
Food Ordering System for the GMIT Catering Company

Ronan Connolly

Vladislav Marisevs

B.Sc.(Hons) in Software Development

APRIL 24, 2016

Final Year Project

Advised by: Dr John Healy

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)



Contents

1	Introduction	8
1.1	Chapter Summaries	9
1.2	GitHub Links	10
2	Context	12
2.1	System Administration and Management Web Application . .	12
2.2	Mobile App	12
2.2.1	Cross Platform Frameworks	13
2.2.2	Tooling	14
2.3	Push Server	14
3	Methodology	16
3.1	Direction	16
3.1.1	Interface	16
3.1.2	Bleeding Edge Technologies	18
3.2	Agility	20
3.2.1	Agile	20
3.2.2	Meetings	21
3.2.3	Team Work	23
3.3	Testing	24
3.3.1	Test Server	24
3.4	Source Control	25
3.4.1	GitHub	25
3.5	Choices	26
3.5.1	PHP	26
3.5.2	SQL	26
3.5.3	JavaScript	26
3.5.4	JSON	26

4	Technology Review	27
4.1	System Administration and Management Web Application . .	27
4.1.1	php	27
4.1.2	Zend Framework 2	27
4.1.3	MySQL	28
4.1.4	Composer	29
4.1.5	WAMP Server	29
4.1.6	Server Grove Hosting	29
4.2	Mobile App	30
4.2.1	HTML/CSS	31
4.2.2	JavaScript	32
4.2.3	AngularJS	32
4.2.4	Ionic Framework	34
4.2.5	Heroku	35
4.3	Stores	36
4.3.1	iOS Store	36
4.3.2	Android Store	37
4.3.3	Windows Phone Store	38
4.4	JS Tools	38
4.4.1	Yeoman	38
4.4.2	Gulp	41
4.4.3	Jasmine	43
4.4.4	Karma	44
4.4.5	Bower	45
4.4.6	NPM	45
4.4.7	BASH	46
4.5	Web App Alternatives	47
4.5.1	Cordova	47
4.5.2	PhoneGap	47
4.5.3	JQueryMobile	47
4.5.4	Xamarin	48
4.6	Push Web Application	48
4.6.1	CouchDB/Cloudant	48
4.6.2	NodeJS	48
4.6.3	ExpressJS	49
4.7	GitHub	49
4.8	JSON REST interface architecture	50
4.8.1	HTTP Requests	51
4.8.2	Custom API	51

5	System Design	52
5.1	Database	52
5.1.1	Purpose	52
5.1.2	Procedures and Functions	52
5.1.3	Tables	55
5.2	System Administration and Management Web Application . .	59
5.2.1	Admin Authentication	60
5.2.2	Voucher system	60
5.2.3	Order system	61
5.2.4	Customer and accountancy	64
5.2.5	Ionic routes & functions	66
5.2.6	PDF make	66
5.3	Mobile App	67
5.3.1	Sections	67
5.3.2	Use Cases	76
5.3.3	Code Snippets	77
5.3.4	Diagram	80
5.4	HTTP RESTful Route Requests	81
6	System Evaluation	86
6.1	Web App	86
6.1.1	PDF make vs DOM PDF	86
6.2	Mobile App	86
6.2.1	Chosen tech, why and how?	86
6.2.2	Alternatives tested	86
6.2.3	Robustness	86
6.2.4	Performance	86
6.2.5	Limitations	86
7	Conclusion	87

About this project

Abstract This project sets out to create a food ordering system for a local company. The systems primary components are a mobile application that the user interacts with and a web application that the staff interact with.

The need for such a system stems from two problems, firstly the issue of rush hour times during business hours where there is a vast number of customers to service, and secondly to bring more presence and promotion to the business, as they are finding it hard to reach out to their current customer base and would be customers.

We aim to solve the first problem by having a system in which customers can pre-order sandwiches and other products via a mobile application. Users will be able to top up their account, order products, pick a collection time, and view their balance, products and past order history.

The second problem will be solved by implementing push notifications into the application so that the company can let customers know about menus, events and various other updates. Another way to increase promotion and presence is by having various information about the company on the application; for instance: opening times, contact details and general information.

All of the information on the web application can be updated; this includes the menus, opening times, user and staff details, and much more. This information is reflected in the web application. Interactions from within the mobile application including: topping up, logging in, registration and ordering go through the web application.

We plan to create a cohesive, thoughtfully designed, robust system that solves these two problems.

Authors This project was created by two fourth year software development students: Ronan Connolly & Vladislav Marisevs, as part of our Bachelors of Science honours degree in Software Development.

Ronan was in charge of creating all aspects of the user facing mobile app. Vladislav was in charge of creating all aspects of the staff facing web app.

We spent most of our shared time coming up with the overall architecture we would implement, and an interface to be used between mobile and web app for transfer of data.

Acknowledgements We would like to acknowledge and thank our supervisor Dr John Healy for all the time and effort he has put into helping us throughout this project, he gave us a good structure and set milestones for us in order to keep on top of things.

We'd also like to thank the GMIT Catering Company staff for the time spent meeting with us in order to continuously improve and adapt the project.

Chapter 1

Introduction

We set out to create a food ordering system for the GMIT Catering Company (known henceforth as the company). The basic structure is a mobile application (henceforth known as mobile app) for the Android and iOS systems that the user interacts with, and a server (henceforth known as web app) that the staff can log into in order to view transactions, user details and to update the mobile app.

The reason such a system is needed is that queues during peak times tend to be enormous and currently it's hard to service all the customers.

Another reason is to encourage customers to get into a habit of repeat ordering, if it is an easy process then it should increase purchases.

Lastly, the company wants to increase presence and promotion in the college, in order to achieve this end we have implemented push notifications where staff can send a notification to all users. On top of this the mobile application itself serves as a promotional device, containing details of various aspects of the company.

In order to develop this system we required to connect different platforms together and let them communicate. We were using Ionic Framework for creating cross mobile application that would talk to Zend Framework 2 which will act as administration website and API (Application Programming Interface) for controlling data transfer between MySQL database and mobile program.

The components contained within the mobile app include pages for login, registration, about the company and user details. There is also a way to top up and order sandwiches. A huge emphasis is put on design for this project, using the company's colour theme and creating a nice icon. This mobile app was created using the Ionic Framework which is programmed primarily using

the AngularJS framework. It's a cross platform mobile application that will authenticate users using their credentials. This option will allow us to create an account wallet, identify a person and their order history. The mobile app design uses native phone features and user interface components to let user operate without any special training.

The components contained within the web app include many pages such as the login system, orders, stock, user details, vouchers(for adding credit to your account), settings(collection and opening times) and accounts(staff) pages. This web app is a administration website which allows user to configure and manage the whole system. Users can change opening, closing and food collection times. In order to access these settings users should authenticate him self and then will be able to nominate new administration members. This website also allows to view list of orders, customers and track their history.

This web app was created in PHP using Zend Framework 2. Most of the information on the web app is reflected on the mobile app.

The two applications talk to each other via JSON over HTTP Get and Post requests.

We set out to create a well thought out, carefully designed, robust food ordering system using modern technologies. This project could be extended in the future to be used

We used an agile structure where we had certain components we needed complete by specific dates. We had various meetings each month with our supervisor and several members of the company.

1.1 Chapter Summaries

Here is a summary of all the chapters in this project report.

Context

Here we talk about how the project came about, what the initial ideas, goals and objectives were. We'll also talk about the main components of our project, how we chose them, the alternatives we tried out and a basic overview of the usage of our food ordering system.

Methodology

An insight into how we began the project, the research, our technology and design choices, our thought process and how we went about allocating tasks and organising meetings.

We'll also talk about the objectives we set out to accomplish and how we went about completing them.

Technology Review

Here we talk about all the technologies that we have mentioned in this report and any others that we may have used in completing the project.

System Design

An overview of the project architecture, including lots of diagrams and screen shots.

System Evaluation

An evaluation of our various project components, including how we tested our system for robustness and performance. We talk about the outcomes that were achieved in relation to what are goals were, how far we strayed from our goals and some issues we came up against. Any limitations or opportunities we encountered in our approach and in the technologies we chose.

Conclusion

A broad overview of our development work-flow, from our thought processes, meetings, technology choices to our issues, problems and perceptions of the project as a whole. We'll touch on our overall experience and what we would have done differently.

1.2 GitHub Links

The web and mobile app repositories are private so you must ask to be added as a collaborator in order to view them. Each section below has a clickable heading and contains a link to the respective GitHub repository.

GMIT Catering Organisation

This GitHub organisation was created to host all repositories related to this project [1]. These projects will be maintained here or forked here on completion.

Gist ReadMe

This contains basic instructions for using each component of the project [2].

Web App

The food ordering web app [3].

Mobile App

The Ionic mobile application [4].

Test Server

The test server that we used initially to test the mobile app [5]. This received requests and sent back mock data that imitated the real server.

Push Web App

This web application is using the MEAN stack technologies [6]. It is hosted on Heroku and is used to save user details. Administrators can log in and send push notification messages to registered users.

Project Report

The project report that you are currently reading [7].

Chapter 2

Context

- Our project consists of creating a food ordering system for the GMIT Catering Company.
- Our basic objectives were to create a mobile and web application to deal with the above item.

Below we will:

- explain the various components of our project.

2.1 System Administration and Management Web Application

I decided to do this because of that, etc PHP is great but maybe a Java web server would have being better, easier to modify, I already know Java. SQL is hard to change over time, maybe implementing a NoSQL database would have been good...

2.2 Mobile App

The mobile application is created using bleeding edge technologies such as the Ionic framework, which utilizes the AngularMVW framework, which in turn is programmed using JavaScript. HTML and CSS were also heavily utilized.

Initially I had no idea about JavaScript, web development or cross platform development. I spent much time studying JavaScript, Angular, Ionic,

and the MEAN Stack (MongoDB, ExpressJS, AngularJS, and NodeJS). I then spent a lot of time trying out various cross platform development frameworks.

I will speak more in detail about these technologies in the technology review chapter.

Some of the features of the mobile application are:

- Login
- Sign-up
- Password Reset
- Food Ordering
- About Information
- Account Information
- Password change
- User History
- Topping up

I will go through these features in detail in the system design chapter.

2.2.1 Cross Platform Frameworks

I first tried out Cordova [8], which I found had the capabilities to do pretty much anything, but the community and framework is very sparse, it's hard to get anything up & running, and the styling's are horrid.

Next I tried out JQueryMobile [9] which had better styling but again I came across many issues.

Finally I came across the Ionic Framework [10] which uses Cordova underneath. This framework has a huge community of developers, it has the support of Google, Microsoft and many other large companies. They closely work with the Angular team at Google and the TypeScript team at Microsoft. They have a 24/7 group chat system set up with various sub rooms. They have amazing documentation, regular blog posts, quick response to questions on the blog, forums and chat. With Ionic you get:

- All the capabilities of Cordova, which allows you to access the Mobiles native APIs easily
- A slick native UI experience. The app changes design depending on the platform it's running on
- Extensive tooling. Starting an app, templates, app store image creation, logo creator, etc
- Rapid development cycle, there are constant updates (which can cause issues, but is usually great)

2.2.2 Tooling

Once I decided on using the Ionic framework I spent my time completing various JavaScript, Angular and Ionic tutorials. This was a steep learning curve as there is so much tooling for all the JavaScript frameworks. I installed NodeJS in order to use NPM (Node Package Manager) in order install Ionic.

Then Ionic came with it's own tools for various tasks, including SASS for programmatic CSS, Bower (Like NPM or Maven) for adding in new components, Grunt for running tasks (like Ant) and some others like Gulp (similar again to NPM).

Each of these tools takes time to learn. I read documentation and completed at least one tutorial for each.

2.3 Push Server

In order to facilitate push notifications I needed to get their device token and save it in a database. I created a controller in the mobile application, once the app starts it gets the device token along with some other information and sends it to the push server.

The push server is always listening for incoming requests, once it receives one it evaluates it, adds a timestamp and saves the user object (JSON) to a CouchDB server (Using IBM's Cloudant Web Server).

The push server is a MEAN Stack, Yeoman scaffolded project that uses NodeJS as the environment, Express as the Web Framework and Angular as the MVC (Model View Controller) framework in order to build the web application.

The reasons we chose to have the push notification server separate is that:

- We did not realise we needed to save the device tokens initially and to implement this new table into the SQL schema is a big effort
- We wanted to try out a MEAN Stack application
- We felt there was no big disadvantage on having the push server separate
- The mobile app is so tightly integrated with the push server (everything is written in JavaScript), that it made creating the web application extremely simple
- The GMIT Catering company may assign the task of push notifications to somebody that they do not wish to have access to the food ordering system, such as a social media expert.

Users can:

- Login/Logout via username and password
- View how many devices are registered for push notifications
- View previously sent push notifications
- Send new push notifications, and see if they were sent successfully.

Chapter 3

Methodology

When we began this project we knew we had to split it up into sections and allocate parts to one another. Since Ronan had worked on many mobile applications before and Vlad had created various web applications in conjunction with large databases we decided Ronan would take on the mobile side and Vlad the web app & database.

We have already spoken in the Introduction why we chose the technologies we chose so here we will focus on our implementation of these technologies.

3.1 Direction

We took some inspiration from the "Manifesto for Agile Software Development" and tried to implement some of the aspects. As well as this we looked into SCRUM and RAD.

We used the GitHub issue tracker to work through tasks, flag bugs and add enhancement requests. We found that the regular meetings and feedback were great for tweaking design but it held us back in many respects. There were times when we needed to overhaul the design that took much time, especially when it involved changing the database scheme (SQL does not like this).

3.1.1 Interface

Due the fact that the mobile app relied on the server which would not be complete for some time, we needed to come up with a common interface we could both program towards. This idea was inspired by Edsgar Dijkstra:

He worked closely with Bram J. Loopstra and Carel S. Scholten, who had been hired to build a computer. Their mode of interaction remains a model of disciplined engineering: They would first decide upon the interface between the hardware and the software, by writing a programming manual. Then the hardware designers would have to be faithful to their part of the contract, while Dijkstra, the programmer, would write software for the nonexistent machine. Two of the lessons he learned from this experience were the importance of clear documentation, and that program debugging can be largely avoided through careful design.

- The University of Texas at Austin - The General Faculty [11]

We decided to create a RESTful API on your web server, to which the mobile application could query. In order to test the mobile application in the mean time, we created a test server that would receive a request and return mock data. We could then switch between the real and test server with one line of code change. This also helped us find bugs, if the test server worked and the real server did not, then we knew there was an error on the real server. This test server was written with the MEAN stack of technologies (except without MongoDB).

MEAN:

- MongoDB
- ExpressJS
- AngularJS
- NodeJS

The reason we chose this is that for RAD (Rapid Application Development) the MEAN stack is very easy to use. Especially since the Ionic framework is already using AngularJS and shares many of the same tools including the NPM (Node Package Manager).

Here is a piece of code from the test server. This code is for logging in:

```
1 // login
2 app.get('/verifyuser/:customerLg/:customerPW', function(req,
  ↪ res){
3     var result = {
4         "customer_id":"11E4E6370B663F0D81B9EC9A743CC2AE",
5         "customer_name":"Regina",
6         "customer_surname":"Walsh",
7         "customer_cash":"10.20",
8         "customer_mobile":"0853243435"};
9
10    res.contentType('application/json');
11    res.send(JSON.stringify(result));
12 });
```

I hosted this test server on Heroku [12]. Heroku hosts projects as git projects, so you just need to add a config file and push to your heroku repository. Once complete your project is running live in the wild. Again this is another nudge towards RAD.

3.1.2 Bleeding Edge Technologies

We wanted to use bleeding edge technologies in our project, since this is our final year project we wanted to use it as a chance to research some novel technologies and see what is possible, while at the same time creating a robust system with a well thought out design.

While cutting edge refers to the newest technology, it also infers that the technology has gone through the wars and is battle-tested (to a degree where it can be considered for review).

Bleeding edge is different in the sense that it is so new, that we don't even know if it will be around for very long, or whether the current merit it is receiving is worthy or is caused by the hype at the time.

Bleeding edge technologies are all around, especially right now within the web development community, it seems there is a new framework or API that you **must** know every other day.

We chose some bleeding edge technologies that we felt would stand the test of time and make it into the *cutting edge* stage. Here is a list of some of these technologies:

- AngularJS
- Ionic Framework

- NodeJs & ExpressJS
- Heroku

AngularJS is owned by Google, and Microsoft have started working with them. This is a good indication of a solid technology, along with all the statistics on it's performance and adoption.

The Ionic Framework has grown to be the most popular cross platform framework. They work closely with the AngularJS team, and they were even presenting at the Microsoft Build 2016 conference, which is interesting since Microsoft just bought Xamarin which would be considered a competitor.

NodeJS and ExpressJS were used for the test server and the push web application. These are still in there infancy but the Node Package Manager (NPM) has in a few short years grown to be the most popular package management system out there.

We did not know any of these technologies coming into this project, so the learning curve was very high and it tested us throughout the development life-cycle. Many things were learned from proper planning, emailing, documenting progress, using Git properly, setting milestones, dealing with clients and the time spent learning/studying technologies compared to actually implementing them.

Here are some graphs showing the growth trend among various package managers [13]:

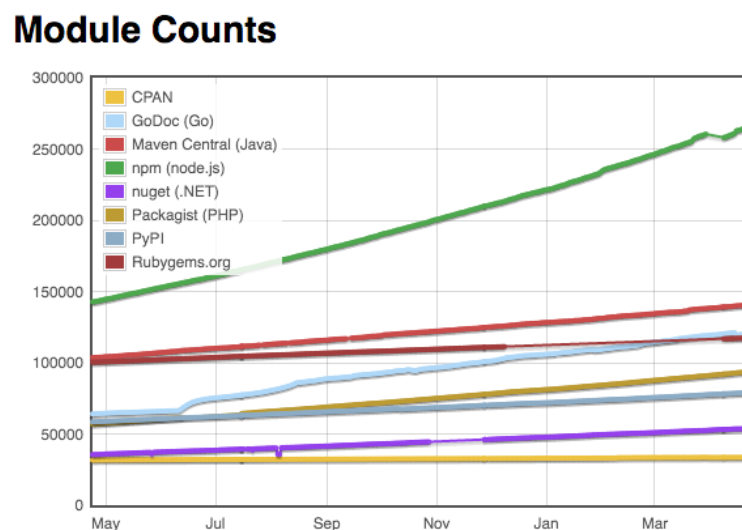


Figure 3.1: Module Count Growth in the last Year

We can see here that NPM is the clear winner. This is due to the fact that the community is so alive, connected and are really into open source software development.

In this graph we can see exponential growth from NPM:

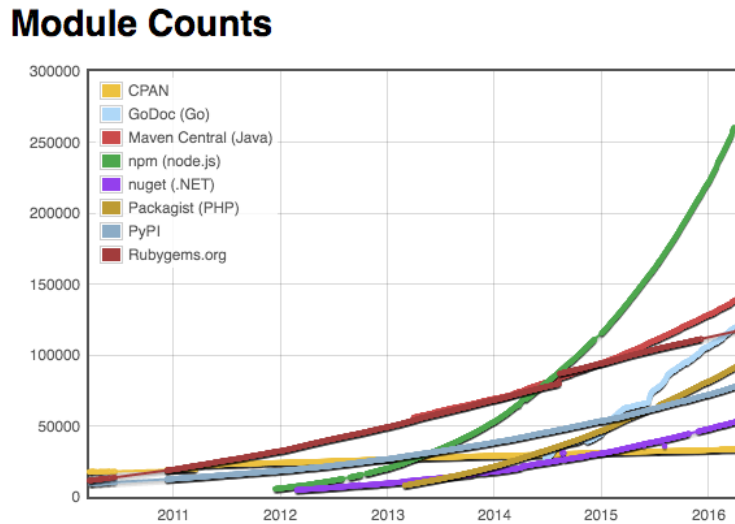


Figure 3.2: Module Count Growth All-time

Server Providers that use the git [14] technology to allow users to manage their servers are becoming increasingly more popular as time goes on. Examples of this our companies like Heroku. This makes it very easy to maintain, manage and utilize a RAD type of development.

3.2 Agility

We wanted to be flexible in our approach since most of the aspects involved from meetings, planning, project development and the technologies were new to use. Agile suited us well.

3.2.1 Agile

We decided that we wanted to go through this project in a modern way so we looked into the *"Manifesto for Agile Software Development"* [15]:



Figure 3.3: Manifesto for Agile Software Development

With these ideas behind us (and some more taken from SCRUM and RAD) we worked to:

- Have regular meetings with our client
- Constantly take feedback on board
- Change designs throughout development where possible
- Break down tasks into their constituents and work through them

3.2.2 Meetings

Initially we had a weekly meeting with our client where sometimes others would be present (one or two other clients or our supervisor). We also held a weekly meeting with our supervisor. As time went on we had meetings every two or three weeks (depending on the workload and what needed to be done), in between our meetings we kept in constant communication making sure that everyone knew their tasks.

Here are the minutes from two of our meetings this year:

Hi all,

Here is an overview of the meeting today.
We will meet up next on the 10th of February at 2pm.

Annemarie tasks:

- Hosting

ServerGrove (approx €100 per year), you should pay monthly in case you decide to switch after 6 months.
[+1 786-999-6585](tel:+17869996585)
<http://servergrove.com/>

- Domain Name

Call Blacknight (€20 per year)
gmitcateringco.ie
<https://www.blacknight.com/search/#gmitcateringco.ie>

- Android Store

Developer account (€25 once off)
<http://developer.android.com/index.html>
<http://developer.android.com/support.html>
-> Click on "developer console"
-> Log in with gmail account (catering one would be best)
-> Follow sign up procedure (should be a way to contact someone, phone or chat panel (may be able to ask for number in chat))

Contact:

<https://support.google.com/payments/#contact=1&topic=6209953>

- iOS Store

Developer account (€99 per year)
1800 804 062
<https://developer.apple.com/>

Vlad

- Update PDF Paper Size

Ronan

- Finish off Push
- upload iOS update to store
- Refactor Code

Next Tasks

- Set up printer at Espresso Bar
- Test on Students/Staff

Regards,

Ronan

Figure 3.4: 2016 Meeting #1 - Minutes

Hi all,

Below is an overview of the meeting today.
We will meet up next on the 24th of February at 2pm.

Costing

Domain, Hosting and Android accounts should be fine.
Apple account may be an issue as it is a recurring charge.

Brid in finance, needs to talk to IT about everything.
Brid is back on Monday.

Top Up Cards:

31st of August all top up cards (including purchased ones will expire).
This needs to be written on the top up cards.

Current text (for ann-marie to update and email back):

"1. Validly activated Euro Vouchers are only redeemable for merchandise at GMIT Catering Company Ltd in Republic of Ireland (excluding Lottery tickets, saving stamps and vending machine sales)
2. Euro vouchers can not be redeemed in full or in part for cash or gift cards.
3. GMIT Catering Company Ltd. can not be held liable for lost, damaged, stolen or out of date vouchers.
4. This voucher may not be used for any other purpose and in particular may not be used for resale or as a prize or for any re-use, be it for commercial use or otherwise without express written permission of GMIT Catering Company Ltd.
5. GMIT Catering Company reserves the right to amend these terms and conditions from time to time when it reasonably considers it necessary to do so. Reasonable notice of such changes will be given where possible. For full terms and conditions see <http://www.gmit.ie/gmit-catering-company>"

Android Store Account

Android store needs a new gmail account.
As we cannot log in with our gmitcatering@gmail.com
We think it may be an incorrect email, it looks like it is owned by someone else.

Vlad

Top up cards updated to new layout for cardboard paper print outs.
Next Tasks
Update details on top up cards (Add date and QR number to card).

Ronan

Created push form, temporarily hosted on heroku.
Next Tasks
Upload iOS update to store.
Hook up push form with app.
(optional) create a terms and conditions document that states to the user the usage of the app and top up cards (including when they expire). [possibly add a agree checkbox when registering].

Website

Ann-marie, try to break website.
<http://canteenapplication.vmarisevs.me/>

...

Ronan

Figure 3.5: 2016 Meeting #2 - Minutes

3.2.3 Team Work

We divided up all the work initially and along the way. We used the GitHub issue tracker to flag bugs and highlight different improvements that could be made to the project. We met up a few times a week to go through

the project and make sure everything was syncing. We learned much about working in teams from this project.

3.3 Testing

Since there are so many different aspects to this project it was hard to bring in a testing framework. I used created some test cases in Java with the Selenium WebDriver library in order to go through each tab of the web app and push server. This helped us verify everything was working fine, we just needed to sit back and watch the test cases pass.

For the mobile app I tried out Karma unit tests in conjunction with the Jasmine library. We had an okay experience with this but we feel it is so cumbersome to set up that it may just be better to do user testing.

We downloaded the mobile app on Android and iOS, and tested all the features out in various locations. We did user testing by getting people from our year and the year below in software development and also some students who are not in the IT field to test all the features out and tell us the best and worst things they encountered within the application. This helped us improve the applications user experience, design and to kink our any bugs. We went through the web application with our client many times and updated it to make it easily usable for them.

3.3.1 Test Server

The test server contained all the routes necessary to mock the real server. Here are the routes used:

Route	Use
/	Homepage
/createuser/[data]	New User
/verifyuser/[data]	Login
/getfood	Get Menu
/userhistory/[data]	Get User History
/buyfood/[data]	Order Food
/changeuserpw/[data]	Change Password
/workingtime/[data]	Opening Times
/topup/[data]	Topup Account
/recoveruserpw/[data]	Reset Password

Table 3.1: Routes table

This is actually a good place to get a simple overview of how the real server works. This is our interface.

In order to switch between the test and real server we only need to change one line of code. This file is located in:

www/app/app.js

```

1  .constant('SERVER', {
2    // test server
3    // url: 'https://catering-test-server.herokuapp.com/',
4
5    // real server
6    url:
7    ↪ 'http://canteenapplication.vmarisevs.me/mobileapplication/',
8  });

```

3.4 Source Control

We made sure to use source control throughout our project, utilizing branches and constant commits. Between the mobile app and web app alone we have a total of **210** commits.

3.4.1 GitHub

We chose to use GitHub as our source control repository vendor as it is tried and trusted, we are used to it, we use it currently in college and it hosts

some of the largest projects on the web. Also the issue tracker, wiki, web site builder and various other features are compelling.

3.5 Choices

We already mentioned in the Context chapter about why we chose Ionic, Zend and Mean stack framework/architectures. Here we will discuss why we chose PHP, SQL, JavaScript and JSON as our core technologies.

3.5.1 PHP

3.5.2 SQL

3.5.3 JavaScript

JavaScript is the language that the Ionic and Angular core development team use for creating applications. You can write any JavaScript from any framework in other languages like CoffeeScript or TypeScript.

These follow stricter rules and allow JavaScript to be strongly typed. This gives you auto-completion amongst other things. It nearly a bit more like you're writing Java with these languages. These languages are transpiled into JavaScript (as opposed to compiled).

**The difference between transpiling and compiling is that transpiling compiles to the same abstraction level.*

TypeScript and JavaScript are at the same abstraction level. Transpiling is a form of compiling (a subset). Transpilers are gaining popularity and often take the direction of up and coming standards. If you learn Microsoft's TypeScript you are learning much of the new JavaScript standards to come.

**TypeScript is being incorporated into Angular2.*

If I were to do this project again in the future I would consider using TypeScript instead of JavaScript.

3.5.4 JSON

We chose to use JSON for passing messages between the mobile and web applications due to the fact that it is so lightweight and easy to create and parse. Since the Ionic application is written with JavaScript it stores all its data in JSON format by default.

Chapter 4

Technology Review

Here we will talk about all the technologies we used.

4.1 System Administration and Management Web Application

After gathering all system requirements and we have done a research and based on them I decided to use PHP based framework because there is a lot of hosting companies that support this server scripting language and it has very cheap pricing.

4.1.1 php

PHP is in the market since 1995 and is one of the widely used server scripting language. It is quite powerful tool for making dynamic web pages and it is free. There are many popular websites that are written on *php* such as Facebook, Wikipedia, Flickr, Mailchimp, Wordpress.com. PHP code can be embedded into html page. This type of structure allow to use it in various web template systems, content management systems or web frameworks. From beginning *php* wasn't object oriented language, but in the later versions they redesigned it.

4.1.2 Zend Framework 2

Prototype was developed using *Zend Framework 2*. This is an open source framework for developing Web applications and services. It was written on *php* and it is loosely coupled architecture, which allow developer to

code each component independently and designed with Model View Controller structure. **Zend Framework 2** uses 100% object-oriented code and utilises most of the new features of **PHP 5.3**, namely namespaces, late static binding, lambda functions and closures. [16]

Barry and Elhakeem [17] argues that Zend Framework enables simple, rapid and agile web application development process, and it also offers AJAX support to convert XML data into JSON format and integrates the most widely used APIs and Web Services of third-party companies such as Google, Microsoft, Amazon, Flickr and Yahoo. ZF provides many options for validation such as Dojo tools it also allow to filter all inputs.



Based on Hyun Jung La and Soo Dong Kim publication we can say that this project has a Balanced Model View Controller Architecture [18]. The **Zend Framework 2** contains applications business logic or Model. And it is also responsible for validating user inputs or Server Side Controller and Web application Views. While **ionic** is responsible for Client Side View and Controller (on the smartphone). In this case Mobile application uses **zf2** Controllers to connect to **MySQL** database.

QR code generator

QR codes are very popular type of two-dimensional barcode [19]. This module was used for generating QR codes for vouchers. When voucher is printed users can scan this code and top up their balance. This module is overridden for Zend Framework 2 and it uses Google Developers QR code generation API. [20]

DOM PDF module

In order to generate **.pdf* file I have used DOM PDF module for Zend Framework 2. This is most common plugin that let me to solve problems with printing remote image. It is easy to use, because it parses HTML as DOM object and generates pdf file based on that. [21]

4.1.3 MySQL

To store information we were using **MySQL** database management system. It's reliability is proven through the years. **MySQL** offers complete ACID (atomic, consistent, isolated, durable) transaction support, unlimited row-level locking, distributed capability, and multi-version transaction



support where readers never block writers and vice-versa. Easy manageable platform that allow a DBA to manage, troubleshoot, and control the operation of many *MySQL* servers from a single workstation. [22]

4.1.4 Composer

For such large distributed projects like this more often used some sort of dependency managers, that allow to force the development cycle and reuse any of already written packages. One of the most popular dependency manager for *php* is *composer*. After installing it we can start using it, simply by creating *composer.json* file which describes the dependencies for your project and may contain other metadata as well. [23]

composer.json example:



```
{
  "require": {
    "php": ">=5.5",
    "zendframework/zendframework": "~2.5",
    "zendframework/zftool": "dev-master",
  }
}
```

As you can see, require takes a key value pairs where key maps to package names and value is version of it. We also can configure the minimal supported release or nominal version. When *composer.json* is completed, we can run command that will download chosen packages and configure them in our project. To do that run this command in console:

```
$ php composer.phar update
```

4.1.5 WAMP Server

4.1.6 Server Grove Hosting

There is not so many hosting companies that supports Model View Controller structured *php* websites, but the good old days when we were querying **.php* files are gone. This type of structure hides the extension and it will confuse the attacker on what sort of platform it was written.

One of most popular hosting companies in Ireland is Blacknight Solutions. For one of my portfolio websites I was using their services. At that time I was not bothered with developing my own website from the scratch and I have used *Drupal* Content Management System. It is quite easy to install and setup CMS for lite blog type of website, but when it comes to connecting 2 different platforms it is no longer useful. When I developed my portfolio website on *zf2* and tried to host it on their server I had a lot of issues with their policy. Customers don't have access to main *php.ini* file which can change write properties that has to be on to use MVC structured websites.

To solve this problems I have found *Server Grove* hosting company that fully supports *Zend Framework 2*, *Git Client* and *composer*. It took me a while to move my domain name from BlackNight Solutions, but once I have it done all technical support on Server Grove is brilliant.

4.2 Mobile App

The technology stack I chose to use for this project is quite extensive. Ionic is heavily based in the JavaScript community which is known for it's enormous amount of libraries and dependencies. It is quite a steep learning curve to get to grips with the basics, and it takes quite a while to get comfortable in the workflow. I came across a tweet that sums up this experience:

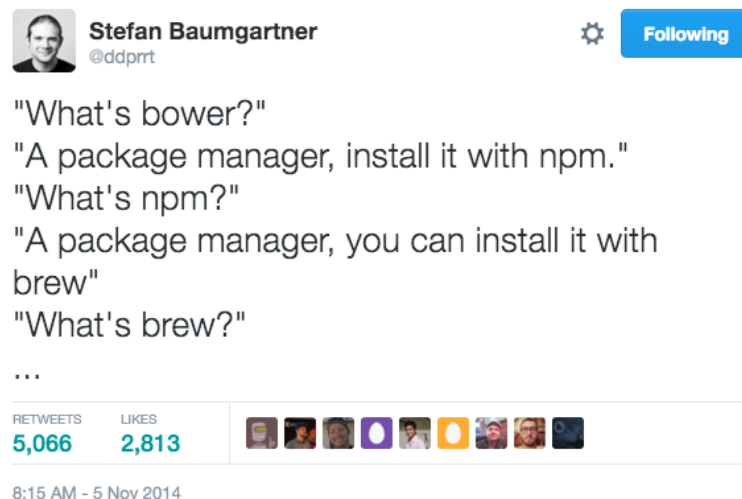


Figure 4.1: JS Developer Learning Curve

I done extensive research on different cross platform frameworks. First I started with JQueryMobile and PhoneGap which had the capabilities but

the community and tooling were quite scarce.

Xamarin was another option, It did not appeal to me as much, during my research I found many people had lots of issues with it and you still have to write some specific code for each platform.

A lecturer of mine mentioned the Ionic framework to me and once I looked into it, it was the obvious choice. Ionic has a huge community, extensive tooling for the majority of tasks needed and it is rapidly becoming the number one option for cross platform development.

Once I had decided to take the Ionic route it meant that I was delving into the world of web development, as Ionic uses AngularJS, which is used with JavaScript, HTML, CSS and all the web development tools and work-flows.

4.2.1 HTML/CSS

I found that learning HTML [24] and CSS [25] was fine, I just needed to know enough to use Angular, which is very basic. I had done some HTML and CSS before so I just needed to brush up on my skills. I took the Web Development [26] course on Code School which helped me quickly get up to speed.



HTML is great because it is a simple structure that is easy to understand. It is just a subset of XML for websites.

CSS makes is really easy to change various parts of the HTML form one place. I used SASS (Sassy CSS), this lets you use CSS in a programmatic way (using variables).

Here is a piece of SASS for the Ionic colours:

```
1 $light:           #fff !default;
2 $stable:          #f8f8f8 !default;
3 $positive:        #5475aa !default;
4 $calm:            #11c1f3 !default;
5 $balanced:        #7baa1e !default;
6 $energized:       #ffc900 !default;
7 $assertive:       #840018 !default;
```

```
8 $royal:                                #886aea !default;
9 $dark:                                #444 !default;
10
11 // The path for our ionicons font files
12 $ionicons-font-path: "../lib/ionicon/fonts" !default;
13
14 // Include all of Ionic
15 @import "www/lib/ionic/scss/ionic";
16
17 // custom styles
18 @import "scss/styles";
```

You can also see that I am importing other style sheets.

4.2.2 JavaScript

Once I had gotten myself into the basic mindset of HTML and CSS I decided to learn JavaScript. I completed three JavaScript courses [27] on Code School. This gave me a great understanding of JavaScript [28].

I found that JavaScript's event based nature took awhile to get the hang of but once understood it was very useful. Promises are a very important concept to understand as JS is asynchronous and tasks may not execute in order, especially blocking tasks. This is useful as you don't need to code in concurrency, it is an inherent feature of the language.

I wrote a literature review on using JavaScript for full stack development [29], this touches on promises, the event based nature, and why JavaScript is so popular.



4.2.3 AngularJS

I completed two AngularJS [?] courses on Code School. Then for each course I completed I done the 2/3 hour screen cast that accompanied it.

The main problem I found with JavaScript is a lack of standards and structure. This is why I believe there are so many JS frameworks out there.



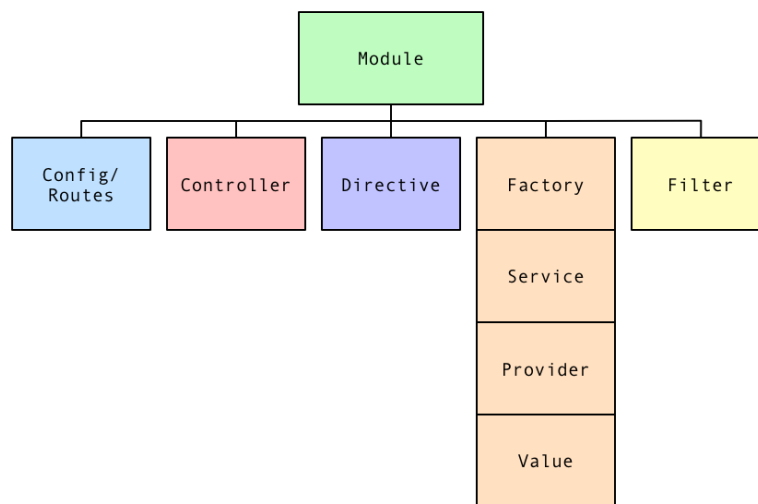


Figure 4.2: Angular Object Diagram

A few times a year there is a new JS framework that all the web developers think is going to be the **one**. AngularJS has risen and is now one of the top JS frameworks, partly I'm sure to do with the fact that Google is the project owner. Competition includes Facebook's React [30] and EmberJS [31].

I completed two Angular courses on Code School. Then for each course I completed I did the 2/3 hour screen cast that accompanied it.

These helped me gain a deep understanding of controllers, services, routing and views. Angular is based around the Model View Controller (MVC) model, where the design, data and logic are all separated. Angular is a Single Page Application (SPA). Only one page ever exists, the *index.html*, then each *view* is loaded into a section of this page. Each *view* has a *controller* attached to it. The Model contains the data which is loaded into a template that you code, this creates the view.

Two way binding is used so that as you enter information it can be seen straight away.

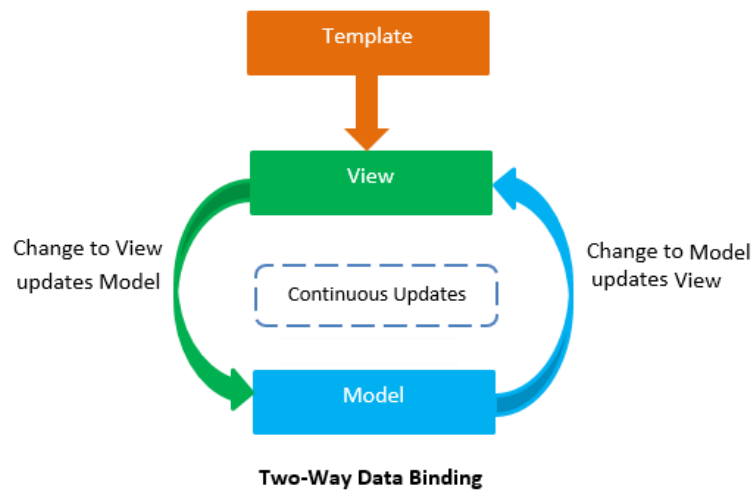


Figure 4.3: Two Way Data Binding

4.2.4 Ionic Framework

Once I understood AngularJS I completed some Ionic tutorials on EggHead.io and Thinkster.io.

The Ionic [10] community is really great, the forum is very engaging, questions are answered very quickly and they post to their blog regularly. The founders Max Lynch, Adam Bradley and Ben Sperry talk about various updates every few months on YouTube, they call this *The Ionic Show* [32].



To Start an Ionic application you run these commands in bash:

```
1 $ ionic start myApp tabs
2 $ cd myApp
3 $ ionic platform add ios
4 $ ionic build ios
5 $ ionic emulate ios
```

Everything is done through command line.

Here is a simple controller:

```
1 (function() {  
2     'use strict';  
3  
4     angular  
5         .module('canteen.controllers')  
6         .controller('AboutCtrl', about);  
7  
8     about  
9         .$inject = ['$scope', 'Orders'];  
10  
11     function about($scope, Orders) {  
12         $scope.opening = Orders.opening;  
13         $scope.closing = Orders.closing;  
14         $scope.times = Orders.collTimes;  
15     }  
16 })();
```

One issue I came across is that there is no definitive way to lay out structures. This is a common issue in JavaScript, since the standards are so loose (which benefits it in order ways).

The above layout follows John Papas AngularJS style guide [33]. John Papa works with the Microsoft team who are implementing strong typing into Angular2 (through the use of MS TypeScript). This gives me the impression that he has a good grasp on best design practices.

As you can see in the above example, I have broken up the code into its constituent components, rather than the usual nested JavaScript hierarchy. I think it is pleasing to the eye and easy to understand.

4.2.5 Heroku

We used Heroku [12] to host the push and test servers. Heroku is great for RAD, it is so simple to get a project live. It treats the project that is hosted as a git repository.



In order to use Heroku you:

- Install the Heroku Tool-belt
- Create a Procfile

- Initialize a Git repository
- Create a Heroku project via command line
- Commit and Push to the Heroku remote

Here are the commands to create a new server and push to it (once pushed it goes live instantly):

```
1 > heroku login
2 > git init
3 > heroku create
4 > git add .
5 > git commit -a
6 > git push heroku master
7 > heroku open
```

Here is the Procfile (this is the Heroku config file for the project, here we give the server the command to start our project):

```
1 web: node server.js
```

4.3 Stores

We decided to release the application on the iOS and Android stores.

iOS App Store Link.

Android App Store Link.

4.3.1 iOS Store

Apples iOS store [34] is tricky to get used to, first you must submit an application for a developer account which costs €99 per year. They require much information. Once complete you have to create a developer certificate, provisioning profile, push notification certificate, download them, and sign them locally. You have to get these certificates for both production and development purposes, as Apple see these as completely separate applications.



Once complete you install XCode (for developing iOS apps), compile the application, login to your profile in XCode, go through the local setup, then archive the application in XCode and submit this to the store (through XCode). After a few hours you get an email saying that the Archive is okay. You then go to the iTunesConnect website, login, and start the store application process, this takes numerous steps.

Once complete you can add the previous archived file and submit for approval to the store, this takes 1 to 2 weeks to complete, often resulting in a change wish the revision team want to see updated. You must provide the revision team with a mock account to test the application.

I got used to this process after my third or fourth update.

This process results in only top quality applications making it into the iOS stores. The iOS store makes more money then the Android store by a long shot.

4.3.2 Android Store

Submitting an app to the Android [35] (Google Play) store is a joy. First you go to the Google Play website, login with your Google account, give some extra details, pay €25 (per year) and you are instantly in. You then click on "create new app", fill out some details and it asks for the app binary file.



In order to generate the Android binary file you must create a local android keystore (for signing) compile the app with the Ionic tool, sign it, zipalign it and then submit it to the Google play store. You submit it by simply dragging it into the Google Play web site window. Within two hours the application will be live.

I looked into why it's so quick, and apparently they test your application after it is submitted, this leads to happy developers.

Here is the bash script I create to automate the Android build process:

```
1 #!/usr/bin/env sh
2
3 ionic build android --release
4 rm gmitcat.apk
5 cp
   ↪ platforms/android/build/outputs/apk/android-release-unsigned.apk
   ↪ .
6
```

```
7 jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1  
  ↪ -keystore ./ronanc.keystore android-release-unsigned.apk  
  ↪ ronanc  
8  
9 zipalign -v 4 android-release-unsigned.apk gmitcat.apk  
10  
11 cp gmitcat.apk ~/Dropbox/releases/
```

**I couldn't do this for iOS as you need to use XCode.*

4.3.3 Windows Phone Store

The Windows Phone store [36] is actually not that bad, I have already submitted 36 applications (mostly micro applications) to this store. It's free to sign up and the set-up process is very simple, lots of check-boxes and drop-down menus.

The reason we didn't create a Windows Phone version is that firstly it is not supported by Ionic. Cordova (which Ionic is based on) does support it, but to get it working takes a lot of work, and we felt that the market for Windows Phone users is so small that it was not worth our time.

Ionic2 will be supporting Windows Universal Platform applications, so in the future this application could be migrated to the Windows Platform quite easily.



4.4 JS Tools

4.4.1 Yeoman

[37] We used Yeoman to scaffold out the push web application. There were some dependency issues, which are not uncommon with NPM. Once these were solved we used Yeoman's Angular Generator [38] to scaffold out a skeleton. Since there are so many frameworks, plug-ins and libraries in the web development community, a tool like this is needed to bring some structure to the chaos of this heterogeneity. It is analogous to a command line wizard.

Here is the output when using the Angular Generator:



```

1 Ronan:~/development/misc$ yo angular
2
3 -----
4 |         | .-----
5 |--(o)--| | Update available: 1.7.0 (current: 1.6.0) |
6 `-----' |   Run npm install -g yo to update.   |
7 ( _U_ )   |-----
8 /__A__\
9 | ~ |
10
11 `--' .-.-.-.-.-'
12     | ° | Y
13
14 -----
15 |         | .-----
16 |--(o)--| |   Welcome to Yeoman,   |
17 `-----' |   ladies and gentlemen! |
18 ( _U_ )   |-----
19 /__A__\
20 | ~ |
21
22 `--' .-.-.-.-.-'
23     | ° | Y
24
25 Out of the box I include Bootstrap and some AngularJS
26   ↳ recommended modules.
27
28 ? Would you like to use Gulp (experimental) instead of Grunt?
29   ↳ No
30
31 ? Would you like to use Sass (with Compass)? No
32
33 ? Would you like to include Bootstrap? Yes
34
35 ? Which modules would you like to include? angular-animate.js,
36   ↳ angular-cookies.js, angular-resource.js, angular-route.js,
37   ↳ angular-sanitize.js, angular-touch.js
38
39 create app/styles/main.css
40 create app/index.html
41 create bower.json
42 create .bowerrc
43 create package.json
44 create Gruntfile.js
45 create README.md

```

```

37 invoke
   ↪ angular:common:/usr/local/lib/node_modules/generator-angular/app/index.js
38 create .editorconfig
39 ...
40
41 I'm all done. Running bower install & npm install for you to
   ↪ install the required dependencies. If this fails, try
   ↪ running the command yourself.

```

As you can see straight away, we are not on the current version. The *create* commands are the files that are created, you can kind of get an idea of the structure from that.

Once *bower install* & *npm install* we ran we were met with this:

```

misc@ /Users/Ronan/development/misc
UNMET PEER DEPENDENCY grunt@>=0.4.x
|-> grunt-karma@0.12.2
    |-> lodash@3.10.1
|-> UNMET PEER DEPENDENCY jasmine-core@*
|-> UNMET PEER DEPENDENCY karma@^0.13.0 || >= 0.14.0-rc.0
|-> karma-jasmine@0.3.8
|-> karma-phantomjs-launcher@1.0.0
    |-> lodash@4.11.1
|-> UNMET PEER DEPENDENCY phantomjs-prebuilt@>=1.9

```

These kinds of errors are common and usually the answer is found in the comment thread of the projects GitHub Issue Tracker.

This can take up to five minutes to complete installing.

Once installed we have a project that has grunt, jasmine, karma, bower and an angular application ready to go! This would usually take much longer to set-up. We went out and learned each of these technologies individually before using this generator, otherwise you don't understand what's happening.

Grunt

Grunt [39] is a task runner like *Ant* or *Gulp*. It runs various tasks for you. It is needed when you have multiple steps in your project. Below are a few parts to the *GruntFile.js* from the push web



app.

Registering some tasks:

```
grunt.registerTask('test', [  
  'clean:server',  
  'wiredep',  
  'concurrent:test',  
  'postcss',  
]);  
  
grunt.registerTask('default', [  
  'newer:jshint',  
  'newer:jscs',  
  'test',  
  'build'  
]);
```

Here is one of those tasks:

```
// Empties folders to start fresh  
clean: {  
  dist: {  
    files: [{  
      dot: true,  
      src: [  
        '.tmp',  
        '<%= yeoman.dist %>/{,*/}*'  
        '!<%= yeoman.dist %>/.git{,*/}*'  
      ]  
    }  
  },  
  server: '.tmp'  
},
```

To run grunt you type one of the below commands:

```
> grunt  
> grunt build  
> grunt test
```

Running *grunt* is the same as running *grunt default*.

4.4.2 Gulp

As said before, Gulp [40] is like Grunt. The main difference is that Gulp is programmatic rather than defined. We are using it for SASS [41] here. SASS is framework that allows one to write CSS in a programmatic way.

Example from the Ionic App (*gulpfile.js*):



```
// imports

var paths = {
  sass: ['./scss/**/*.scss']
};

gulp.task('default', ['sass']);

gulp.task('sass', function(done) {
  gulp.src('./scss/ionic.app.scss')
    .pipe(sass())
    .pipe(gulp.dest('./www/css/'))
    .pipe(minifyCss({
      keepSpecialComments: 0
    }))
    .pipe(rename({ extname: '.min.css' }))
    .pipe(gulp.dest('./www/css/'))
    .on('end', done);
});

gulp.task('watch', function() {
  gulp.watch(paths.sass, ['sass']);
});

gulp.task('install', ['git-check'], function() {
  return bower.commands.install()
    .on('log', function(data) {
      gutil.log('bower', gutil.colors.cyan(data.id), data.message);
    });
});

// error stuff
done();
});
```

4.4.3 Jasmine

[42] Jasmine is just a bunch of functions that test your code. You give it an input or action and state what the result should be. Much like *JUnit*, Jasmine is unit testing.

It is a behaviour driven development framework for testing JavaScript code, and it doesn't rely on any other JavaScript frameworks.

Here is a sample test for the home controller on the push web app:



```
'use strict';

describe('Controller: HomeCtrl', function () {

  // load the controller's module
  beforeEach(module('gmitcat'));

  var HomeCtrl,
      scope;

  // Initialize the controller and a mock scope
  beforeEach(inject(function ($controller, $rootScope) {
    scope = $rootScope.$new();
    HomeCtrl = $controller('HomeCtrl', {
      $scope: scope
    });
    // place here mocked dependencies
  }));

  it('should attach a list of awesomeThings to the scope', function () {
    expect(HomeCtrl.awesomeThings.length).toBe(3);
  });
});
```

We didn't incorporate many of these tests, as we did extensive user testing and only really looked into this near the end of the project, but in future projects I would implement this from the get go.

4.4.4 Karma

[43] Karma is test-framework agnostic, what it does is spawn a web server that executes test code against source code for each of the browsers connected. You can plug any compatible testing-framework into this. Examples are: Jasmine, Mocha, and QUnit.



Once Karma is installed (via NPM) you just need to setup the config file and choose a testing framework.

Here is part of our Karma Config file for the Push Web App:

```
module.exports = function(config) {
  config.set({

    // list of files / patterns to load in the browser
    files: [
      // bower:js
      'app/scripts/**/*.js', 'test/mock/**/*.js', 'test/spec/**/*.js'
    ],
    port: 8080,

    // - Chrome, ChromeCanary, Firefox, Opera, Safari (only Mac),
    //   ↪ PhantomJS, IE (only Windows)
    browsers: [
      'PhantomJS'
    ],

    plugins: [
      'karma-phantomjs-launcher', 'karma-jasmine'
    ],
    singleRun: false,
    colors: true,

    // level of logging
    // possible values: LOG_DISABLE || LOG_ERROR || LOG_WARN ||
    //   ↪ LOG_INFO || LOG_DEBUG
    logLevel: config.LOG_INFO,
  });
};
```

4.4.5 Bower

Bower [44] is a package manager like *Maven* or *NPM*. It is often used in conjunction with NPM. The main difference is the way it loads packages into an application. Bower loads items via dependency injection in the *app.js* file. We can see below the *angular-md5* package being loaded in along with some default.



```
angular
.module('canteen', ['ionic', 'ionic.service.core',
  ↪ 'canteen.controllers', 'canteen.services', 'ngCordova',
  ↪ 'ngMessages', 'angular-md5']);
```

Below is the list of dependencies for the project, these get install when the command: *bower.json* is run.

bower.json file from the Ionic app:

```
{
  "name": "GmitCatering",
  "private": "true",
  "devDependencies": {
    "ionic": "driftyco/ionic-bower#1.1.0",
    "platform.js": "platform#~1.3.0",
    "ionic-platform-web-client": "~0.2.1"
  },
  "dependencies": {
    "angular-md5": "~0.1.10"
  }
}
```

4.4.6 NPM

NPM [45] is the main package manager for nearly all JavaScript projects, including Ionic. It is similar to Bower.

Below is the list of dependencies for the Ionic application:



```
{
  "name": "gmitcatering",
  "version": "1.0.0",
  "description": "GmitCatering: Purchase food with ease",
  "dependencies": {
```

```
"cordova": "^5.3.3",
"gulp": "^3.5.6",
"gulp-concat": "^2.2.0",
"gulp-minify-css": "^0.3.0",
"gulp-rename": "^1.2.0",
"gulp-sass": "^2.0.4"
},
"devDependencies": {
  "bower": "^1.3.3",
  "gulp-util": "^2.2.14",
  "shelljs": "^0.3.0"
},
"cordovaPlugins": [
  "cordova-plugin-whitelist@1.0.0",
  "phonegap-plugin-push"
],
"cordovaPlatforms": [
  "android",
  "ios"
]
}
```

Here we are loading some NPM modules into the gulp file for use:

```
var gulp = require('gulp');
var gutil = require('gulp-util');
var bower = require('bower');
var concat = require('gulp-concat');
var sass = require('gulp-sass');
var minifyCss = require('gulp-minify-css');
var rename = require('gulp-rename');
var sh = require('shelljs');
```

4.4.7 BASH

As you can see throughout the document so far, we have used the terminal for many tasks, and even created some scripts. BASH [46] is invaluable to the development workflow, being able to group various commands into a file and just run the file is a great way to encapsulate processes.



4.5 Web App Alternatives

4.5.1 Cordova

Cordova [8] is an amazing project which gives us the ability to develop a mobile application from a single web view. Cordova allows us to access hardware on the device that would otherwise be very tricky.



They support:

- iOS
- Android
- Windows Phone
- BlackBerry
- UbuntuPhone
- FirefoxPhone
- LGwebOS
- FireOS

Ionic is based on Cordova.

4.5.2 PhoneGap

Adobe own PhoneGap [47], when they purchased the Cordova project, some aspects had to be kept open source. Adobe has created some great services and proprietary APIs for Cordova.



4.5.3 JQueryMobile

JQuery [9] Mobile helps you create JavaScript interfaces for mobile apps and mobile web apps. It can be used with PhoneGap or many other frameworks. Check this website out to view all available frameworks.

I probably would of went for this library mixed with Cordova if Ionic was not available. Although it is way behind in many aspects.

4.5.4 Xamarin

[48] Xamarin is a great choice for developers who want to create cross platform applications in C#, or for those who really don't like the JavaScript world. You code your Windows Universal Platform application up in C# and Xamarin will port your **Core Logic** to Android and iOS. You still have to write the View layer and Native API layers yourself though.



We found after talking to the community on the forums that it was quite buggy and there wasn't much support in some areas. For instance the ported code is usually very different then if you were to create it yourself in Android or iOS so it's very hard to ask for help as all of your code will be completely custom.

4.6 Push Web Application

The Push Web App uses AngularJS, NodeJS, ExpressJS, Cloudant and Heroku. We chose to use these technologies as we needed to quickly create a server to facilitate push notifications. We did not realise that we would need to store a list of device tokens, and it was too late to update the SQL Schema.

4.6.1 CouchDB/Cloudant

IBM's cloudant [49] NoSQL database solution use CouchDB [50] as the database of choice. It has a fantastic user interface and is a joy to use. IBM also guarantee high levels of performance and very low downtime (SLA).



The other alternative I thought of using was Google's Firebase. I think these are both very good NoSQL DB as a service solutions. Another one that has been around for awhile is Mlab (MongoDB as a service).

4.6.2 NodeJS

NodeJS [51] which I talk about in detail in my literature review [29] in detail, has taken the web by storm the past couple of years. It has allowed developers to bring JavaScript from the client side to the server side using



Google's JavaScript interpreter (V8 Engine), this V8 Engine heavily optimizes the JS code and uses JIT compilation to give near native performance, this coupled with the event based nature (which removes the need for threads) makes it a strong choice for server side development.

4.6.3 ExpressJS

ExpressJS [52] is a web development framework based on the NodeJS environment, it abstracts many common tasks into functions, like setting up a web server or HTTP RESTful routes.

express

Here is an example of a simple ExpressJS HTTP Server with RESTful home route:

```
var express = require('express');
var app = express();

app.set('port', (process.env.PORT || 5000));

// home page
app.get('/', function(req, res){
  var result = {"message": "Welcome to the gmit catering api"};
  res.contentType('application/json');
  res.send(JSON.stringify(result));
});

app.listen(app.get('port'), function() {
  console.log('Node app is running on port', app.get('port'));
});
```

4.7 GitHub

We used GitHub [53] throughout our project in order to utilize source control and keeping track of versions. We found that it was very useful in that we could have separate repositories for the different parts of the project. When we did work on the same repository we each had our own branch to which we would merge intermittently into the *master* branch. We also used the *issue tracker* to raise any issues/bugs we noticed on each others projects, or to mention an enhancement that could be incorporated.



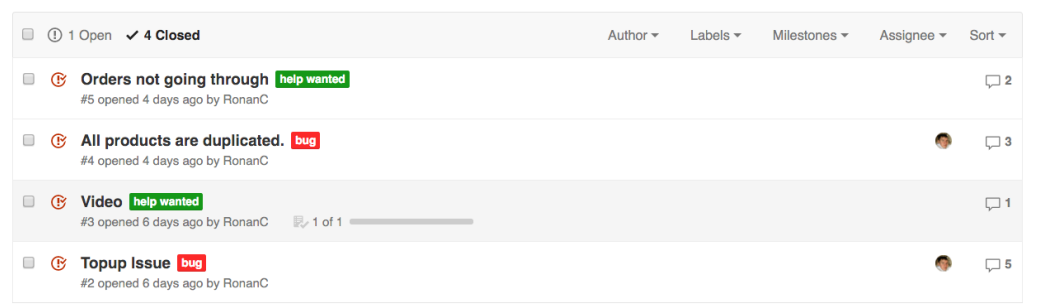


Figure 4.4: Issue Tracker

We also created an organisation, which acts as an entity. We forked all our repositories there and used it as a central hub for the overall project [1].

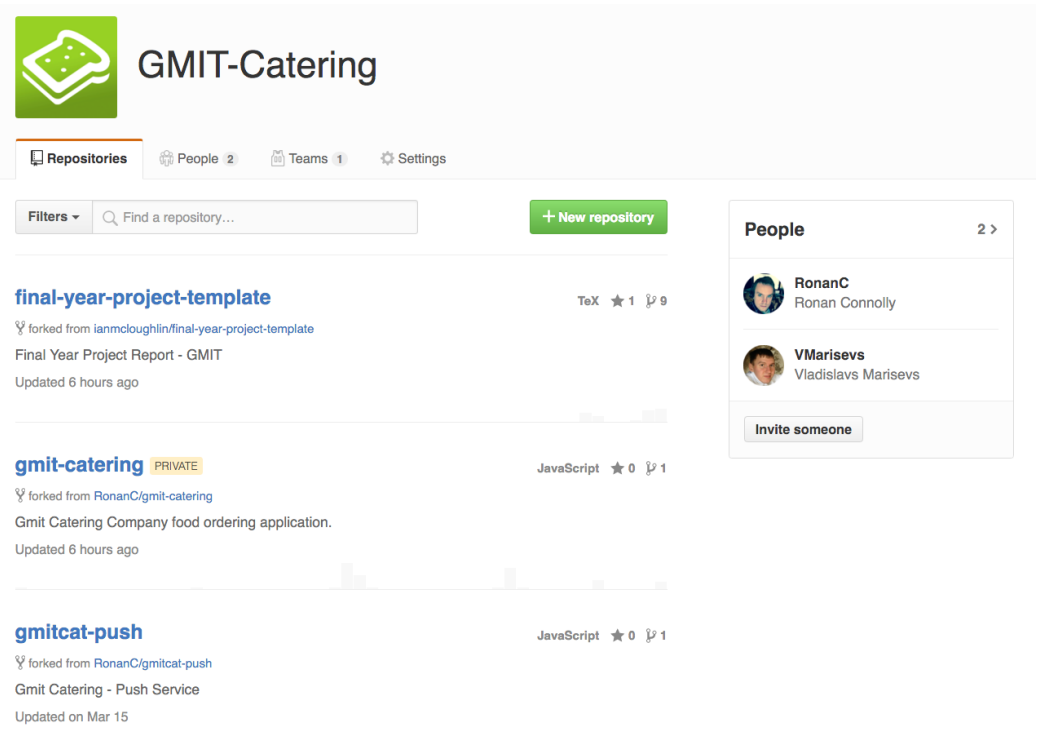


Figure 4.5: GMIT Catering GitHub Org

4.8 JSON REST interface architecture

We chose to use a RESTful architecture [54] with JSON [55] because it is very simple and easy to understand and use. It's reliable and has been battle tested with time. Both setting up and querying a RESTful system is trivial.

express

4.8.1 HTTP Requests

[56] In order to query our RESTful system we used HTTP requests. These are very intuitive, they were designed to be common sense and simple to understand. Here is a list of HTTP Request Type:

express

Name	Function	Example
Get	Gets the data at that URL	/users/Ronan OR /users/
Put	Updates the data at that URL	/users/Ronan
Post	Saves a new user to a URL	/users/
Delete	Deletes the data at that URL	/users/Ronan

Table 4.1: HTTP Requests

4.8.2 Custom API

We created a Custom HTTP RESTful API on our web application. It contains various routes which are discussed in the web app section of the system design and illustrated via angular HTTP code snippets in the mobile app section of the system design below.

We found this to be very quick and easy to set up and would recommend it to others and we ourself would use it in the future without a doubt.

Chapter 5

System Design

5.1 Database

5.1.1 Purpose

This section describes how the database that will support the system with details of the logical and physical definitions. It also provides the functional and non-functional usage of the tables, considerations and requirements.

The database design for the system is composed of definitions for db objects derived by mapping entities to tables, attributes to columns, unique identifiers to unique keys and relationships to foreign keys.

Further in this document you will read the integration aspects of the Database with the Web Application and to understand them I want to explain some word definitions that will be used.

- Customer is a physical person who uses smartphone application.
- User is a physical person who operates with management website.
- Food is entire product of this business.
- Voucher is a printed card with unique code that allow customers to top up their account balance.

5.1.2 Procedures and Functions

One of good reasons to use stored procedures is the added layer of security that can be placed on the database from the calling application. The risk of

a attacker using the account to run a stored procedure that has been written by you is far safer than having the user account have full insert, update and delete authority on the tables directly. Another advantage to use stored procedures is the data functionality that is separated from the application making it easier to manage, document and maintain. They also improves the performance for example to make a payment I am doing just one call and DBMS can process it, which is more efficient than do complete processing on the php side.

- **Verify Customer**

This procedure takes two parameters login name and password, then parses it and returns the result. This procedure is used to connect Ionic application with DBMS and verify customer's credentials. If the password and login matches database returns complete details about current customer. This information can be displayed in the mobile application.

- **Block Customer**

If system administrator sees that some user tried to enter invalid voucher id too many times, he can block this person. To keep a track of all invalid customer inputs I decided to add a basic counter, so that administrator can keep an eye on the malicious users that tries to insert voucher code permutations.

- **Change customer's password**

One of the common procedures when user is allowed to change their password if they are authorized.

- **Insert customer**

This procedure let people register their account as customer of the ordering system. To do that they have to insert their details, once that is done they can start using mobile application.

- **Recover customer's password**

In order to recover the password, user have to insert their email address and login. It will trigger the scenario of generating new temporary password for this user and will send it out to their email address. The procedure of sending emails from web application uses SMTP protocol, by default it is linked with gmail account. The settings are stored statically in configuration file **/config/autoload/email.local.php*. If login and email exists in the database it will save new password.

- **Food insert**
This procedure takes all food information, generates new identification number and returns it.
- **Food update**
In order to update the food, we are not overriding the record, but creating a new one with unique id and setting the original '*active*' column to *false*. This technique let us to keep track of all changes and prevent updates on the old orders.
- **Delete Food**
If user requires to remove the food, he is using this procedure. It just sets '*active*' column to false and it will remove current food from list. But in case there was any orders with this food it will save a link and old details.
- **Create order**
This procedure will create an order in case it receives all valid information and returns new generated id or null which mean something is invalid.
- **Food Order Insert**
When order is created and id returned, it starts filling it with products.
- **Order make payment**
This procedure compares the balance on the account and if it is sufficient to make a payment it will deducts the money from user and mark the order as paid. These operations are separate to prevent incompleteness of the transaction.
- **Order price**
This procedure calculates sum of order items and returns it.
- **Order cancel payment**
To cancel order payment we have to calculate sum of item prices and return them to customers balance.
- **Create voucher**
This function generates new voucher and places it in database.
- **Topup customer account**
Current transaction marks the voucher with customers account identification and adds money to their balance. After that this id is no longer valid for next transaction.

5.1.3 Tables

Order System

Ionic application users table:

Column	Type
id	binary(16)
gnumber	int(11)
email	varchar(40)
password	char(41)
cash	decimal(13,2)
active	tinyint(1)
created	datetime
updated	timestamp
name	varchar(40)
surname	varchar(40)
address	text
mobile	varchar(10)
cheat	int(10)

Table 5.1: Customers table

Vouchers table:

Column	Type
id	varchar(16)
amount	decimal(13,2)
used	tinyint(1)
customer	binary(16)
activated	datetime
created	datetime
modified	timestamp

Table 5.2: Vouchers table

Orders table:

Column	Type
id	varchar(16)
customer	binary(16)
comments	medium text
paid	tinyint(1)
collectionTime	time
collectionDate	date
created	datetime
modified	timestamp

Table 5.3: Orders table

Food table:

Column	Type
id	bigint(20)
name	varchar(40)
description	medium text
price	decimal(13,2)
type	enum('drink', 'roll', 'crisps', 'deal', 'misc')
picture	varchar(128)
active	tinyint(1)
created	datetime
modified	timestamp

Table 5.4: Food table

Food ordered table:

Column	Type
order_id	varchar(16)
food_id	bigint(20)
count	smallint(5) unsigned
created	datetime
modified	timestamp

Table 5.5: Food Ordered table

Administration

Web application users table:

Column	Type
id	unsigned int(10)
email	varchar(100)
password	varchar(100)
status	enum('Y','N')
created	datetime
modified	timestamp

Table 5.6: Users table

Web application resource table:

Column	Type
id	unsigned int(10)
resource	varchar(100)

Table 5.7: Resource table

Web application permission table:

Column	Type
id	unsigned int(10)
permission	varchar(100)
resource	unsigned int(10)

Table 5.8: Permission table

Web application role table:

Column	Type
id	unsigned int(10)
name	varchar(100)
status	enum('Active', 'Inactive')

Table 5.9: Role table

Web application role permissions table:

Column	Type
id	unsigned int(10)
role	unsigned int(10)
permission	unsigned int(10)

Table 5.10: Role permissions table

Web application user role table:

Column	Type
id	unsigned int(10)
user	unsigned int(10)
role	unsigned int(10)

Table 5.11: User role table

Management

Time available for ordering food table:

Column	Type
id	unsigned int(10)
weekday	tinyint(1)
open	time
close	time
active	enum('Y', 'N')
created	datetime
modified	timestamp

Table 5.12: Opening time table

This table stores information about available collection time:

Column	Type
id	unsigned int(10)
collection	time
active	enum('Y', 'N')
created	datetime
modified	timestamp

Table 5.13: Collection time table

5.2 System Administration and Management Web Application

In this section I will talk about functionality of this system. The amount of data that we are storing in our database is large and this system can be extended with minimal changes in the database structure. As you already seen what information we store in the system you can determine that customers details are stored separately from system users.

5.2.1 Admin Authentication

To authenticate users I am using ZF2 Auth ACL module. It is an open source module which can be accessed on GitHub. It contains 5 tables that describes website users, roles and route permissions. To improve systems security I made several changes into this module. [57]

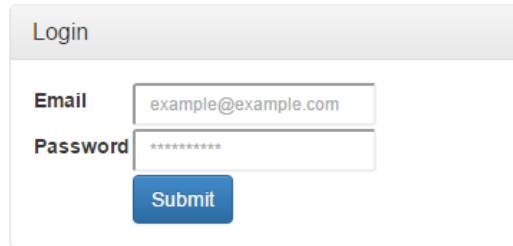

 A login form titled "Login" with a light gray header. Below the header, there are two input fields: "Email" containing "example@example.com" and "Password" containing "*****". A blue "Submit" button is positioned below the password field.

Figure 5.1: Login container

5.2.2 Voucher system

This system required operations with money and we come up with great solution, to implement voucher system. Customers have to buy vouchers to top up their balance on the account. Cards has to be printable and in order to do so I am generating a *.pdf file that can be printed or saved for future printing. Voucher contains short information about company and *QR code* that allow to scan by smartphone to simplify code inserting process. Figure 5.2 shows an example of printed voucher.



Figure 5.2: Voucher example

To design this voucher and make it printable from web browser, I am using *DOM pdf* generator. It allows to display remote pictures and specify fixed positions for text. *QR Code* generating module submits the *HTTP*

request to Google API and receives a dynamic link to an image, which later in page generation I am using to add into *pdf* file.

Lookup vouchers

5.2.3 Order system

Placed orders

When customer using their application places the order and transaction fully completed, then this order appears on the *home page* as this system is designed to be pre order. At the certain time before collection system administrator prints out the list of food that has to be prepared for customer. In order to make a good printable layout it creates a **.pdf* file using *JavaScript* module called PDF make [58]. It uses Document definition object and you do not have to calculate position manually as like in other pdf generation libraries. For example it can be as simple as:

```
var docDefinition = {
  content: [
    // if you don't need styles, you can use a simple string to define a paragraph
    'This is a standard paragraph, using default style',

    // using a { text: '...' } object lets you set styling properties
    { text: 'This paragraph will have a bigger font', fontSize: 15 },

    // if you set pass an array instead of a string, you'll be able
    // to style any fragment individually
    {
      text: [
        'This paragraph is defined as an array of elements to make it possible to
        { text: 'restyle part of it and make it bigger ', fontSize: 15 },
        'than the rest.'
      ]
    }
  ]
};
```

When you have prepared your document-definition-object, you can run the function that will parses it and generated *pdf*:

```
// open the PDF in a new window
pdfMake.createPdf(docDefinition).open();

// print the PDF (temporarily Chrome-only)
pdfMake.createPdf(docDefinition).print();

// download the PDF (temporarily Chrome-only)
pdfMake.createPdf(docDefinition).download('optionalName.pdf');
```

After order has been processed and collected, administrator has to go back to this system and mark collected orders. To simplify this process there is a check box beside each order and when all collected orders are marked user have to click *Collected* button to make an update. In case of emergency some customer canceled their order administrator is allowed to cancel selected order and return money to customers account. In that case this order will not appear in later reports.

Order Reports

In order to make an accounting report in the orders tab, administrator can select two types of reports:

- **Daily report**
- **Periodic report**

To view daily report user should specify the date in *mm/dd/yyyy* format and press submit button as you can see in figure 5.3.

The screenshot shows the GMIT Catering Co. website with a navigation bar containing links: Home, Orders, Kitchen, Accountancy, Vouchers, Settings, Accounts, and Logout. Below the navigation bar, there are two main report sections: 'Day Report' and 'Periodical Reports'. The 'Day Report' section has a 'Date:' input field with a placeholder 'mm/dd/yyyy' and a green 'Submit »' button. The 'Periodical Reports' section has a 'From:' input field with the value '09/01/2015', a 'To:' input field with a placeholder 'mm/dd/yyyy' and a dropdown arrow, and a green 'Submit' button. A date picker calendar is open, showing the month of April 2016, with the year 2016 selected. The calendar shows days from Jan to Dec. At the bottom of the page, there is a copyright notice '© 2016' and a footer 'served. Powered by Zend Framework 2'.

Figure 5.3: Daily/Periodic report page

In the new page it will display report based on the period or specific date. Where user can go further and click order details, see customer's information or print this page as a report, see figure 5.4.

Zf GMIT Catering Co. Home Orders Kitchen Accountancy Vouchers Settings Accounts Logout					
Orders 01/01/2016 - 13/04/2016					
Print					
Order Id	Customer	Order Date	Price	Collected	
11E5C50C4D08E0C087ED2DC144108363	317413	27-01-2016	7.00 €	✓	Details
11E5B9EA7245346AB95FEFBEE74CC5F8	321312	13-01-2016	6.95 €	✗	Details
Total:			13.95 €	VAT(23%):	3.21 €
© 2015 - 2016 by VMarisevs GMIT. All rights reserved. Powered by Zend Framework 2					

Figure 5.4: Daily/Periodic report page

Order details will display full information about customer and items ordered. It can be printed separately as invoice (see figure 5.5) or proceed to go and view customer personal details.

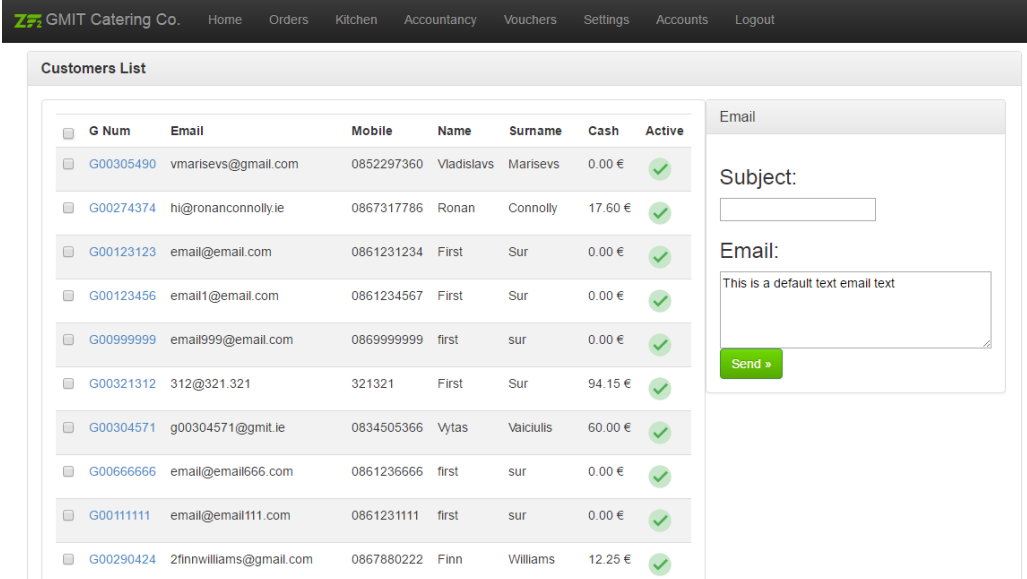
Zf GMIT Catering Co. Home Orders Kitchen Accountancy Vouchers Settings Accounts Logout								
Print								
Invoice Nr.: 11E5C50C4D08E0C087ED2DC144108363								
Customer ID.: 11E5C50BC1E336BC87ED2DC144108363								
Robert Bryant G00317413 g00317412@gmit.ie 0851545250								
Details								
Order								
default.jpg	24151838022434910	Still water	1.00 €	1	DRINK			
default.jpg	24151838022434919	Tayto - S+V	1.50 €	1	CRISPS			
Tuna.jpg	24151838022434926	Tuna	4.50 €	1	ROLL			
			Order Date: 27/01/2016 ✓ Collection Time: 18:00:00 Created: 27/01/2016 Updated: 27/01/2016 Total: 7.00 €					
Comments								
na								
© 2015 - 2016 by VMarisevs GMIT. All rights reserved. Powered by Zend Framework 2								

Figure 5.5: Daily/Periodic report page

5.2.4 Customer and accountancy

Customer details

Authorized person can view all registered customers in accountancy tab and by clicking *view* button. It will display a table with all users see in figure 5.6



Customers List

<input type="checkbox"/>	G Num	Email	Mobile	Name	Surname	Cash	Active
<input type="checkbox"/>	G00305490	vmarisevs@gmail.com	0852297360	Vladislavs	Marisevs	0.00 €	✓
<input type="checkbox"/>	G00274374	hi@ronanconnolly.ie	0867317786	Ronan	Connolly	17.60 €	✓
<input type="checkbox"/>	G00123123	email@email.com	0861231234	First	Sur	0.00 €	✓
<input type="checkbox"/>	G00123456	email1@email.com	0861234567	First	Sur	0.00 €	✓
<input type="checkbox"/>	G00999999	email999@email.com	0869999999	first	sur	0.00 €	✓
<input type="checkbox"/>	G00321312	312@321.321	321321	First	Sur	94.15 €	✓
<input type="checkbox"/>	G00304571	g00304571@gmt.ie	0834505366	Vytas	Vaiciulis	60.00 €	✓
<input type="checkbox"/>	G00666666	email@email666.com	0861236666	first	sur	0.00 €	✓
<input type="checkbox"/>	G00111111	email@email111.com	0861231111	first	sur	0.00 €	✓
<input type="checkbox"/>	G00290424	2finnwilliams@gmail.com	0867880222	Finn	Williams	12.25 €	✓

Email

Subject:

Email:

This is a default text email text

Figure 5.6: Customers list

In this page we can select customers which we want to inform in updates and send them an email from default email. When person is found we can view his details as you can see in 5.7 figure.

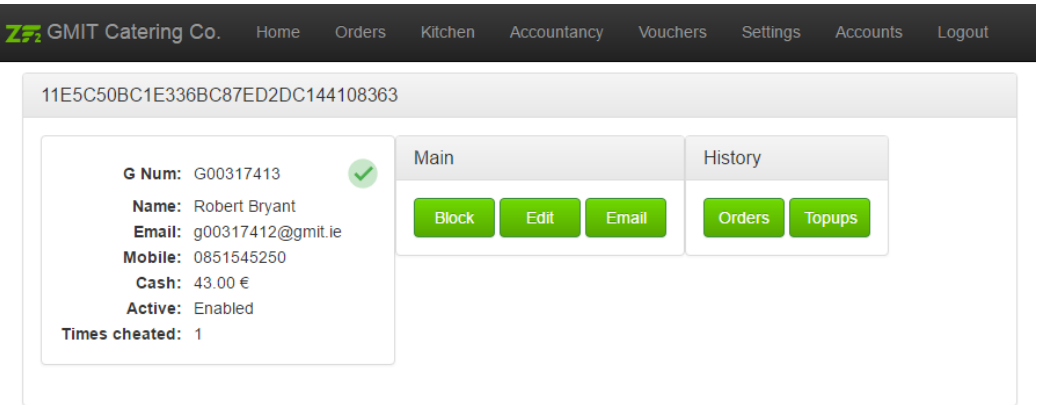


Figure 5.7: Customers details

In this section we are able to do such things like send an email to customer, block him or edit his personal details in case they are wrong. We can determine that this cheated once, this counter increments only when person tries to enter wrong QR code as their voucher.

Customer history

User has two types of histories. First one keeps a track of all user orders, so in future it can suggest user to make similar orders and history of user balance top up.

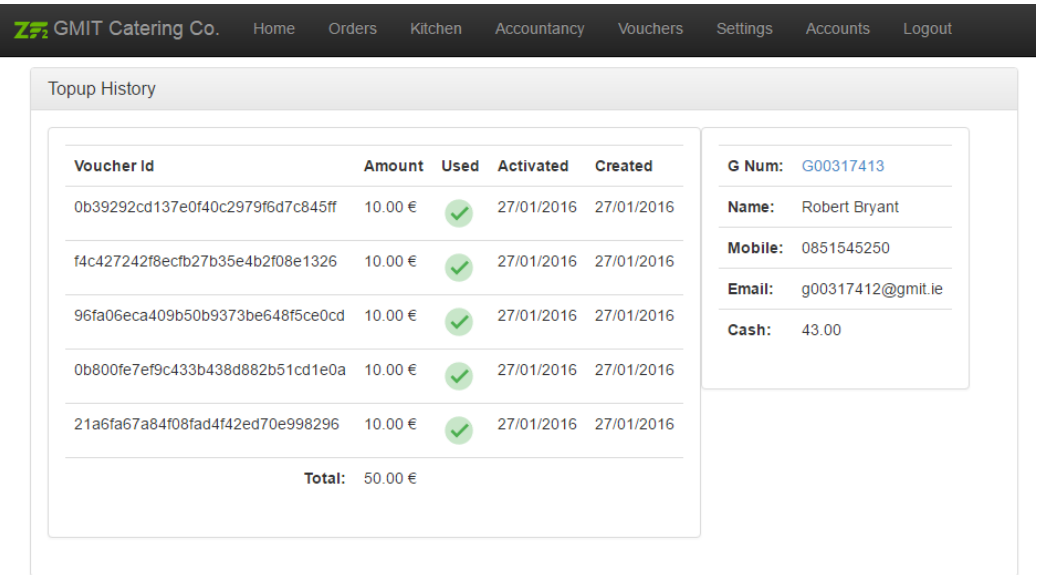


Figure 5.8: Customers top up history

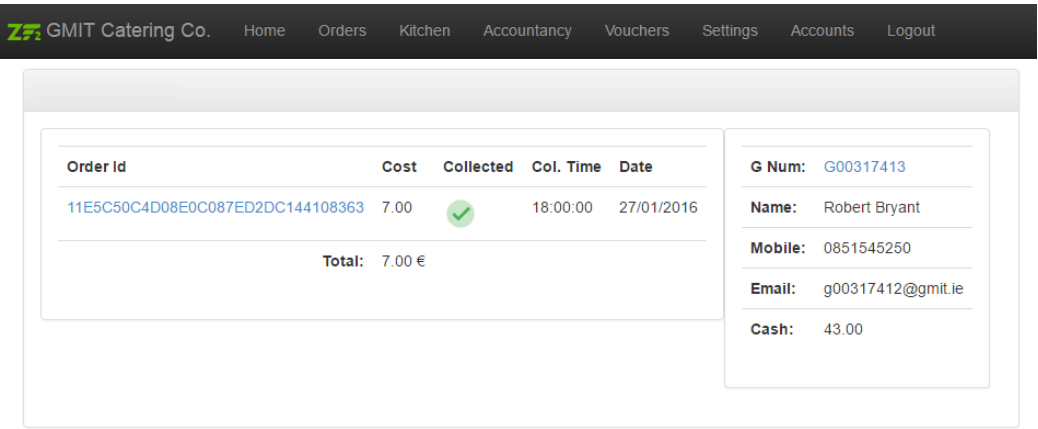


Figure 5.9: Customers order history

5.2.5 Ionic routes & functions

- Authenticate customer
- Recover customers password
- Change customers password
- Sign Up new customer
- Top Up customer balance
- Approve customer's order and purchase
- Show customers history
- Show available food

5.2.6 PDF make

[58]

5.3 Mobile App

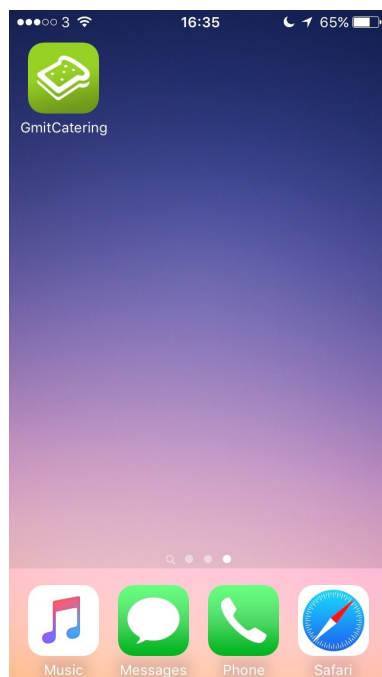
Here we will go through all the different sections of the mobile application.

There are three tabs in the application, Orders, About and Account. These are located on the bottom for iOS and on the top for Android. This gives a native experience, look and feel to the user.

I will refer to each view as a page below.

5.3.1 Sections

A description of each part of the mobile application.



We can see the application icon on the **home screen**.

Initially we were going to try and incorporate the GMIT logo into the design but that seemed too tacky.

We used the GMIT Catering green colour with a slight gradient, then overlaid a simple sandwich icon in white.

Figure 5.10: Home Screen

The first screen you encounter after tapping on the application is the **splash screen**, which is just a larger version of the application icon. You usually only see this the first time you open the application or if you are running on an older device.

Once loaded you will see the **login** page, where we ask for your ID and Password. This form contains validation and error messages above the input boxes. You may only tap the **Sign In** button once the form is valid. We have incorporated this into all forms in the application.

Once logged in you will stay logged in, even after closing the application, you can log out from within the application.

You also have access to the **Sign Up** and **Reset Password** pages.

gmit catering

Required field missing

ID: G/C Number

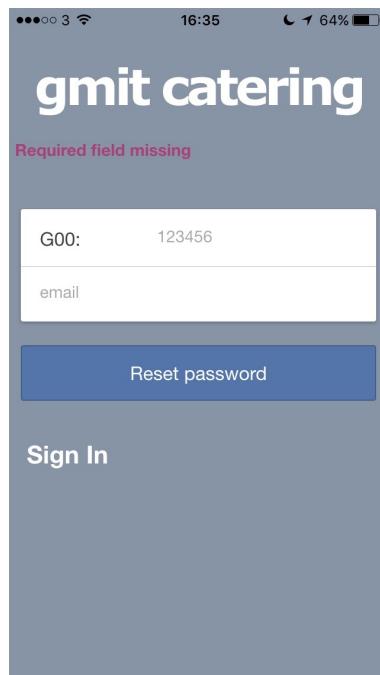
Password *****

Sign in

Sign Up

Reset Password

Figure 5.11: Login

A screenshot of a mobile application interface for "gmit catering". The status bar at the top shows signal strength, 3G, time 16:35, location, and 64% battery. The app header displays the logo "gmit catering". Below the header, a red error message "Required field missing" is shown. The form consists of two input fields: "G00:" with the value "123456" and "email" which is empty. Below these fields is a blue button labeled "Reset password". At the bottom of the form area is a "Sign In" link.

If you forget your password you can enter your ID and Email into this form, and a new password will be emailed to you.

You can then edit your password from within the application later on.

Figure 5.12: Password Reset

The **Sign Up** page contains a few basic instructions above the form, then some error messages, and finally the form itself.

There is extra form validation here, such as regular expressions to make sure that all characters are valid, for instance only alpha characters are allowed in the name fields, no spaces or apostrophes. Once you sign up you will automatically be logged in.

Password input on all forms are hashed with **MD5** instantly. We may input any characters as the password will be hashed into valid characters. It also means that there passwords are not stored in plain text, MD5 hashes cannot be un-hashed.

We felt this was a necessary step for security and privacy purposes.

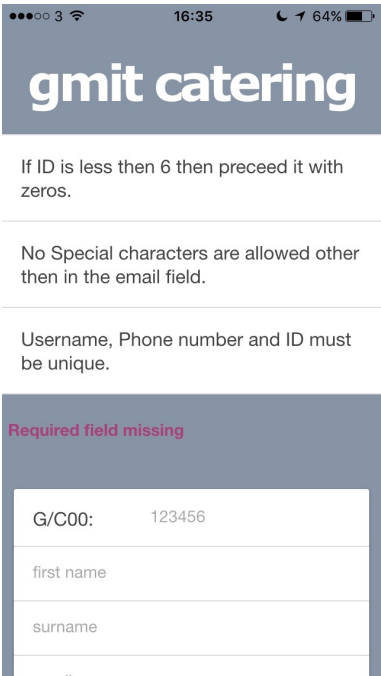


Figure 5.13: Sign Up

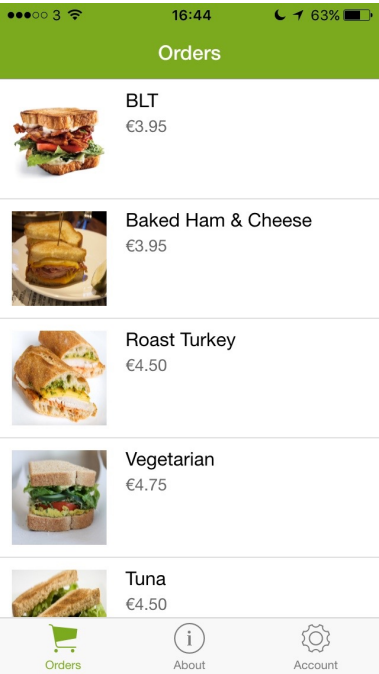


Figure 5.14: Orders

From the **orders** page we can see all the sandwiches that are available.

This data is taken from our web server and can be changed instantly.

When we tap on a sandwich we are brought to the order page for that sandwich.

We can view more information about the chosen sandwich from here.

We must then choose a bread and order time.

There are optional requirements such as: a drink, crisps, up to four miscellaneous items and a text comment. The miscellaneous items mostly include add ons for the sandwich such as re onion, lettuce and things of that nature.



Figure 5.15: Order Details

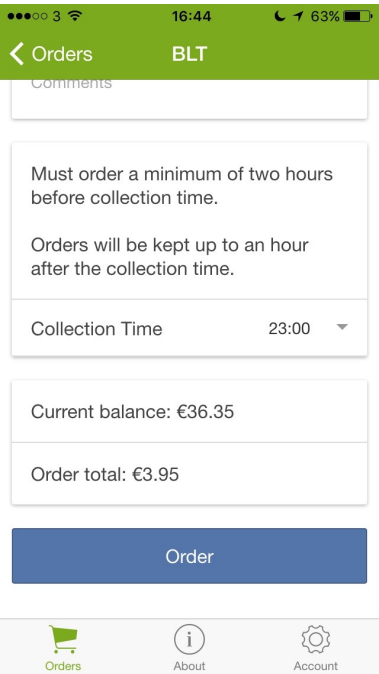


Figure 5.16: Order Details

At the bottom of the **order details** page we can see our current balance, the total cost for our order and a button to order it. We can also see some details about order times.

The **Order button** is only available during valid times and when we have sufficient balance.

Once we press the **Order button** we are given a prompt showing us how much money we will have in our account after the order.

This is important as the user may press the **Order button** by accident. It also gives the user some time to go back and choose extra items.

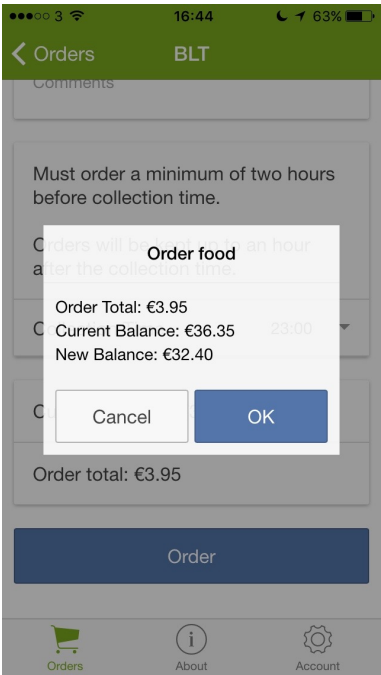


Figure 5.17: Order Confirm

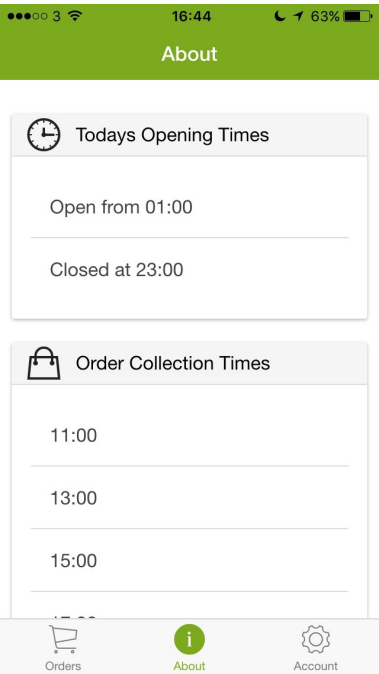


Figure 5.18: Times

On the **About** page we can see today's opening times and order collection times.

We can only order between opening times and may only choose to collect sandwiches at the order collection times allocated.

Further down in the **About** page we can view information on the **Food Zone**, **Espresso Bar** and **Cafe Foyer**.

These are the three **GMIT Catering Company** locations throughout the college.

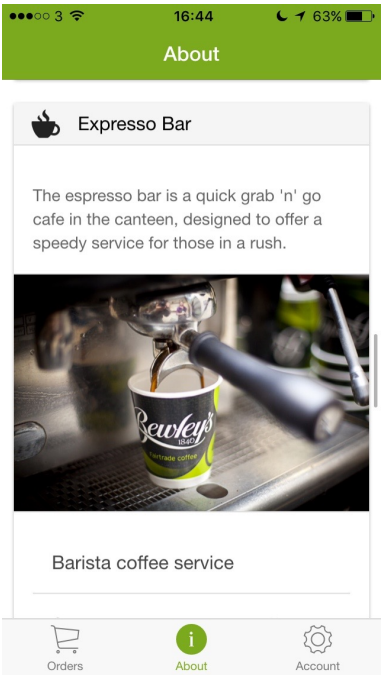


Figure 5.19: Espresso Bar

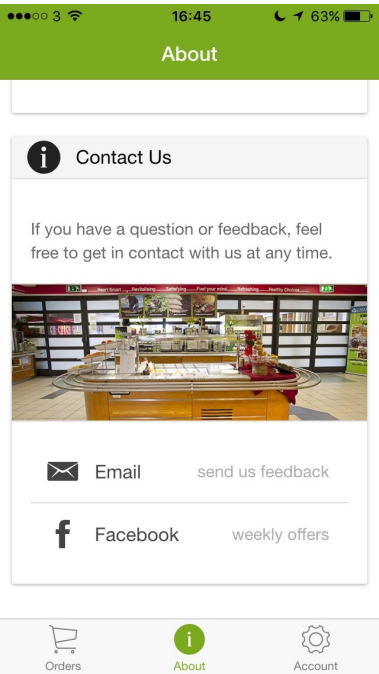


Figure 5.20: Contact Us

At the very bottom of the **About** page we can see the **Contact Us** card.

Which gives the user a simple way to email the **GMIT Catering Company** or to interact with them on Facebook.

This will allow the company to gain more Facebook followers also.

From the **Account** page we can see the users cash.

We also have options to:

- Top Up (QR Reader)
- Log out
- View our information
- View our history
- Change our password

Grouping all these actions into the **Account** page keeps the UI clean and organised.

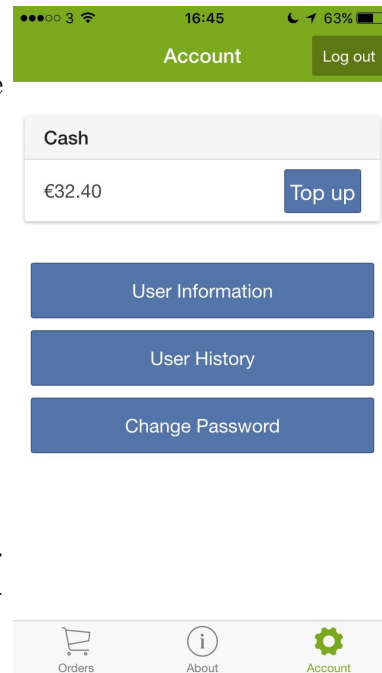


Figure 5.21: Account

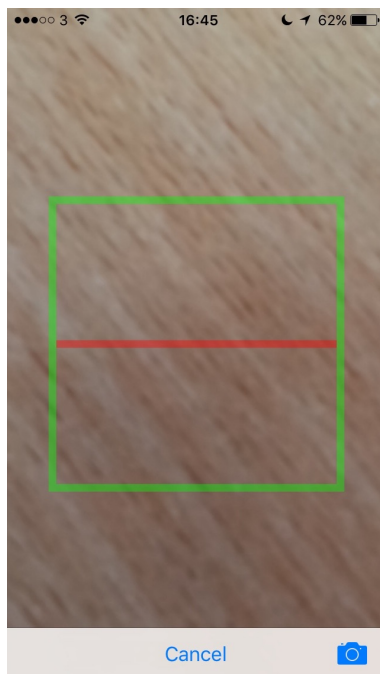


Figure 5.22: QR Reader

When you tap on **Top up** you are given the prompt to purchase a top-up card from the **GMIT Catering** checkout or else continue to scan your card.

A QR reader opens up form within the application and you can scan your card.

On completion you will be let known if the top-up went through successfully or not.

You will then be brought to the User Information page where you can see your cash balance updated (or not).

In the **User History** page you can view all past orders.

You can also see if it was marked as collected.

If the order was cancelled by the **GMIT Catering** staff it will also be marked as collected. Collected basically means it has been dealt with by the staff.

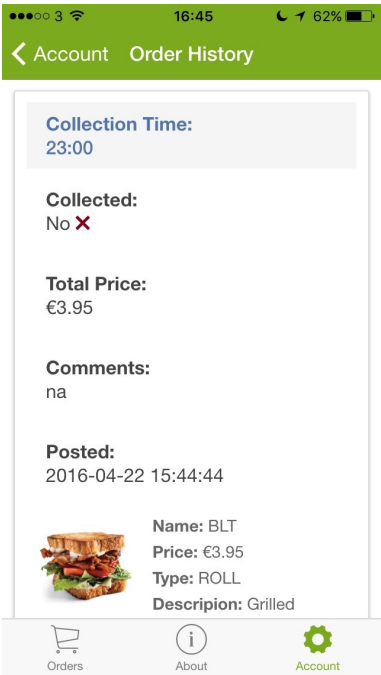
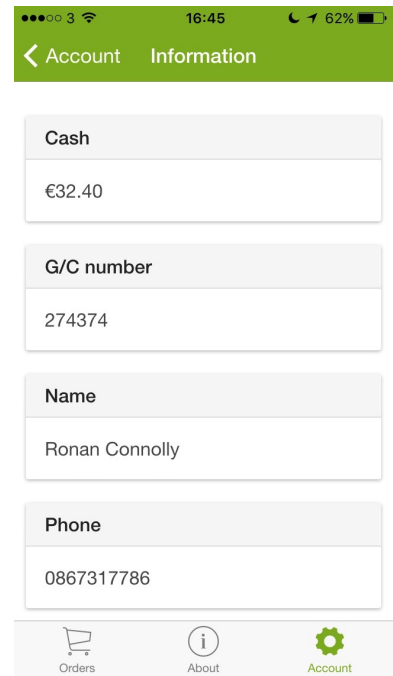


Figure 5.23: Order History



From the **User Information** page we can see the users:

- Cash
- ID
- Name
- Phone

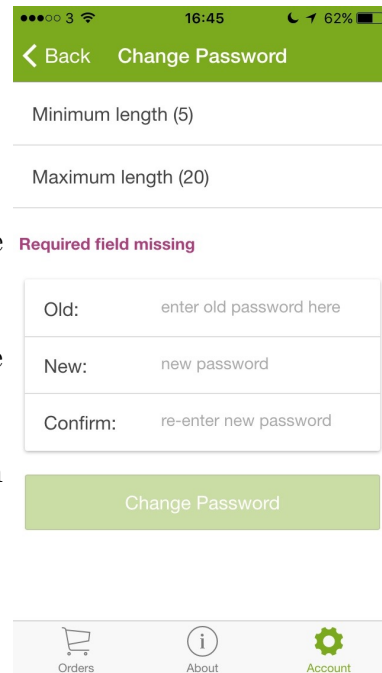
This can easily be extended to show other information.

Figure 5.24: User Information

From the **Change Password** page we can change our password.

This is very useful for when we email the user a temporary password.

As you can see we have full form validation here.



Minimum length (5)

Maximum length (20)

Required field missing

Old: enter old password here

New: new password

Confirm: re-enter new password

Change Password

Orders About Account

Figure 5.25: Change Password

5.3.2 Use Cases

There are many use cases for this application.

First of all it create awareness of the GMIT Catering Company brand within the college. There is various information about the different areas within the college that the company runs.

There are links to email and Facebook which lets the users interact with the company.

Users can view opening times, which is often a reason why many students do not visit the canteen, due to not knowing it's open.

With push notifications the company can get messages straight to the customer each day, giving various offers, deals and promotions.

Users can pre-order food which is a very popular thing to do now a days, especially due to the fact that it can keep you structured and make sure you

follow a certain eating habit, without the evils of hunger to persuade you into binging.

On top of this, it allows customers to skip the queue and for the staff to prepare sandwiches during quite hours (but before rush hours).

5.3.3 Code Snippets

Here we will list some interesting code snippets from the mobile application.

The Ionic/Angular markdown/code for the **Login** page.

This includes the form validation, which uses **ng-messages**.

```
<ion-view view-title="gmit catering" hide-nav-bar="true"
↳ class="splash-page login-page">
  <ion-content>
    <div class="padding logo">
      <h1>gmit catering</h1>
    </div>

    <div>
      <!-- errors -->
      <ng-messages for="loginForm.$error"
↳ class="error-text">
        <ng-message when="required">Required field
↳ missing</ng-message>
        <ng-message when="minlength">Minimum Length not
↳ met</ng-message>
        <ng-message when="maxlength">Maximum Length not
↳ met</ng-message>
      </ng-messages>
    </div>

    <form name="loginForm" class="form"
↳ ng-submit="userLogin(loginForm, loginForm.username,
↳ loginForm.password)" novalidate>
      <div class="card">
        <div class="list">
          <label class="item item-input">
            <span class="input-label">ID:</span>

```

```

        <input name="name" type="tel"
        ↪ ng-model="loginForm.username"
        ↪ placeholder="G/C Number"
        ↪ minlength="6" maxlength="8"
        ↪ required />
    </label>
    <label class="item item-input">
        <span
        ↪ class="input-label">Password</span>
        <input name="pass" type="password"
        ↪ ng-model="loginForm.password"
        ↪ placeholder="*****"
        ↪ minlength="3" maxlength="20"
        ↪ required />
    </label>
</div>
</div>
</div>

<div class="padding-left padding-right">
    <input type="submit" value="Sign in"
    ↪ class="button button-block button-balanced"
    ↪ ng-disabled="loginForm.$invalid" />
</div>
</form>

<div class="padding splash-links splash-margin">
    <a href="#/signup" class="a-no-style">Sign Up</a>
</div>

<div class="padding splash-links splash-margin">
    <a href="#/reset" class="a-no-style">Reset
    ↪ Password</a>
</div>
</ion-content>
</ion-view>

```

Next we can see the **userLogin** function that the above form calls on, here we are inside the **Login Controller**.

If the form is valid then we pass the request onto the **auth** function in the **User service**, we also have some error messages.

We are using the **.then** functionality of the Angular service function

User.auth to allow us to wait until the request has finished, this is a basic form of a promise.

```
$scope.userLogin = function(form, name, password) {
    var username = parseInt(name, 10);
    var success;
    var login = 1;

    if (form.$valid) {
        User.auth(username, password, login).then(function() {
            if (User.session_id === 1) {
                $scope.showSuccess("Login"); // alert
                IServices.pushinit(); // push notify setup and
                ↪ save user
            } else {
                if (User.connectionError === true) {
                    console.log('User.connectionError === true:
                    ↪ failed');
                } else {
                    $scope.showError("Username or Password
                    ↪ Incorrect");
                }
            }
        }, function() {
            console.log('userLogin: failed');
        });
    }
};
```

Below is the regular expression used for the name fields.

Only alpha characters are allowed, this is used in conjunction with ng-pattern in the html form.

```
$scope.regex_name = '^[a-zA-Z]+$';
```

5.3.4 Diagram

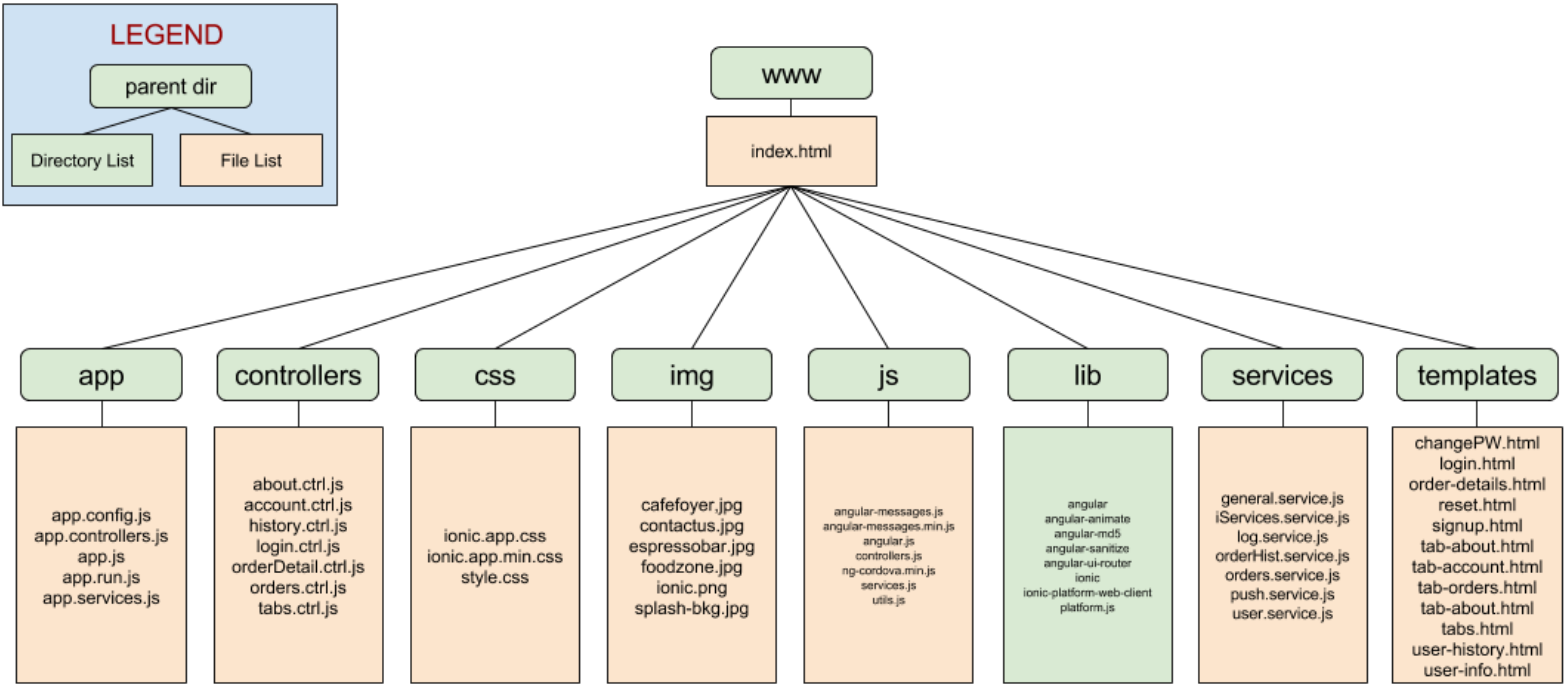
The Ionic mobile application contains many folders and files, some for styling, structure and others for running tasks, downloading templates and building binaries.

Here is a list of the directories within the home directory:

- hooks
- test
- node_modules
- platforms
- plugins
- privacy-policy
- resources
- scss
- typings
- www

There are also around 20 files for various functions as described above. The actual development happens within the **www** folder (www because we are working within a web view).

Here is an illustration of the contents within the **www** folder:



5.4 HTTP RESTful Route Requests

A listing of all the routes, including request and response data. The request data is any data that we give (either in the URL via ':' or in the body of the HTTP request) to the server. The response data is data received from the server in response to our HTTP request (in JSON format or else error 404).

First I will show you an example of how we do a HTTP request in Angular. Since the most common function we use is for authorising the user, we will show you that one:

```
o.auth = function(username, password, login) {
    var passwordMd5 = md5.createHash(password);

    if (login) {
        o.showLoading();
    }

    var postTo = SERVER.url + "verifyuser/" + username
    ↪ + "/" + passwordMd5;

    return $http({
        method: 'GET',
        url: postTo
    }).then(function successCallback(response) {
        if (response.data.hasOwnProperty('customer_id')) {
            o.setSession(username, passwordMd5, 1,
            ↪ response.data.customer_id,
            ↪ response.data.customer_name,
            ↪ response.data.customer_surname,
            ↪ response.data.customer_email,
            ↪ response.data.customer_mobile,
            ↪ response.data.customer_address,
            ↪ response.data.customer_cash);
        }

        if (login) {
            o.hideLoading();
        }
        o.connectionError = false;
    });
}
```

```

    }, function errorCallback(response) {
    if (login) {
        o.hideLoading();
    }
    o.connectionError = true;
    });
};

```

Here we can see we first hash the users password with MD5. Then we show the loading animation/spinner, next we create the URL and make the HTTP request. Once we have a response we save the data and hide the loading animation/spinner, or else show an error message.

The rest of the HTTP requests follow a similar format.

We start the url with a prefix then add on the url function that we wish to use.

Prefix:

- **Production:** `http://canteenapplication.vmarisevs.me/mobileapplication/`
- **Development:** `https://catering-test-server.herokuapp.com/`

Below I will show the URL structure, along with an example request and response.

Create User

URL:

```

createuser[/:customerLogin] [/:customerPW] [/:customerEmail] [/:customerMobile]
↪  [/:customerName] [/:customerSurname] [/:customerAddress]

```

Request:

```

createuser/123123/pw/email@email.com/0861231234/First/Sur/address

```

Response:

```

{"customer_identifier":"11E578C39D1DAE2881B43FAA99013818"}

```

Login/Verify User

URL:

```

verifyuser[/:customerLg] [/:customerPW]

```

Request:

```
verifyuser/123123/pw
```

Response:

```
{"customer_id":"11E4E6370B663F0D81B9EC9A743CC2AE",  
"customer_name":"Regina"  
,"customer_surname":"Walsh",  
"customer_cash":"10.20",  
"customer_mobile":"0853243435"}
```

Get Food**URL:**

```
getfood
```

Request:

```
getfood
```

Response:

```
[{"food_id":"23961698511618135",  
"food_name":"BLT", "food_description":"Grilled Bacon,\nSliced  
→ Tomato,\nCrisp  
→ Lettuce,\nMayonnaise\n", "food_price":"3.95",  
"food_type":"ROLL", "food_picture":"BLT.jpg"},  
→ {"food_id":"23961698511618136",  
"food_name":"Baked Ham & Cheese", "food_description":"Baked  
→ Ham,\nMozzarella Cheese, \nCeddar Cheese,\nCountry  
→ Relish\n", "food_price":"3.95",  
→ "food_type":"ROLL", "food_picture":"BakedHamnCheese.jpg"}]
```

User History**URL:**

```
userhistory[/:customerID] [/:customerPW]
```

Request:

```
userhistory/11E4E6370B663F0D81B9EC9A743CC2AE/pw
```

Response:

```

{"orders":{"11E55B538748A90AA04434E6D77FF0A4":{"order_collected":"0",
"order_comments":"\"some comment\"",
  ↳ "order_collection_time":"12:00:00",
"order_posted":"2015-09-15
  ↳ 02:42:50", "food":[{"food_id":"23961698511618135",
"food_name":"BLT", "food_description":"Grilled Bacon,\nSliced
  ↳ Tomato,\nCrisp Lettuce,\nMayonnaise\n",
"food_price":"3.95", "food_count":"1",
  ↳ "food_type":"ROLL", "food_pic":"BLT.jpg"}, {"food_id":"23961698511618141",
"food_name":"500ml still
  ↳ water", "food_description":"","food_price":"1.00",
  ↳ "food_count":"1", "food_type":"DRINK", "food_pic":"default.jpg"]]]}}

```

Change User Password**URL:**

```
changeuserpw[/:customerID] [/:customerPW] [/:customerNewPW]
```

Request:

```
changeuserpw/11E55B29BCA900A6A04434E6D77FF0A4/pw/pww
```

Responses:

```
{"status":"OK"}
```

```
404
```

Working Time**URL:**

```
workingtime
```

Request:

```
workingtime
```

Response:

```

{"OrdersCollectionTime":["12:00:00","14:00:00"],
  ↳ "TodaysWorkingTime":{"open_time":null,"close_time":null},
  ↳ "CurrentTime":"18:26:50"}

```

Buy Food

URL:

buyfood[/:customerID] [/:customerPW] [/:collectionTime]

Request:

http://canteenapplication.vmarisevs.me/mobileapplication/buyfood/

↪ 11E576B38F799E9281B43FAA99013818/pw/12:00?order_list=

↪ {%221%22:%2224151838022434887%22}&customer_comments=%22Hello%20Vlad%22

Responses:

```
{"status": "OK"}
```

404

Recover User Password

URL:

recoveruserpw[/:customerLg] [/:customerEmail]

Request:

recoveruserpw/305490/vmarisevs@gmail.com

Responses:

```
{"status": "OK"}
```

404

Chapter 6

System Evaluation

6.1 Web App

6.1.1 PDF make vs DOM PDF

why???

6.2 Mobile App

6.2.1 Chosen tech, why and how?

6.2.2 Alternatives tested

6.2.3 Robustness

6.2.4 Performance

6.2.5 Limitations

Native SDK Support

Too new?

Chapter 7

Conclusion

Bibliography

- [1] (2016 (accessed on 2016-04-22)) Gmit catering github org. <https://github.com/GMIT-Catering>.
- [2] (2016 (accessed on 2016-04-22)) Gist readme. <http://gmit-catering.github.io/final-year-project-template/>.
- [3] (2016 (accessed on 2016-04-22)) Web app. <https://github.com/VMarisevs/CanteenOrderSystem>.
- [4] (2016 (accessed on 2016-04-22)) Mobile app. <https://github.com/RonanC/gmit-catering>.
- [5] (2016 (accessed on 2016-04-22)) Mock/test server for mobile app. <https://github.com/RonanC/gmit-catering-test-server>.
- [6] (2016 (accessed on 2016-04-22)) Push notification web app. <https://github.com/RonanC/gmitcat-push>.
- [7] (2016 (accessed on 2016-04-22)) Final year project report. <https://github.com/GMIT-Catering/final-year-project-template>.
- [8] (2016 (accessed on 2016-04-17)) Cordova homepage. <https://cordova.apache.org/>.
- [9] (2016 (accessed on 2016-04-17)) JQuery mobile homepage. <https://jquerymobile.com/>.
- [10] (2016 (accessed on 2016-04-22)) Ionic homepage. <http://ionicframework.com/>.
- [11] (2016 (accessed on 2016-04-22)) In memoriam - edsger wybe dijkstra. <https://www.utexas.edu/faculty/council/2002-2003/memorials/Dijkstra/dijkstra.html>.

- [12] (2016 (accessed on 2016-04-22)) Heroku homepage. <https://www.heroku.com/>.
- [13] (2016 (accessed on 2016-04-22)) Module counts. <http://www.modulecounts.com/>.
- [14] (2016 (accessed on 2016-04-22)) Git homepage. <https://git-scm.com/>.
- [15] (2016 (accessed on 2016-04-22)) Manifesto for agile software development. <http://agilemanifesto.org/>.
- [16] (2016) Zend Framework 2 Overview. <http://framework.zend.com/about/>. [Online; accessed 13-April-2016].
- [17] Y. F. G. M. Elhakeem and B. I. A. Barry. (2013) Developing a security model to protect websites from cross-site scripting attacks using zend framework application. <http://0-ieeeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6634012&tag=1>. [Online; accessed 13-April-2016].
- [18] H. J. La and S. D. Kim. (2013) Balanced mvc architecture for developing service-based mobile applications. <http://0-ieeeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6634012&tag=1>. [Online; accessed 13-April-2016].
- [19] https://developers.google.com/chart/infographics/docs/qr_codes#overview. [Online; accessed 16-April-2016].
- [20] <https://packagist.org/packages/adminweb/qrcode-zf2-module>. [Online; accessed 16-April-2016].
- [21] <https://github.com/raykolbe/DOMPDFModule>. [Online; accessed 16-April-2016].
- [22] O. Corporation. (2016) Top reasons to use mysql. <https://www.mysql.com/why-mysql/topreasons.html>. [Online; accessed 13-April-2016].
- [23] N. Adermann and J. Boggiano. (2016) Composer documentation. <https://getcomposer.org/doc/>. [Online; accessed 13-April-2016].
- [24] (2016 (accessed on 2016-04-17)) Html mdn homepage. <https://developer.mozilla.org/en-US/docs/Web/HTML>.
- [25] (2016 (accessed on 2016-04-17)) Css mdn homepage. <https://developer.mozilla.org/en-US/docs/Web/CSS>.

- [26] (2016 (accessed on 2016-04-21)) Html/css codeschool path. <https://www.codeschool.com/learn/html-css>.
- [27] (2016 (accessed on 2016-04-21)) Html/css codeschool path. <https://www.codeschool.com/learn/javascript>.
- [28] (2016 (accessed on 2016-04-17)) Javascript mdn homepage. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [29] R. Connolly. (2015) The advantages of using javascript for full stack development with an emphasis on node.js. <http://www.ronanconnolly.ie/js-advantages-in-fullstack-dev>.
- [30] (2016 (accessed on 2016-04-21)) React homepage. <https://facebook.github.io/react/>.
- [31] (2016 (accessed on 2016-04-21)) Ember homepage. <http://emberjs.com/>.
- [32] (2016 (accessed on 2016-04-21)) The ionic show // episode 1. <https://www.youtube.com/watch?v=uJAWaE11Jf4e>.
- [33] (2016 (accessed on 2016-04-21)) John papa's angular style guide. <https://github.com/johnpapa/angular-styleguide>.
- [34] (2016 (accessed on 2016-04-22)) ios store. <https://itunes.apple.com/us/genre/music/id34>.
- [35] (2016 (accessed on 2016-04-22)) Android (google play))store. <https://play.google.com/store?hl=en>.
- [36] (2016 (accessed on 2016-04-22)) Windows phone store. <https://www.microsoft.com/en-us/store/apps/windows-phone>.
- [37] (2016 (accessed on 2016-04-17)) Yeoman homepage. <http://yeoman.io/>.
- [38] (2016 (accessed on 2016-04-23)) Angular generator. <https://github.com/yeoman/generator-angular>.
- [39] (2016 (accessed on 2016-04-17)) Grunt homepage. <http://gruntjs.com/>.
- [40] (2016 (accessed on 2016-04-23)) Gulp homepage. <http://gulpjs.com/>.
- [41] (2016 (accessed on 2016-04-23)) Sass homepage. <http://sass-lang.com/>.
- [42] (2016 (accessed on 2016-04-17)) Jasmine homepage. <http://jasmine.github.io/>.

- [43] (2016 (accessed on 2016-04-17)) Karma homepage. <https://karma-runner.github.io/0.13/index.html>.
- [44] (2016 (accessed on 2016-04-17)) Bower homepage. <http://bower.io/>.
- [45] (2016 (accessed on 2016-04-17)) Npm homepage. <https://www.npmjs.com/>.
- [46] (2016 (accessed on 2016-04-17)) Bash homepage. <https://www.gnu.org/software/bash/>.
- [47] (2016 (accessed on 2016-04-17)) Phonegap homepage. <http://phonegap.com/>.
- [48] (2016 (accessed on 2016-04-17)) Xamarin homepage. <https://www.xamarin.com/>.
- [49] (2016 (accessed on 2016-04-17)) Cloudant homepage. <https://cloudant.com/>.
- [50] (2016 (accessed on 2016-04-17)) Couchdb homepage. <http://couchdb.apache.org/>.
- [51] (2016 (accessed on 2016-04-17)) Nodejs homepage. <https://nodejs.org/en/>.
- [52] (2016 (accessed on 2016-04-17)) Expressjs homepage. <http://expressjs.com/>.
- [53] (2016 (accessed on 2016-04-17)) Github homepage. <https://github.com/>.
- [54] R. T. Fielding. (2000) Representational state transfer (rest). http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Online; accessed 13-April-2016].
- [55] (2016 (accessed on 2016-04-23)) Json homepage. <http://www.json.org/>.
- [56] (2016 (accessed on 2016-04-17)) Http requests information. http://www.w3schools.com/tags/ref_httpmethods.asp.
- [57] <https://github.com/arvind2110/ZF2-Auth-ACL>. [Online; accessed 16-April-2016].
- [58] <http://pdfmake.org/index.html#/gettingstarted>. [Online; accessed 16-April-2016].