should care about. Every application needs a beautiful app icon. It is not unusual for people to base their opinions about the app quality and reliability, solely on the look of the app icon. It is like the cover page of a book and the more attractive it is to the user, the more the chances are, to download it to his mobile device. It is the first impression a user has, when he sees the app on the Apple store and it is a great chance to persuade him download it, just from the "cover page". I wanted via the logo of the app, to make users understand what the application is about. So, I created a logo to look like a waiter, with a suit and a bow tie. For the design of it, I used a more specific tool, called Adobe illustrator CC. After designing the app icon, in order to add it to the program, I had to resize it six times (figure #56), in order the logo quality to remain the same, in all iOS device sizes. For rendering the size of the different logos, I used Adobe Photoshop again.

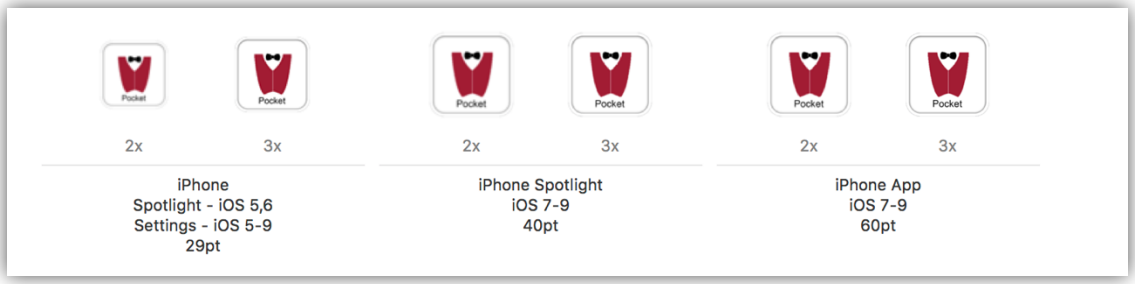


Figure # 56. Different sizes of Pocket Waiter Logo.

In my first attempt in creating the logo for my application, I had not added any label under it, mainly because I had not installed the application to an iOS device to see what it would look like. After installing it to my iPhone, I observed that the whole name did not fit under the application logo.

Figure # 57. First logo VS the updated logo.

Insisting on, that details are those which can attract attention from people, I embedded in the first label "Pocket" inside the app logo, and I named my app "Waiter" (figure #58). In that way, users can see the whole name of the app and also the word "waiter" helps them, find the app faster.

In my updated logo, the reason why I have added black icon borders, is because research has shown that most people prefer to use black backgrounds because it saves them battery, by eliminating LCD light [44] so it will look more elegant on a black background.

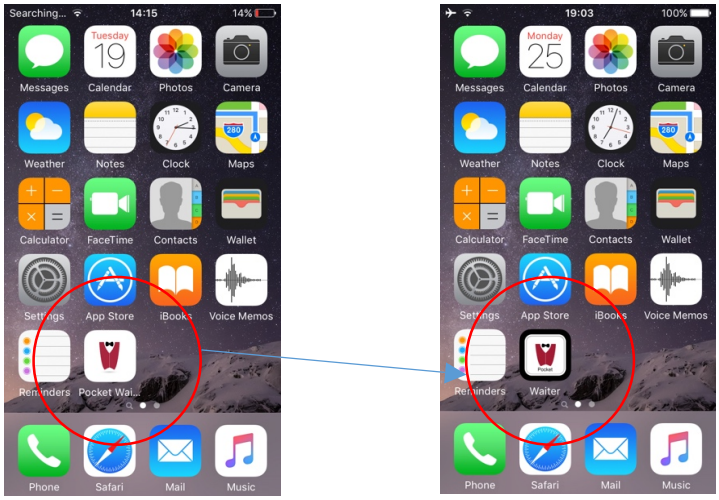


Figure # 58. App Name space differences.

3.2 User Requirements

<u>USER MUST:</u>

- Have an Apple account in order to be able to download the application from Apple App store.
- Have an iOS device like iPhone, with software version bigger than iOS 7.0.

USER SHOULD:

- Have internet connection in order to be able to use most of the application functionalities.
- Have previously added his Facebook account details to his phone settings, under the Facebook icon.
- Have an email account connected with his mobile device so to place orders directly from “Pocket Waiter” app.
- Be sitting to one of the associated restaurants with “Pocket Waiter”.
- Allow the app to use their location services
- Allow the application to access the camera.

3.3 System Design

3.3.1 Table View

This application is designed to fit to all iOS devices. The programming language I used for this project was Swift and the programming platform was Xcode. The main function of this application is to load the menu of the restaurant and place customer orders, via email. For that project I needed two dynamic table views and one detailed view controller. When user clicks on “Restaurants” the first table view appears with all of its data loaded from the server. After user restaurant selection, the second table view appears. This time the user can see all the menu items, followed by their image, their name and their price. Now, if user wants more details about a product, he can simply click on an item and the app will transfer him to a more detail view of the item, with a larger picture and a better description (figure #59).

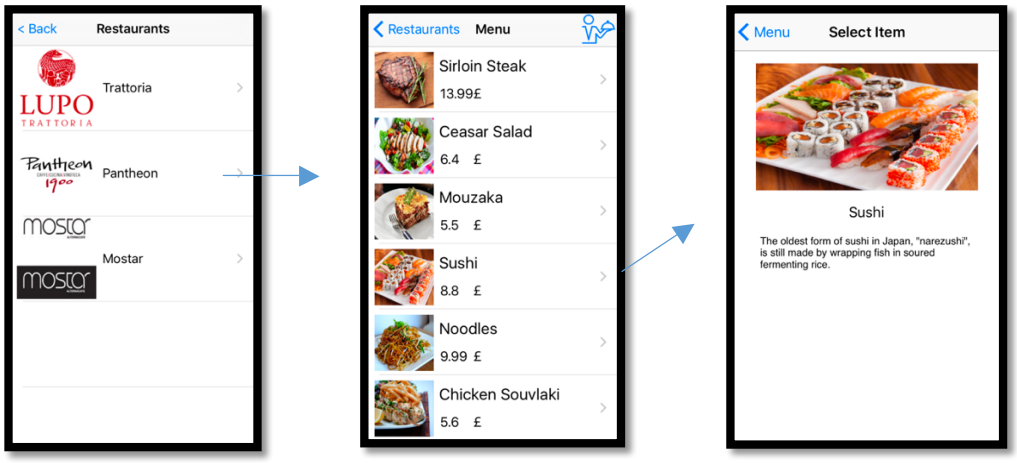


Figure # 59. The Restaurants-->the Menu & the detail view.

All these items and restaurants have been developed in a very dynamic way, in order to be able to retrieve the data from the server and load it to my application. The left image from the figure #60, shows how the menu items table, look like in my storyboard and the right figure shows how the menu table looks dynamically, after loading all these data from the server.

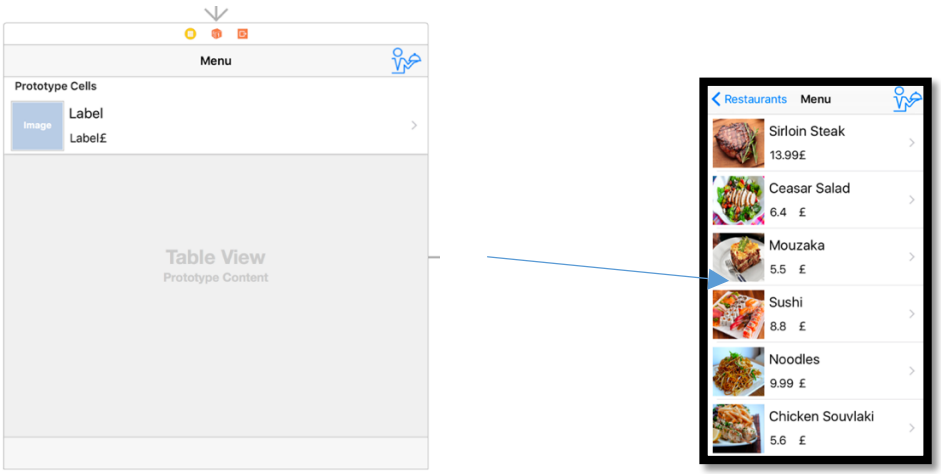


Figure # 60. Dynamic table in storyboard & on device.

After user decides what he wants to order, he can click on the "waiter" button on the top right corner. This button, will load his email account, with the email of the restaurant preinstalled and some prefixed text which reminds users not to forget to select their table number (figure #61). The reason why this application is so unique, is because of that simple function. New generation users, are very experienced with sending and receiving emails and also the inimitable service this system provides is that allows restaurant owners, receive orders from their customers with a system, worldwide known as email, which is provided for free. It also gives the restaurant manager the opportunity to contact their customer if needed and apart from that, the restaurant owner can save the customer's email and inform him about future deals and services, the restaurant provides.

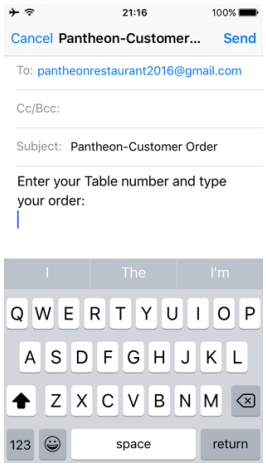


Figure # 61. Email form with restaurant's email prefixed.

The application has been designed especially for low budget restaurants, that cannot afford PDA's, large screens and other expensive systems, in order to place orders for the customers. A system which uses email for exchanging information, is very reliable, especially if the providers of the email account is Google, Yahoo or Microsoft, where all these years have not proven the opposite.

To provide user with best experience and guidance, I have equipped my application with some alerts that will inform user, if something goes wrong within the app. For example, in order the restaurants and the menu items to be loaded from the server, internet connection is required. If user has his Wi-

Fi or mobile data off, then an alert will pop up (figure #62), telling him that in order to be able to view the list of the restaurants, he has to activate the Wi-Fi.

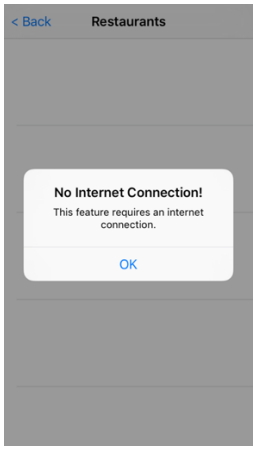


Figure # 62. No Internet Connection Alert.

I have also removed the status bar from the restaurants table view, firstly for design purposes and secondly because Apple suggests that app users should not see unnecessary items within the app. To have the status bar removed, I had to code the following command (figure #63), in each view controller, I wanted to have it removed.

```
//function to hide status bar
    override func prefersStatusBarHidden() -> Bool {
        return true
                                                    }
```

Figure # 63. Algorithm for removing the status bar from an iOS device.

The status bar, is one of these items that Apple embeds in every new view controller automatically, that can only be removed, by adding a small piece of code after the main Class of the file.

3.3.2 QR Code Scanner

This function, provides user with some details of the table he is sitting on, so to place his order successfully. This application, reads all types of QR codes and frames them with a yellow box (figure #64), to help user, locate the QR code on the table (Appendix, 2 QR Code Scan Example).

Figure # 64. QR code reader locates the QR code.

The reason why I prefer the number of the table to be in QR code format rather than a simple number, is because I want customers to use – depend on “Pocket Waiter”. To be more specific here, if a user has visited the restaurant before and knows what he wants to order, if he has previously saved the email of the restaurant then he can process the order directly from his email application without launching “Pocket Waiter”. The only obstacle for him will be the number of the table, and in order to find it, he has to open “Pocket Waiter” app and click on the QR code button.

3.3.3 Restaurants Nearby

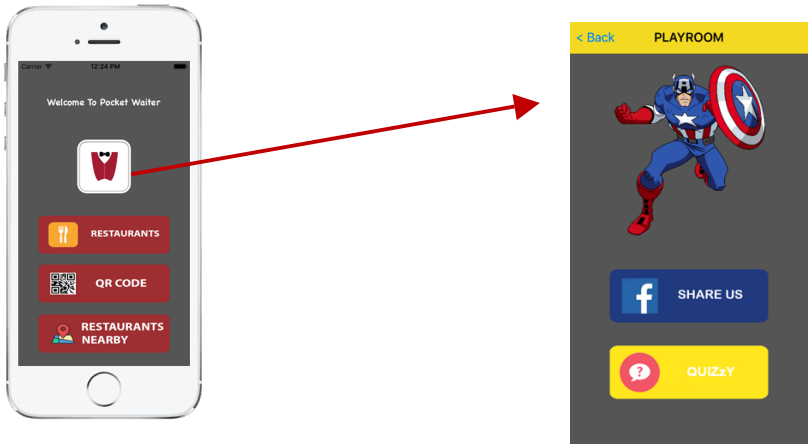
This function, is designed to help users find the location of the restaurants easily, depending on their current position. The user firstly should allow the app to use his current location. User should always be informed when his location is being used within an app. If his location is being used without his permission, then the developer of the app may be confronted with the justice, for illegal location monitoring. In order to avoid having such problems with justice, I added this line of code to my app (figure #65), so the user to be aware when his location is being monitored.

```
self.UserlocationManager.requestAlwaysAuthorization() //use location services only after user approval
```

Figure # 65. Requests authorization from user, for his location monitoring.

3.3.4 Share Us & Quizzy

These two extra functions are located inside the app and they are only visible, if the user taps on the "Pocket Waiter" logo. The main reason why I "hid" them there is because they are less important and according to Apple Human Interface Suggestions, it is not suggested to overload the first page of an app, with second priority functions.



In that view controller, I wanted the user to get a feeling of a friendly and enjoyable environment, where he can relax, while he is waiting to be served. For designing purposes, I have added a comic hero to this page of the app, in order to make it more attractive. If he clicks on the first button, share us, a Facebook form will pop up with the logo of my app embedded in and impossible to be removed. Via this "share us" button, user can upload his post, followed be the "Pocket Waiter" logo. For the ease of the user, instead of requesting him, every time to add his Facebook details, like username and password, I made the app to automatically sign in to his Facebook account via taking his details from the phone Facebook settings. After iOS 8, Apple has given the iOS users the opportunity to post an image, directly from their mobile devices, without having to open the original Facebook app.

On the other hand, the Quizzy, is not just a game which prevents users from getting bored, while they wait for their meal. It is the main source of income of my app. It is designed to display adverts to users and through these adverts to be able to pay for maintaining the application servers. The advert banner is positioned at the bottom of the game, so not to disturb users while playing. Also, its size is very small because it is a banner advert, so not to annoy the app users, while they are playing. The

game is designed to display questions at random order, and the next question is displayed only after the user has correctly answered the previous one. Each question is displayed once, so to avoid repetitions. The questions will be repeated, only if the user has answered all the questions. The design of that game is more like a “tap” game rather than a question game. By the term “tap” I mean that this game does not have a score, so that the user does not have to think too much while he is playing and gets exhausted. It is just a simple quiz game, where the user taps on buttons while he is waiting for his order.

3.3.5 Bug Report

Even the best and most experienced developers have bugs in their applications. After they fix a bug, they release an updated version of the app to the App store and strongly suggest users to download it. Obviously, developers do not release apps with bugs on purpose. The main reason why bugs exist in all applications, is because of the inefficient app testing and the poor user error report feedback. Even Apple company has bugs in its software. Every time it releases a new iOS update, a couple of weeks later, with mathematical accuracy, they will release an updated version of their software, where it will fix some bugs or several security issues. The error reports, Apple receives from costumers, when a new software is being released, is in a much bigger scale than the bug reports from Apple’s software testing team. So, they have customer’s feedback, and they improve their software, faster.

For this purpose, I added this bug report button (figure #66) to my app on the top left of my main app page. When user taps on it, it opens an email form, with my email prefixed and then, user can give me feedback, on how to improve the “Pocket Waiter” app and inform me about errors and bugs that my application has that I had overlooked.

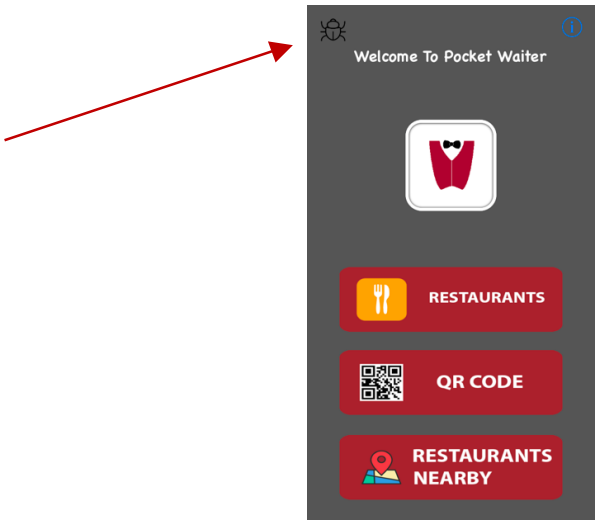


Figure # 66. Bug feedback.

3.3.6 About Us

The About US page, is something that almost all the applications and websites have. It is a page, which give users the opportunity to learn more about the company and the developers, who created that app. At the bottom of each About Us page (figure #67), there is usually the name of the company or the developer who built the app, the year of its production and the name of the app followed by the characteristic trademark symbol, to inform readers that all rights for the app and the name are reserved.

Figure # 67. About us.

3.4 Database Design

For the project purposes, I used an SQL server via phpMyAdmin. For my project, I designed two different tables in my database. The reason why I created two tables instead of one, is to enable me to add more restaurants to my app in the future, dynamically via the database. If I had everything in one table, afterwards it would be extremely difficult for me to insert new restaurants and menu items, because I would have to completely change the structure of my database. With the current design, if I want to add a new restaurant for example, I have to log in my phpMyAdmin, select the restaurants table and create a new restaurant. In my database, I have programmed an error prevention command. To be more specific, if the admin of the database adds a restaurant name and forgets its image, then this new “row” of data will not be imported to the database. This error prevention command, will be helpful to me in the future, when I will have to add new restaurants and items. Also, I have specified the type of each data inside the database tables. The data type (figure #68), is a guideline for SQL to recognize what type of data is expected inside each column and it also identifies how SQL will interact with the stored data. It is helpful again for the admin of the database, because it prevents him from making mistakes with the data types.

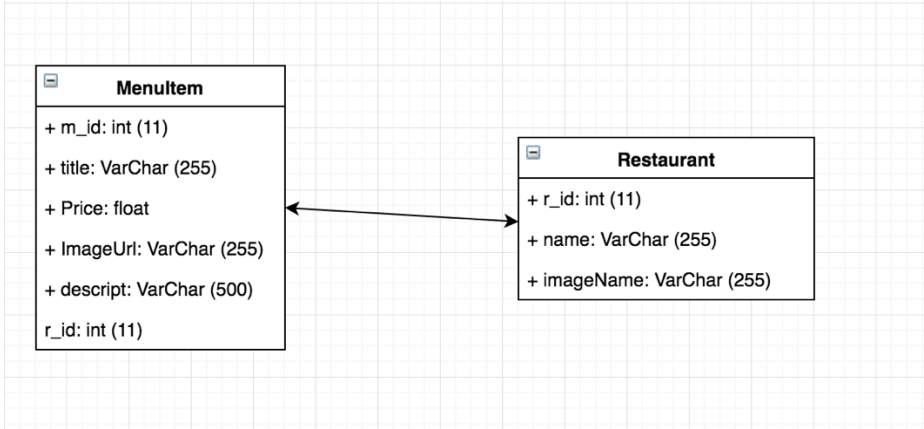
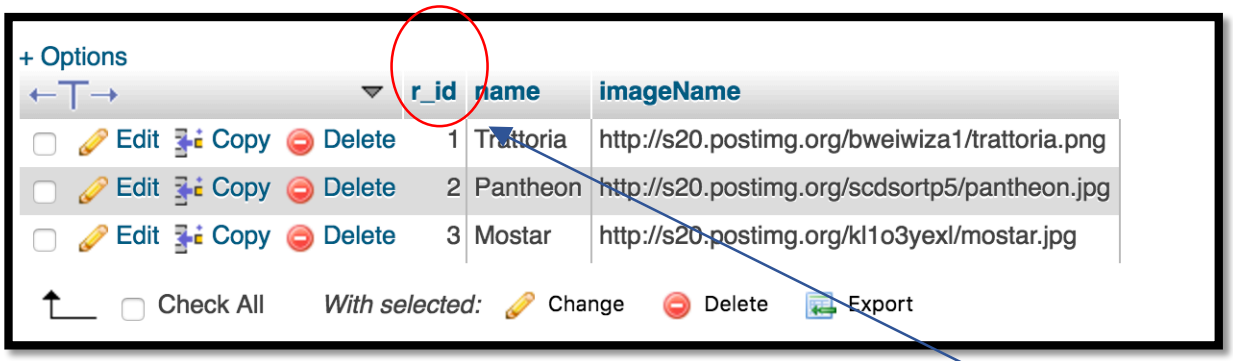


Figure #68. Database ERD (Entity Relationship Diagram)

In order to connect these two tables, I have created an r_id and every time I want a restaurant to have certain items, I have to make sure that in both tables, the restaurant and menu item r_id, is the same.

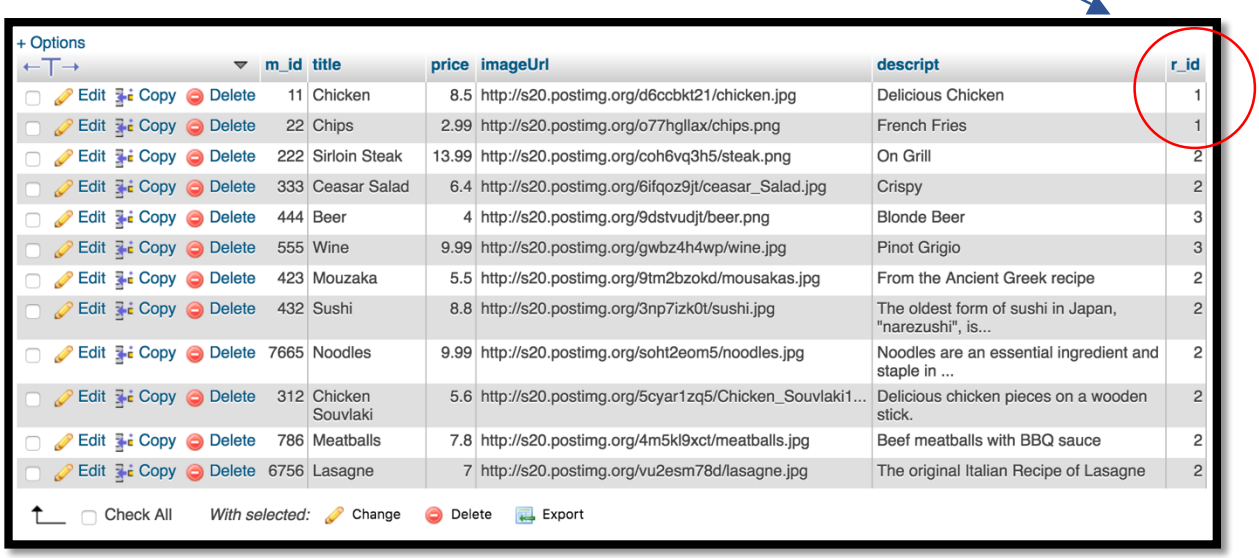


+ Options

	r_id	name	imageName
Edit Copy Delete	1	Trattoria	http://s20.postimg.org/bweiwiza1/trattoria.png
Edit Copy Delete	2	Pantheon	http://s20.postimg.org/scdsortp5/pantheon.jpg
Edit Copy Delete	3	Mostar	http://s20.postimg.org/kl1o3yexl/mostar.jpg

Check All *With selected:* Change Delete Export

Figure # 69. Restaurants Table.



+ Options

	m_id	title	price	imageUrl	descript	r_id
Edit Copy Delete	11	Chicken	8.5	http://s20.postimg.org/d6ccbkt21/chicken.jpg	Delicious Chicken	1
Edit Copy Delete	22	Chips	2.99	http://s20.postimg.org/o77hgllax/chips.png	French Fries	1
Edit Copy Delete	222	Sirloin Steak	13.99	http://s20.postimg.org/coh6vq3h5/steak.png	On Grill	2
Edit Copy Delete	333	Ceasar Salad	6.4	http://s20.postimg.org/6ifqoz9jt/ceasar_Salad.jpg	Crispy	2
Edit Copy Delete	444	Beer	4	http://s20.postimg.org/9dstvudjt/beer.png	Blonde Beer	3
Edit Copy Delete	555	Wine	9.99	http://s20.postimg.org/gwbz4h4wp/wine.jpg	Pinot Grigio	3
Edit Copy Delete	423	Mouzaka	5.5	http://s20.postimg.org/9tm2bzokd/mousakas.jpg	From the Ancient Greek recipe	2
Edit Copy Delete	432	Sushi	8.8	http://s20.postimg.org/3np7izk0t/sushi.jpg	The oldest form of sushi in Japan, "narezushi", is...	2
Edit Copy Delete	7665	Noodles	9.99	http://s20.postimg.org/soht2eom5/noodles.jpg	Noodles are an essential ingredient and staple in ...	2
Edit Copy Delete	312	Chicken Souvlaki	5.6	http://s20.postimg.org/5cyar1zq5/Chicken_Souvlaki1...	Delicious chicken pieces on a wooden stick.	2
Edit Copy Delete	786	Meatballs	7.8	http://s20.postimg.org/4m5kl9xct/meatballs.jpg	Beef meatballs with BBQ sauce	2
Edit Copy Delete	6756	Lasagne	7	http://s20.postimg.org/vu2esm78d/lasagne.jpg	The original Italian Recipe of Lasagne	2

Check All *With selected:* Change Delete Export

Figure # 70. Menu Items Table.

3.5 API Details & Database Components

As I have already mentioned, the database is designed to provide the best usability for the admin of the database. Within a couple of minutes, a restaurant can be added with its menu. The problem with the SQL database that my University provided me with, is that it is not that fast. In order to make it load a little faster, I resized all the icons of the restaurants and items, and converted them to the smallest possible size, so to load faster to the app tables. The images are stored to another database [45] and in my main database, I only had to add their URL. The reason why I uploaded them to another server and not to my main one, is because phpMyAdmin, permitted me to upload images with restricted maximum capacity of 45 KB.

I used the program **draw.io** to create the UML diagram below (figure #71), that shows the interaction between user and app components in a more elegant way. The following figure is the UML diagram I designed, which represents the components in my system and the interaction between them.
This diagram allowed me to represent the communications between the user and the kitchen's chef.

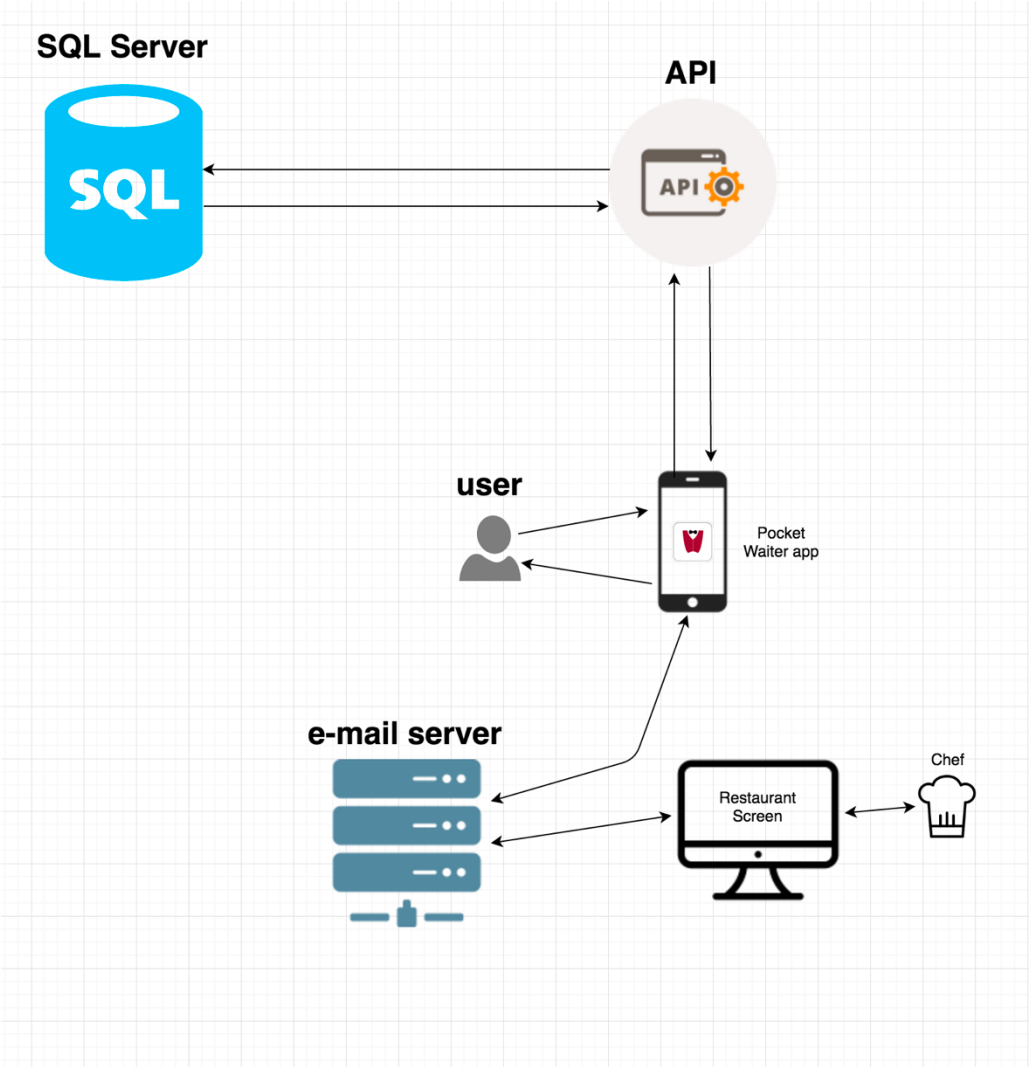


Figure # 71. UML Diagram of the app components & interaction between them.

The diagram from figure #71 shows:

- Interactions with the user and the app UI
- Interactions between the app and the database with the aid of API
- Interactions between the user and the e-mail server
- Interactions between the e-mail server and the chef of the kitchen.

PHP Script Body Design

The main reason why I preferred PDO over MySQLi is mainly for the future evolution of this app. PDO supports twelve different database drivers plus MySQL, whereas MySQLi supports only MySQL. With PDO, if in the future, I want to move to another, faster database, I only have to change the connection string and a few queries, whereas with MySQLi, I would have to rewrite the entire code. Also, according to runtime results, a chart shows [53] that the PDO performance is faster than the MySQLi. Also PDO allows Named parameters like SELECT * FROM RESTAURANT, whereas MySQLi does not allow named parameters. Although MySQLi is easier for newcomers to understand, I insisted on making it in PDO, so afterwards to be able to use it with any database I like.

4 Results & Evaluation

4.1 Testing

Testing was performed in order to make sure that each feature of the application was working properly. There was no formal testing approach while I was building the app in question. During each stage of the application building, each part of the app functionality, was fully tested to make sure that I can move on with the project, without having bugs. This informal app testing during the building process was performed by myself, by running the app to both the Xcode simulator and the iPhone device and fixing everything that was preventing my app from running properly.

The formal testing was performed from third party, after my application had been implemented. To construct a formal testing, I had to use my previous knowledge from Human Computer Interaction subject, from the second year of my academic period. It was a form of user testing. This was accomplished successfully from friends of mine, who had been requested to use my application. Before that, I had given them a test script, with a list of instructions [Appendix, 1 UI Evaluation Testing] they had to follow in order to perform activities on the application. They followed the instructions and afterwards they told me if they agreed or not with the general functionality of the application. However, all of their comments were positive about my application and all of them agreed that the application was very user friendly. From this formal user testing, no problems were identified because I had previously tested and fixed all the bugs in my app.

Example Test Case

Test Case – User: Athanasios Gkavalis			
Step	Procedure	Expected Outcome	Pass/Fail
1	Open the Pocket Waiter Application.	Application opens without crashing.	Pass
2	Click on restaurants button and after view more details about an item.	All the restaurants loaded correctly from the server and when user selects a restaurant, its menu appear and he can have a more detailed view about an item.	Pass
3	Click on the waiter icon, positioned inside the menu, and place an order.	An email form pops-up, with the email of the restaurant preinstalled and after completing the order and clicking send, the order is send via email to the restaurant's email.	Pass
4	Click on QR code button and scan a QR code.	Camera of the iOS device to open and when a QR code is found, a label with its content to appear.	Pass
5	Click on Restaurants nearby button and find the post code of "Pantheon" restaurant.	A map with user current location to load, followed by red pin annotations. When user click on an annotation, he can find more details for the restaurant i.e. its post code	Pass
6	Click on the bug icon and send feedback to developer's email.	An email form pops up with developer's email account preinstalled.	Pass

Figure # 72. Test Case.

4.1.1 Xcode Simulator VS iOS Device Testing

Xcode simulator is a very useful and powerful tool, but it should not be the only way for a developer to test his app. Simulator, is basically an application inside Xcode IDE, which runs on a Mac computer. The Xcode simulator has access to the resources of the computer, including the CPU, memory and network connection. All these resources, most times are probably faster than those found to an iOS device. As a result, the simulator is not a precise test of app's performance and developers should not rely exclusively on it. It is a great tool which helps developers perform hundreds of informal tests during their app implementation. In order the test to provide accurate results, developers should test their app performance on a real iOS device like iPhone. In the simulator the application user interface may run faster and more smoothly than on an Apple's actual mobile device.

My first app tests were performed with the Xcode simulator. It was easier for me to test every small implementation to my app there, instead of every time to run it and test it on my mobile device. The problem with the simulator was that it had some significant hardware, API and OpenGL ES differences from an iOS device. Because of that, I was unable to run some features of my application directly from the simulator, so I had to install the app on my device and test it from there. For example, when I completed the implementation of QR Reader, I wanted to test it on the simulator to ensure that it was working properly. I was continuously getting an error and at first I thought that was a bug within the code. Afterwards, I found out from iOS Developer Library [46], that Audio and Video are not supported from the simulator. Next, I installed the application on my iOS device and I informally test it there with great success.

Although most the functionality of the iOS devices can be simulated in a simulator, some hardware, API and OpenGL ES features, must be tested directly on a device. The reason why developers, prefer simulator for informal testing, is firstly because it is faster than a device and it also produces a "crash logs" report, where developers can locate and fix coding and UI problems, faster. Last but not least, simulator informs developers about the performance of their application while they are running it and

it notifies them, when a function within the application “holds” a bigger amount of Ram memory than usual (figure #73).

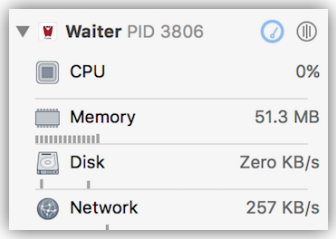


Figure # 73. App performance while running on the simulator.

To sum up, with all these ’tests’ I performed in my application, I realised that developers should never assume that Simulator, mirrors the real-world performance of an application. The final stage test of an application, should always be performed on a real device.

4.1.2 Differences between Simulator and Device testing

Hardware Differences

- Audio and Video, Unsupported
- Motion Support, Unsupported
- Led flashlight, Unsupported

API Differences

- Receiving and sending push notifications, Unsupported
- Handoff support, Unsupported
- Privacy alert for accessing Photos, Calendar and Contacts, Unsupported
- Message UI, Unsupported

OpenGL ES Differences

- Simulator has not a pixel-accurate match to the graphics hardware

4.1.3 How testing is performed

- Via The Xcode Simulator.

  Simulator is designed to assist developers in designing and testing their app, but as I mentioned previously, it should not be the sole platform for testing. To test an application on the simulator is a very easy process. Firstly, you have to select the Xcode project-app you want to run/test, secondly you have to select the device you want your app to be simulated (figure #74), so to give developers an approximate idea of the display and lastly you have to hit “command+b” and the simulator will start compiling/running your app.

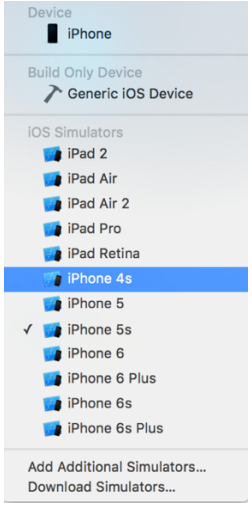


Figure # 74. Different simulators & Device.

- <u>Via The Device.</u>

The application testing on an iOS device can be performed as easily as with the simulator, but the developer has to make some changes in order to be able to test it on his device. Firstly, he has to select the main swift project and select a "Team" (figure #75) for his project. Without a team, Xcode will not allow the application to be installed on another device. This is a security measure Xcode takes, in order to protect developer projects. For being able to run it on my device, I had to request from my University an Apple developers account so to be able to select a "Team".

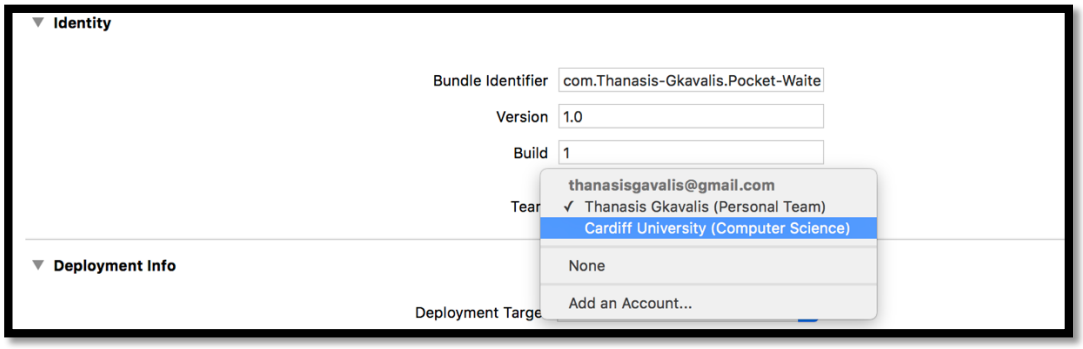


Figure # 75. Project's General Settings.

After selecting Cardiff University team, I went back to Different Simulators and Devices (figure #73) and I selected my mobile device which I had previously connected via USB cable to my Mac computer. With the same process, I hit "command + b" and the application was loaded to my device and then I was able to test it.

4.1.4 TestFlight, the professional way of app testing.

TestFlight is an online service for mobile application testing, owned by Apple Inc. and it is only available to iOS developers. Developers that are signed up with this network, can send their application to internal or external testers, who can send feedback about the application, to developers. Test flight used to support both Android and iOS application testing, but after 2015 [47], Apple Inc. made it available only to iOS developers, who hold a special invitation. Every time a developer tests another developer's application successfully, he gets a special invitation and he can use it afterwards when he wants his application to be tested, from third parties.

Unfortunately, I did not have the benefit of having this professional way of testing of my app. The reason is because this was the first time I had built an application and most specifically an iOS application, and I had not had any invitations from external developers so to have my app tested by them. After receiving an invitation from an external tester, then it will be very easy for someone to publish their app to TestFlight for testing. Firstly, you have to download the TestFlight app from the app store to your phone, and afterwards click on the redeem invitation. Secondly, you have to select the application you want to publish for testing and click on the "send for testing" button. Within a couple of hours, you will receive registered email account feedback from testers to your Apple, on how to improve your app and where your app fails/crashes.

4.1.5 Speed and Runtime Testing

The speed of Pocket Waiter user interface, for retrieving restaurants' data is extremely quick. Although, in the scenario that restaurant and menu items take a while to load, is due to the fact that the "UI" will sometimes have to wait a while for the server to respond. I use an SQL server, for the development of this application, which is provided to me by my University, for free. In the future I have to move my application hosting to a faster responding server, like those which Amazon Inc. provides to provide users with the best experience. Another cause which prevents the application from loading data faster, is the user's poor internet connection. So the response time it depends on the speed of the user's internet connection and the database respond time. In order to make the tables load as fast as possible, all the restaurant and food icons have been rendered from me to the smallest possible size so to help the server to respond faster. To provide users with the best possible feedback, while the restaurant table is loading, I have embedded into the application a loading symbol which shows that processing is taking place.

4.2 SWOT Analysis (Strengths, Weaknesses, Opportunities, Threats)

Strengths

- The application is designed especially for low budget restaurants that can't afford for PDA's or large screens in order to place orders for the customers.
- This application will be on the Apple Store for free, so users can download it and start using it.
- Easy and cheap maintenance of App.

- Easily integrated into existing EPOS systems.
- Branded for individual restaurant.
- All orders are sent via e-mail.

Weaknesses

- Initially only available on iOS operating system, for the moment.
- May reduce the human interaction between restaurant staff (waiter) which some might see as part of the eating out experience.

Opportunities

- Experience from this will provide a platform for future products.
- An opportunity to develop a database of restaurant businesses for associate marketing.
- An opportunity to start a company off the back of this product.
- A great opportunity afterwards will be, adverts to target a selected audience and depending user preferences, a relevant advert to be displayed.
- Another great opportunity for the evolution of this application will be the introduction of bitcoins as a method of payment. Until now, bitcoins have not proven to be a stable and safe currency but in the future, a combination of these two, will bring people closer to Decentralisation.
- Possible commercialisation routes (licensing, merging or sale to existing firms).

Threats

- Slow uptake of product as a result of being a new and untested product.
- Fear of change for those already using existing similar product, including EPOS products.
- Threat from established companies.

4.3 Products, Services and Benefits

4.3.1 Products and Services

'Pocket Waiter' is an iOS-based Mobile Application aimed at the restaurant market/sector. While it is aimed at this sector, emphasis is on small to medium sized restaurants with limited resources and budgetary restrictions.

Pocket Waiter will integrate into existing EPOS (Electronic Point of Scale) systems, where they already exist or act as a stand-alone tool to help restaurants engage better with their existing and potential customers.

Other features provided by the Pocket Waiter include:

- It can act as a stand-alone product or integrated into existing EPOS system
- Provides full menu with information
- Social Media integration
- Provide restaurant location visible on map.
- Take full control over your App content
- App matched with restaurant branding
- QR reader
- Quiz game – add small banner adverts

4.3.2 Benefits

- Boosts business especially from existing customers.
- Increase your orders, visitors and ultimately revenue.
- It is anticipated that the Mobile Application will continue to be updated to meet new demands and keep up with market sector changes and trends. This will mainly be done by constantly reviewing restaurant and customer usage and feedback. This will allow me to extend the shelf life of the product.
- While there is no environmental legislation applicable to the product, I am confident the product will reduce the amount of paper products used by restaurants in the process of order taking as it will predominantly be done electronically.

5 Future Work

My project was completed, unexpectedly well with great success. I have completed my main project plan idea, I have managed to do my two extra functions of the QR code reader and the Restaurants nearby and I also added two more additional features, so to make my app unique and irreplaceable.

If this application enters the market and people start using it, I have to add some additional features so to stand among the competition. In the future, I want to give "Pocket Waiter" users, the ability to pay directly via this application and receive a confirmation of their deposit. Also, I want to enable restaurant owners to send push notifications and messages to costumers about the processing time of their order plus alerts about special offers and discounts.

Furthermore, in the future I want to do research among all the Server providers and move my database to the fastest and the most reliable server, so as to eliminate customer waiting time for the restaurants to load from the server. Unfortunately, I run out of time and I did not implement to my application, a search bar. I have created a different application with a search bar which allows users to view results as they are typing but I did not have time to implement it into my main application.

Last but not least, in the future, I want to prevent "spam" and fake orders which will annoy and confuse the restaurants with its orders. To prevent that, in the future I will change the structure of the application so the customers to be able to order from the selected restaurant only if they have connected to its WIFI network. To be more specific, I have to register the SSID and the mac address of the restaurant's router, inside the application via the CaptiveNetwork interface [54] so to make the menu of the restaurant load, only if the user is connected to the restaurant's network. That, will save valuable time from restaurants, who will receive fake orders from people who are not at the restaurant. For the same reason, in the future I want to hide restaurant's email from users, so not to be able to send an email order from their email app, without Pocket Waiter app intermediary.

To sum up, from my point of view I strongly believe that the development of each application does not have an end point. New technologies and software updates are released at very rapid rhythms, so developers have to continuously learn and follow these changes, in order to keep their apps, updated and able to follow these new technologies.

6 Conclusions

The most interesting conclusion for me is the fact that I learned by myself, without any structured guidance and within a very limited time, how to build an iOS application, using Swift language and I ended up representing Cardiff University, in the yearly Santander Universities Entrepreneurship Awards 2016 competition, for New technologies and Mobile Applications. This project has also broadened my horizons from a business aspect. I learned how to build a business plan properly, why is that important and I got a general idea of the whole cycle of an official app building, from the background searching to the implementation and afterwards to the app release on the market.

One of the major difficulties of this project was the connection with the database and the dynamically data loading from the server. I realised how difficult it is to deal with new problems when you work on a project alone but I also found out how to cope with these stressful situations. It is true, that online you can find everything you are interested in but the problem is that you have to filter many times your result outputs, because the world wide web contains a lot of untrusted and fake resources.

I truly believe that this application can give low budget restaurants, who cannot afford EPOS systems, the opportunity to follow the evolution of technology and introduce this new way of ordering system to their restaurants and get benefited from the multitude of opportunities, Pocket Waiter will provide them. The reason why this application is revolutionary and unique, is because it combines all the already existing systems and application, like online reservation apps or menu catalogue apps. Apart from that, because the main structure of it is designed to be at the lowest possible cost, it can be afforded by anyone.

This project helped me with the expansion of my research and technical skills and besides it made me cope with stressful situations in a much better way than before. Dissertation is the most important coursework for a University student and more or less, the overall grade and future career of a student, depends on it. My decision of creating something new for my dissertation, without any background experience and academic guidance, was very risky because I could have ended up being unable to build something and as a result I could have taken a very low mark or even ended up with a failure. Despite all the difficulties I have encountered in the course of the project and the sleepless nights fretting over the task in question, I feel very lucky that I had the opportunity to live that experience, because it made me reach my limits and realise that the difficulty of something, depends on how extensively one is willing to search and how hard one is willing to work, most of the time under great pressure, to accomplish it. This project has been a real proof for me, that everything is achievable if we really need something and the higher the risk is, the higher the outcome. I insist on the last phrase, because despite all these difficulties I faced, in the end, I had a great result with a prototype app working properly and I got the opportunity to enter a real world mobile application competition.

7 Reflection and Learning

Firstly, this project allowed me, to expand my technical and searching skills and also helped deal with stressful situations under the pressure of time. It is really remarkable the fact that you can actually build an iOS application without any professional guidance, just with the aid of the internet. For this project I used two languages. I used SWIFT, for building the application and adding all the functions on it, and PHP in order to make my application connect with the server. PHP is a language that I have

met in my first year when I had to make a website but this time it was much harder because I had to connect my iOS application using this language and the difficulty with iOS applications is that after every update, you have to update big part of your code including my PHP request script. The big obstacle I faced when I was developing the application, was the fact that I didn't have the flexibility to choose a different programming language rather than SWIFT, which is a programming language that I had to learn, within a short period of time from online recourses, where most of the times the online content in tutorials was outdated and incorrect. The skills that I have acquired from this self-learning, made me more confident with myself and made me realise that everything is achievable if done under a lot of pressure and accompanied by real willingness for the project accomplishment.

I have also learned to use Xcode IDE and I have a clear picture now, of what each item on this development environment is referred to and also I have understood how to choose the best technical tools for iOS applications. What I mean here is that I learned a majority of available libraries in Xcode and I realized how to use them and why developers are 'addicted' to them. The simple reason is because libraries include all the required functionalities a developer wants, to complete his project faster and more accurate. Now after the completion of this application, I feed a better computer scientist, who did something useful for the real-world, away from the "University Guidance". All this exposure to new technical material, gave me a lot of self-esteem which will hopefully help me with my future career.

Moreover, despite the fact that I had a little experience with photo editing I had to learn the Adobe Photoshop CS6 deeper, in order to design my application buttons, to meet with Apple Human interface suggestions and also to provide users with best experience. When you make a mobile application, apart from the code which will run in the background, the developer has to be very careful with the front-end design because even if the design of the background code is perfect, if the application isn't user friendly, then it's worthless. This project made me realise the difficulty that developers face in their effort to convert the hard coded background to simple buttons. After developing my first application, every time I launch an application I appreciate each function of it and I continuously think and send feedback on how the developer could have improved his application.

Furthermore, this project made me manage better myself, deal with stressful situations and managing to finish before the deadline. I have been involved in many group projects in the past, during my first and second year in university, but I had never met such an enormous amount of responsibilities and research, that had to be done by myself, in a very limited period of time. I knew from the beginning of this project that my supervisor isn't allowed to help me with my project and that made me a better "fighter", but Dr Chorley has provided me with very useful advices on how the application would be more functional to the users. To be more specific, in the beginning of this project, I wanted to insert all the restaurants and the food items, in a static way but he strongly advised me to make it in a more dynamic way and make them load from an online database, because it would not be so user friendly, if every time I wanted to insert a new restaurant to my app, I had to release a new update, instead of doing it directly from the server.

Last, via this project I learned how to make a business plan (Appendix, 3 Business Plan) and I realised why is so important for the business market. I have never heard before the term business plan and I wasn't aware of its main purpose. After taking part and being chosen to represent Cardiff University, in the Santander bank Mobile Application Awards 2016, I was asked to provide the bank, with my proposal embedded in a business plan. This business plan, helped me understand how apps enter the real world and how developers request funding from banks or other establishments, so to be able to start developing a project.

8 Glossary & Referencing

8.1 Glossary

API An access point for an app, that can be remotely called. It stands for Application Programming Interface

Baas Provider Backend as a service. It falls under cloud computing category and makes easier for developers to setup, use and operate a cloud backend for their apps.

EPOS The electronic point of sale. It is the time and place where a retail transaction is completed. Under this category, there are all these ordering systems with PDA's and monitors which help with the more efficient management of a company.

FTP Stands for File Transfer Protocol.

IDE An Integrated Development Environment, is a software application, like Xcode, which provide developers with tools necessary for software development.

iOS A software for mobile devices, created from Apple Inc. for all its devices.

JSON A famous object interchange format. JSON stands for JavaScript Object Notation.

Latitude It is the amount of north-to-south distance to display the map.

Longitude It is the amount of east-to-west distance to display the map.

NSObject A universal base class for all Cocoa Touch classes in both Swift and Objective-c.

PHP A server side scripting language. it stands for Personal Home Page.

QR Code A type of matrix barcode (2D) for data storing and fast readability. It stands for Quick Response Code.

UI It is the user interface. It is basically the space where interactions between humans and machines, occur.

8.2 References

[24] @twostraws, P. (2016). *How to scan a QR code – Swift 2 example code*. [online] Hackingwithswift.com. Available at: https://www.hackingwithswift.com/example-code/media/how-to-scan-a-qr-code [Accessed 4 May 2016].

[44] AndroidPIT. (2015). *How black wallpaper can save your Android battery - AndroidPIT*. [online] Available at: https://www.androidpit.com/how-black-wallpaper-can-save-your-battery [Accessed 4 May 2016].

[4] Anon, (2016). [online] Available at: http://www.tutorialspoint.com/swift/swift_tutorial.pdf [Accessed 4 May 2016].

[37] Anon, (2016). [online] Available at: http://www.keloo.ro/doc/10000_intrebari.pdf [Accessed 4 May 2016].

[39] Anon, (2016). [online] Available at: https://developer.apple.com/iad/monetize/Implementing-iAd-in-Your-iOS-Apps.pdf [Accessed 4 May 2016].

[55] Anon, (2016). [online] Available at: http://www.deloitte.com/view/en_GB/uk/industries/thl/43ef03869ef4b310VgnVCM2000003356f70aRCRD.htm [Accessed 4 May 2016].

[3] Anon, (2016). *Swift Book PDF*. [online] Available at: http://carlosicaza.com/swiftbooks/SwiftLanguage.pdf [Accessed 4 May 2016].

[27] Appcoda.com. (2014). *Building a QR Code Reader in Swift*. [online] Available at: http://www.appcoda.com/qr-code-reader-swift/ [Accessed 4 May 2016].

[0] BigHospitality.co.uk. (2016). *Value of restaurant market to reach £52bn by 2017*. [online] Available at: http://www.bighospitality.co.uk/Trends-Reports/Value-of-restaurant-market-to-reach-52bn-by-2017 [Accessed 4 May 2016].

[41] Bort, J. (2016). *Microsoft Invented A Tablet A Decade Before Apple And Totally Blew It*. [online] Business Insider. Available at: http://www.businessinsider.com/heres-visual-proof-of-just-how-badly-microsoft-blew-it-with-tablets-2013-5?IR=T [Accessed 4 May 2016].

[14] Code School Forum. (2016). *App Evolution with Swift 5.3 - prepareForSegue...?*. [online] Available at: https://www.codeschool.com/discuss/t/app-evolution-with-swift-5-3-prepareforsegue/25182 [Accessed 4 May 2016].

[8] Developer.apple.com. (2016). *About Connecting Objects to Code*. [online] Available at: https://developer.apple.com/library/ios/recipes/xcode_help-IB_connections/chapters/AboutConnectingObjectstoCode.html#//apple_ref/doc/uid/TP40014227-CH42 [Accessed 4 May 2016].

[38] Developer.apple.com. (2016). *About iAd*. [online] Available at: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/iAd_Guide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40009881-CH1-SW1 [Accessed 4 May 2016].

[22] Developer.apple.com. (2016). *AVCaptureMetadataOutputObjectsDelegate Protocol Reference*. [online] Available at: https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVCaptureMetadataOutputObjectsDelegate_Protocol/ [Accessed 4 May 2016].

[54] Developer.apple.com. (2016). *CaptiveNetwork Reference*. [online] Available at: https://developer.apple.com/library/ios/documentation/SystemConfiguration/Reference/CaptiveNetworkRef/index.html [Accessed 4 May 2016].

[6] Developer.apple.com. (2016). *Configuring Your Xcode Project for Distribution*. [online] Available at: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/ConfiguringYourApp/ConfiguringYourApp.html [Accessed 4 May 2016].

[30] Developer.apple.com. (2016). *Core Location Constants Reference*. [online] Available at: https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocationConstantsRef/#//apple_ref/doc/constant_group/Accuracy_Constants [Accessed 4 May 2016].

[33] Developer.apple.com. (2016). *Displaying Maps*. [online] Available at: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/MapKit/MapKit.html#//apple_ref/doc/uid/TP40009497-CH3-SW1 [Accessed 4 May 2016].

[5] Developer.apple.com. (2016). *iOS Human Interface Guidelines: Designing for iOS*. [online] Available at: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html?utm_source=twitterfeed&utm_medium=twitter [Accessed 4 May 2016].

[42] Developer.apple.com. (2016). *iOS Human Interface Guidelines: Designing for iOS*. [online] Available at: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html?utm_source=twitterfeed&utm_medium=twitter [Accessed 4 May 2016].

[28] Developer.apple.com. (2016). *main.m*. [online] Available at: https://developer.apple.com/library/ios/samplecode/CurrentAddress/Listings/main_m.html#//apple_ref/doc/uid/DTS40009469-main_m-DontLinkElementID_11 [Accessed 4 May 2016].

[31] Developer.apple.com. (2016). *MapKit Data Types Reference*. [online] Available at: https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKitDataTypesReference/#//apple_ref/c/tdef/MKCoordinateSpan [Accessed 4 May 2016].

[36] Developer.apple.com. (2016). *SLComposeViewController Class Reference*. [online] Available at: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Reference/SLComposeViewController_Class/#//apple_ref/doc/uid/TP40012205-CH1-SW4 [Accessed 4 May 2016].

[7] Developer.apple.com. (2016). *Start Developing iOS Apps (Swift): Create a Table View*. [online] Available at: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/Lesson7.html [Accessed 4 May 2016].

[10] Developer.apple.com. (2016). *Start Developing iOS Apps (Swift): Create a Table View*. [online] Available at: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/Lesson7.html [Accessed 4 May 2016].

[18] Developer.apple.com. (2016). *Start Developing iOS Apps (Swift): Implement Navigation*. [online] Available at: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/Lesson8.html [Accessed 4 May 2016].

[46] Developer.apple.com. (2016). *Testing and Debugging in Simulator*. [online] Available at: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/TestingontheiOSSimulator/TestingontheiOSSimulator.html#//apple_ref/doc/uid/TP40012848-CH4-SW1 [Accessed 4 May 2016].

[50] Docs.cs.cf.ac.uk. (2016). *Accessing Your University Files Remotely from Windows*. [online] Available at: https://docs.cs.cf.ac.uk/notes/accessing-your-university-files-remotely-from-windows-and-vista/ [Accessed 4 May 2016].

[40] Economics.mit.edu. (2016). [online] Available at: http://economics.mit.edu/files/1785 [Accessed 4 May 2016].

[2] Facts, M. (2016). *Topic: Mobile App Usage*. [online] www.statista.com. Available at: http://www.statista.com/topics/1002/mobile-app-usage/ [Accessed 4 May 2016].

[51] GitHub. (2016). *Alamofire/Alamofire*. [online] Available at: https://github.com/Alamofire/Alamofire [Accessed 4 May 2016].

[52] GitHub. (2016). *SwiftyJSON/SwiftyJSON*. [online] Available at: https://github.com/SwiftyJSON/SwiftyJSON [Accessed 4 May 2016].

[21] GitHub. (2016). *yannickl/QRCodeReader.swift*. [online] Available at: https://github.com/yannickl/QRCodeReader.swift [Accessed 4 May 2016].

[48] Hosting, S. (2016). *How to Create Database with PhpMyAdmin Tutorial*. [online] Siteground.co.uk. Available at: https://www.siteground.co.uk/tutorials/phpmyadmin/phpmyadmin_create_database.htm [Accessed 4 May 2016].

[43] Icons8.com. (2016). [online] Available at: https://icons8.com/free-ios-7-icons-in-vector/#/win [Accessed 4 May 2016].

[16] Jacobs, B. (2015). *iOS From Scratch With Swift: Table View Basics*. [online] Code Envato Tuts+. Available at: http://code.tutsplus.com/tutorials/ios-from-scratch-with-swift-table-view-basics--cms-25160 [Accessed 4 May 2016].

[32] Karger, H. (2016). *Geo Tag Generator*. [online] Geo-tag.de. Available at: http://www.geo-tag.de/generator/en.html [Accessed 4 May 2016].

[15] Leist, J. (2014). *iOS Swift Tutorial - passing data between ViewControllers - James Leist*. [online] James Leist. Available at: http://jamesleist.com/ios-swift-passing-data-between-viewcontrollers/ [Accessed 4 May 2016].

[23] Liu, F. (2016). *SwiftQRCode on CocoaPods.org*. [online] Cocoapods.org. Available at: http://cocoapods.org/pods/SwiftQRCode [Accessed 4 May 2016].

[11] Lynda.com - A LinkedIn Company. (2016). *Create a basic table view and a data source iOS App Development*. [online] Available at: http://www.lynda.com/Swift-tutorials/Creating-basic-table-view-data-source/185036/362308-4.html? [Accessed 4 May 2016].

[35] Noyes, A. and Noyes, A. (2016). *Top 20 Facebook Statistics - Updated March 2016*. [online] Zephoria Inc. Available at: https://zephoria.com/top-15-valuable-facebook-statistics/ [Accessed 4 May 2016].

[9] Parse.com. (2016). *Parse*. [online] Available at: https://parse.com/about [Accessed 4 May 2016].

[1] Perez, S. (2016). *Consumers Spend 85% Of Time On Smartphones In Apps, But Only 5 Apps See Heavy Use*. [online] TechCrunch. Available at: http://techcrunch.com/2015/06/22/consumers-spend-85-of-time-on-smartphones-in-apps-but-only-5-apps-see-heavy-use/ [Accessed 4 May 2016].

[45] Postimage.org. (2016). *Postimage.org - free image hosting / image upload*. [online] Available at: http://postimage.org/ [Accessed 4 May 2016].

[53] Potop, R. (2016). *PDO vs MySQLi performance comparison*. [online] Wooptoo.com. Available at: http://wooptoo.com/blog/pdo-vs-mysqli-performance-comparison/ [Accessed 4 May 2016].

[25] Shrikar Archak. (2015). *AVFoundation : Implementing Barcode Scanning in Swift*. [online] Available at: http://shrikar.com/implementing-barcode-scanning-in-ios8-with-swift/ [Accessed 4 May 2016].

[12] Swift, H. (2016). *How to load specific image from assets with Swift*. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/29356574/how-to-load-specific-image-from-assets-with-swift [Accessed 4 May 2016].

[13] Swift, P. (2016). *Prepare for Segue in Swift*. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/24040692/prepare-for-segue-in-swift [Accessed 4 May 2016].

[26] Team, C. (2016). *CocoaPods.org*. [online] Cocoapods.org. Available at: https://cocoapods.org/ [Accessed 4 May 2016].

[20] Techotopia.com. (2016). *An Example Swift iOS 8 iPhone Camera Application - Techotopia*. [online] Available at: http://www.techotopia.com/index.php/An_Example_Swift_iOS_8_iPhone_Camera_Application [Accessed 4 May 2016].

[49] W3schools.com. (2016). *PHP Connect to MySQL*. [online] Available at: http://www.w3schools.com/php/php_mysql_connect.asp [Accessed 4 May 2016].

[19] Whatisaqrcode.co.uk. (2016). *What is a QR Code?*. [online] Available at: http://www.whatisaqrcode.co.uk/ [Accessed 4 May 2016].

[34] Wikipedia. (2016). *Mercator projection*. [online] Available at: https://en.wikipedia.org/wiki/Mercator_projection [Accessed 4 May 2016].

[47] Wikipedia. (2016). *TestFlight*. [online] Available at: https://en.wikipedia.org/wiki/TestFlight [Accessed 4 May 2016].

9 Appendix

9.1 UI Evaluation Testing

General Questionnaire

Pocket Waiter Usability Questionnaire

1. I like the design of the "Pocket waiter" and it is very user friendly.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

2. The navigation inside the app was easy.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

3. I immediately understood the function of each button.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

4. The buttons were well organised and easy to find.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

5. I found easily the "Playroom" inside the app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

6. I found the Pocket Waiter app very complex to use.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

7. I have completely understood after one use, how the app works.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

8. I successfully placed an order via this app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

9. In comparison with other apps I have used in the past, I found Pocket Waiter to meet all the quality criteria.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ Agree

Name:
Surname:
Student Number:

Signature

User #1

Pocket Waiter Usability Questionnaire

1. I like the design of the "Pocket waiter" and it is very user friendly.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☒, 8☐, 9☐, 10☐ **Agree**

2. The navigation inside the app was easy.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☒, 10☐ **Agree**

3. I immediately understood the function of each button.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☒, 8☐, 9☐, 10☐ **Agree**

4. The buttons were well organised and easy to find.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☒ **Agree**

5. I found easily the "Playroom" inside the app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☒ **Agree**

6. I found the Pocket Waiter app very complex to use.
   Disagree 1☒, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ **Agree**

7. I have completely understood after one use, how the app works.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☒ **Agree**

8. I successfully placed an order via this app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☒ **Agree**

9. In comparison with other apps I have used in the past, I found Pocket Waiter to meet all the quality criteria.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☒ **Agree**

Name: Dimitri Georgakis
Surname:
Student Number: C1331994

Signature

User #2

Pocket Waiter Usability Questionnaire

1. I like the design of the "Pocket waiter" and it is very user friendly.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

2. The navigation inside the app was easy.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

3. I immediately understood the function of each button.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

4. The buttons were well organised and easy to find.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

5. I found easily the "Playroom" inside the app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

6. I found the Pocket Waiter app very complex to use.
   Disagree 1☑, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ **Agree**

7. I have completely understood after one use, how the app works.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

8. I successfully placed an order via this app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

9. In comparison with other apps I have used in the past, I found Pocket Waiter to meet all the quality criteria.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

Name: ANNI
Surname: PANASETI
Student Number: 1324546

Signature

User #3

Pocket Waiter Usability Questionnaire

1. I like the design of the "Pocket waiter" and it is very user friendly.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

2. The navigation inside the app was easy.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☑, 10☐ **Agree**

3. I immediately understood the function of each button.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☑, 9☐, 10☐ **Agree**

4. The buttons were well organised and easy to find.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

5. I found easily the "Playroom" inside the app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☑, 8☐, 9☐, 10☐ **Agree**

6. I found the Pocket Waiter app very complex to use.
   Disagree 1☑, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☐ **Agree**

7. I have completely understood after one use, how the app works.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

8. I successfully placed an order via this app.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

9. In comparison with other apps I have used in the past, I found Pocket Waiter to meet all the quality criteria.
   Disagree 1☐, 2☐, 3☐, 4☐, 5☐, 6☐, 7☐, 8☐, 9☐, 10☑ **Agree**

Name: Neofytos
Surname: François
Student Number: 007

Signature

9.2 QR Code Scan example:

9.3 Business Plan.

Introduction

Pocket Waiter is an iOS-based Mobile Application, which will provide "ordering" services to restaurants at an affordable or minimal cost. The Application uses email services, which are provided for free from providers like Google or Yahoo, to process orders. In that way, restaurant owners with limited funds can have the same ordering system standard used by bigger food outlets like McDonalds, at no cost. The Pocket Waiter Application will reduce the time waiters invest in 'waiting' allowing their time to be invested in other areas and improve productivity

This application, will load all the restaurants dynamically from the server and upon user restaurant selection, the respectively menu will appear. After customer decides what he wants to order, he can then place the order.

The procedure of placing the order and sending it to the kitchen is being designed to be free for both. After user sees the main menu, he can then click on the "waiter" icon-button which is positioned on the top left of the menu of each restaurant and a special form will come up, requesting from users to type their order and their table number. In the form, the email of the company will be prefixed.

This new way of ordering, we aim in the future to replace the already expensive systems which require from restaurant owners very high maintenance fees.

There are applications on the market who offer ordering services, but yet none of them is using email for processing of the order. The reason is because large software companies promote centralised systems on the market and they try to sell as more equipment as possible, like PDA's, so restaurant holders to depend exclusively on them.

My application will fall under the freemium applications. These type of applications are much more downloadable and it is the only way to persuade someone to replace his already ordering system with ours. It will be provided for free to both, application users and restaurant owners. If restaurant managers want this application to provide "more" to customers, I will give them the opportunity to upgrade from the basic plan to premium with upon requesting a "friendly" fee.

To sum up, this application will look like a "Swiss knife" so to be preferred among others. I compared "Pocket Waiter" with a Swiss knife, because I don't want just to be a knife but a multifunctional app. I will add extra features like restaurants nearby, QR Reader, social media integration, game and much more, so the user to have everything he needs in just one application.

<u>SWOT Analysis (Strengths, Weaknesses, Opportunities, Threats)</u>

Strengths

- The application is designed especially for low budget restaurants that can't afford for PDA's or large screens in order to place orders for the customers.
- This application will be on the Apple Store for free, so users can download and start using it.
- Easy maintenance of App
- Easily integrated into existing EPOS systems
- Branded for individual restaurant
- All orders are sent via e-mail

Weaknesses

- Initially only available on iOS operating system with intentions to extend to other Operating Systems
- May reduce the human interaction between restaurant staff (waiter) which some might see as part of the eating out experience.

Opportunities

- Experience from this will provide a platform for future products
- An opportunity to develop a database of restaurant businesses for associate marketing
- An opportunity to start a company off the back of this product
- A great opportunity afterwards will be, adverts to target a selected audience and depending on user preferences, a relevant advert to be displayed.
- Another great opportunity for the evolution of this application will be the introduction of bitcoins as a method of payment. Until now, bitcoins haven't proven to be a stable and safe currency but in the future, a combination of these two, will bring people closer to Decentralisation.
- Possible commercialisation routes (licensing, merging or sale to existing firms)

Threats

- Slow uptake of product as a result of being a new and untested product
- Fear of change for those already using existing similar product, including EPOS products
- Lack of track record by the owner/developer of the App
- Threat from established companies

Products and Services

'Pocket Waiter' is an iOS-based Mobile Application aimed at the restaurant market/sector. While it is aimed at this sector, emphasis is on small to medium sized restaurants with limited resources and budgetary restrictions.

Pocket Waiter will integrate into existing EPOS system where it already exists or act as a stand-alone tool to help restaurants engage better with their existing and potential customers. The App will allow customers to view menus, special offers, book their own tables, order from menus and many more features. These can be done before arriving at the restaurant or at the restaurant itself. It will allow for the restaurants to reduce the amount of time required by the waiters to manage other activities.

Other features provided by the Pocket Waiter include:

- It can act as a stand-alone product or integrated into existing EPOS system
- Provides full menu with blurbs and information
- Make reservations and pre-order food
- Take deposit confirmation
- Social Media integration
- Send push notification and messages to customers
- Provide restaurant location visible on map.
- Integrated loyalty coupons
- Take full control over your App content
- App matched with restaurant branding
- QR reader
- Quiz game – add small banner adverts

Benefits

- Boosts business especially from existing customer
- Increase your orders, visitors, customer base, and ultimately, revenue
- It is anticipated that the Mobile Application will continue to be updated to meet new demands and keep up with market sector changes and trends. This will mainly be done by constantly reviewing restaurant and customer usage and feedback. This will allow us to extend the shelf life of the product.

- Our company will aim to own all relevant Intellectual Property Rights to the product. IPR rights currently being considered include Trademark of the name and copyright.
- While there is no environmental legislation applicable to the product, we are confident the product will reduce the amount of paper products used by restaurants in the process of order taking as it will predominantly be done electronically.
- Restaurant will be offered regular after sales updates, but will have access to the back end of the App to enable quick updates (e.g. changing prices for special offers)

Markets and Marketing

The concept of 'restaurant Apps' is not entirely new. Existing 'restaurant Apps' are predominantly used to explore local restaurants, find establishments near you, view menus and order takeaway deliveries. This is great if you are exploring what is available to you.

Pocket Waiter is ideal if you already know which restaurant you want to dine at or your favourite restaurant.

- There are 3 main types of mobile experiences – Mobile Apps, Mobile Websites and QR Codes
- According to a recent research (2012), of the world's 4 billion mobile phones in use, 1.08 billion are smart phones. 91% of all adults have a smartphone.
- 94% of smartphone users look for local information on their phone, and 90% take action as a result.
- 67% say they are more likely to buy a product or service from a mobile-friendly site.
- Apple sold 9 million iPhone 5S & 5C's during the first weekend of availability.
- Customers spend more when using online and mobile apps
- 40% of customers already place orders online
- 20% of quick service restaurants already allow customers to submit orders with their mobile phones.
- Mobile orders can become a significant percentage of restaurant business.

General description

According to the UK Restaurant Market Report 2014 published by Allegra Foodservice, it is estimated that the value of the UK restaurant market will reach £48.2 billion in 2014 and rise to £52 billion by 2017. Branded eateries are forecast to grow by 6.5% over the next three years to reach a value of £17.6 billion by 2017. In particular, fast food outlet sales are forecast to grow by 12.4% and numbers of outlets are expected to increase by 7.6% between 2013 and 2014, with McDonald's and

KFC leading the growth. The report also revealed that the most active consumers in the restaurant sector are aged 18 to 24, indicating opportunities for restaurateurs to engage better with older consumers (www.bighospitality.co.uk/Trends-Reports/Value-of-restaurant-market-to-reach-52bn-by-2017).

A related study published by Allegra Foodservice in July 2014 indicated that independent restaurants are expected to suffer as consumers switch to branded chains, and the total restaurant sector, including independents, is forecast to grow by just 1.7% in 2014 (www.bighospitality.co.uk/Business/Independent-restaurants-could-suffer-as-eating-out-market-grows).

According to a 2014 consumer survey by the NPD Group, the annual traffic in fast casual dining restaurants, such as Nando's and Wagamama grew by 11% between 2009 and the year ending March 2014. This is equivalent to an increase of 47 million restaurant visits. Affordability is one factor that has been attributed to the growth of fast casual dining, with the average bill per visit for one person at this type of eatery working out at £11.90. The report also revealed that the casual dining sector is often the choice for family visits, which account for 36% of visits to casual dining restaurants at dinner time. More than 10% of survey respondents said that they visited casual dining outlets because they are popular with children (www.bighospitality.co.uk/Trends-Reports/Casual-dining-restaurants-are-increasing-in-popularity).

According to a 2012 report by Deloitte entitled 'Taste of the Nation', generational divisions have emerged in terms of the frequency with which individuals go out to eat and drink. Despite financial pressures created by the economic downturn, 18 to 34-year-olds are driving the market by eating out more - on average 31 times a month, up from 25 times a month in 2011. This is nearly double the rate among 35 to 54-year-olds and more than three times that of people aged 55 and over, who eat out on average just 11 times per month. Londoners also tend to eat out the most at over 25 times a month, an 11% increase from 2011 (www.deloitte.com/view/en_GB/uk/industries/thl/43ef03869ef4b310VgnVCM2000003356f70aRCRD.htm).

Taking payment

A till (from £150) will handle basic transactions. Go to www.cashregistergroup.com for examples. Specialist electronic point-of-sale (EPOS) systems, typically including spill-proof touch-screen terminals and software that gives detailed inventory and stock reports are available from around £2,000. Examples of suppliers include South West Systems (www.southwestsystemsuk.com) and Global Retail (www.global-retail.co.uk).

A Chip and PIN machine with portable handsets will be required to process credit and debit card payments. Examples of providers include www.lloydsbankcardnet.com, www.streamline.com andwww.chipandpinsolutions.com. Alternatively, they can be leased from banks. Equipment rental costs between £15 and £35 a month (prices vary according to the supplier and whether the terminal is portable or fixed on the countertop), plus per-transaction charges of around 2%.

Restaurants are increasingly taking payment via smartphone apps and keypads. Examples of providers include iZettle (www.izettle.com), which charges variable rates on a percentage basis depending on sales figures, and WorldPay Zinc, which charges around £60 for a chip and pin keypad and 2.75% per payment with no monthly fees. Go to www.worldpayzinc.com for details. PayPal Here also provides a free app, which requires a card reader costing £99. A fee of 2.75% applies to all

payments accepted with chip and pin cards or via 'PayPal Check-in'. Go towww.paypal.com/uk/webapps/mpp/merchant for more information.

Specialist software

There are a number of restaurant reservation management software packages that allow staff to enter customer reservations onto a calendar. Examples include MICROS (www.micros-systems.co.uk/en-GB/Solutions/Restaurants-and-Catering.aspx) and Resdiary (www.resdiary.com). Prices start from around £80 per month for a restaurant taking up to 350 bookings a month.

Online reservation software that allows customers to make their own reservations via the restaurant's website is also available. Examples include Kernow Software's e-restaurant package (www.kernow-software.co.uk).

Major competitors

There are a range of Mobile and Web-based applications in the market including;

Square Meal
Provides a combination of professional reviews and the fact that you can get details of special offers and make online bookings. All that from a free app.

Time Out
Time Out is a good app for finding good restaurants in and around London. Many of these restaurants have been reviewed by Time Out's team of critics, which we find makes the reviews more credible. only limitation is the fact that it's focused on London.

Poynt
The fact that this app offers more than restaurant information is a factor in its favour, and YouTube is now built into the app.

Toptable
We love the fact that we can book tables online without needing to talk to anyone. The only thing that lets Toptable down is the fact that the app can be limited because not every restaurant you might want to go to offers online booking via Toptable.

Matchbook
Matchbook might be useful if you arrived at a location and wanted to find a place to eat that's nearby, but rather than providing reviews, the onus is on you to write the review, and for that reason it's not really that useful.

The Mobile Food Guide
This app is a little too limited in its offering of restaurants at the moment, and given that at log in it boasts that it is "The UK's Premier Mobile Restaurant Guide" that is a disappointment.

Foursquare
Foursquare has built up a social community in some areas, and in these areas it works well enough as a restaurant recommendation app. Like Forkly and Foodspotting it offers photos of food, but it offers more of a focus on user reviews and more information about the venue.

Forkly
If you like taking photos of your food this app might appeal, but unfortunately it's competing with Foodspotting - and Instagram – and this app really doesn't seem to cover enough restaurants to be useful.

Competitive advantages

Pocket Waiter will be more than just a comparison or search app for restaurants. Being able to reserve a booking and order your entire meal using the app while at the restaurant takes it one step above some of the apps mentioned above.

Our competitive pricing strategy also allows us to be relatively affordable compared to our competitors, taking into consideration our range of feature because the whole system is based on, email, is provided for free. So even restaurants can't afford for a computer screen, the owner can receive his orders directly to his mobile device.

Marketing

Our primary target market is the restaurants themselves. With that in mind, most marketing efforts will be aimed at them. We intend to implement direct marketing, simply by identifying the restaurants, approaching them and possibly doing a demonstration of how the app will integrate and work for them. Once we acquire some early adopters (through incentive), we intend to use these as case studies to engage further users.

We also intend to employ a Social Media campaign to facilitate this process, possibly using a marketing agency with experience in the area (resources permitting)

Management, team and Personnel

Name: Athanasios Gkavalis
Work: Cardiff University School of Computer Science – 3rd year Undergraduate Student
Telephone: (+44) 7716748397
Email: thanasisgavalis@gmail.com
Home Address: Windsor house 6, Westgate Street – Cardiff [CF10 1DG]

Future Developments

There are options for the future development of 'Pocket Waiter' in the long term. Considerations include the potential collaboration of a 3rd party organisation to increase its functionality, integrating detailed inventory and stock report for the restaurants.

There is also consideration in expanding into extended regions depending on the success of the app in the early stages within our target region.

Finance

Current start-up investment already made.

- £2500 total cost of the iMac
- £80/year apple developers' subscription (in order to make it available to the app store)
- £120/year for CS5 Adobe Photoshop
- £40/year for server maintenance

Product / Service cost:

There will be 3 introductory models. These will be differentiated by the range of feature available.

Basic model @ Free:

The basic plan will be provided to all restaurants for free. It will provide them with the basic operations of the Pocket Waiter App, like having their restaurant and their menu on the app, plus allow customers to order.

Premium Model @ £12/month:

The premium plan will provide the restaurant owners with more details of user preferences and also the user-interface will be more advanced. For example, restaurants under premium plan can see what people at that area like most, so to supply their restaurant's warehouse with food items that are mostly preferred. Also the email order will be processed automatically from my team, so in the kitchen to see a more structured email rather that a common email.

Diamond Model @ £19/month:

The diamond plan is the best for large restaurants that already use expensive EPOS systems. It will provide them with all the above and it will also give priority listing inside the App, to the restaurant. When users pass through a restaurant with the diamond plan, Pocket Waiter app will also send users notifications about new deals or discounts the restaurant offers. The price for this plan will not be higher than £19 monthly so restaurants to allow easy transition from premium to diamond plan.

Financial Assumptions:

We do not anticipate generating any significant sales income in the Y1 from the product itself (i.e. the App). As we go through the beta-phase in Y1, the App will be available for free. The only source of income will be from 'ads' available through the App. This will depend on 'click per pay' and challenging to estimate. We have gone for a conservative estimate.