UNIVERSITY OF THE WEST *of* SCOTLAND

UWS

BSc (Hons) Computing Science

How can modern computer technology assist in effective diabetes management?

Christopher Rivett
B00249248

24th March 2017

Supervisor: Glenn Affleck

Declaration Form

This dissertation is submitted in partial fulfillment of the requirements for the degree of Computing Science (Honours) in the University of the West of Scotland.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

Name: CHRISTOPHER RIVETT

Signature: [illegible]

Date: 24th March 2017

Library Reference Sheet

Surname **Rivett**	
First Name **Christopher**	**Initials** **CR**
Borrower ID Number **B00249248**	
Course Code **COMPSCI**	
Course Description **BSc (Hons) Computing Science**	
Project Supervisor **Glenn Affleck**	
Dissertation Title **How can modern computer technology assist in effective diabetes management?**	
Session **2016/2017**	

Computing Honors Project Specification Form

Project Title: How can modern computer technology assist in effective diabetes management?

Student: Christopher Rivett **Banner ID: B00249248**

Supervisor: Glenn Affleck

Moderator: Carolyn Begg

Outline of Project:
The project will investigate how modern computer technology can assist in effective management of Type 1 Diabetes. This will cover solutions that use a small embedded computer used in conjunction with existing glucose monitors and or insulin pumps as well as apps which store blood sugar data and analyse it in a useful manner.

The project will involve the development of a mobile application for the Universal Windows Platform. The app, as well as recording blood sugar readings and insulin doses, will analyse this information in a useful manner to provide feedback on how a user can improve the management of their diabetes. The provided feedback will be about emerging trends in a user's recorded data or advice on how to raise or lower blood glucose levels at specific times to keep a user's average on track and as stable as possible. There will also be a server based component to complement the mobile application which will use a database and allow for data to be synced between devices and the cloud.

A Passable Project will:

(i) Review the literature on computerised diabetes management solutions and diabetes self-management applications
(ii) Identify areas where these solutions work well and how they could potentially be improved
(iii) Develop a diabetes self-management application that can provide basic feedback on a user's management of their diabetes

A First Class Project will:

(i) Provide a more in depth review of the literature
(ii) Develop a more advanced application that provides detailed feedback of a user's diabetes management over longer periods of time

Reading List:
Andrew P Demidowich, Kevin Lu, Ronald Tamler, Zachary Bloomgarden (2012) An evaluation of diabetes self-management applications for Android smartphones [Online] Available: http://jtt.sagepub.com/content/18/4/235.short

Joseph A Cafazzo, Mark Casselman, Nathaniel Hamming, Debra K Katzman, Mark R Palmert (2012) Design of an mHealth app for the self-management of adolescent type 1 diabetes: a pilot study [Online] Available: https://www.jmir.org/ojs/index.php/jmir/article/viewFile/jmir_v14i3e70/2

Emily Yang, Sana Ahmad (2016) 'DIY' artificial pancreas offers hope to patients with type 1 diabetes [Online] Available: http://pharmacytoday.org/article/S1042-0991(16)30697-1/abstract

OpenAPS.org - #WeAreNotWaiting to reduce the burden of Type 1 diabetes [Online] Available: https://openaps.org/

Resources Required: *(hardware/software/other)*

Visual Studio IDE – for Universal Windows Platform (UWP) app development
Suitable database software (e.g. MySQL or SQL Server Express)
Windows 10 Mobile device – for testing on a real device

Marking Scheme:

	Marks
Introduction	10
Background and Literature review	15
Design	20
Implementation	35
Testing	5
Conclusion and Recommendations	10
Critical Self-Appraisal	5

Signed:

Student	**Supervisor**	**Moderator**	**Year Leader**

IMPORTANT: ***By signing this form all signatories are confirming that any potential ethical issues have been considered and necessary actions undertaken and that Mark Stansfield (Module Coordinator) and Malcolm Crowe (Chair of School Ethics Committee) have been informed of any potential ethical issues relating to this proposed Hons Project.***

Contents

Acknowledgements

Firstly, I would like to thank my supervisor Glenn Affleck, and my moderator Carolyn Begg, for all their help and support during the course of the project.

Thanks also goes out to the many lecturers I have had throughout my time at UWS and the fellow students with whom I shared classes, for making my time at university all the more enjoyable.

I would also like to thank the various diabetes teams I have dealt with over the last 6 years at Crosshouse hospital who have always offered great support to help me with the management of my diabetes.

Lastly, I would like to thank my family for their continued support over the last four years whilst I have been working towards my degree.

Abstract

Computer technology can play a big role in modern diabetes management techniques. Poor management of diabetes can cause a huge burden on the NHS in the UK and costs of dealing with complications from poor diabetes management are on the rise. There are a number of computerised management solutions for patients available and these continue to advance as the technology behind them improves.

This project reviews the solutions already available to diabetic patients and creates an exemplar management application which aims to improve on these. The management application which was designed for the Universal Windows implements various functions designed to assist in diabetes management through the capture of readings, charts and automated calculations.

The application produced was written in C#. It targets both the PC and Mobile UWP device families and uses SQLite for storage of data locally. To compliment the app, a web service was developed to allow for the backup of data and syncing between multiple devices. The web service is a REST API written in PHP which uses a MySQL database for the storage of data uploaded from the UWP application.

To conclude the report, ways in which the existing implementation could be improved and also potential areas for future development are explored and discussed. These areas include ports to other platforms and integration with other devices such as continuous blood glucose monitors.

Chapter 1. Introduction

In recent times, modern computer technology has allowed great advances to be made in the area of diabetes management for patients with type 1 diabetes. Thanks to the use of small embedded computers, patients can now get an instant reading of their blood glucose whenever they need to. This enables users with type 1 diabetes to better judge insulin dose requirements before a meal and also to figure out if they are feeling unwell perhaps due to an abnormal blood sugar that is either too low (hypoglycaemic) or too high (hyperglycaemic). Modern technology continues to improve on these solutions, an example being that there now exist more convenient solutions such as continuous glucose monitoring devices that are arguably easier to use than traditional monitoring devices that involve the pricking of a finger to take their readings for users.

Keeping good control of their blood glucose levels is an essential part of a diabetic patient's lifestyle. While the tools and medication available are invaluable to patients, the success of the treatment is still very much down to the individual. There is a gap left in many solutions where while the information gained from them is useful, it is not always obvious what it actually means, especially over longer periods of time.

This project aims to develop an application that can gather the information collected by a patient and store it permanently for archival and analysis. The main focus of this project will be on how useful analysis can be provided to the user to help improve the management of their diabetes. This will be of particular use to users who are perhaps not as confident as they would like to be with their management techniques.

The application will work natively on and primarily target smartphones, but will also work on tablets, traditional PCs and laptops. It will take advantage of features such as the calendar for setting of reminders and on mobile devices, features such as NFC reading will be available on supported devices. A web service will also be developed to go with the app to allow for backup and syncing of the stored information between multiple devices that a user may have. Since the main use is expected to be on smartphones, this will be of particular use to a user if they lose their smartphone or it becomes non-functional.

Chapter 2. Background

2.1. Project Motivation

Dealing with type 1 diabetes for the last 6 years, I have managed to keep good control over my blood sugar levels. They have not been perfect however and I have continued to strive for improvement wherever possible. This got me interested in what solutions exist to help type 1 patients keep a tight grip on their blood glucose levels and provided the motivation for me to take on this project.

2.2 Existing Solutions

There are various different types of diabetes management solutions available to patients currently that make use of at least one computerised component. The main groups of potential users are listed below:

- Insulin pen users with glucose monitor who self-administer insulin injections
- Insulin pump users with an insulin pump which continually delivers insulin and allows variable rates of insulin to be administered around meal times
- Users of artificial pancreas systems that communicate with insulin pumps and continuous glucose monitors to automate the calculation and administration of insulin doses

2.2.1 Insulin pen users

Insulin pen users will usually have a glucose monitor which they must use at least 4 times a day (before each meal and once before bed) to take a blood glucose level reading. Most users are on a basal bolus insulin regime where they will inject a fast-acting bolus insulin dose before each meal and a long acting basal insulin dose to last throughout the day either in the morning or at night. In some cases, the basal insulin dose can be split and taken in two doses at different times of day. Patients in this group have to rely solely on their own judgement to adjust their insulin doses and perform any corrective steps to combat abnormal blood glucose levels.

2.2.2 Insulin Pump users

Patients with an insulin pump commonly have it coupled with a continuous glucose monitoring device. In some cases, this can even be integrated into the pump device itself. The pump does not act independently of the wearer and will require instruction from the patient before it can administer any insulin. Fast acting insulin is delivered in small amounts continuously at a user set rate with additional bolus doses of fast acting insulin given when eating. In this respect, insulin pump users are similar to insulin pen users described above as they require their own judgement along with the initial guidance of their diabetes care team to operate their insulin pump effectively. As insulin delivery from a pump is continuous throughout the day, more frequent monitoring of the user's blood glucose levels is required. This is to ensure corrective action is taken if too much or too little insulin is being delivered by the pump continuously and to keep glucose levels as stable as possible.

2.2.3 Artificial Pancreas Systems

An artificial pancreas system consists of a continuous glucose monitor and an insulin pump that communicate with each other to allow for automatic adjustments to insulin levels just as the Pancreas would do in people who do not have diabetes (diabetes.org.uk, n.d. a). A big advantage to a system such as this is that it helps to keep blood glucose levels stable and in doing so greatly reduces the risks of dangerously low glucose levels or 'hypos' during the night, which are a big worry to some people. These solutions still require the user to have a good knowledge of their diabetes management and make some decisions so it is not a complete "set it and forget" solution. Bolus insulin doses will need to be programmed in manually for each meal just like in a standard insulin pump based solution.

2.2.4 Relevance of a management application for these solutions

A management application can be particularly useful for insulin pen and insulin pump users to give them longer term and more detailed analysis than what their existing monitoring devices can provide. Most glucose monitors will only store reading data and perhaps display an average over set periods of time. The idea of a management app is to provide a much more useful analysis of the stored data that is easy to understand and detect emerging trends before they could potentially become a major issue.

With artificial pancreas systems, the use of a management app such as the one proposed in this project while still helpful, is not as relevant. A companion app would be useful to remotely view the status and analysis of the system from a smartphone; however, this sort of functionality is out with the scope of this project. The recording of insulin doses around meal times would likely be one of the most useful features of the app proposed in this project to a user of a system like this.

2.3 Technical Review

2.3.1 Universal Windows Platform (C#)

Windows 10 introduced the Universal Windows Platform (UWP) which is an evolution of the Windows Runtime (WinRT) that was introduced in Windows 8 and expanded upon in Windows 8.1 to allow for Universal Windows 8 apps that run on both Windows 8.1 and Windows Phone 8.1 powered devices (msdn.microsoft.com, 2016). The big advantage to using UWP for an app is that the base Universal device family API already targets a wide range of devices including IoT devices such as Raspberry Pi, Phones, PCs, Xbox consoles and even the Microsoft HoloLens headset. What this means is that the same base code can be used to power an app on any of these platforms. Adaptive code can be written to trigger certain UI features and device specific APIs on any platform where appropriate. This means that a single application package can be downloaded from the Windows store and expected to run on at least one and possibly all supported Windows 10 UWP platforms.

In the scope of this project, it is planned for the final app produced to be capable of being loaded onto a smartphone or PC/Tablet. One of the main reasons for choosing to develop this app on the UWP platform was the potential for the app to work seamlessly across multiple devices. Although the aim for the project is for Phone and PC compatibility, UWP also allows for the potential to expand to other device families at a later date.

While Android and iOS have much bigger market shares than Windows 10 Mobile, there are far less solutions that target Windows 10 UWP natively and the uniqueness of being able to run the exact same app on multiple device platforms was a factor in the decision to target this particular platform. There is no similar system available on any other vendor's platform for the same level of cross device compatibility and the development time for creating versions of the app for other device types is dramatically reduced by the existing code base being fully compatible.

2.3.2 SQLite

SQLite is a very small and powerful SQL based database engine. It is currently the most widely used database system available (sqlite.org, n.d.). SQLite's small size make it perfect for use in applications on memory constrained devices such as low end smartphones and embedded devices etc. This makes it a perfect choice for local data storage in this project as performance will remain high across a wide range of potential devices both low and high end. Another advantage to SQLite is its cross-platform compatibility. Almost every platform has at least one port of the SQLite engine and the single file that stores the SQL database can be moved between platforms without issue. These will be very useful features if the application will be ported to other platforms at a later date.

2.3.3 PHP and Slim microframework

PHP is a widely used open source server side scripting language that is well suited towards web development (php.net, n.d. a). PHP includes extensions which allow connections to databases such as MySQL which make it ideal for web applications that need to store large amounts of data. It is also the basis for the Slim microframework which will be used for powering the REST API for the web service.

Slim is a microframework for PHP that allows the creation of simple but powerful RESTful APIs. It is very simple due to the fact that at its core, it receives an HTTP request, calls a callback method and returns an HTTP response (slimframework.com, n.d.). This simplicity means that it is very fast and lightweight, requiring little code.

PHP and the Slim framework are compatible with a wide range of web servers such as Apache, IIS and Nginx which leaves open options for hosting across a variety of platforms. The use of a RESTful API means that there is no proprietary interface for communication with the web service, which allows potential access by a wide variety of different clients in the future.

2.3.4 MySQL

MySQL is one of the most popular open source Relational Database Management Systems (RDBMS) available (dev.mysql.com, n.d. a). MySQL stores data in separate structured tables that are linked by key fields, which is a reason it was chosen over a NoSQL solution. NoSQL solutions are more appropriate when dealing with large amounts of unstructured data. As all data stored by the proposed application will be related and of a consistent structure, this makes MySQL a perfect fit for the data storage component of the web service. MySQL's popularity for use with PHP, good community support and its availability on a number of platforms and web hosts were all deciding factors in the decision to choose MySQL.

Chapter 3. Design

3.1 Functional Requirements

Before any initial designs could take place, a set of planned features for both the client and server applications was put together. These are listed below.

Main client application

- Allow for recording of blood glucose levels in mmol/L
- Allow for recording of insulin doses
- Allow the user to set a reason for any insulin dose (meal or bedtime etc.)
- Display a graph of a user's average for set periods (7, 30 and 90 days) and for user defined periods.
- Allow a user to set a target range for their blood glucose (example: 4-10mmol/L)
- Display a pie chart of the total number of on and off target readings for set periods and for user defined periods.
- Allow for local storage of all data (using SQLite) with optional account creation for syncing with cloud storage (using MySQL)
- Offer advice to the user if there is a developing trend (such as blood sugars creeping up at a specific time each day) in their recorded reading data.
- Provide estimated HbA1c calculations to the user.
- Allow for future possibility of reading from an NFC compatible continuous glucose monitor – possibly prototype this using NFC stickers preprogramed with reading data
- Allow for optional notifications (using native device notification features) to the user if a large period of time has passed since their last blood glucose test.
- Allow for a user to export their data to an external file such as a csv file for offline archival purposes.

Server application

- Store readings in a MySQL database for synchronisation between multiple devices
- Allow for updates and deletions to stored data
- Host a management page for users to manage registered devices and change their password or email address
- Allow for the client application to download all of a user's data (useful if a user gets a new device or wants to set up a secondary device)
- Allow for administrators to manage or delete a user's account if necessary

3.2 Initial Screen Designs for Client Mobile App

Based on functionality from the functional requirements identified in the above section, a set of screen designs were sketched. These can be seen on the next few pages:

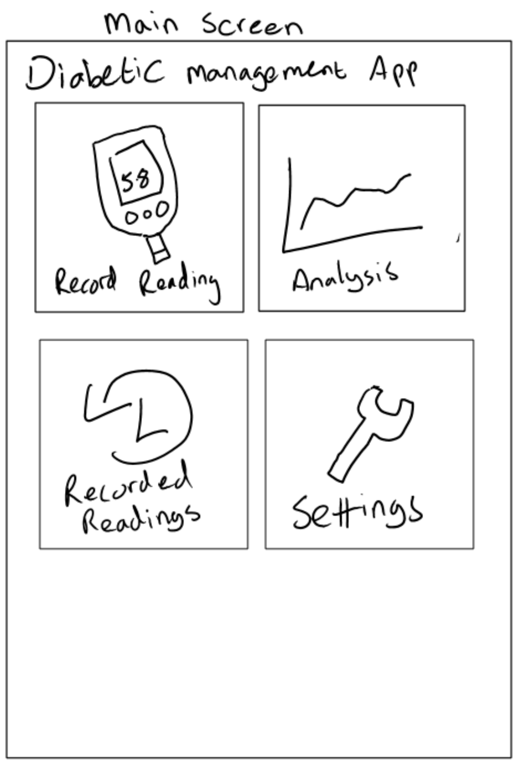


Figure 3.1 - Main Screen Design

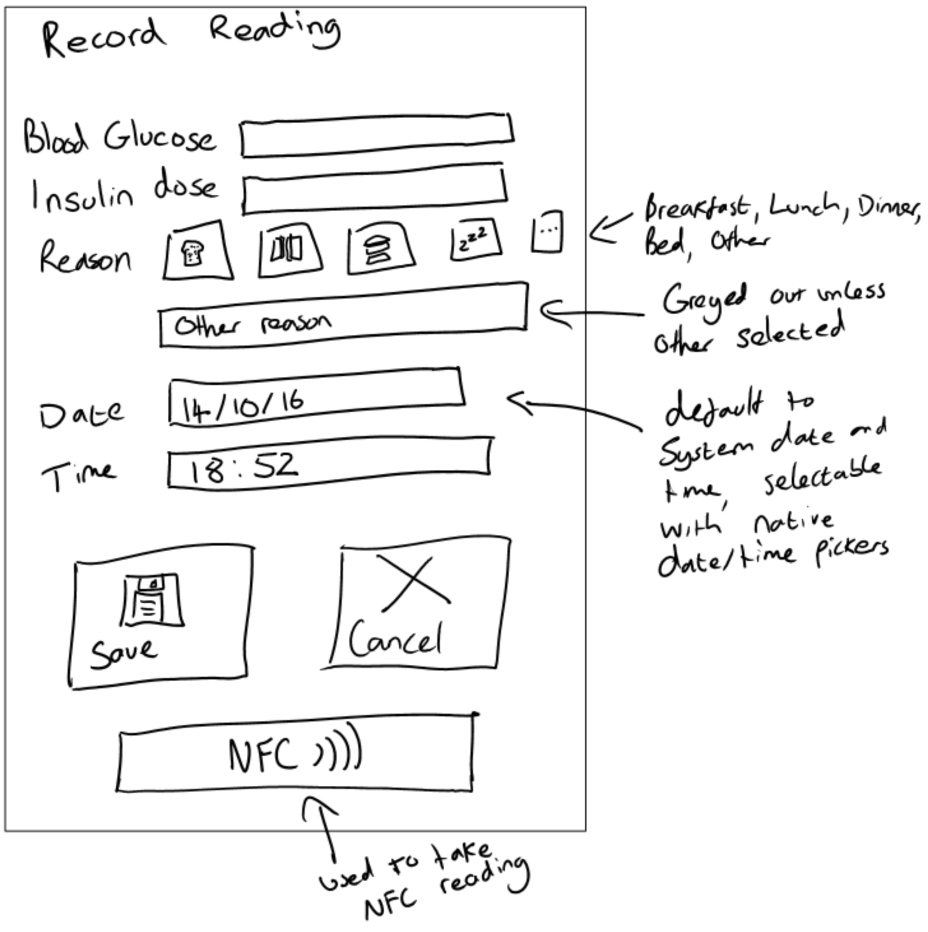

Record Reading
Blood Glucose
Insulin dose
Reason
Breakfast, Lunch, Dinner, Bed, Other
Other reason
Greyed out unless Other selected
Date
14/10/16
Time
18:52
default to System date and time, selectable with native date/time pickers
Save
Cancel
NFC
used to take NFC reading

Figure 3.2 - Record Reading Screen Design

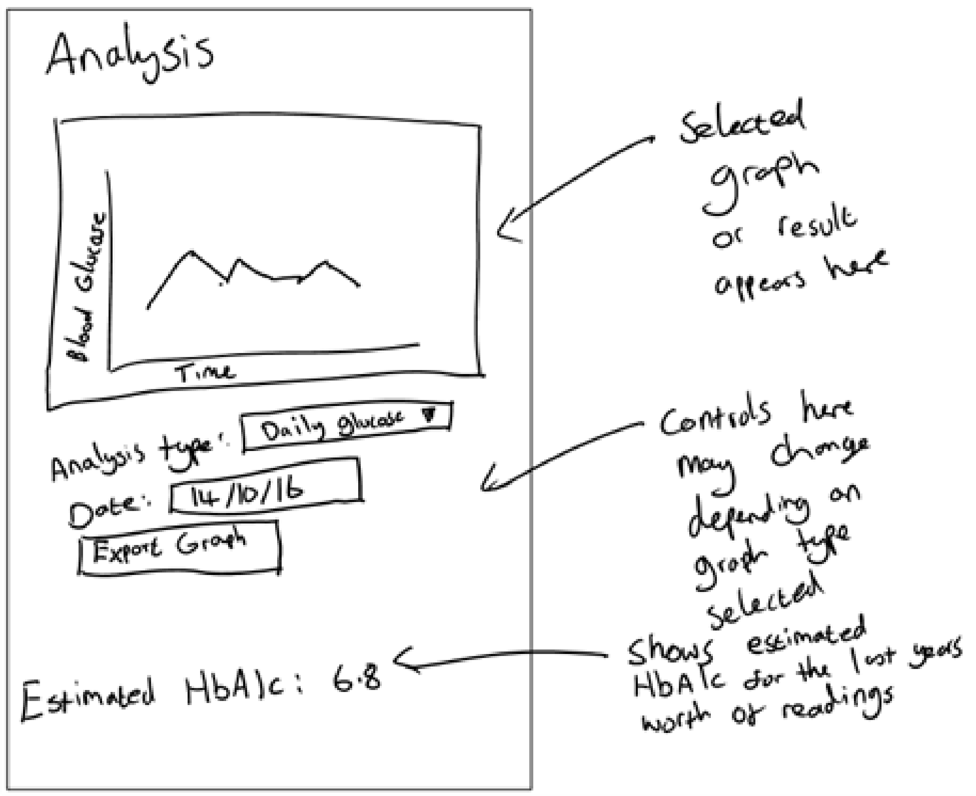

Analysis
Blood Glucose
Time
Analysis type: Daily glucose ▼
Date: 14/10/16
Export Graph
Estimated HbA1c: 6.8
Selected graph or result appears here
Controls here may change depending on graph type selected
Shows estimated HbA1c for the last years worth of readings

Figure 3.3 - Analysis Screen Design

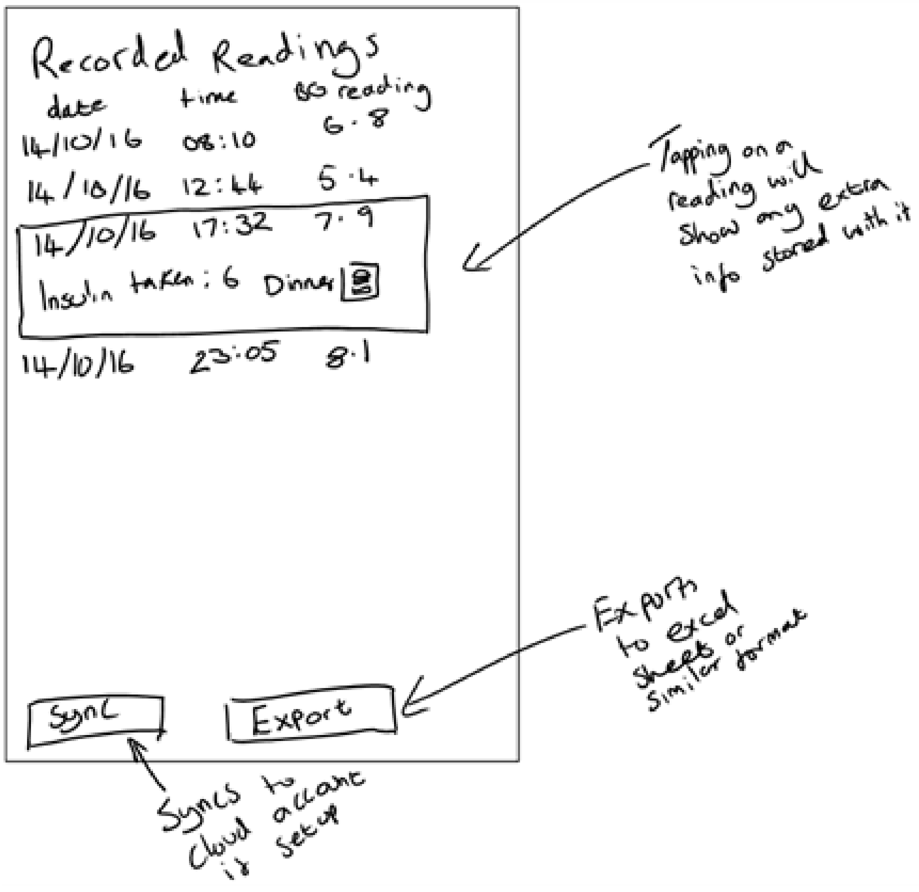


Figure 3.4 - Record Readings Screen Design

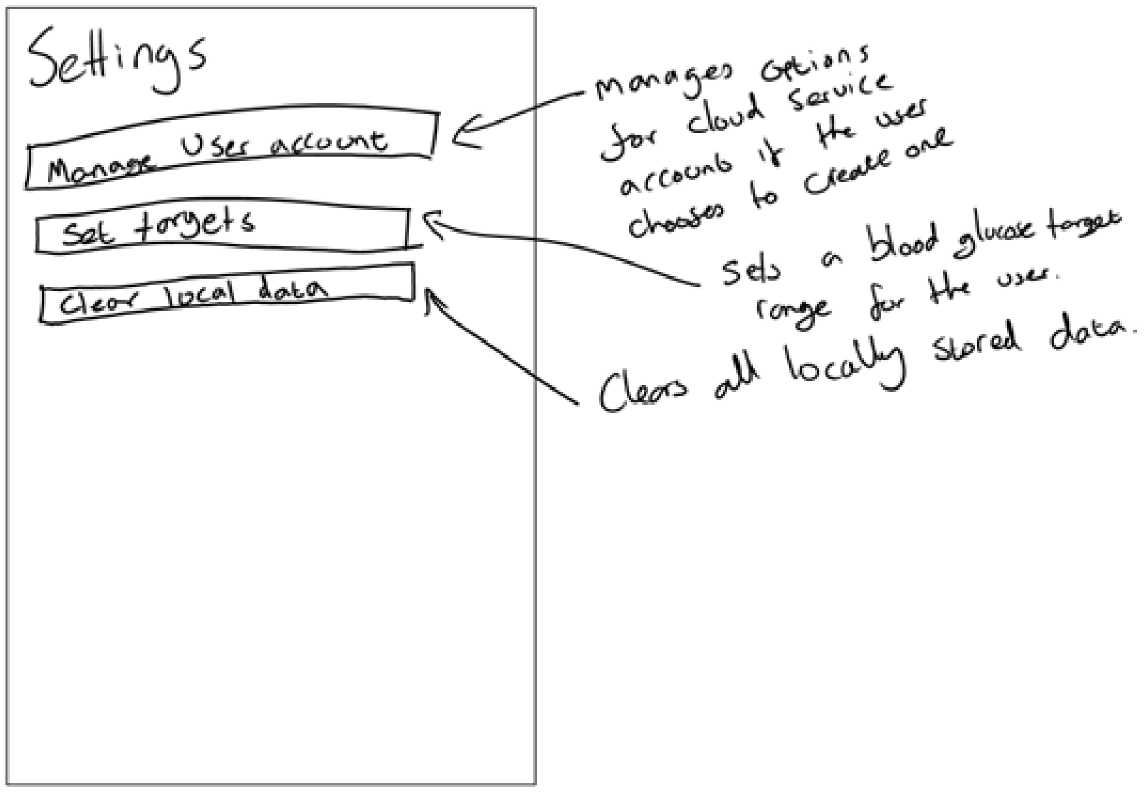

Settings
Manage User account
Set targets
Clear local data
Manages Options for cloud service accounts if the user chooses to create one
Sets a blood glucose target range for the user.
Clears all locally stored data.

Figure 3.5 – Settings Screen Design

3.3 Database design for Web Service

The

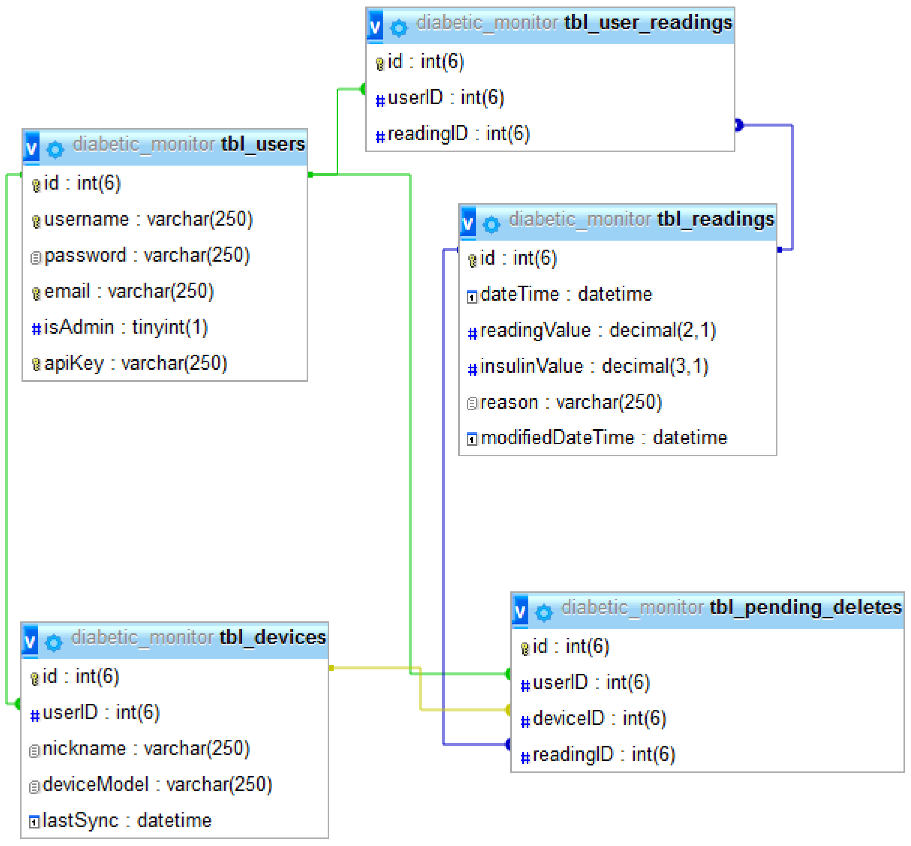


Figure 3.6 – Relational Diagram for Database

Database design is depicted in the relational diagram shown on the previous page in Figure 3.6, additionally a data dictionary has been included in Appendix A. It is designed to store user info, reading data, info on a user's device(s) and details of pending delete operations for devices.

Due to the way MySQL functions, all CHECK constraints are ignored by the database engine (dev.mysql.com, n.d. b). This means that all validation of data going into the database will be handled by either the Web Service or the Client Application.

3.4 Design for Web Service URL Structure

The following table shows the URL Structure for the REST based web service.

HTTP Method	URL	Description	Parameters
POST	/user	Creates a new user account	username, email, password
POST	/login	Logs in to an existing user account	username, password
PUT	/user	Updates existing user e.g. Password, email change or settings	password*, email*
GET	/users	Retrieves a list of all users on the service for admin users only	
DELETE	/user/{id}	Removes an existing user account	
POST	/reading	Adds a new reading	dateTime, readingValue, insulinValue, reason, imgSrc
PUT	/reading/{id}	Updates an existing reading	readingValue*, insulinValue*, reason*, modifiedDateTime
GET	/reading/{id}	Gets an existing reading	
GET	/readings	Gets all readings for a user	
DELETE	/reading/{id}/{did}	Deletes an existing reading	deviceID
POST	/device	Adds a new device and assigns it to the current user	nickname, deviceModel
GET	/devices	Retrieves all registered devices for a user	
DELETE	/device/{id}	Deletes an existing device	
GET	/sync/{id}	Retrieves a list of sync tasks for the user's current device	deviceID

Figure 3.7 - URL Structure of REST API

* - Optional Parameter

All required parameters with the exception of the IDs which are passed in the URLs, will be passed in using JSON in the HTTP Request body. For security, an API key will be used to uniquely identify each user. This API key will be returned to the client when the user logs in so it can be used in later requests. The API Key will be included in the request header for methods which require the user to be authenticated.

3.5 Sync Strategy for Web Service

To allow data to be synced reliably between devices, a sync strategy has been devised. To check for newly added readings, the time the current device last synced will be used to retrieve the IDs of readings which are new to that device or have been updated since the last sync. When syncing, if the client detects that the local copy of an updated reading is newer than the copy on the server, the user will be prompted to choose which one should be kept by the client app. For deletions, if the user only has a single device, then the reading will be removed from the readings table. When a user has multiple devices, an entry will be added to the pending deletes table for each other device that the user has. As devices sync and these details are retrieved, the corresponding entries will be removed. Once all pending sync tasks for a reading have been cleared, the reading itself will then be removed from the readings table.

Chapter 4. Implementation

4.1 Implementation of Client Mobile App

4.1.1 Model View ViewModel (MVVM) Pattern

Once basic implementation of the app had begun, while research took place for best practices in UWP apps, the MVVM pattern was discovered. This is a recommended pattern to follow when developing a UWP app or an app for any other platforms that use XAML for their UI. As suggested by the name, code is split into Model, View and ViewModel classes. The Model classes deal specifically with the data used by the application and often include validation for the data while the ViewModel deals with the logic that manipulates and exposes the data to the View. The View class will ideally only have the bare minimum of code required to create an instance of the user interface.

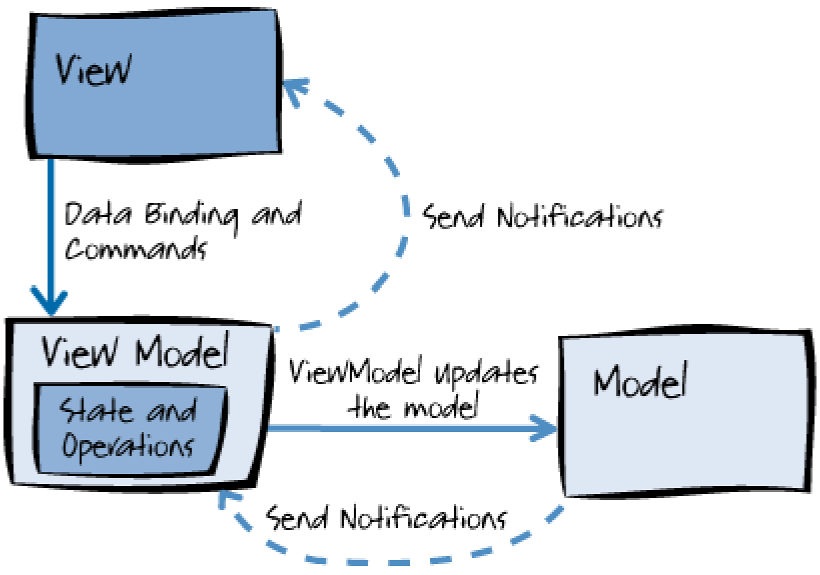


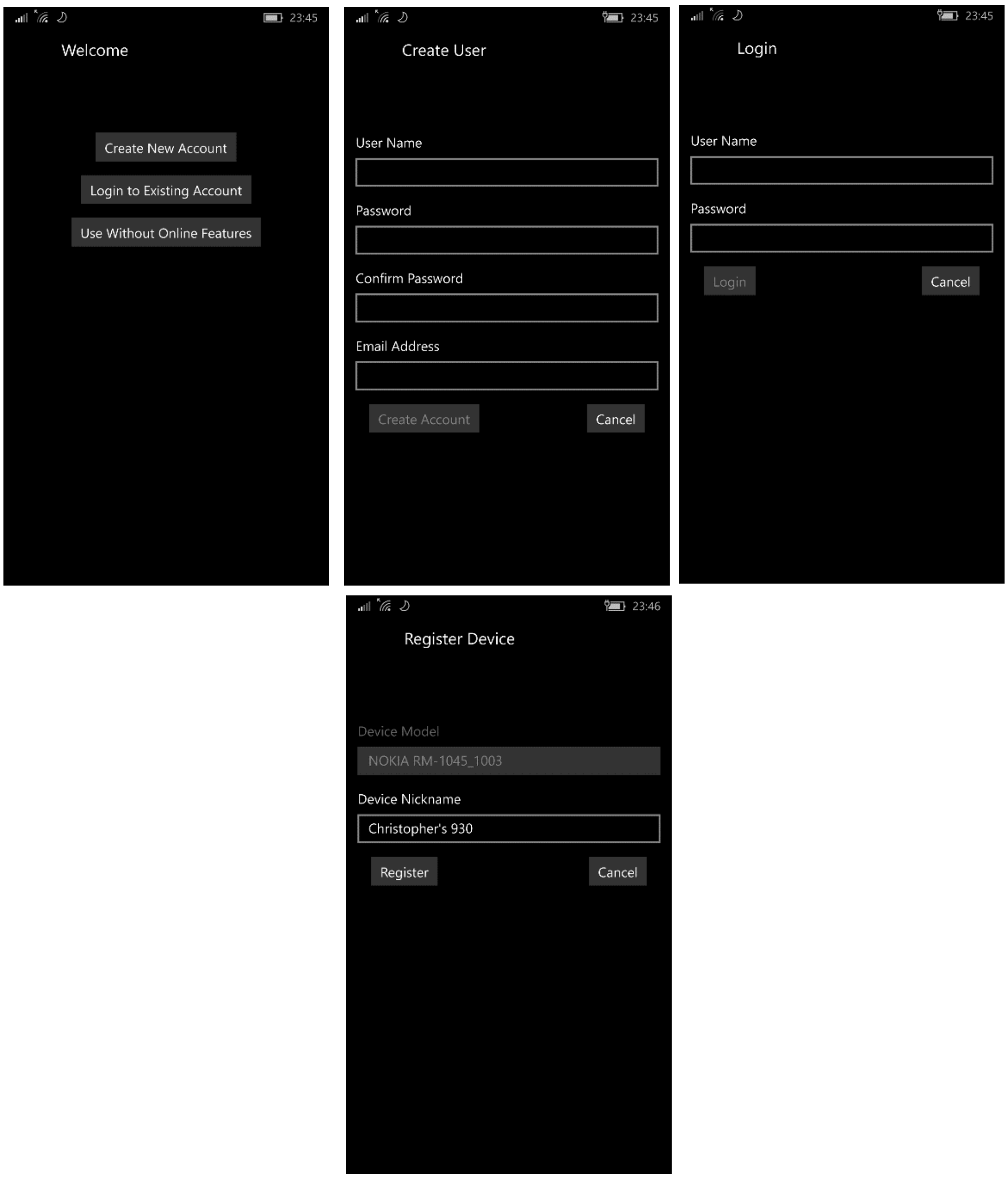
Figure 4.1 – Relationship between MVVM Components (source: msdn.microsoft.com)

The View is attached to a XAML file which defines the UI elements in an XML like way. One of the biggest advantages to using MVVM in this project is that it keeps the UI separate from the underlying logic which increases portability both across the UWP platform and beyond for potential ports to competing platforms. Since all logic will be contained in the ViewModel, multiple user interfaces can be designed and invoked appropriately depending on which device family in the UWP platform the app is currently running on. This was originally planned to be the approach taken to create the UI for the PC device family but it was later decided that the app should use adaptive triggers to enable the existing UI to adapt itself from Phone to PC and vice versa. Other benefits of MVVM include that it is easier to maintain code as each component is loosely coupled and can easily be replaced, testing is also improved as the application logic is held in a separate class that can be instantiated independently from its UI making unit testing much easier.

4.1.2 Template 10 Library

Template 10 is an open source library from Microsoft which provides a set of templates for developers to use and a set of recommended conventions to help boost developer productivity (github.com, 2016 a). It is intended to be used in XAML Apps written in C# for Windows platforms. There are a number of controls, behaviors, services, MVVM classes and more available in the library and many of these are used throughout this project. More detail on how and where these have been used will be discussed in the following, relevant subsections.

4.1.3 First Time Setup Experience



Device Model
NOKIA RM-1045_1003
Device Nickname
Christopher's 930

Figure 4.2 – First Time Setup Implementation

The app can optionally use a web service for data backup, so a welcome page that presents the user with the following options was created:

- Create a new account
- Login to an existing account

- Continue without using online features

When the user creates a new account, they are asked to enter their email and choose a user name and password. After this they are then moved to a device registration page where they are asked to register their current device to the new account by providing a nickname. The device model is also recorded at this stage and displayed in a disabled text box above the nickname.

Logging in to an existing account requires the user to enter their user name and password for the service after which they are moved to the device registration screen described in the previous paragraph.

Providing no connection errors occur in the above steps, any existing data stored on the service is downloaded and the user is then moved to the home screen where they can begin to use the app normally. If the user chose not to use any online features, then they are immediately taken to the home screen.

4.1.4 Navigation

As seen in the previous chapter on Design, the navigation for the app was initially planned to be on the main screen. During development, it was decided to change this to fit in more accordingly with the UWP design guidelines which suggest either a hamburger button menu or a tabbed interface along the top or bottom of the screen for navigation. A hamburger menu was picked for this app as it is very common in mobile app user interfaces across platforms making it easily recognizable as a navigation element amongst users. The menu contains the four options originally planned for the main screen as well as an additional one for returning to the home screen. The final implementation of the menu can be seen in the following screenshot.

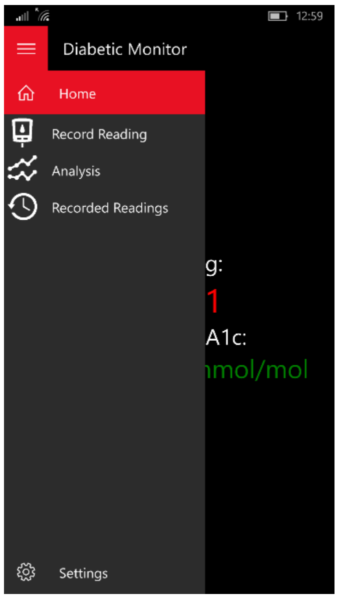


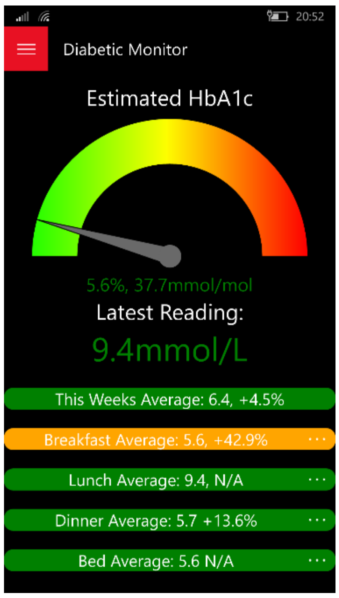
Figure 4.3 Main screen with menu open

Template 10's Hamburger Template was used which had all of the foundations for a hamburger menu control in place, this removed the need to create the control manually.

Usually for icons in the menu, scalable vector graphics from the Segoe MDL2 Assets font are used. However, there are no icons included in the font that fit in with the diabetes related functions of the app. To remedy this, the icons originally intended for the original home screen design were re-coloured white and used instead. Unfortunately, these graphics are PNG files and not scalable vector graphics, meaning they do not scale as well and do not fit in as nicely with the rest of the Segoe MDL2 icons used throughout the app. Given more time, custom scalable graphics could have been produced to be used in place of these images.

The settings button is at the bottom of the menu as it is standard practice in UWP hamburger menus that any user or settings buttons are classed as secondary controls. These appear at the bottom of the menu rather than in the main list with the primary controls.

4.1.5 Main Screen



5.6%, 37.7mmol/mol

Latest Reading:

9.4mmol/L

This Weeks Average: 6.4, +4.5%

Breakfast Average: 5.6, +42.9%

Lunch Average: 9.4, N/A

Dinner Average: 5.7 +13.6%

Bed Average: 5.6 N/A

Figure 4.4 – Main Screen Implementation

The design of the main screen has changed significantly since the original design. Once the decision was made to follow UWP guidelines more closely and use the Hamburger control for navigation, the original design with the tile style buttons was made redundant. It was decided that it may be useful for the user to have a quick summary of their current status on this page instead. Inspired by a diagram on diabetes.co.uk, it was decided to move the HbA1c estimate from the Analysis page to the main screen and create a visual representation similar to a speedometer. The control is a custom coded control in XAML based on an example shown by Microsoft's Jerry Nixon on MSDN (channel9.msdn.com, 2016). To create the control, a RingSegment is used to create the semi-circle shape which then has a fill applied to it with a LinearGradientBrush that produces the tri-colour gradient background. The needle is a custom Path object in XAML created using Microsoft Expression Design 4 which is aligned to be in the centre of the RingSegment. A rotation value is bound to the RotationValue variable in the ViewModel for the main screen. When this is set in the ViewModel, the position of the needle on the View is updated. A piece of code is used to map the percentage value of the HbA1c estimate to degrees so it can be accurately represented by the speedometer control. This is shown below:

```
        private double SetNeedleValue(double a1cPercentageValue)
        {
            if (a1cPercentageValue != 0)
            {
                return (((a1cPercentageValue - 5) * 180 / 7) + -90);
            }
            else
            {
                return -90;
            }
        }
```

Figure 4.5 – Code mapping percentage value to degrees

The range in percentage that can be mapped is between 5 and 12%.

The complete XAML code showing how the speedometer control has been implemented can be seen below:

```
<controls:RingSegment x:Name="HbA1cRing" EndAngle="90"
                                   InnerRadius="100" Radius="150"

RelativePanel.AlignHorizontalCenterWithPanel="True"
                                   RelativePanel.AlignVerticalCenterWithPanel="True"
                                   StartAngle="-90">
            <controls:RingSegment.Fill>
                <LinearGradientBrush EndPoint="1,0" StartPoint="0,0">
                    <GradientStop Color="Red" Offset="1"/>
                    <GradientStop Color="#FF23FF00" Offset="0"/>
                    <GradientStop Color="Yellow" Offset="0.49"/>
                </LinearGradientBrush>
            </controls:RingSegment.Fill>
        </controls:RingSegment>

        <Path Width="20"
              Height="150"
              RelativePanel.AlignHorizontalCenterWith="HbA1cRing"
              RelativePanel.AlignTopWith="HbA1cRing"
              Stretch="Fill"
              StrokeLineJoin="Round" Stroke="#FF000000" Fill="DimGray"
              Data="F1 M 119.667,20.1667L 110.333,276.833L 129,276.833L
119.667,20.1667 Z " RenderTransformOrigin="0.5,1">
            <Path.RenderTransform>
                <CompositeTransform Rotation="{Binding RotationValue,
Mode=OneWay}"/>
            </Path.RenderTransform>
        </Path>

        <Ellipse Height="25" Width="25" Fill="DimGray"
RelativePanel.AlignHorizontalCenterWith="HbA1cRing"
                 RelativePanel.AlignVerticalCenterWith="HbA1cRing" />
```

Figure 4.6 - Speedometer Control XAML Code

The estimated HbA1c (also known as A1c) value displayed by the speedometer control is calculated by putting the user's average blood glucose level over the last 90 days into the formula A1c = (90DayAvg + 2.5944)/1.5944. This is a derivative of the formula used to calculate average blood glucose from an HbA1c value (nedretande.com, n.d.).

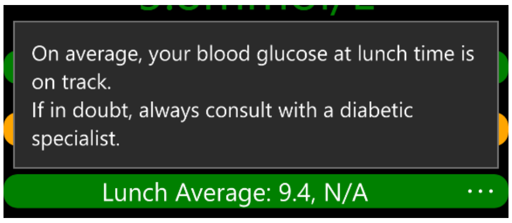
The result is intended to be an estimate of the HbA1c value that is obtained from a blood test at a diabetic review. It gives a rough idea of how high blood sugar levels have been over a period of roughly 90-days. The value of the HbA1c can be calculated either as a percentage or mmol/mol value. Since 2009 mmol/mol has been the default unit in the UK, however many people still receive their A1c as a percentage so the app will calculate the value in both units of measurement. A rough target for this value is 6.5% or 48mmol/mol, however targets are largely down to the individual and patients will usually have a personal target agreed with their care team to aim towards (diabetes.co.uk, n.d. a). Keeping this value on target greatly reduces the risk of a patient developing diabetes related complications later in life. A Model was created to store the result of the HbA1c as both a percentage and mmol/mol value. The code for the function that makes use of this Model and calculates the values is shown over the page in Figure 4.7.

```
        public HbA1c CalcuateHbA1c(ObservableCollection<Reading> range)
        {
            HbA1c result = new HbA1c();
            double totalBG = 0;
            int totalReadings = 0;

            foreach (Reading r in range)
            {
                totalBG = totalBG + r.BloodGlucoseReading;
                totalReadings++;
            }

            result.HbA1cAsPercentage = Math.Round((GetAverage(range) + 2.5944)
                / 1.5944, 1, MidpointRounding.AwayFromZero); // Calculate HbA1c as
percentage and round to 1 decimal place
            result.HbA1cAsmmolmol = Math.Round((result.HbA1cAsPercentage - 2.15)
                * 10.929, 1, MidpointRounding.AwayFromZero); // Calculate HbA1c as
mmol/mol and round to 1 decimal place

            if (double.IsNaN(result.HbA1cAsPercentage))
            {
                result.HbA1cAsPercentage = 0;
                result.HbA1cAsmmolmol = 0;
            }

            return result;
        }
```

Figure 4.7 - CalculateHbA1c method

After the HbA1c value, the last reading the user entered is shown so that they can see what their last blood glucose value was at a glance on opening the app. The colour of both the text displaying the HbA1c value and last reading entered will change colour depending on weather the value is on or off target. To check if the last reading was on target, the app checks it against the target values that the user can specify in the Settings page of the app.

Detailed analysis in the form of averages with overall changes are also included as part of the main screen. Averages for both the current week and previous week are calculated as part of this analysis and an overall change figure is produced. When the overall change is larger than 33% then the colour of the relevant strip will change to orange. An example of this can be seen for the breakfast average in Figure 4.4. When no data is available for average calculations during the current week, the message “Insufficient Data” is displayed and when no data on the previous week is available, “N/A” is displayed in place of the overall change percentage. The colour of each strip will turn red if the average is off target, otherwise it will be green, unless the aforementioned criteria for it to turn orange has been met. To provide feedback to the user, tapping on one of the strips with the ellipsis icon will open a Flyout with some generic feedback. This is shown in figure 4.8 over the page:


On average, your blood glucose at lunch time is on track.
If in doubt, always consult with a diabetic specialist.

Lunch Average: 9.4, N/A

Figure 4.8 – Lunch Average with Flyout open

Since each user will receive different advice from their diabetic specialists, no exact instructions are ever given by the app, just pointers that may help set the user in the right direction for getting back on track. A code listing of the DetailedAnalysis class is included in Appendix B.

4.1.6 Recording Readings

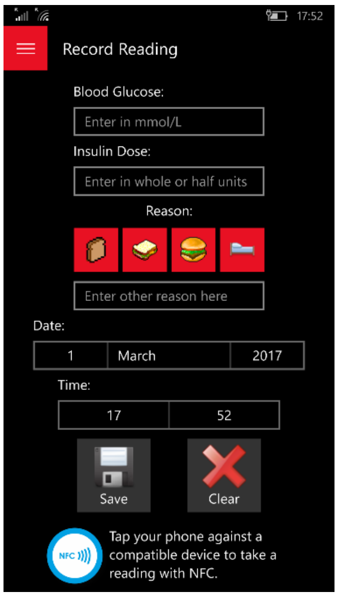


Figure 4.9 – Record Reading Screen Implementation

The Record Readings screen follows the original design very closely with minimal changes. Blood Glucose and Insulin doses can be entered into the relevant text boxes and a reason for the reading can be added either by pressing one of the buttons to add a pre-set value, or providing a custom value in the text box provided. The buttons will set the text in the text box to their corresponding reason. The icons on the buttons were chosen as they represent four of the most common reasons why a potential user would be recording their blood glucose. The date and time pickers default to the current system date and time and can be adjusted to values in the past for recording data that was not entered straight away at the relevant time. Once all the text boxes have been filled, the Save button should become active and allow the reading to be saved if it is valid.

Input validation for the values has been added through the Template 10 Validation library. Originally the validation rules were going to be added to the Reading class that contains the model for reading values, but this caused conflict with the SQLite database used for storage of the data. As a workaround, a second Model class named ValidateReading was created to handle validation of the values. A copy of this class has been included in Appendix C. It uses REGEX expressions to ensure the format of the blood glucose and insulin doses are valid and also checks to ensure that the date and time values are not set in the future. Obviously erroneous values such as over 40 for blood glucose and negative values for both the blood glucose and insulin dose values are also rejected. When any invalid input is detected on a save attempt, a dialog box appears explaining the issues so the user can correct them before trying again.

The Save and Clear buttons both make use of the Template 10 MVVM class "DelegateCommand". Each button has an instance of this class bound to its Command property in the XAML file for the page. When the values bound to the text boxes change, an expression is used to determine whether either button is able to be pressed. The Save button only becomes active when all the text boxes have been filled and the Clear button becomes active whenever one or more of the text boxes have values entered. The DelegateCommand for saving a reading can be seen below:

```
private DelegateCommand _SaveReadingCommand; // Allows SaveReading() to execute
when conditions in CanSave() are met
        public DelegateCommand SaveReadingCommand => _SaveReadingCommand ??
(_SaveReadingCommand = new DelegateCommand(SaveReading, CanSave));
```

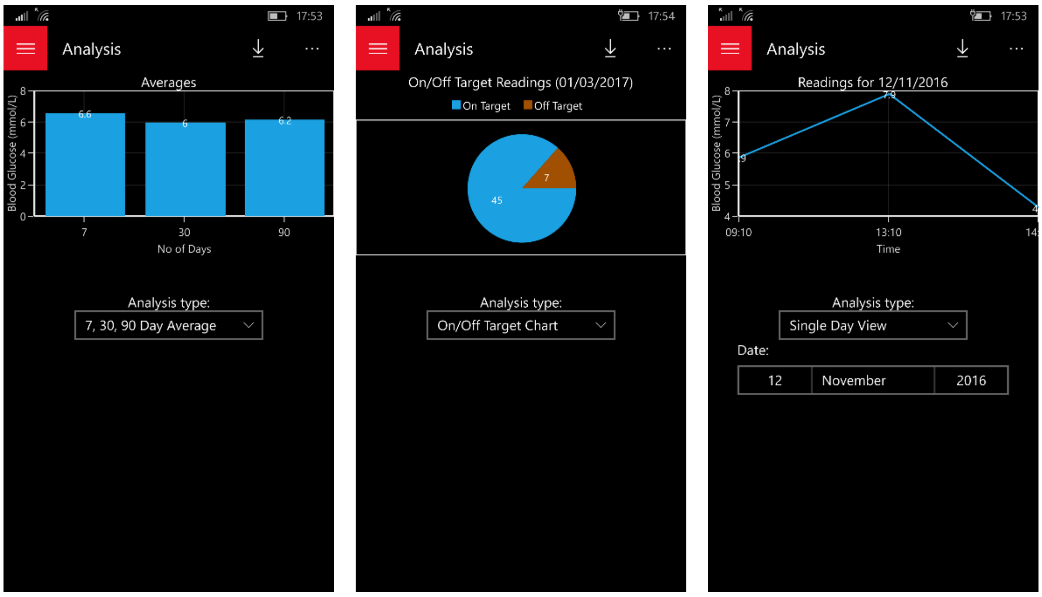
Figure 4.10 - Delegate Command for saving a reading

When defining a new DelegateCommand, it takes two parameters, one is the method to be executed, while the other is a function that returns a Boolean value to determine whether the method can be executed or not.

When a reading is saved, it is stored as an instance of the Reading class in the SQLite database and if a user account has been linked to the app, an attempt is made to upload the reading to the web service. Details of how the SQLite database is implemented are available in Appendix D and details of how the app communicates with the web service are included in Appendix E.

An alternative method of adding readings to the app is provided in the form of NFC support. The idea behind this is to support direct entry from compatible glucose monitoring devices that enable the user to use an NFC device such as their phone to scan for a glucose reading rather than having to prick their finger for a small blood sample. Currently as this app is only a prototype, it does not work with any real devices that provide this functionality. However, the NDEF Library for Proximity APIs / NFC (github.com, 2016 b) has been used to write some code that will read values from pre-programmed NDEF Tags and automatically enter them in the blood glucose value text box in the app. For devices that do not support NFC, such as phones that lack the capability or PCs without a reader, code has been written to handle this to prevent the app crashing and the NFC icon will simply not appear in these cases. A source listing for the NFC code is included in Appendix F.

4.1.7 Analysis



Analysis type: 7, 30, 90 Day Average

Analysis type: On/Off Target Chart

Analysis type: Single Day View

Date: 12 November 2016

Figure 4.11 – Analysis Screen Implementation

The analysis page offers the user the ability to create 4 different types of graphs listed below:

- 7,30,90 Day Average Bar Chart
- On/Off Target Pie Chart
- Single Day View Line Graph
- Custom Average Bar Chart

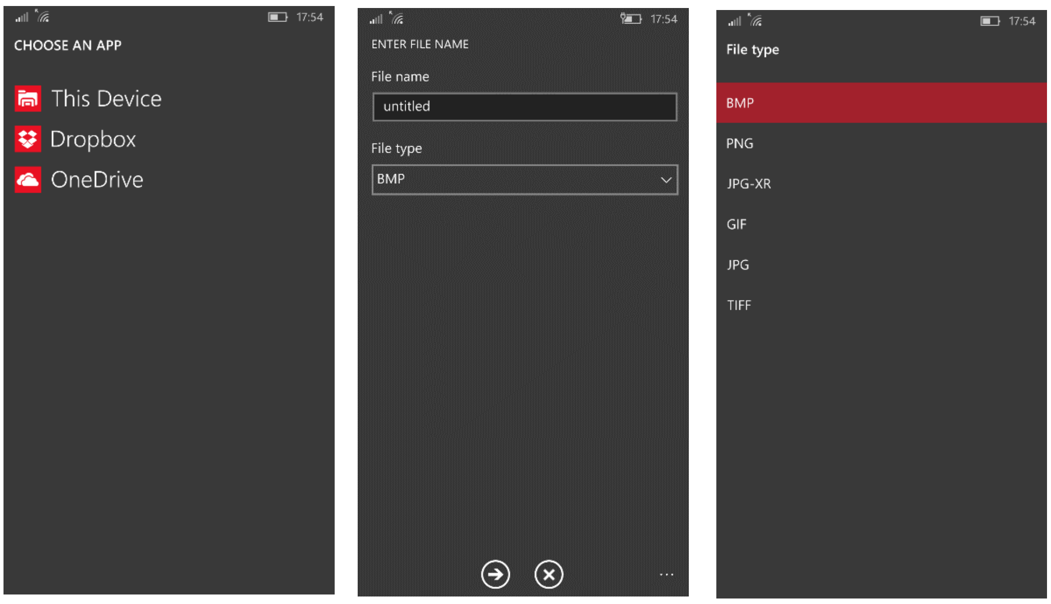
A ComboBox is used to select which type of graph is to be drawn and an event handler for the combo box triggers changes to the visibility of each graph when a new value is selected. On the XAML code for the page, each type of chart (pie, line and bar) is defined separately and the visibility, title and content is manipulated once the ComboBox value is changed.

The graphs are drawn using a third-party set of components available for Visual Studio called Syncfusion. The SfChart control from the Syncfusion package is what is used to create the graphs instead of the built-in tools from Visual Studio and the UWP platform itself. This approach was chosen as Syncfusion is available for a number of platforms including iOS and Android. This leaves the code in a very portable state if it was ever to be used in a port to one of these platforms in the future. The XAML code for the Single Day View chart is shown over the page in Figure 4.12.

```
<syncfusion:SfChart Name="singleDayChart" AreaBackground="Black"
AreaBorderBrush="White" Background="Black"
                                    Foreground="White" Header="{Binding ChartTitle,
Mode=TwoWay}"

RelativePanel.AlignHorizontalCenterWithPanel="True">
                    <syncfusion:LineSeries ItemsSource="{Binding Points}"
                                           XBindingPath="TimeOfReading"
                                           YBindingPath="BloodGlucoseReading" >
                        <syncfusion:LineSeries.AdornmentsInfo>
                            <syncfusion:ChartAdornmentInfo ShowMarker="True"
ShowLabel="True" />
                        </syncfusion:LineSeries.AdornmentsInfo>
                    </syncfusion:LineSeries>
                    <syncfusion:SfChart.PrimaryAxis>
                        <syncfusion:CategoryAxis Header="Time" />
                    </syncfusion:SfChart.PrimaryAxis>
                    <syncfusion:SfChart.SecondaryAxis>
                        <syncfusion:NumericalAxis Header="Blood Glucose (mmol/L)"/>
                    </syncfusion:SfChart.SecondaryAxis>
                </syncfusion:SfChart>
```

Figure 4.12 – Single Day Chart XAML Code

The Export Graph function was very easy to implement using the tools available with Syncfusion. It outputs an image file in a variety of formats including BMP, JPEG and GIF which can be saved onto a cloud service such as Dropbox or OneDrive or the device itself. Examples of the output produced for each type of graph are shown on page 33. The process of exporting a graph once the export button has been pressed is shown below:



File name: untitled
File type: BMP

Figure 4.13 – Process of exporting a graph

The user is first presented with a choice of where to save their exported graph, followed by an option to name the file and choose a file type. Syncfusion provides a Save method that can be run on each instance of the SfChart control, however a limitation of using the parameter-less version of the method is that it does not take a parameter to override the file name of the chart, so this must be specified manually by the user and has a default value of “untitled” as seen above.

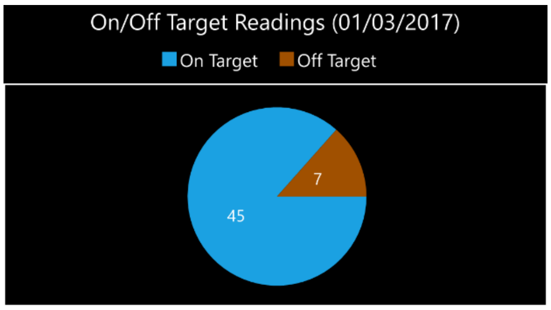


Figure 4.14 – On/Off Target chart

The on/off target chart gives a visual representation of how many times the user’s readings have been on or off target.

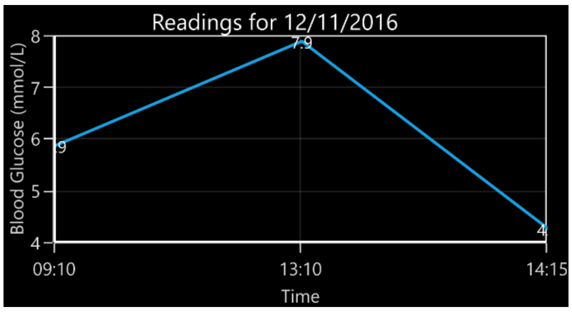


Figure 4.15 – Single Day View chart

The Single Day View chart will plot points of each blood glucose reading recorded for a given day and gives a visual representation of how stable the user’s blood glucose readings have been throughout the day. The target would be for the user to try and keep the line as straight as possible.

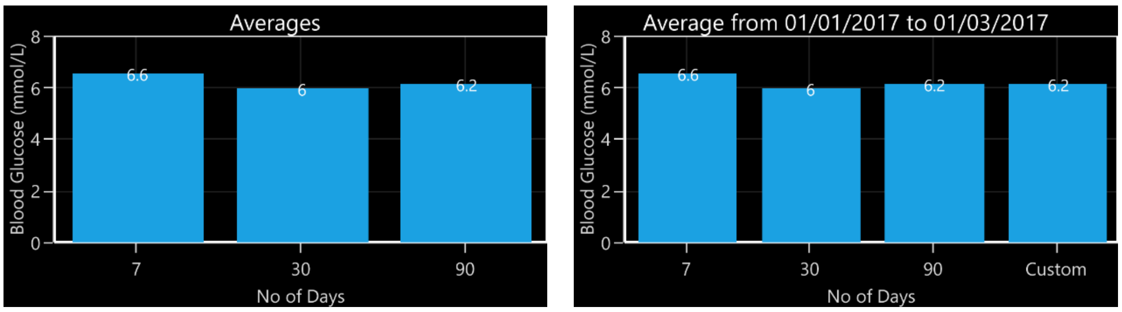
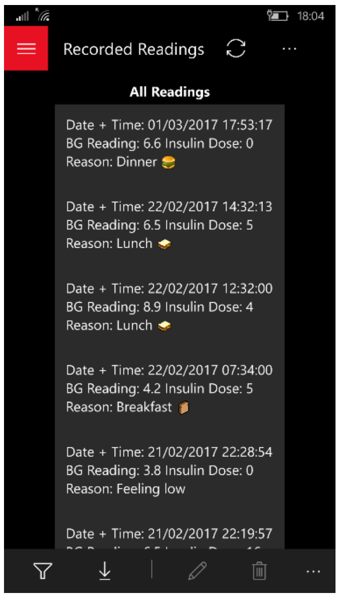


Figure 4.16 – Average chart (left) + Custom Average chart (right)

The average charts show how well a user has managed to control their glucose levels on average over set periods of time. The fourth bar in the Custom average chart represents the time span shown in the title of the graph, which is set by the user when the graph is produced.

4.1.8 Recorded Readings


18:04

Recorded Readings

All Readings

Date + Time: 01/03/2017 17:53:17
BG Reading: 6.6 Insulin Dose: 0
Reason: Dinner

Date + Time: 22/02/2017 14:32:13
BG Reading: 6.5 Insulin Dose: 5
Reason: Lunch

Date + Time: 22/02/2017 12:32:00
BG Reading: 8.9 Insulin Dose: 4
Reason: Lunch

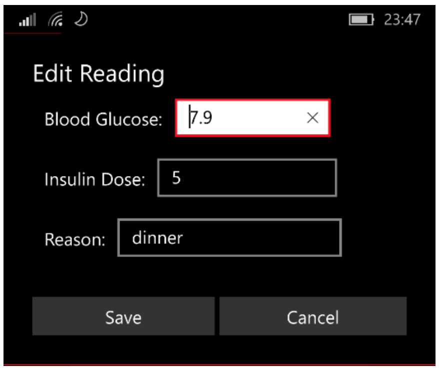
Date + Time: 22/02/2017 07:34:00
BG Reading: 4.2 Insulin Dose: 5
Reason: Breakfast

Date + Time: 21/02/2017 22:28:54
BG Reading: 3.8 Insulin Dose: 0
Reason: Feeling low

Date + Time: 21/02/2017 22:19:57

Figure 4.17 – Recorded Readings Implementation

Recorded readings are displayed in a ListBox where each item can be selected by tapping on it which enables the Edit and Delete options in the CommandBar at the bottom of the screen. When Edit is selected, a pop up will appear that enables the blood glucose value, insulin dose and reason to be edited. Once this has been updated, the reading is saved back to the SQLite database and a PUT request to update the reading on the web service is sent if applicable. A similar process happens when Delete is pressed where the user is asked to confirm whether they really want to delete the reading before it is removed from the SQLite database and a DELETE request sent to the web service. The editing dialog uses the same input validation described in section 4.1.6 on Recording Readings. An example of the dialog can be seen below:


23:47

Edit Reading

Blood Glucose: 7.9

Insulin Dose: 5

Reason: dinner

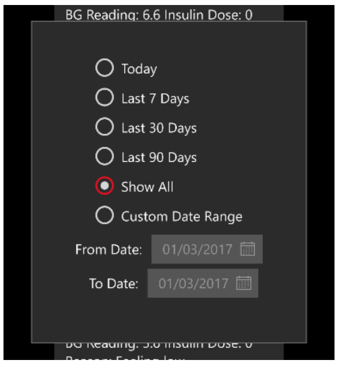
Save Cancel

Figure 4.18 – Edit Reading Dialog

When editing a reading, if the modified date and time on the app and web service differ, the user will be presented with a dialog asking them which version of the reading to keep.

The ListBox has an ItemTemplate to set the formatting and content of each of the items that are added. Bindings are used so that when a Reading object is added to the list, its values are bound to the appropriate TextBlocks in the ItemTemplate. This means each item is uniform with the rest in terms of look and feel even after any edits are made.

The left most button in the CommandBar at the bottom of the screen is for filtering readings. When this is pressed, a Flyout opens with a set of radio buttons and calendar pickers for the user to specify ranges of readings to be displayed. An example of this is shown in the following screenshot:



BG Reading: 6.6 Insulin Dose: 0

Today
Last 7 Days
Last 30 Days
Last 90 Days
[✓ Show All]
Custom Date Range

From Date: 01/03/2017
To Date: 01/03/2017

Figure 4.19 - Filter readings flyout

Once an option has been selected, with the exception of the "Custom Date Range" option, the flyout closes to reveal the updated contents and title for the reading list which reflects the user's choice. If the user chooses the "Custom Date Range" option, the Calendar Pickers become enabled to allow a custom date range to be set. When a valid range has been selected, the Flyout will close to reveal the selected range in the list box. As is standard practise for Flyout controls, tapping away from the Flyout will close it without making any changes.

The button next to the Filter button allows the user to export their readings to a CSV file for reading in an external spreadsheet application such as Microsoft Excel. This uses the 3rd party library Chilkat (chilkatsoft.com, n.d.) which provides easy creation of CSV files by defining column titles and then values for each cell. Once the CSV file has been created it is exported as a string which is then saved as a CSV file to either the local file system or a cloud service such as OneDrive or Dropbox using a file picker method very similar to the process of exporting a graph shown in Figure 4.13. The file name for the file is defaulted to "Exported Readings" and the only option for file type is CSV. A source listing of the Export method is included in Appendix G.

At the top right of the page is a sync button, this will only be enabled if the user has linked the app to an account with the web service and makes use of the DelegateCommand class provided in the Template 10 MVVM implementation to determine whether it can be tapped or not. When a sync is run, the app will check for any sync errors stored in the sync errors table of the SQLite database and retry any additions, updates or deletions that were made previously but could not be made on the web service due to a lack of internet connectivity. After any sync errors are dealt with, the app will then make a sync request to the web service which will return any readings that have been updated or added since the device currently running the app last synced with the service, as well as the IDs of any readings that have been deleted server side and still need to be removed on that particular device.

4.1.9 Settings

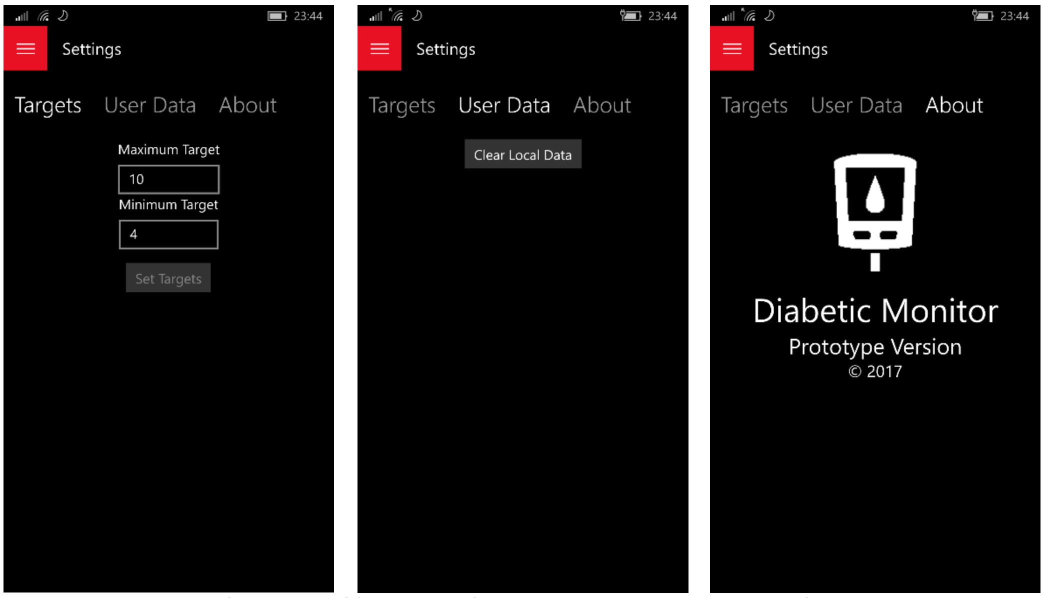


Figure 4.20 – Settings Screen Implementation

The settings page provides a way for the user to set their target blood glucose range and clear all local data stored by the app. A pivot control has been used to separate the content by category, as seen above in Figure 4.20. The Set Targets button uses a DelegateCommand to determine when the values in the textboxes for target values have change d and only becomes active when at least one of these values is different from when the user navigated to the Settings page. When the user chooses to "Clear Local Data" they are presented with a confirmation dialog to make sure that is what they really want to do.

The values for the settings themselves are stored in the areas provided by the UWP platform for application settings. There are 2 types of settings defined by the platform that are used in the app, the first is LocalSettings and the second is RoamingSettings. LocalSettings are settings that are stored permanently for as long as the app requires them on the local device whereas RoamingSettings are stored in an area where they are synced to the user's Microsoft Account. This allows settings to be synchronised between devices with minimal effort and enabled the design for the web service to be simplified as it did not have to deal with providing a way to synchronise a user's target range to the service.

Template 10 offers an easier way to gain access to the designated settings storage in the UWP platform through its SettingsService. Settings are added to the SettingsService.cs file template provided and can be set to use either Local or Roaming setting storage. An example of how settings are implemented using the SettingsService in this app is shown below for the Minimum Target Value.

```
        public double MinTarget // saves to roaming storage for syncing between
other UWP platform devices
        {
          get { return _helper.Read<double>(nameof(MinTarget), 4.0,
SettingsStrategies.Roam); } // gets MinTarget or returns default of 4.0
          set { _helper.Write(nameof(MinTarget), value, SettingsStrategies.Roam); }
        }
```

Figure 4.21 – MinTarget setting implementation

4.1.10 Reminders

When a low sugar reading is entered, the app will set a toast notification to pop 15 minutes from the time of entry. This is so the user can retest their sugars to check if their blood glucose levels are rising again after their chosen method of correction. When a person with diabetes suffers from a low blood sugar or 'hypo', they are required to take corrective action by taking 15-20g of fast acting carbohydrate. This may be in the form of a small glass of a sugary drink such as Coca Cola (non-diet version), at least 3 glucose tablets or a few sweets such as jelly babies. It is very important to test again after a hypo is experienced to ensure the chosen treatment has been effective and that the person's blood sugar is on the rise again. Examples of the toast notification are shown below with the PC version on the left and the mobile version on the right.

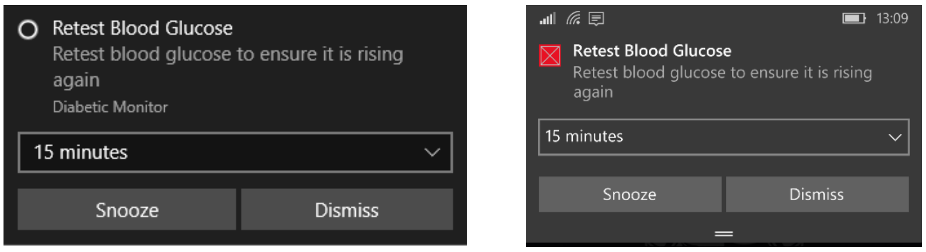


Figure 4.22 - Retest toast notifications (PC + Phone)

A listing of the code used for creating reminders is included in Appendix H.

4.1.11 Live Tile

A feature unique to the Windows platform is live tiles. These can provide continually updating data to the user from an application without it even running. In this app, if the live tile is pinned to the user's start screen or start menu, it will update with their last entered reading and latest HbA1c value. Examples of the live tile showing Last Reading and HbA1c values can be seen in the figure below:

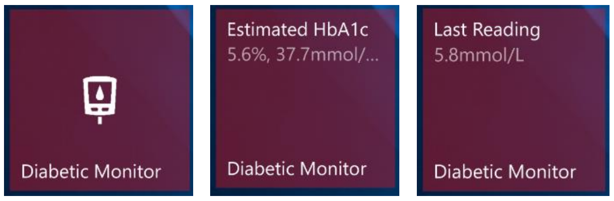


Figure 4.23 - Live Tile Example

The tile uses Tags to ensure that duplicate notifications of the HbA1c and Last Reading values do not occur and an expiry time of 4 hours is set for each notification. Both Medium (shown in Figure 4.23) and Wide tile sizes are supported for notifications. A code listing of the LiveTile.cs class that implements the live tile functionality is included in Appendix I.

4.1.12 Adaptive UI

The app is usable on both PC and Phone platforms, to achieve this, adaptive UI elements were used in the XAML code for each page. On most pages, Relative Panels were used so that the main content where related items are usually grouped with StackPanels could center itself properly when displaying on a larger screen size than that of a Windows Mobile 10 device. In the case of the Analysis page, adaptive triggers where used so that when the app reaches certain “Break Points” in screen resolution, the charts will grow or shrink to suit window size changes. An example of the Analysis page with an enlarged graph is shown below:

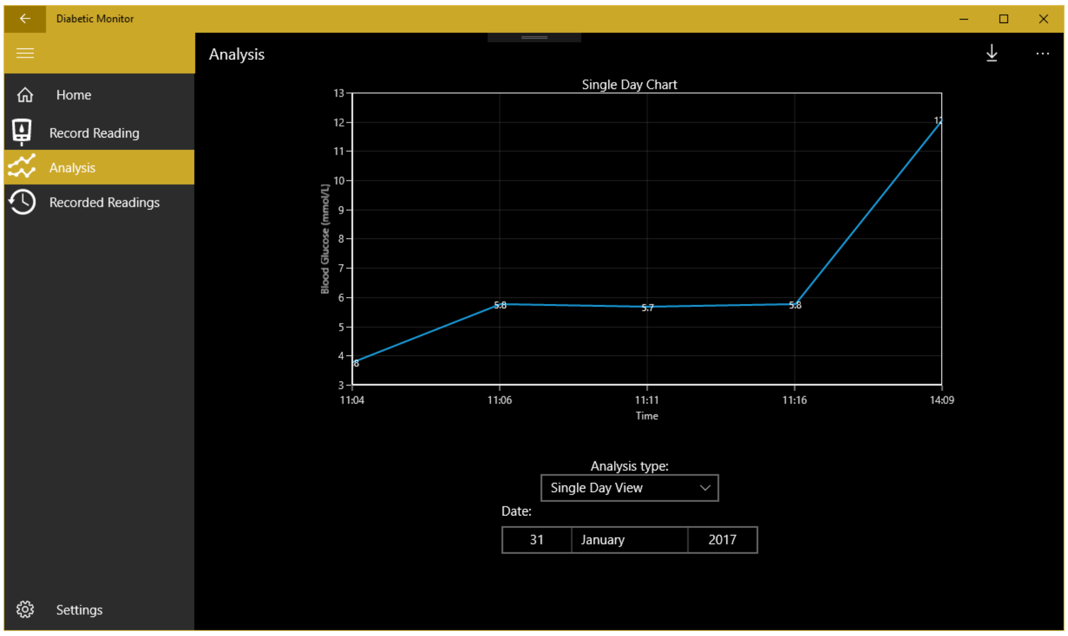


Figure 4.24 - Analysis Page running on PC with Adaptive UI

Examples of the other pages in their enlarged states are available in Appendix K and a copy of the XAML code for the adaptive triggers used on the Analysis page shown above is included in Appendix J.

4.2 Implementation of Web Service

The database has been implemented as detailed in the design and all of the originally planned features have been completed. The web service itself is a REST API and is implemented in PHP 7 using the Slim 3 microframework to create the endpoints that the client app uses to perform CRUD operations on the service. All requests and responses to and from the service are sent as JSON.

4.2.1 User Accounts

When a new user is created, an API key is generated by using the random_bytes method in PHP 7. This generates a string which is cryptographically secure (php.net, n.d. b). The code for this is shown below:

```
//generate apikey
$apikey = bin2hex(random_bytes(16));
```

Figure 4.25 - API Key Generation code

The API key is verified by methods that require authentication by checking for it in the request header and querying the database if it is present. An appropriate error is returned if the key is missing or invalid. If the check is successful, the user ID that the key corresponds to is retrieved for use in the method that required authentication. All methods with the exception of the POST methods for logging in and creating a user account require authentication via the API key.

4.2.2 Storing Data on the Service

Data such as reading data and info on a user's devices is stored on the web service and HTTP POST methods are provided for creating this data on the service. When a post request is received, the JSON request body is parsed to retrieve the variables before a method from the DBHandler class is called to create a record containing the parsed data in the MySQL database.

4.2.3 Updating Data on the service

Updating data on the service such as amending stored data on a reading is done much the same way as storing data on the service, with the exception that a PUT request is used instead. Using a PUT request has the advantage over a POST request in that if the request is sent multiple times due to network errors etc., it will have the same effect as if it was only sent once. If a POST method was used in such a situation, there is a risk that duplicate entries could be created.

4.2.4 Retrieving Data on the service

GET methods are provided for retrieving data from the service such as individual readings, a list of all readings a user has stored on the service and a list of all the devices a user has registered with their account.

4.2.5 Removing Data on the service

DELETE methods are provided to remove readings and devices from the service and also to remove a user account and all data associated with it completely from the service. The delete method for readings is different than the others, this is because as described in Chapter 3 on design, when a user has multiple devices linked to their account and a deletion occurs, potentially multiple entries are made to the pending deletes table in the database.

The reading will not be deleted on the service until all devices have removed it from their local databases.

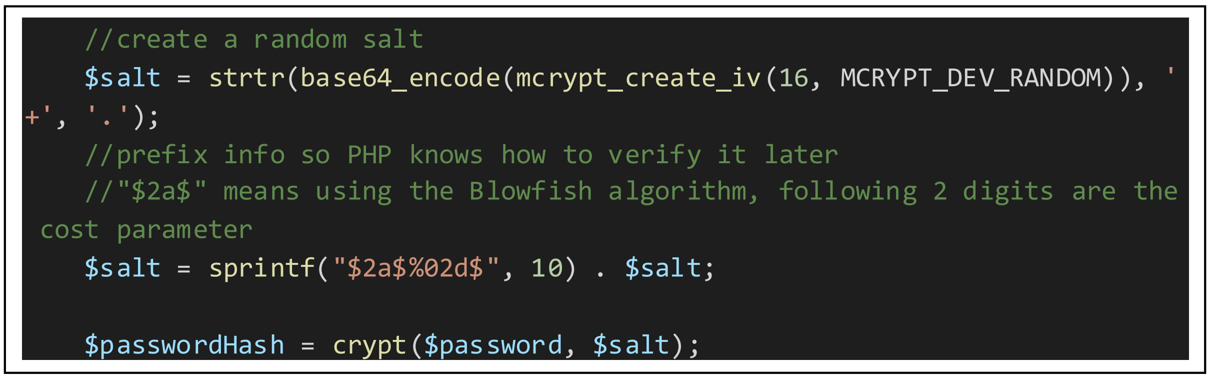
4.2.6 Syncing

A GET method is provided that retrieves a list of Sync tasks for the device that made the request, which as described earlier in the section on Recorded Readings, includes new or updated readings and IDs of any readings that need to be removed from the device.

4.2.7 Hosting and Security

The Web Service was designed to be as flexible as possible with regards to how it could be hosted. Both PHP and MySQL run on a variety of operating systems and are compatible with a variety of different web servers. In theory, this means that the web service part of this project will run on most web server platforms. This has been proven true as during development an XAMPP stack which contains Apache was used and for a working demonstration prototype that works with the client app, an IIS based Web App hosting platform on Microsoft's Azure cloud based service was used and is available online at https://diabeticmonitor.azurewebsites.net. Hosting on Azure is free and also has the added benefit for the application of supplying an existing SSL certificate meaning HTTPS can be used to enable secure connections between the client app and web service without the purchase and creation of specific certificates.

Following guidance provided on alias.io (alias.io, 2010), passwords are stored securely in the database by a process of hashing them against a random salt value using a cryptographically strong hashing algorithm. In this case the crypt function in PHP is used. The code that performs these steps is shown below:



```
    //create a random salt
    $salt = strtr(base64_encode(mcrypt_create_iv(16, MCRYPT_DEV_RANDOM)), '
+', '.');
    //prefix info so PHP knows how to verify it later
    //"$2a$" means using the Blowfish algorithm, following 2 digits are the
 cost parameter
    $salt = sprintf("$2a$%02d$", 10) . $salt;

    $passwordHash = crypt($password, $salt);
```

Figure 4.26 - Password hashing code

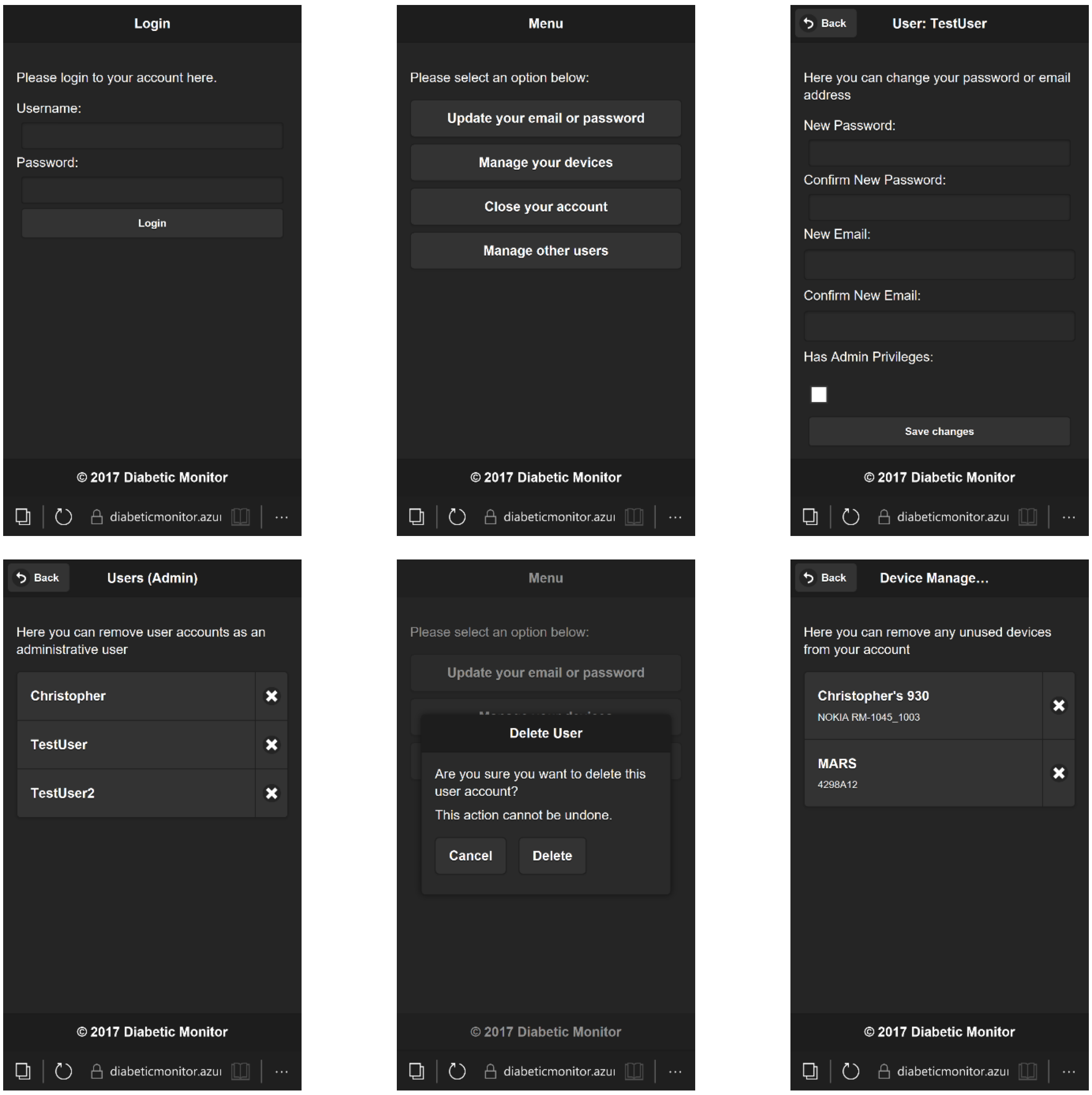
4.2.8 Management Page

To fulfil the requirements specified for the management of the web service, a very basic JavaScript page was created. This page takes advantage of jQuery Mobile and the responsive UI elements this provides to create an interface that works in both a mobile and PC browser.

The main purpose of the page is to provide very basic and easy access to administer the service without dealing with the backend database implementation directly. In the database, there is a column in the user table that specifies if a user has administrative privileges. This is used to determine whether the user can access the functions on the page that allow them to change another user's password or even remove a user account other than their own.

Other functionality provided by this page is a basic function to allow the user to view all of the devices that they have currently linked to their account and remove any that they no longer have or wish to use with the service. Users can also choose to update their password or email address. The facility to reset a lost password is not implemented as this is a prototype, but this function would likely be available in a production version of the application.

Some screenshots of the management page running in Microsoft Edge on Windows 10 Mobile can be seen below:


Login

Please login to your account here.

Username:

Password:

Login

© 2017 Diabetic Monitor

diabeticmonitor.azu

Menu

Please select an option below:

Update your email or password

Manage your devices

Close your account

Manage other users

© 2017 Diabetic Monitor

diabeticmonitor.azu

Back

User: TestUser

Here you can change your password or email address

New Password:

Confirm New Password:

New Email:

Confirm New Email:

Has Admin Privileges:

Save changes

© 2017 Diabetic Monitor

diabeticmonitor.azu

Back

Users (Admin)

Here you can remove user accounts as an administrative user

Christopher

TestUser

TestUser2

© 2017 Diabetic Monitor

diabeticmonitor.azu

Menu

Please select an option below:

Update your email or password

Delete User

Are you sure you want to delete this user account?

This action cannot be undone.

Cancel

Delete

© 2017 Diabetic Monitor

diabeticmonitor.azu

Back

Device Manage...

Here you can remove any unused devices from your account

Christopher's 930

NOKIA RM-1045_1003

MARS

4298A12

© 2017 Diabetic Monitor

diabeticmonitor.azu

Figure 4.27 – Management Page Screenshots

Chapter 5. Testing

5.1 Client Application

5.1.1 Scenario based testing

To test the application under the circumstances of normal usage, a set of scenarios were created with expected results and the actual results from the testing recorded. The results of these tests for each part of the application are shown below including details of how any issues were overcome:

5.1.1.1 First Time Setup

Test Undertaken	Expected Result	Actual Result
Select "Use Without Online Features" option	App moves to the main screen and can be used normally	App moved to the main screen and could be used normally
Select "Create New Account" option and provide user and device information	A new account is created, the current device is registered to that account after which the app can be used normally	New account created and device was registered correctly. The app could then be used normally
Select "Login to Existing Account" option and enter existing login information	Login is successful and any existing readings are downloaded so they can be used in the app	Login was successful and existing readings stored on the service were downloaded successfully
Select "Login to Existing Account" option and enter incorrect login information	Appropriate error message displayed	Application crashed*
Select "Create New Account" option and provide an invalid email address	Appropriate error message should be displayed	Appropriate error message was displayed
Select "Create New Account" and enter an existing username	Appropriate error message should be displayed	Appropriate error message was displayed

Figure 5.1 - First Time Setup Test Data

* This issue was fixed by updating the verifyCredentials method on the web service to return a response in the usual {success: true/false, message: "message"} format when invalid credentials were supplied. The application crashed because this method would return false on error while the JSON.Net library was looking for a success variable in the returned object and would throw an exception when parsing the response containing only "false".

5.1.1.2 MainScreen

Test Undertaken	Expected Result	Actual Result
A number of readings added	HbA1c value is displayed as text and visually on the speedometer control	HbA1c value was displayed as text and on the speedometer control
Reading of value 5.6mmol/L added	Value of 5.6mmol/L is displayed in green as it is on target	5.6mmol/L was shown as the last reading in green
Reading of value 11.4mmol/L added	Value of 11.4mmol/L is displayed in red as it is off target	11.4mmol/L was shown as the last reading in red
Readings for the last and current weeks added	Averages, overall change in averages and feedback for all readings and at meal times is available to the user	Averages complete with overall changes and feedback was produced

Figure 5.2 - Main Screen Test Data

5.1.1.3 Record Reading

Test Undertaken	Expected Result	Actual Result
Enter text or invalid values in text boxes for blood glucose and insulin dose then attempt to save	Appropriate error message displayed	Appropriate error message displayed
Enter valid reading and insulin dose data without a reason	Reading added successfully	Reading was added successfully
Clear all values	Text boxes empty and date/time values reset to system date and time	Text boxes were cleared and date/time reset to system values
Add a valid reading	Reading is added successfully and uploaded to the web service	Reading added and uploaded to web service

Figure 5.3 - Record Reading Test Data

5.1.1.4 Analysis

Test Undertaken	Expected Result	Actual Result
Single Day View chart requested for date 22/02/17	Line graph shown with all blood glucose values plotted against the time they were taken	Line graph produced with each blood glucose value plotted against the time it was taken
7,30,90 days average chart requested	Bar chart shown with each average correctly displayed	Bar chart produced with each average displayed as expected
Custom average chart requested from 22/02/17 to	Bar chart shown with averages for 7,30,90 days	Bar chart produced with correct averages for 7,30,90

10/03/17	and 22/02/17 to 10/03/17	days and 22/02/17 to 10/03/17
On/Off target chart requested	Pie chart shown displaying how many readings were on and off target in total out of all readings entered	Pie chart produced displaying correctly how many readings were on or off target in total out of all readings entered.
Export an example of each of the graphs produced in the above tests	Images of the graphs produced are saved to the device storage	Images of the graphs produced were saved correctly in the pictures folder of the device

Figure 5.4 - Analysis Test Data

5.1.1.5 Recorded Readings

Test Undertaken	Expected Result	Actual Result
Filter readings by each pre-set range and from 1st Feb to 28th Feb	Readings should be filtered and displayed correctly in the list box	Readings were filtered correctly and displayed in the list box
Sync readings	Any readings added or updated on another device should be downloaded and added or used to update existing data	Readings that were added and updated on another test device were downloaded and dealt with properly
Edit Reading	Dialog box for editing should appear and allow editing. The edited reading should then be updated in both the SQLite database and on the web service	Dialog box appeared and reading was updated on the web service on completion of the edit
Edit Reading that has already been edited on another device	Dialog box should appear asking whether the local version or server version should be kept.	Dialog box appeared explaining that the reading was last updated on the server after the local modified date, giving option to keep the server or local copy
Edit Reading that has been removed by another device	Error stating that reading could not be edited	Reading is edited successfully, but is removed when device next syncs with the service
Delete Reading	Reading should be removed from the SQLite database and a request made to the web service for its deletion	Reading removed from SQLite database and request for deletion made to web service
Delete Reading that has been removed already on	Reading should be removed from the SQLite database	Reading removed from SQLite database*

another device		
Export Readings from all pre-set ranges and 1st Feb to 28th Feb	CSV files for each exported range should be produced and saved onto the device	CSV files were produced for each range and saved onto the devices storage

Figure 5.5 - Recorded Readings Test Data

* - See known issue 4 in Section 5.3

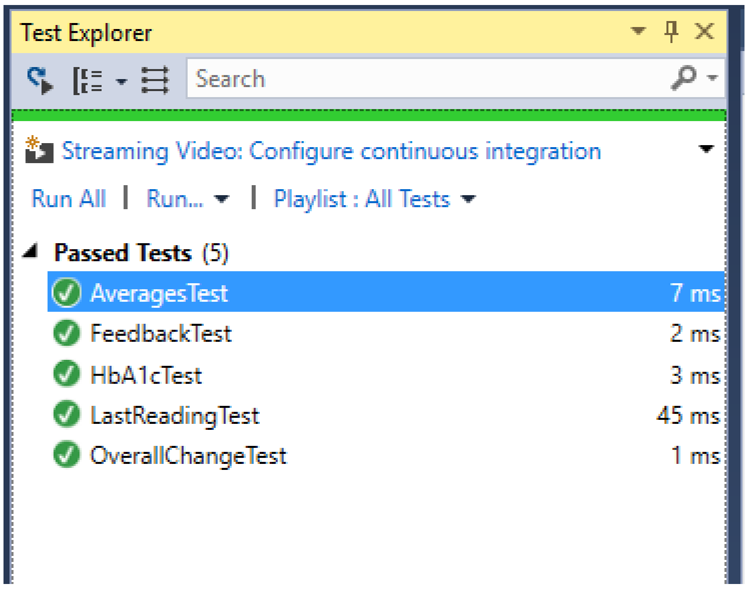
5.1.1.6 Settings

Test Undertaken	Expected Result	Actual Result
Change Minimum Target	Minimum Target is updated in roaming storage and syncs across devices	Minimum Target was updated and synced across devices
Change Maximum Target	Maximum Target is updated in roaming storage and syncs across devices	Maximum Target was updated and synced across devices
Clear Local Data	All local app data is removed and the app should return to the Setup screen	All local app data was removed and the app returned to the Setup screen

Figure 5.6 - Settings Test Data

5.1.2 Unit testing

As the design and implementation of the app closely follows the MVVM pattern where possible, this means that the components are loosely coupled. As a result, parts of the application that are not easily tested under a scenario based test such as the detailed analysis functions of this app can be tested in isolation from the rest of the components. This allows them to be fed appropriate test data in an attempt to produce the desired results. For example, to test the class that deals with producing the HbA1c calculations and averages etc. for the main screen, unit tests were created to supply the appropriate functions with different sets of data. Each dataset was designed to produce different results that tested every possible valid outcome. Results of these unit tests are shown below:



Passed Tests (5)

AveragesTest	7 ms
FeedbackTest	2 ms
HbA1cTest	3 ms
LastReadingTest	45 ms
OverallChangeTest	1 ms

Figure 5.7 - Unit Test Results

Although the above screenshot shows the unit tests passing, initially the AveragesTest and OverallChangeTest tests failed. This was due to a rounding error when using the Math.Round method which skewed the results slightly. To remedy this, a third parameter (`MidpointRounding.AwayFromZero`) was added to each call of Math.Round. Code listings of the class that supplies the datasets and the test class itself are included in Appendix L. Details of how the unit testing features in Visual Studio were used to perform the unit tests are included as Appendix M.

5.2 Web Service

For testing purposes, test requests were sent from Postman which is an application that provides an easy to use interface to assist in API development and testing (getpostman.com, n.d.). Test results are shown below:

Test undertaken	Expected Response	Actual Response

Create a user
POST Request body:
{
"username": "TestUser",
"password": "Passw0rd",
"email":
"testuser@example.com"
}
	{
	"success": true,
	"message": "User created successfully!",
	"userID": (assigned id here),
	"apiKey": (assigned API Key here)
	}
		{
		"success": true,
		"message": "User created successfully!",
		"userID": "31",
		"apiKey":
		"c3838acbf5fe4844d1027bc8bacb8117"
		}

Create a user with invalid email
POST Request body:
{
"username": "invalid",
"password": "invalid",
"email": "notanemail"
}
	{
	"success": false,
	"message": "The email address provided was invalid"
	}
		{
		"success": false,
		"message": "The email address provided was invalid"
		}

Get list of all users (not admin)	{
	"success": false,
	"message": "An account with admin privileges is required to perform this operation"
	}
		{
		"success": false,
		"message": "An account with admin privileges is required to perform this operation"
		}

Update user details
PUT Request body:
{
"password": "Changed",
"email":
"testuser@testing.com",
"isAdmin": 1
}
	{
	"success": true,
	"message": "User updated successfully!"
	}
		{
		"success": true,
		"message": "User updated successfully!"
		}

Get list of all users (admin)	{
	"success": true,
	"users": [
	{
	"id": 31,
	"username": TestUser,
	"isAdmin": 1
	}
	]
	}
		{
		"success": true,
		"users": [
		{
		"id": "31",
		"username": "TestUser"
		"isAdmin": 1
		}
		]
		}

Login to service
POST Request body:
{
"username": "TestUser",
"password": "Changed"
}
	{
	"success": true,
	"id": 31,
	"email":
	"testuser@testing.com",
	"apiKey": (API Key value),
	"isAdmin": 1
	}
		{
		"success": true,
		"id": "31",
		"email": "testuser@testing.com",
		"apiKey":
		"c3838acbf5fe4844d1027bc8bacb8117",
		"isAdmin": "1"
		}

Login to service (wrong password)
POST Request body:
{
“username”: “TestUser”,
“password”: “Wrong”
}
	{
	“success”: false,
	“message”: “Invalid password”
	}
		{
		"success": false,
		"message": "Invalid password"
		}

Add Device
POST Request body:
{
“nickname”: “TestUserDevice”
“deviceModel”: “TestDevice”
}
	{
	“success”: true,
	“message”: “Device added successfully!”,
	“deviceID”: (assigned ID)
	}
		{
		"success": true,
		"message": "Device added successfully!",
		"deviceID": "10"
		}

Add Reading
POST Request Body:
{
“dateTime”: “2017-03-10 12:16:35”,
“readingValue”: 9.4,
“insulinValue”: 6.0,
“reason”: “Lunch”
}
	{
	“success”: true,
	“message”: “Reading added successfully!”,
	“readingID”: (Assigned ID)
	}
		{
		"success": true,
		"message": "Reading added successfully!",
		"readingID": "86"
		}

Update Reading
PUT Request body:
{
“readingValue”: 8.4,
“insulinValue”: “6.0”,
“reason”: “Lunch”
“modifiedDateTime”: “2017-03-10 12:16:35”
}
	{
	“success”: true,
	“message”: “Reading updated successfully”
	“modifiedDateTime”: “2017-03-10 12:20:36”
	}
		{
		"success": true,
		"message": "Reading updated successfully!",
		"modifiedDateTime": "2017-03-10 12:20:36"
		}

Get single reading	{
	“success”: true,
	“reading”: {
	“id”: “86”,
	“dateTime”: “2017-03-10 12:16:35”,
	“readingValue”: “8.4”,
	“insulinValue”: “6.0”,
	“reason”: “Lunch”,
	“modifiedDateTime”: “2017-03-10 12:20:36”
	}
	}
		{
		"success": true,
		"reading": {
		"id": "86",
		"dateTime": "2017-03-10 12:16:35",
		"readingValue": "8.4",
		"insulinValue": "6.0",
		"reason": "Lunch",
		"modifiedDateTime": "2017-03-10 12:20:36"
		}
		}

Get all readings	{ “success”: true, “readings”: [{ “id”: “86”, “dateTime”: “2017-03-10 12:16:35”,	{ "success": true, "readings": [{ "id": "86", "dateTime": "2017-03-10 12:16:35", "readingValue": "8.4",

	"readingValue": "8.4", "insulinValue": "6.0", "reason": "Lunch", "modifiedDateTime": "2017-03-10 12:20:36" }] }	"insulinValue": "6.0", "reason": "Lunch", "modifiedDateTime": "2017-03-10 12:20:36" }] }

Delete a reading	{
	"success": true,
	"message": "Reading deleted successfully"
	}
		{
		"success": true,
		"message": "Reading deleted successfully!"
		}

Retrieve a list of devices for the user	{
	"success": true,
	"devices": [
	{
	"id": "10",
	"nickname": "TestUserDevice"
	"deviceModel": "TestDevice"
	}
	]
	}
		{
		"success": true,
		"devices": [
		{
		"id": "10",
		"nickname": "TestUserDevice",
		"deviceModel": "TestDevice"
		}
		]
		}

Sync Data	{
	"newOrUpdated": []
	"deleted": []
	}
		{
		"newOrUpdated": [],
		"deleted": []
		}

Try to edit or delete a reading without permission	{
	"success": false,
	"message": "No permissions for this reading or it does not exist"
	}
		{
		"success": false,
		"message": "No permissions for this reading or it does not exist"
		}

Remove Device	{
	"success": true,
	"message": "Device deleted successfully"
	}
		{
		"success": true,
		"message": "Device deleted successfully!"
		}

Remove User (other account, not admin)	{
	"success": false,
	"message": "Only an Admin can close another user account"
	}
		{
		"success": false,
		"message": "Only an Admin can close another user account"
		}

Remove User (own account)	{
	"success": true,
	"message": "User deleted successfully"
	}
		{
		"success": true,
		"message": "User deleted successfully!"
		}

Figure 5.8 - Web Service Test Data

5.3 Known Issues

It should be noted that the following issues are known about at present with the current implementation:

1. On the Recorded Readings page, the CommandBar that contains the controls at the bottom of the screen will cover the Settings button on the hamburger menu when the app is not running in the narrow width on the PC.
2. When the app is reset to factory settings, as the targets set by the user are reset as well, any other device the user is using will see the effect of this sync through the roaming settings linked to the user's Microsoft Account.
3. When editing a reading, if a validation error occurs, the editing dialog box will close after the error message has been displayed.
4. Duplicate pending deletes for any other devices are added to the database when a reading has been deleted on a device that has not synced and does not know a delete request has already been made for the affected reading.

Chapter 6. Conclusion and Recommendations

This section provides an overview of the project and discusses how the app produced compares to and is positioned with the competition already available as well as identifying possible future improvements.

6.1 Overview

This project set out to investigate how modern computer technology could assist with effective diabetes management and develop a prototype application that could extend and complement existing solutions. These goals were achieved through research into various existing solutions and development of an app using technologies that include C#, SQLite, PHP and MySQL. The app is powerful, functional and designed with multiplatform support in mind. It has the potential to be extended in a number of ways, especially around integration with other devices and solutions to aid in effective diabetes management.

6.2 Comparison to existing apps

The idea of a diabetic management app for a mobile device was not new, and several applications already exist for iOS, Android and the older Windows RT/Windows Phone platforms. Most of these apps offer similar functionality in that they allow data to be analysed by creating graphs and generating averages from recorded data. Some also go further than the functionality of this app by including facilities for insulin dose calculations and carbohydrate counting diaries. The app developed as part of this project sits somewhere in the middle between the top and low end apps already available across the three main mobile platforms (iOS, Android and Windows). Apps at the top end such as mySugr, which is the "Most popular diabetes diary app in the world based on five-star reviews and ratings" (play.google.com, 2017) offer similar, more advanced features such as data sync and backup with an online service and HbA1c estimates. Other apps at the lower end tend to be basic diary applications that only record information locally and provide basic feedback in the form of averages.

The biggest differentiation between most of the apps already available and this app, is the fact that this app has been designed from the ground up to support both PC and Mobile devices on the Universal Windows Platform whereas most apps are designed solely for mobile devices and pay little attention to the Windows platform if at all. Advantages of the UWP platform include the possibility of future support for the Xbox platform and Microsoft HoloLens VR headset. On these platforms, although they are unlikely to be the main device the app is used on, important notifications about a user's diabetes management could be displayed during gameplay or other uses of these devices. The app is also ready to integrate with NFC compatible glucose monitors which is not something many of the existing apps that exist at the moment support.

6.3 Potential improvements and future functionality

As mentioned in the above subsection on existing applications, there are several areas which the app does not currently cover but could in the future. These along with potential improvements on the existing implementation of the app will be discussed in the following subsections.

6.3.1 Potential improvements to the app

There are a number of ways in which the features currently implemented in the app produced could be improved for both the client itself and the accompanying web service.

The detailed analysis function on the main screen could be expanded so that rather than just displaying data for the current and previous weeks, it contains a log of all of these calculations that could be looked back upon in the future, perhaps in the form of visual representation such as a line graph similar to the single day view graph implemented in the analysis page of the app.

A background service for syncing with the web service component could also be added so that any data created or modified on another device is already synced and waiting for the user on return to the app on another device.

Since data can be exported to a CSV file, a way to re import this data from the CSV file if required at a later date may be a useful feature to consider in a future version of the app. This would be of significant benefit to users who choose not to use the cloud backup service provided but still wish to be able to backup and restore their data if needed.

Although the app provides the facility for the user to set a target range, this could be expanded on so that rather than just allowing for a general target to be set, individual targets for different times such as meals and bed time could be set. This could be useful for a user in a situation where they may wish to have a slightly different target such as a slightly higher than usual blood glucose level before bed in an attempt to try and combat a potential hypo during the night.

For the web service, a more advanced security and login system that provides integration with a user’s Microsoft, Google or Facebook account could be provided by something such as the OAuth authentication library rather than the very simple custom API Key authentication method currently in use by the app. This would increase both security of the user’s data and convenience for the user as they would not have to create another proprietary account just to manage their data stored by the app.

6.3.2 Carb Diary and Insulin Dose Calculator

A very common practise among people with diabetes is carbohydrate counting. Carbohydrate counting is a method that can be used by a person to get a clearer picture of how carbohydrates affect their blood glucose levels and insulin requirements (diabetes.co.uk, n.d. b). A section could be added to the app that allows at least an estimate of any carbohydrate contained in food to be recorded alongside blood glucose values. It can be hard to get an exact carbohydrate value due to some products not listing carbohydrate content on the label or where it is not feasible to get even an estimated value such as when eating out at a restaurant.

Expanding on the carb diary functionality, the information gathered could be used to calculate estimated insulin dose values. Functions could be added to the app to work out using a user's carb to insulin ratio, what their insulin doses should roughly be based on the amount of carbohydrate in the food they wish to eat. Carb to insulin ratios are usually worked out with assistance from a diabetic professional and a user may have different ratios for different meal times. The app could allow for the storing of potentially multiple carb to insulin ratios to assist in insulin dose calculations.

6.3.3 Gamification

Keeping people engaged with and interested in their diabetes treatment can be a major issue that places a massive burden on the NHS in the UK when a patient's management is poor. An estimated 80% of the NHS budget for diabetes is spent on dealing with complications (diabetes.org.uk, b), many of which could be avoided through improved management techniques. This is particularly a problem amongst younger patients with diabetes and adding gamification into monitoring applications is one way to try and keep users coming back to the app. Setting targets to be met in an Xbox Live style achievements system could provide motivation and encouragement for users to keep their control on track more often.

6.3.4 Integration with other devices

A brief look at integration with other devices was looked at in this project through the NFC prototype which used preprogramed NDEF NFC Tags to simulate a scenario where a user could use a glucose monitor such as the FreeStyle Libre which continuously monitors blood glucose 24 hours a day (freestylelibre.co.uk, n.d.) and allows for retrieval via NFC on its own proprietary device or with a compatible mobile phone.

Integration with other devices could be taken further by adding support for solutions such as Nightscout to the app. Nightscout is an open source solution that allows for remote monitoring of a user's blood glucose in real time, using data from their existing compatible continuous glucose monitor (CGM) (nightscout.info, n.d.). Data can be retrieved either wirelessly from a CGM, or directly via a USB cable to a compatible mobile device. Data from Nightscout is already used in existing solutions available to people using artificial pancreas systems based on the OpenAPS project. OpenAPS is another open source effort, and aims to use already approved diabetes medical devices such as continuous glucose monitors and insulin pumps combined with some DIY to build a safe and effective artificial pancreas system (APS) (openaps.org, n.d.). One of the driving factors behind the project is that it allows people to get their hands on an APS system cheaply and much quicker than they would otherwise by waiting for a pre-approved APS device that suits them to become available, using their already familiar devices where possible.

For people who may have a compatible continuous glucose monitor that can be used with the Nightscout project, integration with the service in the app could provide huge benefits for the user. A great example of how this could improve existing solutions would be in monitoring glucose levels overnight. If the app had access to a user's continuous glucose monitoring data, when running on a phone it could start ringing to alert the user to any dangerously low or high glucose levels during the night. This would remove a lot of worry from people who perhaps fear their blood glucose levels falling during the night and removes the need for purposely high glucose levels before bed as a possible way of trying to combat the issue. Notifications such as this from a user’s phone would also be useful in situations where a user suffers from reduced hypo awareness where they can find it difficult to realise when they are experiencing a hypo.

6.3.5 Support for other localisations

Currently the app is tailored towards British users and uses the UK date format and units of measurement for blood glucose. Testing has not been performed with the app on a device that is set to a country such as the United States who use a different date format and mg/dL (milligrams per decilitre) rather than mmol/L (millimoles per litre) as their preferred unit of blood glucose measurement. Future versions of the app could be improved upon to include different default units depending on the system locale, with the option to swap between supported units of measurement in the settings menu.

6.4 Future development work

While the Universal Windows Platform is a great platform to develop for and offers many advantages for apps that are designed to run across different device types, its market share particularly in the mobile device sector leaves a lot to be desired. For this reason, it would be very likely that porting the app to competing platforms such as iOS and Android alongside the UWP implementation would be recommended for a production version of the app. As pointed out at various points during this report, many parts of the app use cross platform tools and components which make this a much easier task than if future portability hadn’t been considered during the design and development process.

Chapter 7. Critical Self Appraisal

I chose this honours project topic as it was an area in which I had recently taken an interest in, and although I have Type 1 Diabetes myself and have a good understanding of it, I am not a user of any advanced computerised management solution and rely on my blood glucose monitor and over 6 years of experience to manage my diabetes. My biggest reason for choosing this topic besides the fact that it interested me greatly, was that it offered me a chance to write an app in C# for the Universal Windows Platform, which is my programming language and platform of choice. One of the goals I set for myself before starting the project was to enhance my knowledge and skills in C# programming.

7.1 Research

When conducting the research for the project, although I knew a fair bit about most of the areas I needed to research already, I still managed to learn more about each area. While researching HbA1c values in particular, I learned much more about what this value actually is and means along with how it helps in relation to diabetes management, which was a pleasant surprise. What I most enjoyed about the research I did was learning more about the Artificial Pancreas Systems and while they did not feature prominently in the project, they are definitely something that I feel is worth keeping an eye on in the future of diabetes treatment.

7.2 Development Work

Building the UWP app itself was probably my favourite part of the project and I learned a lot more about C# programming conventions and Model View ViewModel (MVVM) design in the process. MVVM design was not something I expected to use when I first set out to do the project and is something that I went from knowing very little about beforehand to properly understanding the benefits and why it is used in most apps that use XAML for their UI by the end of the project. The app also gave me a much bigger canvas with which to apply everything I'd learned about Object Oriented Programming over my time at University and greatly increased my understanding of the concepts involved. I think this was easily the most valuable part of the project for me personally and it will hopefully help in my future career. It certainly meant that I achieved my goal of enhancing my skills and knowledge in C# programming even more than I had hoped for at the start of the project.

There are various parts of the implementation that I would have liked to improve upon or take further, but did not manage to due to time constraints. For example, the syncing of data with the web service is something I feel I could have maybe implemented better if I had allocated more time to it. As can be seen from the testing chapter, there are situations where the syncing of data appears to work fine to the user, but leaves undesired results behind the scenes. I did however, manage to get all the main features I set out to include finished to a good and acceptable standard for a working prototype and the app is easily my best work out of all the programming projects I have done so far.

While the web service is not as secure or production ready as I maybe would like, as a prototype it meets all the requirements I envisioned for it. I was maybe a bit too ambitious when setting out on creating a management page for it to perform administrative tasks and user management, as this was not as relevant to the purpose of the project as other areas, but this provided me with an opportunity to showcase some more of the skills I have gained throughout my time at University with developing HTML and JavaScript applications. I could have chosen to use a service such as Firebase to build the backend web service rather than taking the DIY route with PHP and the slim microframework. However, I feel that I made the right choice in building a custom service myself. This is because it provided me with more experience in building a custom web API to suit my own needs. I also improved my knowledge and understanding of programming in PHP. Which was a bonus since the previous web API work I had done used Microsoft WCF based web services. As with the UWP app, this is also something I feel is one of the better bits of programming work I have done so far and would definitely consider taking forward in my own time now that the project is complete to enhance my skills even further.

One of the things that held me back in terms of development was the computing power I had available to properly run the Visual Studio IDE. While my laptop ran it very well for the most part, it showed its age when running the XAML designer while implementing the user interface for the UWP app. It performed sluggishly and would often hang. This resulted in a lot of time spent waiting for the designer to unfreeze and was very frustrating when all I wanted to do was make small adjustments. Near the end of the project, Visual Studio 2017 launched celebrating Visual Studio's 20th year. This brought a few improvements that would have benefitted the project if it was available earlier. One of the new features that I would have benefitted from during the development of the app is the live editing of XAML code to adjust the UI of a UWP app while it is running. I was able to make some use of this towards the end of the development of the app to make some small changes to the UI to tidy it up and was very impressed with how it works. Other new features that would have been beneficial earlier on in the project would have included the faster performance and enhanced unit and regression testing features.

7.3 Testing

When planning how to properly test the project, I was initially set on just scenario testing. This was because I didn't have any knowledge or experience in other methods of testing from my time at university or even work I'd done in my own time. I did have a brief look into unit testing while still developing the app, but deemed that it was perhaps too much of a hassle to look into at the time. When looking back over work I had done later, I remembered that one of the benefits to the MVVM pattern was that it could make unit testing a lot easier. This led to my decision to revisit it. This was probably the best decision I made during the testing stage of the project as it turned out to be a great experience and ended up fitting in with the testing for the detailed analysis functions perfectly. Unfortunately, due to conflicts which I didn't have time to properly look into, I could not test the ViewModel classes with unit testing as I had initially hoped. However, I did manage to successfully use unit testing for the DetailedAnalysis class as detailed in the Testing chapter and Appendix M.

This proved to be a very valuable learning experience and also showed me that testing not only helps to find out if the app performs correctly, but can provide a greater understanding of how some parts of an application or method function. This is because when testing the app, I found an error with the way I was using the Math.Round method to round up average blood glucose values which required me to have a better understanding of the method to work out a solution to the problem.

When testing the other parts of the application with scenario testing, I made up some basic scenarios to test the various functions of the app. There is very little in the way of extreme test data, so with more time I feel that I could have produced a more thorough testing regime. It would also have been nice to have got other people to test the app, but finding other people with diabetes who also use a Windows Phone would likely have been a problem.

7.4 Project Management

In terms of project management, I feel that I performed well and kept to the timescales that I had set myself. If I ever slipped behind on anything due to factors such as personal commitments, or commitments for other modules, I was able to make up for that time and catch up to where I wanted to be quite quickly.

7.5 Conclusion

I am very pleased with the outcome of the project as a whole and feel that it has given me an excellent opportunity to apply what I have learnt over the last 4 years spent at university as well as expanding upon the skills developed in areas I have covered in my own time.

The project meets the criteria specified in the specification form by providing a brief review of existing solutions available to diabetic patients as well as identifying where the app can improve these solutions by complimenting them in various ways. The app also provides both basic short term analysis of blood glucose and more in-depth longer term analysis through features such as the estimated HbA1c calculations and weekly data.

One thing I would have liked to include with the project was some kind of formal feedback from my diabetes care team about the app. I had discussed it with them during an appointment where I received positive verbal feedback on the idea of the app and what it could do, but unfortunately due to time constraints was not able to get any official feedback to include in the report.

Overall, the project has successfully helped to identify ways in which modern computer technology can assist in diabetes management. This is shown both through the research that has been carried out and the application produced. The project has identified a number of exciting areas for future enhancement and development for the App and solutions in this space.

References

alias.io. (2010). alias.io — How to store passwords safely with PHP and MySQL. [online] Available at: https://alias.io/2010/01/store-passwords-safely-with-php-and-mysql/ [Accessed 15 January 2017]

channel9.msdn.com. (2016). Faster than a speeding XAML, your custom speedometer in UWP | Demo of the Day | Channel 9. [online] Available at: https://channel9.msdn.com/Shows/demooftheday/speedingxaml [Accessed 28 January 2017]

chilkatsoft.com. (n.d.). Csv C# Reference Documentation. [online] Available at: https://www.chilkatsoft.com/refdoc/csCsvRef.html [Accessed 23 February 2017]

dev.mysql.com. (n.d. a). MySQL :: MySQL 5.7 Reference Manual :: 1.3.1 What is MySQL? [online] Available at: http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html [Accessed 13 October 2016]

dev.mysql.com. (n.d. b). MySQL :: MySQL 5.7 Reference Manual :: 14.1.18 CREATE TABLE Syntax [online] Available at: http://dev.mysql.com/doc/refman/5.7/en/create-table.html [Accessed 18 December 2016]

diabetes.co.uk. (n.d. a). What is HbA1c?, Units, Conversion, Testing & Control. [online] Available at: http://www.diabetes.co.uk/what-is-hba1c.html [Accessed 18 October 2016]

diabetes.co.uk. (n.d. b). Carb Counting & How to Count Carbs [online] Available at: http://www.diabetes.co.uk/diet/carbohydrate-counting.html [Accessed 15 March 2017]

diabetes.org.uk. (n.d. a). Research Spotlight – The Artificial Pancreas. [online] Available at: https://www.diabetes.org.uk/Research/Research-round-up/Research-spotlight/Research-spotlight-the-artificial-pancreas/ [Accessed 20 November 2016]

diabetes.org.uk. (n.d. b). Diabetes UK Cost of Diabetes Report.pdf. [online] Available at: https://www.diabetes.org.uk/Documents/Diabetes%20UK%20Cost%20of%20Diabetes%20Report.pdf [Accessed 9 March 2017]

freestylelibre.co.uk (n.d.). FreeStyle Libre | Blood Glucose Monitoring System. [online] Available at: https://www.freestylelibre.co.uk/libre/ [Accessed 4 October 2016]

getpostman.com. (n.d.). Postman | Supercharge your API workflow [online] Available at: https://www.getpostman.com/ [Accessed 16 December 2016]

github.com. (2016 a). Home · Windows-XAML/Template10 Wiki · GitHub. [online] Available at: https://github.com/Windows-XAML/Template10/wiki [Accessed 10 October 2016]

github.com. (2016 b). GitHub/ndef-nfc: NDEF Library for Proximity APIs / NFC. [online] Available at: https://github.com/andijakl/ndef-nfc [Accessed 16 January 2017] msdn.microsoft.com. (2012). The MVVM Pattern. [online] Available at: https://msdn.microsoft.com/en-us/library/hh848246.aspx [Accessed 25 October 2016]

msdn.microsoft.com. (2016). Intro to the Universal Windows Platform. [online] Available at: https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide [Accessed 10 Oct 2016]

netredande.com. (n.d.). A1C and Estimated Average Glucose (eAG). [online] Available at: http://nedretandre.com/Medical/a1c.php [Accessed 28 October 2016]

nightscout.info. (n.d.). The Nightscout Project | We Are Not Waiting. [online] Available at: http://www.nightscout.info/ [Accessed 15 March 2017]

openaps.org (n.d.). What is #OpenAPS? – OpenAPS.org. [online] Available at: https://openaps.org/what-is-openaps/ [Accessed 8 October 2016]

php.net. (n.d. a). PHP: What is PHP? [online] Available at: http://php.net/manual/en/intro-whatis.php [Accessed 11 October 2016]

php.net. (n.d. b). PHP: random_bytes. [online] Available at: http://php.net/manual/en/function.random-bytes.php [Accessed 16 November 2016]

play.google.com. (2017). mySugr: Diabetes logbook app – Android Apps on Google Play. [online] Available at: https://play.google.com/store/apps/details?id=com.mysugr.android.companion&hl=en [Accessed 10 March 2017]

slimframework.net. (n.d.). Documentation – Slim Framework [online] Available at: https://www.slimframework.com/docs/ [Accessed 29 October 2016]

sqlite.org. (n.d.). About SQLite. [online] Available at: https://sqlite.org/about.html [Accessed 11 October 2016]

Bibliography

alanfeekery.com. (2015). POSTing JSON data using HttpClient – Alan Feekery [online] Available at: https://alanfeekery.com/2015/04/13/posting-json-data-using-httpclient/ [Accessed 10 January 2017]

androidhive.info. (2014). How to create REST API for Android app using PHP, Slim and MySQL – Day 1/2 http://www.androidhive.info/2014/01/how-to-create-rest-api-for-android-app-using-php-slim-and-mysql-day-12-2/ [Accessed 3 December 2016]

arjunphp.com. (2016). Slim 3, Basic PDO CRUD [online] Available at: https://arjunphp.com/creating-restful-api-slim-framework/ [Accessed 18 December 2016]

blogs.msdn.microsoft.com. (2015 a). A Minimal MVVM UWP App – JohnShew's Blog [online] Available at: https://blogs.msdn.microsoft.com/johnshews_blog/2015/09/09/a-minimal-mvvm-uwp-app [Accessed 27 October 2016]

blogs.msdn.microsoft.com. (2015 b). Quickstart: Sending a local toast notification and handling activations (Windows 10) – Tiles and Toasts [online] Available at: https://blogs.msdn.microsoft.com/tiles_and_toasts/2015/07/08/quickstart-sending-a-local-toast-notification-and-handling-activations-from-it-windows-10/ [Accessed 1 March 2017]

blogs.msdn.microsoft.com. (2015 c). UWP: New features of Visual State Manager (part 1) – Canadian Developer Connection [online] Available at: https://blogs.msdn.microsoft.com/cdndevs/2015/06/26/uwp-new-features-of-visual-state-manager-part-1 [Accessed 25 February 2017]

blogs.msdn.microsoft.com. (2015 d). Quickstart: Sending a local tile notification in Windows 10 – Tiles and Toasts [online] Available at: https://blogs.msdn.microsoft.com/tiles_and_toasts/2015/10/05/quickstart-sending-a-local-tile-notification-in-windows-10/ [Accessed 5 March 2017]

blogs.u2u.be. (2015). Using SQLite on the Universal Windows Platform. [online] Available at: http://blogs.u2u.be/diederik/post/2015/09/08/Using-SQLite-on-the-Universal-Windows-Platform.aspx [Accessed 20 October 2016]

codeproject.com. (2010 a). Model-View-ViewModel (MVVM) Explained – CodeProject [online] Available at: https://www.codeproject.com/articles/100175/model-view-viewmodel-mvvm-explained [Accessed 26 October 2016]

codeproject.com. (2010 b). One Way, Two Way and One Time Bindings using Silverlight [online] Available at: https://www.codeproject.com/Articles/37250/One-Way-Two-Way-and-One-Time-Bindings-using-Silver [Accessed 12 February 2017]

codingcage.com. (2015). PHP PDO CRUD Tutorial using OOP with Bootstrap | Coding Cage [online] Available at: http://www.codingcage.com/2015/04/php-pdo-crud-tutorial-using-oop-with.html [Accessed 18 December 2016] c-sharpcorner.com. (2015 a). Binding Collection To ListView Control In UWP. [online] Available at: http://www.c-sharpcorner.com/UploadFile/5ef5aa/binding-collection-to-listview-control-in-uwp-explained/ [Accessed 22 October 2016]

c-sharpcorner.com. (2015 b). ComboBox Control For Windows 10 [online] Available at: http://www.c-sharpcorner.com/UploadFile/4e4e9d/combobox-control-for-windows-10/ [Accessed 21 October 2016]

depblog.weblogs.us. (2016). UWP – Close a Flyout MVVM way with UWP XamlBehaviors – Depechie's outing [online] Available at: http://depblog.weblogs.us/2016/01/31/uwp-close-a-flyout-mvvm-way-with-uwp-xamlbehaviors/ [Accessed 27 February 2017]

diabetesinscotland.org.uk. (2009). HbA1c_Lab_leaflet_0509.pdf [online] Available at: http://www.diabetesinscotland.org.uk/Publications/HbA1c_Lab_leaflet_0509.pdf [Accessed 4 January 2017]

docs.microsoft.com. (2014). Calling a Web API From a .NET Client (C#) | Microsoft Docs [online] Available at: https://www.asp.net/web-api/overview/advanced/calling-a-web-api-from-a-net-client [Accessed 12 January 2017]

docs.microsoft.com. (2017 a). Guidelines for password boxes | Microsoft Docs [online] Available at: https://docs.microsoft.com/en-us/windows/uwp/controls-and-patterns/password-box [Accessed 10 February 2017]

docs.microsoft.com. (2017 b). Save a file with a picker | Microsoft Docs [online] Available at: https://docs.microsoft.com/en-us/windows/uwp/files/quickstart-save-a-file-with-a-picker [Accessed 26 February 2017]

gallery.technet.microsoft.com. (2015). TechNet How to show message dialog in windows 10 [online] Available at: https://gallery.technet.microsoft.com/How-to-show-message-dialog-32d6c39e [Accessed 21 October 2016]

github.com. (n.d.). GitHub - Windows-XAML/Template10.Validation [online] Available at: https://github.com/Windows-XAML/Template10.Validation [Accessed 15 January 2017]

mcnextpost.com. (2015). #UWPXAML – Compiled Binding – Binding to events | Le Post de MCNEXT [online] Available at: https://mcnextpost.com/2015/11/10/uwpxaml-compiled-binding-binding-to-events [Accessed 12 February 2017]

msdn.microsoft.com. (2016). Navigation history and backwards navigation (Windows apps) | Microsoft Docs. [online] Available at: https://msdn.microsoft.com/en-us/library/windows/apps/mt465734.aspx [Accessed 20 October 2016]

newtonsoft.com (n.d.). Json.NET – Newtonsoft [online] Available at: http://www.newtonsoft.com/json [Accessed 10 January 2017] sitepoint.com. (2015). Re-introducing PDO: the Right Way to Access Databases [online] Available at: https://www.sitepoint.com/re-introducing-pdo-the-right-way-to-access-databases-in-php/ [Accessed 17 December 2016]

stackoverflow.com. (2008). c#? – How can I get the DateTime for the start of the week? - Stack Overflow [online] Available at: http://stackoverflow.com/questions/38039/how-can-i-get-the-datetime-for-the-start-of-the-week [Accessed 5 March 2017]

stackoverflow.com. (2009). python, maintaining ratio - Stack Overflow [online] Available at: http://stackoverflow.com/questions/929103/convert-a-number-range-to-another-range-maintaining-ratio [Accessed 2 February 2017]

stackoverflow.com. (2010). .net# - Convert DateTime MySQL using C# - Stack Overflow [online] Available at: http://stackoverflow.com/questions/3633262/convert-datetime-for-mysql-using-c-sharp [Accessed 28 December 2016]

stackoverflow.com. (2011). How to insert the current timestamp into MySQL database using a PHP insert query – Stack Overflow [online] Available at: http://stackoverflow.com/questions/6075926/how-to-insert-the-current-timestamp-into-mysql-database-using-a-php-insert-query [Accessed 18 January 2017]

stackoverflow.com. (2012 a). php – PDO were rows affected during execute statement - Stack Overflow [online] Available at: http://stackoverflow.com/questions/10522520/pdo-were-rows-affected-during-execute-statement [Accessed 28 December 2016]

stackoverflow.com. (2012 b). asp.net – Regular expression validator accept decimal numbers 0 or 5 - Stack Overflow [online] Available at: http://stackoverflow.com/questions/12225931/regular-expression-validator-accept-decimal-numbers-0-or-5 [Accessed 15 January 2017]

stackoverflow.com. (2014). c# - Combining Datepicker and Timepicker values Win 8.1 - Stack Overflow. [online] Available at: http://stackoverflow.com/questions/23250452/combining-datepicker-and-timepicker-values-win-8-1 [Accessed 21 October 2016]

stenobot.wordpress.com. (2015). UWP App Development: Styling the Mobile Status Bar – hi, i'm andy. [online] Available at: https://stenobot.wordpress.com/2015/07/08/uwp-app-development-styling-the-mobile-status-bar/ [Accessed 20 October 2016]

top5solutions.com. (2016). MVVM Pattern in UWP Windows 10 Apps [online] Available at: http://www.top5solutions.com/mvvm-pattern-uwp-windows-10-apps/ [Accessed 27 October 2016]

w3schools.com. (n.d.). PHP Update Data in MySQL [online] Available at: http://www.w3schools.com/PHP/php_mysql_update.asp [Accessed 17 December 2016]

Appendices

Appendix A – Data Dictionary for MySQL Database

tbl_users

Name	Description	Type	Null	PK/FK	Default	Extra
id	Unique ID	INT (6)	No	PK		Auto Increment
username	User’s desired username	VARCHAR (250)	No			Must be unique
password	User’s password	VARCHAR (250)	No			
email	Email address for the user to use for tasks such as password reset	VARCHAR (250)	No			Must be unique
isAdmin	Used to determine if user has admin privileges	TINYINT (1)	No		0	
apiKey	A unique API key to use for authenticating the user	VARCHAR (250)	No			

tbl_readings

Name	Description	Type	Null	PK/FK	Default	Extra
id	Unique ID	INT (6)	No	PK		Auto Increment
dateTime	Date and time when reading was taken	DATETIME	No			
readingValue	Value of the reading in mmol/L	DECIMAL (2,1)	No			
insulinValue	Amount of insulin taken in Units	DECIMAL (3,1)	No		0	
reason	Reason reading was taken	VARCHAR (250)	Yes			
modifiedDateTime	Stores date and time when reading was last edited	DATETIME	Yes			

tbl_user_readings

Name	Description	Type	Null	PK/FK	Default	Extra
id	Unique ID	INT (6)	No	PK		Auto Increment
userID	ID of User	INT (6)	No	FK		FK from tbl_users On Delete Cascade On Update Cascade
readingID	ID of Reading	INT (6)	No	FK		FK from tbl_readings On Delete Cascade On Update Cascade

tbl_devices

Name	Description	Type	Null	PK/FK	Default	Extra
id	Unique ID	INT (6)	No	PK		Auto Increment
userID	ID of User	INT (6)	No	FK		FK from tbl_users On Delete Cascade On Update Cascade
nickname	Nickname for device as specified by user	VARCHAR (250)	No			
deviceModel	Model of device for easy identification	VARCHAR (250)	No			
lastSync	Time when device last synced with the service	DATETIME	YES			

tbl_pending_deletes

Name	Description	Type	Null	PK/FK	Value	Extra
id	Unique ID	INT (6)	No	PK		Auto Increment
userID	ID of User	INT (6)	No	FK		FK from tbl_users On Delete Cascade On Update Cascade
deviceID	ID of Device	INT (6)	No	FK		FK from tbl_devices On Delete Cascade On Update Cascade
readingID	ID of Reading	INT (6)	No	FK		FK from tbl_readings On Delete Cascade On Update Cascade

Appendix B – DetailedAnalysis.cs Code Listing

```
using Diabetic_Monitor.Models;
using System;
using System.Collections.ObjectModel;

namespace Diabetic_Monitor.HelperClasses
{
    public class DetailedAnalysis
    {
        // Calculates an HbA1c value from a range of readings
        public HbA1c CalcuateHbA1c(ObservableCollection<Reading> range)
        {
            HbA1c result = new HbA1c();
            double totalBG = 0;
            int totalReadings = 0;

            foreach (Reading r in range)
            {
                totalBG = totalBG + r.BloodGlucoseReading;
                totalReadings++;
            }

            result.HbA1cAsPercentage = Math.Round((GetAverage(range) + 2.5944)
                / 1.5944, 1, MidpointRounding.AwayFromZero); // Calculate HbA1c as
percentage and round to 1 decimal place
            result.HbA1cAsmmolmol = Math.Round((result.HbA1cAsPercentage - 2.15)
                * 10.929, 1, MidpointRounding.AwayFromZero); // Calculate HbA1c as
mmol/mol and round to 1 decimal place

            if (double.IsNaN(result.HbA1cAsPercentage))
            {
                result.HbA1cAsPercentage = 0;
                result.HbA1cAsmmolmol = 0;
            }

            return result;
        }

        // Gets last reading as a string
        public string LastReadingAsString(ObservableCollection<Reading> range)
        {
            if (range.Count == 0)
            {
                return "No readings";
            }
            else
            {
                return range[range.Count - 1].BloodGlucoseReading.ToString() +
"mmol/L";
            }
        }

        // Gets last reading as a double
        public double LastReadingAsDouble(ObservableCollection<Reading> range)
        {
            if (range.Count == 0)
            {
                return 0;
            }
            else
            {
                return range[range.Count - 1].BloodGlucoseReading;
            }
        }

        // Generates analysis for the current week
        public WeeklyAnalysis GetWeeklyAnalysis(ObservableCollection<Reading>
thisWeeksReadings, ObservableCollection<Reading> lastWeeksReadings)
        {
            WeeklyAnalysis result = new WeeklyAnalysis();

            result.ThisWeekAverage = GetAverage(thisWeeksReadings);
            result.BreakfastAverage =
GetAverage(FilterReadingsByReason(thisWeeksReadings, "Breakfast"));
            result.LunchAverage = GetAverage(FilterReadingsByReason(thisWeeksReadings,
"Lunch"));
            result.DinnerAverage =
GetAverage(FilterReadingsByReason(thisWeeksReadings, "Dinner"));
            result.BedAverage = GetAverage(FilterReadingsByReason(thisWeeksReadings,
"Bed"));

            double lastWeekAverage = GetAverage(lastWeeksReadings);
            result.OverallChange = Math.Round(((result.ThisWeekAverage -
lastWeekAverage)
                / lastWeekAverage) * 100, 1, MidpointRounding.AwayFromZero);

            double lastWeekBreakfastAverage =
GetAverage(FilterReadingsByReason(lastWeeksReadings, "Breakfast"));
            double lastWeekLunchAverage =
GetAverage(FilterReadingsByReason(lastWeeksReadings, "Lunch"));
            double lastWeekDinnerAverage =
GetAverage(FilterReadingsByReason(lastWeeksReadings, "Dinner"));
            double lastWeekBedAverage =
GetAverage(FilterReadingsByReason(lastWeeksReadings, "Bed"));

            result.OverallChangeBreakfast = Math.Round(((result.BreakfastAverage -
lastWeekBreakfastAverage)
                / lastWeekBreakfastAverage) * 100, 1, MidpointRounding.AwayFromZero);
            result.OverallChangeLunch = Math.Round(((result.LunchAverage -
lastWeekLunchAverage)
                / lastWeekLunchAverage) * 100, 1, MidpointRounding.AwayFromZero);
            result.OverallChangeDinner = Math.Round(((result.DinnerAverage -
lastWeekDinnerAverage)
                / lastWeekDinnerAverage) * 100, 1, MidpointRounding.AwayFromZero);
            result.OverallChangeBed = Math.Round(((result.BedAverage -
lastWeekBedAverage)
                / lastWeekBedAverage) * 100, 1, MidpointRounding.AwayFromZero);

            return result;
        }

        // Updates a WeeklyAnalysis instance with feedback
        public WeeklyAnalysis GenerateFeedback(WeeklyAnalysis analysis, double
minTarget, double maxTarget)
        {
            if (analysis.BreakfastAverage >= maxTarget)
            {
                analysis.BreakfastFeedback = "Your blood glucose is high on average at
Breakfast, " +
                    "you may wish to take some extra insulin overnight to combat this
if your " +
                    "glucose levels rise in the evening.";
            }
            else if (analysis.BreakfastAverage <= minTarget)
            {
                analysis.BreakfastFeedback = "Your blood glucose is low on average at 
Breakfast, " +
                    "you may wish to lower your insulin doses in the evening or take 
an extra snack" +
                    " before bed to combat this.";
            }
            else
            {
                if (double.IsNaN(analysis.BreakfastAverage))
                {
                    analysis.BreakfastFeedback = "Lack of data for feedback.";
                }
                else
                {
                    analysis.BreakfastFeedback = "On average, your blood glucose at 
breakfast time is on track.";
                }
            }

            if (analysis.LunchAverage >= maxTarget)
            {
                analysis.LunchFeedback = "Your blood glucose is high on average at 
Lunch, " +
                    "you may wish to take some extra insulin to cover your Breakfast 
or any snack " +
                    "you have in the morning to combat this.";
            }
            else if (analysis.LunchAverage <= minTarget)
            {
                analysis.LunchFeedback = "Your blood glucose is low on average at 
Lunch, " +
                    "you may wish to lower your insulin doses at Breakfast or take an 
extra snack" +
                    " during the morning to combat this.";
            }
            else
            {
                if (double.IsNaN(analysis.LunchAverage))
                {
                    analysis.LunchFeedback = "Lack of data for feedback.";
                }
                else
                {
                    analysis.LunchFeedback = "On average, your blood glucose at lunch 
time is on track.";
                }
            }

            if (analysis.DinnerAverage >= maxTarget)
            {
                analysis.DinnerFeedback = "Your blood glucose is high on average at 
Dinner, " +
                    "you may wish to take some extra insulin overnight to combat this 
if your " +
                    "glucose levels rise in the evening.";
            }
            else if (analysis.DinnerAverage <= minTarget)
            {
                analysis.DinnerFeedback = "Your blood glucose is low on average at 
Dinner, " +
                    "you may wish to lower your insulin doses at Lunch or take an 
extra snack" +
                    " during the afternoon to combat this.";
            }
            else
            {
                if (double.IsNaN(analysis.DinnerAverage))
                {
                    analysis.DinnerFeedback = "Lack of data for feedback.";
                }
                else
                {
                    analysis.DinnerFeedback = "On average, your blood glucose at 
dinner time is on track.";
                }
            }

            if (analysis.BedAverage >= maxTarget)
            {
                analysis.BedFeedback = "Your blood glucose is high on average at Bed 
time, " +
                    "you may wish to take some extra insulin to cover your evening 
meal or any snack " +
                    "you have in the evening to combat this.";
            }
            else if (analysis.BedAverage <= minTarget)
            {
                analysis.BedFeedback = "Your blood glucose is low on average at Bed 
time, " +
                    "you may wish to lower your insulin doses in the evening or take 
an extra snack" +
                    " before bed to combat this.";
            }
            else
            {
                if (double.IsNaN(analysis.BedAverage))
                {
                    analysis.BedFeedback = "Lack of data for feedback.";
                }
                else
                {
                    analysis.BedFeedback = "On average, your blood glucose at bed time 
is on track.";
                }
            }

            return analysis;
        }

        // Filters readings by a reason value
        private ObservableCollection<Reading> 
FilterReadingsByReason(ObservableCollection<Reading> range, string reason)
        {
            ObservableCollection<Reading> result = new 
ObservableCollection<Reading>();
            foreach (Reading r in range)
            {
                if (!string.IsNullOrEmpty(r.Reason) && r.Reason.Equals(reason, 
StringComparison.Ordinal))
                {
                    result.Add(r);
                }
            }
            return result;
        }

        // Generates an average from a given range of readings
        private double GetAverage(ObservableCollection<Reading> range)
        {
            int count = 0;
            double total = 0;
            foreach (Reading r in range)
            {
                total = total + r.BloodGlucoseReading;
                count++;
            }
            return Math.Round(total / count, 1, MidpointRounding.AwayFromZero);
        }

    }
}
```

Appendix C – ValidateReading.cs Code Listing

```
using System;
using System.Text.RegularExpressions;

namespace Diabetic_Monitor.Models
{
    public class ValidateReading : Template10.Validation.ValidatableModelBase
    {
        public ValidateReading()
        {
            // Validation logic
            Validator = Validator = i =>
            {
                if (!string.IsNullOrEmpty(DateOfReading))
                {
                    if (DateTime.Parse(DateOfReading).Date > DateTime.Now.Date)
                    {
                        Properties[nameof(DateOfReading)].Errors.Add("Date must not be 
in the future");
                    }
                }

                if (!string.IsNullOrEmpty(TimeOfReading))
                {
                    if (DateTime.Parse(TimeOfReading).TimeOfDay > 
DateTime.Now.TimeOfDay)
                    {
                        Properties[nameof(DateOfReading)].Errors.Add("Time must not be 
in the future");
                    }
                }

                if (!string.IsNullOrEmpty(BloodGlucoseReading))
                {
                    var glucoseRegex = @"^([1-9]\d*|0)(\.\d)$";
                    Match glucoseMatch = Regex.Match(BloodGlucoseReading, 
glucoseRegex, RegexOptions.IgnoreCase);
                    if (!glucoseMatch.Success)
                    {
                        Properties[nameof(BloodGlucoseReading)].Errors.Add("Blood 
Glucose must be a positive integer (1-40) with 1 decimal place");
                    }
                }
                else
                {
                    Properties[nameof(BloodGlucoseReading)].Errors.Add("Please enter a 
Blood Glucose value");
                }

                if (!string.IsNullOrEmpty(InsulinDose))
                {
                    var insulinRegex = @"^([1-9]\d*|0)(?:\.[05]0?)?$";
                    Match insulinMatch = Regex.Match(InsulinDose, insulinRegex, 
RegexOptions.IgnoreCase);
                    if (!insulinMatch.Success)
                    {
                        Properties[nameof(InsulinDose)].Errors.Add("Insulin Dose must 
be a positive integer or a decimal ending in '.5'");
                    }
                }
                else
                {
                    Properties[nameof(InsulinDose)].Errors.Add("Please enter an 
insulin dose value");
                }
            };
        }
        public string DateOfReading { get { return Read<string>(); } set { 
Write(value); } }

        public string TimeOfReading { get { return Read<string>(); } set { 
Write(value); } }

        public string BloodGlucoseReading { get { return Read<string>(); } set { 
Write(value); } }

        public string InsulinDose { get { return Read<string>(); } set { Write(value); 
} }

        // Returns errors found by the validation logic as a string
        public string ErrorText()
        {
            string errorText = "";
            foreach (var error in Errors)
            {
                errorText = errorText + error.ToString() + "\n";
            }

            return errorText;
        }

    }
}
```

Appendix D – SQLite Database Implementation

The SQLite Database used by the application has two tables, one for storing Readings and one for storing SyncErrors. Listings of these Models are given below:

Reading.cs

```
using System;
using SQLite.Net.Attributes;

namespace Diabetic_Monitor.Models
{
    public class Reading
    {
        public int ReadingID { get; set; }

        [PrimaryKey]
        public DateTime DateTimeOfReading { get; set; }

        public double BloodGlucoseReading { get; set; }

        public double InsulinDose { get; set; }

        public string Reason { get; set; }

        public string ImgSrc { get; set; }

        public DateTime ModifiedDateTime { get; set; }

        public override string ToString()
        {
            return ReadingID.ToString() + ", " + DateTimeOfReading.ToString() + ", " +
                BloodGlucoseReading.ToString() + ", " + InsulinDose.ToString() + ", "
+ Reason;
        }

        // Returns a string value containing the location of a reason image
        public String SetReasonImage()
        {
            if (string.IsNullOrEmpty(Reason))
            {
                return null;
            }

            if (Reason.Equals("Breakfast", StringComparison.Ordinal))
            {
                return "../Assets/RecordScreenIcons/Toast-icon.png";
            }

            if (Reason.Equals("Lunch", StringComparison.Ordinal))
            {
                return "../Assets/RecordScreenIcons/Sandwich-icon.png";
            }

            if (Reason.Equals("Dinner", StringComparison.Ordinal))
            {
                return "../Assets/RecordScreenIcons/burger-2-icon.png";
            }

            if (Reason.Equals("Bed", StringComparison.Ordinal))
            {
                return "../Assets/RecordScreenIcons/bed-icon.png";
            }

            return null;
        }
    }
}
```

SyncError.cs

```
using System;
using SQLite.Net.Attributes;

namespace Diabetic_Monitor.Models
{
    public class SyncError
    {
        public string ErrorType { get; set; }
        [PrimaryKey]
        public DateTime DateTimeOfReading { get; set; }
        public int ReadingID { get; set; }
    }
}
```

A class to interface with the database and manage the two tables that use the above Models was written and provides methods for various tasks. These tasks are shown below with appropriate code samples:

Adding and updating data

The InsertOrReplace method is used when adding or updating data stored in the SQLite Database. This means that the same method is used for both types of task. The method to add a Reading instance to the database is shown below:

```
        public void AddReading(Reading readingToAdd) // Adds or updates readings in
the database
        {
            using (var db = new SQLiteConnection(new SQLitePlatformWinRT(),
dbPath))
            {
                db.InsertOrReplace(readingToAdd);
            }
        }
```

Figure D.1 - SQLiteDB.cs AddReading method

As seen above, the method takes an instance of the Reading class as a parameter which then gets added to the database. If the Reading passed into the method already exists in the Readings table, then the existing values are simply overwritten with those of the Reading that is being saved.

An almost identical method is in place to deal with SyncError values.

Retrieving data

Retrieving data from the database is done in a variety of ways depending on the task being performed in the app. When using the filter feature on the RecordedReadings page, the GetPreviousRange method is used. This method is listed below:

```
        public ObservableCollection<Reading> GetPreviousRange(int days) // Gets
range of readings from a specified number of days
        {
            ObservableCollection<Reading> readings = new
ObservableCollection<Reading>();
            var currentDateTime = DateTime.Now;
            var previousDateTime = currentDateTime.AddDays(-days);

            using (var db = new SQLiteConnection(new SQLitePlatformWinRT(),
dbPath))
            {
                var readingRange = from r in db.Table<Reading>() where
r.DateTimeOfReading < currentDateTime && r.DateTimeOfReading > previousDateTime
select r;
                foreach (var reading in readingRange)
                {
                    readings.Add(reading);
                }
            }

            return readings;
        }
```

Figure D.2 - SQLiteDB.cs GetPreviousRange method

As seen above, the method takes an integer representing the number of days of previous data it should retrieve, and returns an ObservableCollection containing the Readings it retrieves, if any.

Other implemented ways that a reading can be retrieved include getting all readings, specifying a specific date range, specifying a specific day and specifying the reading id or DateTime value. A method to retrieve all SyncErrors for the sync function is also implemented.

Removing Data

Similar to adding or editing data in the database, removing an item is as simple as passing the object to be removed into the appropriate method. The DeleteReading method is listed below:

```
        public void DeleteReading(Reading readingToDelete) // Deletes a reading
        {
            using (var db = new SQLiteConnection(new SQLitePlatformWinRT(),
dbPath))
            {
                db.Delete(readingToDelete);
            }
        }
```

Figure D.3 - SQLiteDB.cs DeleteReading method

There is also a method implemented for removing a reading by its id assigned by the web service for when any lists of ids from readings deleted on another device are received from the web service.

Clearing the database

In the event that the user wishes to reset the app to factory settings, the ClearLocalData method is used.

```
        public void ClearLocalData() // Clears all data stored in the SQLite
database
        {
            using (var db = new SQLiteConnection(new SQLitePlatformWinRT(),
dbPath))
            {
                db.Execute("DROP TABLE IF EXISTS READING");
                db.Execute("DROP TABLE IF EXISTS SYNCERROR");
            }
        }
```

Figure D.4 – SQLiteDB.cs ClearLocalData method

This drops both the Reading and SyncError tables from the database, which are recreated when required once the app has been set up again.

Appendix E – Web Service Communication Implementation

Handling Requests

To make requests to the web service, two instances of the HttpClient class were used, one for authenticated requests (dealing with user data) and one for non-authenticated requests (signup, login etc.). An example of how requests were made in the code is shown below with code extracts from the AddReading method with performs a POST request:

```
            JObject jsonRequestBody = new JObject();
            jsonRequestBody.Add("dateTime",
readingToAdd.DateTimeOfReading.ToString("yyyy-MM-dd HH:mm:ss"));
            jsonRequestBody.Add("readingValue",
readingToAdd.BloodGlucoseReading.ToString());
            jsonRequestBody.Add("insulinValue",
readingToAdd.InsulinDose.ToString());
            jsonRequestBody.Add("reason", readingToAdd.Reason);
```

Figure E.1 – Building the Request Body

First the request body is created using data from the reading to be added which is passed to the method as a parameter. Next the request is made using the following line of code:

```
HttpResponseMessage result = await httpClientAuth.PostAsync("/reading", new
StringContent(JsonConvert.SerializeObject(jsonRequestBody).ToString(),
Encoding.UTF8, "application/json"));
```

Figure E.2 – Getting the Response message

Once this returns back, it is parsed by JObject.Parse method which is part of the JSON.Net library, and checked for validity. If valid, the received data is handled accordingly. In this case, it is a reading ID that is returned which is added to the local database copy of the reading so the app can identify it on the web service if required for updates or deletion at a later time. This is shown in the code snippet below:

```
                        readingToAdd.ReadingID = (int)responseObject["readingID"];
// Set reading id assigned by server
                        dbConn.AddReading(readingToAdd); // Save updated reading
                        // Attempt to remove any SyncError
                        try {
dbConn.DeleteErrorByDateTime(readingToAdd.DateTimeOfReading); } catch { }
```

Figure E.3 – Updating local reading and removing any SyncErrors

As seen above, a try catch statement is used to find and delete any SyncErrors associated with the reading which may have been created on previous failed attempts to upload the reading. When an attempt fails, a SyncError is added so that when a Sync is performed from the Recorded Readings page, the operation can be retried. SyncErrors are created only when POST, PUT or DELETE operations for a Reading fail. The method that adds SyncErrors to the database is shown over the page:

```
        private void AddError(DateTime dateTime, int id, string type) // Adds entry
to the SyncError table in SQLite
        {
            SyncError error = new SyncError {DateTimeOfReading = dateTime,
ReadingID = id, ErrorType = type};
            dbConn.AddSyncError(error);
        }
```

Figure E.4 – AddError method

Sync Operations

The following method is used when a Sync operation is performed from the Recorded Readings screen:

```
        private async Task Sync()
        {
            await webConn.Sync(_settings.DeviceID);
            List<SyncError> errors = new List<SyncError>();
            errors = dbConn.GetSyncErrors();
            foreach (SyncError se in errors)
            {
                switch (se.ErrorType)
                {
                    case "ADD":
                        await
webConn.AddReading(dbConn.GetReadingByDateTime(se.DateTimeOfReading));
                        break;
                    case "UPDATE":
                        await
webConn.UpdateReading(dbConn.GetReadingByDateTime(se.DateTimeOfReading));
                        break;
                    case "DELETE":
                        await webConn.DeleteReading(se.ReadingID,
se.DateTimeOfReading, _settings.DeviceID);
                        break;
                }
            }
            UpdateReadingList();
        }
```

Figure E.5 – Sync method

First a call to the Sync method in the WebStorage class is made which performs a GET request to retrieve any new or updated readings as well as any IDs of readings that need to be removed from the local SQLite database. Next a list of SyncErrors is retrieved which are then processed with a switch statement that calls the appropriate method in the WebStorage class to retry any of the failed operations stored by the SyncErrors.

Appendix F – NFC Code Listing

```
        #region nfc
        private ProximityDevice _device;
        private long _subscriptionIdNdef;

        public void InitNFC()
        {
            // Initialize NFC
            _device = ProximityDevice.GetDefault();
            // Subscribe for arrived / departed events
            if (_device != null)
            {
                _device.DeviceArrived += NfcDeviceArrived;
                _device.DeviceDeparted += NfcDeviceDeparted;
            }
            // Update status text for UI
            SetStatusOutput(_device != null ? "StatusInitialized" :
"StatusInitFailed");
        }

        private void NfcDeviceDeparted(ProximityDevice sender)
        {
            SetStatusOutput("DeviceDeparted");
        }

        private void NfcDeviceArrived(ProximityDevice sender)
        {
            SetStatusOutput("DeviceArrived");
        }

        private void SubscribeForNFC()
        {
            // Only subscribe for messages if no NDEF subscription is already active
            if (_subscriptionIdNdef != 0) return;
            // Ask the proximity device to inform us about any kind of NDEF message
received from
            // another device or tag.
            // Store the subscription ID so that we can cancel it later.
            _subscriptionIdNdef = _device.SubscribeForMessage("NDEF",
MessageReceivedHandler);
            // Update status text for UI
            SetStatusOutput(string.Format("StatusSubscribed", _subscriptionIdNdef));
        }

        private void StopSubscription(bool writeToStatusOutput)
        {
            if (_subscriptionIdNdef != 0 && _device != null)
            {
                // Ask the proximity device to stop subscribing for NDEF messages
                _device.StopSubscribingForMessage(_subscriptionIdNdef);
                _subscriptionIdNdef = 0;
                // Update status text for UI - only if activated
                if (writeToStatusOutput) SetStatusOutput("StatusSubscriptionStopped");
            }
        }

        private void MessageReceivedHandler(ProximityDevice sender, ProximityMessage
message)
        {
            // Get the raw NDEF message data as byte array
            var rawMsg = message.Data.ToArray();
            NdefMessage ndefMessage;
            try
            {
                // Let the NDEF library parse the NDEF message out of the raw byte
array
                ndefMessage = NdefMessage.FromByteArray(rawMsg);
            }
            catch (NdefException e)
            {
                SetStatusOutput(string.Format("InvalidNdef", e.Message));
                return;
            }

            // Parse tag contents
            try
            {

                // Parse the contents of the tag
                ParseTagContents(ndefMessage);

                // Update status text for UI
                SetStatusOutput(string.Format("StatusTagParsed"));
            }
            catch (Exception ex)
            {
                SetStatusOutput(string.Format("StatusNfcParsingError", ex.Message));
            }

        }
        private async void SetReading(string nfcReading)
        {
            await messageDispatcher.RunAsync(CoreDispatcherPriority.Normal, () => {
                BloodGlucoseValue = nfcReading;
            });
        }
        private void ParseTagContents(NdefMessage ndefMessage)
        {
            // Loop over all records contained in the NDEF message
            foreach (NdefRecord record in ndefMessage)
            {
                var recordType = record.CheckSpecializedType(true);
                if (recordType == typeof(NdefTextRecord))
                {
                    NdefTextRecord readingRecord = new NdefTextRecord(record);
                    SetReading(readingRecord.Text);
                }
                else
                {
                    SetStatusOutput("Unsupported Tag Data");
                }
            }
        }

        private Windows.UI.Core.CoreDispatcher messageDispatcher =
Window.Current.CoreWindow.Dispatcher;
        private async void SetStatusOutput(string newStatus)
        {
            // Update the status output UI element in the UI thread
            // (some of the callbacks are in a different thread that wouldn't be
allowed
            // to modify the UI thread)
            await messageDispatcher.RunAsync(CoreDispatcherPriority.Normal, () => { if
(NFCStatus != null) NFCStatus = "Status: " + newStatus; });
        }
        #endregion nfc
```

Appendix G – Export as CSV Code Listing

```
        private async Task Export()
        {
            Csv csv = new Csv();
            // Set column names for csv file
            csv.HasColumnNames = true;
            csv.SetColumnName(0, "Date + Time");
            csv.SetColumnName(1, "BG Reading");
            csv.SetColumnName(2, "Insulin Dose");
            csv.SetColumnName(3, "Reason");

            // Add currently displayed readings to csv file
            int count = 0;
            foreach(Reading reading in ReadingList)
            {
                csv.SetCell(count, 0, reading.DateTimeOfReading.ToString());
                csv.SetCell(count, 1, reading.BloodGlucoseReading.ToString());
                csv.SetCell(count, 2, reading.InsulinDose.ToString());
                csv.SetCell(count, 3, reading.Reason);
                count++;
            }

            // Save csv to file
            string csvFile;
            csvFile = csv.SaveToString();

            var savePicker = new Windows.Storage.Pickers.FileSavePicker();
            savePicker.SuggestedStartLocation =
                Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
            // Dropdown of file types the user can save the file as
            savePicker.FileTypeChoices.Add("CSV", new List<string>() { ".csv" });
            // Default file name if the user does not type one in or select a file to
replace
            savePicker.SuggestedFileName = "Exported Readings";

            Windows.Storage.StorageFile file = await savePicker.PickSaveFileAsync();
            if (file != null)
            {
                // Prevent updates to the remote version of the file until
                // we finish making changes and call CompleteUpdatesAsync.
                Windows.Storage.CachedFileManager.DeferUpdates(file);
                // write to file
                await Windows.Storage.FileIO.WriteTextAsync(file, csvFile);
                // Let Windows know that we're finished changing the file so
                // the other app can update the remote version of the file.
                // Completing updates may require Windows to ask for user input.
                Windows.Storage.Provider.FileUpdateStatus status =
                    await
Windows.Storage.CachedFileManager.CompleteUpdatesAsync(file);
                if (status == Windows.Storage.Provider.FileUpdateStatus.Complete)
                {
                    await dialogs.MessageDialog("Readings Exported
Successfully!").ShowAsync();
                }
                else
                {
                    await dialogs.MessageDialog("Failed to export
readings").ShowAsync();
                }
            }
        }
```

Appendix H – Reminders.cs Code Listing

```
using System;
using Microsoft.Toolkit.Uwp.Notifications;
using Windows.UI.Notifications;

namespace Diabetic_Monitor.HelperClasses
{
    public class Reminders
    {
        private ToastContent content = new ToastContent()
        {
            Launch = "action=viewEvent&eventId=1983",
            Scenario = ToastScenario.Reminder,

            Visual = new ToastVisual()
            {
                BindingGeneric = new ToastBindingGeneric()
                {
                    Children =
                    {
                        new AdaptiveText()
                        {
                            Text = "Retest Blood Glucose"
                        },

                        new AdaptiveText()
                        {
                            Text = "Retest blood glucose to ensure it is rising again"
                        }
                    }
                }
            },

            Actions = new ToastActionsCustom()
            {
                Inputs =
                {
                    new ToastSelectionBox("snoozeTime")
                    {
                        DefaultSelectionBoxItemId = "15",
                        Items =
                        {
                            new ToastSelectionBoxItem("5", "5 minutes"),
                            new ToastSelectionBoxItem("15", "15 minutes")
                        }
                    }
                },

                Buttons =
                {
                    new ToastButtonSnooze()
                    {
                        SelectionBoxId = "snoozeTime"
                    },

                    new ToastButtonDismiss()
                }
            }
        };

        private ScheduledToastNotification RetestReminder()
        {
            // Create a reminder that will pop 15 minutes from now
            return new ScheduledToastNotification(content.GetXml(),
DateTime.Now.AddMinutes(15));
        }

        public void AddReminder()
        {
            // Schedule the reminder

ToastNotificationManager.CreateToastNotifier().AddToSchedule(RetestReminder());
        }
    }
}
```

Appendix I – LiveTile.cs Code Listing

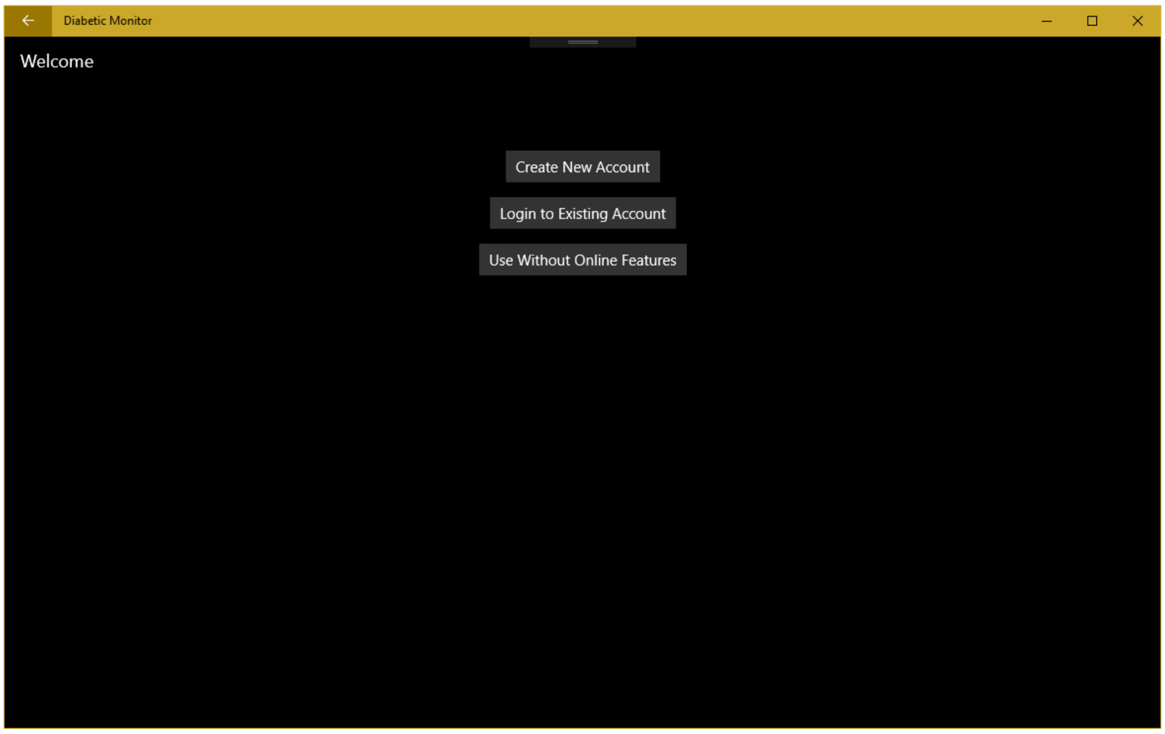
```
using Windows.UI.Notifications;
using Microsoft.Toolkit.Uwp.Notifications;
using System;

namespace Diabetic_Monitor.HelperClasses
{
    public class LiveTile
    {
        public void CreateNotification(string title, string message, string tag)
        {
            TileContent content = CreateTileContent(title, message);
            var notification = new TileNotification(content.GetXml());
            notification.ExpirationTime = DateTimeOffset.Now.AddHours(4);
            notification.Tag = tag;
            TileUpdateManager.CreateTileUpdaterForApplication().Update(notification);
        }
        private TileContent CreateTileContent(string title, string message)
        {
            // Construct the tile content
            TileContent tileContent = new TileContent()
            {
                Visual = new TileVisual()
                {
                    TileMedium = new TileBinding()
                    {
                        Content = new TileBindingContentAdaptive()
                        {
                            Children =
                            {
                                new AdaptiveText()
                                {
                                Text = title
                                },

                                new AdaptiveText()
                                {
                                    Text = message,
                                    HintStyle = AdaptiveTextStyle.CaptionSubtle
                                }

                            }
                        }
                    },

                    TileWide = new TileBinding()
                    {
                        Content = new TileBindingContentAdaptive()
                        {
                            Children =
                            {
                                new AdaptiveText()
                                {
                                    Text = title,
                                    HintStyle = AdaptiveTextStyle.Subtitle
                                },

                                new AdaptiveText()
                                {
                                    Text = message,
                                    HintStyle = AdaptiveTextStyle.CaptionSubtle
                                }

                            }
                        }
                    }
                }
            };

            return tileContent;

        }
    }
}
```

Appendix J – Analysis Page Adaptive Triggers Code Listing

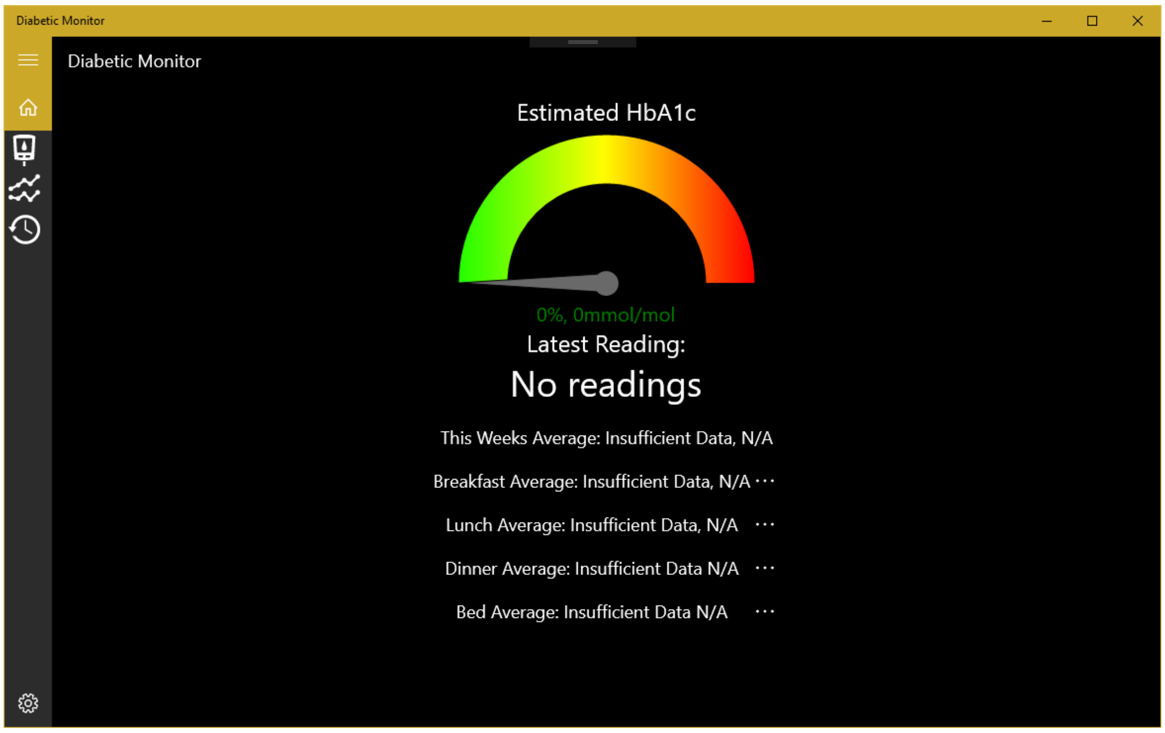
```
<VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="WindowStates">
                <VisualState x:Name="WideState">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="800" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <Setter Target="singleDayChart.Width" Value="720" />
                        <Setter Target="singleDayChart.Height" Value="400" />
                        <Setter Target="onOffTargetChart.Width" Value="720" />
                        <Setter Target="onOffTargetChart.Height" Value="400" />
                        <Setter Target="averageChart.Width" Value="720" />
                        <Setter Target="averageChart.Height" Value="400" />
                    </VisualState.Setters>
                </VisualState>
                <VisualState x:Name="NarrowState">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="0" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <Setter Target="singleDayChart.Width" Value="360" />
                        <Setter Target="singleDayChart.Height" Value="200" />
                        <Setter Target="onOffTargetChart.Width" Value="360" />
                        <Setter Target="onOffTargetChart.Height" Value="200" />
                        <Setter Target="averageChart.Width" Value="360" />
                        <Setter Target="averageChart.Height" Value="200" />
                    </VisualState.Setters>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
```

Appendix K – Adaptive UI PC Examples

First Time Setup



Main Screen



Estimated HbA1c

0%, 0mmol/mol

Latest Reading:

No readings

This Weeks Average: Insufficient Data, N/A

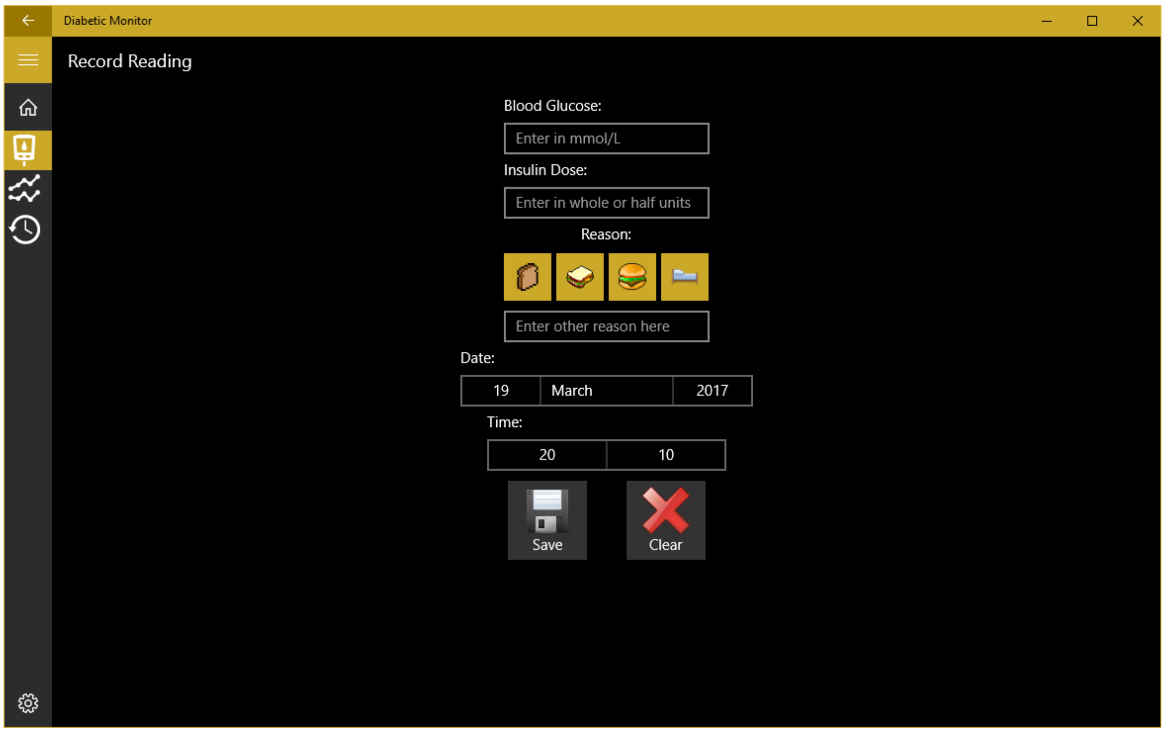
Breakfast Average: Insufficient Data, N/A ⋯

Lunch Average: Insufficient Data, N/A ⋯

Dinner Average: Insufficient Data N/A ⋯

Bed Average: Insufficient Data N/A ⋯

Record Readings



Blood Glucose:

Enter in mmol/L

Insulin Dose:

Enter in whole or half units

Reason:

Enter other reason here

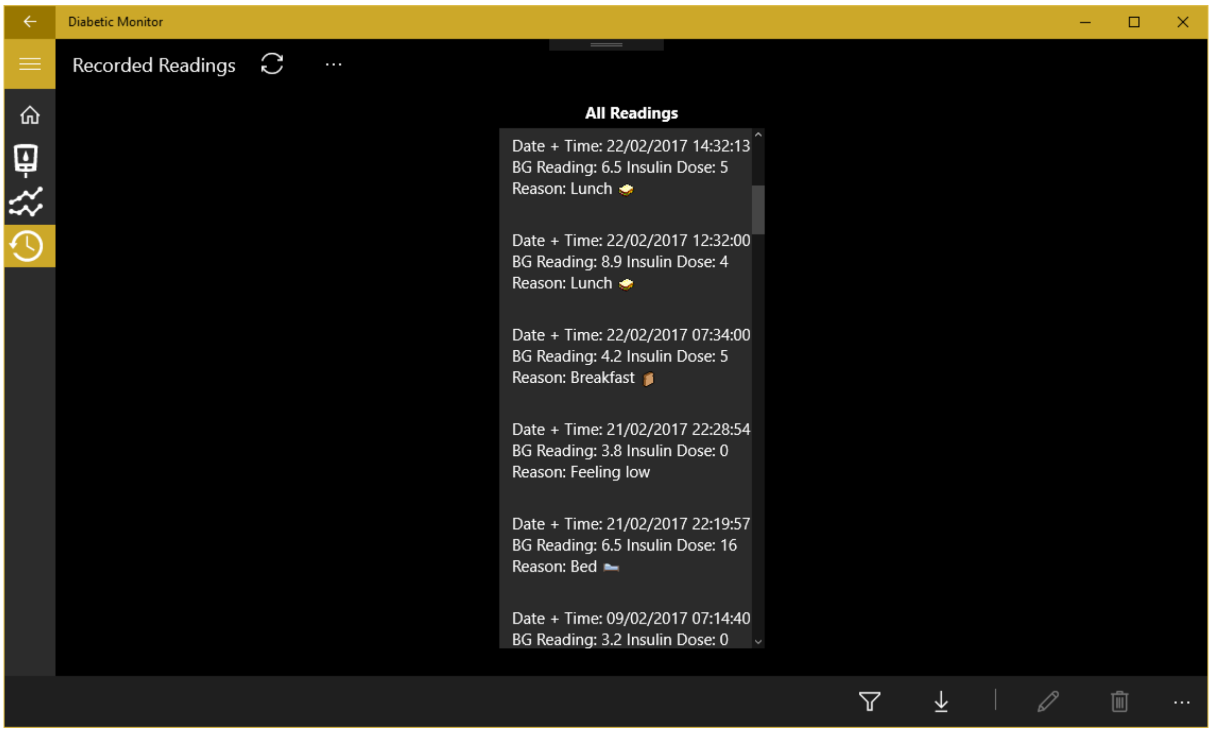
Date:

19 March 2017

Time:

20 10

Recorded Readings



All Readings

Date + Time: 22/02/2017 14:32:13
BG Reading: 6.5 Insulin Dose: 5
Reason: Lunch

Date + Time: 22/02/2017 12:32:00
BG Reading: 8.9 Insulin Dose: 4
Reason: Lunch

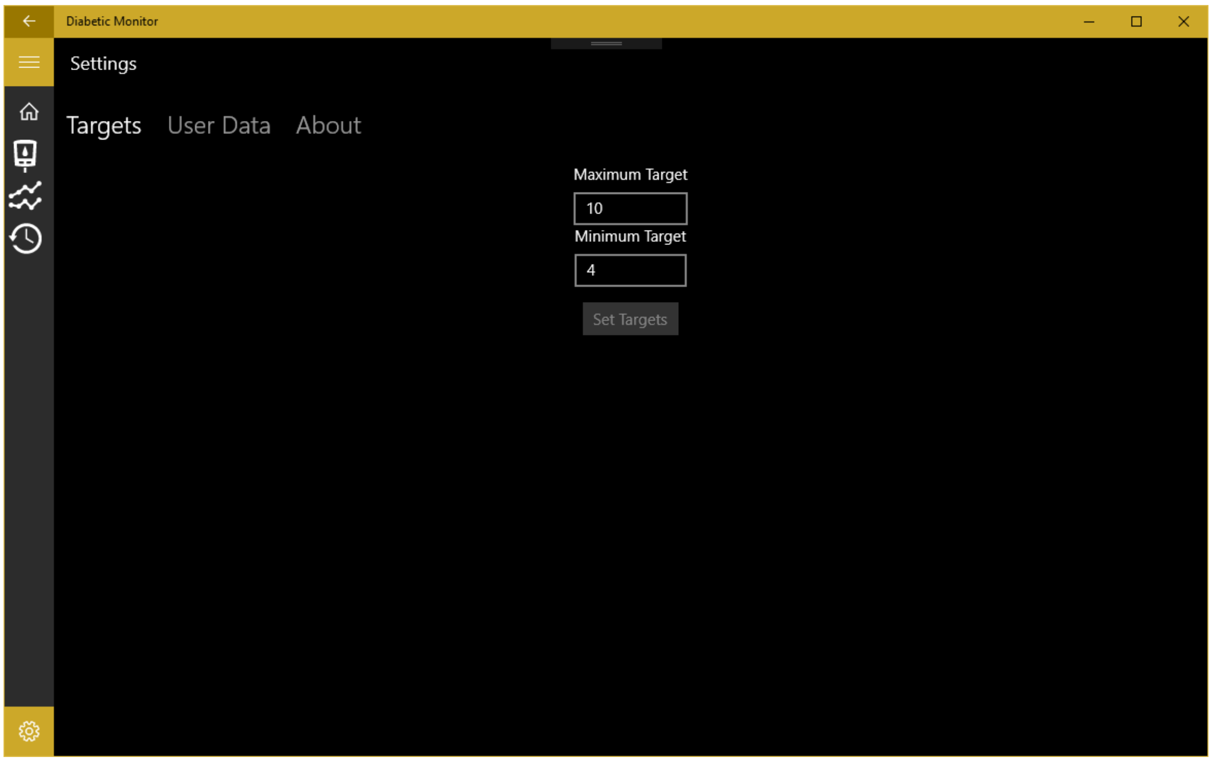
Date + Time: 22/02/2017 07:34:00
BG Reading: 4.2 Insulin Dose: 5
Reason: Breakfast

Date + Time: 21/02/2017 22:28:54
BG Reading: 3.8 Insulin Dose: 0
Reason: Feeling low

Date + Time: 21/02/2017 22:19:57
BG Reading: 6.5 Insulin Dose: 16
Reason: Bed

Date + Time: 09/02/2017 07:14:40
BG Reading: 3.2 Insulin Dose: 0

Settings



Maximum Target

10

Minimum Target

4

Appendix L – Unit Test code listings

DetailedAnalysisTestData.cs

```
using System;
using Diabetic_Monitor.Models;
using System.Collections.ObjectModel;

namespace Diabetic_Monitor_Tests
{
    public class DetailedAnalysisTestData
    {
        public ObservableCollection<Reading> HighReadings()
        {
            ObservableCollection<Reading> range = new ObservableCollection<Reading>();
            range.Add(new Reading { BloodGlucoseReading = 11.4, InsulinDose = 8,
                Reason = "Breakfast", DateTimeOfReading = DateTime.Parse("26/02/17
08:22")});
            range.Add(new Reading { BloodGlucoseReading = 10.4, InsulinDose = 7,
                Reason = "Lunch", DateTimeOfReading = DateTime.Parse("26/02/17
12:32")});
            range.Add(new Reading { BloodGlucoseReading = 12.6, InsulinDose = 9,
                Reason = "Dinner", DateTimeOfReading = DateTime.Parse("26/02/17
18:22")});
            range.Add(new Reading { BloodGlucoseReading = 13.4, InsulinDose = 18,
                Reason = "Bed", DateTimeOfReading = DateTime.Parse("26/02/17
23:47")});
            range.Add(new Reading { BloodGlucoseReading = 11.4, InsulinDose = 8,
                Reason = "Breakfast", DateTimeOfReading = DateTime.Parse("27/02/17
08:32")});
            range.Add(new Reading { BloodGlucoseReading = 10.4, InsulinDose = 7,
                Reason = "Lunch", DateTimeOfReading = DateTime.Parse("27/02/17
12:44")});
            range.Add(new Reading { BloodGlucoseReading = 12.6, InsulinDose = 9,
                Reason = "Dinner", DateTimeOfReading = DateTime.Parse("27/02/17
17:58")});
            range.Add(new Reading { BloodGlucoseReading = 13.4, InsulinDose = 18,
                Reason = "Bed", DateTimeOfReading = DateTime.Parse("27/02/17
23:54")});

            return range;
        }

        public ObservableCollection<Reading> LowReadings()
        {
            ObservableCollection<Reading> range = new ObservableCollection<Reading>();

            range.Add(new Reading { BloodGlucoseReading = 3.8, InsulinDose = 4,
                Reason = "Breakfast", DateTimeOfReading = DateTime.Parse("26/02/17
08:22")});
            range.Add(new Reading { BloodGlucoseReading = 3.5, InsulinDose = 5,
                Reason = "Lunch", DateTimeOfReading = DateTime.Parse("26/02/17
12:32")});
            range.Add(new Reading { BloodGlucoseReading = 3.9, InsulinDose = 3,
                Reason = "Dinner", DateTimeOfReading = DateTime.Parse("26/02/17
18:22")});
            range.Add(new Reading { BloodGlucoseReading = 3.4, InsulinDose = 14,
                Reason = "Bed", DateTimeOfReading = DateTime.Parse("26/02/17
23:47")});
            range.Add(new Reading { BloodGlucoseReading = 3.9, InsulinDose = 4,
                Reason = "Breakfast", DateTimeOfReading = DateTime.Parse("27/02/17 
08:32")});
            range.Add(new Reading { BloodGlucoseReading = 4.0, InsulinDose = 5, 
                Reason = "Lunch", DateTimeOfReading = DateTime.Parse("27/02/17 
12:44")});
            range.Add(new Reading { BloodGlucoseReading = 3.7, InsulinDose = 3, 
                Reason = "Dinner", DateTimeOfReading = DateTime.Parse("27/02/17 
17:58")});
            range.Add(new Reading { BloodGlucoseReading = 3.9, InsulinDose = 14, 
                Reason = "Bed", DateTimeOfReading = DateTime.Parse("27/02/17 
23:54")});

            return range;

        }

        public ObservableCollection<Reading> NormalReadings()
        {
            ObservableCollection<Reading> range = new ObservableCollection<Reading>();

            range.Add(new Reading { BloodGlucoseReading = 6.6, InsulinDose = 6, 
                Reason = "Breakfast", DateTimeOfReading = DateTime.Parse("26/02/17 
08:22")});
            range.Add(new Reading { BloodGlucoseReading = 6.7, InsulinDose = 4, 
                Reason = "Lunch", DateTimeOfReading = DateTime.Parse("26/02/17 
12:32")});
            range.Add(new Reading { BloodGlucoseReading = 5.6, InsulinDose = 8, 
                Reason = "Dinner", DateTimeOfReading = DateTime.Parse("26/02/17 
18:22")});
            range.Add(new Reading { BloodGlucoseReading = 8.8, InsulinDose = 16, 
                Reason = "Bed", DateTimeOfReading = DateTime.Parse("26/02/17 
23:47")});
            range.Add(new Reading { BloodGlucoseReading = 5.9, InsulinDose = 7, 
                Reason = "Breakfast", DateTimeOfReading = DateTime.Parse("27/02/17 
08:32")});
            range.Add(new Reading { BloodGlucoseReading = 6.8, InsulinDose = 5, 
                Reason = "Lunch", DateTimeOfReading = DateTime.Parse("27/02/17 
12:44")});
            range.Add(new Reading { BloodGlucoseReading = 7.7, InsulinDose = 9, 
                Reason = "Dinner", DateTimeOfReading = DateTime.Parse("27/02/17 
17:58")});
            range.Add(new Reading { BloodGlucoseReading = 7.5, InsulinDose = 16, 
                Reason = "Bed", DateTimeOfReading = DateTime.Parse("27/02/17 
23:54")});

            return range;

        }
    }
}
```

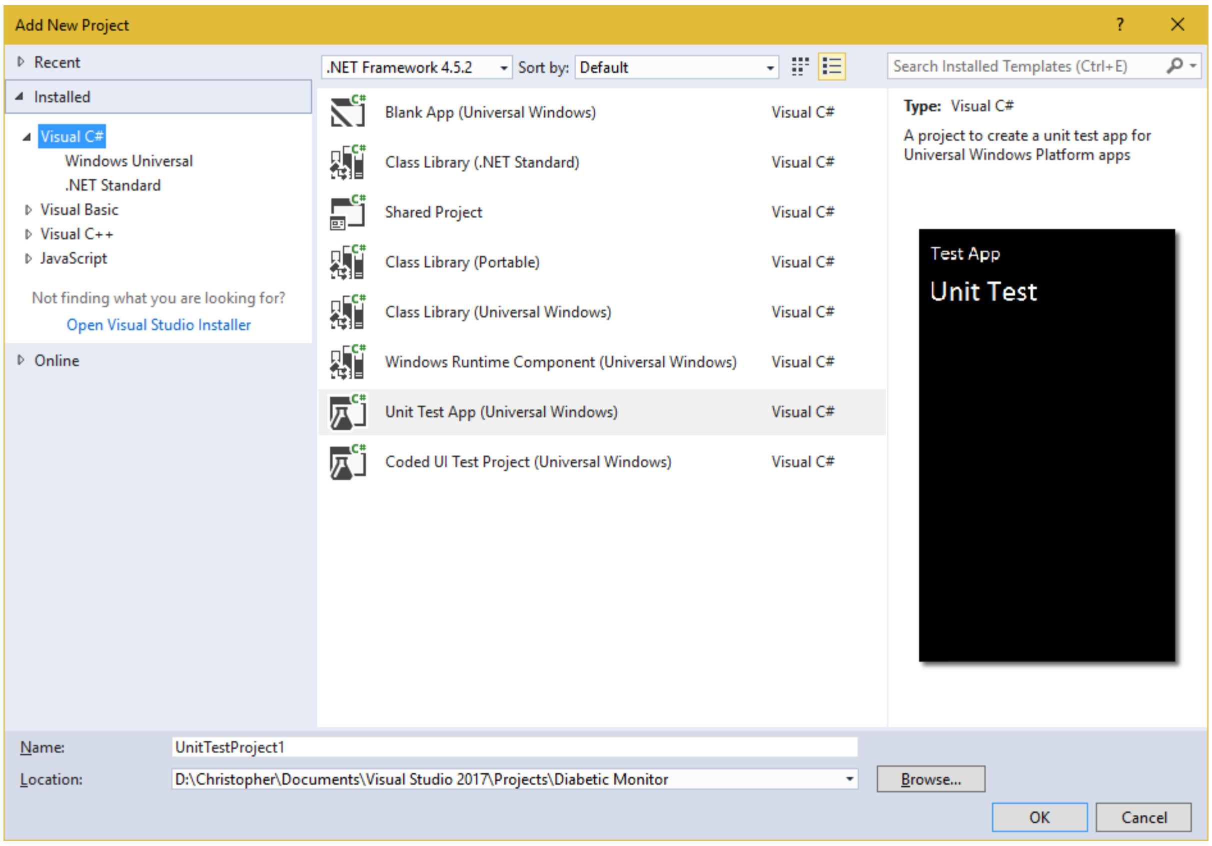
DetailedAnalysisTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Diabetic_Monitor.HelperClasses;
using Diabetic_Monitor.Models;
using System.Collections.ObjectModel;

namespace Diabetic_Monitor_Tests
{
    [TestClass]
    public class DetailedAnalysisTests
    {
        private DetailedAnalysis Analysis;
        private DetailedAnalysisTestData TestData;
        public DetailedAnalysisTests()
        {
            Analysis = new DetailedAnalysis();
            TestData = new DetailedAnalysisTestData();
        }

        [TestMethod]
        public void LastReadingTest()
        {
            // test with readings
            ObservableCollection<Reading> range = TestData.NormalReadings();
            double testDouble = Analysis.LastReadingAsDouble(range);
            Assert.AreEqual(7.5, testDouble);
            string testString = Analysis.LastReadingAsString(range);
            Assert.AreEqual("7.5mmol/L", testString);
            // test without readings
            ObservableCollection<Reading> emptyRange = new
ObservableCollection<Reading>();
            double emptyTestDouble = Analysis.LastReadingAsDouble(emptyRange);
            Assert.AreEqual(0, emptyTestDouble);
            string emptyTestString = Analysis.LastReadingAsString(emptyRange);
            Assert.AreEqual("No readings", emptyTestString);
        }

        [TestMethod]
        public void HbA1cTest()
        {
            HbA1c testResult = Analysis.CalcuateHbA1c(TestData.NormalReadings());
            HbA1c emptyTestResult = Analysis.CalcuateHbA1c(new
ObservableCollection<Reading>());
            Assert.AreEqual(42.1, testResult.HbA1cAsmmolmol);
            Assert.AreEqual(6.0, testResult.HbA1cAsPercentage);
            Assert.AreEqual(0, emptyTestResult.HbA1cAsmmolmol);
            Assert.AreEqual(0, emptyTestResult.HbA1cAsPercentage);
        }

        [TestMethod]
        public void AveragesTest()
        {
            // Low averages
            WeeklyAnalysis lowTest =
Analysis.GetWeeklyAnalysis(TestData.LowReadings(), new
ObservableCollection<Reading>());
            Assert.AreEqual(3.8, lowTest.ThisWeekAverage);
            Assert.AreEqual(3.9, lowTest.BreakfastAverage);
            Assert.AreEqual(3.8, lowTest.LunchAverage);
            Assert.AreEqual(3.8, lowTest.DinnerAverage);
            Assert.AreEqual(3.7, lowTest.BedAverage);
            // Normal averages
            WeeklyAnalysis normalTest =
Analysis.GetWeeklyAnalysis(TestData.NormalReadings(), new
ObservableCollection<Reading>());
            Assert.AreEqual(7.0, normalTest.ThisWeekAverage);
            Assert.AreEqual(6.3, normalTest.BreakfastAverage);
            Assert.AreEqual(6.8, normalTest.LunchAverage);
            Assert.AreEqual(6.7, normalTest.DinnerAverage);
            Assert.AreEqual(8.2, normalTest.BedAverage);

            // High averages
            WeeklyAnalysis highTest =
Analysis.GetWeeklyAnalysis(TestData.HighReadings(), new
ObservableCollection<Reading>());
            Assert.AreEqual(12.0, highTest.ThisWeekAverage);
            Assert.AreEqual(11.4, highTest.BreakfastAverage);
            Assert.AreEqual(10.4, highTest.LunchAverage);
            Assert.AreEqual(12.6, highTest.DinnerAverage);
            Assert.AreEqual(13.4, highTest.BedAverage);
        }

        [TestMethod]
        public void FeedbackTest()
        {
            // Low feedback
            WeeklyAnalysis lowTest =
Analysis.GenerateFeedback(Analysis.GetWeeklyAnalysis(TestData.LowReadings(),
                new ObservableCollection<Reading>()), 4.0, 10.0);
            Assert.AreEqual("Your blood glucose is low on average at Breakfast, " +
                    "you may wish to lower your insulin doses in the evening or take
an extra snack" +
                    " before bed to combat this.", lowTest.BreakfastFeedback);
            Assert.AreEqual("Your blood glucose is low on average at Lunch, " +
                    "you may wish to lower your insulin doses at Breakfast or take an
extra snack" +
                    " during the morning to combat this.", lowTest.LunchFeedback);
            Assert.AreEqual("Your blood glucose is low on average at Dinner, " +
                    "you may wish to lower your insulin doses at Lunch or take an
extra snack" +
                    " during the afternoon to combat this.", lowTest.DinnerFeedback);
            Assert.AreEqual("Your blood glucose is low on average at Bed time, " +
                    "you may wish to lower your insulin doses in the evening or take
an extra snack" +
                    " before bed to combat this.", lowTest.BedFeedback);

            // Normal feedback
            WeeklyAnalysis normalTest =
Analysis.GenerateFeedback(Analysis.GetWeeklyAnalysis(TestData.NormalReadings(),
                new ObservableCollection<Reading>()), 4.0, 10.0);
            Assert.AreEqual("On average, your blood glucose at breakfast time is on
track.", normalTest.BreakfastFeedback);
            Assert.AreEqual("On average, your blood glucose at lunch time is on
track.", normalTest.LunchFeedback);
            Assert.AreEqual("On average, your blood glucose at dinner time is on
track.", normalTest.DinnerFeedback);
            Assert.AreEqual("On average, your blood glucose at bed time is on track.",
normalTest.BedFeedback);

            // High feedback
            WeeklyAnalysis highTest =
Analysis.GenerateFeedback(Analysis.GetWeeklyAnalysis(TestData.HighReadings(),
                new ObservableCollection<Reading>()), 4.0, 10.0);
            Assert.AreEqual("Your blood glucose is high on average at Breakfast, " +
                    "you may wish to take some extra insulin overnight to combat this 
if your " +
                    "glucose levels rise in the evening.", 
highTest.BreakfastFeedback);
            Assert.AreEqual("Your blood glucose is high on average at Lunch, " +
                    "you may wish to take some extra insulin to cover your Breakfast 
or any snack " +
                    "you have in the morning to combat this.", 
highTest.LunchFeedback);
            Assert.AreEqual("Your blood glucose is high on average at Dinner, " +
                    "you may wish to take some extra insulin overnight to combat this 
if your " +
                    "glucose levels rise in the evening.", highTest.DinnerFeedback);
            Assert.AreEqual("Your blood glucose is high on average at Bed time, " +
                    "you may wish to take some extra insulin to cover your evening 
meal or any snack " +
                    "you have in the evening to combat this.", highTest.BedFeedback);
        }

        [TestMethod]
        public void OverallChangeTest()
        {
            // Negative overall changes
            WeeklyAnalysis negativeChangeTest = 
Analysis.GetWeeklyAnalysis(TestData.LowReadings(), TestData.NormalReadings());
            Assert.AreEqual(-45.7, negativeChangeTest.OverallChange);
            Assert.AreEqual(-38.1, negativeChangeTest.OverallChangeBreakfast);
            Assert.AreEqual(-44.1, negativeChangeTest.OverallChangeLunch);
            Assert.AreEqual(-43.3, negativeChangeTest.OverallChangeDinner);
            Assert.AreEqual(-54.9, negativeChangeTest.OverallChangeBed);

            // Positive overall changes
            WeeklyAnalysis positiveChangeTest = 
Analysis.GetWeeklyAnalysis(TestData.NormalReadings(), TestData.LowReadings());
            Assert.AreEqual(84.2, positiveChangeTest.OverallChange);
            Assert.AreEqual(61.5, positiveChangeTest.OverallChangeBreakfast);
            Assert.AreEqual(78.9, positiveChangeTest.OverallChangeLunch);
            Assert.AreEqual(76.3, positiveChangeTest.OverallChangeDinner);
            Assert.AreEqual(121.6, positiveChangeTest.OverallChangeBed);
        }
    }
}
```

Appendix M – Unit Testing in Visual Studio

Unit testing in Visual Studio was used to conduct the unit testing part of the testing performed for the project. The process of how this was done and the issues encountered will be discussed over the next few pages.

First, another project of the type “Unit Test App (Universal Windows)” was added to the solution.



Blank App (Universal Windows)	Visual C#
Class Library (.NET Standard)	Visual C#
Shared Project	Visual C#
Class Library (Portable)	Visual C#
Class Library (Universal Windows)	Visual C#
Windows Runtime Component (Universal Windows)	Visual C#
Unit Test App (Universal Windows)	Visual C#
Coded UI Test Project (Universal Windows)	Visual C#

Type: Visual C#

A project to create a unit test app for Universal Windows Platform apps

Test App

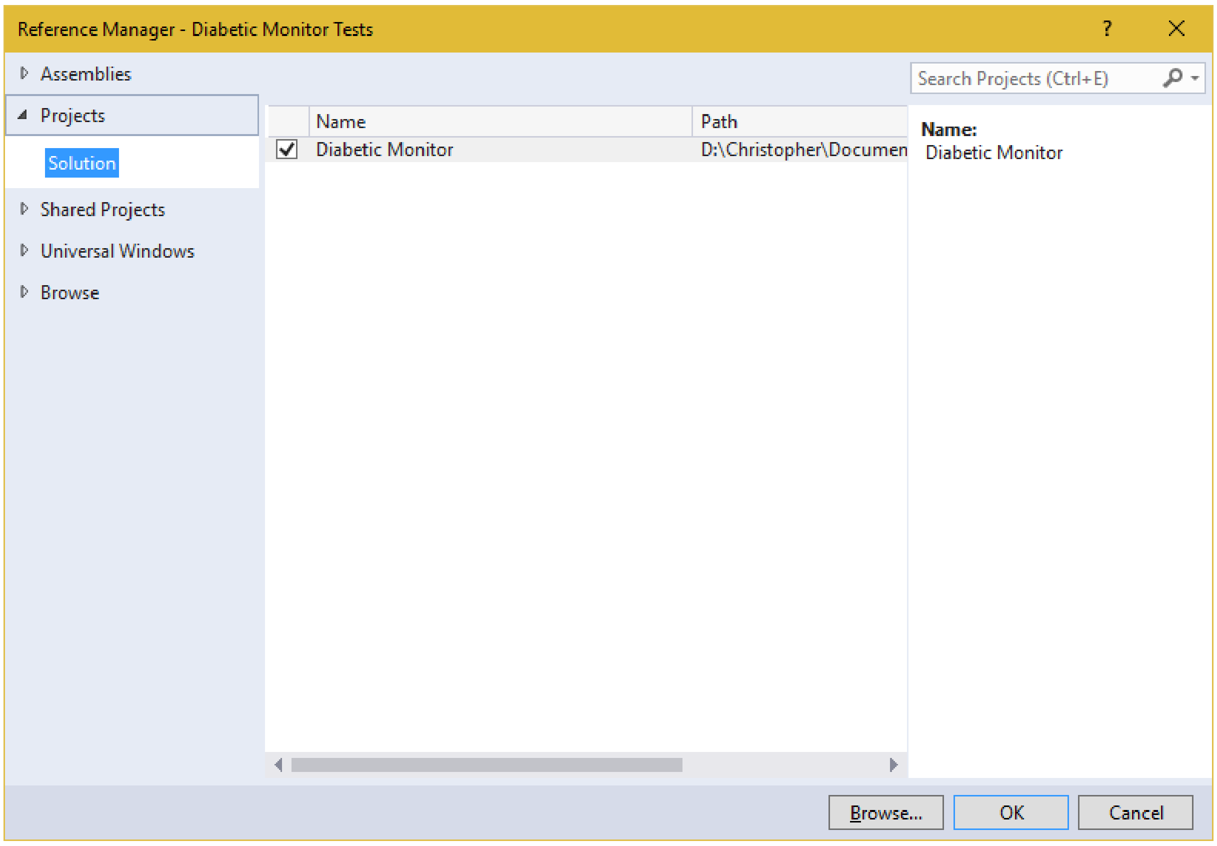
Unit Test

Name: UnitTestProject1

Location: D:\Christopher\Documents\Visual Studio 2017\Projects\Diabetic Monitor

Figure M.1 – Add New Project

Once the unit test project has been created, a reference to the UWP app project itself must be added. This gives the unit test project access to the code contained within the UWP app project so that it can create instances of the classes that will be tested with unit testing. The reference can be added by right clicking on “References” within the unit test project in the solution explorer.



	Name	Path
✓	Diabetic Monitor	D:\Christopher\Documen

Name:
Diabetic Monitor

Figure M.2 – Reference Manager

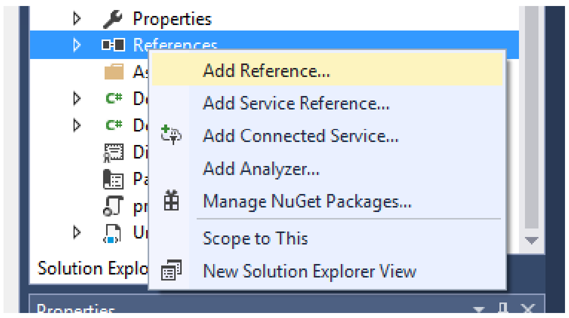
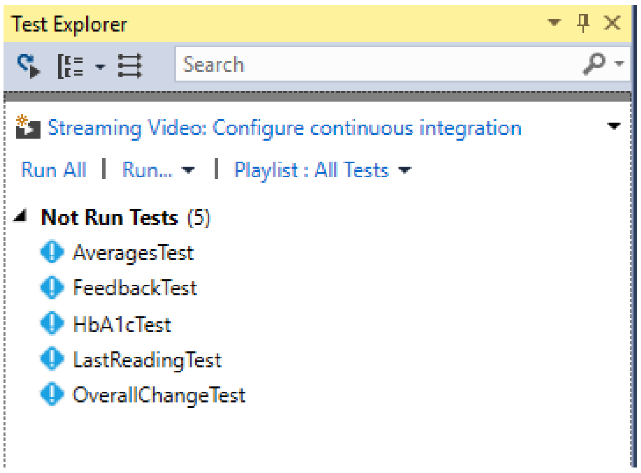


Figure M.1 – Adding a reference

Once the reference manager opens, as can be seen in Figure M.3 above, the UWP app solution needs to be selected after which, the dialog can be dismissed by clicking OK.

When the reference has been added successfully, the actual test classes themselves can be written. These are just standard C# classes that contain the line "`using Microsoft.VisualStudio.TestTools.UnitTesting;`" at the top of the file and "`[TestClass]`" above the class declaration. Similar to the class declaration, methods that contain tests need to have the line "`[TestMethod]`" placed above them. This allows the method to be identified as a test at build time. When the solution is built, these tests become available in the Test Explorer which can be seen in Figure M.4.

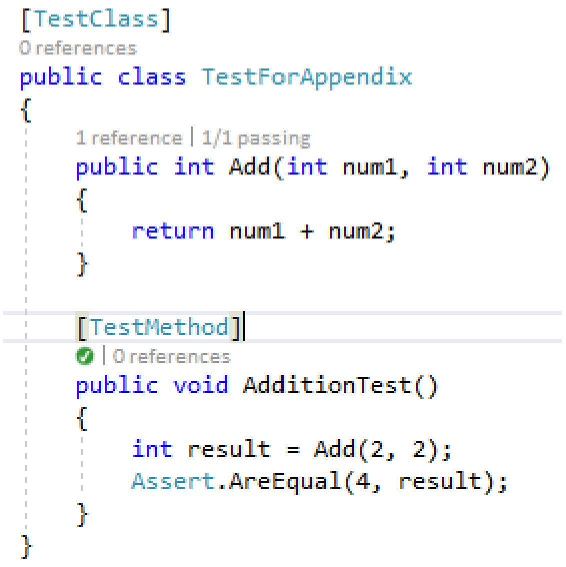

Test Explorer
Search
Streaming Video: Configure continuous integration
Run All | Run... | Playlist : All Tests

Not Run Tests (5)
- AveragesTest
- FeedbackTest
- HbA1cTest
- LastReadingTest
- OverallChangeTest

Figure M.3 - Test Explorer

Tests initially show as not run and change to show whether they ran successfully or not after they have been run. An example of tests that have been run successfully can be seen in Figure 5.7 showing unit test results in Chapter 5.

In this project, the tests written are for the DetailedAnalysis class and code listings for the classes that make up these tests can be seen in Appendix L. The tests have a value that they expect from the method they are testing and then use the Assert.AreEqual method to verify that the result matches what was expected. A very simple example of how this works can be seen below:



```
[TestClass]
0 references
public class TestForAppendix
{
    1 reference | 1/1 passing
    public int Add(int num1, int num2)
    {
        return num1 + num2;
    }

    [TestMethod]
    ✓ | 0 references
    public void AdditionTest()
    {
        int result = Add(2, 2);
        Assert.AreEqual(4, result);
    }
}
```

Figure M.4 - Test Example

The Assert.AreEqual method looks for the value 4 and compares it to the result that was returned from the call of the Add method. If the result does not match, then the test is marked as a failure. As can be seen in Figure M.5, above methods that have at least one test written for them is a note alongside the number of references that alerts the developer to how many tests that involve the method currently pass.

Tests can be debugged when results are not as expected and this functionality can be accessed by right clicking on the test(s) that require debugging and selecting "Debug Selected Tests" as seen in Figure M.6 below:

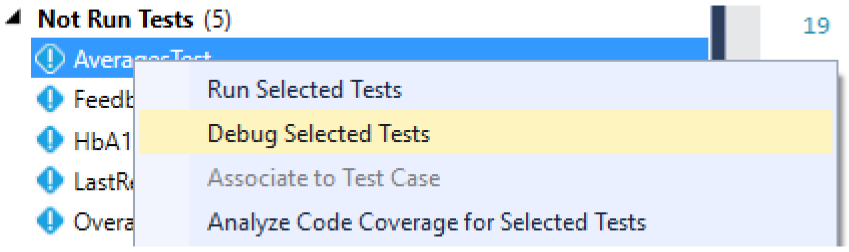
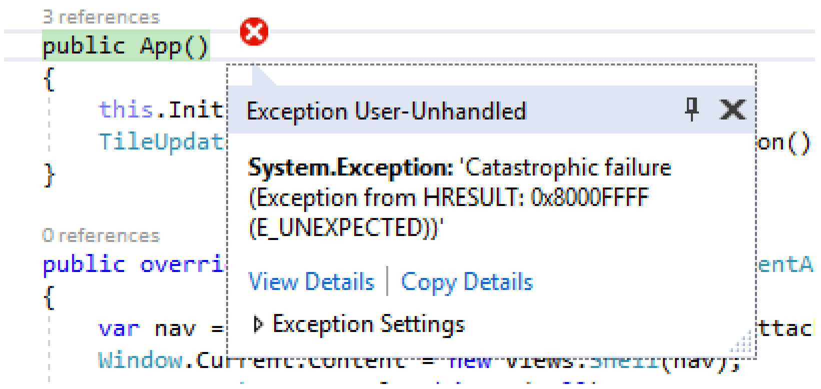


Figure M.5 – Test Explorer Context Menu

Initially, in this project debugging of the tests was not possible due to the following exception which prevented the process of debugging taking place.



```
3 references
public App()
{
    this.Init
    TileUpdat                                             on()
}

0 references
public overri                                             entA
{
    var nav =                                             ttac
    Window.Current.Content = new Views.Shell(nav);
```

Exception User-Unhandled

System.Exception: 'Catastrophic failure (Exception from HRESULT: 0x8000FFFF (E_UNEXPECTED))'

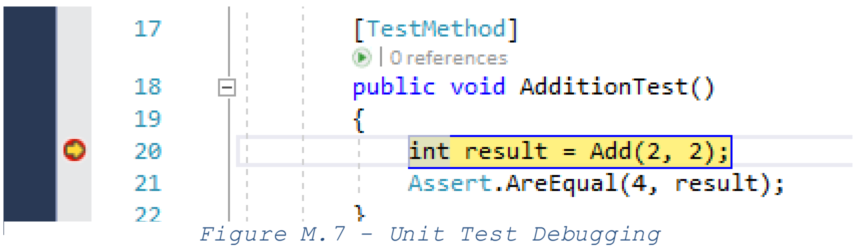
View Details | Copy Details

Exception Settings

Figure M.6 – Debugging Exception

In theory, the test app should have been able to be debugged in a similar manner to the UWP app itself. Upon investigation, discussions on the Template 10 github page lead to the conclusion that it is some kind of conflict with the fact that Template 10 based apps use a custom Application class which causes this behaviour. A workaround discussed on the Template 10 github page is to disable the "Break when this exception type is user-unhandled" option in the Exception Settings. This allows debugging to continue as normal and was a great help in the testing phase of the project and led to the discovery of the rounding error discussed in Chapter 5.

An example of unit test debugging functioning correctly in the same was as usual debugging in Visual Studio is shown below in Figure M.8:



```
[TestMethod]
0 references
public void AdditionTest()
{
    int result = Add(2, 2);
    Assert.AreEqual(4, result);
}
```

Figure M.7 – Unit Test Debugging