UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Using .Net to Develop an Electronic Prescription System

Name: Ian Burke

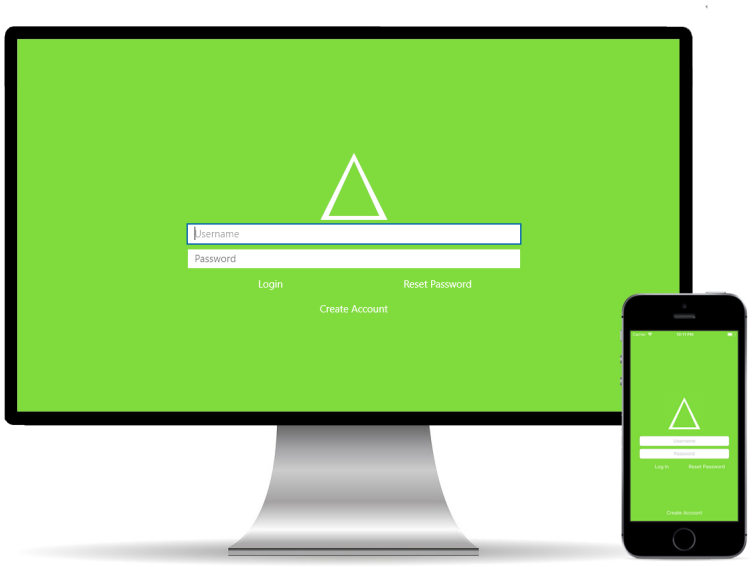
ID Number: 13122525

Supervisor: Reiner Dojen

Course: LM118 - B.Eng. in Electronic and Computer Engineering

Year: 2018

Department: Electronic and Computer Engineering



Using .NET to Develop an Electronic Prescription System

Ian Burke - 13122525 - 4th Year, B.Eng. in Electronic and Computer Engineering

Abstract

This project is designed to address and correct issues which exist in the healthcare industry today. The main aim is to make the prescription process easier for Doctors, Pharmacists and Patients by providing applications which handle the documentation and communication aspects of prescriptions electronically.

The applications are built using Microsoft's .NET framework, Oracle's Database Architecture and Xamarin's cross-platform development software. This allows applications to be deployed on platforms which target a wide range of users while sharing code between platforms. Users can expect similar designs across all supported hardware, which makes it easy to switch between desktop, mobile and tablet versions of the applications while keeping their data up-to-date.

Table Of Contents

Glossary of Terms

This section compiles all the abbreviations and unusual terms which are used in this project.

AOT	Ahead of Time (Compilation)
CLR	Common Language Runtime
eRx	Electronic Prescription
JIT	Just in Time (Compilation)
PCL	Portable Class Library
RDBMS	Relational Database Management System
UI	User Interface
UX	User Experience
UWP	Universal Windows Platform
VM	Virtual Machine
WYSIWYG	What you see is what you get
XAML	Extensible Application Markup Language

1 - Introduction

1.1 - Preface

One of the most frustrating experiences of my life was handing a prescription to a pharmacist who couldn't read it. The pharmacist rang the doctor for clarification, who was so busy that they couldn't come to the phone to answer the question. I had to go back to the doctor, get the prescription written again and return to the pharmacy…only to find that the second script was also illegible.

I didn't get my prescription that day.

This happened about two years ago. While annoying, I didn't spend a lot of time thinking about it after I (finally) got my medicine. However, the incident reoccured when I was considering the direction of my FYP - I figured there must be a way to modernise and improve the current prescription system that we have in place in Ireland today.

This project aims to do just that. Electronic Prescriptions (e℞) are available in many countries around the world, including Denmark, Finland, Sweden, England, and the United States [1], among others. While the European Union is on the path to having a cross-border e℞ healthcare system in Europe [2], Ireland is only now taking the 'first steps' towards getting an e℞ system up and running [3]. The aim of this project is to build an e℞ system based on feedback from pharmacies, doctors and patients. The final vision is to have a system which all parties can use to communicate with each other in the most efficient way possible.

1.2 - Aims and Objectives

At the end of the project, the objective is to have an application which runs on Windows,, Android and iOS. There will be three different user types: **Patients**, **Doctors** and **Pharmacies**. These user types will share features, but not all features will be available to all users. Patients will be able to log in and see information about their prescriptions such as the name of the drug, the recommended dosage, if a pickup is due, etc. Doctors will be able to view a full list of available medications, with information and interactions being shown on each drug. They will be able to prescribe drugs to patients using this screen. Doctors will also be able to see a history of patients they have treated in the past and what was prescribed to them. Pharmacies will also be able to view the drug list, but they won't be able to prescribe medications. Pharmacies also have the ability to receive prescriptions from patients and check out prescriptions when a customer arrives to pick it

up. This is a general overview of the application and the different views which are available.

1.3 - Justification

“Is there a demand for technology to help solve the problems in the health industry?”

There are numerous issues in the health industry right now - doctors are overworked due to the shortage of doctors in the country [4]. Pharmacies constantly need to contact doctors to make sure that prescriptions are correct. Patients have to visit the doctor, then take their script to the Pharmacy manually to get their drugs. Even then, they may be unsure about how to take their medicine and may be too embarrassed to contact their doctor about it. These are just a few of the issues that are being dealt with in the health industry right now and it’s clear that many of them could be solved with technology.

Over the summer of 2017, extensive research was carried out into how we might go about addressing some of these issues. Most of the research involved visiting doctors and pharmacies to get their input on what works and what does not work in the industry today. The insight and recommendations gained from the medical professionals surveyed were critical in creating the specification for the project. A common complaint which came up when features and solutions were described is that while said features exist in a program somewhere, there is no centralised application which has all the features they would like. The response to the proposal of an app which has all the features mentioned above was overwhelmingly positive.

1.4 - Solution

“How will this project address the issues mentioned above?”

At the end of the day, the application is designed to make everyone’s lives easier. Eliminating problems and replacing them with helpful features in the application will certainly contribute towards making the health industry easier to work in. Doctors will need to spend less time looking up drugs and writing out prescriptions - the system will be able to handle this for them. Pharmacies will be able to keep a central list of all medications sold and contact doctors directly should a problem occur. Patients won’t need to get a script written - the prescription can be sent electronically to their account, which can then be sent to the pharmacy for pickup.

1.5 - User Interviews

Once the project title was settled on, it was clear that extensive user interviews would be required. A number of doctors and pharmacies were visited over the summer to try and get an idea of what sort of features they want and to get an insight into the industry from the people working in it. For this report, doctors and pharmacies surveyed will be kept anonymous. A summary of the findings of these interviews are listed below.

Doctors

The following is a list of features which doctors are interested in:

- Streamlining and digitising the prescription process
- Making sure that prescriptions are being taken correctly
- Patient histories (this makes prescribing new medications easier and safer)
- Making sure that prescriptions have been picked up
- Comparing similar drugs side-by-side
- Being able to print prescriptions if the connection ever goes offline

Doctors are also very concerned about patient privacy - every doctor contacted specified that features such as patient histories and whether a prescription was collected would be great to have, but the decision on whether to share this information has to be up to the patient themselves (See **Section 2.5 - User Privacy and Patient Sharing** below).

Pharmacies

The following is a list of features which pharmacies are interested in:

- Better integration with doctors and patients
- Being able to see drug interactions when looking at a certain drug
- Patient histories (walk-in customers were described as ‘dangerous’ by one pharmacist)
- Being able to locate a pharmacy and have all their details available in-app
- Having the prescribing doctor attached to a prescription they issued
- Patients being able to send prescriptions to the pharmacy directly for pickup
- Physical backup process if the system ever goes offline

Most pharmacists surveyed made it clear that they have to take the patient at their word when it comes to their health - while not a problem for drugs which require prescriptions, over-the-counter medications can be problematic. While pharmacists are very interested in seeing patient histories, they were also very concerned about user privacy - every

pharmacist who I pitched the histories feature to made it clear that they think that customers should have the choice on whether to share their information or not.

These interviews were critical in creating the project specification - the feature list and overall look and feel of the application would be nowhere near as comprehensive without this input. The doctors and pharmacies were also very interested in seeing the final product and most asked that I contact them again once the project is complete to show the final result to them.

Patient users were also interviewed - however, these interviews were a lot more informal, as they mostly consisted of personal reflection on features that I would like to see as well as conversations with friends and family to find out what they would like to see in an application like this.

While these interviews with doctors, pharmacies and patients were not recorded (notes were taken throughout) a compilation of questions and answers is included in the appendices of this report.

Discussion and Justification

2.1 - Why Multiplatform?

During the research phase of this project, a lot of time was spent speaking to pharmacies and doctors, as well as examining their existing systems. Most doctors and pharmacies use software on Windows machines and many said that they have trouble adapting to new systems. This would suggest that Windows would be the ideal platform to develop on - after all, the healthcare professionals are used to doing business on this system.

However, discussions with patient users make this question a little more complicated. Patients are open to the idea of having a desktop application where they can view their data, but most requested a mobile app for their phones or tablets. This makes sense - Patients want to have their accounts available on the go and have the ability to visit multiple pharmacies, while doctors and pharmacies want a permanent machine available in their place of business. This presents a problem however - should we develop for desktop or mobile?

The answer is both - develop an application that works on desktop and mobile platforms, with platform-specific features. The apps should have identical feature sets, while also taking advantage of platform-specific advantages, such as larger screen real estate on desktop or NFC features on mobile. The platforms chosen for development are Microsoft's Windows 10, Apple's iOS and Google's Android (a version for MacOS was planned, but this build was dropped during development - see **Section 2.9 - What about MacOS?** below).

[5]

[6]


[7]

2.2 - .NET, C# and XAML

[9]

The .NET Framework is a 'free, cross-platform, open source developer platform for building many different types of applications'[8] which is developed by Microsoft. .NET Applications run in a software environment called the Common Language Runtime (CLR) which is a virtual machine (VM) that provides security, memory management and exception handling. .NET is focused on UI and web application development, which made it an ideal choice for creating an application like this. Xamarin is also built from the ground up to take advantage of .NET's features, which is another reason why the Framework was chosen (See **Section 2.3 - Why Xamarin?** below).

While .NET has support for multiple languages, including F# and Visual Basic, C# was chosen for project development. C# is one of the most powerful and versatile programming languages available in the industry today [10]. - it combines the simple grammar and syntax from C with more advanced networking features you would expect from a web development package. It was designed in conjunction with .NET, and first appeared in 2000. C# 7.2 is the most recent stable release at the time of writing [11]. While .NET allows interchangeability between all three of these languages in a solution, most of the project was written in C# and XAML.

.NET was also picked over other frameworks, such as Oracle Application Development Framework (ADF) or the Microsoft Foundation Class Library (FCL). This was mostly a personal decision - I have far more experience working with C# in a .NET environment than with Java in ADF or C++ in FCL. While Java in ATF is more portable and C++ in FCL is more optimisable, I picked .NET because it allows me to target the platforms the users specified in a manner that I'm familiar with.

XAML stands for Extensible Application Markup Language [12]. This is a language based on XML and used for initialising and laying out UI elements on-screen. XAML was used extensively in the Windows build for this project, both for prototyping and creating final views for the project. XAML could have been used on iOS and Android too but separate UI elements were used on these platforms (See **Sections 2.4** and **4.2** for more on this).

2.3 - Why Xamarin?

[13]

Deploying applications on multiple platforms was made possible with Xamarin. While writing native code for deployment on each platform was not out of the question, this would require extensive knowledge of each of the programming languages used to build native apps on each system. Essentially, the core .NET architecture would need to be designed separately, then implemented on all the platforms we want to target.

This is where Xamarin comes in. Xamarin uses C# to write native code on all platforms. This allows the core .NET functions to be called directly from the apps themselves, rather than being emulated or translated on a per-device basis. This also means that devices don't need to translate the source code from C# to Java or Swift on mobile platforms - Xamarin uses "Ahead of Time (AOT) compilation to produce an ARM binary that's compatible with Apple" devices and uses "Just in Time (JIT) compilation for Android devices"[14].

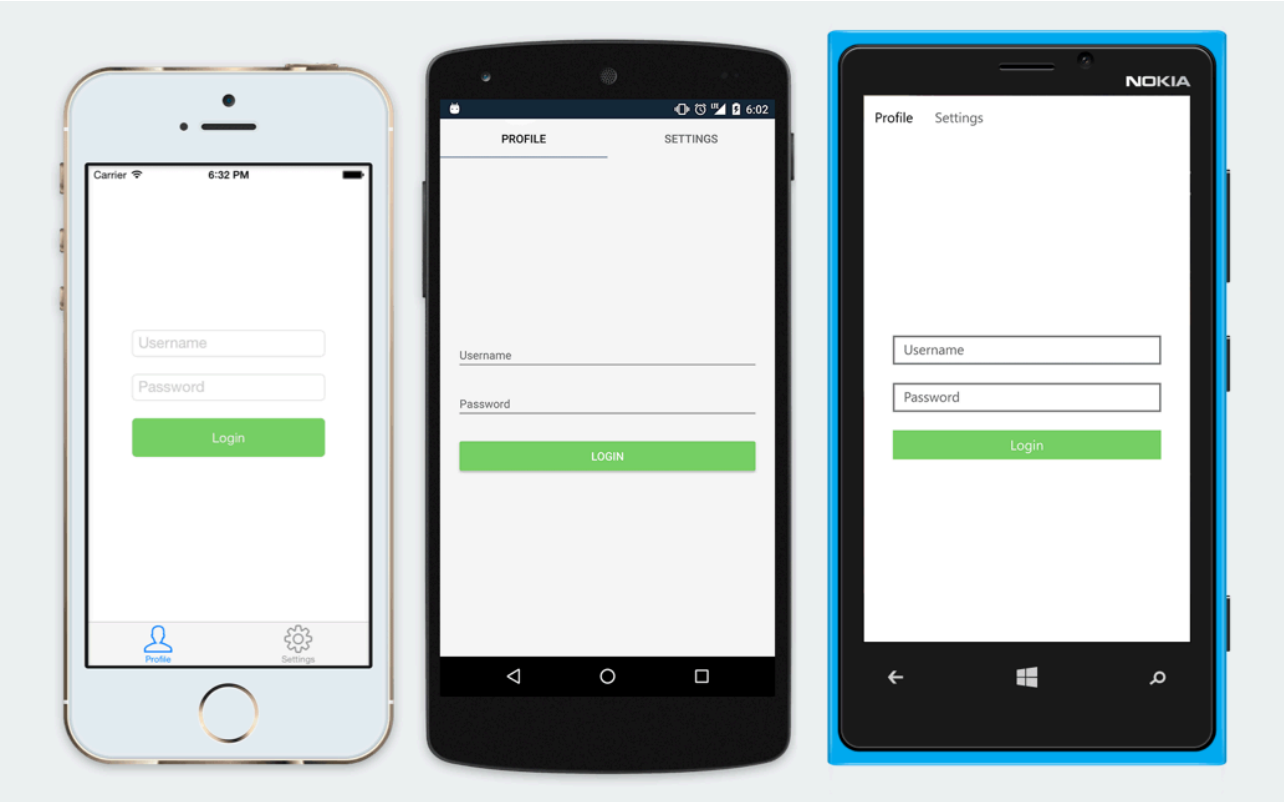
Xamarin also includes a feature called 'Portable Class Libraries' (PCLs) which 'generate managed assemblies which can be referenced by more than one .NET platform'[15]. PCLs are used extensively in this project because they allow us to write code once and deploy it to multiple platforms. This PCL code makes up the backbone of the application - it allows core database operations and functionality to be coded once then reused multiple times when porting applications to .NET platforms. PCLs have been used in the past to 'significantly reduce the number of projects required for multi-targeted applications while providing contracts for enterprise-level applications'[16].

Xamarin allows us to target every one of the platforms mentioned above, but developing on UWP has some added benefits - developing for Windows 10 will theoretically allow the app to run on Xbox One, Windows Phone and Hololens platforms. These platforms aren't directly being targeted for this project (and they haven't been tested), but it's interesting to note that the app could run on these additional platforms with very little additional development.

Other cross-platform development software solutions exist too - frameworks such as Qt and Ionic can develop native applications on multiple systems. One of the main reason sthat Xamarin was chosen over these is the programming language - Xamarin code is written in C#, a language I was already familiar with when I was selecting a framework. Qt uses C++ and Ionic uses JavaScript for development. Xamarin is also built from the ground up to interact with the .NET Framework, which is the other main reason why it was chosen.

2.4 - Xamarin Forms VS Xamarin Native

There are two main methods of developing mobile apps with Xamarin - Xamarin.Forms makes use of XAML and C# to develop a groundwork for the application, which is mapped to platform-specific UI elements at runtime. For example, in the image below a tabbed app is shown with two text entry bars and a login button. These apps are running natively on iOS, Android and Windows Phone respectively.
[17]



What makes Xamarin.Forms so powerful is the fact that this screen was written once in XAML - Xamarin takes the XAML code and compiles it to device-specific UI elements before deploying to the device.

This suggests that we should use Xamarin.Forms to build our applications. Surely that would make our job much easier?

Easier, yes. Better, no. Xamarin.Forms was **not** used in this project - the core .NET code was written to be portable, but the device-specific UI code was written natively for each device. To understand why this decision was made, look no further than Xamarin's recommendations for Xamarin.Forms and Xamarin Native[18]:

Xamarin.Forms is best for:
- Apps that require little platform-specific functionality
- Apps where code sharing is more important than custom UI
- Developers comfortable with XAML

Xamarin Native is best for:

- Apps with interactions that require native behaviour
- Apps that use many platform-specific APIs
- Apps where custom UI is more important than code sharing

Due to the fact that we want the apps to be as responsive as possible (which calls for native implementations), make use of platform-specific APIs (such as maps) and make sure that the UI works perfectly across all platforms, Xamarin Native was chosen for this project.

Another reason why Xamarin Native was chosen was the lack of customisation on a per-device basis - UI elements tend to use default UI settings when building with Xamarin.Forms. This is impractical for this app, because changes needed to be made to keep the visual elements familiar, yet in line with the platform the app is running on (See **Common Visual Language** below).

2.5 - User Privacy and Patient Sharing

Something which came up multiple times while speaking to the doctors and pharmacists was the issue of privacy. At the end of the day, no matter what the doctor and pharmacist recommend, it's up to the patient to take their medicine however they see fit. While this has a number of obvious problems, the system needs to account for this. The most important privacy feature is that the Patient has the **option** to allow doctors and pharmacies to see their history **only if they want them to** - The patient will be asked about this during account creation and the setting can be changed at any time. If the Patient turns all sharing off, the only information that the doctor or pharmacy will see is the information regarding the current prescription - all historical entries are hidden. If the account creation process is interrupted or otherwise cancelled, the options default to 'do not share'.

2.6 - Security

Security is a feature which needs to be brought up in anything which handles sensitive user data - especially so in this case, where the data is very identifiable and extremely sensitive. Security is one of the few features which was **not** implemented in the final application. There are a number of reasons for this.

Firstly, applications must be built from the ground up with security in mind. Every single facet of the app must be scrutinised and checked to make sure that there are no vulnerabilities which a malicious attacker could use to gain access to user data. Multiple different layers of security, including HTTPS, firewalls, DMZ and encryption must all be implemented [19] [20].

Secondly, while all this would technically be feasible, securing an application of this scope could be the foundation for a whole other FYP. There is simply too much work for one person to implement viable security while also building the application(s) from scratch.

For these reasons, a number of design decisions were made:

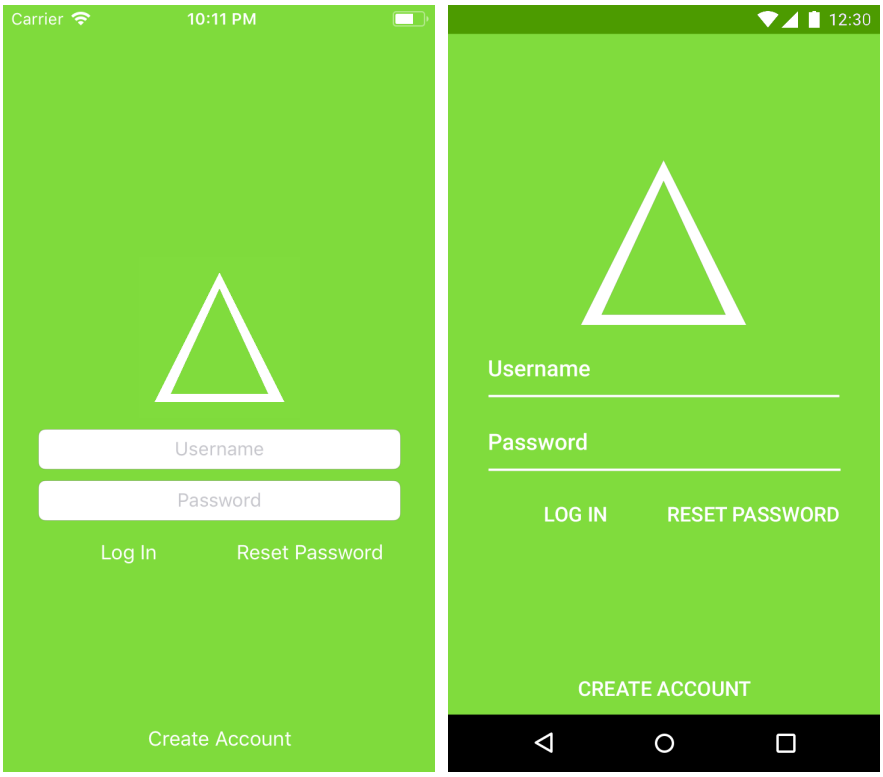
- No real user data was used - dummy names, addresses, phone numbers and e-mails will be used when setting up user accounts. This allows the system to be tested without using any sensitive personal data.
- Existing practice and Pharmacy management systems will not be integrated into the system - opening up these applications and connecting them to this unsecured application would give access to any malicious hacker who can access the FYP app.
- The drugs list and drug interactions information is readily available online - these features were implemented as originally intended, as there's no threat to anyone should they be compromised
- Every attempt was made to use secure connections and components wherever possible - however, if the security features were blocking development progress, the security features were skipped over to allow development to continue

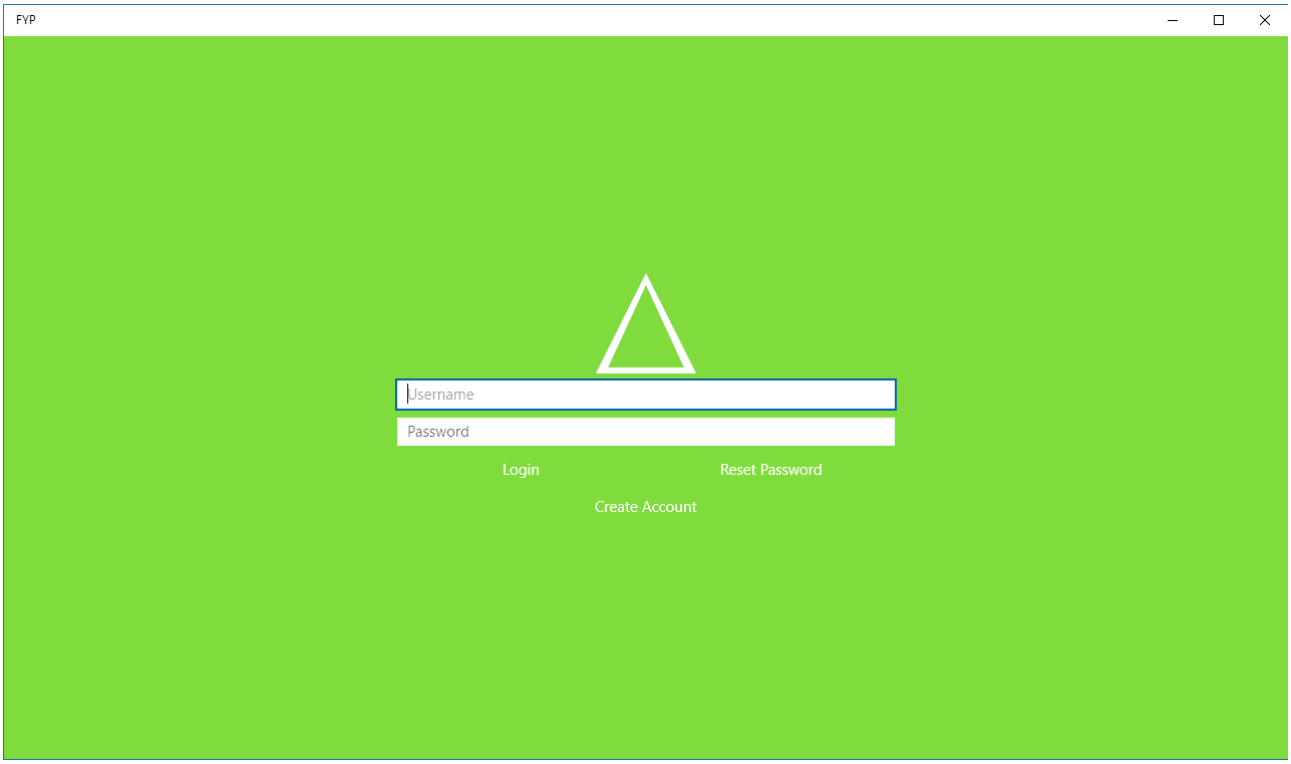
While not ideal, the design decisions mentioned above are necessary - getting bogged down in securing the application when there's critical coding work to be done will do nothing but hurt the final product. If the application was aiming for commercial release, security would absolutely need to be implemented to the highest possible standards (See **Section 5.5 - Future Development** below). However, for this release, security features will not be implemented.

2.7 - Common Visual Language

Something all developers should try to avoid while building an application is a ‘Porting Disaster’[21] - an app ported to a platform it was not initially designed for, which is inferior to the original version due to poor porting.

This presents an issue - how do you make your app familiar, accessible and responsive across all platforms, without sacrificing performance or confusing users due to the port?





Clockwise from top: iOS login screen, Android login screen and Windows login screen

Looking at the applications side by side, it's clear they share a common visual language while still appearing different. This was a conscious design decision - users expect slightly different controls on different devices, even if they perform the same functions. A good example of this is the login page, as shown above. The controls perform the same function regardless of platform - data entered in the text fields is checked against the database when the user hits the 'Log In' button. However, notice how the login button has a slightly different font and appearance on all three platforms - the fonts are 'Segoe UI' on Windows, 'Roboto' on Android and 'San Francisco UI' on iOS (versions of iOS prior to iOS 11 used Helvetica Neue, which is the font this report is shown in). Users expect these fonts to be used in navigation elements in-app - this is a way of subconsciously guiding users towards the navigation elements on the page. Even though the apps may differ on a per-device basis, users should feel comfortable using any version if they have previous experience with it. So for example, if a user of type Patient is used to using the application on their phone, they could install the Windows build on their laptop or PC and expect to know their way around the Windows version of the app.

2.8 - Why Oracle?

The Oracle relational database management system (RDBMS) was a personal choice - while on co-op, I spent a lot of hours working with test and production databases for a large international corporation which used Oracle systems. While SQL is (mostly) the same between all major RDBMSs, I grew used to the Oracle system and it's mannerisms. For this project, database schemas, tables and queries were designed using SQL Developer. These queries can then be triggered from the applications using .NET and the appropriate data can be returned and shown to the user (see **Section 4.3 - Database Architecture** below).

2.9 - What about MacOS?

During early planning and research phase, implementations were planned for Windows, iOS, Android and MacOS. The final deliverables include Windows, iOS and Java applications. So, what happened to the Mac build? Basically, it had to be cut due to time constraints. Issues with the Android build in particular caused early development on all platforms to take longer than anticipated. In the end, it was decided that the MacOS build should be shelved to make sure that the other versions are up to scratch in time. This additional time was put into additional testing and debugging on the other systems.

If the deadline was a month or two further out or if we had multiple developers working on the project, developing for a fourth target platform would probably have been viable to do within the deadline. However, once the project is complete (and fourth year exams are over) I fully intend to return to the MacOS build and finish it off over the summer.

2.9 - Currently Available Alternatives

Digitising the processes that doctors and pharmacies undergo each day is not a new idea - there are a number of commercial solutions available in the market today that are designed to help medical professionals do their jobs. This section will focus on what this software is and who makes it.

Some commercially available pharmacy management softwares include **QicScript Plus** from ClanWilliam Health, **McLernons Pharmacy Software** from McLernons and **TouchStore RX** from TouchStore.

[22] [23] [24]

These tools are designed with pharmacists in mind - they provide features such as scheduling appointments, printing receipts, stock control and drug interaction checks.

Clinic Management softwares include **Helix Practice Manager** from ClanWilliam, **Socrates** also from ClanWilliam and **CompleteGP** which is the name of the product and the company.

[25] [26] [27]

These tools are designed to help doctors run their practices - features they provide include prescribing and printing prescriptions, warning about drug interactions and maintaining patient records.

The main issue affecting each one of these software packages is that they're all highly specialised - they may perform one or more functions extremely well, but they're missing features which other packages provide. Pharmacies and doctors are often forced to use more than one of these solutions at the same time. This project is designed to focus on every step of the process from the patient visiting the doctor, to sending the prescription to the pharmacy, to the patient picking up the prescription. **Section 5.4 - Comparison to currently existing tools** has a more in-depth comparison between this project and these software packages.

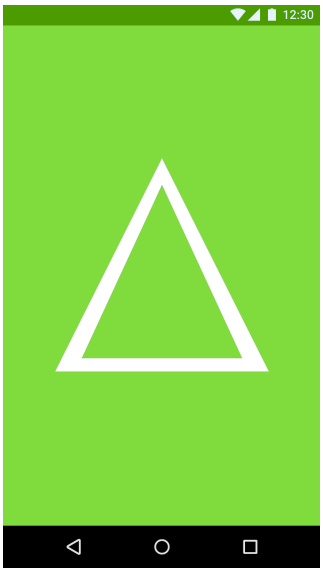
3 - Application Operation

The following pages run through the operation of the application and all it's various screens. For most of the screenshots shown in this document, a test user (with my name) is shown - this user has some settings and options set in advance to better detail how the app would operate for an end user.

3.1 - Splash Screen

When users boot up the application, the first thing they see is a splash screen with the application's logo. Depending on their hardware, this should disappear after a second or two but may take longer on older machines. This splash screen serves a number of purposes:

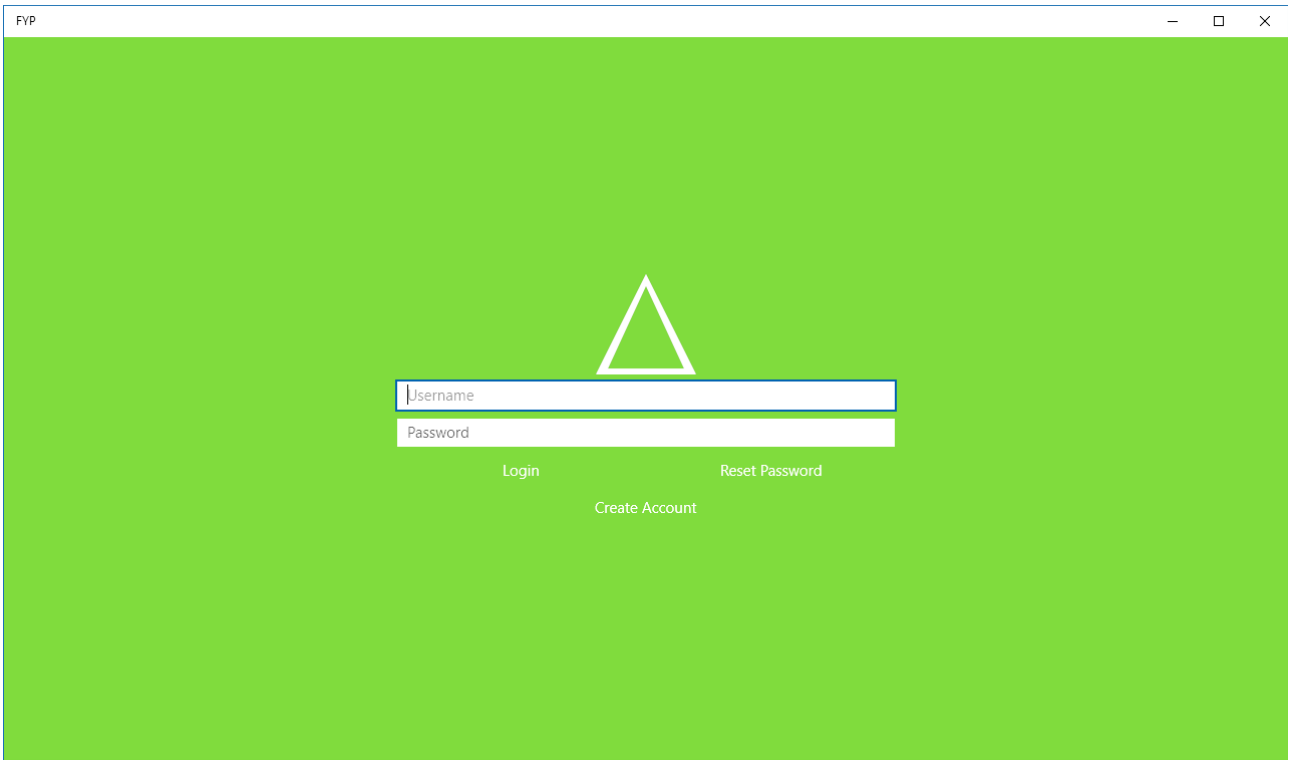
- **Hiding load times:** The splash screen masks the boot-up time for the application by displaying a still image on screen. If the load takes longer than three seconds, a spinner appears on screen to let the user know that the application is still working.
- **Startup checks:** The application performs a number of checks during startup - these include checking that a network connection is available, if this is the first time that the application has been launched and if the previous run ended in an error. Each of these cases will trigger a flag which affects the startup process - network connection or previous run errors will create an alert on screen, while the 'firstTime' flag will ask the user if they want to view a quick tutorial on using the application.



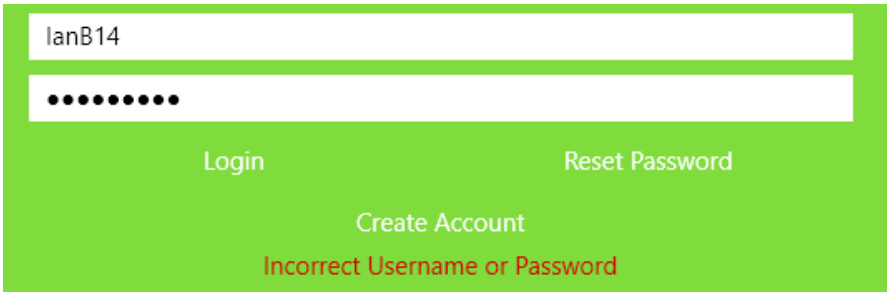
If the application boots successfully and no flags are triggered, then the user is presented with the login screen

3.2 - Login Screen

The login screen appears similarly on all devices - the user has access to two text fields to enter their username and password and buttons to attempt a login, to reset their password or to create a new account.

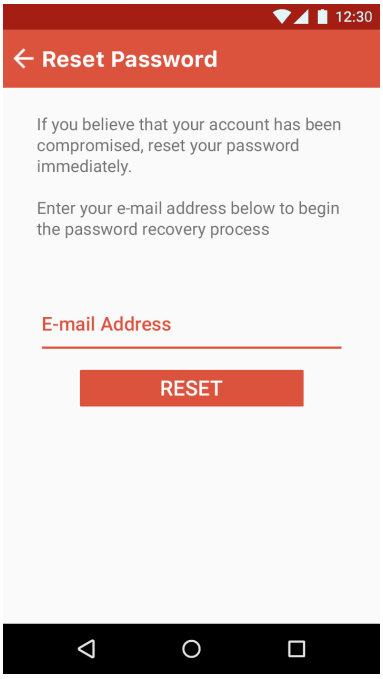


The text fields are slightly different to each other: the **username** field will display text as the user types, but the **password** field is protected - the letters appear as stars so they can't be read. The three buttons perform three different functions - the **Login** button will accept the text entered into the username and password fields and query the database to check if they match. On a successful login, the app will transition to the **Main Menu** - an unsuccessful login will display the following text onscreen:



The **Reset Password** button will navigate to the **Reset Password** Page and the **Create Account** button navigates to the **Create Account** page.

3.3 - Reset Password

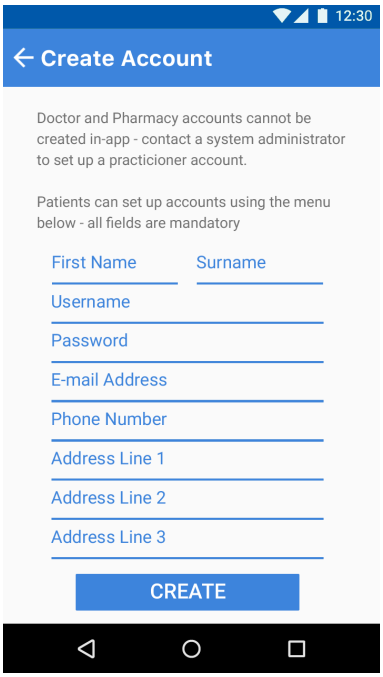


This is a simple page with only two controls and a text box - the text box warns users that they should reset their password if they expect that their account has been compromised and to enter their e-mail address to begin the password reset process.

The text field accepts an e-mail address, which it uses to query the database. Clicking the 'Send Password Reset E-mail' button will trigger this request.

The application queries the database to see if an account with the e-mail address attached exists - on a successful check, the application informs the user that an e-mail was sent to the specified address. On an unsuccessful query, the app informs users that the e-mail they entered does not exist in the system.

3.4 - Create Account



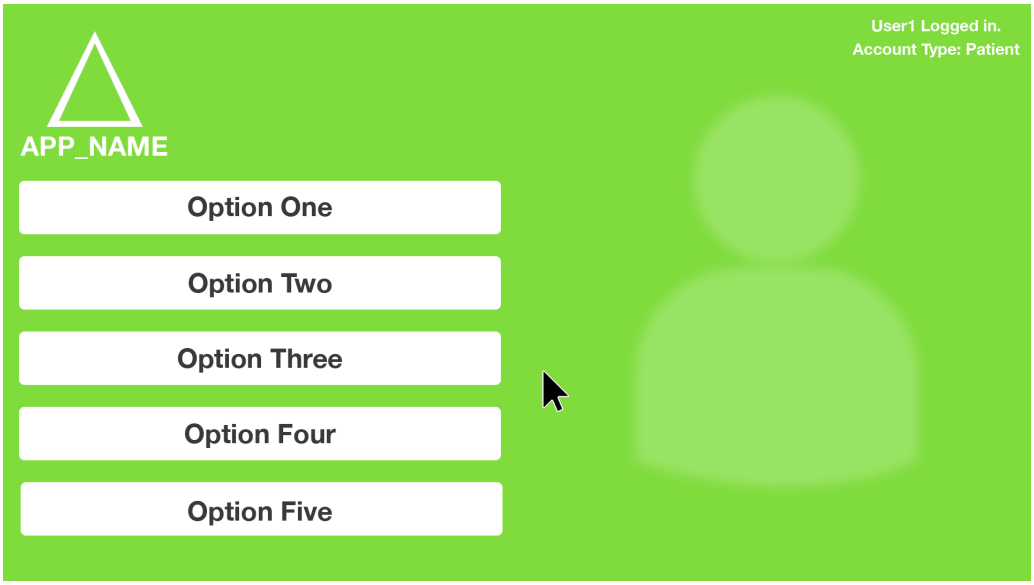
This screen has a lot of user data entry - it allows users to set up a new **Patient** account (note that doctor and pharmacy accounts need to be set up by an administrator - allowing any user to set themselves up as a doctor or pharmacy could cause major legal issues).

The text boxes all have placeholders to let the user know which details they need to enter - these placeholders disappear when a user selects a box and starts typing.

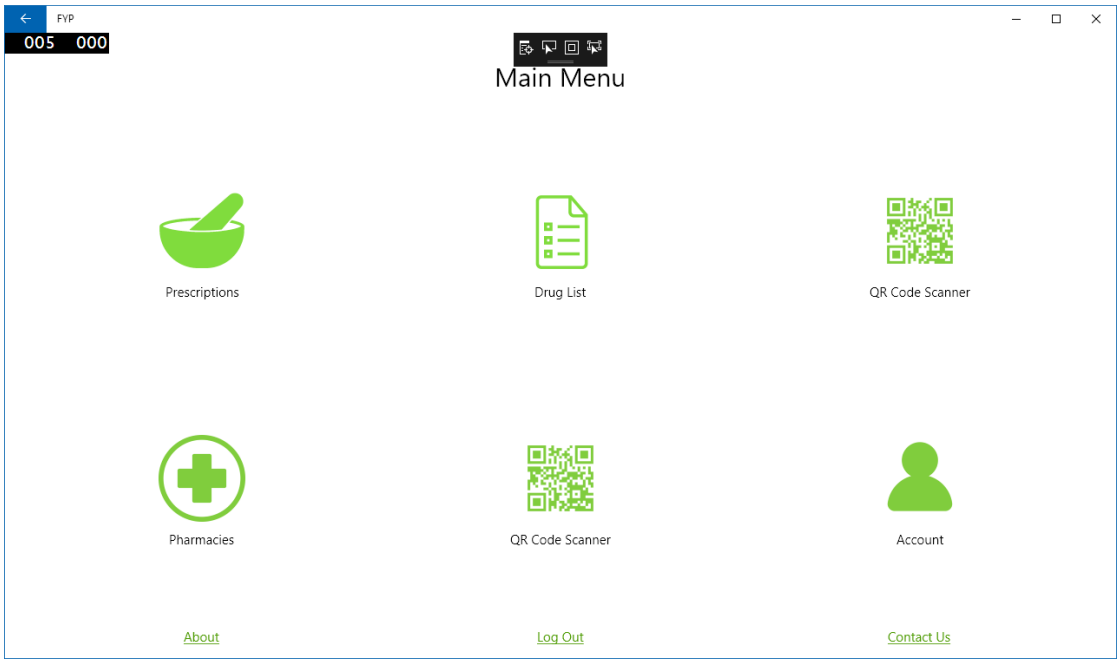
Once all fields have been filled in, the user can click the create account button (the button is greyed out and unavailable until text has been entered in all fields). This button triggers a database query to see if all the data is valid - if there are no conflicts, it creates an entry in the database for this user and generates a unique account number for them. On an unsuccessful request, the user is warned which error occurred (usually a missing field or data already exists in the database) and is prompted to try again. Otherwise, the screen turns from blue to green and a message appears to let the user know that their account was created successfully. They then have the option to return to the login screen to attempt login again.

3.5 - Main Menu

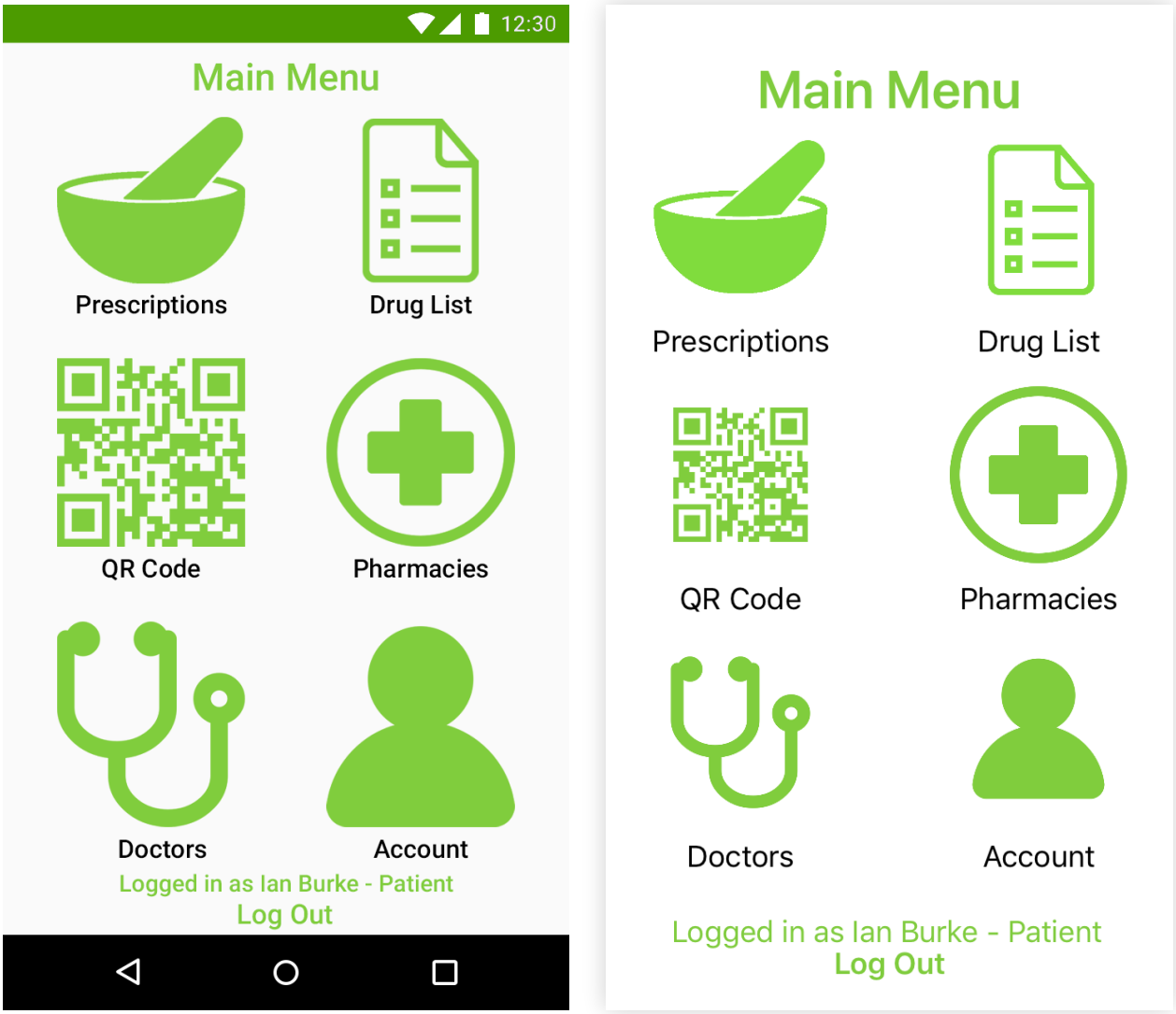
This screen went through a lot of iterations during development. The initial concept (shown below) was a very basic menu with static menu buttons. While this was viable on desktop, it didn't make clear to the user exactly what each button did, even when the button labels were changed to reflect the pages they transition to.



During mobile testing, these buttons also became difficult to press reliably on a small screen. For these reasons, it was decided to change to much larger buttons which have a large hit area and picture which represent what they will transition to:



This change makes it much clearer what the buttons will do and they are also much easier to hit on a mobile device, as shown on the following page.



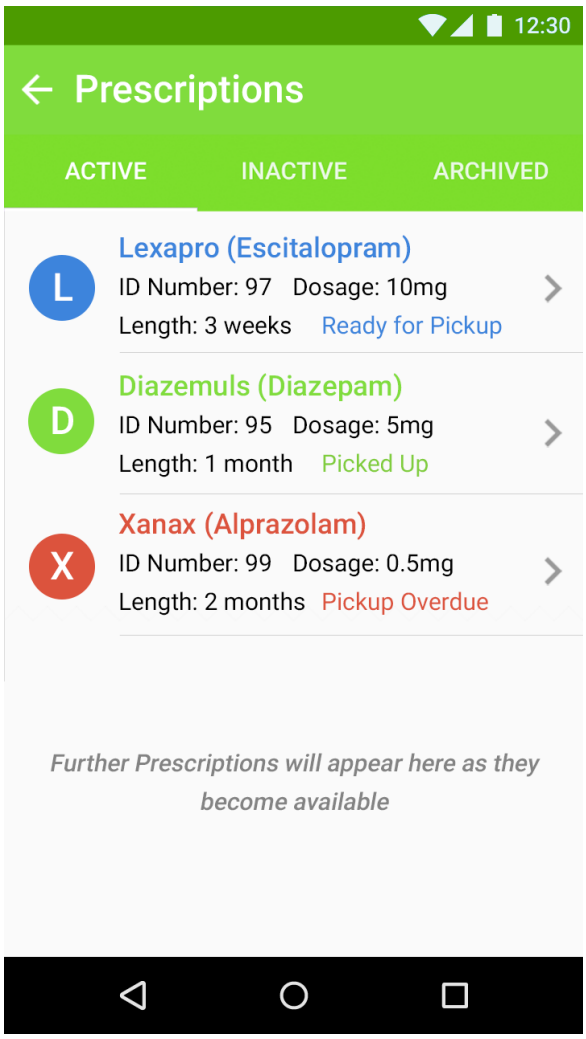
(*Note that in this image, the iOS version has a shadow around it to show where the edges of the screen are*)

Again, notice how the iOS and Android versions are similar, yet slightly different - the images need to be slightly closer together and the text slightly smaller on Android to make room for the software buttons along the bottom of the screen (this doesn't occur on devices which have hardware buttons as they have more screen space). The images and text labels align and resize themselves to the size of the screen they're on - on tablets, the images and text can be much bigger to take advantage of the larger screen real estate available to them.

The Windows version also has a couple of added properties that the mobile versions don't require - the window can be resized as the user sees fit and the icons/text dynamically resize and relocate themselves depending on the size and shape of the window to make sure that all elements are on screen at all times. Each of these buttons will perform a transition when clicked or touched. The screens which are available from this menu are described on the following pages.

3.6 - Prescriptions (Prescription List)

The first option in the menu is probably the one which would see the most use from a patient's perspective - the prescription list. From here, users can view the prescriptions that have assigned to their account (prescribed). These prescriptions are separated into three categories:



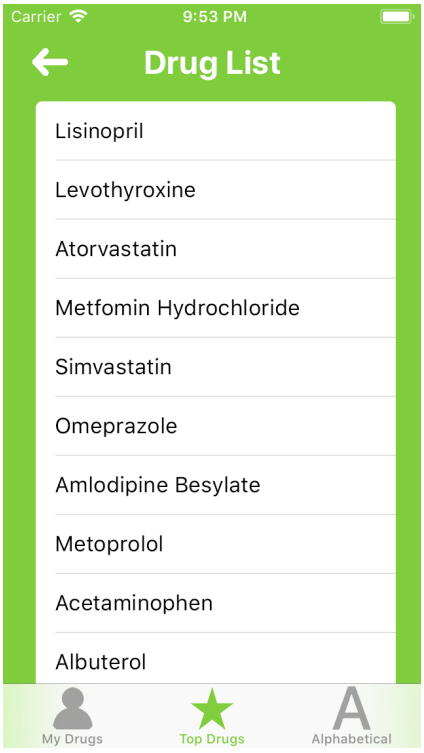
- **Active:** The patient is currently taking this drug, is due to begin taking this drug, or is just finished.
- **Inactive:** The patient has finished the course and has not renewed their prescription in over two weeks
- **Archived:** Prescriptions become archived after 3 months in the inactive category - they cannot be interacted with, but they're still available to view.

From these three categories, the most used is definitely the **active** category - this is where the patient can see what they're taking, how much their dosage is, how long the course is, etc. The prescriptions are also colour coded to show their current state:

- Blue prescriptions have just been prescribed by a doctor and are ready for pickup at a pharmacy
- Green prescriptions have been picked up successfully at the pharmacy
- Red prescriptions are overdue - the drug has been prescribed and sent to the pharmacy, but the user has not arrived to pick it up with a certain amount of time

These colours help the user to see the state of their medicines at a glance when they open this screen.

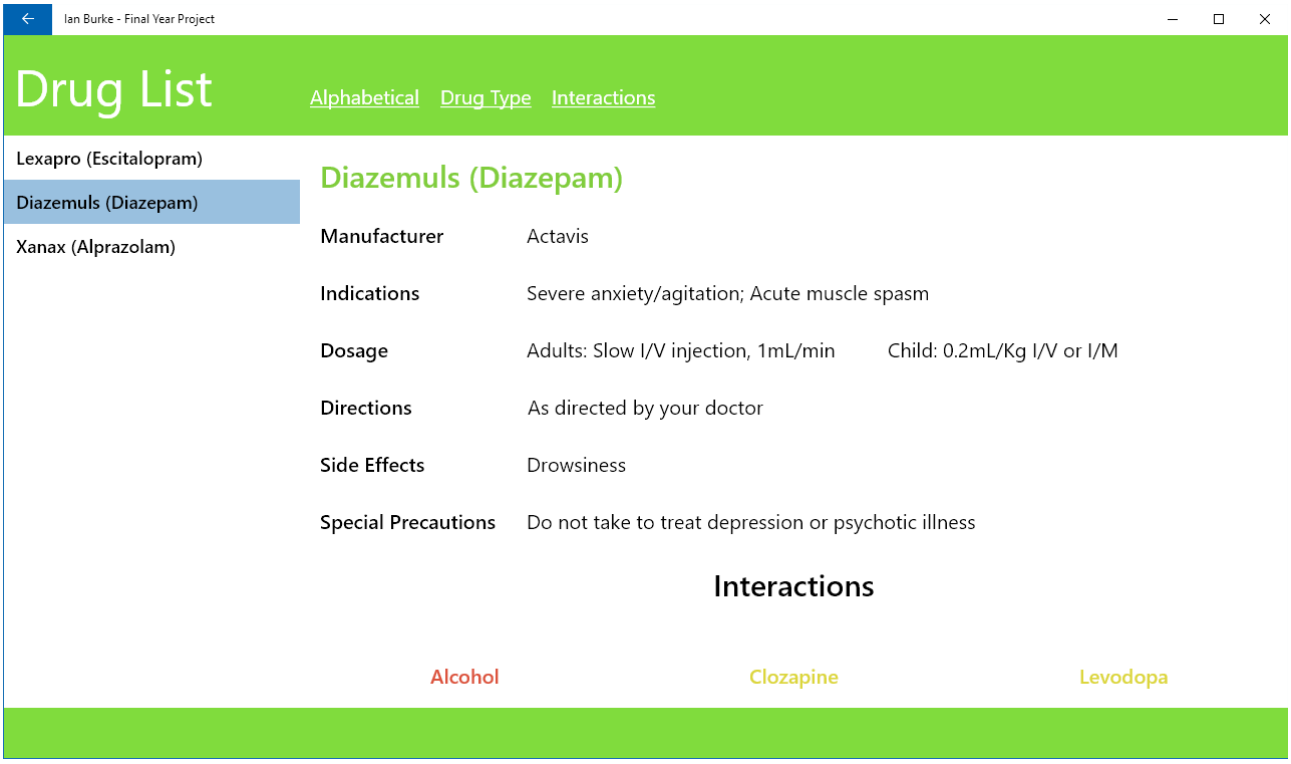
3.7 - Drug List



This screen is the second option available from the main menu - it shows a list of drugs, which can be selected to show additional information. The drugs can be organised in alphabetical order, in the order of most popular in the system, or it can detect which drugs that the user is currently using or previously used and show a list of them.

On mobile, each of these options can be tapped to show a detail view, which has further information on each drug. This information is stored in the database, which has a separate table for these drugs - information in this database includes the name, manufacturer, recommended dosages, indications, special precautions and any interactions with other drugs or substances. The desktop and some tablet versions take advantage of the increased screen real estate to display both the drugs and their information at the same time using a master-detail view - the screens still show the same information as the mobile versions.

For the test user with three drugs on their account, the details screen would look like this:



3.8 - QR Code

This is a simple screen which shows a text block and a QR code - this QR code is unique for each user and is automatically generated when they set up their account. The code contains the following details:

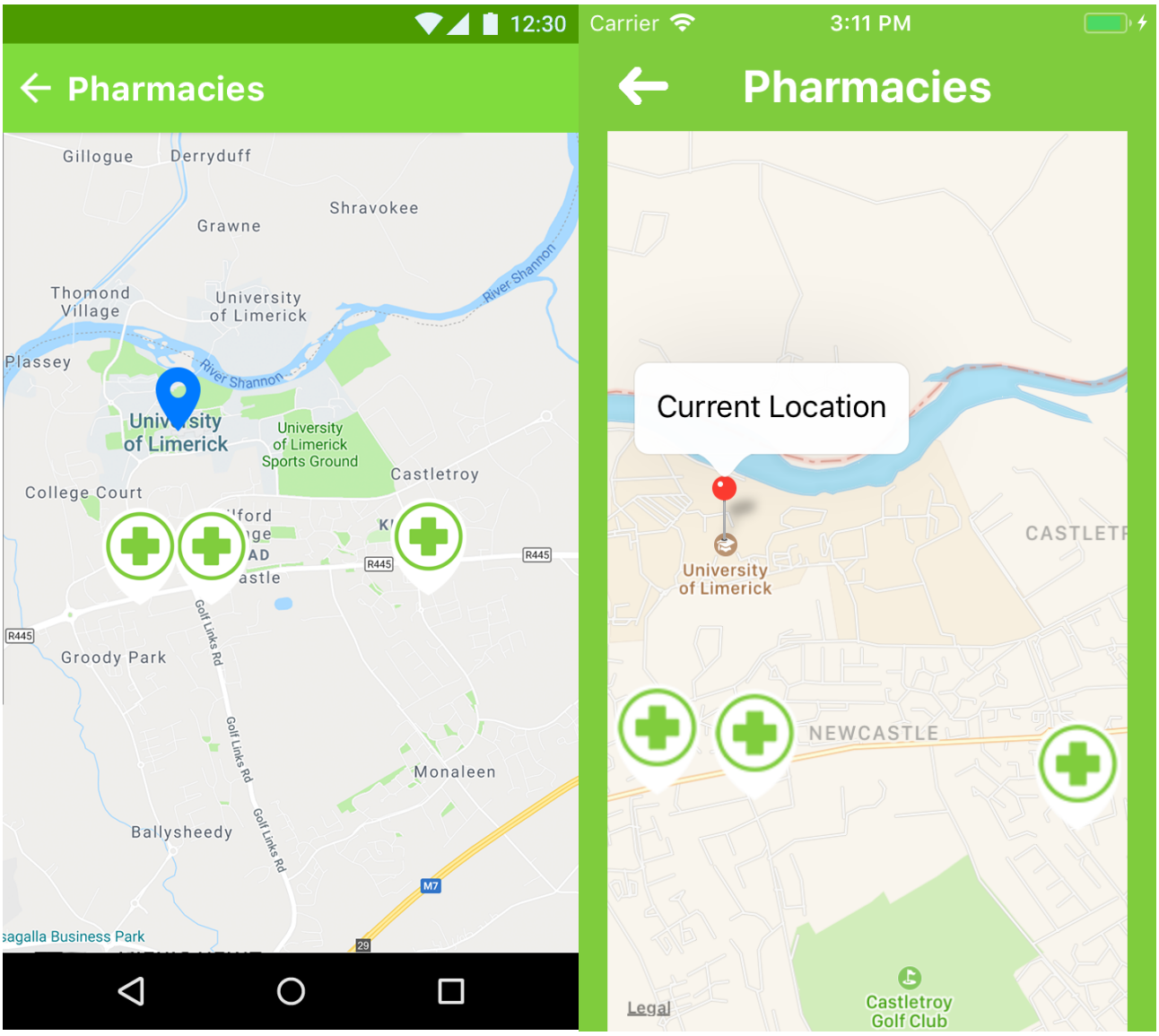
- Name
- Username
- Drug History

Right now, the QR code just contains these three details - in future, it could be extended to hold more information, but the idea of the QR code is to allow doctors or pharmacies to scan the code to gain access to a user's account quickly, without having to enter usernames, passwords, etc.

There are benefits and drawbacks to a system like this - the system greatly speeds up identification and pickup, but it relies on the user sharing their details. This can be controlled in the user's account, should they not want to share all of these details.

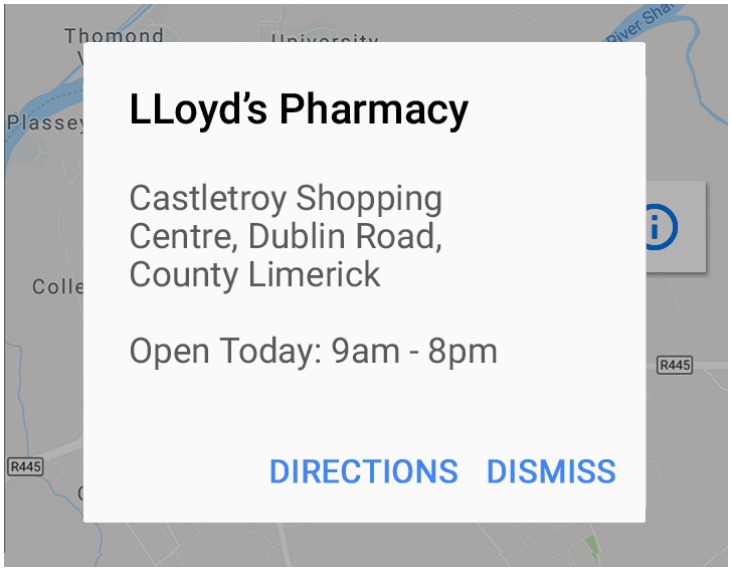
3.9 - Doctors and Pharmacies

These screens have similar functionalities - showing where doctors and pharmacies are located so the user can have access to them. The screens look very similar across all platforms, as shown below.



Minor differences in how the application looks on different platforms are down to the maps APIs - Android uses Google Maps, iOS uses Apple Maps and Windows uses Bing Maps. All three of these have their own quirks and idiosyncrasies, but are all similar enough so that a user comfortable with one version should be able to interact with the interface on a different system with very few problems. In the above screenshots, the cross icons on the map represent pharmacies and doctors are represented by a stethoscope icon (these mirror the icons used on the main menu). Tapping on one of these icons gives the user a popup which tells them the name of the pharmacy or doctor's practice, as shown to the right.

This popup has a further disclosure icon - tapping this will show an alert on screen:



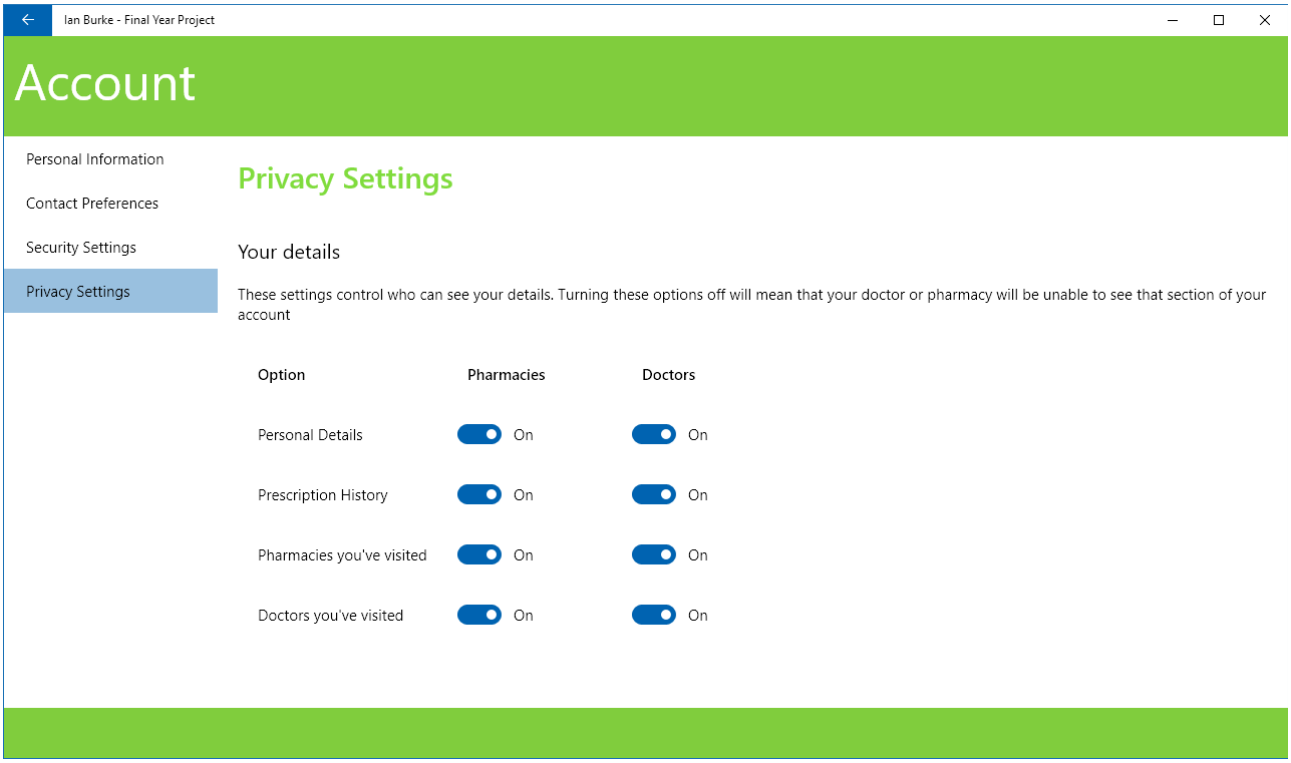
This alert shows the user the pharmacy or doctor's address, as well as their opening hours for today (if available). This alert can then either be dismissed or be used for directions - tapping the directions button opens up the current system's Maps application and passes it the stored co-ordinates for the doctor or pharmacy.

All of the information used to power these screens is again stored in the database - information includes latitude, longitude, street address, name, type (doctor/pharmacy), opening hours, etc. As mentioned above, each of the maps APIs has a slightly different way of handling all this data, but the builds are implemented in such a way that they should all work similarly enough that a user should feel comfortable using any version on any device.

An amusing bug occurred when implementing the application on Windows - the map is designed to automatically centre on the user's current location when the map object is being created. However, the code for this got mixed up with the code to set the map's view when it's displayed on screen - instead of centring the view on the user's location, it got confused and centred the view on the origin of the map…which happened to be in the Democratic Republic of the Congo. This caused some severe confusion during early testing, when all my pharmacies started appearing in the middle of Africa when the screen was launched, but this was easy to fix once I worked out what was going on.

3.10 - Account Screen

From here, the user can view and edit details relating to their account, security and privacy settings. The screen works similarly to the drugs list screen (see **Section 3.7 - Drug List** above) as both use a master-detail view to separate options into specific classes.



The screen shown above is the user's privacy settings - turning the toggle switches on and off will let the system know whether a doctor/pharmacy will be able to see that section of the patient's details when they access the patient's account. This screen in particular was added to help protect patient's privacy - something which both doctors and pharmacies emphasised in the interviews.

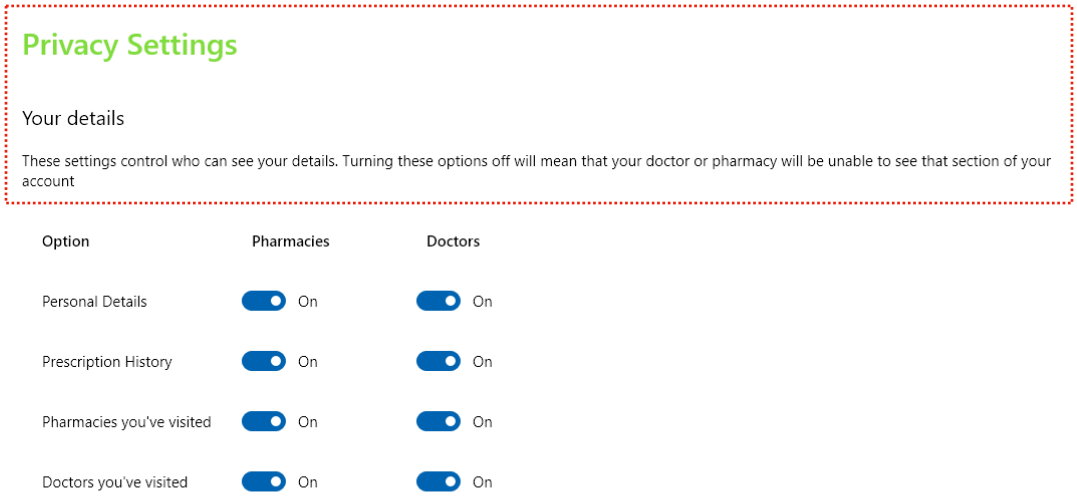
Initially, the master options on the left hand side of the screen were sorted into their own screens available to click on the main menu, rather than grouped together under the 'account' screen as they are in the final version (this screen initially just held the 'personal information' settings). However, test users had problems finding the exact option they were looking for and the main menu had an additional line of options available - this contributed to making the application difficult to navigate and difficult to use. Instead, these options were grouped together into the Account screen to make them easier to find - now all the account customisation is found here. The screen has kept its initial name for now, but it may be advisable to change it to 'Options' or simply 'Settings' to make it clearer what it contains. A beta test with end users could help to make a decision on this.

4 - Implementation

So now that we've seen the application in action, how does it work? In it's most basic terms, the applications use a XAML and C# front end to control the applications' look and feel, while a C#, .NET and SQL backend controls the data coming in and out. The following pages detail how the applications shown in Section 3 were created.

4.1 - XAML and C# (Windows)

XAML is used to create the layout for the application's pages - elements are placed on-screen and oriented relative to each other in a logical manner. The following is an example of how a specific part of the Privacy Settings page was created using XAML:



The XAML code below controls the text boxes highlighted by the dashed red line in the image above. Every element in the right hand side pane is embedded in a StackPanel control, each of which is given a name and attributes. Using these names, we can change how an element look at runtime, including their appearance and the values they display.

```
<StackPanel Grid.Column="1" Grid.Row="1" x:Name="privacySettingsPanel"
            Background="White" Visibility="Collapsed">

        <TextBlock x:Name="privacySettingsTitle"
                   Text="Privacy Settings"
                   Margin="10 10 10 10"
                   Padding="10"
                   FontSize="30"
                   FontWeight="SemiBold"
                   Foreground="#80dc3d"/>

        <TextBlock x:Name="privacyText"
                   Text="Your details"
                   FontSize="20"
                   Padding="10"
                   Margin="10 10 10 10"
                   />

        <TextBlock x:Name="privacyInformation"
                   Text="These settings control who can see your details."
                        + "Turning these options off will mean that your doctor"
                        + "or pharmacy will be unable to see that section of"
                        + "your account"
                   Margin="20 0 0 0"
                   TextWrapping="Wrap"
                   />
```

XAML makes it easy to control exactly how these elements will appear on-screen when the screen is created. While these controls are created using XAML, they don't actually do anything until they're hooked up to the code-behind. This backend code is what actually tells the application to take a certain action once the control is clicked. This is controlled by action listeners and events in the code-behind, which listens for actions taken in the GUI and performs appropriate actions when it detects these actions occurring. For example, the below screenshots show how a main menu button is connected to the code-behind with the 'Click' event. When a user clicks on that button, the 'Click_Prescriptions' event is triggered, which is picked up by the listener in the source code, which triggers a transition to the prescriptions page. Most events in the application are handled in this way - an event is triggered, which causes an action to occur based on it's context.

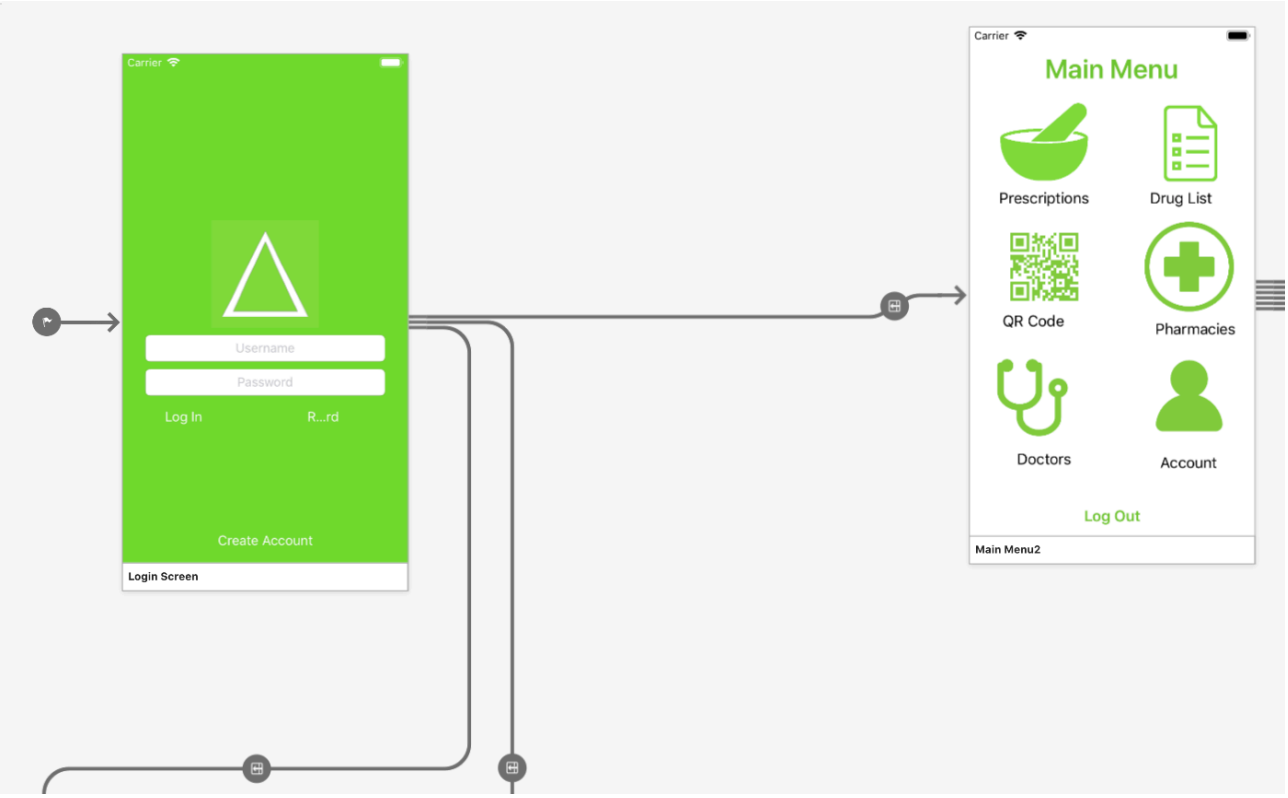
```
private void prescriptionButtonClick(object sender, RoutedEventArgs e)
{
    this.Frame.Navigate(typeof(prescriptionsPage), new DrillInNavigationTransitionInfo());

}
```

Prescriptions

4.2 - Android Activities and iOS Storyboards

On the UWP, it's very easy to lay out and hook up code elements using XAML and C# - however, because we decided **not** to use Xamarin.Forms early in development, this means that we **can't** use XAML to set up the applications for mobile platforms. Instead, we need to use Activities on Android and Storyboards on iOS to set up how the application will look.

Xamarin iOS Storyboards are based on Apple's Xcode Interface Builder - in fact, the Visual Studio interface for building iOS apps directly calls the Interface Builder when compiling storyboards. The following screenshot is an example of how the iOS application looks when it's being constructed in Visual Studio:

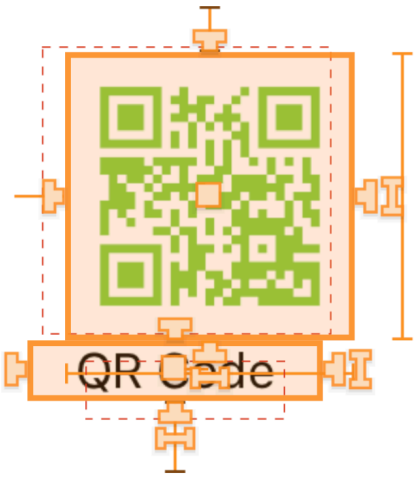


- The flag on the left hand side of the screen shows us the application's entry point - this is what the application will display once the splash screen finishes loading the app (Splash Screens on iOS are defined separately from the main storyboard). In this case, the application will always show the user the login screen when the app boots up.
- The lines connecting screens represent transitions - the arrow shows which screen initiates it, and the screen that the arrow points to is the screen where the transition will end. Another helpful feature of the Xamarin infrastructure is the ability to 'unwind' transitions - when a transition is triggered, it's stored in a stack. When an unwind action is triggered, the initial transition is reversed - it plays backwards as we return to an earlier screen. These unwind transitions are very easy to implement once the initial transition is set up: all we have to do is define a new action with the name

'UnwindTo(screenName)', then give it a function that accepts the initial transition as a parameter (as shown on the following page). Xamarin then handles the transition for us. These unwinds are helpful for many reasons - they cut down on the amount of explicit transitions we need to create (which are expensive) and they also help the user understand that they're moving backwards through the application

```
[Action("UnwindToLoginScreen:")]
public void UnwindToLoginScreen(UIStoryboardSegue segue)
{

}
```

- Note that some of the buttons and UI elements appear to be misplaced or misaligned in the above screenshot (for example, the QR code icon and it's text). This is because these elements are actually aligned programatically - custom constraints have been added to these elements to make sure that they align properly on multiple different screen sizes - in this case, the text and icon are designed to be distributed evenly between the screen border and the centre line. The image to the right shows how these constraints are applied in the storyboard - the orange handles show that constraints have been applied to the element and the dotted orange lines show where the images will appear on the selected test device (in this case, an iPhone SE).



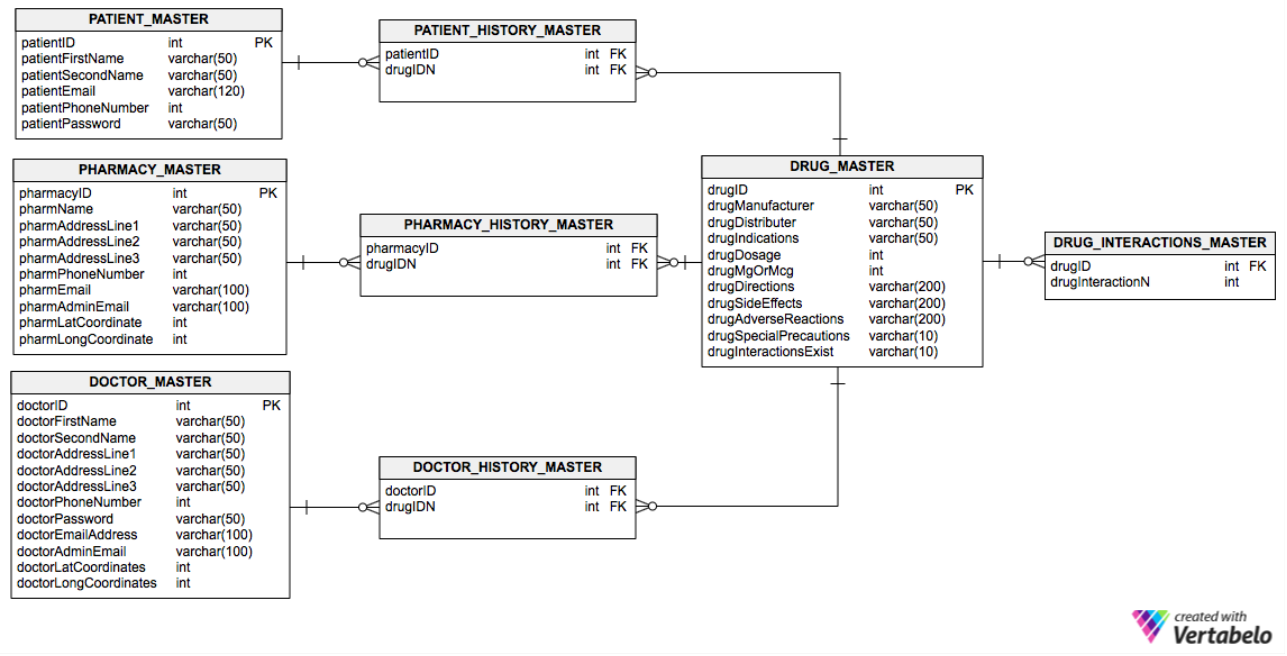
Using iOS storyboards in this manner allow us to get a WYSIWYG representation of what the application will look like at runtime, while also allowing us to tweak the UI elements to make sure they appear correctly.

Android Activities work in a similar manner - Android Activities files are separated into separate layout .AXML files (which control the layout) and code-behind C# which controls how the UI elements will function. There's no unified storyboard of activities on Android. Usually, Android activities are written in Java but Xamarin allows us to write them in C# - this makes it much easier to connect Android UI elements to the existing C# code base which performs application logic and database operations

The advantages of iOS Storyboards and Android Activities is that we can do out a rough layout of our UI elements, then hook them up to the code and set them up exactly how we want - this is similar to the way we use XAML and C# on the UWP to create the application

4.3 - Database Architecture

The project uses an Oracle database to store data. Oracle Express Edition 11g is is provided free of charge to 'develop, deploy and distribute' [28]. The following image shows how the database is organised:



The PATIENT_MASTER, PHARMACY_MASTER and DOCTOR_MASTER tables store data relevant to the user type named in the title. Each of these tables are related to the relevant HISTORY_MASTER table - these history tables are populated as users start to use the app and prescribe/dispense/receive the medications. Note that these history tables are based on a user's ID, which is used as a Foreign Key in each of these history tables. This allows each user to have their own history which is permanently linked to their ID. These history lists are populated using the drug tables below.

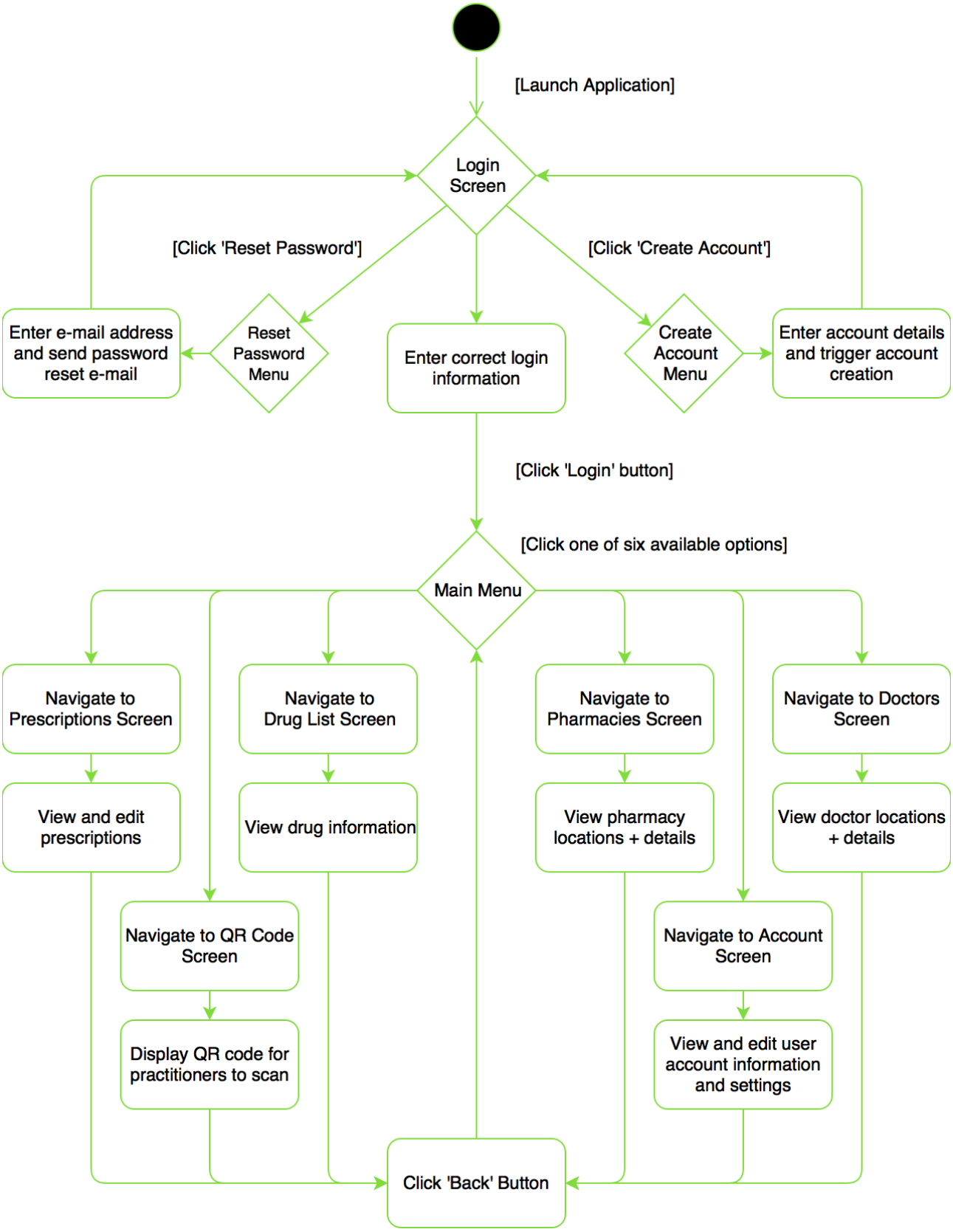
The DRUG_MASTER table stores information relevant to medications - this is different from the user tables, as it has to store different types of data. Note the 'drugInteractionsExist' - this is a column which was supposed be used to query the DRUG_INTERACTIONS_MASTER table should drug interactions exist. However, this feature was removed from this build of the application due to time constraints (see **Section 5.2 - Cut Features** below).

Prepared statements are used to query the database from the applications - for example, one of the most common queries is the login process. The user enters their username and password on the login screen and hits the 'log in' button. This triggers a function in the code to send the following query to the database: "SELECT * FROM PATIENT_MASTER WHERE patientUsername = <text entered in the username field>". This lets us check if the entry exists - if it does, we can grab the corresponding password text for this account and log the user in if it matches. Obviously, sending and receiving plaintext passwords

would be a disaster in an application that deals with user's personal and medical details but this project isn't designed with security in mind for this release (see **Section 2.6 - Security** above).

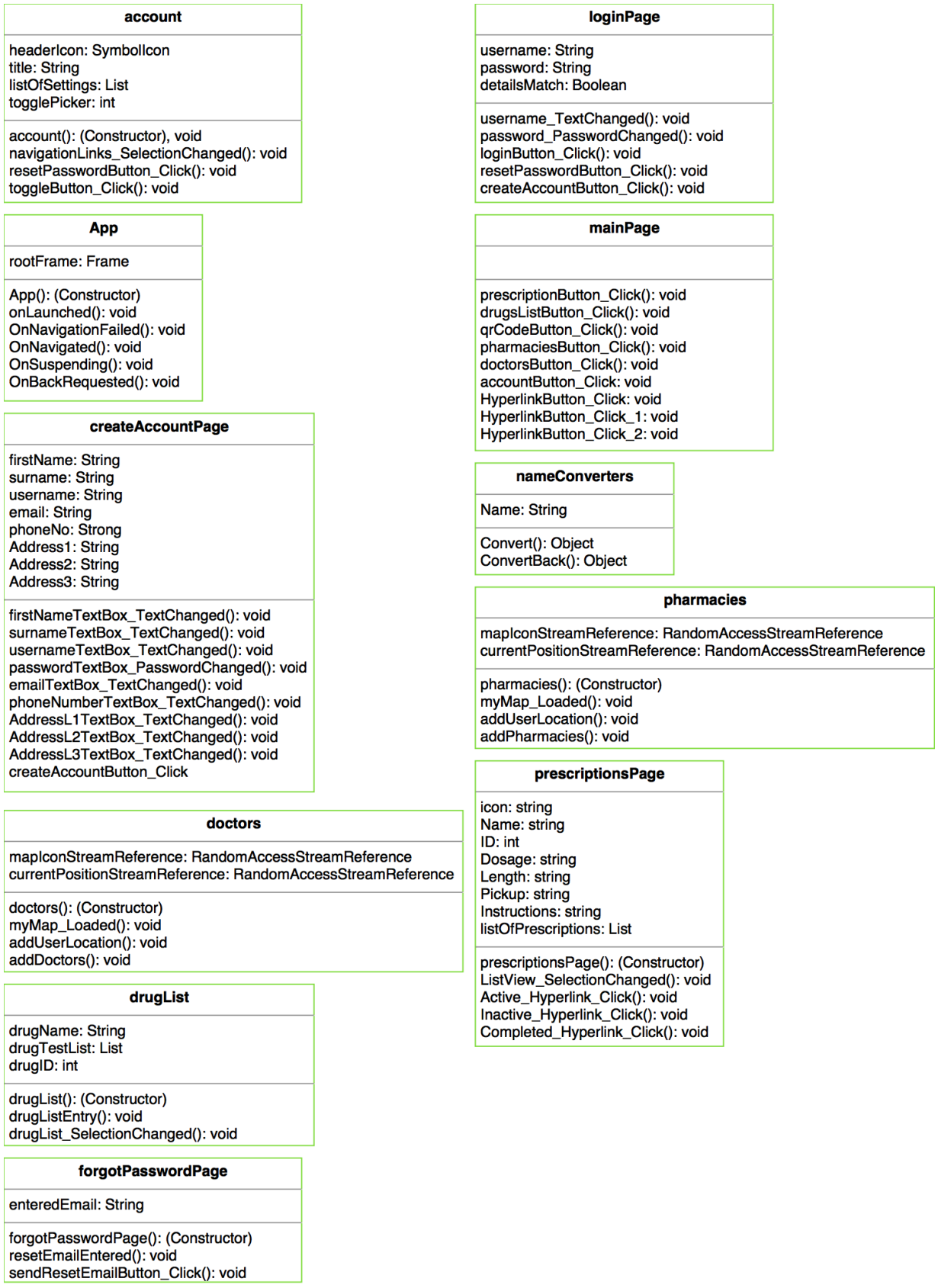
4.4 - Activity Diagram

This diagram shows the activities that the user can perform in the application:



4.5 - Class Diagram

Most classes in the application are designed around a particular screen - all functions in that class are focused on making that particular screen operate. However, there are a number of standalone classes which are used for other operations as well.



Note how many of these classes have very little or no variables declared in the class diagram. This is because almost all of these methods are the code-behind for either a XAML view, an Android activity or an iOS storyboard - the classes are controlling UI elements, which don't need to be declared again in the classes themselves. This also explains why most functions are void - they don't need to return anything, because the UI will be updated if the function performs correctly. Some classes are called in others to perform a specific function - for example, nameConverters accepts a string when called and uses the String to switch the colour of a UI foreground element. This is used on the prescriptions screen to switch the colour of the drug name and pickup text, depending on the value of the pickup text (see **Section 3.6 - Prescriptions** above).

5 - Discussion of Results

The following pages detail the results obtained once the project was completed.

5.1 - Implementation

In the past, asking a single developer to design, create, test and deploy three versions of an application on three different platforms would probably have been an extremely difficult task, even given a large amount of time. Using contemporary development tools and frameworks, we were able to develop a database driven application which runs natively on Windows, iOS and Android within 8 months. The fact that this was done by a student developer while also keeping up on assignments, deadlines and exams for college really speaks to the utility and capabilities for the .NET Framework and Xamarin.

Going forward, this experience with .NET, C#, Xamarin and Oracle will be invaluable in the workplace. A huge amount of software for Windows (and other Operating Systems) uses the .NET Framework and by extension C#. C# is also widely used outside .NET - it's the main programming language used for building Windows desktop apps in UWP and it's the primary language used in the Unity game engine too. Xamarin is also a hugely beneficial topic to have covered - knowing Xamarin means that applications can be published to a wider range of platforms even without in-depth knowledge of the platforms architecture. Finally, Oracle databases are the number one most popular database engines in the market today, according to db-engines.com[29]. Having experience working with Oracle's systems will be invaluable going forward.

5.2 - Cut Features

While a lot of features that were suggested in the initial specification were implemented successfully, a number of features had to be cut due to time constraints, or to make sure that other, more critical parts of the application were ready in time for the deadline. The following table shows a list of features which were initially intended for the application (this table appeared in the interim report), along with check marks (✔) to indicate that a user of this type has access to this feature. Features which made it into the application are marked in green, cut features are marked in red in the list on the left hand side of the table.

Feature List			
Feature	**Patient**	**Pharmacy**	**Doctor**
View Drug Information	✔	✔	✔
View Patient History	✔	✔	✔
Check upcoming prescriptions	✔	✔	✔
View Pharmacy Map	✔	✔	✔
View Doctor Map	✔	✔	✔
Check if a prescription has been collected	✔	✔	✔
View Patient List		✔	✔
View Complete Drug List		✔	✔
Compare Drugs		✔	✔
Prescribe a Drug			✔
Trigger Pharmacy Pickup	✔		
Edit Patient Sharing Options	✔		
View Incoming Prescriptions		✔	
Fill / Mark Prescription Complete		✔	

Most of the features that were cut from the application were done so due to time constraints - it was decided that all versions of the application should have feature-parity as the deadline approached. This meant that certain features were not ready in time for the final deadline as development time needed to be spent implementing existing features on existing platforms and making sure they worked properly. It was decided that the Patient view of the application should be prioritised - this is the version of the app which will see the most use and is the most likely view which will be used on mobile devices. Notably, most of the features marked in red above were half-finished when they were cut and references to these features exist in the late debug versions of the applications - however, references to these features were removed from the release version. Also, features which could be considered critical to application operation (for example, ‘prescribe a drug’) do not need to be implemented in the application for testing. For testing, the test user (Shown above in screenshots in **Section 3 - Application Operation**) has had the drugs in it’s account manually added to the database - this means that all the functionality that the patient user will see can be tested, even though there’s no way for a doctor user to prescribe the drug in the current version of the application.

5.3 - App Alternatives

An interesting observation was made about halfway through the project implementation - at this stage, early Windows and iOS prototypes were up and running on my devices and I was finally able to show people what I was working on. While showing the iOS version to my Grandparents on a tablet an observation was made - they were impressed by the graphics and were able to follow along with me as I explained the various app features. However, when I passed the iPad over so they could have a go, it quickly became clear that they didn't understand how to use the application.

This got me thinking about alternatives - what if a user wants to use the service, but is unable (or unwilling) to use the applications? A solution was suggested in the form of a card which contains a user's account details, which they can present to a pharmacy or doctor. In the below example, the user's details are stored in the QR code (which will match the QR code generated in the app) but the cards could also contain a magnetic strip or a barcode to store information. This card system would be an alternative for users who are uncomfortable using the application to store their details.

5.4 - Comparison to currently existing tools

In **Section 2.9**, a number of currently available tools were mentioned and briefly described. In this section, the project will be compared to these tools. The software packages mentioned above have one major disadvantage versus our application - they all still rely on paper receipts. Patients still need to visit the doctor, get the prescription written or printed and then manually bring it to the pharmacy before they can get their medicine. With our system, we can cut out these steps - the doctor pushes the prescription details to the patient's account and the user can select their pharmacy and send the prescription there for pickup. This cuts a number of steps out of the process, while still maintaining patient's confidentiality (see **Section 2.5** above).

Another advantage we have over the others is the application comprehensively covers the prescription process - from the patient visiting the doctor to picking up and taking their drugs. This also keeps a record of every step of the process in the HISTORY tables in the database. Doctors and pharmacies who are involved with a transaction will also be stored in the database, so the patient can easily find their contact details if they need to discuss anything regarding their prescriptions. By enabling closer contact between doctors, pharmacies and patients, we should be able to drastically speed up the process and take a lot of pressure off the medical professionals.

5.5 - Future Development

Going forward, the application may be of use to the medical community - however, a number of issues need to be addressed before the application would be ready for commercial launch:

- The cut features above need to be implemented on all platforms - these are planned to be up and running over the summer
- The MacOS version of the application needs to be completed, tested and debugged so that it has feature parity with the already existing versions
- As the application deals with sensitive user data and doctor/pharmacy work data (including medical records) a large amount of security processes will need to be implemented (See **Section 2.6 - Security** above).
- Additional features could be developed and implemented
- Beta versions need to be dispensed to medical professionals and a small group of test users. Feedback on the application in use by end users will be invaluable to future development

Once a secure, reliable, useful multi-platform application has been developed and strenuously tested, the application could be ready for deployment in the medical industry.

At this stage, a large amount of certification and qualification processes would need to be completed - for example, the proposed IEEE standard P1073 [30] .

6 - Conclusion

In conclusion, we set out to create an electronic prescription system using .NET. We wanted to create a multi-platform application which allows patients to more easily interact with doctors and pharmacies. With the project complete, an application has been created which will allow doctors, pharmacies and patients to interact with each other in a more direct manner while keeping a record of every transaction that occurs. Most of the features that were planned were implemented successfully. Users will be able to use a wide range of desktop and mobile devices to interact with the system and will be able to keep track of their medications more easily.

References

[1] M. Ahmadi, M. Samadbeik, F. Sadoughi and A. Garavand, "A copmarative review of electronic prescription systems: Lessons learned from developed countries", Journal of Research in Pharmacy Practice, vol. 6, no. 1, p. 3, 2017.

[2] P. Kierkegaard, "E-Prescription across Europe", Health and Technology, vol. 3, no. 3, pp. 205-219, 2012.

[3] L. Mudiwa, "Serious shortage of doctors — IMO", Irist Medical Times, 2016.

[4] P. Lynch, "The First Steps to E-Prescribing in Ireland", The Medical Independant, 2016.

[5] Apple Inc, iOS Wordmark. 2016.

[6] Microsoft Corp, Windows 10 Logo. 2016.

[7] Google LLC, Android Robot (Logo). 2014.

[8] "What is .NET?", Microsoft.com, 2018. [Online]. Available: https://www.microsoft.com/net/learn/what-is-dotnet. [Accessed: 14- Mar- 2018].

[9] Microsoft Corp, .NET Logo. 2018.

[10] C. Wodehouse, "C#: The Language of the .Net Framework", Upwork.com. [Online]. Available: https://www.upwork.com/hiring/development/c-sharp-developer/. [Accessed: 14- Mar- 2018].

[11] M. Torgerson, "Welcome to C# 7.2 and Span", .NET Blog, 2017. .

[12] J. Walker, M. Satran, N. Estabrook and A. Koren, "XAML overview", docs.microsoft.com, 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/uwp/xaml-platform/xaml-overview. [Accessed: 14- Mar- 2018].

[13] Xamarin Inc., Xamarin Logo. 2016.

[14] D. Junk, "How does Xamarin Work?", Aperta - Web Design and Development, 2015. .

[15] M. Wenzel, M. Jones, M. Hoffman and L. Latham, "Cross-Platform Development with the Portable Class Library", Docs.microsoft.com, 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/cross-platform/cross-platform-development-with-the-portable-class-library. [Accessed: 14- Mar- 2018].

[16] B. Kratochvil, "Multi-Platform Development - Portable Class Libraries: A Primer", MSDN Magazine, no. 28, 2011.

[17] Xamarin Inc, Xamarin.Forms Example Image. 2016.

[18] "Build a Native Android UI & iOS UI with Xamarin.Forms - Xamarin", Xamarin.com, 2018. [Online]. Available: https://www.xamarin.com/forms. [Accessed: 14- Mar- 2018].

[19]B. Nantz and L. Moroney, Expert Web services security in the .NET platform. Berkeley, CA: Apress, 2005.

[20]Applicure Technologies Ltd., "Best Practices for Database Security", Applicure.com, 2018. [Online]. Available: http://www.applicure.com/blog/database-security-best-practice. [Accessed: 14- Mar- 2018].

[21]"Porting Disaster", TV Tropes, 2018. [Online]. Available: http://tvtropes.org/pmwiki/pmwiki.php/Main/PortingDisaster. [Accessed: 14- Mar- 2018].

[22]Clanwilliam Health, QicScript Plus Logo.

[23]McLernons, McLernons Pharmacy Software Logo. 2018.

[24]TouchStore, TouchStore RX Logo. 2018.

[25]ClanWilliam Health, Helix Practice Manager Logo. 2018.

[26]ClanWilliam Health, Socrates Logo. 2016.

[27]Complete GP, Complete GP Logo. 2017.

[28]"Oracle Database Express Edition 11g Release 2", Oracle.com, 2018. [Online]. Available: http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/index.html. [Accessed: 14- Mar- 2018].

[29]"DB-Engines Ranking - Popularity ranking of database management systems", Db-engines.com, 2018. [Online]. Available: https://db-engines.com/en/ranking. [Accessed: 14- Mar- 2018].

[30]"Proposed standard IEEE P1073 Medical Information Bus", Ieeexplore.ieee.org, 2018. [Online]. Available: http://ieeexplore.ieee.org/document/37448/. [Accessed: 14- Mar- 2018].

Acknowledgements

I’d like to thank Reiner Dojen for supervising the project, my parents for putting me through college and everyone I spoke to at Xamarin Inc. for their support throughout the development of this project.

Appendices

Final application code for all three platforms has been included on a USB stick with this report. A representation of the answers received from interviews with doctors, pharmacies and prospective users has also been included. These appendices can also be viewed on GitHub at https://github.com/IanB14/FYP_FINAL .

Copy of Poster

Using .NET to Develop an Electronic Prescription System

UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

Ian Burke

B.Eng. in Electronic and Computer Engineering

Introduction

This project is designed to address and correct issues which exist in the healthcare industry today. The main aim is to make the prescription process easier for Doctors, Pharmacists and Patients by providing applications which handle the documentation and communication aspects of prescriptions electronically.

The applications are built using Microsoft's .NET Framework, Oracle's Database Architecture and Xamarin's cross-platform development software. This allows applications to be deployed on platforms which target a wide range of users while sharing code between the platforms.

Users can expect similar designs across all supported hardware, which makes it easy to switch between desktop, mobile and tablet versions of the applications while keeping their data up-to-date.

Aim

The main aim is to develop applications in .NET for multiple platforms which allow users to view and control their prescriptions electronically. Patients, doctors and pharmacies will share certain features, but not all features will be available to all users:

- Patients can log in and see information about their prescriptions such as the date it was prescribed, if a pickup is due, etc. Patients are also able to view doctors and pharmacies on a map
- Doctors will be able to view patient and drug information, as well as prescribe medications
- Pharmacies are also able to view patient and drug information and can dispense medications prescribed by doctors.

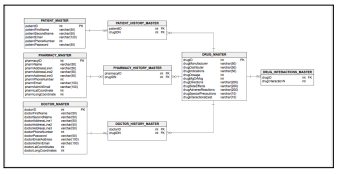
Method

Before any software engineering was begun, the following steps were taken:

- Speak to doctors and pharmacies, find out what technology they currently have access to and what they want to see in an application like this
- Use information gained from the interviews to create the design specifications

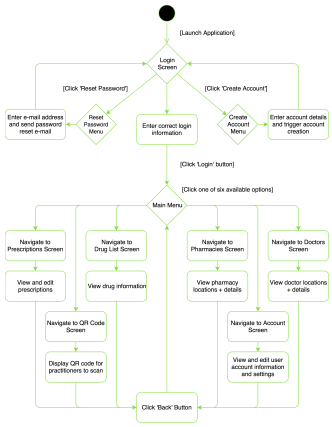
Once the spec is complete, the following steps can be undertaken:

1. Create the database and populate it with tables and data
2. Write code to access and edit data programmatically from the applications
3. Write shared C# code which can be run on all devices
4. Create device specific UI and API implementations for each target platform
5. Test and debug each platform in turn in preparation for release



Database Architecture

The applications are designed to be accessible and intuitive across all platforms – users should be comfortable using the app on a new platform if they already have experience with it. To help achieve this, all versions of the app have identical activities available (this diagram shows the patient's perspective):

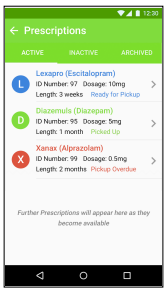


Activity Diagram

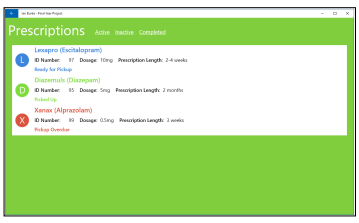
Results

The applications were completed and deployed to numerous test platforms. The following screenshots were taken on a OnePlus X, an iPhone SE and a custom Windows 10 PC. All three of these screenshots show the same screen on the various devices, highlighting the common visual language between the three.

iOS Screenshot



Android Screenshot



Windows 10 Screenshot

Login Screen Screenshot, showing desktop and mobile builds

In it's completed form, the project is supported on Windows 10 (desktop and touch devices), Android phones and tablets (6.0 and up) and iOS phones and tablets (iOS 10.0 and up). These platforms were picked to target a wide range of users across a wide range of device ecosystems – patients are expected to make more use of the mobile app, while pharmacies and doctors are expected to get more use from a computer running the app in their workplace.

In it's current form, the project isn't ready for use in the industry – it would need to undergo a huge amount of certification and qualification processes to make sure it's secure and reliable. However, the finished product has been shown to a number of doctors and pharmacies (many of who were interviewed during the planning phase of the project) and the replies were very positive.

Furthermore, the project has room to grow in the future – additional functionality could be added, existing features could be expanded and optimised and more platforms could be targeted. Xamarin provides support for porting to MacOS and Windows Phone devices and Mono (The framework Xamarin is built on) supports Linux distributions.

Conclusion and Personal Reflection

We set out to create a multi-platform application which uses Xamarin and the .NET Framework to operate. With the project complete, applications are available for Windows, Android and iOS devices which allow patients and medical professionals to communicate and work together easily.

If I was to begin the project again, I'd probably use Xamarin.Forms instead of Xamarin Native to construct the applications – this would have removed a significant amount of development time during the project by allowing even more code to be shared between devices.

Finally, the experience gained with Xamarin, C#, .NET and Oracle Databases will be extremely useful when working in industry after graduation.

Acknowledgements

I'd like to thank Reiner Dojen for supervising this project, my parents for putting me through college and everyone I spoke to at Xamarin Inc. for their support throughout the development of this project.

E&CE Department of Electronic & Computer Engineering