Keele University

Developing a multi-platform mobile application using web-based languages for Fletcher Moorland Ltd

Callum Neil Lucking

MSc in Advanced Computer Science 2017-18

14th September 2018

SCHOOL OF COMPUTING AND MATHEMATICS
Keele University
Keele
Staffordshire
ST5 5BG

Abstract

The intent for this project was to plan, design and develop a mobile application for a mechanical repairs company which was looking to innovate in an industry that is often slow to adapt to new technological methods and practices. The application was designed to be customer facing, and, enable the customer to issue new collection requests on broken units, view all orders and provide the customer with any necessary information about their order or about the company. The project will first examine and review the different mobile application development routes available, starting with a look at the differences between native and multi-platform applications. Further reviews take place on the development methodologies available, frameworks which were considered and used and the industry's lack of technological innovation. After these examinations, the project will discuss and evaluate the development of the application, from initial discussions that occurred, to the planning, development and testing of the mobile application itself. Throughout this evaluation, analysis on key components are highlighted which would involve the design choices of the application or the fundamental code design within. Also, comparisons are made on how the best practice choices for development have been used, as well as identifying when they haven't or couldn't be used. Finally, a review of the success of the application takes place, with further discussions on potential new functionality that this company could develop in the future.

Acknowledgements

I would like to thank the company Fletcher Moorland for allowing this project to happen and the support that Matthew Fletcher, Diane Mansell, Phil Nolan and Elliott Broomhall have provided during this time.

I would also like to thank my tutor Charles Day for providing helpful advice throughout the project and allocating time to discuss the progression of both the application and the academic work.

I would like to further express my gratitude to the Ionic team for allowing me to use their open-source framework which was vital during the application development. Also, thanks to Figma and AMPSS for allowing use of their software respectively which allowed for a greater ease when presenting and developing alike.

Finally, I would like to thank my family for their support throughout this project and helping me to focus on both the development of the application itself and through the writing of this report. Also, a special mention to Lauren Millington who assisted in proof-reading and fixing grammatical errors.

Contents

List of Figures

1 Introduction

1.1 Background

Fletcher Moorland (Ltd, F. M, 2018) is a company located in Stoke-On-Trent that specialises in electronic and electro-mechanical equipment repairs. The company also takes pride in offering 24/7 open hours and support for its customers. Within an industry that is renowned for being slow to adopt modern technical practices, Fletcher Moorland strive to be innovative at all corners to enhance their customers' experience with them. As part of this innovation strategy, Fletcher Moorland have looked to incorporate a mobile application where the clients can request collections of broken machines or units, track their order progress and contain further quality of life features. These features could include instant contact to relevant teams, sections to highlight further business that Fletcher Moorland can offer and contain rich video information on how the client's machine is being fixed. This project aims to design and develop a mobile application which could be used by Fletcher Moorland's clients as well as evaluate and analyse options available when it comes to the application development.

1.2 Aim

To design & release a mobile application for Fletcher Moorland that allows customers to request collection of faulty units and check all orders that have been requested.

1.3 Objectives

The following objectives were the result of a culmination of discussions with Fletcher Moorland on where the company would ideally see as the outcomes of the project:

1. Creation of a functional application - ensuring all functional aspects work effectively

2. Implementation of the Ionic Framework within the application

3. In-depth wireframe for final iteration of the mobile application

4. Completion of a minimum functional application to be released onto mobile platforms

1.4 Literature Review

1.4.1 Cross Platform Vs Native

The first decision to be made when looking to develop a mobile application is whether to follow a native or cross-platform route. The native route looks to develop an application specifically for a certain operating system, such as Android, by designing in that operating systems language, in Android's case this would be C#. The cross-platform route looks to develop using a software such as Cordova (Cordova, A, 2018) to act as a middleware. This middleware allows the use of web-based languages (HTML, CSS, and JavaScript) to be used to write an application as if it were a website. When the application is compiled by the software, it is then translated and packaged into a code base for any mobile operating system chosen. Andre Charland (2011,) discusses the relevant pros and cons for native or cross-platform app development.

Charland found the major factors that come into play when comparing these are the focus on cost, time to develop and the resources which are available to the developer or business. Notably, resources in this context refers to both the knowledge of the developer as well as the resources of hardware. The leading idea is that if a developer has a low-cost barrier, shorter development time until release and limited resources, then a cross-platform application will in most cases be the optimal route to go down. On top of this, only needing the knowledge of web languages means another lower entry language barrier for developers.

Figure 1: Comparison of languages & tools required between cross-platform & Native applications (Puthiya, D, 2014)

The same considerations needed to be applied to assess which development route to be used for the application. Firstly, Fletcher Moorland is a small to medium sized business, meaning that the cost factor needed to be considered, especially with only one developer initially working on the application. Secondly, the factor of time constraints had to be considered due to the short timeframe for the project. Finally, the Fletcher Moorland development team, consisting of two members, were familiar with web-based languages due to the need for website and Business Operating System (BOS) design and upkeep. Therefore, implementation of a web-based application would allow for an easier integration with Fletcher Moorland's team and allow for a smoother

transition post project. An additional assessment of the complexity of the application would also provide some insight into whether to opt for native or cross-platform development. With Fletcher Moorland, the application was designed to serve the clients with the ability to issue a unit collection request, view orders which are pending or active, and allow for ad-hoc information to be provided. Due to this set of functionalities, there was not a need to make high levels of manipulation directly with the operating system.

With cross-platform development selected as the route to go down, there was a further sub-layer of two choices to be made. These were between developing using native cross-platform tools or using the hybrid web-based languages, usually falling between the decision to develop using Xamarin or Cordova. An article by Medium studies the differences between the two cross-platform styles (Klubnikin, A, 2017) and from this analysis, the biggest differences were found to be the coding languages used and the access to native libraries. With the higher level of access to native libraries from cross-platform native development, this can lead to higher performance levels. However, as the software of web-based cross-platform frameworks improve, so do the plugins and application programming interfaces (API) which are used to access those native libraries, allowing for a greater performance level. Ultimately, the choice made falls to the knowledge of the developer and the complexity of the application design in question. With these considerations in mind, the web-based cross-platform route was chosen as the ideal tool base for this project.

1.4.2 Methodologies

Before heading into the core of the development, there was a need to implement a key methodology which would direct the development cycle during the course of the project. When it comes to software development, an agile methodology tends to be the most popular choice due to its flexible nature and quick evolutionary styled development (Reynosa, J, 2017). However, the development of a mobile application has slight differences compared to most other software creations due to the need to develop quickly and ensure that there is a high compatibility between multiple operating systems. An article by the University of Oulu delves into the design and implementation of a "Mobile-D" methodology which involves an early delivery of an application, tough test-driven methods, multiple iterative development and user-focused feedback (Abrahamsson, P, et al, 2004). This hybrid style of agile methodology fits in well with the project's scope and was the chosen methodology for this applications life-cycle. Most notably, being able to illustrate to Fletcher Moorland how the application could potentially function, as well as its user interaction design, would be a key process in ensuring that the direction of the application would portray the ideals of both Fletcher Moorland and their clients.

A common methodology called "Waterfall" was a strong consideration to be implemented within this project. A journal of information technology and business management (S.Balaji, D. M. S. M, 2012) goes into detail on comparing waterfall and agile development life cycles, as well as providing an overview for each model. The first reason for its consideration for this project was due to the forcing of a clear set of requirements on the software from the start of development. This would ensure that there were clear targets to be achieved by the conclusion of the project which Fletcher Moorland would be able to compare against. A second reason would be that during the

different stages of the application development, there would be a need to produce clear documentation on those stages to ensure a greater quality throughout the lifecycle. This would have benefited Fletcher Moorland greatly when it came to handing over the application to the IT team after the project was completed as there would be more resources available on the applications design. However, the downfalls of this methodology were its linear progression from start to finish, and the strict structure required between completions of each phase. Having linear progression rather than a cyclical progression would not allow for large quantities of testing to occur throughout. As such, greater levels of testing would occur towards the end of the project which may highlight issues that affect the application severely. In contrast, the mobile-D methodology allowed for an iterative design enabling the constant evaluation of the application creating an improved development cycle over time.

1.4.2.1 Methodology implementation

To aid in tracking the progress of the project, as well as providing a helpful tool to Fletcher Moorland on the scope of the project, a Gantt chart was created to illustrate a weekly task list (fig. 2). The purpose of this Gantt chart was to ensure that a strict management of task completion was met, and, to facilitate in visually identifying certain large sets of tasks that would need to be accomplished to further the progression of the project and achieve the objectives set. This is especially the case when the project is based on software development due to the various underlying issues that can occur throughout. A study by Beyond IT Future (Krigsman, M, 2008) states that 68% of technology-based projects do not meet deadlines or will fail outright. A contributor to this percentage is a business's inability to analyse the project properly. Therefore,

having tools such as the Gantt chart, along with well-constructed aims and objectives, can put the percentage of success into the project's favour. A further strength of having a Gantt chart is that it gives a reference point for project progression so in meetings the company can visualise and understand at an in-depth level the decisions that have been made, as well as aiding in discussing delays and issues in the progression of the project. A downside to the use of a Gantt chart is that it can be difficult to create one that is not too simple or overly complex. If the Gantt chart is too simple, it may not contain much accuracy overall and in most cases not represent the majority of the tasks needing to be completed. If it is too complex, the issue is the exact opposite and there will be too many tasks broken down to effectively assign time too. The Gantt chart illustrates how the mobile-D style methodology was implemented, with a full first version, testing and evaluation of the application taking place within the first 6 weeks of the project.

	25th Jun	2nd Jul	9th Jul	16th Jul	23rd Jul	30th Jul	6th Aug	13th Aug	20th Aug	27th Aug	3rd Sep	10th Sep	17th Sep	24th Sep
Task	WEEK													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Project Plan														
first prototype - functional design														
Checking all functional parts link														
Review of prototype with client														
wireframe for 2nd prototype														
Poster														
second prototype creation														
Review of second prototype														
Testing & Quality assessment														
Finalize second prototype														
Release of first version to customers														
Report														
Documentation														

Figure 2: Gantt chart

Another analytic tool used at the outset of the project was a risk evaluation table (appendix 1), which is a set of preliminary educated assumptions on any potential risks that could occur, the probability of its occurrence, how to prevent the risk of occurrence and how to solve the situation (if it did occur). One example of concern to this project is the issue with frameworks being updated during development. If a plugin which is vital to the application becomes deprecated, then at the minimum, a knowledge of other

plugins which are considered best practice in the future would need to be implemented for the benefit of the application.

1.4.3 Analysis of Frameworks

As the popularity of using web-based languages to develop mobile applications increases, so has the software designed to help facilitate these implementations. One such software is an open-sourced, front-end framework called Ionic (Ionic team, 2018). This framework is designed to allow for native looking applications through the use of its user interface (UI) components as well as allowing use of existing Progressive Web Apps (PWA) for compatibility between devices and desktop (LePage, P, 2018**)**.

With the multitude of frameworks now available for PWA development, it can be difficult to decide which framework is best suited for use or even if a framework should be used at all. A competitor to Ionic is another framework called "Xamarin" (Petzold, C, 2015) owned by Microsoft, that uses C# as its code base for application development. In a ranking of the top 5 PWA frameworks (Trivedi, R, 2018), Xamarin didn't make it onto the list, whereas Ionic appeared as the second-best framework which indicates the ease of use and flexibility which the Ionic Framework has. The top-ranking framework was React Native (Eisenman, B, 2015) which has been rapidly growing interest over the past couple of years. However, when comparing Ionic to React Native, there is not much difference in terms of implementation aside for React using JavaScript and Ionic making use of Angular and TypeScript. On top of this, Ionic has been a strong framework knowledge base for many years now, meaning there is a greater depth of resources and information available when it comes to aiding in development and code design.

1.4.4 Industrial Review

Research into larger companies showcases exactly how stale in technology the mechanical engineering industry is compared to other industries, especially when it comes to technology designed to facilitate business to customer communication. Insight from Fletcher Moorland found that due to the nature of the industry, most customers are used to dealing with phone call-based arrangements, even more so when it comes to dealing with small businesses or individual clients. This is further proved by the fact that there are no large-scale applications for customers available from the top 10 engineering companies (Construction, B. D, 2016). From this set of information, it can be rightly assumed that there is a need for innovation, which Fletcher Moorland are actively looking to chase.

Regardless of the industry in which the application is being developed, the need for development which is user-centred as well as easily usable are must haves for the application to be successful. When it comes to commercial applications solely designed for a specific need with the general public, there were multiple issues to consider in aiming to obtain user retention such as aesthetics, ease of navigation and overall usability. An article by TechCrunch (Perez, S, 2015) mentioned that 84% of smartphone users spend the majority of their time only using 4 to 5 applications. Although this analysis involves commercial applications, the same principals can be applied when it comes to working with business to customer applications. Firstly, making sure there was a balance between personalising the application to make it associate with Fletcher Moorland and making sure it was clean and professional was tough to design. Furthermore, being able to minimize the time from opening the application to fulfilling the customers' needs would be a great measurement for assessing the success of its use.

2 Application Development

2.1 First version development - wireframe and overview of application

To ensure that the app would represent the functionality and aesthetics that Fletcher Moorland desired, a preliminary meeting occurred with the company owner, Technical director and the IT team. Within this meeting, the Fletcher Moorland IT team had created a drawn wireframe containing their thoughts on what the app should include. When it comes to projects with clients, the direction of the project tends to contain two potential routes at the outset. Firstly, the client such as Fletcher Moorland will have an idea from their industrial perspective on how the software or application should look, function and feel. Secondly, the developer themselves will have an idea on how this application should function from a technical stand-point as well as understanding the nuances and implications that the first design proposal may have. At this point, it is vital to obtain information from the client as to what they need from the project rather than what they want, which an article in Forbes explains (Roth, R, 2018). Being able to obtain this needed information will inevitably lead to a more successful and viable product for the client in the long term. In the case of Fletcher Moorland, they wanted to have collection requests, order views, live chats and potential links to all their existing products in place. From the discussions, it was clear that the customers' ability to issue a quick collection request and view their pending, current and archived orders would be the most important features. With these core functions in place, implementations of more ad-hoc functions, such as a live chat system, could be introduced further down the product life cycle span. This would further back up the implementation of an agile methodology by allowing for a clean environment where

new iterations would provide added functionality on-top of the first version without affecting any existing functionality.

A wireframe was used as a tool to aid in presenting information about the application to Fletcher Moorland, as well as being a useful document to refer too when the application was being developed. A software called Figma (Figma, 2018) was used to design this application wireframe due to its ability to act as an interactive page by page presentation style (fig. 3) as well as its ease of use for template design and the ability to generate a line map which displays the connectivity of the pages. As a presentation tool, it can be extremely useful in showing the client you understand their requirements, and, can be a great way to discuss ideas on what was liked or disliked due to it being an almost exact representation of how the application would look and function. In this case, Fletcher Moorland were able to highlight where more information needed to be provided, so that it would benefit the user. Also, it would allow Fletcher Moorland to see from a developer's point of view how the full application environment would work, such as how a customer would go about issuing a new collection request.

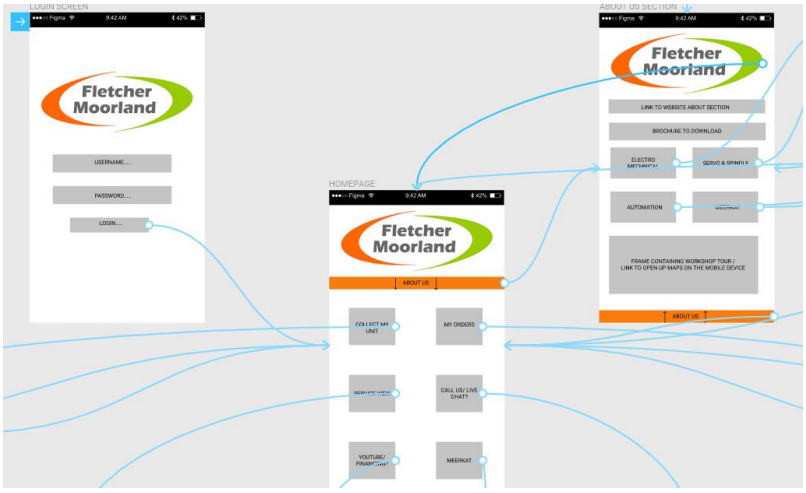


Figure 3: Snapshot of Figma navigation design for first version of application

2.2 Data-Structure

Ensuring that a well-structured database and it's relating tables were designed from the start were pivotal to the applications overall development. With Fletcher Moorland, a discussion was made on whether it made more sense to add new tables within their existing database or whether to create a completely isolated database which would operate within the same server. A combination of the two would be adopted with the Customers table being accessed from the existing database and all other tables being added into the new application database. Any information solely needed or obtained from the application would be a new table (such as orders requested). This would both avoid any duplications of tables and ensure that there was a clear segregation between the applications database and the current internal database.

Fletcher Moorland currently have a relational database setup for their internal server which runs with MySQL as a query language. With the application requiring standard tabular information, following the same setup would allow for an easier time with implementing into Fletcher Moorlands server. Another key aspect to consider with relational databases was the ability to use primary and foreign keys to allow for linked tables. An example of this within the application is when a request to collect a unit is issued by a user. Instead of having the base64 image string as part for the collections table, Fletcher Moorland requested that the images table be separated. As such, the use of a collection id as a primary key allowed for the images table to reference this as a foreign key, which allows for queries on a collection to be able to access both tables' data and insert new data correctly.

2.2.1 Localhost

Throughout both iterations of the application, a locally hosted server was used which held the database as well as the php API files. The software used to enable the locally hosted server was called AMPPS (Apache, MySQL, PHP, Perl, and Python) which was made by Softaculous (Softaculous, 2018). Being able to test locally made it perfect for a development environment as there was full control over both the front-end and back-end. Furthermore, if the locally hosted server and a mobile device were connected to the same Wi-Fi connection, then in most cases (dependant on the security of the Wi-Fi service), wireless testing could be carried out on the device. Finally, the locally hosted environment used was a close representation of how Fletcher Moorlands live server is setup. Therefore, when migrating the application database and php files into the live server, there would theoretically be no issues in compatibility. The downside to the use of a local host over a cloud provider was that testing was only available when the server was online and over Wi-Fi. In this scenario, being able to test the application on device with weaker network connectivity was difficult to assess and these issues could have a major affect when it came to live server deployment.

2.2.2 Data Life-cycle

One topic that surrounds all projects that deal with data is how to use this data during its life-cycle and what happens to it when the life-cycle ends. A standard structure for a data's life-cycle contains 7 stages (Bloomberg, 2015) which starts with the capturing of data and finish with the purging and archiving of data. In best practice, data would be captured, maintained, used, archived and purged to achieve data optimisation and ensuring there is a constant control when the data continues to grow in size. These

considerations need to be accounted for within a mobile application in the same way as any other form of data collection.

Fletcher Moorland currently backup all databases regularly, which follows the maintenance stage of data. However, data is not removed from the live servers and archived within a data warehouse upon finishing other stages of the life-cycle. This doesn't comply with Bloomberg's final stages and, overtime, can cause issues. The common issue is that the database becomes bloated with old data that is no longer used and can dramatically slow down queries which are being run. In the case of the mobile application, making a query to obtain client orders could lead to a large time delay as the query takes time to be completed due to more rows needing to be searched through. This can be fixed by setting a business standard archive date based off of the unit's completion date, and then removing this data and storing it in a data warehouse which can still be accessed if needed. This allows for greater efficiency within the live server, as well as complying with the 7-stage life cycle of data.

2.3 Data-Flow

There are a few differences in the way Ionic works as a framework when it comes to comparing it to a normal website structure. The key difference in structure is that with Ionic, the front-end and back-end frameworks are completely separate and contain no binding code compared to an integrated php page for a website. Figure 4 illustrates exactly how that theoretically works, and how the data is passed throughout the app. The highlight within this is that a middle API service will obtain data from the front-end set in a common language, which in this case and most cases is JavaScript Object Notation (JSON).

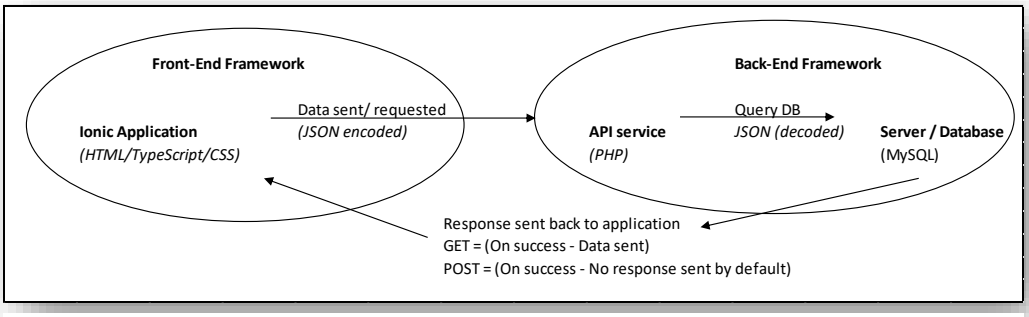


Figure 4: Illustration of how data is passed between frameworks

Manipulating the JSON data, both when sent and received, allowed for consistency through the application. The construction of the data when being sent to an api follows the required pattern by JSON, which is constructing key, value pairs with the value containing the relevant information needed by the api for its query to be run. For example, when a new collection request was made, a new variable was created which held an array of key, value pairs, with the key representing the title of the value, and the value itself containing either the user input or a local storage value. Figure 5 illustrates this construction of the JSON body in TypeScript and then sending that body within the POST request. Conversely, when data is being sent back to the application

from the API, it will follow the same construction, with some differences accounted for due to the construction taking place with PHP. A variable was still created but use of the "array" function then allowed to construct the key, value pairs obtained from the query. An extra step required in the API file was to pass the variable through the "json_encoded" function and then echo the newly encoded variable, so that it could be received within the application. This is illustrated in figure 6 which was part of the API file used to obtain customer information once they had logged in.

```
let body = { "serviceType": serviceType, "collectDate": collectDate, "instructions": instructions, "unitPhoto": imagePost, "userID": userID };

this.http.post(this.requestOrderUrl, body)
```

Figure 5: Code snippet of JSON construction for data to be sent to an API for a new collection request

```
$data = array('id'=>$id, 'username'=>$username, 'name'=>$name, 'secLvl'=>$secLvl, 'email'=>$email, 'customerId'=>$customerId);
$response = json_encode($data);
echo $response;
```

Figure 6: Code snippet of JSON construction for data that would be sent back to the application as a response

The strongest feature of using Ionic as a front-end framework is that due to this decoupling of communication, the developer is able to use any back-end that they wish to use and then setup an API service to act as a translator to the front-end. This works perfectly with Fletcher Moorland due to the constraints in place for using MySQL as a database language and ensuring that the app is flexible based on the internal server setup.

For Fletcher Moorland, a standard PHP file would sit inside the server and act as that API service for any requests made by the application, such as obtaining customer data or making a new collection request. A sample design of this API can be seen in (appendix 2), where the data is obtained, JSON decoded, processed and then a response is sent back to the application which is decided on whether the request was made successful or not.

Because of the decoupling of frameworks, the way in which a server request is managed on the front-end is different to a normal website request with php. Ionic makes the use of a plugin from Angular called "HttpClient" (Angular, 2018) which is a library designed to facilitate the request and response of get or post requests. After the request is sent, and a successful response has been obtained, the application needs to then access this response by subscribing to it. This is called an Observable and it allows for a greater control over the response, as well as gaining access to the response received and then dealing with it as seen fit by the developer.

Inside the application, all requests that were made were post requests due to the potential privacy of the data and more importantly, by sending through the user's ID, would ensure that any data obtained was only data relating to the current logged in user. An issue with post requests however was that they needed to have pre-flight headers in place which match the server's headers to allow for the communication to occur. This issue is highlighted in greater detail in the section handling POST request's (2.5.2).

2.4 First version development – creation and testing of application

2.4.1 Overview of app

In line with the mobile-D agile methodology in place for the project, fast creation of an initial application needed to first be developed. This initial application was in place to ensure that all functionality proposed by Fletcher Moorland, and what was showing on the wireframe, would actually work. Examples of crucial function testing included the ability to have persistent login, the ability to take and send a photo from the device camera to the database and being able to split the orders correctly with the relevant orders sections. Being able to manipulate the data within the application would be key

in having full control on how to organise the relevant pages. Furthermore, consistency from page to page regarding both aesthetics and functionality would benefit the further development of the application. Also, thorough testing of the http requests made to the APIs was a key point to ensure that when the live iteration was developed, it would have consistency with communicating with the database. Finally, developing a strong knowledge of the front-end framework would further aid in the optimisation of the live version, as well as being confident in using relevant framework components effectively.

2.4.2 Use of ionic

With the use of a structured framework in place, ensuring that best practice is used throughout can help in a multitude of potential issues. Firstly, like all frameworks, the use of best practice coding should lead to a more optimised application overall, as well as allowing for any potential developers taking over the project to understand exactly what they are looking at compared to the documentation provided. Also, the use of best practice should minimize any potential bugs throughout the life cycle of the application. An example of this would be if a developer did not make use of the grid system which Ionic uses to help create responsive designs. The developer may create a design which works on a certain screen size but fails to become responsive when this sizing changes due to not implementing the recommended grid component.

2.4.3 Key practices of Ionic

As with any framework in software development, Ionic has its own way with dealing with certain aspects of application functionality and design. One such case is how the framework handles the navigation between pages. With website design, a common method to navigate is the use of the “href” tag within html and pass a URL which links to the next webpage desired. With Ionic, page navigation is dealt with by using a stacking setup, where a user will effectively ‘push’ a page onto this stack, or ‘pop’ a page off the stack, with the initial page on the stacking representing the first page of an application upon opening up (fig. 7).

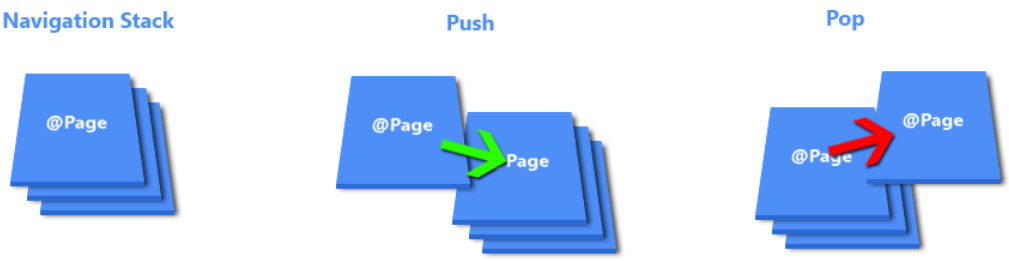


Figure 7: How navigation works in Ionic (McGivery, A, 2016)

Another feature of Ionic is that it allows for the same Document Object Model (DOM) design in the same way JavaScript works with HTML within a website. The feature behind this is it allows for 2-way manipulation; a user a can insert information into a text field to be accessed by the TypeScript file asynchronously and conversely, data within the TypeScript file can be accessed by the HTML file. Because Ionic uses Angular, allowing the 2-way manipulation occurs through the use of methods such as “*ngFor” or “[(ngModel)]” within the relevant tags in the HTML file. Assigning this as a variable within the TypeScript class joins the communication between them (fig. 8).

```
<div [ngSwitch]="orders">
  <ion-list *ngSwitchCase="'Current'" color="primary">
    <button ion-item *ngFor="let cData of currentData" (click)="currentOrder(cData)">
      <ion-thumbnail item-start>
        <img [src]="cData.UnitImage | safeUrl" />
      </ion-thumbnail>
      <h2>ORDER #{{cData.CollectionID}}</h2>
      <p><i>{{cData.Priority}}</i></p>
      <p><i>Due Date: {{cData.CollectionDate}}</i></p>
    </button>
  </ion-list>
```

Figure 8: Code snippet showing the use of angular methods inside the html

2.4.4 Navigation choices

Fletcher Moorland would provide their clients with direct login details so there was no need to implement a register page for the application. The home page would act exactly as it should do; as a way for the client to navigate to the various pages provided within. Being able to navigate quickly between the required pages would also be a strong feature to implement to make the application as desirable as possible to the client. This desire for quick navigation would increase the applications usability and allow for quicker times in requesting collections of broken units or being able to access orders by the client.

2.4.5 Testing of application

With the use of web-based languages, testing can be seen from a few different view-points. A great starting point when it comes to testing an application is by using a browser to deploy the application as if it was a website. Ionic uses the Command Line Interface (CLI) (Ionic team, 2018) to assist in testing, building and deploying the application, amongst other features such as quick app or page generation. Using the command “Ionic Serve –lab” will open the browser with the application inside an emulator of any chosen platform (fig. 9). However, many of the plugins used would

make use of Cordova plugins designed to link the application with the native phone functions, such as the camera or touch screen etc. In this case, the use of usb debugging would download a debug apk file of the application onto the device, and then through remote control within google chrome, run the application and generate console logs, network views etc too aid in testing.

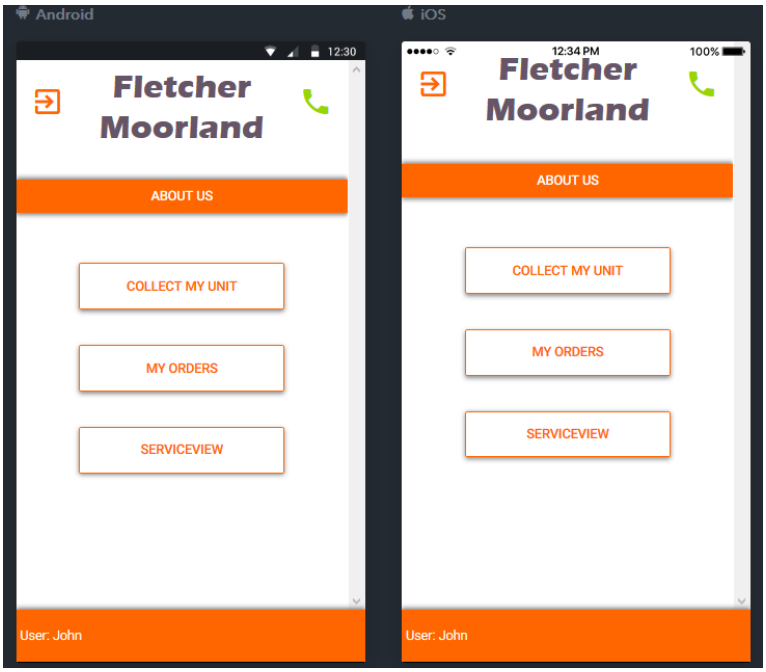


Figure 9: Preview of application in android & IOS emulators using Ionic Lab testing

2.5 Issues aware of

2.5.1 Testing on device

During the development, the majority of Wi-Fi access time was from Keele University, which had strict connection access and policies. The major issue here was that because of the use of a localhost, trying to connect to the database from an external source, such a mobile device, was impossible as it would not have rights to access the specific IP Address of that locally hosted machine. Therefore, any testing required on a device

would need to have occurred during hours within Fletcher Moorland. Knowing that this was an issue meant that development plans could be designed around times of access to a more open wi-fi connection. In particular, being aware of how device plugins such as the camera or pop-over pages work, meant that these could still be implemented at any point, and then tested more thoroughly during those access hours. If there was a need to urgently test on device, a device hotspot could have be used to act as the joining wi-fi connection between server and application. This was the case when needing to ensure the camera was sending data correctly before being able to progress with the application.

2.5.2 Handling post requests

As discussed in the section Data Flow, the Angular library "HttpClient" was used to facilitate the actions of sending POST or GET requests to the API. The cause for most issues within this was both making sure the data was sent in the correct format, as well as then ensuring the API could handle the data sent and respond accordingly. The issue that is triggered is called Cross-Origin Resource Sharing (CORS) (Hossain, M. & Hausenblas, M, 2018) and it is designed by browsers to ensure that any requests sent by the application source have permissions to access the server. These are sent as pre-flight headers which look to check for this issue via the use of header options (fig. 10) and upon success, will then proceed to allow the body of the request to be sent. The easiest fix to this solution is to install a browser plugin which forces itself to "Allow-Control-Allow-Origin: *" which will allow access to the API URL from any source. For testing purposes on a locally hosted system, this is a great solution. However, before deploying onto a live server, changing this origin to only allow access via the

application would be a greater security method. Another best practice, which was not implemented in this application, was to create a proxy to give greater control over configurations as well as ensuring a higher level of security. The reason for this was that once the headers were correctly working, there was limited time available in the scope of the project timeframe to attempt to implement this.

```
let headers = new Headers();
headers.append("Accept", 'application/json');
headers.append('Content-Type', 'application/json');
let options = new RequestOptions({ headers: headers });
```

Figure 10: Code snippet example of headers being created within the TypeScript file

The other issue was that due to the de-coupling of front-end and back-end frameworks, the use of JSON as a common language was vital. Within the php API, this de-coupling was handled by json decoding the data upon request and sending a response back to the application in a json encoded format. Handling the response itself is also crucial to maintain consistency per request. If the request requires data to be passed back, this needs to be properly formatted as a key, value pair to match the json encoding method used. Otherwise, clean, text-based responses are used to enable checks within the application to occur as well. For example, when a user attempts to login, the responses available are ‘success’ or ‘fail’, and then an if statement after the response is received will save the username in local storage and then push the user to the homepage screen if it was successful. If the response is ‘fail’, then a toast, which is a floating pop-up box, will notify the user that the information provided is incorrect.

Finally, with any request to a server, security needs to be in an adequate place to ensure only authorised users have access. One method is to send an authorisation token, which the API can then look to match up to ensure that correct access is given. This wasn’t

implemented at this point due to other priorities in ensuring that the functionality was in place and that they performed consistently. If this application was to be pushed into a live server, these authorisation checks would be created.

2.6 Issues unaware of

2.6.1 Testing with the camera, local storage understanding and protocol

Dealing with Cordova native plugins can become more difficult to test and debug due to the fact that an actual device is needed before they are accessible. The first issue was ensuring that the camera would function properly and save the image correctly within the database upon success. Isolating this procedure was a strong step towards working out the individual moving components of the plugin, and then from this understanding, be confident in manipulating it to work with additional data as part of the collection request.

When developing a website, a common method to locally track and store data is through the use of cookies. This is helpful when it comes to implementing a persistent login system or being able to track what user is sending data. In the case of a cross-platform application, this idea is translated into using the devices local storage to hold this information. The plugin used for this was designed to setup an internal relational database which would save information in a key, value store format, the same as cookies. The plugin makes accessing this data relatively easy by simply assigning methods that would set, get or remove data from the local storage. Once more, the difficulty within this is that when testing a device, there would need to be checks and measures in place to ensure that this local storage would not be deleted. On top of this, an authorisation check had to be developed and properly tested that each time the

application was opened, would check to see if the user was currently logged in and set the root page accordingly.

2.6.2 Piping – image viewing security

Displaying images as a base64 format within a browser can become an issue as browsers will refuse the URL and deem it unsafe. This is due to the browser itself not knowing exactly where the URL has come from and therefore, blocks it. The way in which this issue had to be resolved was through the use of an angular library called "Dom Sanitizer" that was implemented within a pipe. A pipe is a type of provider which allows the calling of a method within a html tag (fig. 11 & 12). The library has a method which allows you to tell the browser that the attached URL, which in this case is a base64 image string, is safe and allowed to be shown. Although this could be directly coded into any page, the pipe ensures a global availability and can then be injected into any page as required.

```
import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizer, SafeUrl } from '@angular/platform-browser';

@Pipe({
  name: 'safeUrl',
})
export class ImageSanitizePipe implements PipeTransform {

  constructor(public sanitizer: DomSanitizer) { }

  transform(value: string): SafeUrl {
    return this.sanitizer.bypassSecurityTrustUrl(value);
  }
}
```

Figure 12: Example code snippet of the creation of a Pipe provider in a TypeScript

```
<ion-thumbnail item-start>
  <img [src]="cData.UnitImage | safeUrl" />
</ion-thumbnail>
```

Figure 11: Example use of the pipe method from fig 8 within the HTML file

2.6.3 Proposed Vs Revised Plan

Initially, a full implementation of an application without any core aesthetics was the proposed plan for the first iteration of the project. In part, this was accomplished when developing most of the core functions such as passing data between pages and API requests. However, many specific libraries or plugins used initially needed to be tried and tested within a clean environment to gain a full understanding of how it worked. One such example was when testing the camera plugin, where being able to completely isolate and send a base-64 encoded image to the database (without any navigation) or extra data worries ensured a strong ground up understanding of the plugin. These isolated plugin tests ended up taking over the majority of the time set aside for the first iteration, taking away a lot of time to integrate within the initial core first application file. As such, the completion of a full first iteration was cancelled due to the added time it would have taken to implement the same functions again. With the mobile-D methodology being used in the project, delays in the completion of certain tasks within the Gantt chart would need to be assessed and future tasks would need to be re-evaluated to accommodate for those delays. The Gantt chart in figure 13 illustrates a snapshot view of the progression that had occurred by weeks 7 and 8 compared to the original Gantt chart. Although the issues that arose caused a delay within the project progression, because of the knowledge of how they function at a greater depth understood, would allow for time to be saved during the second iteration of the application.

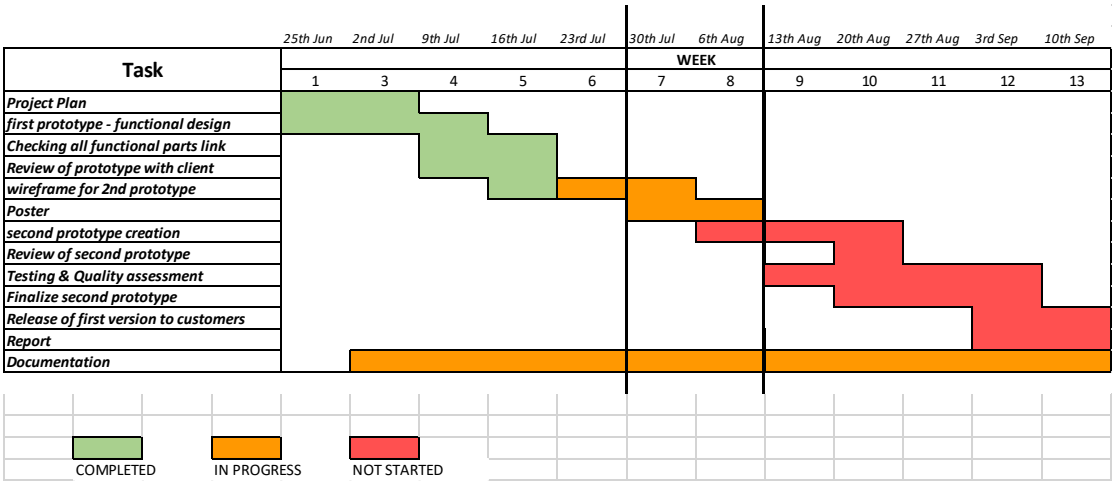


Figure 13: Gantt chart coloured to represent proposed progress Vs progress made by weeks 7 and 8

2.7 Second version development – live iteration

2.7.1 Wireframe redesign

A second iteration of the wireframe was created to ensure there would be a strong piece of reference material for both aesthetics and navigation when it would come to developing the second, live iteration of the mobile application, linking into the mobile-D agile methodology in place. The core navigation would remain the same as this was an aspect which Fletcher Moorland found effective for both implementing current functionality, as well as enabling future development to occur further down the life-cycle of the application. The core difference between the first and second iterations of the wireframe revolved around the aesthetics and trying to incorporate a user-centred design (fig. 15). A strong checklist to work through when considering the design of the application was through the use of Jakob Nielson's 10 usability Heuristics for User Interface design (Nielson, J, 1995). From the list, more concentration went into ensuring that the application was aesthetic and of minimal design, flexible and efficient to use, as well as making sure there was user freedom and consistency. From an aesthetic standing point, the minimal use of colour and effective box placement,

allowed for the application to look clean, easy on the eyes whilst still maintaining a high level of professional personalization to Fletcher Moorland. Secondly, flexibility and efficiency revolved around ensuring that time taken from opening the application to getting to the section required was as quick as possible. Currently, if a client wished to request a new collection, assuming they have logged in before, would take 9 taps of the screen (outside of any special requests typed by the client). It would then only take 3 extra taps from that completion screen to then view the specific order. Finally, the consistency came from having a clean theme throughout the application. The use of a persistent header title with navigation icons either side and a core body section with sufficient white space allowed for a user to quickly and confidently understand the functionality and, over time, ensure that the same familiarisation could lead to improved efficiency of use. Also, the header was designed to resemble the Fletcher Moorland logo itself, with the text matching the logo text colour and font, and then the left icon being orange and the right icon being green (fig. 14).

Figure 15: Comparison of homepages from first to second iterations of the application wireframe

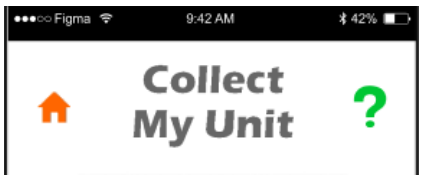


Figure 14: Screenshot of 2nd version application title design

2.7.2 Wireframe evaluation

Before heading into the development of the live application, a second meeting with Fletcher Moorland occurred which was designed with the intent to showcase the new and completed wireframe and obtain any further feedback on both functionality and design. During the first iteration, it was planned to attempt the implementation of a chat-based system. However, in ensuring that there would be sufficient time spent on some more vital aspects, this was tabled to one side for this iteration. This was addressed with Fletcher Moorland who agreed that developing and optimising the core use of the application would prove a lot more fruitful in both the short and long term. Aesthetically, there was no objections to the overall theme of the application, aside from potentially looking to use standard blue buttons (which are provided with Ionic) to help initially with maintaining consistency across operating systems. The requested changes involved what information would be shown within the orders page and their respective individual orders. Showcasing the image as a thumbnail in the main orders page meant that there was no need to clutter the image within the specific order page. Secondly, the need to display a due date was paramount as this would be the first piece of information a client would look for. Finally, a simple notes section was added into individual orders where Fletcher Moorland could send out text-based updates on the progress of the unit repair (fig. 16).

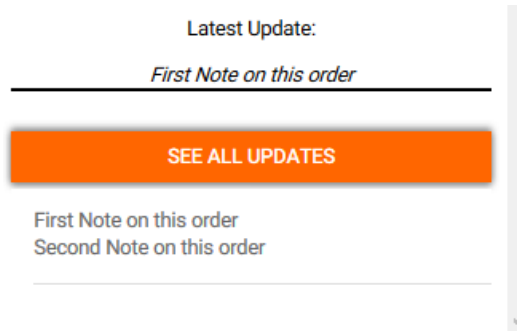


Figure 16: Screenshot of 2nd version application order notes display

2.7.3 Front-end development

Upon making adjustments to the wireframe post meeting with Fletcher Moorland, development could then begin on the live iteration of the application. Being able to reference the wireframe meant that the generation of all pages could happen quickly without needing to initially worry about the navigation between them. For the most part, the development itself involved taking the previous sets of code and re-writing them to fit into the application overall. From a developer's standpoint, it was also vital to maintain consistency with TypeScript layouts and variable referencing styles throughout. This was due to the fact that the development team at Fletcher Moorland would be taking over the application in the future and, therefore, the consistency allows for the code to document itself.

When using Ionic, the best practice with data manipulation is to implement a provider TypeScript file which can then be injected into the relevant pages. This would enable the reuse of a method such as one which would make a request to the API. When injecting the provider, the method within would be called and finally subscribed too to access the response from the API. However, early issues with being able to handle these effectively led to creating the POST request methods within their relevant pages. Converting these methods into a provider down the line would still be a good idea to optimize load times and allow for reusability of code with no need to duplicate per page.

2.7.4 Ionic components

The key to ensuring efficient cross-platform design is through using as many framework components as possible when relevant. This holds true in almost all cases as the components would be designed to adapt to individual operating systems on

deployment of the application. An example of this is when using buttons within the html file. Inserting a class of 'ion-button' within the button tag will transform the button to comply with certain styling rules which relate to their respective operating systems (fig. 17). Another more prevalent example is through the use of the plugin called 'FormBuilder' which is optimised for Ionic by Angular to generate a form block on a page, enable variable referencing and provide a great depth of methods to be applied. Within the application, this is first seen on the login screen, where the use of Regex is used to provide a set of validation rules. Initially, the submit button will be disabled until all validations have been met, and then the submit request can be made by the user. However, these validations themselves provide limited security as a knowledgeable user could view their application in a browser and be able to use developer tools to change the html. Therefore, strict validation would also need to occur within the API to ensure that issues like SQL injections (Janot, E. & Zavarsky, P, 2008) would not happen.

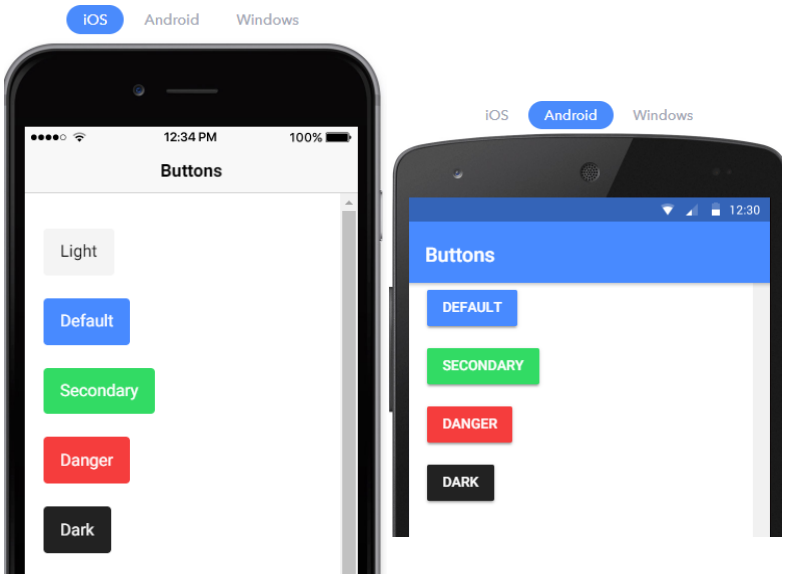


Figure 17: Comparison of ionic components based on operating system (Ionic team, 2018)

Another helpful Ionic component used throughout the application was the popover plugin. This plugin allows for a second page to appear in front of the current page, to provide the illusion that the user is still on the same page, but with a focused box now showing(fig. 18). This is used for providing additional information on pages where there is a question mark icon. This style of presenting information is recommended by Ionic as it is a great way to provide additional information without cluttering a page, and, presents a professional aesthetic to the overall design. Another style of pop-over used in the application is a 'toast' pop-over . These are effective at prompting a user when an error has occurred when making a request as they can be tied in with an http request method. The example in this case is that when an unsuccessful login attempt occurs, the toast will pop up in the middle of the page, prompting the user of the unsuccessful attempt before fading out again after a set time.

A feature which was not tested within the first iteration was with the use of Ionic loading bars plugin (fig. 18). These allow for not only a greater user experience, but also a way to minimize the issues that could occur from a user changing pages while data hasn't fully loaded on the current page. Like toast popovers, loading popovers can be implemented within an http request method. The strong feature of this is that the loader can be presented upon a request to the server or the pushing of a new page, meaning it will only be dismissed after a certain amount of time specified, or, when there has been a successful response from the server. A note here, however, is that this is based on obtaining any response from the API, regardless of whether the MySQL query was successfully run. Therefore, handling any errors would also need to be considered to avoid infinite loading screens.

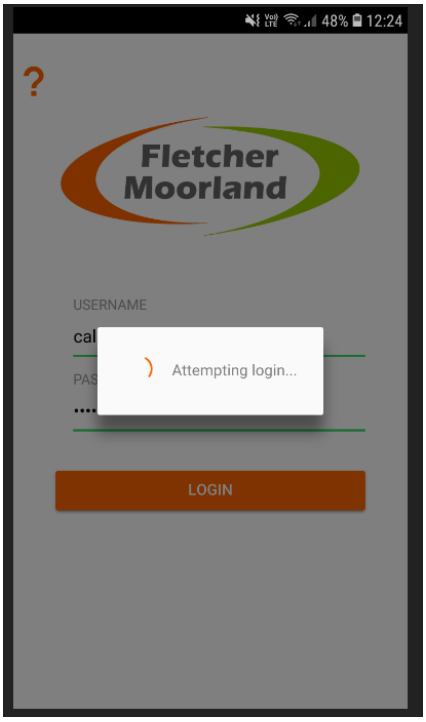


Figure 18: Example use of the loading component by Ionic within the application

2.7.5 Use of local storage

As discussed already, the way to track a logged-in user is through the use of the local storage plugin provided by Ionic. The first example of this use is when authenticating the login credentials provided by a user. The authentication itself will check that the username and password, which is hashed, exist within the database. If this is successful, not only will the homepage be pushed and set as the root page, but the username will be saved into local storage. Initially, the password was also saved within local storage, however this was considered bad practice and, hence, was removed. A further check occurs every time the application is opened up and this is implemented inside the 'app.component.ts' file which is the initial constructor before loading any pages within the application (fig. 19). An authentication check occurs here that obtains a reference to the key called 'username' and checks to see if it is empty or not and sets the root page back to login if it is not found. Although there are some security concerns in place

here, the principal is the same across most persistent logins where by it is the user's responsibility to ensure no other user accesses the device. If the user clicks the logout button on the home page, the locally set storage is removed and then upon reopening, will be shown the login screen once more.

```
//Current simple check for if there was a previous successful login - NEVER STORE THE PASSWORD LOCALLY HERE
checkAuth() {
  if (this.storage.ready) {
    this.storage.get('username')
      .then((data) => {
        if (data != null) {
          this.rootPage = HomePage;
        } else {
          this.rootPage = LoginPage;
        }
      });
  }
}
```

Figure 19: Code snippet of the authorisation check to allow for persistent login

Apart from authentication and persistent login checks, the local storage also holds another crucial role within the application. Firstly, relevant customer data is loaded into the local storage when the user goes onto the homepage. One piece of user data obtained is their customer ID from the database, which is then sent with any POST requests made by the user, which ensures that data responded by the server is only relevant to that customer.

2.7.6 Back-end and API development

Although the focus of an application can be the front-end, it is vital to develop a well-designed back-end system to allow for efficient use, as well as allowing for changes to occur in the future. The 'Data Structure' section goes into detail on the design of the database tables themselves and their interactions with each other. However, with the strict decoupling of frameworks there needs to be a well-developed middle layer which holds the API files, which can be accessed through their relevant URL links. These API

files sit inside the server, thus ensuring correct access is given would be paramount for when the application is pushed into the live server. With small differences between the individual API files, overall, they follow the same structure throughout. Firstly, the initial connection needs to occur with the database, which itself is a php file. By including this file within each API, it helps prevent any sort of duplication, and when there are database or server changes, they only need to be made within one file to affect all of the API's. When the connection has been established, the headers are designed to match against the request for security due to being sent via POST. The body of data sent by the request is then json decoded and assigned to a variable, and where needed, extra variables are assigned for specific key, value pairs. The relevant query is then made to the database if all the setup has been successful. Upon completion of the query, an if statement is carried out to form some responses based on the success of the query. This response is then assigned as a key, value pair and finally json encoded and sent as a completed response.

Due to the fact that front-end security is a visual deterrent, a couple of security features are in place within the API. Firstly, the header only accepts a body of data which has been sent through as json, and any other formatting results in a denial of acceptance from the server. Secondly, if the data is successfully obtained, it runs through the 'my_sqli_real_escape_string' method to counter against any attempts of SQL Injections. Finally, the use of prepared statements within the query mean that the API has full control over the acceptance and running of a query.

Upon migration of the database and API files, a few critical changes would occur to allow the API's to work again. Notably, within the live server there would now be two databases, the current database of Fletcher Moorland and the mobile application database respectively. When locally testing, all tables were located on just the mobile

application database, however, the Customers table that would be used on deployment resides inside Fletcher Moorland's existing database. As such, multiple database connections would be implemented, and the queries involving the communication to the Customers table would be altered to allow for this change. Figure 20 illustrates how the tables relating to the application interact with each other when split between two databases.

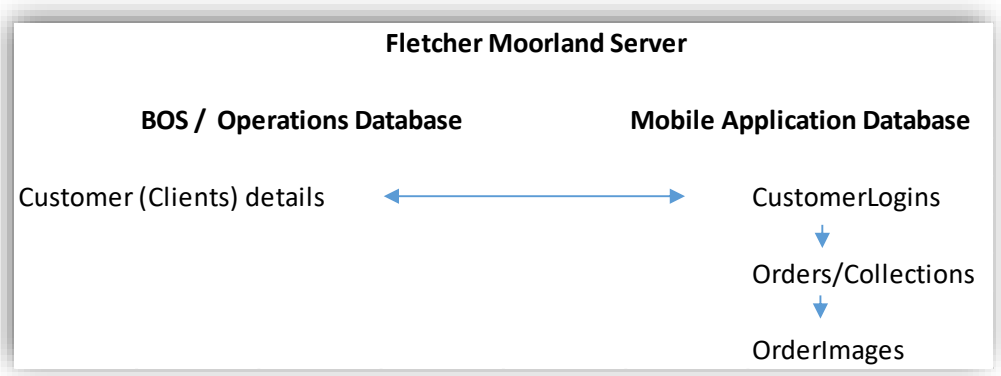


Figure 20: communication channel between tables & databases inside Fletcher Moorlands Server

2.7.7 Technical Documentation

The development of a technical document in practice should allow for a developer who hasn't worked on the project to be able to read through and learn the various technicalities contained. In the context of this project, the technical document would provide a way for the IT team at Fletcher Moorland to learn and understand the different environments and structures used during the application's development. It would also allow the team to have a well-structured manual too refer to when making further developments within the application in the future (appendix 4). Key areas which needed to be addressed within the document were the initial setting up of the development environment, notes on the framework used and its functions, design notes on the structure of certain pages or functions and finally how to carry out further development and deployment on the application. A key point considered when

designing the documentation was who the audience would be and why they would be using it. With the intended audience being the IT team, it was a further benefit to have discussions on the team's knowledge regarding the languages, frameworks and structures which were used within the project's development. The document was then tailored to provide more information on the Ionic framework, Angular and TypeScript Languages and less emphasise on the construction of the API files due to the teams stronger understanding of the standard web-based languages. This tailoring allows for the team to have a documentation that contains more information relevant to the on-going development of the mobile application.

2.8 Testing

An advantage to building cross-platform mobile applications is that live testing is very easy when testing in browser, and relatively easy when it comes to device testing. Therefore, many of the benefits when testing a website can be applied to this project as well, such as the use of a developer's console in a browser to monitor console logs, local storage, networks etc. This became an extremely powerful tool when testing POST requests made by the application to the API URL. The 'lab' command used in the CLI was helpful in being able to instantly view how the aesthetics may change through showing multiple platform emulators within the browser. However, this does not always represent how it may actually look on a device, especially when dealing with layouts of pages compared to the different screen sizes available.

An example of using the developer tools to solve an issue was when trying to figure out if the body of data sent in a request was in the correct format and contained the correct information. Because the API sends any response that is echoed (by echoing the

retrieved data and then adding an exit afterwards) any queries being run are stopped. The network tab in the developer's console shows all http requests made from the application, and by viewing the individual request made, will show the response which was being sent back. This became a powerful tool to test anything network related throughout the project. This was especially useful with testing the pre-flight issues that may have occurred and viewing the response and console description could determine the cause of a failed request.

The latter stages specifically involved testing on mobile devices and having a permanent local host setup up at Fletcher Moorland meant that the development team themselves could begin a more thorough test of the application and feedback on any issues found.

3 Results and evaluation

3.1 Evaluation of project

Reflecting on the success of a project of this nature can sometimes be difficult to quantify. A good place to start is to analyse the aims and objectives and review whether they have been achieved throughout. The aim looked to develop and release a mobile application for Fletcher Moorland which contained a set of minimum viable functionality. An overview of the application in action would suggest that this aim has been mostly achieved. The reason for stating this as not fully achieved is that significant testing would need to take place before the application is released onto the stores. Testing was incorporated within the scope of the project; however, a live environment test never took place which would need to be carried out first. This would entail both

in house testing, using a testing template to work from, to ensure any major bugs that are found can be resolved. From there, handing the application out to a sample group of clients to trial run would produce some clear feedback on what was working and not working well, as well as first hand recommendations from the client based on what they feel should be included or not included.

The objectives of the project provide a checklist, which form an illustration of the progress which was made, signifying key landmarks achieved. Objectives 2 and 3 were confidently achieved with the wireframe showcasing the full aesthetics and functionality of the application, and Ionic being an integral part of the mobile application development itself. Objective 1 became slightly different on completion as discussed in the 'Proposed Vs Revised Plan' section. This objective can still confidently be seen as being achieved as all functional aspects were tested and then fully implemented within the live iteration. Finally, objective 4 looked to produce a minimum viable product to release onto the store. As discussed, the application contained all aesthetics and functionality which was represented within the Fletcher Moorland agreed wireframe, but significant testing had not taken place meaning it would not be ready for a store release.

Another source to assess the progression of the project throughout was using the Gantt chart paired with the Mobile-D agile methodology. From the side list of tasks to be completed, only the release to clients was unable to be achieved. However, the methodology enabled the rapid introduction of the first version of the application, which paired with the tough test-driven design, ensured a quick understanding of the framework and ability to produce the functionality required. From this evaluation with Fletcher Moorland, the second version would theoretically function in the same way, with the hindsight of knowledge on the issues which were caused and resolved, a strong

foundation could be implemented which would allow for a well-developed minimum viable application to be released onto the store.

3.2 Questionnaire feedback – Fletcher Moorland

The questionnaire was designed to obtain un-biased qualitative feedback regarding the success of the project. The questions themselves would provide answers to the success of the individual objectives set at the start on the application and feedback on presentations that occurred throughout. An example of this can be highlighted with question 5 (appendix 3) which obtains feedback on functionality and, as a result, provides feedback indirectly to the success of objective 1.

Three questionnaires were sent internally to both IT members and the Technical director at Fletcher Moorland, from which one response was obtained. Statistically, the response rate for internal questionnaires is 30-40%, (Fryrear, A, 2015) which conforms to a response rate of 33% in this case. If the questionnaires were to be sent out to a larger internal group, it may benefit to provide a multiple-choice style answer sheet, ranging from 1 to 5, instead of written answers. This would allow for less time to complete, as well as producing more quantitative data as feedback. Also, this would allow the inclusion of more questions to target specific areas of the project such as the success of certain pages or functions.

Amongst the answers provided from the questionnaire, question 3 looked to obtain feedback on the overall completion of the project by asking for a response which would reflect on the project aim specifically. Point B in the response highlights the concern for the chat function that was removed from the initial product on release. Although this was a desire from Fletcher Moorland at the start of the project, during the first

iteration evaluation, the issue that was highlighted was that the time that it would take to implement against the time allocated would restrict a lot of further development in other, more important areas of the application. The decision to hold off on this chat functionality would allow for a structured minimal viable product. This would ensure that all key functions a customer would use, worked consistently and effectively. By having this structure, the development of further functionality would be implemented more effectively during later iterations of the application.

4 Conclusion

4.1 Future progress of the mobile application

Overall, the development of the application has been successful when compared to the Gantt chart progression. Objective 4 was partially accomplished due to the delays in completion of the first iteration, as well as the need for more vigorous testing that needed to occur before deployment to the market. The combined use a developed plan, agile methodology and constant feedback from Fletcher Moorland when using wireframes enabled objectives 1,2 and 3 to be achieved.

With the project life cycle completed, there are a few features that could be implemented within the mobile application in the short-term future. Firstly, using a software by Google called firebase, the application can be registered and from here, allow the development and control over deploying local notifications to the devices using the application. The main focus of these notifications would be to activate whenever a pending order has been accepted or when an order is ready to be sent back

to the customer. Over time, these could then be further customised based on user feedback.

A second piece of functionality that could be integrated is matching up their ‘Meerkat’ system to the application to allow for an all in one portal for the client’s needs. Meerkat is a project by Fletcher Moorland which allows for monitoring of the condition of certain machinery which have been attached with monitors. The application could have a page dedicated to show data and graphs relating to individual machines, as well as producing notification alerts when certain machine failures occur.

Finally, the introduction of the in-app live chat could be integrated once the application has been on the market for a while. There are a multitude of live chat plugins and libraries available and hooking these into the application could be a long-term development choice if the customers felt like it would be used.

With a well optimised minimum viable application deployed into the different markets, Fletcher Moorland would benefit from an application which serves as a highly useful product for their customers. Also, they could further branch out into a variety of new features based on customer feedback and company needs or desires.

Bibliography

Abrahamsson, P. Hanhineva, A. Hulkko, H. Ihme, T. Jaalinoja, J. Korkala, M. Koskela, J. Kyllonen, P. & Salo, O., 2004. *Mobile-D: An Agile Approach for Mobile Application Development.* [pdf] Available at: <https://arxiv.org/ftp/arxiv/papers/1709/1709.06820.pdf/> [Accessed 5 September 2018].

Andre Charland, B. L., 2011. *Mobile Application Development: Web vs. Native*. [online] Available at: < https://www.cs.ucsb.edu/~mturk/Courses/CS290I-2012/misc/WebVsNative.pdf/> [Accessed 7 September 2018].

Angular., 2018. *HttpClient documentation*. [online] Available at: <https://angular.io/guide/http/> [Accessed 9 September 2018].

Bloomberg, 2015. *7 Phases of a data life cycle.* [online] Available at: <https://www.bloomberg.com/professional/blog/7-phases-of-a-data-life-cycle/> [Accessed 11 September 2018].

Construction, B. D., 2016. *Top 55 Engineering Firms in Building Design + Construction.* [online] Available at: < https://www.bdcnetwork.com/top-55-engineering-firms/> [Accessed 5 September 2018].

Cordova, A., 2018. *Cordova*. [online] Available at: <https://cordova.apache.org/>[Accessed 9 September 2018].

Eisenman, B., 2015. *Learning React Native*. [e-book] O'Reilly Media, Inc. Available through: Google Books <https://books.google.co.uk/books?hl=en&lr=&id=274fCwAAQBAJ&oi=fnd&pg=PR2&dq=React+Native&ots=tFsmdIf7j_&sig=winBxVZKH1eS8QNRf0Msl-fVy84#v=onepage&q=React%20Native&f=false> [Accessed 9 September 2018].

Figma., 2018. *Figma wireframe software*. [online] Available on: <https://www.figma.com/> [Accessed 6 September 2018].

Fryrear, A, 2015. *What's a good survey response rate?* [online] <https://www.surveygizmo.com/resources/blog/survey-response-rates/> [Accessed 11 September 2018].

Hossain, M. & Hausenblas, M., 2018. *enable cross-origin resource sharing.* [online] Available on: <https://enable-cors.org/> [Accessed 5 September 2018].

Ionic team., 2018. *Ionic Framework*. [online] Available at: <https://ionicframework.com/> [Accessed 4 September 2018].

Ionic team., 2018. *Ionic CLI*. [online] Available at: <https://ionicframework.com/docs/cli/> [Accessed 4 September 2018].

Ionic team, 2018. *Ionic UI Component Documentation – buttons* [online] Available at: <https://ionicframework.com/docs/components/#buttons> [Accessed 10 September 2018].

Janot, E. & Zavarsky, P., 2008. *Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM.* [pdf] Available at: <https://pdfs.semanticscholar.org/0d1c/57269a6437caf883880cc5ba2e71101f8196.pdf> [Accessed 9 September 2018].

Klubnikin, A., 2017. *Cross-platform vs. Native Mobile App Development: Choosing the Right Dev Tools for Your App Project*. [online] Available at: <https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81> [Accessed 4 September 2018].

Krigsman, M., 2008. *Study: 68 percent of IT projects fail*. [online] Available at: <https://www.zdnet.com/article/study-68-percent-of-it-projects-fail-6103001175/> [Accessed 8 September 2018].

LePage, P., 2018. *Your First Progressive Web App.* [online] Available at: < https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/> [Accessed 9 September 2018]

Ltd, F. M., 2018. *Fletcher Moorland website.* [online] Available at: <https://fletchermoorland.co.uk/> [Accessed 4 September 2018].

McGivery, A., 2016. *Understanding Ionic 2: Navigation with NavController*. [online]

Available at: <http://mcgivery.com/understanding-ionic-2-navigation-navcontroller/> [Accessed 9 September 2018].

Nielson, J., 1995. *10 Usability Heuristics for User Interface Design.* [online] Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/> [Accessed 6 September 2018].

Perez, S., 2015. *Consumers Spend 85% Of Time on Smartphones in Apps, But Only 5 Apps See Heavy Use.* [online] Available at: <https://techcrunch.com/2015/06/22/consumers-spend-85-of-time-on-smartphones-in-apps-but-only-5-apps-see-heavy-use/?guccounter=1> [Accessed 5 September 2018].

Petzold, C., 2015. *Creating Mobile Apps with Xamarin. Forms.* [e-book] Microsoft press. Available through: <https://books.google.co.uk/books?id=mUMNCAAAQBAJ&printsec=frontcover#v=onepage&q&f=false> [Accessed 5 September 2018].

Puthiya, D, 2014. *Mobinius: Mobile Native Vs Cross Platform*. [online] Available at: <https://www.slideshare.net/damodarputtur/mobinius-native-vs-cross-platform> [Accessed 8 September 2018].

Reynosa, J., 2017. *Agile Methodology: The Complete Guide to Understanding Agile Testing.* [online] Available at: <https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing/> [Accessed 5 September 2018].

Roth, R., 2018. *Give Customers What They Need (Not What They Want).* [online] Available at: <https://www.forbes.com/sites/forbescoachescouncil/2018/05/03/give-customers-what-they-need-not-what-they-want/#7058c3c140ca> [Accessed 9 September 2018].

S.Balaji, D. M. S. M., 2012. *Waterfalls v-model vs agile: a comparative study on sdlc.* [e-journal] 2, pp.26-30. Available through: International Journal of Information Technology and Business Management <http://jitbm.com/Volume2No1/waterfall.pdf> [Accessed 1 September 2018].

Softaculous., 2018. *AMPPS local host server.* [online] Available at: <https://www.ampps.com/> [Accessed 5 September 2018].

Trivedi, R., 2018. *Top 5 hybrid Mobile App Frameworks in 2018 – Choose the best one for you.* [online] Available at: <https://www.websoptimization.com/blog/hybrid-mobile-app-frameworks/> [Accessed 5 September 2018].

Appendix 1: Risk evaluation table

Risk Description	***Probability of Occurrence***	***Prevention Errors***	***Remedy***
loss of data	Low	Keep multiple copies of all produced work.	Recover from the most recent version.
Shortage of time for complete project/ iterations	Low	A structured plan with weekly meetings to check on progress.	Complete key parts and remove non-essential features.
Application not working as requested on release	Medium	Testing of deployment within first iteration and throughout	Researching the issues in question through academic/ additional resources
Data being sent incorrectly from app when live	Medium	multiple stage testing with test and live company servers	Backing up database daily
Faulty communications to Database	Medium	Rigorous design of database structure and code which communicates	Working with IT team to resolve issues on server side. Review & fixing of front-end code
Framework updates	Low/Medium	Maintaining updated code, and using non-depreciated libraries	Agile Methodology ensures these best practices are up kept
Hardware failure	Low	Documentation of how-to setup the environment	following the documentation & picking up data from backed up area

Appendix 2: API example – user authentication

```php
<?php
include('serverConnect.php');

header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Headers: Content-Type");
//header('Access-Control-Allow-Headers: Authorization, Content-Type' );


$data = file_get_contents("php://input");

if(isset($data)){
    $request = json_decode($data);
    $username = $request->Username;
    $password = $request->Password;

    $username = mysqli_escape_string($conn, $username);
    $password = mysqli_escape_string($conn, $password);
    $passwordHash = hash('sha256', $password);
}

$query = "SELECT CustomerLoginID FROM CustomerLogin WHERE Username = '$username' AND Password = '$passwordHash'";
$result = mysqli_query($conn, $query);
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
$active = $row['active'];
$count = mysqli_num_rows($result);

if($count > 0) {
    $response = "success";
} else {
    $response = "fail";
}

$responseEncoded = json_encode($response);
echo $responseEncoded;
?>
```

Appendix 3: Questionnaire Reponses

Project Review Questionnaire

September 2018

1. **What are your thoughts on the meetings & presentations that have occurred during the development of the mobile application?**
   A. The presentations and meetings were well thought out and professional and any questions that we needed to ask were answered in an informative yet concise manner.

2. **What are your thoughts on the overall design of the mobile application?**
   A. The UI is very well designed with the end user in mind. Using the app is self-explanatory and flows very well. Furthermore, the colours and design scheme match the company's existing website and colour scheme.

3. **Did the application meet the expectations you had from the outset?**
   A. The finished design of the app was as we expected from the initial designs, and the functionality of each component works as necessary.
   B. We would have liked to have expanded further with a chat function for our clients to communicate with us about their orders however, in the meetings that we had with Callum we decided that this wasn't feasible in the timeframe that we had.

4. **Do you think the use of an interactive wireframe was a useful tool during meetings?**
   A. Yes, I feel that it helped to explain the functionality of the application components in a clear and explicit manner.

5. **What are your thoughts on the overall functionality of the mobile application?**
   A. Overall, we think the application works brilliantly and will help both our customers and our staff by not only making it easier for customers to book jobs in, but by making it quicker and easier for our staff which will inevitably, make aspects of our business better.

6. **Overall, do you feel the mobile application meets the needs of your clients?**
   A. Yes we do, as stated previously, it will make it easier for our customers' to arrange collections, book jobs in and view the status of their repairs.
   B. Additionally, it allows our customers to view 'live' data, which will save not only their time but our time progressing by phone or email their repair status.
   C. Yes, it provides our clients with a simple and easy way of notifying us about items they have for collection. This will make it easier for us to get product information, a picture of the unit and notifications that would otherwise need to be done via email/telephone.

Appendix 4: Technical Document

Initial setup of environment

Used: Ionic Framework / Git for windows (to allow npm installs) Ionic CLI (cmd console) /node.js / Android Studio (for use of android sdk) / Angular & TypeScript Languages used

Useful websites:

https://ionicframework.com/docs/ (contains all plugins etc)

Angular https://angular.io/

Steps:

1. Install Git for Windows https://git-scm.com/download/win
2. Install Node.JS to use its command console https://nodejs.org/en/
3. Install Cordova (software designed to access plugins/ apis to native devices)

   ```
   >npm install -g cordova
   ```

   (-g means a global install)
4. Install Ionic (Front-End framework)

   ```
   >npm install -g ionic
   ```
5. Environment should now be setup and ready for use.

Some Ionic commands (which need to be run within the applications folder) (tip: use -> cd) for folder path changes. See https://ionicframework.com/docs/cli/commands.html or use ionic --help in cli

Setting default root directory on Node.JS cmd (cli)

You can mean to change default directory for "Node.js command prompt", when you launch it. If so, then (Windows case)

1. go the directory where NodeJS was installed
2. find file nodevars.bat
3. open it with editor as administrator
4. change the default path in the row which looks like

```
if "%CD%\"=="%~dp0" cd /d "%HOMEDRIVE%%HOMEPATH%"
```

with your path. It could be for example

```
if "%CD%\"=="%~dp0" cd /d "c://MyDirectory/"
```

if you mean to change directory once when you launched "Node.js command prompt", then you can run in the command line of NodeJS command prompt:

```
cd c:/MyDirectory/
```

Change to fletcher moorland application from set directory example

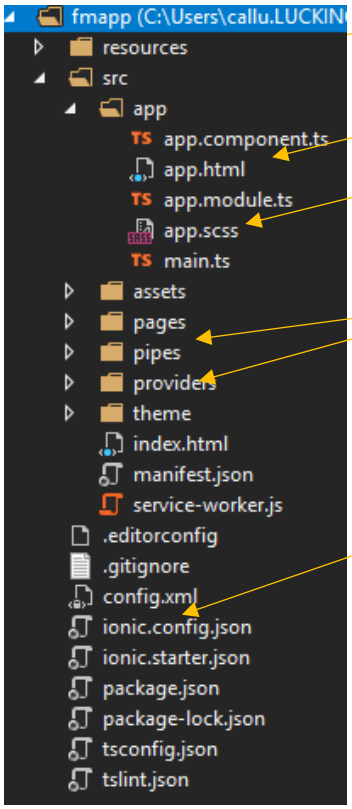
```
Node.js command prompt
Your environment has been set up for using Node.js 8.11.2 (x64) and npm.

c:\Users\callu.LUCKINGPC\Documents\FLETCHER MOORLAND WORK\PhoneGap Practice>cd fmapp

c:\Users\callu.LUCKINGPC\Documents\FLETCHER MOORLAND WORK\PhoneGap Practice\fmapp>
```

- New project (from root folder) -> ionic start blank “project name”
- New Page in app -> ionic g page “page name”
- New provider / pipe -> ionic g provider/pipe “provider/pipe name”
- Install ionic lab for testing -> npm install @ionic/lab
- Test app in browser -> ionic serve (--lab will bring up platforms within browser) (-c will produce more debugging in command console)
- Test on device (have debugging mode on phone, plugged in with usb
    - Check device is found -> adb devices
    - If found, run -> cordova run android
    - Go into chrome, find remote devices (in dev tools – more tools), find device, see active browser window -> inspect.

Ionic File / Framework Structure



Contains resources for android / ios – splash screens, logos etc

This ts file is where the app goes first to initialize (contains the checking of local storage for persistent login, splash screen

Where all imports (pages/ plugins / apis) have to be imported before they are able to be used throughout the application.

Where all the pages / providers / pipes are put when running ionic g page “name”. Will contain a subset of folders for each page. In which contain a html, css & typescript file for that specific page.

This contains all the information of the application (name of app/ version, preferences both global & platform specific)

The framework makes high use of DOM manipulation and use of Angular to pass data between the html & TS files of each

Breakdown of using the plugins / certain aspects of the application (pipes / localstorage / making calls to the api's / testing with the use of a mobile device

Example structure of a pages TypeScript File

```
import { Component } from '@angular/core';
import { NavController, NavParams, LoadingController, AlertController } from 'ionic-angular';
import { CollectMyUnitPage } from '../collect-my-unit/collect-my-unit';
import { Storage } from '@ionic/storage';
import { Http, Headers, Response, RequestOptions } from '@angular/http';
import 'rxjs/add/operator/map';
import { MyOrdersPage } from '../my-orders/my-orders';
import { AboutUsPage } from '../about-us/about-us';
import { LoginPage } from '../login/login';
import { CallNumber } from '@ionic-native/call-number';
import { CollectCompletePage } from '../collect-complete/collect-complete';
```

At top of the page – use these to import apis/ plugins or pages that you wish to use within this page

(e.g import storage allows the ability to use functions from ionic storage, import LoginPage allows to push or pull this page)

```
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {

  customerDataUrl: string = "http://localhost/fmapp/getCustomerData.php";
  //customerDataUrl: string = "http://192.168.1.133/fmapp/getCustomerData.php";

  customerInfo = [];
  userId: any;
  userUsername: any;
  userName: any;
  userSecurityLevel: any;
  userEmail: any;
  customerId: any;
```

Below the class construction (pre made with ionic g), you can create the variables to wish to use throughout the page. Note: the use of :any means that you do not need to specify the type.

A standard declaration would look as such =

"name": string; (if no data currently)

"name": string = "Sam";

"name" = []; empty array object

```
constructor(public navCtrl: NavController, public navParams: NavParams, public storage: Storage,

  this.loadCustomerData();

}

ionViewDidLoad() {

}
```

Constructor is where you will inject the relevant api/ plugin imports before being able to use them within functions.

Example: (public storage: Storage) makes access public, named “storage” and points to the Storage library

IonViewDidLoad(){} is a premade function where by any call of functions/ methods will be run once the page is fully loaded.

(If call to methods are used in the constructor method, these will be run upon the page being selected)

Button example:

```
<button ion-button id="aboutUs" full block (click)="aboutUs()">About Us</button>
```

In HTML, using (click)=”function” can then be referenced within the TS page upon click action

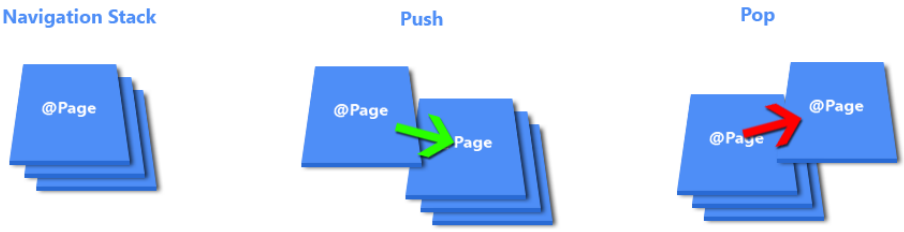
```
aboutUs() {
  this.navCtrl.push(AboutUsPage);
}
```

Example shows that when the aboutUs button is clicked, the NavController plugin pushs the About Us page (moves it onto the page).

Page Navigation

Uses Push/Pop stack style of page navigation:

Root page –> push -> Second Page (now sits ontop) -> Pop -> removes the second page from stack (effectively moving back to Root page in this case)



Breakdown of specific plugins used

Local Storage - https://ionicframework.com/docs/storage/

```
if (res == "success") {
  //console.log("Ionic success");
  this.storage.set("username", this.inputUsername);
  //this.storage.set("password", this.inputPassword);
  this.navCtrl.push(HomePage);
  this.navCtrl.setRoot(HomePage);
```

Use of “Set”, “Get” & “Remove”.

“This.storage” references this pages contructor of storage import. This allows access onto the methods above.

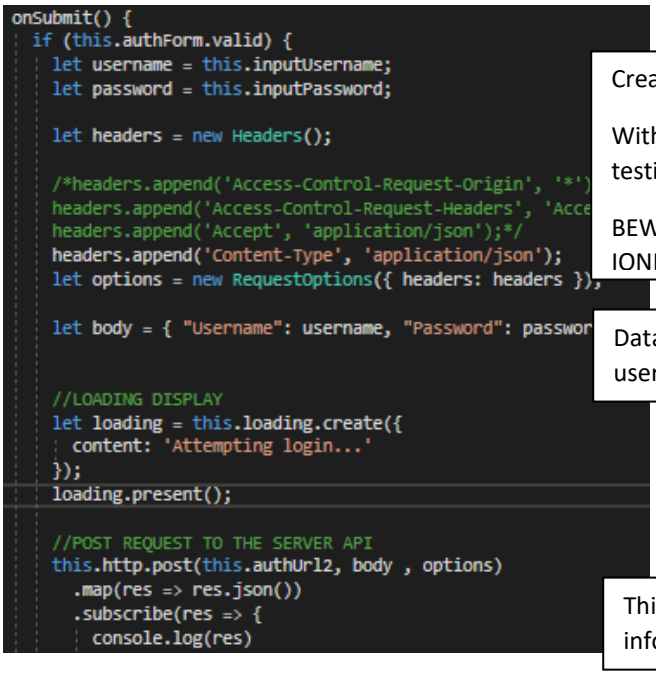
In example, this.storage.set(“username”, this.inputUsername) – assigns a key called username, and the value being the input which the user has input from the html input tab.

```
<ion-input formControlName="username" [(ngModel)]="inputUsername" type="text"></ion-input>
```

In the HTML, the use of [(ngModel)] allows the ability to then pass the string input by the user as a variable within the TypeScript file.

Sending API Requests – HTTP - https://angular.io/guide/http

Because of the need for all api calls to be assigned with the logged user, all requests are sent via POST method.



Creating the body & headers that need to be sent.

With Post requests, a pre-flight response needs to occur, effectively testing to see if the request matches the api headers.

BEWARE, THIS IS WHERE CORS ISSUES CAN OCCUR. PLEASE GOOGLE IONIC CORS ISSUES TO UNDERSTAND THIS MORE!!!!

Data has to be sent using JSON formatting. Therefore, the body assigns username & password as keys, with user inputs as their variables.

This is where the http import is used. Post requires a URL link, body of information & headers.

The post itself is called an Observable. Essentially, the post is sent and then receives a response. The .map converts the response to JSON, and then .subscribe allows you to then pull that "res" (data) and begin to proceed with dealing with it. In this case, just logging the response to check it is correct. Find out more on this with @ https://www.joshmorony.com/an-introduction-to-observables-for-ionic-2/

Ionic Components - https://ionicframework.com/docs/components/#overview

All the different api imports can be found in this documentation. These can allow things such as the pop-overs used for extra information, alerts/ toasts for incorrect login and loading icons to allow data to load.

API's are preloaded with an app, as such they just need to be imported on the page needed.

With Git APIs, you need to use the CLI to install from their npm. Example: Native Camera plugin:

```
$ ionic cordova plugin add cordova-plugin-camera

$ npm install --save @ionic-native/camera
```

Once installed through the cli, you need to add this to your application module TS file (found in the app folder, app.module.ts.

Import it at the top, then add it into the providers array:

```
providers: [
  StatusBar,
  SplashScreen,
  {provide: ErrorHandler, useClass: IonicErrorHandler},
  Toast,
  Camera,
  InAppBrowser,
  CallNumber,
  FileTransfer,
  File
]
```

You can now import it onto any page you want, and add it into the constructor before being able to access its methods.

FINAL NOTES ON ANGULAR

(click)=”” -> enables a method to be assigned to a click of a button

[(ngModel)]=”variable” -> allows to create a variable in the TypeScript file that would accept the user input

“let user of userData” -> UserData would be an array in the TypeScript file, just a standard data loop

{{user.name}} -> this would print the value of the key “name” within the user array (from above iterator)

[src]=”user.image | safeUrl” -> this uses a pipe. A pipe in this case is used to navigate around an image security issue for all images within the array. You can find this within the pipe folder – containing a pipe called safeUrl. This doesn’t need to be imported into the TS file.