Food Ordering System for the GMIT Catering Company

Ronan Connolly

Vladislav Marisevs

B.Sc.(Hons) in Software Development

April 25, 2016

Final Year Project

Advised by: Dr John Healy

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

Contents

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 companies 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

The PHP is widely used as a popular server side language and great number of open source software and company's web sites use PHP since it can enable high software productivity[8].

IBM Research group from Tokyo Research Laboratory reports that PHP as a web service engine performs competitively with Axis2 Java for web services involving small payloads, and greatly outperforms it for larger payloads by 5-17 times. As the authors expected, Axis2 C performs best, but the experimental results demonstrate that PHP performance is closer to Axis2 C with larger payloads[9].

In order to develop a secure web application, Zend Framework 2 was chosen as base of this project. From the various types of PHP frameworks it was designed by Zend Technologies Ltd, who also developed PHP license. Appendix A describes cross-site scripting attacks and how to prevent them using Zend Framework.

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 fo 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 [10], 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 [11] which had better styling but again I came across many issues.

Finally I came across the Ionic Framework [12] 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 scafolded 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 dissadvantage on having the push server seperate
- 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 [13]*

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:

```
// login
app.get('/verifyuser/:customerLg/:customerPW', function(req,
 ↪  res){
    var result = {
        "customer_id":"11E4E6370B663F0D81B9EC9A743CC2AE",
        "customer_name":"Regina",
        "customer_surname":"Walsh",
        "customer_cash":"10.20",
        "customer_mobile":"0853243435"};

    res.contentType('application/json');
    res.send(JSON.stringify(result));
});
```

I hosted this test server on Heroku [14]. 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 [15]:

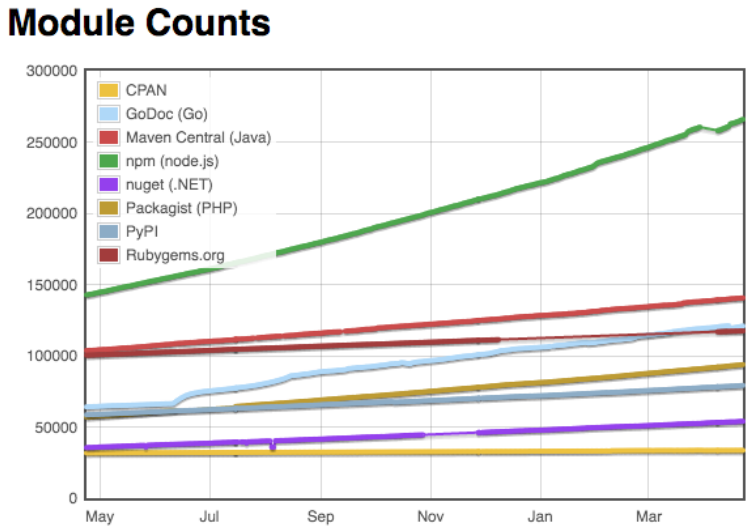


Figure 3.1: Module Count Growth in the last Year

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:

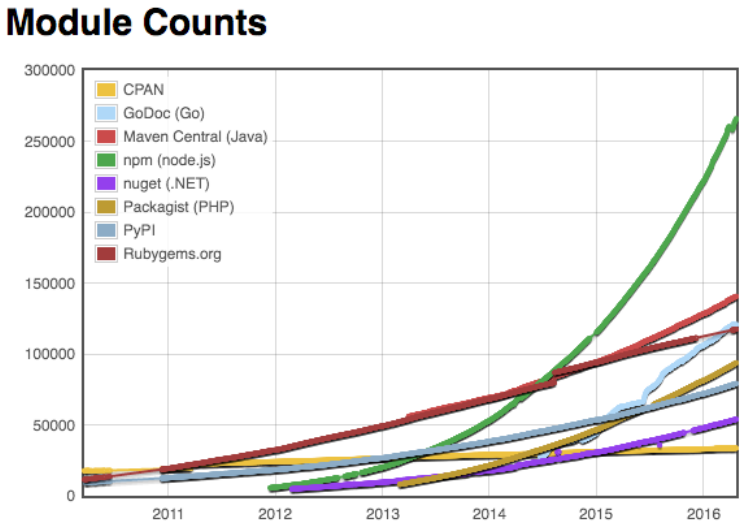


Figure 3.2: Module Count Growth All-time

Server Providers that use the git [16] 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"* [17]:

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 there 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
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 Expresso 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 my 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

```
.constant('SERVER', {
  // test server
  // url: 'https://catering-test-server.herokuapp.com/',

  // real server
  url:
   ↪  'http://canteenapplication.vmarisevs.me/mobileapplication/',
});
```

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

PHP is server side scripting language that is widely used and specially suited for web development it can be embedded into html page.

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Example</title>
   </head>
   <body>

      <?php
         echo "Hi, I'm a PHP script!";
      ?>

   </body>
</html>
```

It also has a wide range of protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM and others, that can be used to talk to other services. PHP also has support for instantiation of Java objects and using them as native PHP objects.

3.5.2 RDBMS (SQL)

In order to chose between noSQL and RDBMS we thought about is there a reason why we should use SQL? Yes there is. In current project we are not just populating database with information, we also have to query the storage on the monotonic types, such as order date or all valid users. In this case we have large set of structured queries and Relational Database Management

System can solve this problem. because one of their advantages is atomic transactions. This mean that if we have to do set of transactions and one of them fail then, all others will be canceled. NoSQL type databases are using eventual consistency that cannot at any time guarantee that at flow of transactions that comes together are all done at current point in time. If you will try to access data while transaction is in process you might receive corrupted data.

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 it's data in JSON format by default.

PHP has a native function that parses object into JSON format string. In the example below we can see code snippet from Controller that prepares to return response back to user and in the content we are parsing array into JSON using *json_encode* function.

```php
<?php
public function indexAction{
  ...
  $response = $this->getResponse();
  $response->setStatusCode(200);
  $response->setContent(json_encode($result_array));
  return $response;
}
```

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. [18]

Barry and Elhakeem [19] 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 [20]. 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 [21]. 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. [22]

DOM PDF module

In order to generate **.pdf* file we use 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. [23]

PDF make

In order to generate a large pdf file and move the heavy bit from hosting server, we are using JavaScript module that structures the information and generates the file. [24]

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. [25]

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. [26]

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

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. Alongside, PhpMyAdmin allows you to manage easily your databases[27].

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 porfolio 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 [28] and CSS [29] 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 [30] 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:

```
$light:                                #fff !default;
$stable:                               #f8f8f8 !default;
$positive:                             #5475aa !default;
$calm:                                 #11c1f3 !default;
$balanced:                             #7baa1e !default;
$energized:                            #ffc900 !default;
$assertive:                            #840018 !default;
$royal:                                #886aea !default;
$dark:                                 #444 !default;

// The path for our ionicons font files
$ionicons-font-path: "../lib/ionic/fonts" !default;

// Include all of Ionic
@import "www/lib/ionic/scss/ionic";

// custom styles
@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 [31] on Code School. This gave me a great understanding of JavaScript [32].
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 [33], 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.

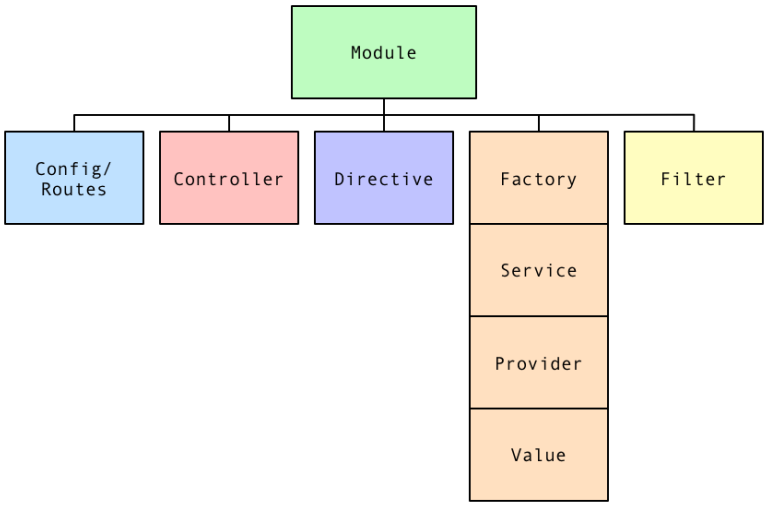


Figure 4.2: Angular Object Diagram

A few times a year there a 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 [34] and EmberJS [35].

I completed two Angular courses on Code School. Then for each course I completed I done 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 Modal 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 is can be seen straight away.

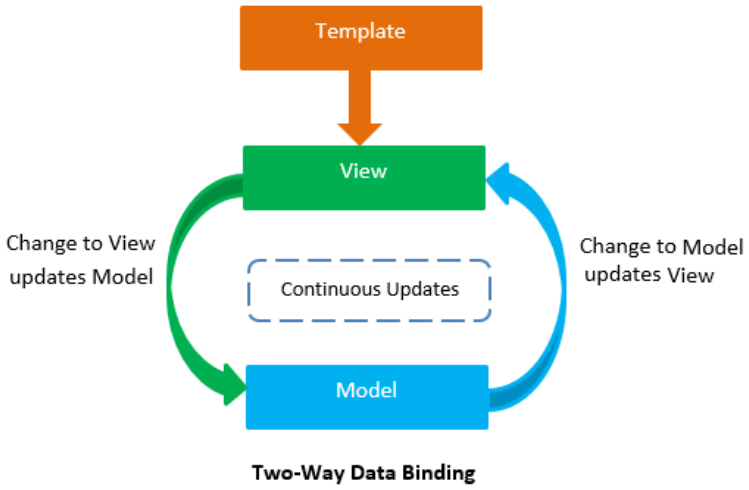


Figure 4.3: Two Way Data Binding

4.2.4 Ionic Framework

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

The Ionic [12] 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* [36].

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

```
$ ionic start myApp tabs
$ cd myApp
```

```
$ ionic platform add ios
$ ionic build ios
$ ionic emulate ios
```

Everything is done through command line.

Here is a simple controller:

```
(function() {
    'use strict';

    angular
        .module('canteen.controllers')
        .controller('AboutCtrl', about);

    about
        .$inject = ['$scope', 'Orders'];

    function about($scope, Orders) {
        $scope.opening = Orders.opening;
        $scope.closing = Orders.closing;
        $scope.times = Orders.collTimes;
    }
})();
```

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 [37]. 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 then the usual nested JavaScript hierarchy. I think it is pleasing to the eye and easy to understand.

4.2.5 Heroku

We used Heroku [14] 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):

```
> heroku login
> git init
> heroku create
> git add .
> git commit -a
> git push heroku master
> 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):

```
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 [38] 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 [39] (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
```

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
 ↪  -keystore ./ronanc.keystore android-release-unsigned.apk
 ↪  ronanc

zipalign -v 4 android-release-unsigned.apk gmitcat.apk

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 [40] 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

[41] 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 [42] 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:

```
Ronan:~/development/misc$ yo angular

 _-----_
|       |    .-------------------------------------------.
|--(o)--|    | Update available: 1.7.0 (current: 1.6.0) |
`---------´   |     Run npm install -g yo to update.      |
( _´U`_ )     '-------------------------------------------'
/___A___\
|  ~  |
__'.___.'__
´   `  |° ´ Y `


 _-----_
|       |    .--------------------------.
|--(o)--|    |    Welcome to Yeoman,    |
`---------´   |   ladies and gentlemen!  |
( _´U`_ )     '--------------------------'
/___A___\
|  ~  |
__'.___.'__
´   `  |° ´ Y `

Out of the box I include Bootstrap and some AngularJS
 ↪  recommended modules.

? Would you like to use Gulp (experimental) instead of Grunt?
 ↪  No
? Would you like to use Sass (with Compass)? No
? Would you like to include Bootstrap? Yes
? Which modules would you like to include? angular-animate.js,
 ↪  angular-cookies.js, angular-resource.js, angular-route.js,
 ↪  angular-sanitize.js, angular-touch.js
create app/styles/main.css
create app/index.html
create bower.json
create .bowerrc
create package.json
create Gruntfile.js
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 [43] 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 tunning *grunt default*.

4.4.2 Gulp

As said before, Gulp [44] is like Grunt. The main difference is that Gulp is programmatic rather then defined.
We are using it for SASS [45] 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

[46] 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 is 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 rely 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

[47] Karma is test-framework agnostic, what is 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 [48] 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 [49] 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 [50] 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 [10] 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 [51], 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 [11] 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


[52] 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 [53] NoSQL database solution usese CouchDB [54] 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 NoSLQL DB as a service solutions. Another one that has been around for awhile is Mlab (MongoDB as a service).

4.6.2 NodeJS

NodeJS [55] which I talk about in detail in my literature review [33] 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 [56] 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.

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 [57] 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.

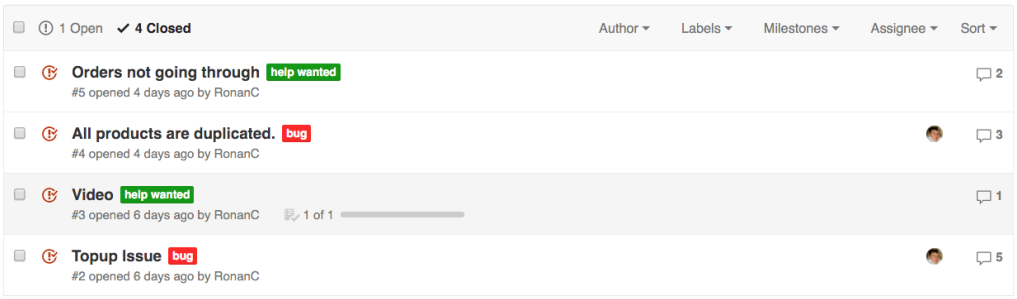


Figure 4.4: Issue Tracker

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].

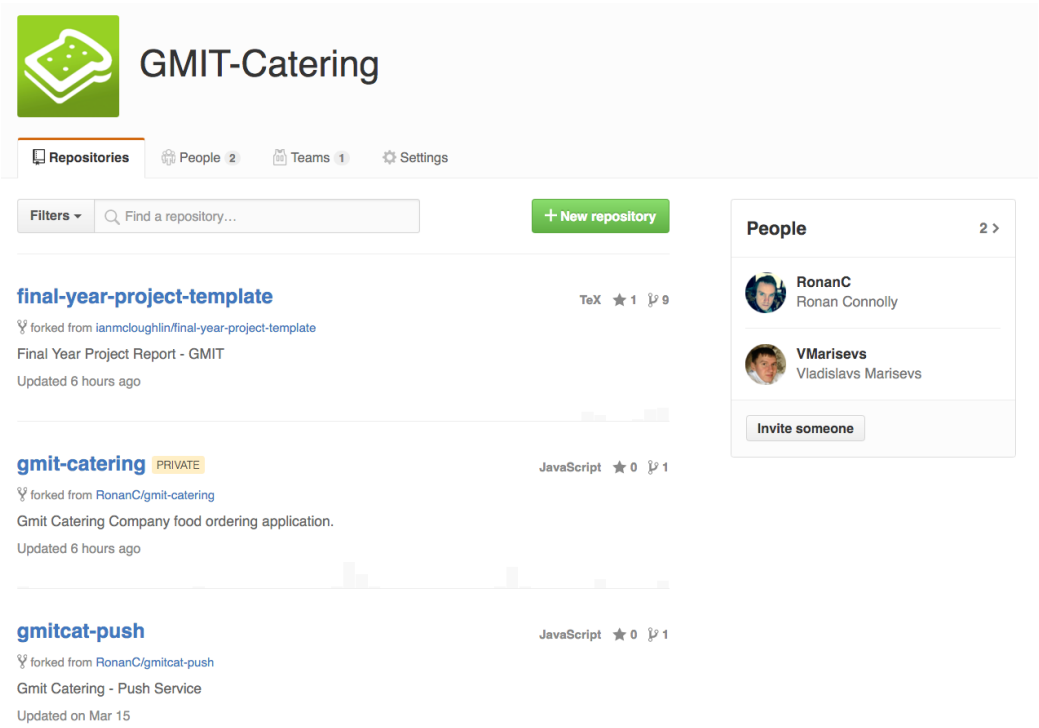


Figure 4.5: GMIT Catering GitHub Org

4.8 JSON REST interface architecture

We chose to use a RESTful architecture [58] with JSON [59] 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.

4.8.1 HTTP Requests

[60] 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:

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.

The web server has a custom API that the mobile app uses We will go into it some more below in the System Design section.

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. [61]

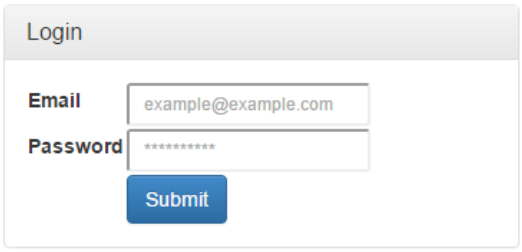


Figure 5.1: Login container

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 [24]. 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 paragr
    '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 t
        { 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 figuer 5.3.

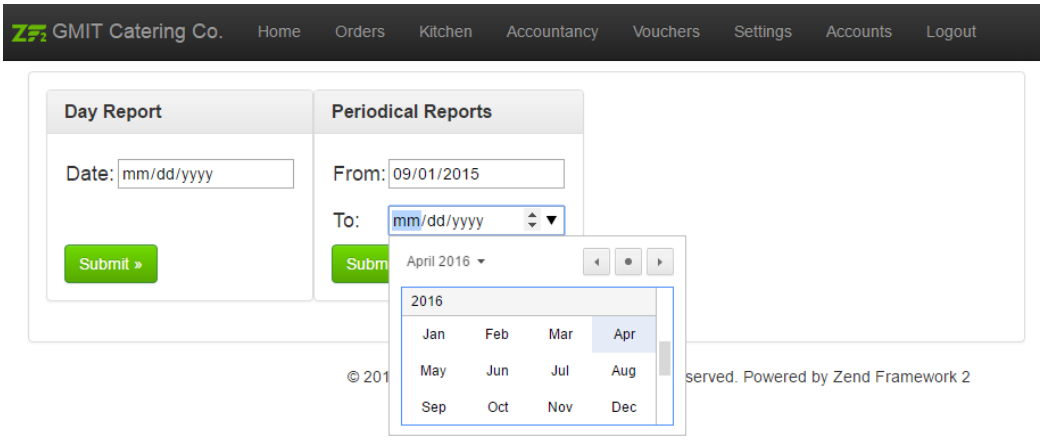


Figure 5.3: Daily/Periodic report page

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.

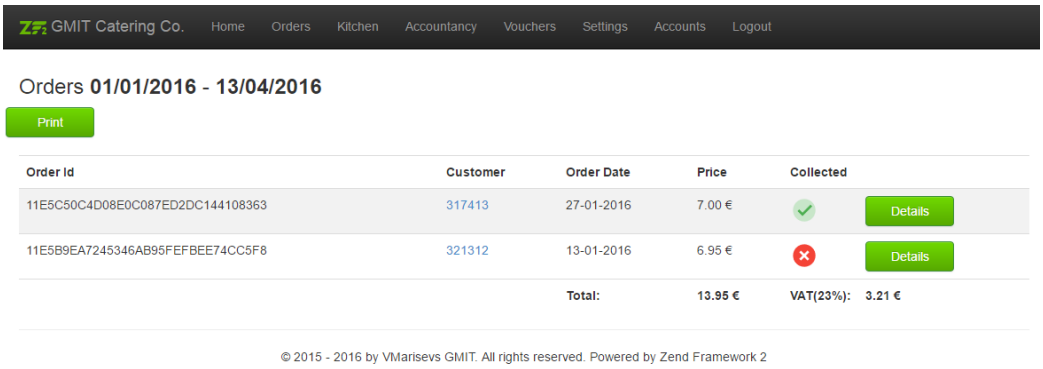


Figure 5.4: Daily/Periodic report page

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.

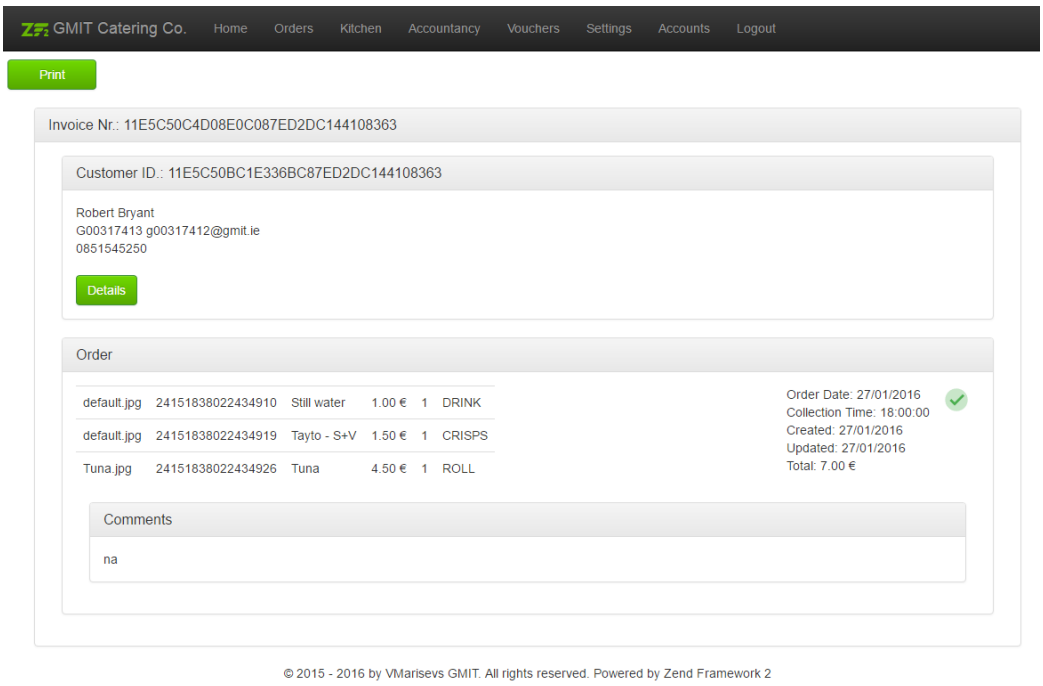


Figure 5.5: Daily/Periodic report page

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

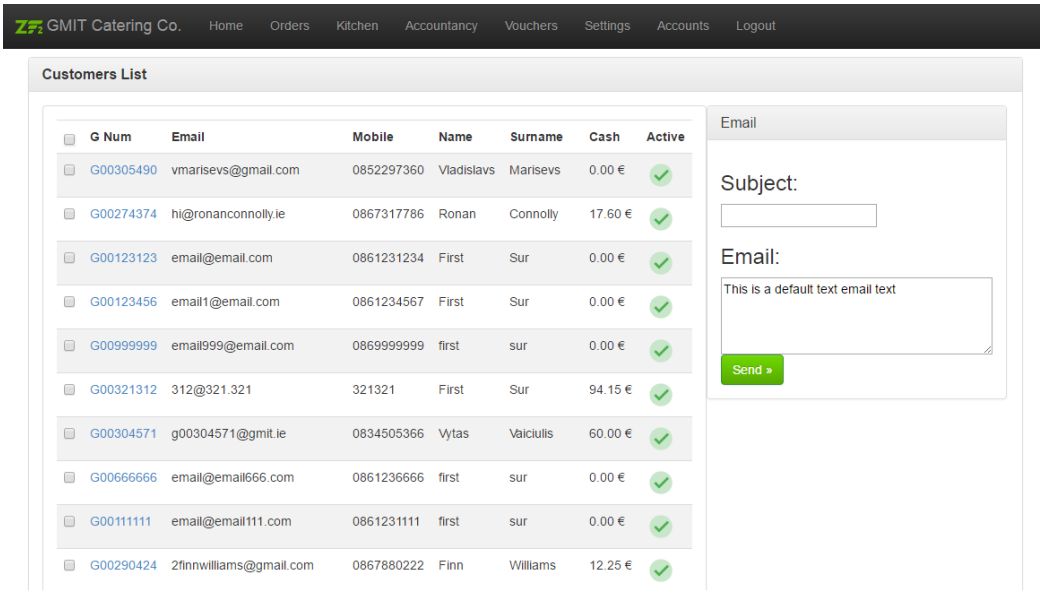


Figure 5.6: Customers list

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.

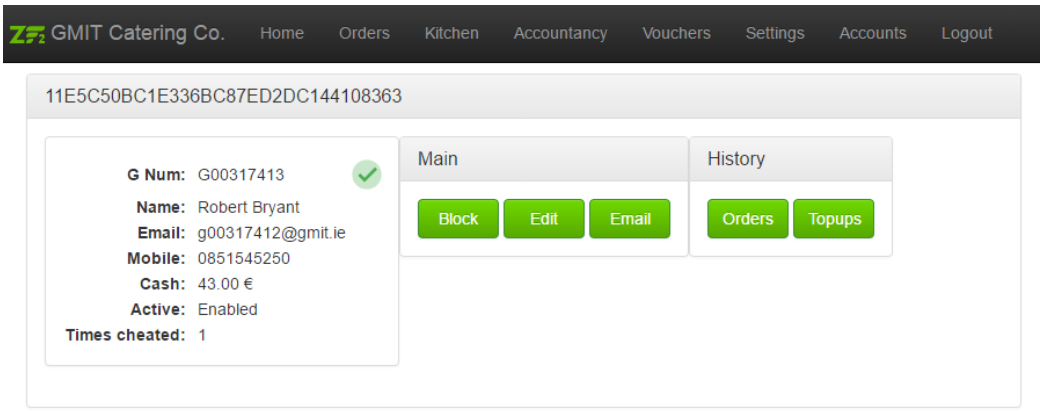


Figure 5.7: Customers details

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.

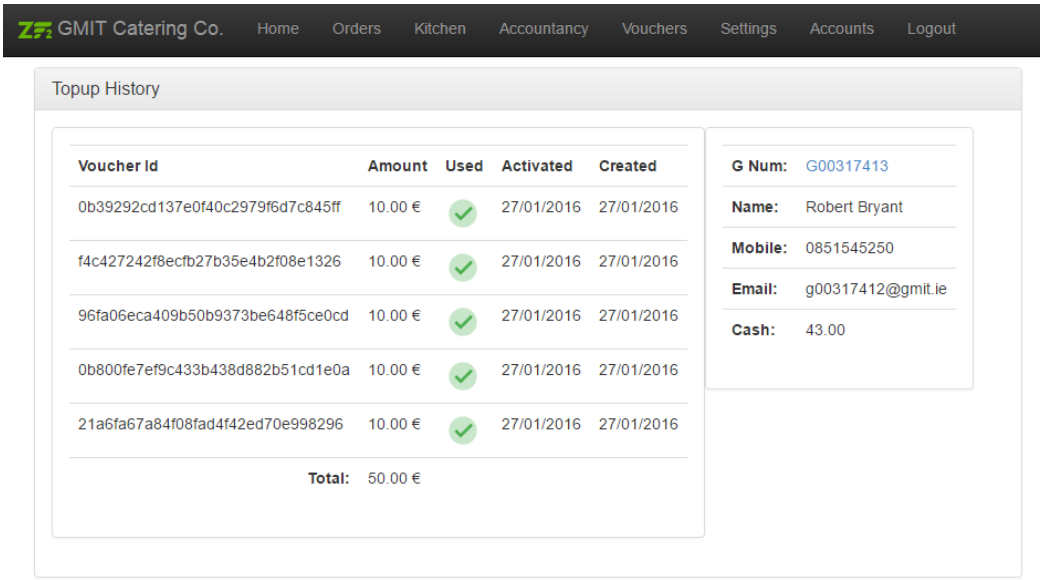


Figure 5.8: Customers top up history

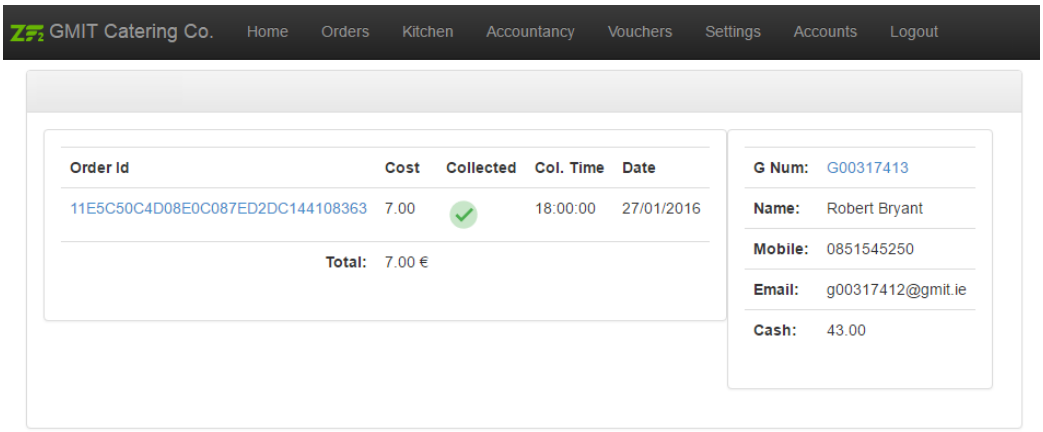


Figure 5.9: Customers order history

5.2.5 Ionic routes & functions

In order to establish interconnection between Ionic and Zend Framework we are using JSON interface. To secure the routes that available for mobile application we added special paths on which API would response. When application queries the *http://.../getfood/* it will return JSON type string with all food available. In order to deal with customers we have specialized roots for them, for example *http://.../userhistory[/:customerID][/:customerPW]*. Using this path RDBMS will return complete user order history in JSON format. Note that the password is hashed on the Ionic side and passed just a hash code and user id. This sort of combination would be quite difficult to guess. We also have more routes that are designed using similar principle:

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

5.3 Mobile App

Here we will go though 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.

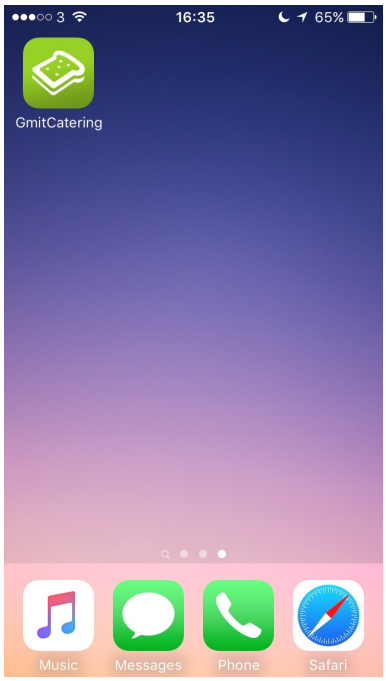


Figure 5.10: Home Screen

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.

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.

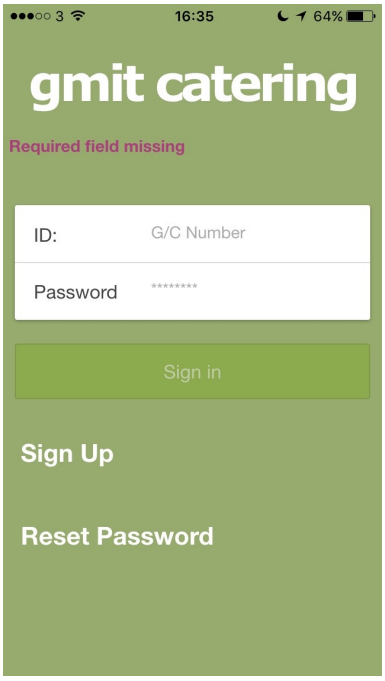


Figure 5.11: Login

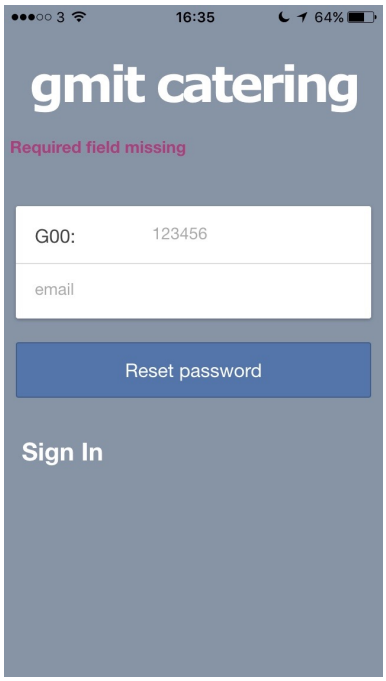


Figure 5.12: Password Reset

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.

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.

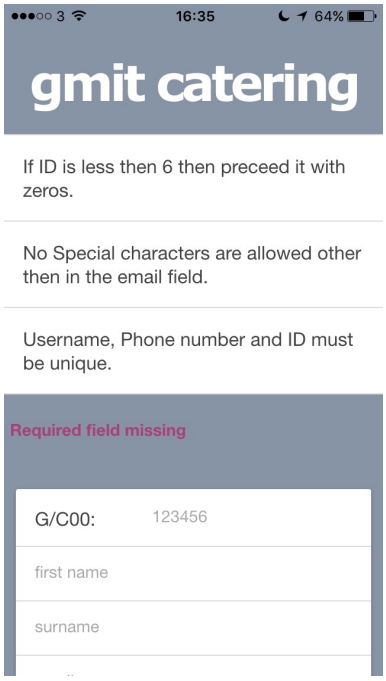


Figure 5.13: Sign Up

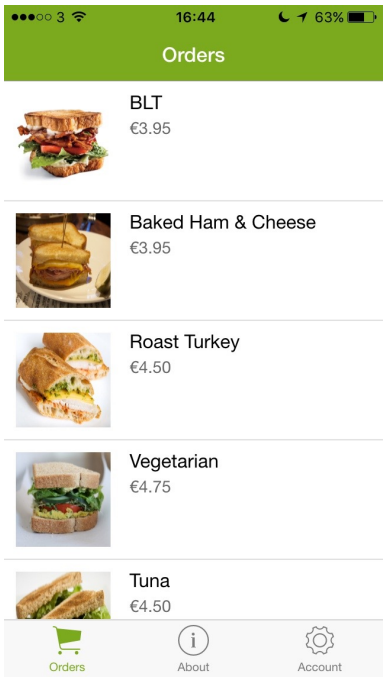


Figure 5.14: Orders

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 our 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.

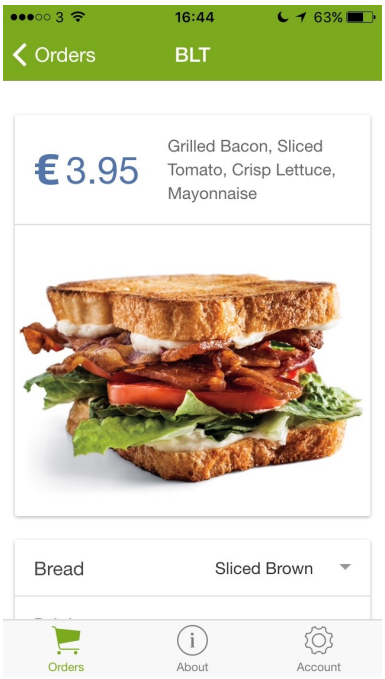


Figure 5.15: Order Details

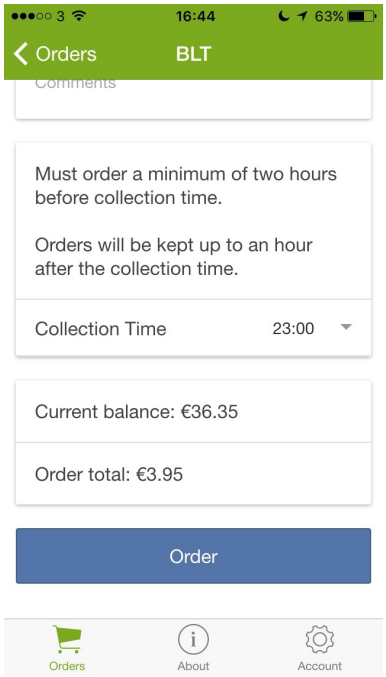


Figure 5.16: Order Details

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.

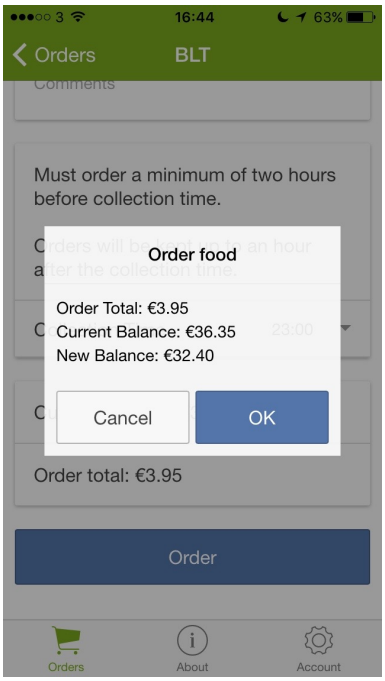


Figure 5.17: Order Confirm

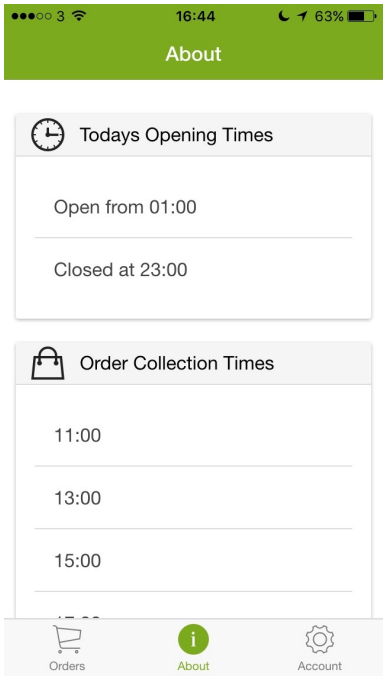


Figure 5.18: Times

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**, **Expresso Bar** and **Cafe Foyer**.

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

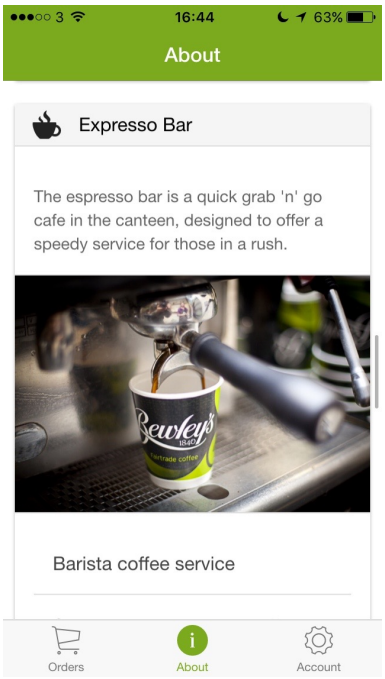


Figure 5.19: Espresso Bar

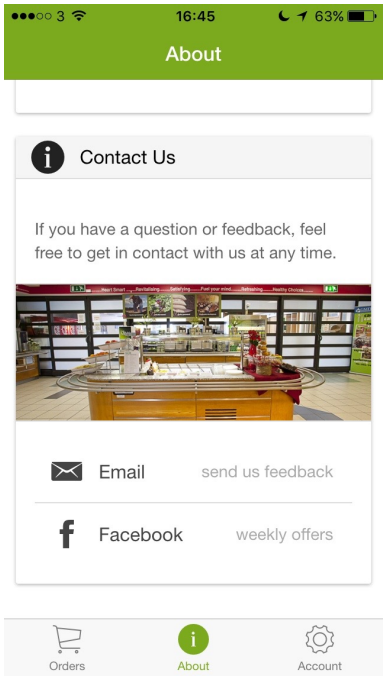


Figure 5.20: Contact Us

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.

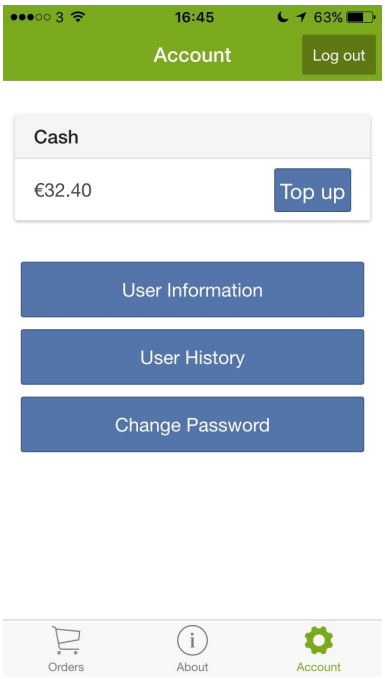


Figure 5.21: Account

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 is has been dealt with by the staff.

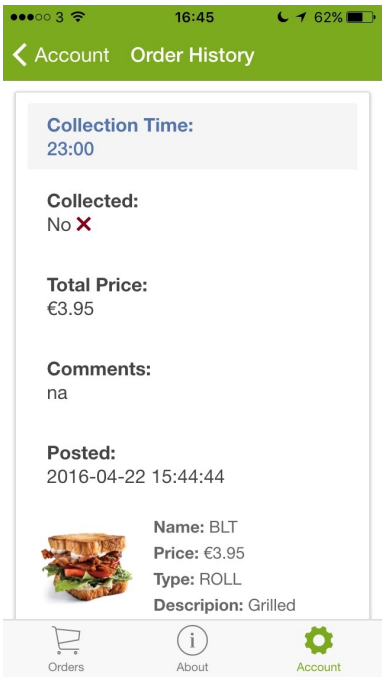


Figure 5.23: Order History

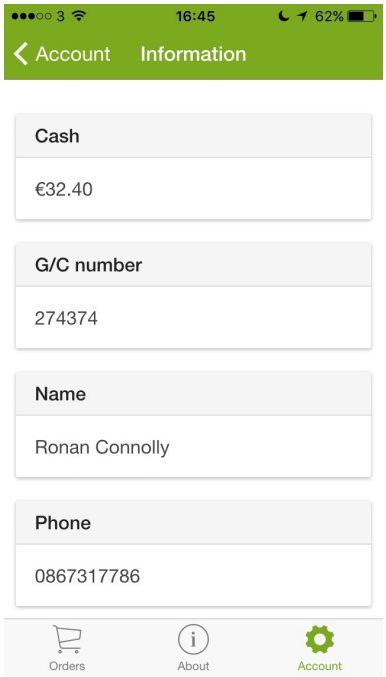


Figure 5.24: User Information

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

- Cash
- ID
- Name
- Phone

This can easily be extended to show other 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.

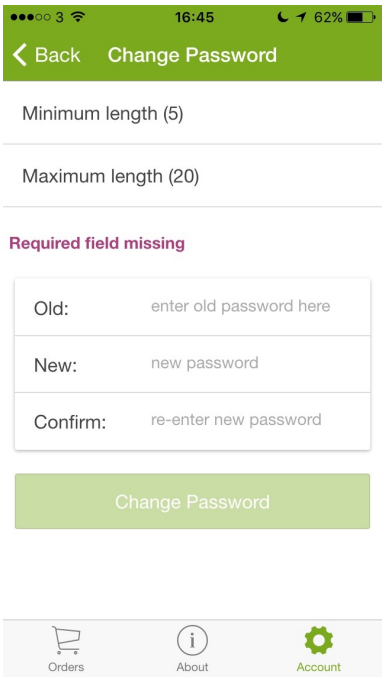


Figure 5.25: Change Password

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 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

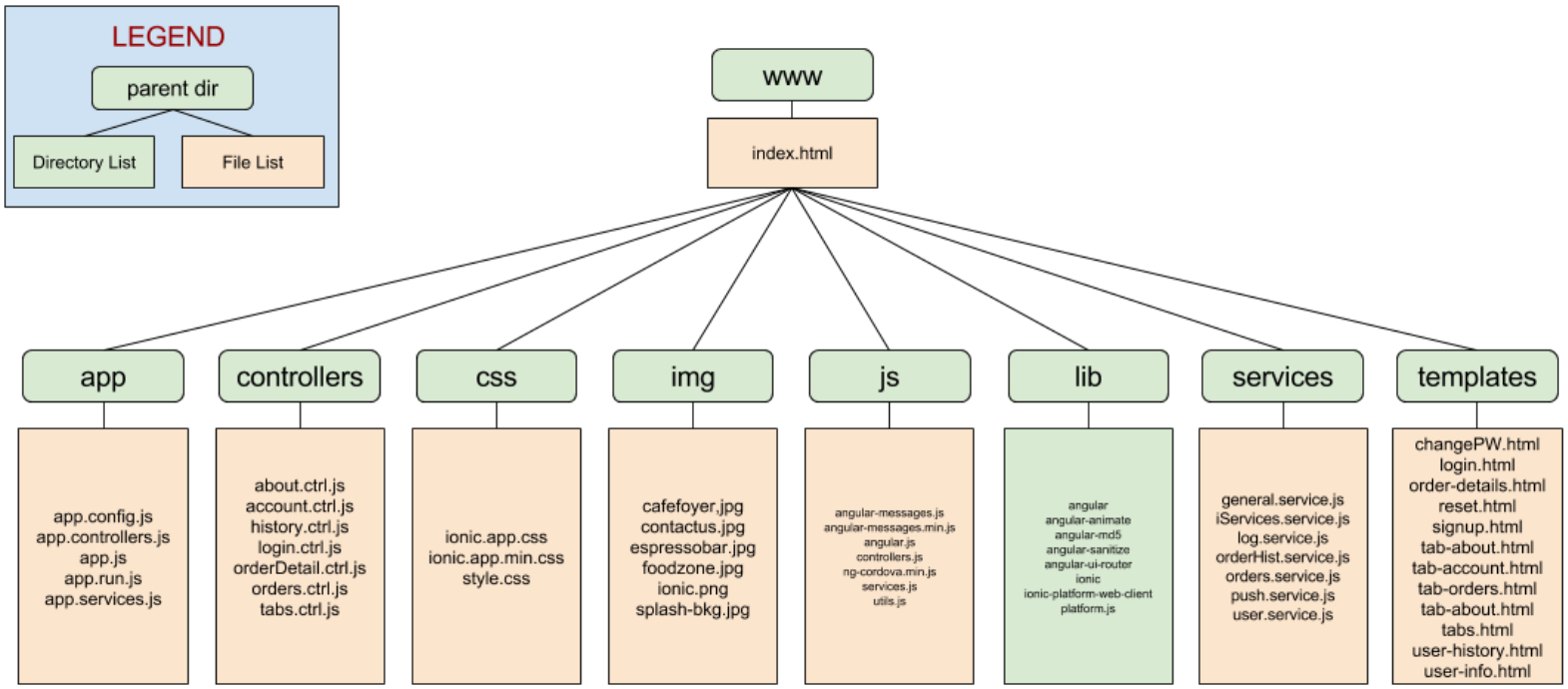
The Ionic mobile application contains many folders and files, some for styling, structure and others for running tasks, downloading templates and building binaries.

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, \nCheddar 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:%22241518380224348 87%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 Management Website

6.1.1 PDF make vs DOM PDF

One of most commonly used modules for generating *pdf* is DOM PDF for Zend Framework 2. This module let you generate file based on html type content. It has advantages such as easy to install using *composer*, interprets html page as structure of pdf file, loading pictures from remote sources. But it also has a big disadvantage to generate large file it will require more PHP hosting memory. At the start all reports were planned to be made using DOM PDF module, but after testing, it show that there is not enough memory available for PHP services.

After doing some research on this problem, best solution was to increase RAM memory available for PHP service. On the local machine in the WAMP server you have to just change the *memory_limit* in the ***php.ini*** file. That is a quick fix of this problem, but when it will come to upload this web application into real hosting, you will have same issue, because for increasing the performance of your service you have to pay more money.

To solve this problem and keep minimal required service performance, we decided to move pdf generation module from server side to client side. In this case website returns whole data that user required and it generates a report on the client computer. The idea was to use JavaScript that is interpreted by browser to create a pdf file [24].

Current web application uses both modules, domPDF is used for creating new vouchers. It generates maximum one page with 8 vouchers on it. They have specific margins for special paper and remotely created QR code to let users simply scan the code and top up their account balance. pdfMake can not be used for loading remote images, they have to be on the same origin.

But when it comes to generate large list of invoices we are using a pdfMAKE that will send out the data to users computer and using JavaScript generates a pdf report that can be printed or saved.

6.2 Mobile App

6.2.1 Chosen tech, why and how?

We chose to use Ionic because it is the most cutting edge cross platform technology out there at the moment. It has the largest and most interactive developer community. It seems like the obvious choice, even if JavaScript can be a pain in the neck.

Angular and the web development tool-set was essential as that's what Ionic requires you to use, no choice here I'm afraid. Steep learning curve but lots learned indeed.

Heroku as a server was a joy to use. iOS store was tricky but once learned not so bad. Android store was simple and fun to learn.

6.2.2 Alternatives tested

Tested out Cordova, PhoneGap, JQueryMobile and Xamarin. All were lacking in many areas.

Again, Ionic was the clear winner.

6.2.3 Robustness

JavaScript is lacking in robustness, this is inherent to the language itself. Loose typing, implicit variable definitions and no real official guidelines or structure at all. JavaScript doesn't even have a standard library.

While this causes issues (there is not much error checking or auto-completion), you can get around it with plenty of console logging and getting comfortable with the Google Chrome Developer Tools. Using transpiled languages like Microsoft's TypeScript gives more structure such as auto-completion and error checking.

Google have done a good job at bringing structure to the JavaScript (and web development) world with the advent of Angular.

We done much user testing in order to kink out any errors. Something that unit tests may not find.

6.2.4 Performance

Ionic is as performant as we needed. Our application contains various buttons, forms mixed with CSS and Animations. There is not too much heavy duty calculation going on.

Say if we were creating a game, or processing image data, then perhaps using Java(Android) or Swift would be something we'd have to look into.

In regards to the structure of this project compared to Android and iOS, I think Angular does a good job at bringing structure to JavaScript. The application is very modular and understandable (check out the diagram above to see how simple it is).

6.2.5 Limitations

Native SDK Support

With project Cordova we get access to most of the hardware available on the device, this still doesn't mean that we get access to Apples Kits (Health, Game, etc), but it does give us access to things like the TouchID sensor.

Some project may rely on the proprietary SDKs, but many project don't, and that's fine.

Too new?

We faced many issues while developing in the web development world, every day we had to update various dependencies. If you haven't updated in a week, well you will be waiting a while before you start working.

If a package decides to be removed from a repo, or a breaking change occurs on a package that one of your dependencies depends on. Well prepare to navigate the GitHub issue tracker looking for others who have the same issue.

If you have a large update, like for XCode, iOS or Ionic, then there may be a ripple effect.

Example: When I updated to iOS 8.2 on my iPhone, I tried to update my app through the Ionic command line tool, I get a strange error, so I run it with XCode, similar error, but I notice XCode needs to be updated as well. I go to update XCode, XCode states that there is an update for my Mac that needs to be installed.

The Mac OSX update takes an hour, XCode update takes an hour. Then once all finished I have to update Ionic. Sometimes when updating XCode it removes the configuration file that you have set up previously, so you have to edit that.

This is the kind of thing that happens with each major update and the first few times it is very confusing, but this is the world we live in, fast paced and never ending.

Chapter 7

Conclusion

We set out to create a food ordering system for the GMIT Catering Company. Goals were set, meetings scheduled and plans put in place.

Throughout the project we constantly hit goals while being pulled back by new objectives. With meetings came more items on our task list. We worked through all of this and created a well designed product that the client wants.

In the future we would:

- Use the GitHub issue tracker from the beginning.
- Try out various other GitHub features like Milestones.
- Perhaps look into using other web development frameworks like Jetty or MEAN stack (as opposed to a PHP based one).
- Try out a transpiler language like TypeScript, in order to code Ionic/Angular in a more structured, robust way.
- Write unit tests for every function in the mobile application in order to check for bugs with each commit.
- Use TravisCI (or another continuous integration testing framework) to automatically run unit tests with each commit to Github.
- Investigate NoSQL databases, as SQL is not very easy to change, which is an issue in an Agile project.
- Document the minutes of every single meeting (and email it to all parties involved).

We learned much in this project, more then in any other project before. Some of the major topics that resonate with us when thinking back over this project are:

- Teamwork

- Collaboration
- Rapid application development
- Prototyping
- Agile

All of these things are life skills that will help us throughout our working lives.

Being able to research, pick up and implement a new technology that we only just heard of is something that was done constantly throughout this project. Keeping up with trends, but choosing wisely with precise research is a huge skill we have developed. Being involved with different project communities on their Websites, GitHub repo, Slack, StackOverflow and other channels is second nature at this stage.

This was a huge project, a real one for a real company. We have extensively detailed all aspects of our journey in this project report.

We hit the goals, attended the meetings, stuck to our plans and adapted when needed.

We tried to keep our project as simple as possible, from the architecture, technologies and algorithms used, to the documentation, readme's and websites built.

> *Simplicity is prerequisite for reliability.*
> *- Edsger W. Dijkstra [62]*

Appendices

Appendix A

Literature Review about Zend Framework

MVC Architecture enables rapid and secure Zend Framework Web based Application Development

Vladislavs Marisevs

School of Science & Computing
Galway-Mayo Institute of Technology, Ireland.

Introduction

Modern software development world has several design patterns, one of them is *Model View Controller* it is also called as *Presentation Abstraction Control* (or *PAC*) (Coutaz 1987). The key idea of this pattern is to divide the systems architecture into three different layers. The following is an extract from Wang's publication:

> The pattern isolates "domain logic" (the application logic for the user) from the user interface (input and presentation), permitting independent development, testing and maintenance of each (separation of concerns).
>
> (Wang 2011)

According to Guanhua Wang (Wang 2011) *MVC* is not a new software architecture design concept; it was described in 1979, by Trygve Reenskaug, then working on Smalltalk at Xerox PARC.

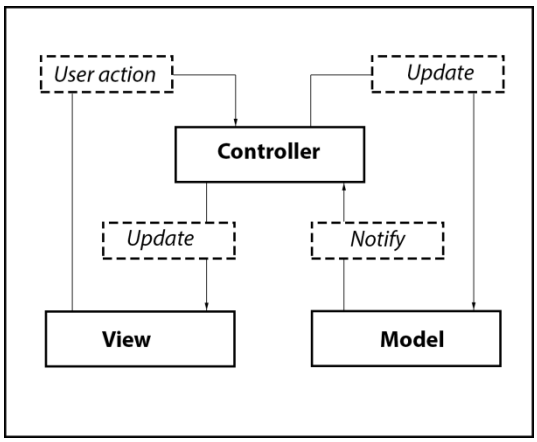


Figure 1. Mechanism of MVC

Model View Controller design divides a development process so that each part can be developed, maintained and tested independently without affecting other parts. The Model manages the application data and the core functionality. The Views displays the user interface and represents program's data. And Controller interprets the user inputs and handles events for views. This structure is shown in Figure 1. The application's front end can be modified without changing business logic. It also can handle different interfaces, like tabular information representation, charts or graphs. Also user's access permissions may vary, and each of access groups could have their own view.

According to Selfa's, Carrillo's and del Rocio Boone's publication (Selfa 2006) MVC's advantages are:

- Less coupling
- Higher cohesion
- Views provide high flexibility
- More design clarity
- Large scalability

According to example taken from Leff's and Rayfield's publication, Web applications also can be designed using Model View Controller architecture:

> "Java approach uses Entity Enterprise JavaBeans as the Model, constructs the View through HTML and JavaServer Pages, and implements the Controllers through Servlets and Session EJBs."
>
> (Leff, Rayfield 2001)

But this literature review will be concentrated on PHP programming format with the MVC architecture. By analysing the statistic report posted by Craig Buckler (Buckler 2015) it is easy to determine, that PHP is in the top 5 category. Moreover it has a good position as the web scripting language. This programming language went through the transformation since it was created in 1994. After 3.0 upgrade, PHP became an Object Oriented Programming language. This paradigm uses data structures and methods, it allows to use features like abstraction, encapsulation, polymorphism and inheritance. OOP side of PHP programming combined with Model View Controller structure improved code efficiency and reusability, but it also have a defect that Guanhua Wang is describing in his publication (Wang 2011) it is lack in operating system efficiency.

To prove this, Wang was using the programs Intel core2 Duo 2.00GHz, Apache 2.2.12, PHP 5.3.0, Chrome 7.0.517.44 Browser. Traditional PHP is a plain HTML page with combination of PHP scripts that are executed on the server side, so user would receive dynamically generated HTML formatted page. The MVC framework was Zend Framework 1.10 (ZF), it was preloading all default functions. And the last test was based on Lightweight MVC-like, this application doesn't have any of default validations or security against hacking attacks. The result of this page request test is shown in Table 1.

TABLE I. PAGE REQUEST TIME

Traditional	MVC	Lightweight MVC-like
24ms	174ms	46ms

According to the table above we can determine that ZF response time is slower than lightweight MVC-like and traditional PHP format. However, the provided functionality of ZF should be taken into account.

Body

The fast growth of the Internet also increases the amount of information available. Tim Berners-Lee invented the *World Wide Web* in 1989. Based on Vikas K. Malviya et al. (Malviya, Saurav, Gupta et al. 2013) publication, the motivation of *WWW* was to share information between scientists. With the growth and commercialization of *World Wide Web*, plain *HTML (Hypertext Markup Language)* was very limited. New scripting languages were invented to build dynamic websites. On the server-side languages like *PHP, ASP, JSP* and client side *JavaScript*.

Communication between web browser and server is implemented using HTTP (Hypertext Transfer Protocol). When user sends a request to a web server browser receives *HTML* page as response, it also can receive some scripts that will bring dynamic behaviour. Won Kim in the publication "The dark side of the Internet: Attacks, costs and responses" states that web technologies were invented to improve the world, but

with threats such as malware, hacking, denial of service attacks, invasion of privacy, etc started taking place. As from Mukesh Kumar Gupta publication:

> "In 2013, Open Web Application Security Project (OWASP) and Common Vulnerabilities and Exposures (CWE) reported Cross-Site Scripting (XSS) as one of the most serious vulnerability in web applications."
>
> (Gupta et al. 2015)

Cross-site Scripting attack Definition and Classification II

Web application's vulnerabilities are the main reason behind the Cross-site Scripting attacks. Weak or lack of validation of user input data provides the chances of XSS (Malviya, Saurav, Gupta et al. 2013). This application gap allows hackers to insert malicious scripts, that browser couldn't identify as a malware and will execute. In this case scenario attacker can hijack victim's session, cookie, deface websites, insert hostile content or conduct phishing attacks (Wang et al. 2011). Browser can't identify that this malware code is not a part of original page, because it is comes as HTTP response from origin server. There are 4 types of XSS attacks:

Stored or Persistent Cross-site Scripting Attacks

Bazara Barry (Elhakeem, Barry 2013) argues that Stored XSS is the most powerful kind of Cross-site scripting attack. This attack stores malicious script, which has been injected in the database as forum's message, visitor log, comment field or any other text type field. Victim receives the code when tries to access page with attacker's comment or message page.

Figure 2 illustrates the sequence how stored Cross-site Scripting attack is performed; this example is taken from Mukesh Kumar Gupta papers (Gupta et al. 2015).

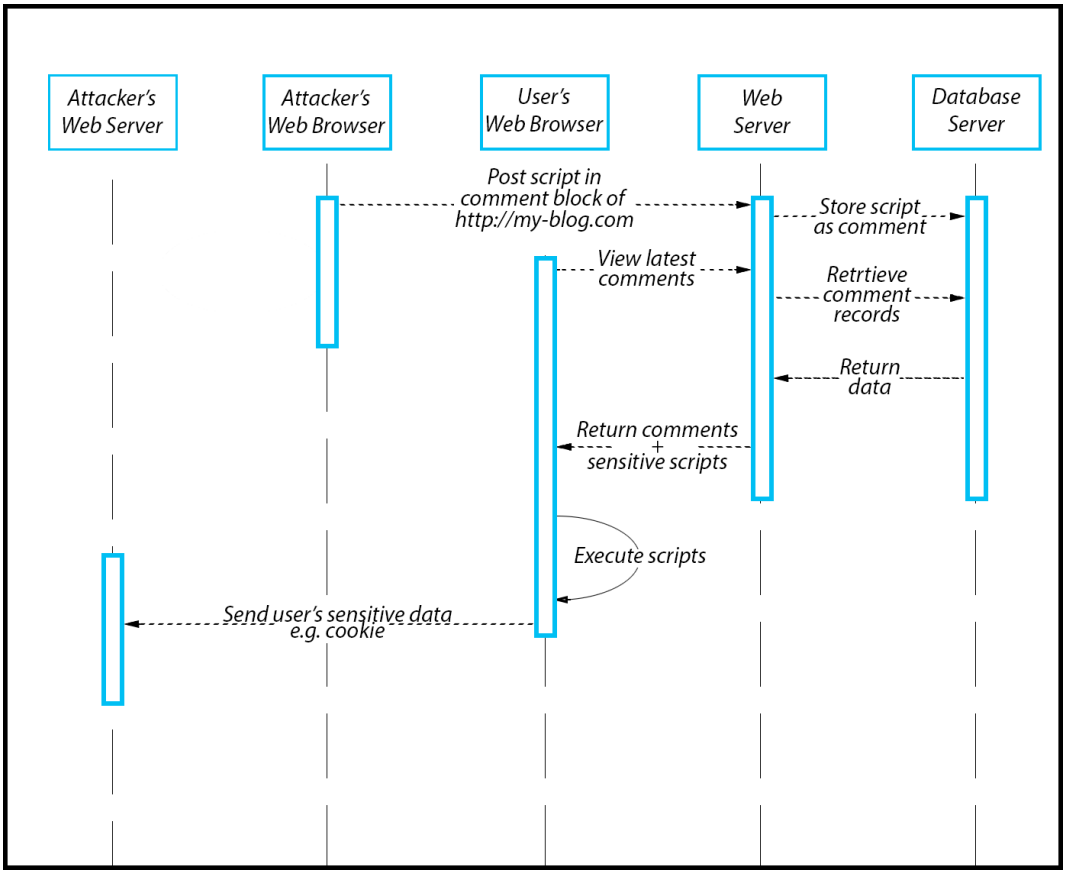


Figure 2. Sequence Diagram to Represent Stored XSS Attack Scenario

The attacker uses a blog website with vulnerability in the input type field, and he passes the malware scripts in the comment box. The comment box doesn't strip the tags and stores this script in the database. When legitimate-user tries to view comments and his browser requests the page. It receives malicious scripts as response and browser processes them like a part of the page. This piece of code can send HTTP request with sensitive user data (e.g. session id, cookie) to a malicious-user's server.

Reflected or Non-persistent Cross-site Scripting Attacks

Reflected XSS attacks code is not stored on the web server. In this type of XSS attacks malicious links are sent to victims using email or embedding the link in hacker's web server page (Malviya, Saurav, Gupta et al. 2013).

Figure 3 illustrates the sequence how malicious-user performs their attack on legitimate-user. Hacker is looking for a web application which returns input without proper validation as an error message, search result or any other response. Using this link, attacker crafts his own link by inserting some malware scripts that will post HTTP request into hacker's server. And this crafted link will be shared with victims via email, ad or etc.

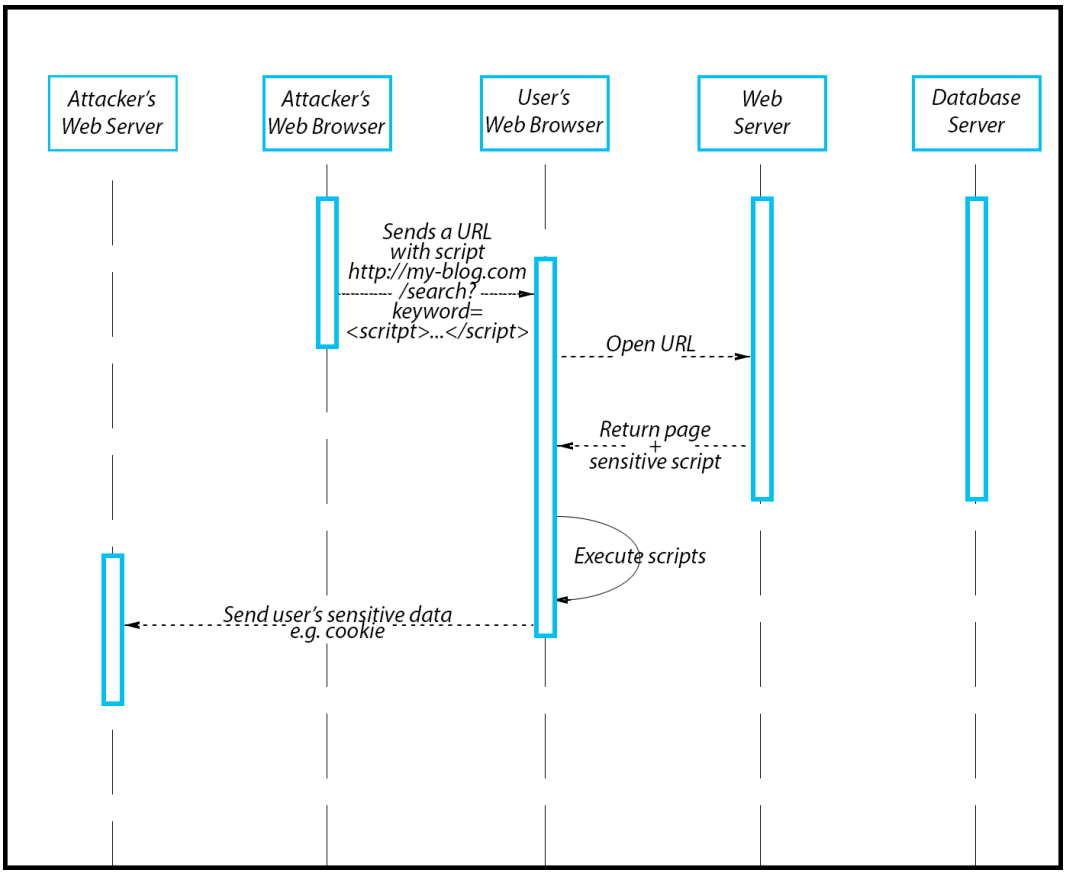


Figure 3. Sequence Diagram to Represent Reflected XSS Attack Scenario

DOM-based Cross-site Scripting Attack

XSS vulnerability occurs when client access malicious-user's crafted link, which has invalidated input that dynamically, obtained from *DOM* structure (Gupta et al. 2015). Di Paola and Fedon have found a vulnerability in the browsers that is described in their publication (Di Paola, Fedon 2006). Victim's browser allowed hijacking information, when they requested link with malicious script. For example *JavaScript* code that can be executed in the browser *"http://site.com/file.pdf#FDF=javascript:alert("Test Alert")"*, taken from Di Paola pages (Di Paola, Fedon 2006).

Induced Cross-site Scripting Attacks

According to Malviya papers this attack can take a place only when web application has *HTTP Response Splitting vulnerability* (Malviya, Saurav, Gupta et al. 2013). In this case attacker can manipulate the *HTTP header* of the server's response.

Let's say the request for *index* page returns in a *302 redirect* (*HTTP/1.1 302 Moved Temporarily*) to *http://www.abc.com/index.php?lang=en*. Another user wants to display the page in German and he changes the language, sends a request and receives *302 redirect http://www.abc.com/index.php?lang=german* . Comparing these two responses we can determine that only parameter *lang* has been changed, it is a target property for an attacker using *HTTP Response Splitting*. Malicious-user makes a link which contains 2 responses:

http://www.abc.com/index.php?lang=german%0d%0aContent-Length:%200%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2041%0d%0aHello, you have been phished

These responses separated by *%0d%0a* and the idea that he can infect the users behind his proxy server. The first response is mapped to first request, but second response hangs as there is no matching request. When user tries to access this page, he gets a response from proxy server, because proxy caches responses for requests often made. This example is taken from Arvind Doraiswamy publication (Doraiswamy 2011)

Traditional web proxy principles III

Ding Lan states that:

> "a traditional Web proxy provides the user an additional protection that unsuported by the existing personal firewalls, which allows users to control all the connections related with the browsers."
>
> (Lan et al. 2013)

After in-depth analysis by security experts, they found a solution, that the attacker couldn't use a static link for stealing the user's sensitive information, but only dynamically generated links could be used to submit data to third party domain (Lan et al. 2013). In this case Proxy will have a table with rules, where it temporary stores all static links that can be accessed by user without warning. Dynamically generated links should not be stored in the table, because they might allow transferring data to another party. Next section will discuss binary encoded Cross-site scripting attack, that has been taken from Ding Lan and Wu ShuTing publication (Lan et al. 2013).

Binary-encoded XSS attack

Embedded JavaScript in Web pages can be used also to edit *Document Object Model (DOM)* nodes in *HTML* dynamically (Nadji, Saxena, Song et al. 2009). Figure 4 presents the example of pseudo code based binary-encoded Cross-site scripting attack (Lan et al. 2013).

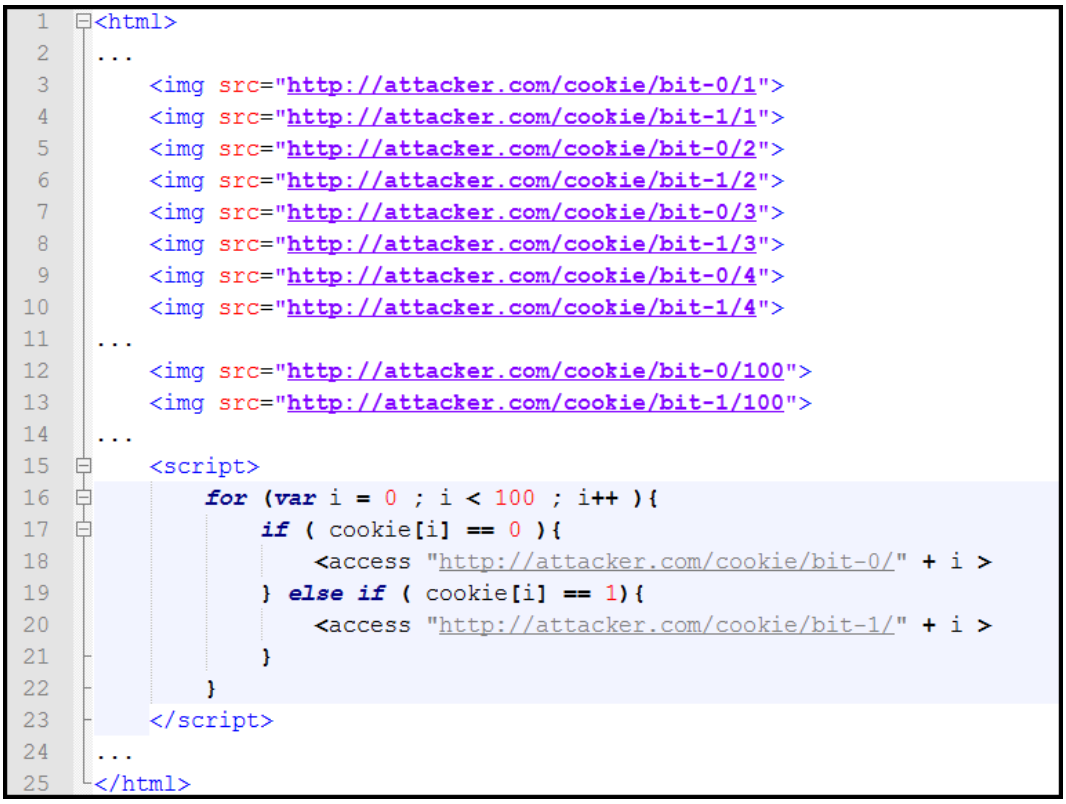


Figure 4. The pseudo code of binary-encoded XSS attack for stealing the user's cookie

Assume that legitimate-user have a 100 bit size cookie, and attacker aiming to steal that. In this case hacker uses *DOM* functionality to modify *HTML* Web page to add static links and for 100 bits = 200 links so each bit can be represented as 1 or 0 and each of them are unique link. And attacker also specifies the loop which will run through cookie and each bit will be submitted via dynamic link. As we discussed earlier proxy will pass through only static links, but in this case they are equals. Attacker can reconstruct the cookie based on accessed links which identifies bit position and value (Lan et al. 2013).

Analysis

Based on Mukesh Kumar Gupta (Gupta et al. 2015) publication, they build a public *GIT repository* (Stivalet, Delaitre 2014) that contains a synthetic test case generator. This application contains 9408 PHP samples. 5600 are safe and 3808 unsafe that are categorised. Gupta argues that evaluation of the proposed methods are performed on this dataset, as it provides mostly all the cases required for Cross-site scripting vulnerability prediction (Gupta et al. 2015). It is a free open source dataset with PHP source code with their vulnerability labels.

To avoid XSS script vulnerabilities developer should follow *The Open Web Application Security Project (OWASP)* guidelines (Keary at al. 2015). Malviya states that following these approaches XSS vulnerabilities can be prevented; the difficulty with these approaches is that they are totally dependent on developers (Malviya, Saurav, Gupta et al. 2013). The Cross-site scripting vulnerabilities are caused by improperly sanitized user input and this input finally reaches one of the sensitive sinks (i.e. saved in database, or 'echo').

There are several nodes in the proposed security model (Elhakeem, Barry 2013).

1. One of them is Security Awareness and it is based on developer expertise because they either do not know what cross-site scripting is, or they do not take security issues into consideration while developing Web applications and websites states Bazara Barry (Elhakeem, Barry 2013).
2. Another node is Server Security that is very important. Because this is a key point where application connects with the rest of the Internet.
    - Web application files should be always stored on a separate partition or drive other than system files. If attacker could get an access to servers root directory, he could also get access to other files on this partition.
    - To secure Server application they should use mechanisms and protocols that are specialized in particular area. For example Pretty Good Privacy (PGP) is specialized software that provides privacy. Commonly used for authentication and encryption/decryption of messages to increase the protection. Or Secure Electronic Transaction (SET) is a set of encryption used to protect data. It was supported initially by MasterCard, Visa, Microsoft, and others (Elhakeem, Barry 2013).
3. Client Security is also very important. Because they should think, before accessing any untrusted links. Browser must be up to date and it must support the greatest possible security and privacy.
4. Developers should follow Design Guidelines to make their Web applications safe. To reduce the websites vulnerabilities and present a new and efficient way of coding, it was referred to as Web Application Framework states Barry (Elhakeem, Barry 2013).

> "Web Application Frameworks are groups of program libraries, components and tools organized in an architecture system which helps the developers to build a complex Web application projects."
>
> (Elhakeem, Barry 2013)

Zend Framework is an open source framework for developing Web applications and services. It was written on PHP and it is loosely coupled architecture, which allows developer to code each component independently and designed with Model View Controller structure. This MVC paradigm we discussed earlier in this paper.

Barry and Elhakeem (Elhakeem, Barry 2013) 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 for validation and filtering of inputs.

This is quite flexible web application framework that brings open source modules into application. Nowadays web application URL doesn't specify the actual file on the server, and ZF is also designed like that. It has a runner file that is responsible for whole application and all business logic is loosely coupled to provide more efficient development.

Conclusion

Detailed definition of Model View Controller paradigm underlines why it is useful for agile development. This approach inherits more important advantages than lack in operating system efficiency.

This paper has well defined information about Cross-site scripting attacks including examples combined from different sources. The publication concentrates on XSS attacks and how they are dangerous for legitimate-user that their personal information can be stolen. And usually it is under developers' responsibility to create a safe web application or service. To do that they must follow proposed security model and *The Open Web Application Security Project* standards to make it secure.

The proposed model uses an open source web application framework that adopts *Model View Controller* design pattern and allows using default filtering techniques for user input and it is called Zend Framework. Based on the tests described in introduction we can determine that it has longer latency comparing to Traditional PHP and Lightweight MVC-like, but the secure functionality that it is doing can cover that disadvantage. Various tests using XSS attacks on this model can certify its design quality.

References:

Buckler, C. (2015). "What's the Best Programming Language to Learn in 2015?". Viewed 1th November 2015 at *http://www.sitepoint.com/whats-best-programming-language-learn-2015/*

Coutaz, J. (1987). "PAC, an Object-Oriented Model for Dialog Design". Elsevier Science Publishers, Proceedings of Human-Computer Interaction - INTERACT, pp. 431-436, viewed 1th November 2015 at http://mvc.givan.se/papers/PAC_an_Object_Oriented_Model_for_Dialog_Design.pdf

Di Paola, S., G. Fedon (2006). "Subverting Ajax". pp. 1-8, viewed 5th November 2015 at https://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf

Doraiswamy,A. (2011). "HTTP Response Splitting Attack". Viewed 5th November 2015 at http://resources.infosecinstitute.com/http-response-splitting-attack/

Elhakeem, Y. F. G. M., Bazara I. A. Barry (2013). "Developing a security model to protect websites from cross-site scripting attacks using ZEND framework application". pp. 624-629, viewed 5th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6634012

Gupta et al. (2015). "Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications". pp. 162-167 Viewed 4th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=7219789

Keary at al. (2015). "XSS (Cross Site Scripting) Prevention Cheat Sheet". Viewed 6th November 2015 at https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Lan et al. (2013). "Analysis and prevention for cross-site scripting attack based on encoding". pp. 102-105 Viewed 5th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6835463

Leff, A., J. T. Rayfield (2001). "Web-Application Development Using the ModelNiewlController Design Pattern", Enterprise Distributed Object Computing Conference, pp. 118-127 viewed 1th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=950428

Malviya, V. K., S. Saurav, A. Gupta et al. (2013). "On Security Issues in Web Applications through Cross Site Scripting (XSS)" pp. 583-588 Viewed 4th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6805456

Nadji, Y., P. Saxena, D. Song et al. (2009). "Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense". Viewed 6th November 2015 at http://www.cs.berkeley.edu/~dawnsong/papers/2009%20dsi-ndss09.pdf

Selfa, D. M., M. Carrillo, Ma. del Rocío Boone (2006). "A Database and Web Application Based on MVC Architecture", Proceedings of the 16th IEEE International Conference on Electronics, Communications and Computers, p. 48. Viewed 1th November 2015 at http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=1604744

Stivalet, Delaitre (2014). "Php vulnerabilities test suite". Viewed 6th November 2015 at https://github.com/stivalet/PHP-Vulnerability-test-suite

Wang, G. (2011), "Application of lightweight MVC-like structure in PHP",
IEEE Conference Publications, pp. 74-77 vol 2, viewed 1th November 2015 at
http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=5917846

Wang, Yi, Zhoujun Li, Tao Guo et al. (2011). "Program Slicing Stored XSS Bugs in Web Application" pp. 191-194
viewed 5th November 2015 at
http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6041609

Appendix B

The advantages of using JavaScript for full stack development with an emphasis on Node.js

Galway Mayo Institute of Technology

Software Development

Research Methods

The advantages of using JavaScript for full stack development with an emphasis on Node.js

Author:
Ronan Connolly

Supervisor:
Dr. Sean Duignan

December 7, 2015

Contents

1 Introduction

Currently JavaScript is the only language that we can use to develop both web based applications in both the client and the server side [5].

Using JavaScript throughout both the client and server allows developers to advance their knowledge to a deeper level than time would usually allow [1].

By using just JavaScript on the server side and for cloud side applications, we can promote software re-usability and bring some order to the chaos that is heterogeneity [2].

The quite recent introduction of Google's V8 Engine, CSS3, HTML5 and the lower hosting costs has altered the web a lot. Since these changes there has been an explosion in web-based applications being developed by even the most novice developer [3].

There are many server-side frameworks available such as Java's Spring, Python's Django, PHP's CodeIgnitor/Zend, Ruby's Rails and JavaScript's Node.js. Node.js (from now on to be referred to simply as Node) is special in the way that it solves well-known challenges in teaching web development. Node allows for the consolidation of language throughout the language stack. The platform supports a smooth learning curve, allowing developers to build their knowledge gradually through the use of modular, open source components. As web technologies progress, these new frameworks offer increased utility and distributed connectivity [1].

This short paper endeavors to clarify some of the thinking behind using JavaScript for full-stack development and beyond. Full-stack meaning where JavaScript is used on both the client and the server. Beyond meaning using JavaScript for many other uses, utilising NoSQL technologies such as CouchDB and MongoDB, using packages like Fuzzy.io for Artificial Intelligence, or Johnny Five for the Internet of Things. There is such a vibrant community of JavaScript developers out there pushing the threshold of what JavaScript is and can do. With the soon arrival of ECMA Script 2015 (known to many as JavaScript6 or ECMAScript 6), things are only going to improve for the JavaScript community. Many ideas and Object Oriented Fundamentals like lexical scope binding, classes and sets/maps are included in the update. I will describe the asynchronous nature of Google's V8 nature, how node has taken advantage and why it's pushing JavaScript forward as a winner for full-stack development. I will also touch on a few problems people

may encounter on their endeavors with JavaScript and Node.

2 Heterogeneity

Stepp [9] has published informal survey results which indicate HTML/CSS and JavaScript are covered in virtually every course covering client-side development - their dominance on the client-side leaves little choice in adoption. There is little consensus on the server however, with PHP, Java/JSP, Ruby on Rails, ASP.NET, and Python being common choices among educators [1].

Having JavaScript already being the number one choice for client side development, NoSQL technologies primarily being used with JavaScript and with Node rising in popularity, it makes sense to consider using JavaScript for full-stack development.

It can take quite a bit of time constantly switching between different frameworks and languages. If you are using just one language and a handful of similar frameworks (more then likely in the same community of developers), you will be able to gain a deeper understanding and come up with more interesting and vibrant solutions.

We should remove the trivial and mundane task of doing a context switch from one language to another. Edsgar Dijkstra believed that programming wasn't about writing code but about coming up with new solutions to problems: "In order to compose, you have to write scores, but to be a composer is not to write scores; to be a composer is to conceive music "[10]. It was a different time when he said this, C was the higher language and he was referencing machine code as being analogous to writing scores but the point still prevails, the need to remove trivial mundane tasks is of utmost importance, this leads to smarter solutions for more efficient code.

3 JavaScript

The original and dominant use of JavaScript remains in web applications run in a web browser. Here, events are emitted in response to user actions like mouse clicks or keyboard presses, and internal browser events like the completion of a network

request. Events are propagated along the structure of the HTML document, using the event capturing and bubbling strategies. Listeners are attached to HTML elements and can cancel (stop the propagation) of events programatically.

With the event-driven JavaScript interpreter like Google's V8 VM that Node is based on, applications register an interest in certain events, like when data is ready to be read on a certain socket. Asynchronous I/O is important for event-driven programming because it prevents the application from getting blocked while waiting for an I/O operation. Event-driven programming can be problematic just like multi-threaded programs. One of these problems is that not all inter-process communication can be tied into the event notification facilities. The sheer complexity of writing asynchronously in some languages like C can be incredibly perplexing. As Tilkov et al. states "Many such applications end up being little more than impenetrable, un-maintainable tangles of spaghetti code and global variables." [4].

> *By the way, on the matter of hype: People have always been quick to announce "the next software development revolution," usually about their own brand-new technology. Don't believe it. New technologies are often genuinely interesting and sometimes beneficial, but the biggest revolutions in the way we write software generally come from technologies that have already been around for some years and have already experienced gradual growth before they transition to explosive growth. This is necessary: You can only base a software development revolution on a technology that's mature enough to build on (including having solid vendor and tool support), and it generally takes any new software technology at least seven years before it's solid enough to be broadly usable without performance cliffs and other gotchas. As a result, true software development revolutions like OOP happen around technologies that have already been undergoing refinement for years, often decades. Even in Hollywood, most genuine "overnight successes" have really been performing for many years before their big break.* *Herb Sutter [11]*

Whatever you think about JavaScript as a programming language, there is little doubt that it has come about as being the dominant language of choice when

doing any web development work or any HTML based application such as Electron (desktop apps) or Ionic (mobile apps).

Server-side JavaScript is a logical next step, enabling the use of a single programming language for all aspects of a distributed web-based application. Server-side JavaScript isn't yet a mainstream approach as it's only recently gotten huge exposure due to Node.

3.1 ES2015

ES2015 is the newest update to arrive for JavaScript, it arrived in June 2015. It is the biggest update since the introduction of JavaScript by Netscape (Brendan Eich) in 1995. It was originally planned to be called ECMAScript6 but they revised the naming convention and changed it to ES2015, and from now on there will be a release every year following the new naming convention. ECMAScript6: is the next version of JavaScript standard, it brings a variety of features found in other languages such as Java and C for creating high-performance applications to JavaScript [2]. Two really important features are Promises and Typed Arrays.

Asynchronous programming introduces a simplified way to handle operations, as opposed to conventional callback-based approaches. "Promises" bring a structured flow to applications by chaining events so that the order of events is guaranteed. It does so with the aid of a newly introduced method called "then", which takes two parameters - success and error callbacks [2].

Promises have being available through external libraries such as "Q" and "Async" but now they will be introduced fully into the language.

With the upcoming introduction of "Promise as a language feature", we expect an increase in interest, and believe that many developers will shift to this better practice of using promises rather then callbacks as stated by E. Brodu et al. in [7].

Promises were introduced to solve the problem of the constant use of callbacks that leads to intricate imbrications of function calls and callbacks, commonly known as "callback hell" or the "pyramid of doom". Basically where there can be a mess of nested callbacks in the code.

4 Node.js

Node is a hugely successful platform that combines the popular JavaScript language with an efficient runtime tailored for a cloud-based architecture. JavaScript has many advantages for web development. It is the de facto dominant language for client-side applications and it offers the flexibility of dynamic languages. In particular it allows the easy combination or mash-up of content and libraries from varying third parties[12]. This works perfectly with Node as we can easily plug in different modules with a simple "npm install ⟨package⟩" from the command line. It has never been so easy to go from design and development to testing and production.

Currently, a server environment for running a Node application can be easily created and/or accessed on a cloud server, and thus, a run-time environment is obtained without the user/developer explicitly having to install any software. This may result in a change in the business model of software applications [13].

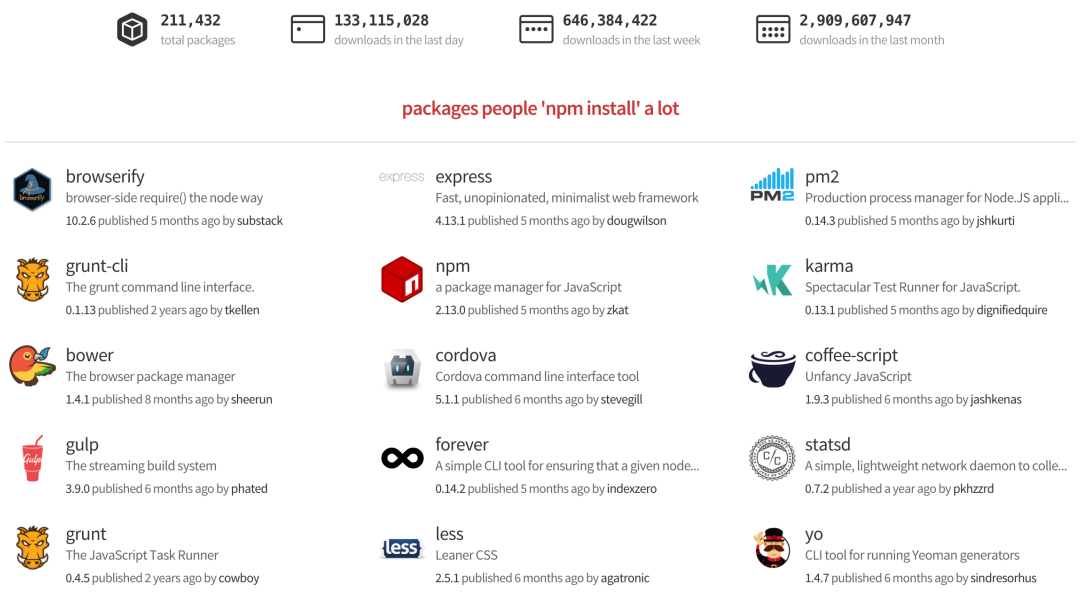


Figure 1: NPM Homepage

As eveidence to the popularity of Node, the NPM package repository has recently surpassed Maven Central and Ruby Gems with more than 211,377 available

packages and over 250 packages added every day [8]. Just last month as you can see from figure 1 above, there were 2,909,607,947 downloads from the NPM package repository.

Some of the most popular NPM packages are “jade”, “grunt”, “mongoose”, “socket.io”, “express”, “browserify” and “gulp”. These packages contain many packages themselves, creating a kind of tree like structure. Packages within packages, etc. Many of the mentioned packages contain more then 200 libraries each. “express” includes 138 packages [12].

Among the various server-side frameworks, Node has emerged as one of the most popular. Its strengths are the use of JavaScript, an efficient run-time tailored for cloud-based event parallelism, and thousands of third-party libraries [12].

4.1 Interpreter and Threads

First what is the JavaScript interpreter and who built it? Node is a JavaScript interpreter based on Google’s V8 virtual machine. It’s designed to use asynchronous input/output by default, including an event model which makes event-driven programming really easy. Since it is basically a JavaScript interpreter, sequential and synchronous programs are possible and, because of the speed of the underlying JavaScript virtual machine, it has high performance, but the best way of utilizing it is by taking advantage of the asynchronous I/O features that make it different from other JavaScript interpreters such as Spider-monkey or Rhino and, in fact, closer to the event-driven programming that is usual in browsers [6].

You should try to think of the Node server process as a single-threaded daemon that contains the JavaScript engine to support customization. This is quite different from most eventing systems for other programming languages, which come in the form of libraries. The eventing model in Node is supported at the language level. JavaScript is great for this approach because it supports event callbacks. For example, when a browser completely loads a document, a user clicks a button, or an Ajax request is fulfilled, an event triggers a callback. JavaScript’s functional nature makes it extremely easy to make anonymous function objects that you can register as event handlers [4].

Although many developers have successfully used multi-threading in production applications, most of them will agree that multi-threaded programming is very

difficult. It has many problems that can be difficult to find and fix, such as deadlock and failure to protect resources shared among threads. Developers also lose some degree of control when drawing on multi-threading because the OS typically decides which thread executes and for how long. Event-driven programming offers a more efficient, scalable alternative that provides developers much more control over switching between application activities [4].

4.2 Why Node?

The I/O approach Node takes is strict. Its Asyncronous interactions aren't the exception, they're the rule. All of Nodes I/O operations are handled by higher-order functions (functions taking functions as a parameter), that state what happens when there is something to do [4]. JavaScript is a functional language (but not purely functional) and as such, supports higher-order functions. When writing Node programs you will see functions everywhere. The main flow of the program is

The programs main flow is set by the functions that are explicitly called. These functions never block anything I/O related, but rather register appropriate handler callbacks. If you've seen a similar concept in other eventing libraries you may be wondering where the event loop hides?

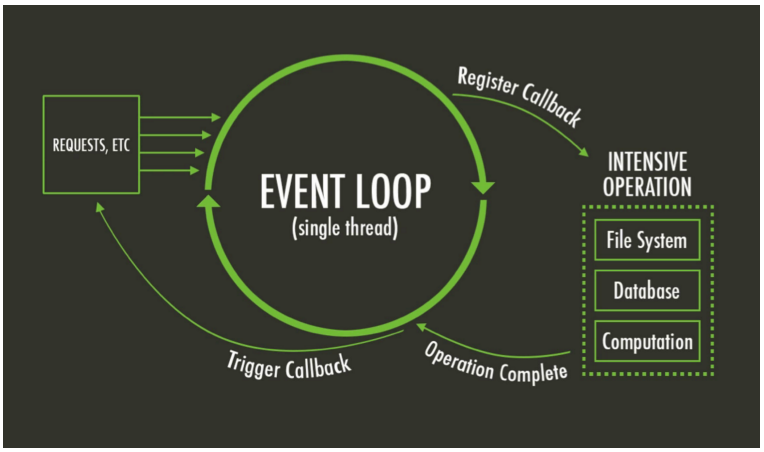


Figure 2: Node Event Loop

The event loop idea is so important in Node's behavior that it hsa hidden the implemenation of it. The main purpose of the program is to simply set up appropriate handlers [4]. Lots of registration calls happen in the loop, no actual I/O happens. By having asyncronous I/O as the default forces

developers to use the asyncronous model from the beginning, this is one of the main differences between other Node and asyncronous I/O in other programming environments, where this would be thought of as too complex or advanced.

Google's V8 engine (created in C++) on which Node is based uses JIT (just in time) compilation to achieve near-native performance, hugely outperforming traditional interpreter-based approaches[1].

Importantly, the V8 engine (created in C++) on which Node is based uses JIT (just in time) compilation to achieve near-native performance, vastly outperforming traditional interpreter-based approaches[1].

In most web development projects JavaScript knowledge is a prerequisite and the option of usin just one programming language for everything becomes quite tempting. Node's architecture makes it very easy to yse a highly expressive, functional language for server-side programming, without any sacrifice on performance [4].

Node deeply embraces the Unix style of programming, encouraging small, interchangable modules. When you look closely at an many Node programs, you will see similar concepts to those seen in Unix, like for instance piping input and output streams in sockets, files and stdin [1].

What's unique about Node is it's ability to provide a high level of language consistency using JavaScript across all facets of the stack (back-end, front-end, database, etc). It provides a distinct mix of high-level abstraction and at the same time an exposure to low level detail [1].

5 Is JavaScript always the best?

I have laid out arguments about the speed of JavaScript through the use of Google's V8 engine and in such the speed of programs using the Node platform. The V8 runtime environment on which Node is based (and javascript applications run in Google's Chrome browser) is extremely fast and achieves near native speed due to JIT (just in time) compilation. JIT works by compiling certain sections of code based on heuristics laid out by the runtime environment. These heuristics could be based on code segments that are more likely to be used. When code is compiled is is kept in memory in a cache (so as not to have to compile it multiple times).

This gives applications near native performance, everybody can agree that Oracles JVM JIT compilation gives excellent speeds, but that is compiling "bytecode", in V8 we are compiling source code. Runtime environments utilising JIT compilation tend to have a "warming up" phase where the program may lag abit, then once a sufficient amount of code is compiled the near native performance kicks in.

There is a spectrum between native and interpreted languages. Native sits on one side and interpreted on the other. There will always be a space for native code (developed with the likes of C++) where performance is key like for things such as gaming and stock trading, but for portability, scalability, testing and ease of use JavaScript is on top, although you do take a performance hit due to the interpretation that takes place. Then you have intermediately compiled languages like Java that sit in the middle of the spectrum, which compiles to bytecode, is portable (to any machine that has a JVM set up) and achieves near native performance (through JIT compilation).

Since all you need to run JavaScript is a browser (which every computer has, compared to Java where you have to install the JVM) and with the recent introduction of object oriented concepts and new collection types in ES2015 JavaScript is started to get closer performance to Java.

This is a space to be watched.

6 JavaScript Beyond

We now understand the power of JavaScript on the server and client side but there are many other areas that developers can transfer their JavaScript knowledge to.

With the introduction of WebGL (an implementation of OpenGL in JavaScript), JavaScript can be used to create amazing simulations and 3D games within a web browser.

NoSQL (not only SQL) technologies like CouchDB, MongoDB and Neo4J are becoming more and more popular. MongoDB stores it's data in JSON format and uses a JavaScript syntax for all of it's administration & querying facilities; and so do many other frameworks [1].

The Cordova project has brought the power of low level phone components (called hooks) into the hands of JavaScript developers, hooks allow developers to use things like GPS, Bluetooth, Gyroscope and other hardware components in a

webview within an app. This means you can create an entire mobile application using JavaScript. Adobes PhoneGap has built off of Cordova, and the very successful Ionic is built on top of Cordova. The Ionic team have created an amazing framework for building modern mobile applications with ease in JavaScript, mobile apps that can truly stand up against native apps.

Achieving competency in JavaScript allows a developer to quickly become productive with these growing technologies.

7 Conclusion

I have talked about the chaos of heterogeneity and the issues it brings, the benefits of having one language across many frameworks, how the JavaScript V8 Interpreter works and why Node has benefited from it so vastly. Node is only going to get stronger, more and more libraries are being created everyday, the new JavaScript standard is just out recently, it is based on Object Oriented Principles, if developers think the recent rise in JavaScript is short lived then think again, this is only the beginning for this little scripting language that was released 19 years ago. It is debatable whether Node is always the best choice at the professional level as sometimes you need that full native perfomance that is not possible with an interpreted language, however it is clear that JavaScripts ubiquity provides a level of consistency simply not possible when adopting other server-side languages [1].

References

[1] S. Frees. A Place for Node.js in the Computer Science Curriculum, Consortium for Computing Sciences in Colleges, 2015.

[2] R. Karthik. SAME4HPC: A Promishin Approach in Building a Scalable and Mobile Environment for High-Performance Computing, ACM SIGSPATIAL, 2014.

[3] B. Anderson. CoPerformance: A Rapid Prototyping Platform for Developing Interactive Artist-audience Performances with Mobile Devices, MobileHCI, 2014.

[4] S Tilkov, S. Vinooski. Node.js: Using JavaScript to Build High-Performance Network Programs, IEEE Computer Society, 2010.

[5] J. Merelo, A. Esparcia-Alczar, V.Rivas-Santos. Assessing Different Architectures for Evolutionary Algorithms in JavaScript, GECCO, 2014.

[6] J. Merelo, A. Esparcia-Alczar, V.Rivas-Santos. NodEO, a Multi-Paradigm Distributed Evolutionary Algorithm Platform in JavaScript, GECCO, 2014.

[7] E. Brodu, S. Frnot, F. Obl. Toward automatic update from callbacks to Promises, AWeS, 2015.

[8] M. Madsen, F. Tip, O.Lhotk. Static Analysis of Even-Driven Node.js JavaScript Applications, OOPSLA 2015.

[9] M. Stepp, J. Miller, V. Kirst. A "CS 1.5" introduction to web programming, Proc. of the 40th ACM technical symposium on Computer Science education, 2009

[10] E. Dijkstra, Turing Award Speech, 1972.

[11] H. Sutter. A Fundamental Turn Toward Concurrency in Software: http://www.drdobbs.com/web-development/a-fundamental-turn-toward-concurrency-in/184405990, 2005.

[12] W. Groef, F. Massacci, F. Piessens. NodeSentry: Least-privilege Library Integration for Server-Side JavaScript, ASAC 2014.

[13] S. Okamoto, M. Kohana. Rapid Authoring of Web-based Multiplayer Online Games, iiWAS, 2013.

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] Php usage. http://php.net/usage.php. [Online; accessed 22-April-2016].

[9] T. Suzumura, S. Trent, M. Tatsubori, A. Tozawa, and T. Onodera. (2013) Performance comparison of web service engines in php, java and c. http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=4670199. [Online; accessed 22-April-2016].

[10] (2016 (accessed on 2016-04-17)) Cordova homepage. https://cordova.apache.org/.

[11] (2016 (accessed on 2016-04-17)) Jquery mobile homepage. https://jquerymobile.com/.

[12] (2016 (accessed on 2016-04-22)) Ionic homepage. http://ionicframework.com/.

[13] (2016 (accessed on 2016-04-22)) In memoriam - edsger wybe dijkstra. https://www.utexas.edu/faculty/council/2002-2003/memorials/Dijkstra/dijkstra.html.

[14] (2016 (accessed on 2016-04-22)) Heroku homepage. https://www.heroku.com/.

[15] (2016 (accessed on 2016-04-22)) Module counts. http://www.modulecounts.com/.

[16] (2016 (accessed on 2016-04-22)) Git homepage. https://git-scm.com/.

[17] (2016 (accessed on 2016-04-22)) Manifesto for agile software development. http://agilemanifesto.org/.

[18] (2016) Zend Framework 2 Overview. http://framework.zend.com/about/. [Online; accessed 13-April-2016].

[19] 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-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6634012&tag=1. [Online; accessed 13-April-2016].

[20] H. J. La and S. D. Kim. (2013) Balanced mvc architecture for developing service-based mobile applications. http://0-ieeexplore.ieee.org.library.gmit.ie/stamp/stamp.jsp?tp=&arnumber=6634012&tag=1. [Online; accessed 13-April-2016].

[21] https://developers.google.com/chart/infographics/docs/qr_codes#overview. [Online; accessed 16-April-2016].

[22] https://packagist.org/packages/adminweb/qrcode-zf2-module. [Online; accessed 16-April-2016].

[23] https://github.com/raykolbe/DOMPDFModule. [Online; accessed 16-April-2016].

[24] http://pdfmake.org/index.html#/gettingstarted. [Online; accessed 16-April-2016].

[25] O. Corporation. (2016) Top reasons to use mysql. https://www.mysql.com/why-mysql/topreasons.html. [Online; accessed 13-April-2016].

[26] N. Adermann and J. Boggiano. (2016) Composer documentation. https://getcomposer.org/doc/. [Online; accessed 13-April-2016].

[27] http://www.wampserver.com/en/. [Online; accessed 16-April-2016].

[28] (2016 (accessed on 2016-04-17)) Html mdn homepage. https://developer.mozilla.org/en-US/docs/Web/HTML.

[29] (2016 (accessed on 2016-04-17)) Css mdn homepage. https://developer.mozilla.org/en-US/docs/Web/CSS.

[30] (2016 (accessed on 2016-04-21)) Html/css codeschool path. https://www.codeschool.com/learn/html-css.

[31] (2016 (accessed on 2016-04-21)) Html/css codeschool path. https://www.codeschool.com/learn/javascript.

[32] (2016 (accessed on 2016-04-17)) Javascript mdn homepage. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[33] 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.

[34] (2016 (accessed on 2016-04-21)) React homepage. https://facebook.github.io/react/.

[35] (2016 (accessed on 2016-04-21)) Ember homepage. http://emberjs.com/.

[36] (2016 (accessed on 2016-04-21)) The ionic show // episode 1. https://www.youtube.com/watch?v=uJAWaE11Jf4e.

[37] (2016 (accessed on 2016-04-21)) John papa's angular style guide. https://github.com/johnpapa/angular-styleguide.

[38] (2016 (accessed on 2016-04-22)) ios store. https://itunes.apple.com/us/genre/music/id34.

[39] (2016 (accessed on 2016-04-22)) Android (google play))store. https://play.google.com/store?hl=en.

[40] (2016 (accessed on 2016-04-22)) Windows phone store. https://www.microsoft.com/en-us/store/apps/windows-phone.

[41] (2016 (accessed on 2016-04-17)) Yeoman homepage. http://yeoman.io/.

[42] (2016 (accessed on 2016-04-23)) Angular generator. https://github.com/yeoman/generator-angular.

[43] (2016 (accessed on 2016-04-17)) Grunt homepage. http://gruntjs.com/.

[44] (2016 (accessed on 2016-04-23)) Gulp homepage. http://gulpjs.com/.

[45] (2016 (accessed on 2016-04-23)) Sass homepage. http://sass-lang.com/.

[46] (2016 (accessed on 2016-04-17)) Jasmine homepage. http://jasmine.github.io/.

[47] (2016 (accessed on 2016-04-17)) Karma homepage. https://karma-runner.github.io/0.13/index.html.

[48] (2016 (accessed on 2016-04-17)) Bower homepage. http://bower.io/.

[49] (2016 (accessed on 2016-04-17)) Npm homepage. https://www.npmjs.com/.

[50] (2016 (accessed on 2016-04-17)) Bash homepage. https://www.gnu.org/software/bash/.

[51] (2016 (accessed on 2016-04-17)) Phonegap homepage. http://phonegap.com/.

[52] (2016 (accessed on 2016-04-17)) Xamarin homepage. https://www.xamarin.com/.

[53] (2016 (accessed on 2016-04-17)) Cloudant homepage. https://cloudant.com/.

[54] (2016 (accessed on 2016-04-17)) Couchdb homepage. http://couchdb.apache.org/.

[55] (2016 (accessed on 2016-04-17)) Nodejs homepage. https://nodejs.org/en/.

[56] (2016 (accessed on 2016-04-17)) Expressjs homepage. http://expressjs.com/.

[57] (2016 (accessed on 2016-04-17)) Github homepage. https://github.com/.

[58] 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].

[59] (2016 (accessed on 2016-04-23)) Json homepage. http://www.json.org/.

[60] (2016 (accessed on 2016-04-17)) Http requests information. http://www.w3schools.com/tags/ref_httpmethods.asp.

[61] https://github.com/arvind2110/ZF2-Auth-ACL. [Online; accessed 16-April-2016].

[62] (2016 (accessed on 2016-04-24)) How do we tell truths that might hurt? http://www.cs.virginia.edu/~evans/cs655/readings/ewd498.html.