



**DUBLIN INSTITUTE
of TECHNOLOGY**

Institiúid Teicneolaíochta Bhaile Átha Cliath

Glucose Coach Final Year Project

**DT228
BSc in Computer Science**

Alex Kiernan
Supervisor: Dr. James D. Carswell

School of Computing
Dublin Institute of Technology

4th April 2017

Abstract

This report details the design, structure and conclusions reached during a final year project which aim to create a tool to help with people in the management of type 1 diabetes. The objective of the project is to create a comprehensive management system, which when trained by the user, will provide accurate predictions of insulin dosages needed.

In researching various diabetes management systems, it was apparent that the simple logging of data was their main feature. This system aims to utilize the machine learning feature to create an individual user profile. From this individual user profile, conclusions can be reached, i.e. the individual reaction to exercise taken and carbohydrates consumed. This feature would be invaluable, as it enables the user to keep track of their individual trends and their response to any insulin taken.

This project utilizes the linear regression algorithm to predict the insulin required based on the data the user has previously submitted. This project is comprised of a mobile app for data entry, a backend API for handling network requests and a desktop tool for syncing individual USB enabled blood glucose meters.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

A handwritten signature in black ink that reads "Alex Kiernan". The script is cursive and fluid.

Alex Kiernan

04/04/2017

Acknowledgments

I would like to thank:

- My project supervisor, Dr. James D. Carswell, for his advice and guidance throughout the project.
- The staff in the Department of Endocrinology & Diabetes Mellitus at St. Vincent's University Hospital for the valuable help and feedback they provided.
- Thanks also to my family, friends, and everyone who contributed to the user acceptance testing phase, helped with design decisions and participated for interviews.

Table of Contents

<i>Abstract.....</i>	<i>i</i>
<i>Declaration</i>	<i>ii</i>
<i>Acknowledgments.....</i>	<i>iii</i>
<i>Table of Contents.....</i>	<i>iv</i>
<i>Table of Figures</i>	<i>vi</i>
<i>Introduction.....</i>	<i>1</i>
a. Overview	1
b. Objectives.....	2
c. Challenges.....	2
d. Structure of Report.....	4
<i>Research</i>	<i>6</i>
a. Background Research.....	6
b. Alternative Existing Solutions	8
c. Solution Definition.....	13
d. Technologies Researched.....	15
e. Other Relevant Research	18
f. Resultant Findings and Requirements	20
<i>Design</i>	<i>21</i>
a. Design Methodology	21
b. UI Design.....	23
c. Features and Use-cases	29
d. Other Design Documents.....	36
<i>Architecture and Tools.....</i>	<i>40</i>
a. System Architecture.....	40
b. Development Tools.....	42
c. Source Code Layout	44
<i>Development.....</i>	<i>45</i>
a. Mobile App.....	46
b. API.....	54
c. Machine Learning System	57
d. Meter Tool.....	65
<i>System Validation</i>	<i>67</i>
a. Testing.....	67
b. Demonstration	71

Conclusion and Reflection	72
a. Project Summary	72
b. Future Work.....	73
c. Reflection	75
Bibliography.....	76
Appendix	77
a. Project Plan	77
b. API Endpoints	79
c. Partial Dataset.....	80
d. Amazon Machine Learning	81
e. Project Fair Poster	82

Table of Figures

Figure 1: Screenshot of GlucoseBuddy app where a user can enter data and a chart detailing a user's weekly results.	8
Figure 2: MyNetDiary showing web user interface displaying various user data and charts.	9
Figure 3: Glooko web interface displaying daily reading chart based on user data	10
Figure 4: Example of a logbook for people with type 1 diabetes	18
Figure 5: Example of material design	19
Figure 6: The PXP process phases	21
Figure 7: PC sync tool allowing users to upload data from device	23
Figure 8: Prototype of mobile client home screen with option menus	23
Figure 9: Prototype of mobile client home screen	23
Figure 10: Demonstration of insulin recommendation	24
Figure 11: Final mobile client home screen	24
Figure 12: Data entry menu that persists through fragments	25
Figure 13: Sliding menu for navigation	25
Figure 14: Sidebar menu that persists through fragments	26
Figure 15: Signup screen showing possible errors	26
Figure 16: Login screen for mobile client	26
Figure 17: Line chart showing history of user's blood glucose results	27
Figure 18: Logbook showing a user's logs	27
Figure 19: Meter sync tool login	28
Figure 20: Main screen with last sync date and sync button	28
Figure 21: Use case showing user login and registration interaction with the system	29
Figure 22: Use cases for entering data manually into the system	30
Figure 23: Glucose meter upload use case showing user interaction with the upload tool	31
Figure 24: Use case diagram showing suggestions being given to user's	32
Figure 25: Use case for training model	33
Figure 26: Use case for user viewing their previous results and options to view daily and monthly charts and statistics	34
Figure 27: Use case diagram of user's options to edit various settings	35
Figure 28: ERD showing the database entities that were initially used in the system	36
Figure 29: Updated ERD	37
Figure 30: Blood glucose log table	38
Figure 31: Prediction facts table	38
Figure 32: Trigger structure for logs table	39
Figure 33: Technical architecture diagram showing the different parts of the system and how they interact	40
Figure 34: Example of MySQL Workbenches query performance information	42
Figure 35: Postman endpoints for testing Glucose Coach API	43
Figure 36: Project structure of Android App	47
Figure 37: Source code for Android REST client –controller/ ApiManager.java	48
Figure 38: Custom interceptor for authentication headers – auth/AuthenticationInterceptor.java	49

Figure 39: Login service call to return a token – activities/LoginActivity.java - line 92	50
Figure 40: Initial home screen	51
Figure 41: Source code for the logbook observable - fragments/LogbookFragment - line 52	52
Figure 42: Example of add blood glucose log validation – activities/AddBgReadingActivity – line 114	53
Figure 43: Source code layout of API	54
Figure 44: Example of a blood glucose log class with mapping to the database - resources/bg_log – line 9	55
Figure 45: BG Logs endpoint with route and auth decorator – routes/bg_logs – line 12	56
Figure 46: Graph of actual data vs predicted data using NuPIC generated model	58
Figure 47: Response to query on multi-field inputs	58
Figure 48: Root mean square error of Amazon ML generated model	60
Figure 49: Regression algorithm in mathematical notation where y is the insulin dosage	61
Figure 50: Histogram detailing the diabetes dataset	61
Figure 51: Source code for script to compare machine learning methods	62
Figure 52: Cross validation results of each algorithm	63
Figure 53: Source code layout for meter tool	65
Figure 54: Docker file for building flask API container	67
Figure 55: Chart of exercise taken by user 1	69

Introduction

a. Overview

Type 1 diabetes is a metabolic disease that requires constant management. People with diabetes must keep careful track of what they eat, the exercise they do and the insulin they take in order to effectively manage their condition and reduce the risk of complications in later life.

Currently, there does not seem to be an effective system for profiling users, analyzing data which has been recorded and providing feedback on insulin requirements using machine learning. Apps in the market at present serve only as basic diary applications. Their main functionality is the logging and displaying of user data, and very few support the upload of data from user devices.

The aim of this project is to create a complete suite of features to allow people with type 1 diabetes to better manage their condition. This unified system, called "Glucose Coach", tracks a user's blood sugar levels, the food they have eaten and their physical exertion. Using these variables, the system will provide suggestions to users to enable them to better manage their glucose levels.

Many alternative diabetes management systems lack the ability to provide useful feedback to users, based on the data they have provided. Glucose Coach is more than a digital logbook, it provides useful information for users to better manage their diabetes.

Glucose Coach is designed with ease of use in mind. The ability of Glucose Coach to facilitate syncing of blood glucose logs enhances the speed and effectiveness of the system. Ultimately, it aims to provide accurate predictions with the assistance of the integrated machine learning component. Its main component is a front-end mobile app that communicates with a back-end service. The front-end allows users to enter and view their data. The back-end computes various recommendations using machine learning, such as what insulin dosage to take after a meal.

For ease-of-use, there is a desktop service that automatically pulls data from a user's glucose meter. This data is then uploaded to their profile, removing the need for manual entry into the system. Based on the data users have uploaded, accurate models of their blood sugar levels can be quickly and easily created.

Testing for Glucose Coach included user acceptance tests and software testing. User acceptance tests were carried out for a period of two weeks. Software testing was performed using Python and services such as Postman. During the design and development process, advice was sought from the diabetes center in St. Vincent's University Hospital on issues such as test data.

b. Objectives

The objectives of this project are as follows:

- To collect data that has been inputted by users with type 1 diabetes.
- To use the collected data to construct an individual model of the user.
- To persistently store model the model so that its available for access by the user on demand.
- To create an easy to use and intuitive user interface.
- To allow users to sync their blood glucose meters with the system.

c. Challenges

Machine Learning Concerns

For Glucose Coach to deliver its main system functionality, it needs to use a machine learning system. To accomplish this, research needed to be conducted and a sound process had to be developed, in order to get user information, process it in the machine learning system and export accurate results to the user.

The first issue was the unknown learning curve associated with working with a new machine learning system. There were unforeseen challenges in setting up the development environment, figuring out how to import data and creating a test environment for the results.

The second issue was the time needed to create test data for supervised learning. This is necessary, as Glucose Coach needs some form of test data to ensure that insulin dosage recommendations have a high degree of accuracy. This required every blood glucose training example input, to have the correct output.

Implementing Unit Tests

The development of Glucose Coach utilizes an agile approach which necessitates the use of unit tests. For each bit of functionality in Glucose Coach (mobile client, pc tool), an appropriate test suite had to be identified and implemented to ensure any code written meets its requirements.

The first issue was the time required to research and learn a test suite. Depending on the technology being used, each test suite requires the learning of a new syntax and method of implementation, which is time consuming. This had impacts on the ability to meet timeframes for weekly goals.

The second issue was being able to identify and create a comprehensive range of tests for the functionality of code written. This again required a significant amount of time to plan, write and run any unit tests. A problem which occurred was that the development of unit tests were given continuously less priority as deadlines approached which lead to some minor problems when integrating components into the project.

PC Tool Implementation

Glucose Coach has the feature of being able to sync a user's blood glucose meter with their PC to automatically pull data. This required research into how to navigate and pull the required information from a device and then push it to the database.

The proposed approach was the creation of a background tool. This tool needed to be able parse the data from the format used by the device, create a dataset and push the dataset to the remote database. The issue is that the meter is a closed, embedded system and research would need to be done to find out how to communicate with it.

d. Structure of Report

This report is structured as follows:

Research

Research consisted of the following:

- Background research conducted into diabetes and problems faced by people with diabetes.
- Comparison of Glucose Coach to other systems.
- Discussion of possible solutions to requirements.
- Research into technologies that could be utilized.

Design

Design consisted of the following:

- Identification of design methodology used for project.
- Designs for user interface.
- Features and use-cases available in Glucose Coach.
- Diagrams of initial and updated ERDs and triggers.

Architecture and Tools

Architecture and tools consisted of the following:

- Discussion of system architecture created and components chosen.
- Overview of development tools and version control used.

Development

Development consisted of the following:

- Details of each components developed for Glucose Coach.
- Overview, initial planning, application structure and development details.

System Validation

System validation consisted of:

- Results of testing.
- List of various problems found by users and feedback received.

Conclusion and Reflection

Conclusion and reflection consisted of the following:

- Summary of Glucose Coach system.
- Discussion on future work.
- Personal reflection on completing this project.

Bibliography & Appendix

Research

a. Background Research

Diabetes Overview

People with type 1 diabetes have lost the ability to produce insulin. In order to control their blood glucose levels, and keep them within an acceptable range, they need to take insulin. In order to calculate the required amount of insulin to achieve this, it is necessary to, measure their blood glucose readings, calculate the carbohydrate they have eaten and finally balance this against the exercise they may have performed [1].

Insulin is the primary method of controlling a person's blood sugar levels. Type 1 diabetics lose the ability to produce insulin in their pancreas due to β -cell destruction. This usually leads to absolute insulin deficiency [2]. Diabetics who use a basal bolus regimen must take multiple daily injections of insulin. The basal bolus regimen involves the taking of short acting insulin around meal times and a longer acting insulin in the morning or evening.

A1C is a lab test which shows the average level of a person's glucose levels over a period of three months. It is used as an indicator of how well a person is controlling their condition. People without diabetes have an A1C of less than 5.7%. However, a person with diabetes should aim for an A1C of around 7% [3].

A blood glucose test is typically performed before each meal. This test consists of a person taking a droplet of blood and analyzing its sugar content through the use of a glucometer [4]. The meter displays the amount of sugar in a sample in mmol/L (milimoles per liter) in Ireland and Canada, or mg/dl (Milligrams per Deciliter) in the United States. Meters typically record the history of a user's blood glucose tests and a timestamp for when they were performed.

Carbohydrate Counting

Nutritional management is an important aspect of diabetes management. Studies have found that the amount of insulin a person should take before a meal is directly related to the units of carbohydrate contained in what they have eaten [5]. This type of diabetes control is referred to as 'Carbohydrate Counting'.

Carbohydrates are the main source of energy for the human body. There are three main types of carbs in food; starches, sugars and fibers. When people consume food containing carbohydrates, the digestive system breaks down the digestible parts into glucose which then enters the bloodstream. In a person without diabetes, the pancreas releases insulin, a hormone which tells cells to absorb sugar for energy or storage.

The most common method of estimating the amount of insulin to take is through the process of carbohydrate counting. The benefits of this process include; spreading carbohydrate intake across the day, weight management and keeping blood glucose under control [6].

A person who is carb counting should first know what types of food contain carbohydrates. Common foods that have a high number of carbs include bread, pasta and potatoes. They should then calculate how many servings of the food they are having. A serving contains 15 grams of carbs, this can also be typically found on the nutritional information of a product. With this they can then deduce the amount of short acting insulin to take [6] (specifically for people on the basal bolus regimen).

Insulin

As mentioned above, insulin is the hormone used to regulate the amount of sugar in the blood stream. Type 1 diabetics must take insulin to prevent their blood sugars from going to high (hyperglycemia). They must also take sugar if their levels go too low (hypoglycemia).

Insulin comes in various types to treat diabetes. Rapid-acting insulin acts within 15 minutes of administration and peaks at about 1 hour. It is typically taken just before a meal. Long-lasting insulin last for up to 24 hours and is often used alongside other insulins [7].

Diabetics who use the basal bolus regimen take multiple daily injections of rapid-acting insulin [3]. The amount of rapid-acting insulin taken is based on the amount of carbohydrates they have calculated through carb counting. This carb-to-insulin ratio may differ from person to person, but is typically 1 unit of insulin to every 15 grams of carbs.

Impact of Exercise

Exercise is an important factor of diabetes management and can help in the control of blood glucose levels. Any exercise a person with diabetes does needs to be managed carefully as blood sugar levels can increase or decrease rapidly.

When a person exercises, their insulin sensitivity is increased. Cells are more able to use insulin in the body to take up glucose from the blood stream during and after exercise. When muscles contract during exercise, cells in the body take up glucose for energy whether insulin is available or not.

Exercise can lower a person's blood glucose level for up to 24 hours after the activity. Each person is different, but generally a person with type 1 diabetes would need to lower the amount of insulin they take in the hours after doing exercise [8] and increase the amount of carbohydrates they consume. Low blood sugar is most likely to occur when doing strenuous or prolonged amounts of exercise.

b. Alternative Existing Solutions

Glucose Buddy

Glucose Buddy is a free diabetes management application. It allows users to enter their blood glucose results into an iPhone or Android app (Figure 1). Users can also enter the insulin they have taken and the food which they have eaten.

Paying users can sync their results to their online profile and access limited information about them (such as weekly graphs) from a web browser. The app can also recognize blood glucose trends based on carbohydrate intake and the time of day.

In comparison to Glucose Coach, Glucose Buddy is more of a data storage utility than a fully-fledged management system. It does not feature a web client and it can only take user input through the mobile app. The input it can take is the standard data points such as blood glucose readings, insulin, food and exercise. It does not have any prediction or suggestions mechanisms to give users advice and does not have any support for uploading users results to their account from their blood glucose meters.



Figure 1: Screenshot of GlucoseBuddy app where a user can enter data and a chart detailing a user's weekly results.

MyNetDiary

Based on calorie counter applications, MyNetDiary is a paid for application that provides comprehensive blood glucose and carbohydrate counting features. They utilize their large food database to offer users accurate nutritional info on the food they have eaten.

The app and web client use a minimal UI that gives users a detailed insight into their various readings. It provides reminders to users in case they forget to log results. It also uses predictive analysis when users are typing to allow for quick and easy logging of food.

The system provides detailed daily and weekly analysis which can be viewed in the form of charts and graphs (Figure 2). With multiple overlays, this helps users to better understand the relationship between the food they have eaten, insulin they have taken and their blood glucose results.

The system also provides access to various third party APIs. This includes linking a user's fitness account from companies such as *Fitbit*. It also provides access to large and comprehensive food databases in order to provide the most up-to-date nutritional information.



Figure 2: MyNetDiary showing web user interface displaying various user data and charts.

In comparison to GlucoseBuddy, MyNetDiary has more features and provides a more complete system for a user managing their diabetes. It provides the logging of the normal data points, but has the addition of accessing a large and frequently (daily) updated database of nutritional information. A carb counting feature is also available to users.

In comparison to Glucose Coach, it lacks predictions or suggestions for users based on user data supplied. The only processing of data MyNetDiary provides is rudimentary daily and weekly analysis and the creation of charts and reports. A feature it does have which Glucose Coach lacks is the use of a user's GPS for exercise tracking.

Glooko

Glooko is a diabetes management system comprising of a web and mobile application. It is approved by the FDA and is HIPPA compliant. Its main selling point is its ability to pull in data from popular fitness trackers and other biometric devices. It also has support for syncing with certain blood glucose meters.

The primary method of input for the system is through the users mobile app. This app facilitates the entry of standard diabetes related data such as blood glucose, diet and fitness data into a digital logbook. Users can also access various charts and graphs detailing their blood glucose levels through an online web app (Figure 3).

The system has a limited blood glucose meter syncing facility. Users need to purchase a plug-in device that links a phone to a meter in order to sync their data. This is due to the reluctance of American vendors to open communication protocols for health care devices such as blood glucose meters and the need for FDA approval for such devices.

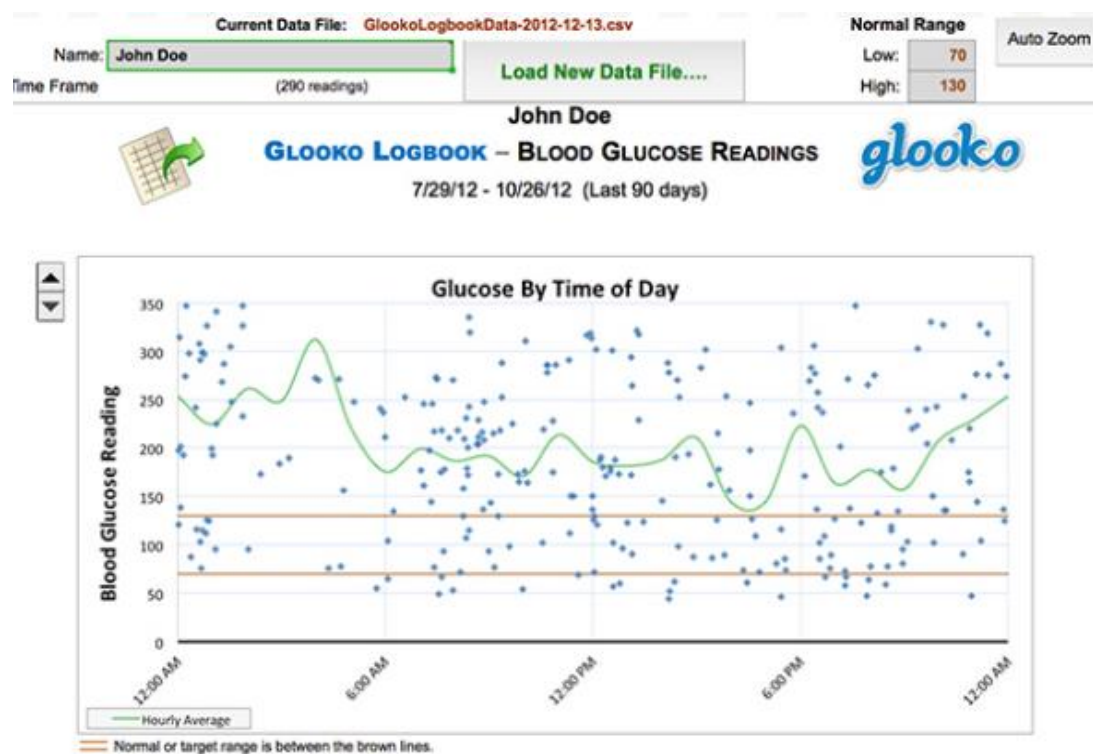


Figure 3: Glooko web interface displaying daily reading chart based on user data

In comparison to Glucose Buddy and MyNetDiary, Glooko is the nearest diabetes management implementation to Glucose Coach. Glooko's main feature is its ability to sync with supported meters through the use of a physical connector between a monitor and a phone. Like the other two systems, it supports user input for the common data points like food and carbs eaten. It also has access to a large nutritional database which is similar, but not as detailed as the one used by MyNetDiary.

Glucose Coach and Glooko share unique features, such as providing users with blood glucose level predictions based on their data and the automatic uploading of data from

user's monitors. However, its blood glucose level predictions are based on counting the number of high and low blood sugar readings rather than identifying trends in user's data using a machine learning algorithm. A feature it lacks, which is available on Glucose Coach, is the ability to provide insulin dosage suggestions to users based on machine learning data analysis.

Comparison Chart

	Web Client	Mobile Client	Insulin Tracking	Food Tracking	Exercise Tracking	Database Sync	A1c Analysis	Predictions	Charts/ Reports	Meter Uploads	Third Party API	Cost
Glucose Buddy		✓	✓	✓	✓	✓	✓		✓			Free
MyNetDiary	✓	✓	✓	✓	✓	✓	✓		✓		✓	9 euro a month
Glooko	✓	✓	✓	✓	✓	✓	✓	—	✓	✓	✓	60 euro per year
Glucose Coach		✓	✓	✓	✓	✓		✓	✓	✓		Free

c. Solution Definition

Glucose coach aims to provide users with a complete diabetes management system. The main objective is to give users accurate insulin predictions based on the data which they have entered into the system. Additionally, the system aims to be simple and easy to use in order to encourage users to consistently enter data. Based on these requirements, research done and the study of other competing systems, these components were chosen:

- A mobile application for taking user input, viewing data and serving back predictions to users.
- A machine learning library that can accurately predict the insulin a user needs to take based on data elements. The library should be able to persist models generated by the user to be used.
- A back-end REST API for handling interactions between the database and clients. The API would require authentication and use a framework that is easy to maintain and expand on.
- A desktop tool for syncing data from user blood glucose meters into the system via USB using HID packets.
- A script to generate data that simulates a person using the system. The data would have to reflect a typical person with Type 1 diabetes and how factors, such as exercise, would affect the output data.
- A RDBMS which is easy to use, quick and has support for stored procedures and triggers.

Mobile Application

The main interface for the system is an Android mobile app. The app allows users to enter information related to their daily diabetes management. There is support for user accounts whereby users can sign up and then log in to the system. Users can view their previous entries, see a chart detailing their blood glucose results and can choose to retrain their machine learning model based on new data.

The app is designed to be easy to use and visually appealing. It follows the material design principles to deliver a cleaner interface and to give users tactile feedback and a better user experience. This is important, as Glucose Coach relies heavily on the user regularly entering information into the system.

Machine Learning Library

The machine learning system needed to be easy to use, fast and provide accurate results. The preferable language for the library was Python, as this is simple, powerful and is the preferable language for machine learning libraries which brings good community support.

The machine learning system is responsible for creating models for each user. Insulin calculations are tailored to each individual user of the system as the response to exercise taken and carbohydrates consumed will be unique to each user. These calculations are

based on the machine learning models generated by the system once a threshold of data has been collected from the user. The models use linear regression in order to find patterns in the user's data based on the time of day, their blood glucose result, any exercise they did and the amount of carbohydrates consumed.

Backend

The back end for Glucose Coach consists of a Flask based REST API which handles user requests and responses. It also contains the Python machine learning library for the machine learning aspect of the Glucose Coach. Each resource on the API requires authentication. Users can request a token by submitting their credentials in order to avoid the user having to submit their details with each request.

The API allows for users to submit information related to their daily diabetes management, such as insulin taken, blood glucose results, carbohydrates consumed and exercise done. The API is also responsible for handling insulin predictions and model generation. Insulin predictions use data submitted by the user and returns an insulin value. Model generation uses data from a denormalized fact table to generate a model for the user.

Meter Syncing Tool

A desktop glucose meter syncing tool is included in the system which allows users to pull data from certain USB enabled blood glucose meters. This tool runs in an Ubuntu environment and can be invoked via the command line. Users can sign in to their account and sync any new data from their meter.

Database

For the database layer of Glucose Coach, a simple RDBMS is most suitable. The database should be easy to manage, fast and provide functionality such as triggers. A benefit of choosing a typical RDBMS, like MySQL, is that it is industry proven and has good community support. For hosting the database, a local server may be used for development and an instance of AWS RDS can be used for the live version.

d. Technologies Researched

The primary method of user input for Glucose Coach is through a mobile app. The app stores information related to the user's daily life, and presents the data in an easy to read format. The client communicates with a database which stores the user's data. The system provides insulin predictions based on user data. A Unix based automatic upload system for certain blood glucose meters is also available.

Android

The chosen platform for Android development is Android Studio. The IDE has strong integration with Git version control and has easy to use testing and debugging features. This app is targeting Android Lollipop and above as this allows for the use of the material design user interface.

Alternatives to a native application are hybrid apps, such as Cordova. However, from previous experiences and research done, a native application was found to be the better solution for a number of reasons:

- Native apps provide a faster and richer experience for users by leveraging the power of the native platform. Glucose Coach requires that users regularly log data into the system, so a sluggish experience through a hybrid app needs to be avoided.
- Native apps can take advantage of the various, extensive APIs offered on the Android platform. Examples of APIs that are integral to the Glucose Coach app include the Material Design library, background syncing of data using threads and SharedPreferences for storing user information.
- Android Studio provides a rich and extensive development experience. It allows for unit testing using the proven Junit library and provides extensive debugging features. Learning another toolchains compiler and facing the same problem that would normally be faced when developing a native application would just add more confusion during development.
- Hybrid apps require internet connections, whereas native applications can store data for as long as necessary. Glucose Coach acts as a logbook for user data, so users should always be able to log data if there is no internet connection available.
- Community support for native apps is extensive. Common issues that would be faced in developing Glucose Coach have been encountered numerous times on forums such as Stack Overflow.

NuPIC

Numenta Platform for Intelligent Computing (NuPIC) is an open source machine learning system. It could be integrated into the suggestions for user's tool which would suggest to users what insulin they should take based on the data entered. It trains itself using the time-based patterns in data, can predict future values, and detects anomalies. It is written in Python and contains comprehensive documentation for implementation [9].

scikit-learn

Like NuPIC, scikit is an open source machine learning library written in Python. It is one of the most popular libraries and has a huge number of features for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, linear regression and naïve Bayes.

After having researched and tested both NuPIC and scikit-learn, scikit-learn was chosen as the most suitable machine learning library for Glucose Coach. Included are some of the findings:

- scikit-learn has more extensive and comprehensive documentation than NuPIC due to its popularity. There are numerous examples and tutorials available to help setup a development environment and begin coding.
- NuPIC is more suited to streams of time based data using time-based continuous learning algorithms. scikit utilises supervised and unsupervised learning algorithms that are more suited to the type of data that is used within Glucose Coach.
- scikit-learn models were much quicker to generate and easier to store than the models generated by NuPIC. A NuPIC model took over 3 minutes to generate and was very CPU intensive. In contrast, a linear regression model generated by scikit took only a few seconds to generate.

Flask

Database communication takes place between the mobile client and the database through a RESTful API. The API is written in Python and is based on the Flask microframework. Flask was chosen as it is simple to setup and run, provides powerful extensions allowing for additional functionality and has good community support and documentation.

Token authentication in the API is provided to prevent users sending credentials with each request and having to store them locally in plaintext. Tokens are implemented using the 'itsdangerous' library. This library uses cryptographically signed messages for providing authentication. This has the advantage over traditional token systems in that it does not require server side storage of tokens. Each token also has an associated timeout for additional security.

Route authentication is provided using the 'Flask-HTTPAuth' library. This provides a decorator design pattern that routes all API calls through an authentication function before they can proceed. The function can accept either tokens or username/passwords.

Django

An alternative to Flask was Django. Django is a large framework with an active community and provides extensive functionality. It provides all the required components needed for the Glucose Coach system, such as serialization, authentication, data models and routing.

Flask was chosen as it is more lightweight and suitable for the requirements of Glucose Coach as well as providing a better learning experience. As Django uses a strict MVC architecture, the requirements of an API for Glucose Coach would have to be molded around the framework. Flask supports most of the functionality offered by Django through extensions.

Meter Syncing Tool

The meter syncing tool is a desktop application that interacts with a user blood glucose meter. The library chosen for interacting with meters was PyUSB and the graphical user interface library chosen was TKInter. Both of these libraries are written in Python, therefore the meter syncing tool was written completely in Python.

MySQL

The database itself is a typical RDBMS, MySQL running on a remote server. MySQL was chosen as it is open source, has high performance and flexibility and is easy to use and industry proven. MySQL supports the creation of triggers and stored procedures which is required for Glucose Coach, something which alternatives like MongoDB do not.

USB OTG

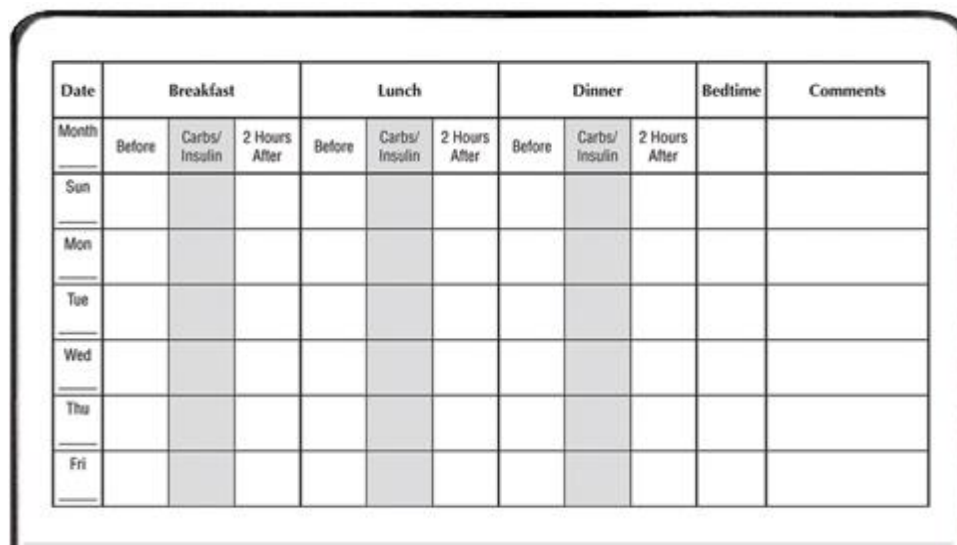
A planned feature is the use of an Android OTG cable that allows users to connect their blood glucose meters directly to their smartphone. Data can then be pulled and parsed in much the same way as can be done through the PC client program above.

USB On-The-Go is a specification that allows USB devices, such as Android smartphones, to act as hosts for other USB devices. Support for OTG was added in Android 3.1, but only newer devices support it. Glucose Coach could utilize OTG to connect to users USB enabled blood glucose meters, where the smartphone would act as a master and the glucose meter as a slave. Data could then be pulled from the meter, removing the need for users to connect their meter to a PC.

e. Other Relevant Research

User Logbook

People with type 1 diabetes need to keep detailed accounts of what they ate, the insulin they took, the exercise they did and their blood sugar readings. This data is normally written into a paper logbook (Figure 4). Information in feedback revealed several flaws in the system. These were, lack of room to write additional info, sloppy or smudged handwriting and the inconvenience of having to carry it around [10]. With Glucose Coach, the user can enter the standard data points (blood glucose, food, exercise and insulin) as well as be able to identify trends in their data.



Date	Breakfast			Lunch			Dinner			Bedtime	Comments
Month	Before	Carbs/ Insulin	2 Hours After	Before	Carbs/ Insulin	2 Hours After	Before	Carbs/ Insulin	2 Hours After		
Sun											
Mon											
Tue											
Wed											
Thu											
Fri											

Figure 4: Example of a logbook for people with type 1 diabetes

Material Design

Material design is a design language developed in 2014 by Google. It specifies a conceptual design philosophy that outlines how app should look and work on mobile devices. This design principle was the best suited for Glucose Coach as the app is Android based and the examples available best matched the vision for the mobile application (Figure 5).

The principles of material design include:

- Realistic visual cues: The design is grounded in reality and actually inspired by design with paper and ink.
- Bold, graphic and intentional: Fundamental design techniques drive the visuals. Typography, grids, space, scale, colour and imagery guide the entire design. Elements live in defined spaces with a clear hierarchy. Colour and type choices are bold and deliberate.
- Motion provides meaning: Animation is a key component of Material Design, but it can't just be there for the sake of movement. Animations need to happen in a single environment, serve to focus the design and include simple and easy transitions. Movements and actions should mirror the physical world.

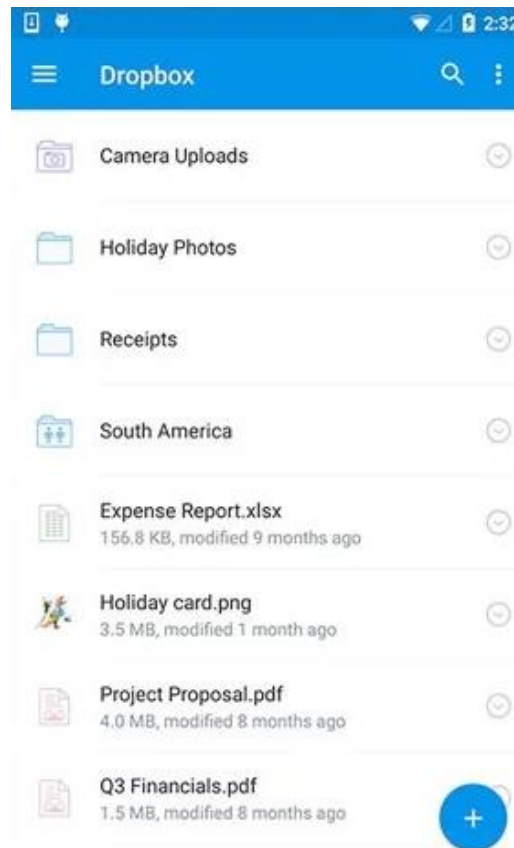


Figure 5: Example of material design

Amazon Web Services

Amazon Web Services (AWS) offers a suite of cloud computing services that can be leveraged in the development of Glucose Coach. A requirement of this project is that an API is developed to be accessed by clients. An AWS EC2 instance can be used to host the API. To make the API easier to deploy, a Docker container can be used to encapsulate the program and its dependencies. An AWS RDS instance can also be used to host the MySQL database.

Using AWS allows for secure and nearly always-on service. This means that the Glucose Coach backend is guaranteed to always be available from anywhere in the world. Additional security can be easily implemented using the AWS management console.

f. Resultant Findings and Requirements

Many alternative diabetes management systems lack the ability to provide useful feedback to users based on the data they have provided. Glucose Coach is more than a digital logbook and provides useful information for users to better manage their diabetes.

Glucose Coach has a simple and accessible user interface. Apps like MyNetDiary provide users with an easy and intuitive way to enter data into the app and has features, such as food suggestion, to aid and speed up user input. As Glucose Coach is heavily based on user data, the system encourages users to continuously input their blood glucose readings, insulin taken, food eaten and exercise performed. The UI is designed in such a way that makes the user view the act of entering data as less of a chore, and more as a part of their daily routine.

Glucose Coach can handle different types of data and provide contextual information to users based on that data. MyNetDiary gives users daily and weekly charts to help them see patterns in their blood glucose data but there is no correlation between what they ate or the exercise they did. Glucose Coach takes into account the effect that the food eaten, insulin taken and exercise performed has on blood glucose levels.

The system can pull data automatically from a user blood glucose meter and push it up to the database. Glooko supports the upload of user's data from select devices via a physical cable attached to a mobile phone, but this is cumbersome and error prone according to users. Glucose Coach's implementation for pulling user data is mostly automated and provides users with a simple user interface. It has integrated error prevention which is not visible to the user, such as a *last synced* timestamp and a *successful sync* flag.

Glucose Coach provides meaningful feedback to users based on trends identified by the data they have provided. Apps like MyNetDiary simply identify trends in blood glucose results, and fail to take into account levels of activity and carb consumption. Trends could be identified by examining various factors, such as mode of transport to work in the morning and carbohydrate content eaten at breakfast. If a particular model has been proved to be correct based on the user's blood glucose results, it is marked as being a successful fuzzy model and is more likely to be suggested in the future.

Design

a. Design Methodology

As this is a single developer project and I have experience of working on a scrum team during my work placement, I am using the agile software methodology Personal Extreme Programming (PXP) (Figure 6). This allows me to identify my progress towards goals, the ability to target small releases in my sprints and to heavily utilize Test Driven Development in my python framework and android app.

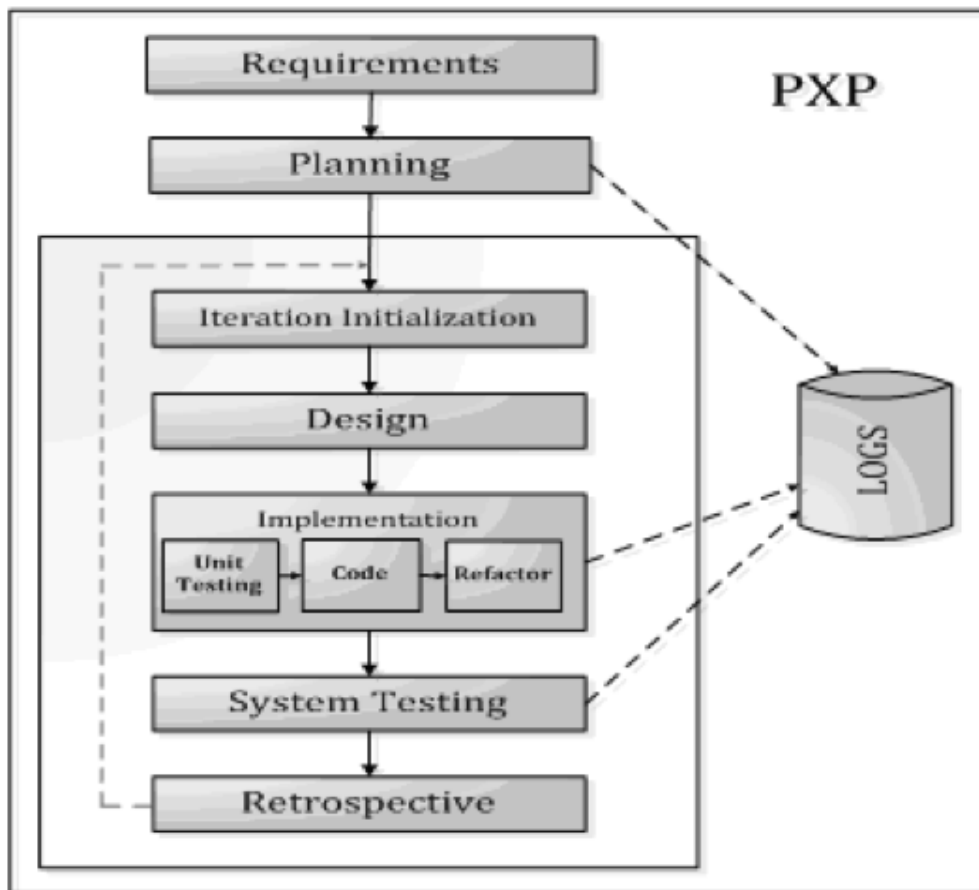


Figure 6: The PXP process phases

The first step of PXP is to define all the functional and non-functional requirements, which has been dealt with in the preceding section. Each development iteration consists of an iteration initialization and then a retrospective. During the process, I was required to keep a short log with information regarding task planning, actual duration, improvement suggestions and any problems found with details.

In the planning phase, I assembled a list of tasks that are based on the functional requirements. Each task is then composed of a list of smaller tasks, where I estimate the time for each task based on my previous experiences. The sum of these tasks form the time estimate for the parent task.

Iteration initialization indicates the beginning of each iteration. I started by selecting the tasks for the iteration which are my primary focus. Each iteration lasted between 1 to 3 weeks depending on the scope of the tasks. An iteration can result in a possible release build which I used to get feedback from users who are willing to test the system.

During the design phase, I modeled the system modules and various classes that are required in the ongoing iteration. I only designed the system to meet the requirements that were originally set out, and avoided any sort of feature creep.

Code was written in the implementation phase. I implemented the features laid out in the design phase and also wrote tests for them. This phase consisted of three phases: testing, code generation and code refactoring. To finish the implementation phase, the code I wrote should compile without errors and pass all tests successfully. Any problems found are recorded and fixed in system testing.

Iteration retrospective marks the end of the iteration. I needed to verify whether or not the estimated task time met the actual time taken. If the task time was over or under the estimated time, the reasons needed to be identified and logged. I also identified problems which could be addressed in future iterations.

b. UI Design

Prototypes

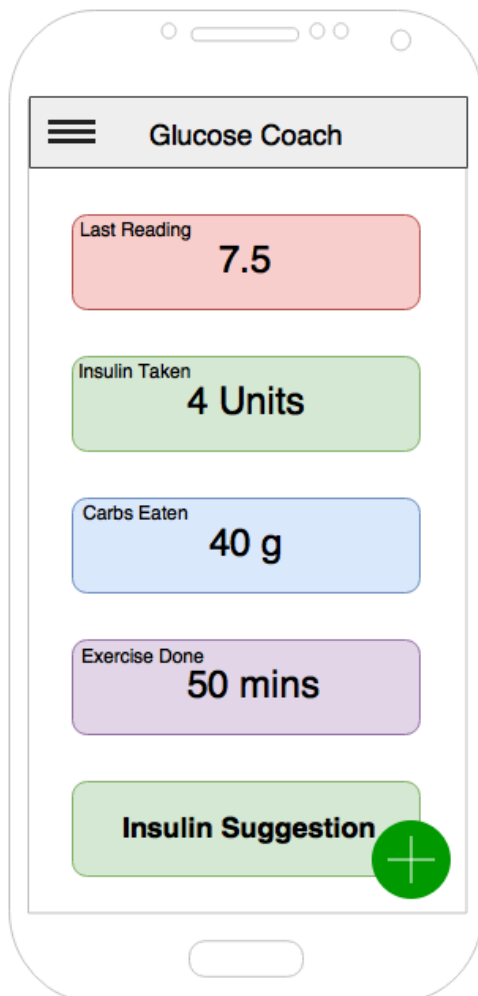


Figure 9: Prototype of mobile client home screen



Figure 8: Prototype of mobile client home screen with option menus

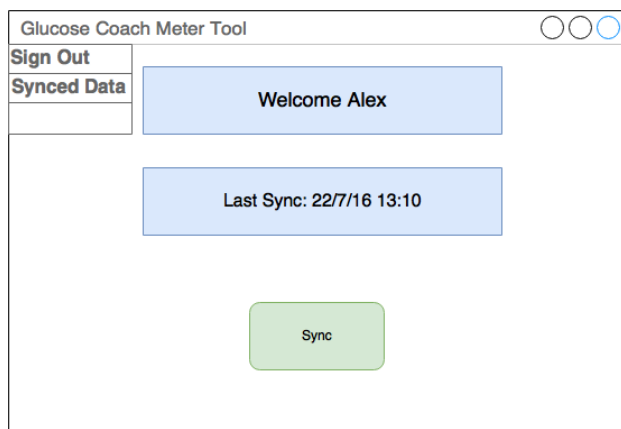


Figure 7: PC sync tool allowing users to upload data from device

Home Screen

The main home screen displays information which has been entered by the user in the period in which its being viewed, i.e. morning, afternoon, evening and night (Figure 12). The parameter for the morning is 5am – 12am, afternoon is 11 am to 5pm, evening is 5 pm to 8 pm and night is from 8 pm to 5 am. For example, in figure 12, *the red card displays the blood glucose reading entered by the user in the afternoon period. Other information that is displayed is the insulin dosage (green card), carbohydrates consumed (blue card) and calories burnt during exercise (purple card). These cards are for displaying information only and cannot be used to enter data.*

If the user wants to know what insulin they should take (i.e. before their next meal, or before they take exercise.), they can touch the grey 'Get Insulin Value' button. This instantiates a popup dialog (Figure 10). Based on the information the user has provided on the home screen, the machine learning system in the backend returns a suggested insulin dosage. The user can choose to accept (which will then be logged) or reject this suggestion. Acceptance of the suggestion results in the dosage being logged.

The aim of this design is to reduce the learning time, make the speed of data entry as fast as possible and have relevant information displayed on the home screen.

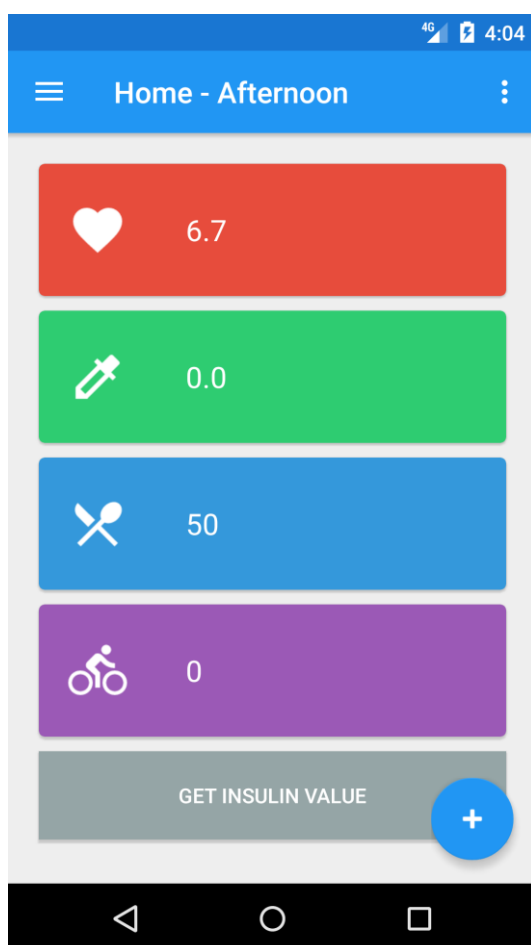


Figure 10: Final mobile client home screen

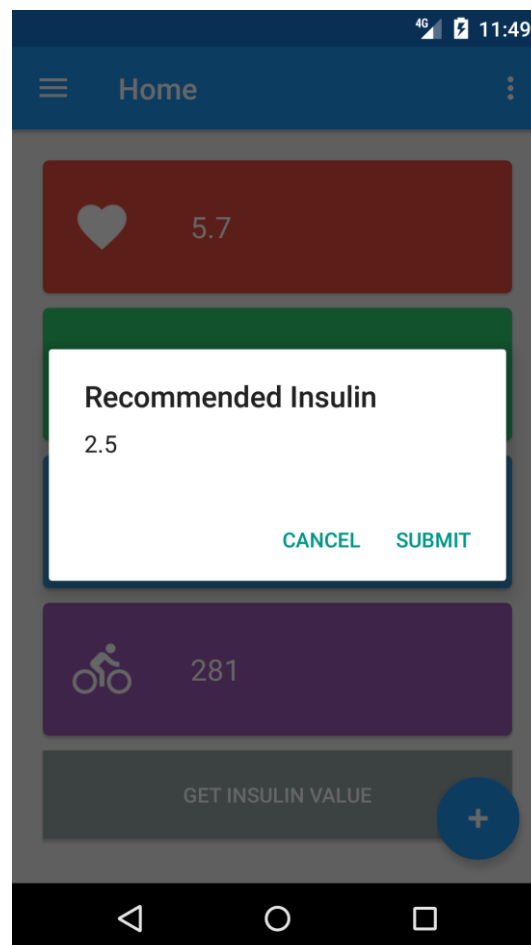


Figure 11: Demonstration of insulin recommendation

Options on Home Screen

A top bar is consistently displayed throughout the app. The top bar displays the name of the current screen and also displays the slide out menu button and the overflow menu used for signing out. Its intention is to give users a sense of accessibility and ease of navigation throughout the app.

It is comprised of a dropdown menu which displays a user's name profile picture at the top (figure 14). In addition, users can sign out by selecting the 'sign out' option in the overflow menu on the right-hand side of the top bar. The menu allows a user to navigate out of the current screen to any other part of the app. For example, a user can choose to return to the home screen from the logbook screen or can choose to edit settings from the home screen (Figure 13).

The action button is located on the home screen and any other screen which can be navigated to from the sidebar menu. When pressed, it expands out to give users options on which type of data they wish to add to the database. When an option is selected, they are taken to a new page that allows them to enter information regarding the type of data they are recording (Figure 12).

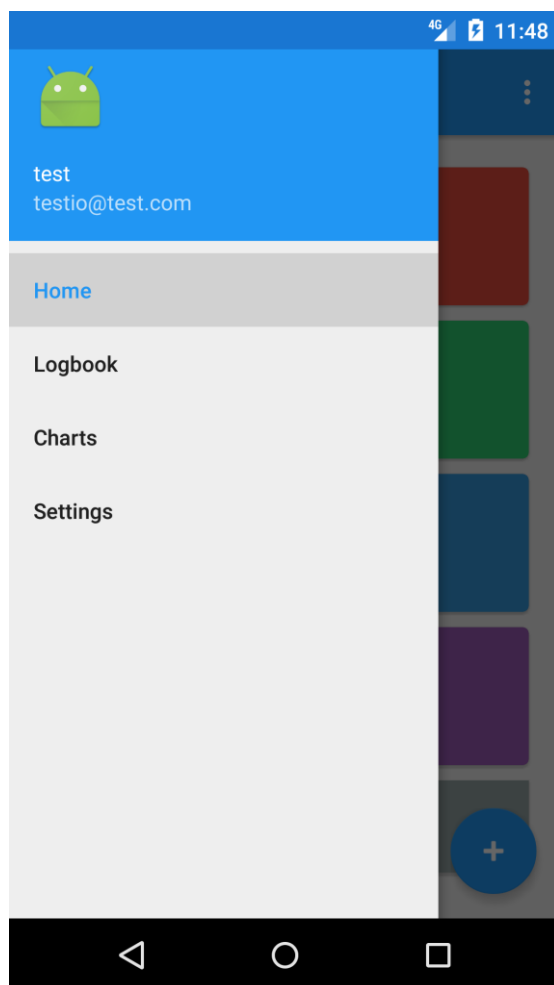


Figure 13: Sliding menu for navigation

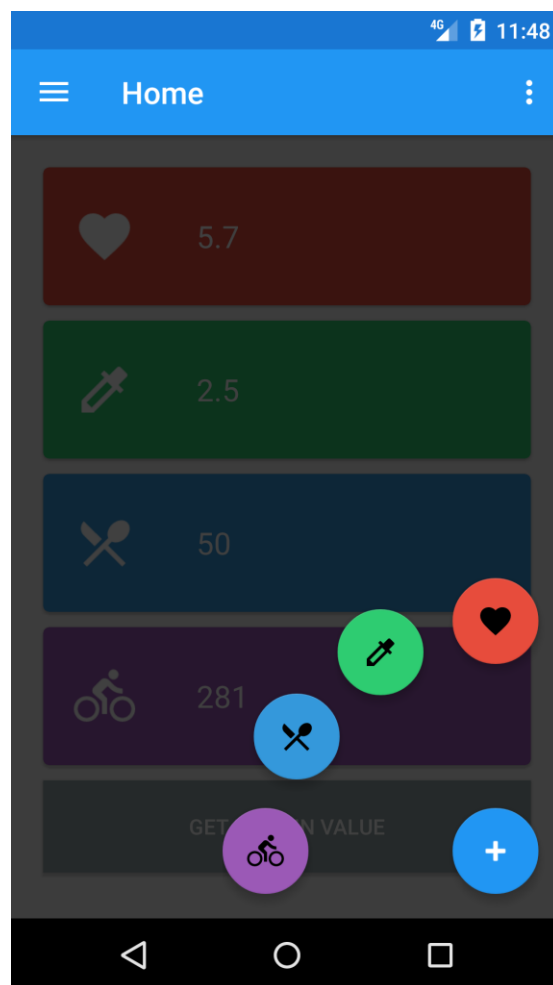


Figure 12: Data entry menu that persists through fragments

Login/Signup

The login screen is where users gain entry into the application (Figure 16). Its job is to verify that the user's credentials coincide with those stored in the database. If the sign in information entered is incorrect, a popup message appears to this effect. Once a user has logged in, they are not able to return to this page unless they explicitly opt to sign out from the top bar menu within the application. Users can sign up for an account by following the text link at the bottom of the screen.

The signup screen allows users to create a new account as mentioned above. They must enter a username, email, password and a password verification. Each value is passed through a regex validation function. If any of the data does not pass the validation, an error is displayed over the appropriate field (Figure 15). Once a user has signed up successfully, they are brought back to the sign in screen.

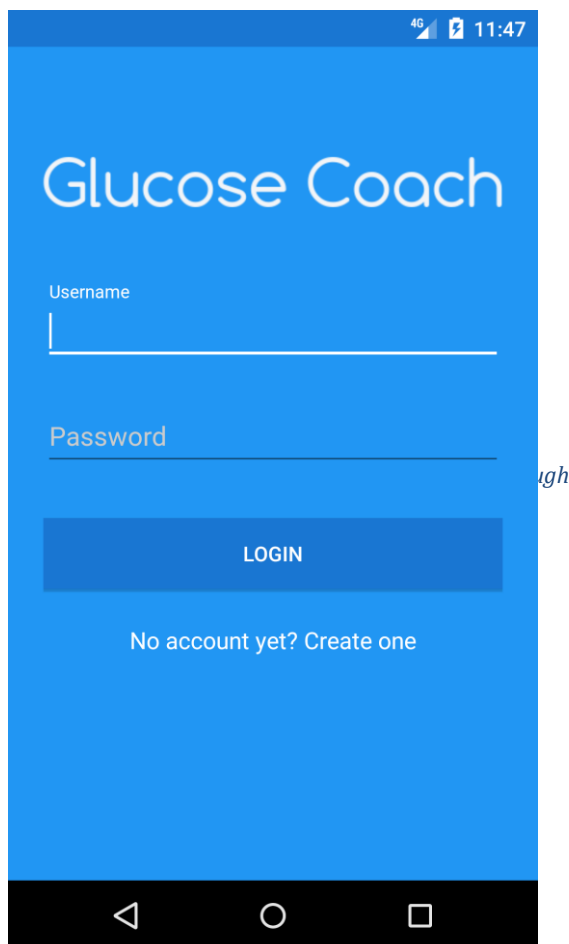


Figure 16: Login screen for mobile client

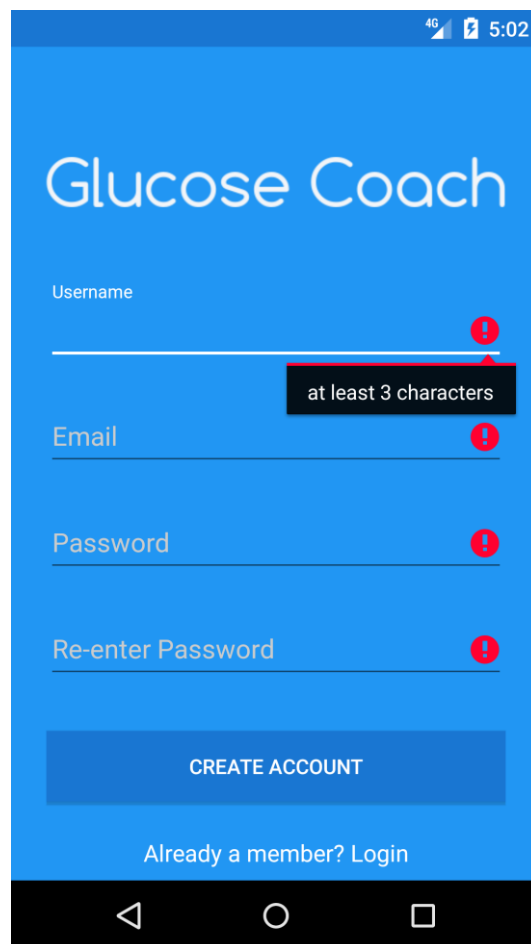


Figure 15: Signup screen showing possible errors

Other Screens

A logbook screen allows users to view and update the values of previous logs. Each log is uniquely identified by a date and time of day, either morning, afternoon, evening or night (Figure 18). Each log contains a user's blood glucose value, the insulin they took, the carbohydrates they consumed and any exercise they took.

A charts screen allows users to view their previous blood glucose results in a line chart (Figure 17). Users can tap on a point and it displays the value for that entry. The charts are horizontally scrollable

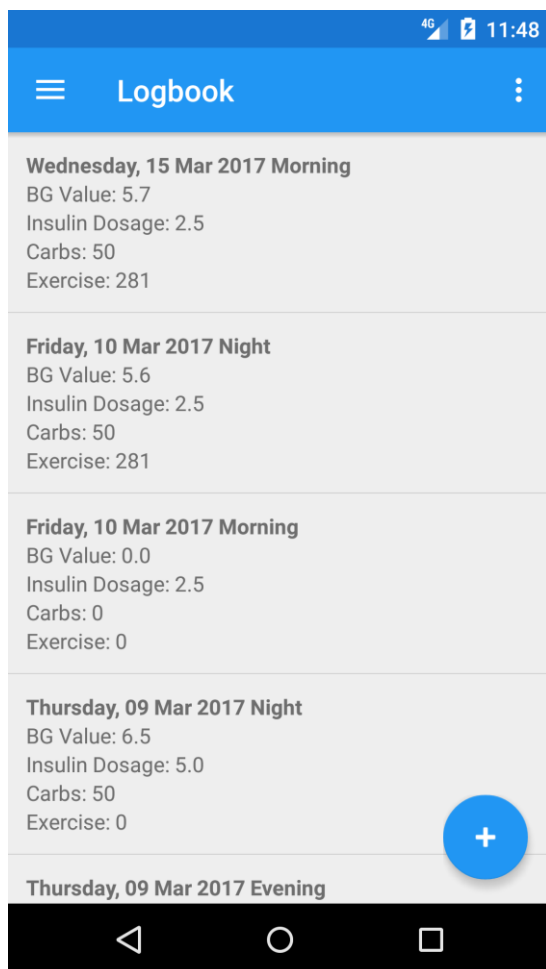


Figure 18: Logbook showing a user's logs

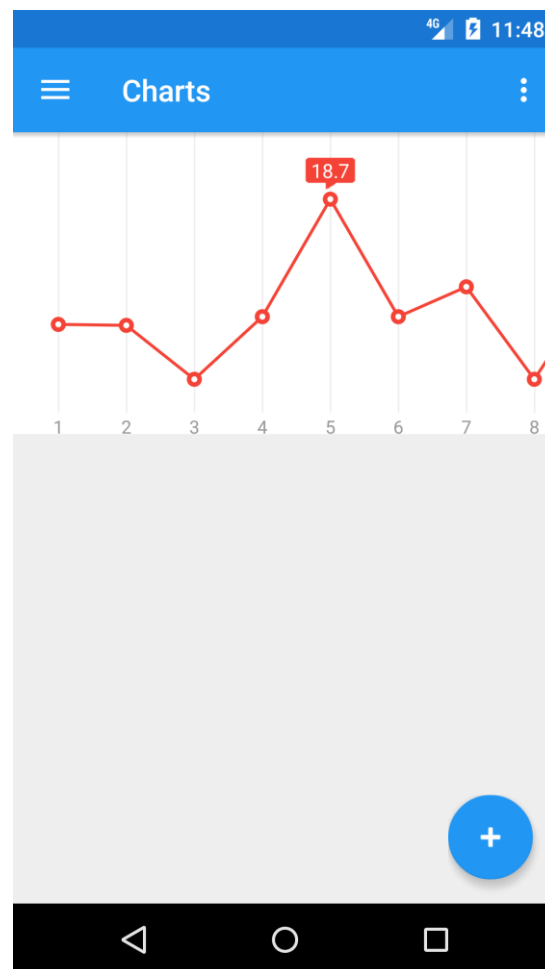



Figure 17: Line chart showing history of user's blood glucose results

Meter Tool UI

The blood glucose meter sync tool uses a simple UI. Users can sign in to their profile (Figure 19), view when they last synced data and then choose to sync. This allows the user to quickly sync data to their profile without needing to learn how to interact with the system.

The main screen UI is composed of a textbox displaying the current user, the last time they successfully synced data and a button allowing them to sync (Figure 20). Once a user

syncs their meter, the date updates to reflect the timestamp of the last synced record. The user cannot sync if no new records are available.



Meter Tool

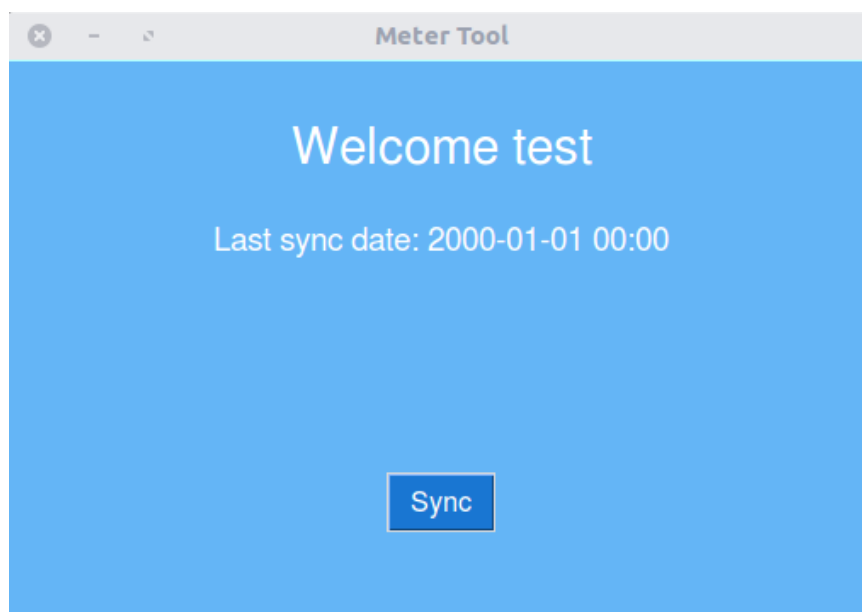
Glucose Coach

Username

Password

Login Close

Figure 19: Meter sync tool login



Meter Tool

Welcome test

Last sync date: 2000-01-01 00:00

Sync

Figure 20: Main screen with last sync date and sync button

c. Features and Use-cases

Register and Login

A user can create an account, and if they already have an account, they can then login. When users select the registration option, they are presented with a form that requires specific details. When users select the login option, they are prompted for a username and password. If a user has forgotten their username or password, they can then retrieve it by having an email sent to their address containing the info. These system interactions can be seen below (Figure 21).

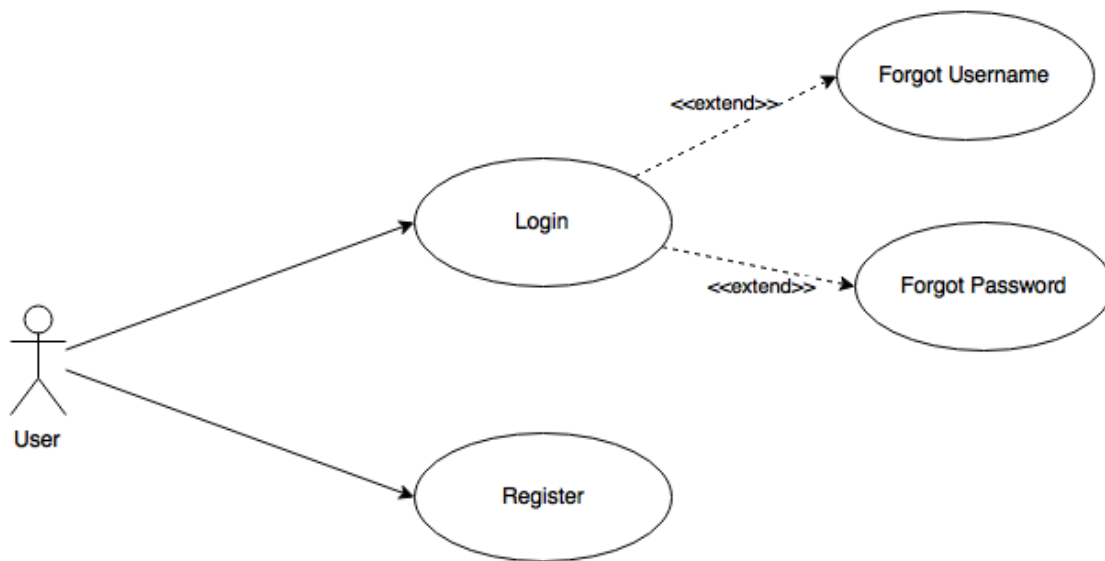


Figure 21: Use case showing user login and registration interaction with the system

Function	Purpose
Login (username, password)	Take in two strings, username and password. Returns a login token such as a cookie for the web app or stores session data in shared preferences in the Android app.
forgotUsername (email) and forgotPassword (username)	Take in a string and compares it to see if a matching username exists in the database. If yes, send an email to the associated email address, if not return error.
registerUser (username, password, email)	Take three strings as arguments. If any of the strings are not valid (after having regex validate them) then an error will occur.

Table 1: Functions that are invoked in the register/login use case

Enter a Reading

The main method of user input to the Glucose Coach is through the mobile app. There are 4 main types of data that the user can record, a blood glucose reading, an insulin dosage, any food eaten and any exercise taken (Figure 22). This data is grouped by time of day into morning, afternoon, evening and night. A user can also add a note to each blood glucose or insulin entry. A user can enter a new food type into the database if one does not already exist.

For example, a user could enter a blood glucose reading of 6.7 mmol/L, 4 units of quick-acting insulin, 40g of Cornflakes and a 60-minute cycle for the morning of 1/11/2016. They could then add a note to the insulin reading, “reduce insulin to carb ratio from 4:1 to 3:1 in the morning”.

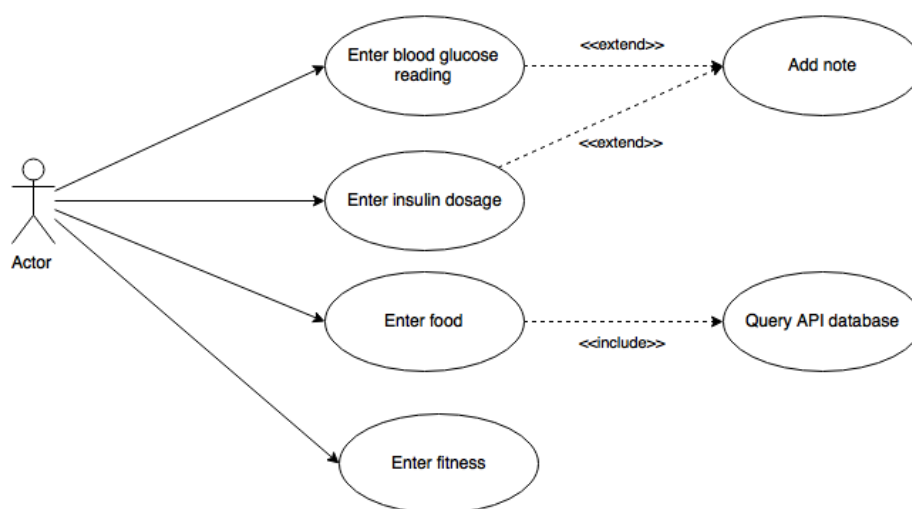


Figure 22: Use cases for entering data manually into the system

Function	Purpose
insertBGReading (userId, timeOfDay, value)	Insert a user's blood glucose reading into the database for a specified time of day.
insertInsulinDosage (userId, timeOfDay, type, value)	Insert a user's insulin dosage into the database for a specified time of day.
searchFoodDB (foodName)	Search the food database API for a string matching foodName. Return a string list of matching food items.
insertFood (userId, timeOfDay, foodItem, quantity) insertFood (username, timeOfDay, foodItem [])	Insert a user's type of food eaten with a quantity. Function overloaded by an array of foodItem's with corresponding quantities. foodItem is an object with attributes: name, energy, carbs and sugars.
enterExercise (userId, timeOfDay, fitnessType, intensity, duration)	Insert a new user workout for a specified time of day. The workout has an intensity and duration. Workouts are predefined and are suggested to user based on input.

Table 2: Functions that are invoked in the data entry use case

Automatic Meter Upload

A user is given the option of automatically uploading results from their Contour® Next USB blood glucose meter (Figure 23). The meter needs to be plugged into a user's computer where the tool prompts them to download their data. The tool UI is a basic form that displays various bits of information related to the data sync. It asks the user whether or not they want to sync their data, display a timestamp of when they last synced data and various other bits of information related to the user.

Hidden from the user is comprehensive error prevention and detection. This is necessary as data can only be read from a device by converting a HID report packet into a standard file, like CSV, and then analyzing the data. The tool can then compare when the data was last synced to a timestamp of data in the CSV file, and only sync the new data.

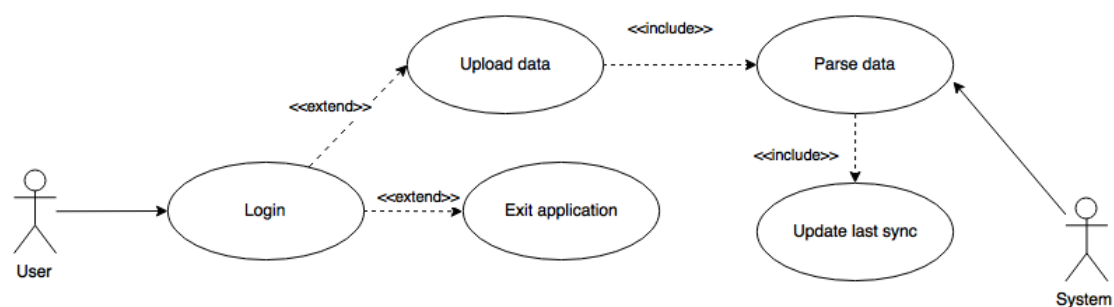


Figure 23: Glucose meter upload use case showing user interaction with the upload tool

Function	Purpose
Login (username, password)	A user must login before then can use the tool so data is tied to their account. They will to provide two strings, a username and password.
parseData (bytestream)	Convert byte stream that is read from USB meter and parse info, such as timestamp, then dump into a CSV file.
putData (userDateUrl)	Read data from CSV file and construct JSON containing additional info such as username and push this data to REST API URL.
setLastSyncDate (userUrl)	Set the last sync date for a user's meter sync. URL contains a username and date.
getLastSyncDate (userUrl)	Return the last sync date for a specified user.

Table 3: Functions that are invoked in the meter upload use case

Generate Suggestion

An insulin dosage recommendation is suggested to the user based on the data they are entering into the app (Figure 24). The suggestions are based on additional variables, such as the time of day and the user's previous results. If a user is happy with the suggested insulin dosage, they can save it to the database. Else, the user can enter their own value.

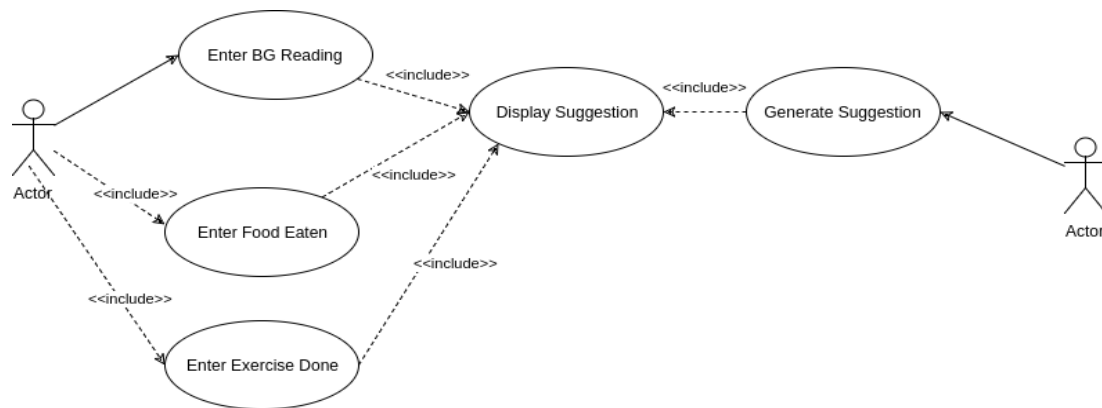


Figure 24: Use case diagram showing suggestions being given to user's

Function	Purpose
createSuggestion (userId, BGReading, carbohydrates, exerciseUnits)	Takes a username, their blood glucose reading (if any), the total carbohydrates the have eaten (if any) and the total amount of exercise they have done (if any).

Table 4: Functions that are invoked in the generated suggestion use case

Train Model

At any time, the user should be able to retrain their model. This is required as users personal blood glucose trends and insulin requirements can change over time. The data used to train the model should pass certain requirements in order to generate an accurate model. The data should not have missing blood glucose or insulin values and should not be older than 30 days (Figure 25).

By default, users are given a default model which has been generated using data generated to closely match a typical person with diabetes. This model is queried if no model yet exists for that user. Once a user creates a model, that model is used for future predictions.

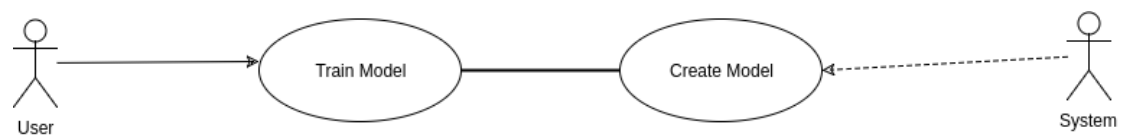


Figure 25: Use case for training model

Function	Purpose
createModel (userId)	The mobile client asks the API to create a new model for the user.
createModel (userId) - API side	The API selects data suitable for creating a new model for the user. If it successfully creates a model it returns a success message, else returns a failure message/

Table 5: Functions that are used when user chooses to train model

View Logbook

A user can see the data they have previously entered into Glucose Coach. In the mobile app, they can do this by selecting the logbook tab which presents the user with data from previous days. The user can then tap a log to view more detailed information on that day. A user can view trends in their readings by viewing charts detailing weekly and monthly trends (Figure 26).

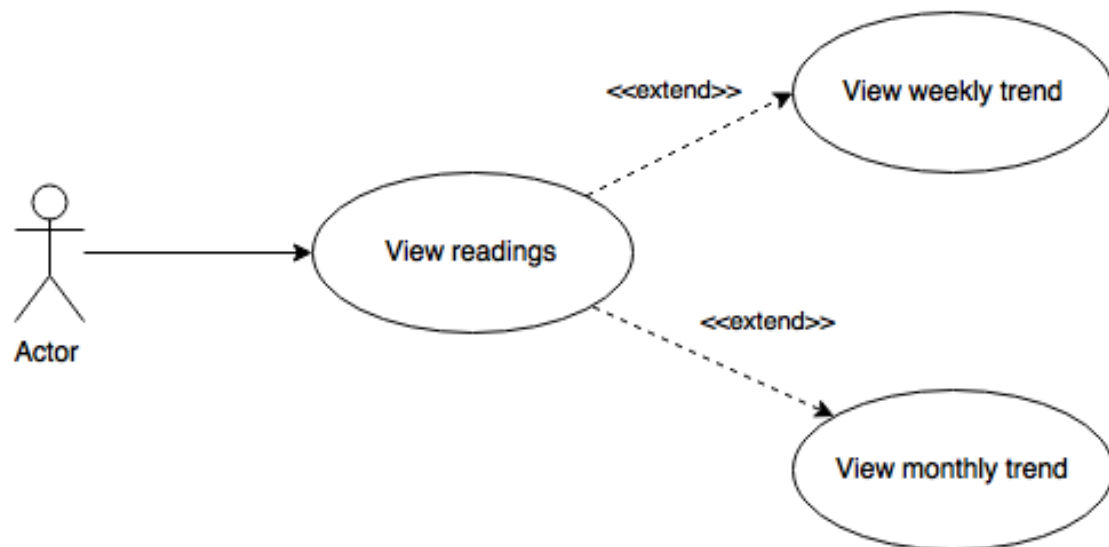


Figure 26: Use case for user viewing their previous results and options to view daily and monthly charts and statistics

Function	Purpose
getData (userId)	Get the various bits of data (blood glucose readings, insulin, food, exercise) for a specified user on a specified date.
displayWeeklyData (week)	With the fetched data, display data corresponding to a specified week. Data can then be displayed in a chart.

Table 6: Functions that are invoked in the view logbook use case

Edit Settings

A user is able to edit various settings related to the app and their personal requirements. The user can set the range of blood glucose readings that would constitute low blood sugar and high blood sugar. The user can set a profile image, their age, weight and height (Figure 27). These values can also be used to provide more accurate insulin recommendations for a specific user. As an example, weight has an effect on the daily recommended long term insulin dosage.

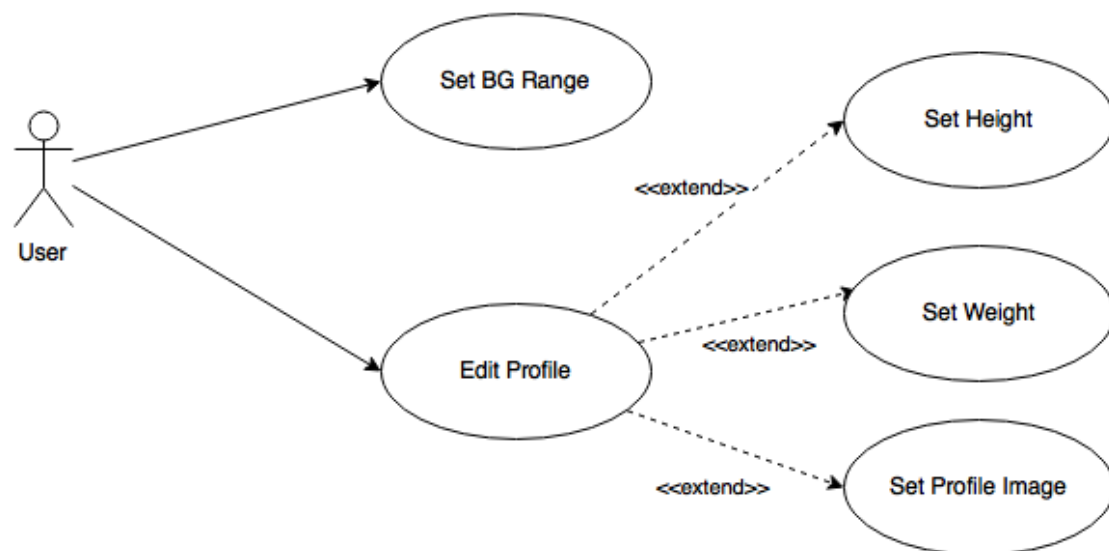


Figure 27: Use case diagram of user's options to edit various settings

Function	Purpose
setLowRange (userId, value) setHighRange (userId, value)	Update the user's local database with the values for their desired low and high blood sugar.
setHeight (userId, value)	Set the users height.
setWeight (userId, value)	Set the users weight.
setProfileImg (userId, value)	Set a profile image for the user.

Table 7: Functions that are invoked in the edit settings use case

d. Other Design Documents

Initial Database ERD

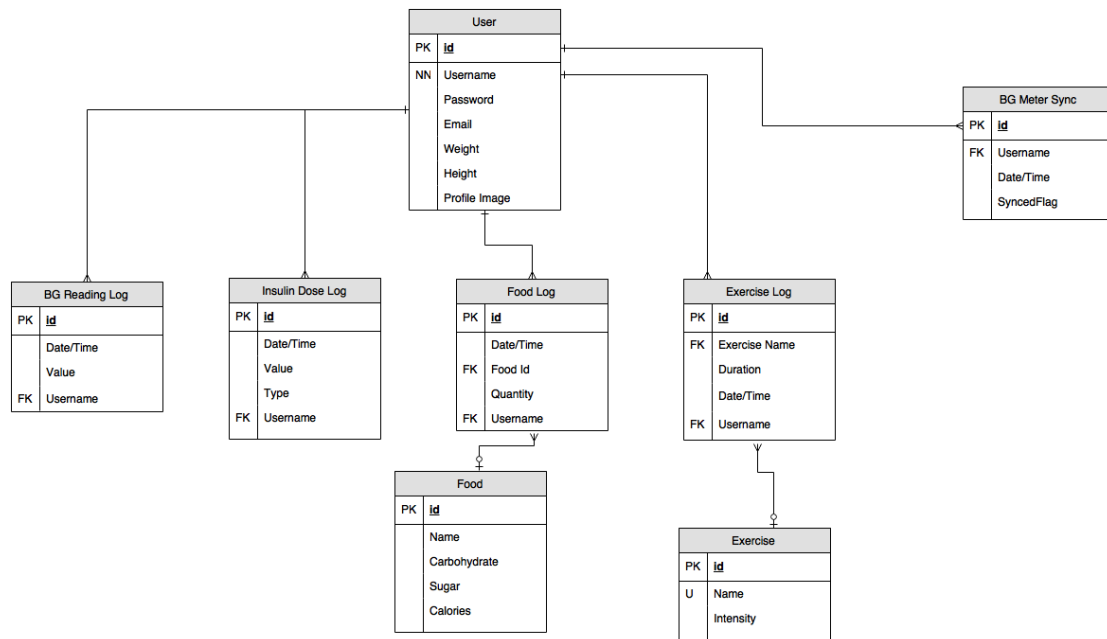


Figure 28: ERD showing the database entities that were initially used in the system

The ERD (Figure 28) was the initial database design created during the first design iteration of the project. The design was centered around a user's table that contained information related to a user. The password was going to be stored in plaintext and each username was going to have to be unique.

A user could log many blood glucose readings, insulin dosages, foods eaten and exercise done. Each food and exercise is unique. Only one of these readings can be associated with a single user. A food would contain sugar and calorie information and an exercise would have a corresponding intensity.

A users last BG meter sync is stored in the database. For error prevention, it contained a flag field that is checked if the data has been successfully pushed to the database.

Updated Database ERD

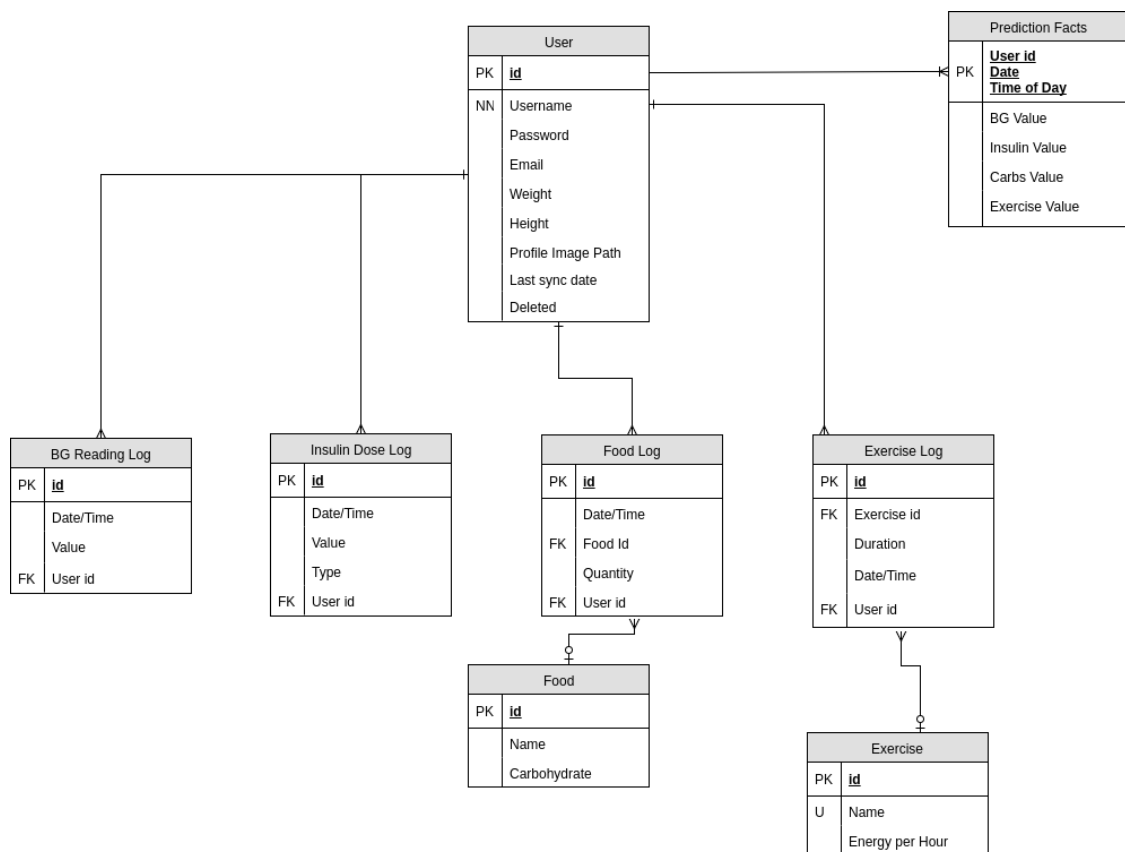


Figure 29: Updated ERD

After the first development iteration using the scikit-learn library and having implemented the basic framework of the android application and meter upload tool, the ERD was altered to reflect some new requirements (Figure 29). Various fields in the multiple logs tables and foods/exercises tables were simplified or removed and the BG meter sync table was removed altogether. A new prediction facts table was added.

The user table was updated to incorporate the last sync date of a meter. This allowed for removal of the BG meter sync table, as it was not needed. The meter upload tool was designed in a way that made the checking for errors within the tool possible.

Having done more extensive research on how food and exercise affects the daily routine of people with diabetes, these database tables were simplified to provide more accurate data. It was then possible for each exercise to have a name detailing its intensity with a corresponding energy per hour value rather than having intensity values for each exercise. For example, 'Cycling' at intensity level 2 could be replaced by 'Cycling at 23 km/h'. Similarly, foods do not require to have calorie or sugar information, just carbohydrate content.

The major alteration to the schema was the addition of a denormalized predictions fact table for holding users daily log data. Initially when creating machine learning models for users, the data required needed to be pulled from each of the four log tables. This required complex SQL queries and its performance quickly became a problem, particularly if a user

had many thousands of records. The new predictions fact table acts like a fact table in a data warehouse star schema in that it holds the measurements of the business process and is located at the center of the logs tables, which act like dimension tables.

Each row in the prediction facts table is identified by a user id, a date and a time of day (Figure 31). Time of day is an integer value from 1 to 4 which corresponds to either morning, afternoon, evening or night. When a user submits an entry into one of the log tables (Figure 30), a trigger fires for that log table and updates the appropriate column in the prediction fact table row with the new value.

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
◇ id	int(11)		NO			select,insert,update,references	auto_increment
◇ user_id	int(11)		NO			select,insert,update,references	
◇ bg_value	float(3,1)		YES			select,insert,update,references	
◇ bg_timestamp	datetime		YES			select,insert,update,references	

Figure 30: Blood glucose log table

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
◇ pf_date	date		NO			select,insert,update,references	
◇ pf_time_of_day	int(11)		NO			select,insert,update,references	
◇ bg_log_id	int(11)		YES			select,insert,update,references	
◇ ins_log_id	int(11)		YES			select,insert,update,references	
◇ food_log_id	int(11)		YES			select,insert,update,references	
◇ exer_log_id	int(11)		YES			select,insert,update,references	
◇ user_id	int(11)		NO			select,insert,update,references	
◇ bg_value	float(3,1)	0.0	NO			select,insert,update,references	
◇ ins_value	float(2,1)	0.0	NO			select,insert,update,references	
◇ food_value	int(11)	0	NO			select,insert,update,references	
◇ exercise_value	int(11)	0	NO			select,insert,update,references	

Figure 31: Prediction facts table

Triggers

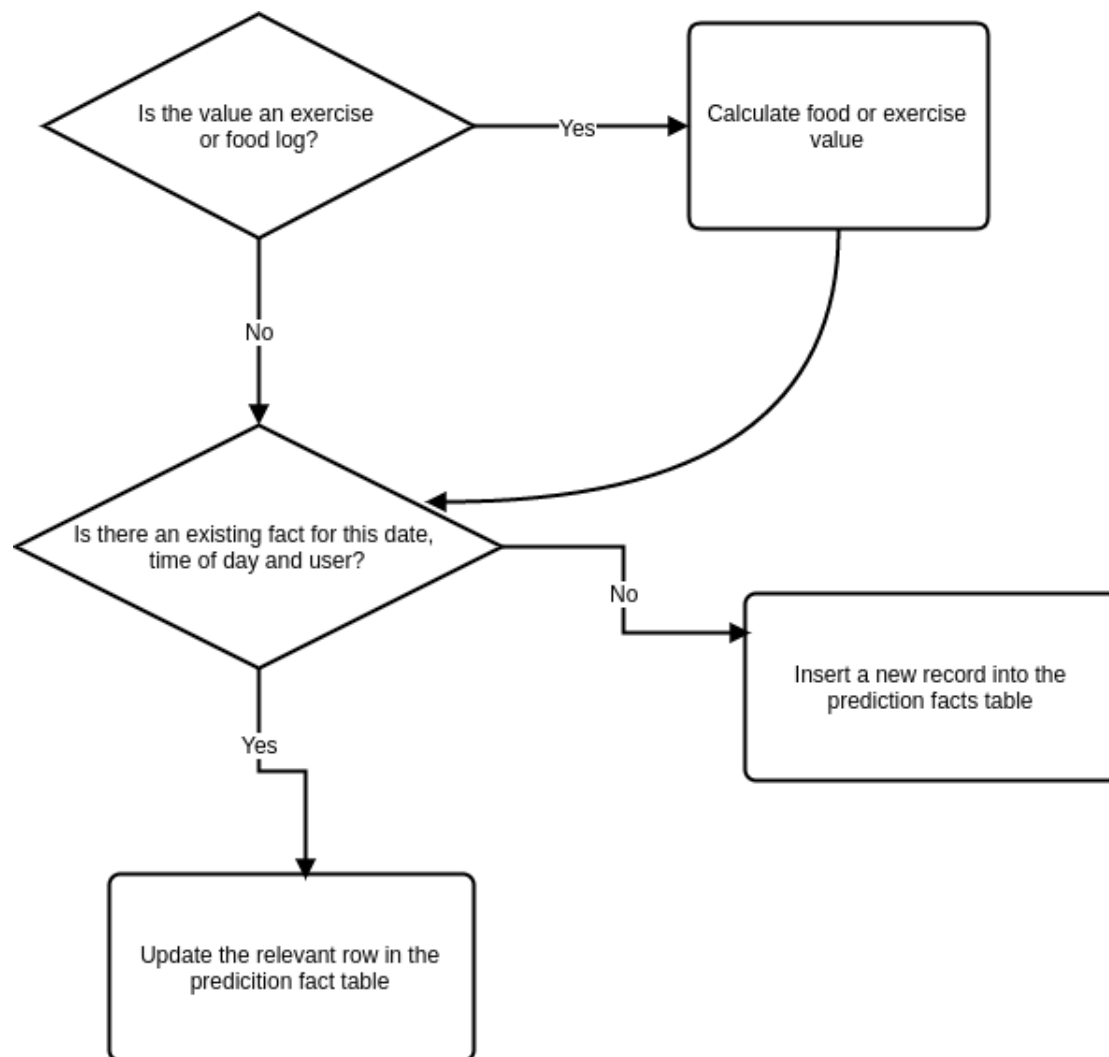


Figure 32: Trigger structure for logs table

The flow chart (Figure 32) details the logic of the triggers needed to add or update entries in the predictions fact table as a result of changes needed in the second design phase. Each of the four log tables has its own trigger. When a value is added to one of these tables, the trigger fires.

If a fact already exists for the date and time of day that the value has been submitted, then the row is updated, else a new row is added. If the value is an exercise log then the exercise value is calculated by multiplying the duration of the exercise submitted by its energy per hour value from the exercises table. If it is a food log, then the carbs contained in the food per 100 g is multiplied by the quantity in the food log to give the food value.

Architecture and Tools

a. System Architecture

Glucose Coach uses a client-server architecture. Clients communicate with the server through a RESTful service (Figure 33). The server contains the relational database, a Flask server containing the RESTful service and the machine learning system. Users can plug in their blood glucose meters through a PC client meter upload tool via USB and push data to the database through the API.

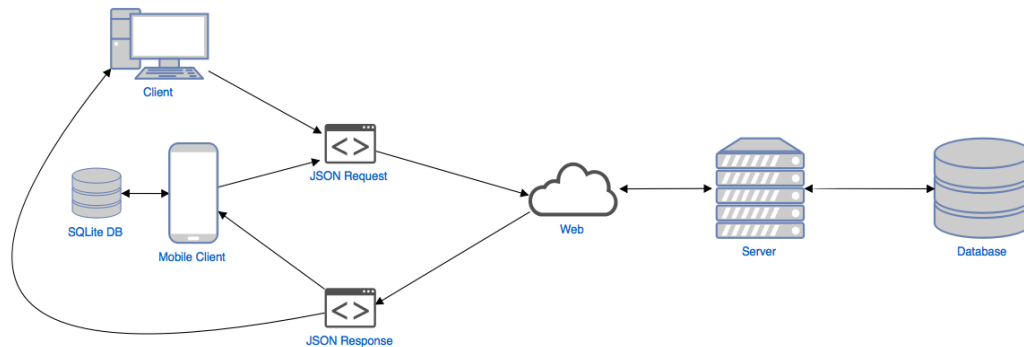


Figure 33: Technical architecture diagram showing the different parts of the system and how they interact

Clients

Clients take the form of an Android mobile app and a PC client tool running on Ubuntu. Users sign up/login and can then add data and view predictions through the mobile app. Users can plug in their blood glucose meter into a PC running the meter upload tool which allows them to push data from their device to the database.

The Android app forms the bulk of communication between the user and the database. The mobile client utilizes Android's SharedPreferences local key-value database for additional storage. Users can submit logs to the database, and retrieve predictions from the machine learning system through the API.

Server

The Flask server is running in a Docker container which is running on an AWS EC2 instance. The MySQL database is running on an AWS RDS instance. All client requests are made through the REST API running on the Flask server. For development purposes, the Flask server can be run locally using a Python virtual environment.

The Flask server handles all authentication by validating usernames/hashed passwords and returning a hashed token. The Flask server handles the machine learning component of Glucose Coach by running data through the scikit-learn libraries. Users can ask the API to generate a new insulin prediction or to generate a new completely new model for predictions.

JSON

To allow for the separation of the database service and the client, a RESTful service is used. JSON is the language used for communication between clients and the API. Clients can send data to the database as a JSON string encoded in the body of a request and receive data as JSON, which is then parsed (Table 8). This uniform interface is intended to make it easier to implement interactions between the mobile app/PC tool and the API.

URL Sent	JSON Response
GET http://[hostname]/users/alexkiernan	{ "id": 1, "name": "Alex Kiernan", "username": "alexkiernan", "email": "C13451458@mydit.ie", "height": "177", "weight": "65" }
GET http://[hostname]/users/bgreadings	[{ "userId": 1, "id": 1, "value": 5.7, "time": "13:37" }, { "userId": 1, "id": 2, ... }]

Table 8: Examples of various request to the REST API and the JSON responses

b. Development Tools

Android Studio

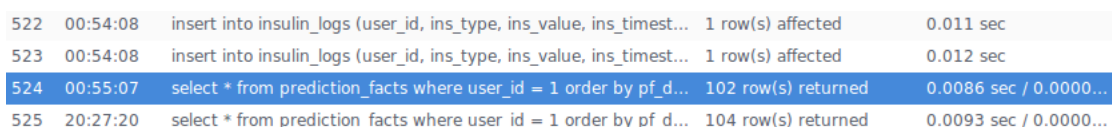
Android Studio is the official IDE for Android application development and was the primary development platform for the Glucose Coach mobile app. It features extensive developer features, such as Git integration, code assistance, and powerful testing and debugging tools. The 2.0 version recently added instant run which allows for rapid application deployment as it only pushes changes made to an APK.

A major advantage of using Android Studio is its built-in emulator. With this emulator, different versions of Android were tested with Glucose Coach to identify any potential problems that might occur outside of the test environment. The emulator was also given root access to allow for in depth debugging of system tools, such as user information stored in SharedPreferences.

MySQL Workbench

MySQL Workbench was used to remotely query and develop the live MySQL database hosted on an Amazon RDS instance and the development database used locally. MySQL workbench was chosen as it offers an easy to use and well-designed interface as well as offering useful shortcuts for executing queries on tables.

A feature of MySQL Workbench that was particularly useful was its ability to visually detail the performance of a query (Figure 34). After a query executes, the time it took to return a result is displayed, as well as the tables affected by the query. Combining this information with the results of an explain plan allowed for query optimizations which prompted changes to design decisions in the database.



522	00:54:08	insert into insulin_logs (user_id, ins_type, ins_value, ins_timest...	1 row(s) affected	0.011 sec
523	00:54:08	insert into insulin_logs (user_id, ins_type, ins_value, ins_timest...	1 row(s) affected	0.012 sec
524	00:55:07	select * from prediction_facts where user_id = 1 order by pf_d...	102 row(s) returned	0.0086 sec / 0.0000...
525	20:27:20	select * from prediction_facts where user_id = 1 order by pf_d...	104 row(s) returned	0.0093 sec / 0.0000...

Figure 34: Example of MySQL Workbenches query performance information

PyCharm

As multiple components of Glucose Coach heavily utilize Python, a powerful, easy to use and Python specific IDE was required. PyCharm Professional was chosen based on its recognition within the Python community and the availability of a professional license. PyCharm provides built-in developer tools, code assistance, version control implementation, and debugging features. These features made development smoother when it came to creating and testing the sample user generator scripts.

PyCharm made it possible to create and manage virtual environments within the IDE. This made the management of the multiple Python projects within Glucose Coach easier than if everything to be done through the console.

Postman

Postman is a GUI platform that allows for the documentation and testing of APIs. With Postman, HTTP routes of an API can be defined and data can be supplied in the request bodies to test if a particular endpoint works as expected. An account can be made with Postman to sync collections of endpoints as well as any ad-hoc requests that might be made.

Each of the endpoints of Glucose Coach are given a folder in Postman containing various HTTP methods implanted in the API (Figure 35). The logging endpoints contained the GET, GET all, POST and PUT methods for getting a log, getting all logs, adding a new log and updating an existing log. An endpoint also exists for acquiring an access token (which requires a username and password) as well as getting an insulin prediction and retraining a model.

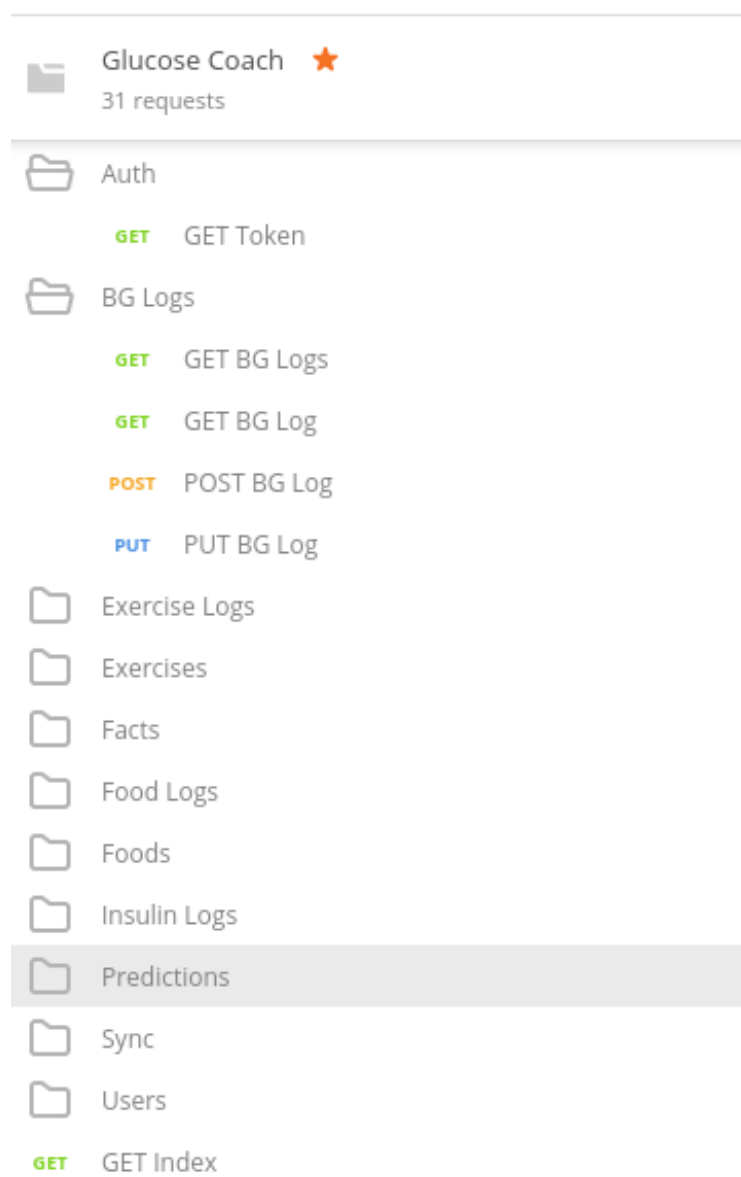


Figure 35: Postman endpoints for testing Glucose Coach API

c. Source Code Layout

Git and GitHub were used for managing the various components of the project. Several private repositories were created for each of the components which allowed for detailed change tracking. Commits were made frequently with the idea of having a new, working feature after every commit. A development branch was initially created for each repository to be merged with the main branch once a feature had been implemented successfully.

	Android App	API	Meter Upload Tool	SQL Scripts	Machine Learning Tests
Commits	88	94	24	20	22
Additions	9957	4311	1091	232	822

Table 9: Details on Glucose Coach Git repositories

Development

As described in the design methodology section, a personal extreme programming approach to this project was taken. Glucose Coach development was broken into week long iterations (Appendix A). At the beginning of each week, tasks were selected for the current iteration. The initial weekly iterations did not contain releasable builds of the project, but focused on getting the frameworks and boilerplate code in place as well as testing what would and wouldn't work.

At the end of each iteration, a quick retrospective was carried out. The assessment would look at whether or not the goals had been met for that week and any problems that had occurred would be noted. If a problem had occurred, the reasons that caused it were examined and plans were put in place made to avoid similar problems occurring in the future.

As mentioned in the solution definition section, this project was broken into multiple components. Subsequently, these components were further broken down into smaller parts. This allowed each component of the project to be built iteratively in parallel during each iteration. It also allowed for effective timing of development of each part. In the event of delays occurring, timeframes were updated to reflect this.

The components of Glucose Coach are:

- Android Mobile Application
- Flask API
- Machine Learning System
- Meter Upload Tool

a. Mobile App

Overview

Based on research done into other existing systems and after having decided on the functionality required in the requirements phase, the following tasks were drawn up during the planning phase:

- Create an “Add Blood Glucose Log” activity to allow for blood glucose logging.
- Create an “Add Insulin Dosage Log” activity to allow for insulin dosage logging.
- Create an “Add Carbohydrates Consumed Log” activity to allow for the selection of
- Create an “Add Exercise Done Log” activity where users can select the exercise they did and its duration.
- Implement the logic for the main screen fragment where users can see their data for that time of day at a glance and ask for an insulin prediction.
- Create a “User Sign-in” activity to allow users to sign in and handle the logic of a user session, such as handling tokens.
- Create a “User Sign-up” activity, add validation for user information.
- Create a “Logbook” fragment that allows users to view their previous results.
- Create a “Charts” fragment that allows users to see their previous blood glucose results in a line chart.
- Clean-up the UI’s and make sure they conform to good design principles.
- Refactor network request code to ensure efficient operation.

Beginning Implementation

The first step into implementing the Glucose Coach application was becoming acquainted with the current best practices involved in designing Android applications. Research was done in the initial iteration design phase into the best ways to implement the various requirements. Research included reading blogs about people’s experiences, following tutorials on implementing various design patterns and libraries and listening to podcasts, such as *Fragmented*, on the newest ways of solving Android specific problems.

After the initial research phase, the main screen and an initial logging activity were the first parts of the app to be designed and implemented. Based on lessons learned and mistakes made in implementing these components, future development was made easier. In addition, during this phase modifications were made to the appearance of the user interface based on feedback from testers.

Application Structure

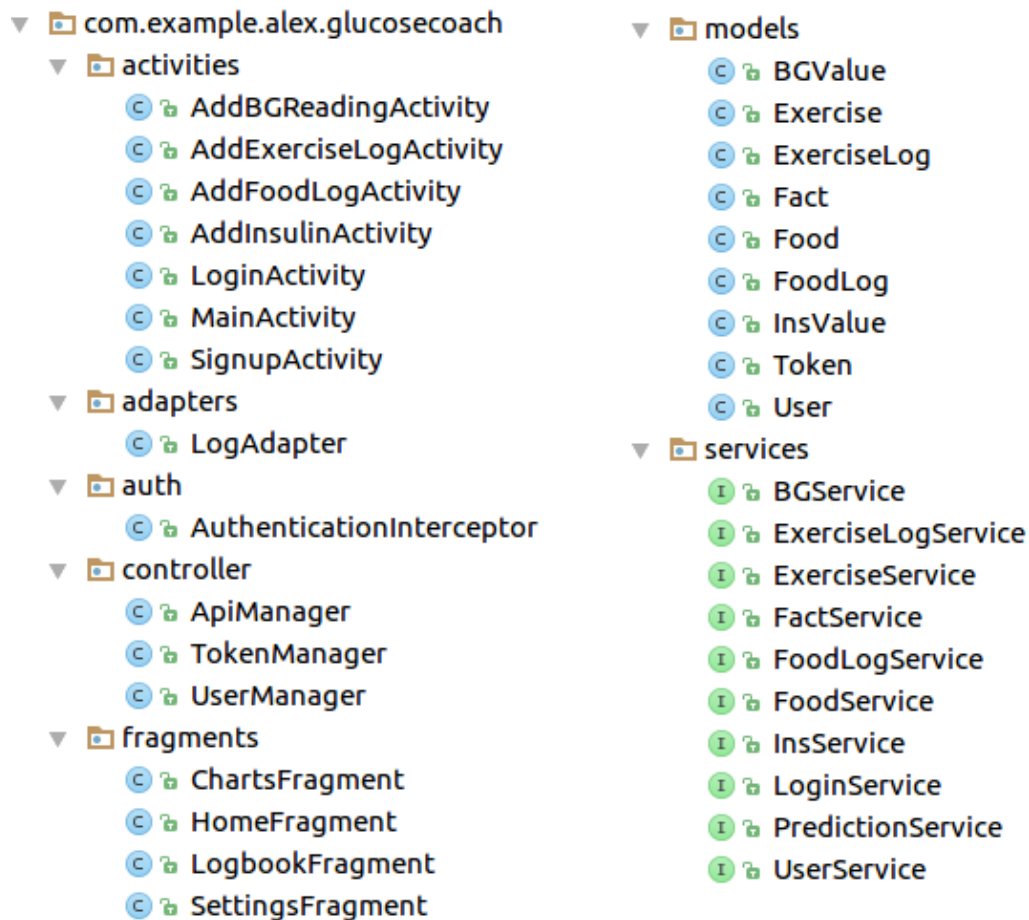


Figure 36: Project structure of Android App

Figure 36 is an overview of the Glucose Coach application source code. The application uses a basic model-view architecture where user interface logic is interspersed with business logic. This architecture was chosen as it was simple to implement and maintainable for the scale of this application.

The activities and fragments packages contain the logic for navigating between screens, handing user input and updating the user interface. `MainActivity` is the initial class loaded by the application. From here, the various fragments are initialized and listeners created. If a user is not signed in they are taken to the login activity. From the login activity, the user can choose to sign up for an account or if an account exists already, login to the app. When logged in, the home fragment will initialize and user information is stored locally.

The controller package offers helper classes to which work can be delegated. These classes are designed to be small, tight and focused. They contain functionality that aids with code reuse in the app. The API manager class provides routing for interacting with the API through the use of methods which return interfaces. The user manager class acts as an interface for data stored in `SharedPreferences` that is related to the user. Similarly, the token manager class provides helper methods for interacting with token objects stored in `SharedPreferences`, such as adding and deleting tokens.

The models package contains information related to the business entities in the app. Each model that has a database entry, such as User, has a label mapped to each of its attributes which is linked to the associated column name in the database. These specific models were generated by using the APIs JSON response to form plain old java objects. Each model has the standard getters/setters as well as toString methods. Some models have additional methods for additional functionality.

Development - Networking

The first iteration of development was concerned with establishing network communications. The initial requirement was the creation of a REST client within the app to handle API interactions. The second was a method of handling HTTP requests in a way that would correctly handle errors.

The library chosen to handle API communication was Retrofit 2. Retrofit is a HTTP client for Android that makes it relatively easy to retrieve and upload JSON via a REST based service. It is a type-safe HTTP client, which means that the developer is only concerned with the semantics of queries that are sent over the network, rather than the details of how to construct URLs, specifying parameters, etc.

```
public class ApiManager {

    private static String BASE_URL = "http://ec2-34-248-62-100.eu-west-1.compute.amazonaws.com:5000/glucose_coach/api/v1.0/";
    //private static String BASE_URL = "http://192.168.1.101:5000/glucose_coach/api/v1.0/";

    private static OkHttpClient.Builder httpClient = new OkHttpClient.Builder();

    private static Retrofit retrofit;

    private static Retrofit.Builder builder =
        new Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .addCallAdapterFactory(RxJava2CallAdapterFactory.create());

    private String authToken;
    private AuthenticationInterceptor interceptor;

    public ApiManager() {
        retrofit = builder.build();
    }

    public ApiManager(String token) {
        this.authToken = Credentials.basic(token, "unused");
        this.interceptor = new AuthenticationInterceptor(this.authToken);

        httpClient.addInterceptor(interceptor);

        builder.client(httpClient.build());
        retrofit = builder.build();
    }

    public ApiManager(String username, String password) {
        this.authToken = Credentials.basic(username, password);
        this.interceptor = new AuthenticationInterceptor(this.authToken);

        httpClient.addInterceptor(interceptor);

        builder.client(httpClient.build());
        retrofit = builder.build();
    }

    public LoginService getLoginService() {return retrofit.create(LoginService.class);}
```

Figure 37: Source code for Android REST client –controller/ ApiManager.java

Figure 37 exhibits the API manager code. This class allows for a single point within the application for network calls to be processed. This allows for quick adjustments to network settings, particularly when swapping between development and live APIs. Its main purpose is to supply endpoint services to clients, the example shown demonstrates the login service.

The API manager provides two constructors. One is for initial logins when a user supplies a username and password, and the other for when the user has acquired a token. When an API manager is created, a retrofit object is instantiated and a custom interceptor (Figure 38) is attached to it. This interceptor sets the authorization header value for any HTTP request executed.

```
public class AuthenticationInterceptor implements Interceptor {

    private String authToken;

    public AuthenticationInterceptor(String token) {
        this.authToken = token;
    }

    @Override
    public Response intercept(Interceptor.Chain chain) throws IOException {
        Request original = chain.request();

        Request.Builder builder = original.newBuilder()
            .header("Authorization", authToken);

        Request request = builder.build();
        return chain.proceed(request);
    }
}
```

Figure 38: Custom interceptor for authentication headers – auth/AuthenticationInterceptor.java

Development – Token/User Manager

A helper class for handling user and token management was created as a result of refactoring code during an early iteration. The logic for storing tokens and user objects in SharedPreferences local storage was found to be repeated in multiple locations and quickly made code bloated and unreadable.

The solution was to create classes which could be instantiated anywhere in the app and have methods for handling the process of setting/retrieving the objects. Each of the classes takes a context object as its only parameter. After the object is created, methods can be invoked on each to perform various operations. These operations are asynchronous, so as not to impede performance of the main process.

Development – Login/Sign up

The second iteration of development added login and signup functionality to the application. Users are redirected to the login activity if their token stored in SharedPreferences is not valid (either expired or does not exist). Appropriate error handling is implemented for both activities, such as passwords being of certain length and email passing a regex validation.

As discussed above, network calls are handled using a retrofit client. The client implements callbacks so that network requests are handled asynchronously. If the network request returns a result, the *'onResponse'* callback is triggered. Else, the *'onFailure'* method is called. Below is an example of the call to the login service (Figure 39).

```
apiManager = new ApiManager(username, password);
LoginService loginService = apiManager.getLoginService();
Call<Token> call = loginService.basicLogin();
call.enqueue(new Callback<Token>() {
    @Override
    public void onResponse(Call<Token> call, Response<Token> response) {
        progressDialog.dismiss();

        if (response.isSuccessful()) {
            Token token = response.body();
            Log.d("token", token.getTokenValue());
            onLoginSuccess(token, username);
        } else {
            onLoginFailed();
        }
    }
});

@Override
public void onFailure(Call<Token> call, Throwable t) {
    progressDialog.dismiss();
    onLoginFailed();
    Log.d("Server connect error", t.getMessage());
}
});
```

Figure 39: Login service call to return a token – activities/LoginActivity.java - line 92

Development – User Interfaces

Subsequent iterations of development focused on implementing the logic for the user facing portions of the application. During the early planning phase, it was decided that fragments would be used to hold primary information screens, such as the home screen, the logbook, charts and settings. A main activity would be used to encapsulate these fragments and additional activities would be used for displaying views within fragments. This allowed for a more modularized and easier to extend structure.

The home screen was one of the first implementations. The requirements for the main screen as defined in the initial iteration was that:

- Users can view their recent data at a glance.
- Users can request an insulin prediction, which they can choose to accept or reject.
- A floating action button is available to allow users to enter data.
- The screen is fast and easy to read.

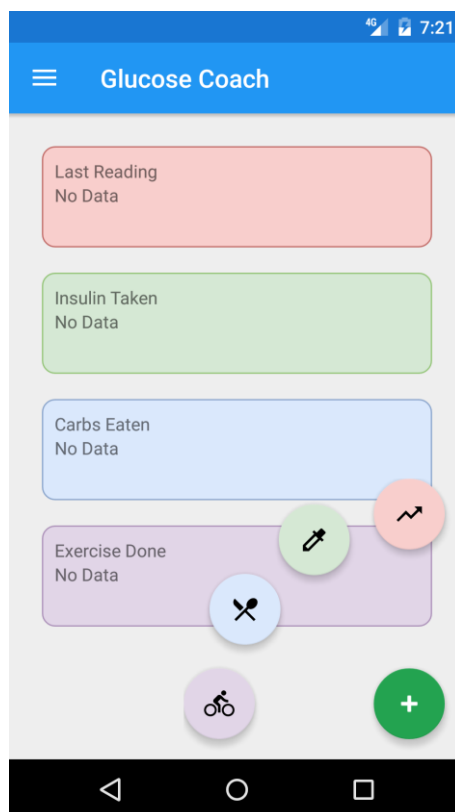


Figure 40: Initial home screen

After the initial iteration, feedback was received on its design and functionality (Figure 40). The pastel theme was found to not be eye catching, and users did not like its look and feel. Additionally, the green floating action button in the bottom right was found to not offer any sort of feedback to the user when clicked.

During the subsequent iteration, these problems were reviewed and the UI was redesigned (as can be seen in Figure 13). After consulting the material design guidelines, cards were chosen as the most suitable way to display data on the home screen. Cards are defined as “a sheet of material that serves as an entry point to more detailed information.” Icons were used instead of text to identify the bits of information on the home screen. The colors of the app were updated to a more bold and eye-catching theme which followed the material design guidelines.

In addition, the floating action button was redesigned to provide more feedback to the user. When clicked, the background becomes greyed out in order to focus the user’s attention on the options. When the user touched away from the menu, the menu would collapse. This resulted in making the menu functionality more intuitive.

The logbook fragment is responsible for displaying past user data. During the initial iteration, it was found that network calls lagged slightly when pulling records of more than 500 records. The solution was to utilize an observable stream based networking solution using the RxJava library to asynchronously speed up network requests (Figure 41).

```
factService.getFactsObservable(_userManager.getUsername())
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Observer<List<Fact>>() {
        @Override
        public void onSubscribe(Disposable d) {

        }

        @Override
        public void onNext(List<Fact> value) {
            listView.setAdapter(new LogAdapter(getActivity(), value));
            setListViewListener();
        }

        @Override
        public void onError(Throwable e) {
            Log.d("fact_log", "Error fetching facts");
        }

        @Override
        public void onComplete() {

        }
    });
```

Figure 41: Source code for the logbook observable - fragments/LogbookFragment - line 52

In RxJava, an Observable is a class that emits a stream of data or events, and a Subscriber is a class that acts upon the emitted items. Once a stream has been received, a callback is triggered and the response can be used in the application. Using observables resulted in a noticeable improvement in performance in network requests and resulted in the logbook fragment loading almost instantly.

The charts fragment is responsible for displaying a user's previous blood glucose results in a line chart. The chart is interactive in that it allows users to tap on a point on the graph to display its value (Figure 17). The initial iteration used a bar chart that grouped data by times of day. However, users found this difficult to understand so a line chart was found to be a suitable alternative.

The settings fragment was originally going to host standard settings included with apps. However, due to time constraints, the model retraining option is the only option within this menu. When the model retraining button is pressed and the user's intent confirmed, a request is sent to the API to retrain the users model. If successful, the user is notified.

The four data entry activities are broadly similar in implementation. Each of the activities have their own validation logic. Regex is mostly used to ensure that user inputted data conforms to requirements (Figure 42). After being verified, the data is sent to the API where it will be parsed further and then processed by the database.

```
public boolean validate() {
    boolean valid = true;

    String bgValue = _bgValueText.getText().toString();
    String bgTime = _bgTimeText.getText().toString();

    if (bgValue.isEmpty() || !bgValue.matches("^\\d{0,2}(?:\\.\\d)?$")) {
        _bgValueText.setError("enter a valid value");
        valid = false;
    } else {
        _bgValueText.setError(null);
    }

    if (bgTime.isEmpty() || !bgTime.matches("^[01]\\d|2[0-3]):?([0-5]\\d)$")) {
        _bgTimeText.setError("enter a valid value");
        valid = false;
    } else {
        _bgTimeText.setError(null);
    }

    return valid;
}
```

Figure 42: Example of add blood glucose log validation – activities/AddBgReadingActivity – line 114

b. API

Overview

From the start of the project, an API was needed to link the mobile client and desktop client to the database and to provide login functionality. The API also needed to provide security, token generation and communication with the machine learning system. Based on research done, the flask microframework was chosen. Flask is designed to provide a minimal basic server with the ability to add extensions were needed.

Based on research and planning done in early iterations, the following requirements were identified:

- Provision of endpoints for adding, reading, updating and deleting logs and users.
- Creation of tokens for API access when given a username and password.
- The ability to hash user passwords.
- To return appropriate errors to the client.

Beginning Implementation

To implement the requirements listed above, research was conducted in the initial iteration into best practices. The main source of information was the Flask documentation. This documentation provides extensive details on each Flask function as well as providing comprehensive tutorials on how to implement basic Flask APIs.

The initial iteration was focused on creating much of the boilerplate code. The subsequent iterations were focused on developing in parallel with the Android app to provide functionality. Identifying mistakes made and learning from them in early iterations allowed for smoother integration of components in later iterations.

Application Structure

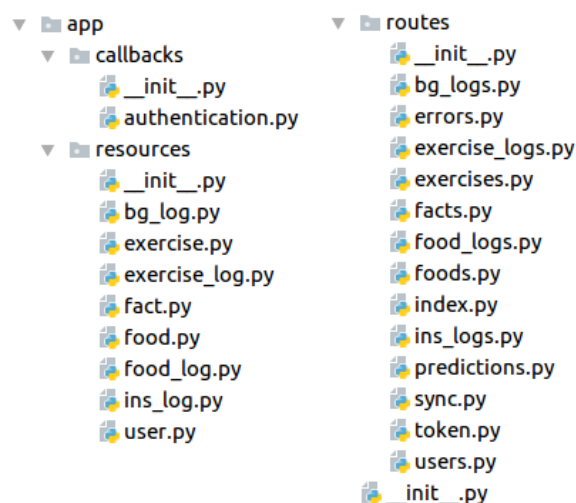


Figure 43: Source code layout of API

Figure 43 details the application layout for the Glucose Coach API. Flask leaves the organization of an application up to the developer. Application logic can either be contained in one single file, or spread across multiple packages. RunServer.py is the main entry point for the application. Here, a WSGI server is initialized and can begin serving clients. Flask configuration is loaded from a private config.py file which is required to run the application.

The API is comprised of two main packages. The resources package contains the business entities that reside in the database. The routes package contains the endpoints for interacting with the classes contained in the resources package. A third package contains authentication handling.

Development

Each class in the resources directory is mapped to a table in the database. The library used to handle object relational mapping is SQLAlchemy. SQLAlchemy was chosen as it is well regarded within the Flask community, is open source and provides a powerful and easy to use ORM.

The use of an ORM removes the necessity of writing tedious and error-prone raw SQL statement which are inflexible and hard-to-maintain. Figure 44 shows an example of the mapping of a blood glucose log class to the equivalent table in the database.

```
class BGReading(db.Model):
    __tablename__ = 'bg_logs'
    id = db.Column('id', db.Integer, primary_key=True)
    user_id = db.Column('user_id', db.Integer)
    bg_value = db.Column('bg_value', db.Float)
    bg_timestamp = db.Column('bg_timestamp', db.DateTime)

    def serialize(self):
        return {
            'id' : self.id,
            'user_id' : self.user_id,
            'bg_value' : self.bg_value,
            'bg_timestamp' : self.bg_timestamp
        }
```

Figure 44: Example of a blood glucose log class with mapping to the database - resources/bg_log – line 9

Routing was handled using Flask's built in router. A route decorator function allows Flask to bind a URL to a function (Figure 45). This allows for the straightforward injection of additional functionality to one or more functions. Routes are designated as one of the HTTP methods, GET, PUT, POST or DELETE.

```
@app.route('/glucose_coach/api/v1.0/users/<string:user_name>/bgreadings', methods=['GET'])
@auth.login_required
def read_all_bgs(user_name):
    user = User.query.filter_by(username=user_name).first()
```

Figure 45: BG Logs endpoint with route and auth decorator – routes/bg_logs – line 12

Authentication was also implemented as a decorator function. For security, any endpoints that expose personal user information require this decorator function. The function will first try to authenticate a token. If that fails, it then will try to authenticate credentials and if that fails, it will return a 401 'access denied' status. A configuration file is necessary to run the server. The config file requires a database location and a secret key.

c. Machine Learning System

Overview

A machine learning system that can be trained to output accurate insulin predictions is a central component of Glucose Coach. During the initial iterations, multiple machine learning systems were tested and evaluated. The criteria for evaluation included; time to learn, ease of use, documentation, performance and the ability to persist models generated. The dataset used can be found in Appendix C.

The dataset was generated using a python script to simulate the daily routines of a person with type 1 diabetes. The parameters for the data were derived from research done in the above research section and is heavily based on the UCI diabetes dataset [11]. The script starts by generating a random blood sugar value, exercise value and carbohydrates value. An insulin value is then generated based on the other three parameters.

After the initial research phase, the three systems that were chosen to be evaluated where:

- NuPIC: Numenta Platform for Intelligent Computing
- Amazon Machine Learning
- scikit-learn

NuPIC

NuPIC uses a process called swarming to generate the best model for a given dataset. Swarming works by creating and trying multiple different models on a dataset and outputting the parameters the model that performed the best (generated the lowest error score) [12]. NuPIC is similar to traditional artificial networks, but is based around HTM theory which aims to implement how the brain uses time to affect decisions.

During the initial iteration, a popular tutorial was followed for implementing NuPIC predictions. This tutorial was then adapted to work with a standard Type 1 Diabetic dataset [Appendix C]. This dataset contained a time of day (1, 2, 3 or 4), blood glucose value, food value, exercise value and insulin value. The aim of the using this dataset with NuPIC was to produce an accurate insulin value prediction.

Model generation for NuPIC running at a medium accuracy score took about 3 minutes on a high-performance desktop computer. The output of the model generation swarming is a text file that completely describes the components of the model and its parameters. The accuracy of the model was then tested by running the model against the dataset to provide a comparison of predictions to the actual data. The results can be seen in Figure 46.

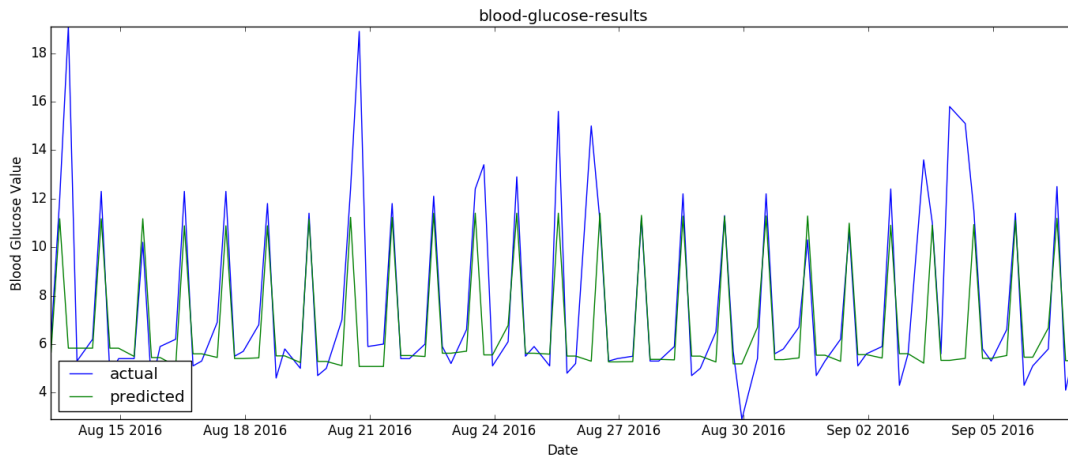


Figure 46: Graph of actual data vs predicted data using NuPIC generated model

Figure 46 shows the results of comparing the dataset (blue line) against data which NuPIC predicts (green line). As can be seen, the data that NuPIC predicts is accurate in most cases. However, it cannot account for sudden spikes in blood glucose levels. These spikes can be caused among other things by for example, a person not taking a large enough insulin dosage to cover the amount of food consumed or by not reducing dosage after performing exercise.

Research did not provide an immediate answer to this problem. After the result of an inquiry to the Numenta mailing list, a possible solution was provided (Figure 47). Due to time constraints in that week's iteration and the complexity involved with implementing inputs for multiple streams of data, the choice was made not to attempt to implement this solution.

I can think of 2 approaches

- 1.) Normalize both input data streams to the same time series [may be with averaging out the changes during the difference in time stamps] and then feed it as a 1 time series.
- 2.) Use a hierarchical model to train 2 independent regions with the separate time series data which feeds into 1 higher level region. This should allow for the formation of independent low level representations of the data. And they may connect at a higher level region. [This will be more experimental and not much work has been done with hierarchy]

Figure 47: Response to query on multi-field inputs

After this research and attempts to implement NuPIC with the diabetes dataset, the following advantages were identified:

- Is completely written in Python.
- Is open-source and well documented.
- Has a relatively active community that were happy to respond to questions.
- It has potential to be very accurate.
- It provides detailed and easy to read model files.

NuPIC has the following drawbacks:

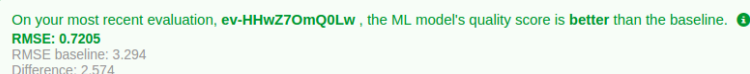
- It is relatively new and has little in the way of community adoption and support, StackOverflow has very few NuPIC related questions.
- It is still in the experimentation phase.
- It is more suited towards streaming data rather than sporadic data that would be logged by users in Glucose Coach.
- An initial large dataset of 500–1000 rows is required, which Glucose Coach users would not be able to easily supply.
- It uses too many resources and model generation takes a long time to complete.
- It would require more complex implementation to integrate into the Glucose Coach system.

Amazon Machine Learning

Amazon's offering of machine learning (AWS ML) was briefly looked at. AWS ML handles the model selection process by training and testing a lot of complex models tuned with different parameters to choose the best one. It also handles all aspects of input normalization, dataset splitting and model evaluation for the user. Once a model has been created and stored, it can then be accessed as a resource using an API hosted by Amazon.

AWS ML was tested using the diabetes dataset. The dataset was stored in an Amazon S3 bucket as a requirement. The model was then generated by following the simple steps on the management console, the only requirement being to specify the target column. The model took 2 minutes to generate. When finished, it is stored for later use and endpoints can be created to query it.

The model was evaluated using the AWS ML evaluation tool. The results show that the model was very accurate with a root mean square error (RMSE) of only 0.7205 (Figure 48). A test prediction was done with the generated model (Appendix d). The predicted result and further predictions made matched the actual insulin value accurately.



On your most recent evaluation, **ev-HHwZ7OmQ0Lw**, the ML model's quality score is **better** than the baseline. ⓘ
RMSE: 0.7205
RMSE baseline: 3.294
Difference: 2.574

Figure 48: Root mean square error of Amazon ML generated model

Amazon ML was not chosen for the following reasons:

- Primarily, its cost of 0.42 cents was very high. Further costs are incurred for hosting datasets and opening endpoints which could result in a bill of 100s of euro by the end of development.
- The generated models RMSE of 0.7205 was higher than scikit-learn's linear regression RMSE of 0.1480.
- There was very little tangible learning experience gained from using Amazon ML as all aspects were automatically handled by the system.

scikit-learn

scikit-learn is an open source machine learning library written in Python. It supports most of the popular machine learning regression algorithms, such as linear regression and k-nearest neighbor. The broad aim of these algorithms is to model the relationship between a scalar dependent variable y and one or more explanatory variables x .

Due to the nature of the data used in the diabetes dataset, the generalized linear model was chosen. The generalized linear model contains a set of algorithms for regression in which the target value is expected to be a linear combination of the input variables (Figure 49).

$$\hat{y}(w, x) = w_0 + w_1x_1 + \dots + w_px_p$$

Figure 49: Regression algorithm in mathematical notation where y is the insulin dosage

The explanatory variables in the diabetes dataset (Appendix C) used were:

1. Time of day: The inputs must be integer values. Therefore morning, afternoon, evening and night were classified as 1, 2, 3 and 4 respectively.
2. Blood glucose value: Values range between 2.0 and 21.0
3. Carbohydrates consumed: Values range between 0 and 120
4. Exercise taken: Values range between 0 and 500

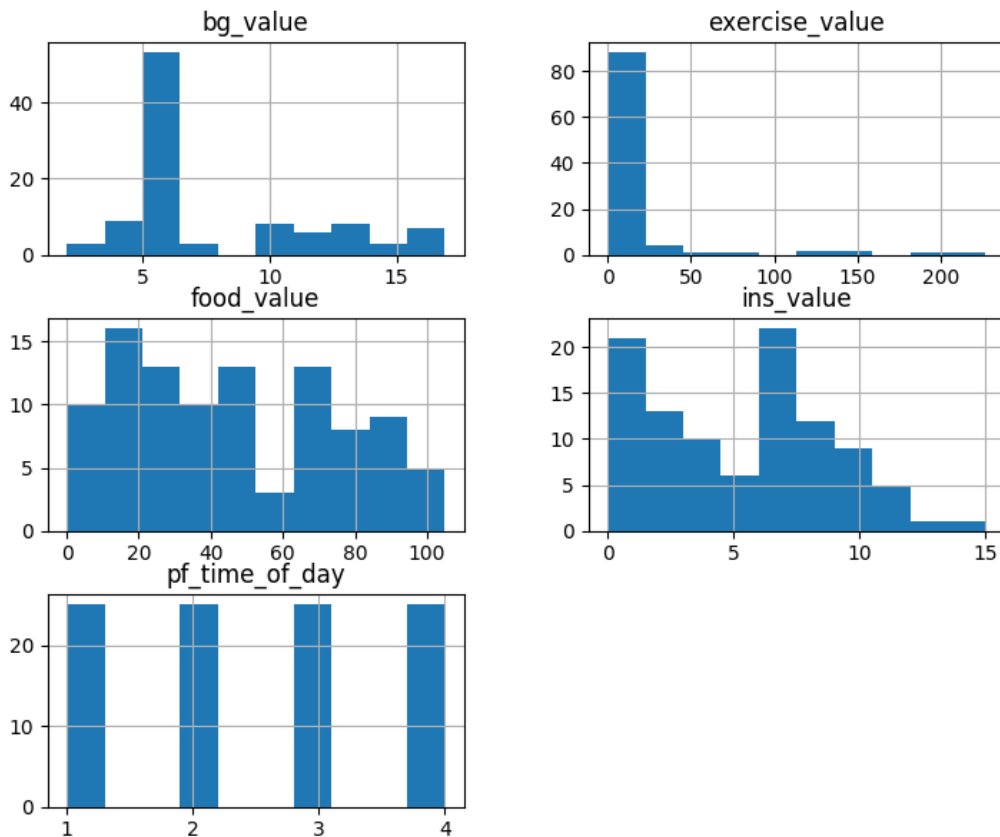


Figure 50: Histogram detailing the diabetes dataset

The target value was insulin dosage. This is a value ranging between 0 and 20. As discussed in the background research insulin section above, insulin dosage is correlated to the amount of food eaten and exercise taken and is unique to each person.

Five machine learning algorithms were chosen for evaluation. The Diabetes dataset which was used to test NuPIC and two other similar datasets evaluated their accuracy and performance. The machine learning algorithms chosen were:

- Linear regression (LR)
- Ridge regression (RID)
- Lasso (LAS)
- Elastic net (EN)
- Bayesian ridge (BR)

```
def compare_algos():
    # Split-out validation dataset
    array = dataset.values
    X = array[:, 0:4]
    Y = array[:, 4]
    validation_size = 0.20
    seed = 7
    X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size=validation_size,
                                                                                    random_state=seed)

    # Spot Check Algorithms
    models = []
    models.append(('LR', LinearRegression()))
    models.append(('RID', Ridge()))
    models.append(('LAS', Lasso()))
    models.append(('EN', ElasticNet()))
    models.append(('BR', BayesianRidge()))

    # evaluate each model in turn
    results = []
    names = []
    for name, model in models:
        cv_results = model_selection.cross_val_score(model, X_train, Y_train)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
```

Figure 51: Source code for script to compare machine learning methods

The above script (Figure 51) was written to evaluate each of the algorithms. The pandas library is used to load the dataset into a pandas data structure for easier processing. The X array is assigned the dependent values (time of day, etc.) and the Y array is assigned the target values (insulin dosages). The script then randomly splits the diabetes dataset in two, 80% is used to train each of the models and 20% is held back as a validation dataset for future testing purposes.

Cross validation is used on each model to calculate the number of errors that occurred in generating the model. The cross-validation results performed on the initial Diabetes dataset can be seen in (Figure 52). Linear regression, ridge regression and Bayesian ridge provided the most accurate results with a mean of nearly 98% accuracy. Over the three datasets, linear regression provided the lowest root mean square error of 0.1480.

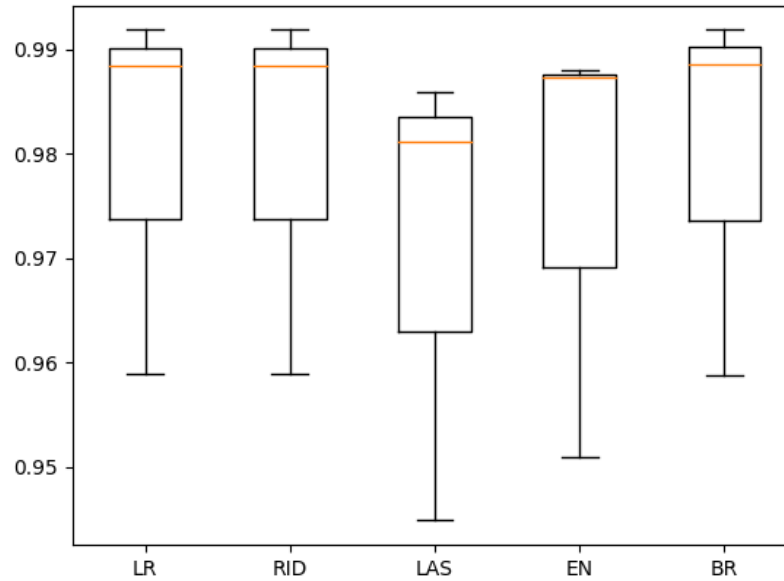


Figure 52: Cross validation results of each algorithm

Linear regression was chosen as the most suitable algorithm for providing insulin dosage predictions. Based on the data in Figure 52, it outperformed the other algorithms as it returned the lowest error margin. Table 10 shows the results of validation testing. The generated model was found to provide accurate results with a low error margin. Highlighted in red are these errors. It was found that enlarging the dataset provided more accurate results.

Predicted	10.15	10.03	6.42	4.1	6.01	8.83	11.04	1.27
Actual	11	10.5	6	4	6	9	11	1

Table 10: Predicted insulin dosage vs actual insulin dosage

In later iterations, the requirement for model persistence was implemented. This was achieved using the *pickle* library for dumping and loading models. When a model is trained, the model is given a filename which is based on the user's id and is saved in a directory on the server. The model can then be loaded in the future when the user requests an insulin prediction.

The above components of model generation and persistence were then integrated into the flask API. To train a model, the user navigates to the secured '<user_id>/train' endpoint. To ensure that an accurate model is generated for users, the user must have more than 20 records within the last 30 days and each record must contain a non-zero blood glucose entry.

To get an insulin prediction, the user navigates to the '<user_id>/train' endpoint. The request body of the HTTP request is parsed to get the time of day, blood glucose level, food eaten and exercise taken. These values are then fed into the user's model and the insulin prediction is returned. A default theoretical model also exists if no user model exists.

Subsequent to finishing the machine learning iteration, the following benefits of using scikit-learn were identified:

- scikit-learn is written in python which makes for seamless integration into the flask API and has the bonus of being open-source.
- Basic functionality is provided from a single library allowing for simple setup.
- Documentation is extensive and easy to read and helpful tutorials are available.
- Community support is very good due to its popularity.
- Model generation is very fast.
- Extensions are readily available to offer additional functionality, such as data structure management and model persistence.

scikit-learn was chosen as the machine learning library for Glucose Coach. Compared to NuPIC, model generation is significantly faster and predictions proved to be more accurate. Due to scikit-learn's popularity, problems that occurred during development could be easily answered by a quick Google search whereas NuPIC problems required significant additional research.

d. Meter Tool

Overview

A tool is required to sync data from a user's blood glucose meter to their Glucose Coach profile. The aim of this is to get users quickly integrated into the system by removing the need for manual entry of data. This data could possibly then be used to generate a personal model for delivering insulin predictions.

The requirements for this tool were:

- Users being able to login into their account to associate data with their identity.
- The ability to record a 'last sync date' so only new records are identified and pushed to their account.
- The ability to providing a graphical user interface (GUI).

Beginning Implementation

During the initial iteration, it was found that each USB enabled blood glucose meter has a unique vendor id and methods required for parsing packets returned from the device. As a result, only one type of meter was chosen for simplicity. The meter chosen was the 'CONTOUR® NEXT USB Meter' due to its popularity.

The main library used for development is PyUSB. A tutorial was used for much of the development and was adapted to work with the meter's requirements [13]. The HID interface interaction is based on reverse engineering work done by Ben Jones [14]. The GUI was developed using the TKInter library, as can be demonstrated in the UI design section above (Figure 20).

Application Structure

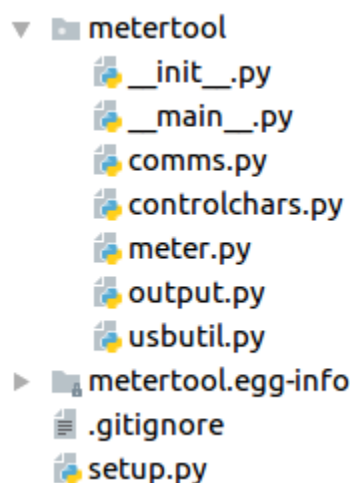


Figure 53: Source code layout for meter tool

Development

As can be seen in Figure 53, the project is contained in a single package. *Setup.py* contains information for installing the tool on a user's machine. The tool can then be invoked by typing 'sudo metertool' into a console. During the initial development iteration, the requirements for this component were merged with the PyUSB tutorial code and the reverse engineering logic for parsing HID packets returned by the meter.

The *init.py* file contains the logic for creating the GUI, handling user input and iterating over data dumped from the meter. The *comms.py* file is responsible for handling error checking on each frame returned by the meter. *Meter.py* is a class that is modeled after the physical meter. It supports read and write to the meter using various methods. *Output.py* is used to parse data from each of the frames dumped by the meter. It is also responsible for syncing data to the database. Finally, the *usbutil.py* file contains methods for finding the USB meter in the system.

System Validation

a. Testing

Testing was an essential part of the Glucose Coach development process. At the end of each weekly development iteration, system testing was performed. During the final weeks of the project, acceptance testing was performed with the participation of three users. The users had an APK of the app installed on their phones and were required to use it daily. Feedback received was used to fix minor bugs and generate suggestions for future work.

System Testing

System testing consisted of checking if each of the component features worked as required. A spreadsheet was used which contained the functionality of each of the component features. The process consisted of iterating over the checklist and testing each of the entries (Appendix A). In particular, it was important that this process was carried out after a new feature was added. Due to time constraints, unit tests were not fully implemented.

Deployment

Before testing could begin, the flask API needed to be deployed on a publicly accessible server. An Amazon AWS EC2 instance was chosen due to its ability to easily integrate with Glucose Coach's database hosted on an Amazon RDS instance and the provision of an IPv4 DNS hostname for the API. A Dockerfile (Figure 54) was written and the flask API was deployed in a Docker container on the EC2 instance.

```
FROM ubuntu:16.04

MAINTAINER Alex Kiernan

# Update OS
RUN apt-get update -y
RUN apt-get -y upgrade

# Install Python
RUN apt-get install -y python-pip python-dev build-essential
RUN apt-get install -y libmysqlclient-dev

# Create app directory
COPY . /app
WORKDIR /app

# Update pip and install reqs
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

# Expose listening port
EXPOSE 5000

# Run server
ENTRYPOINT ["python"]
CMD ["runserver.py"]
```

Figure 54: Docker file for building flask API container

User Acceptance Testing Setup

User acceptance testing was performed with three users. The users were aged between 21 – 34 years old. Each of the users had type 1 diabetes for between 5 and 20 years. The users considered themselves as being experienced in their diabetes control and were very familiar with the process of regulating their individual blood glucose levels. Users were made aware that the application was a prototype under development and that results should not be taken at face value. They were requested to take a note of insulin predictions given by the application and also to record the insulin they actually took based on their own experience.

The aims of user testing were as follows:

- To identify the initial accuracy of models generated using a small amount of user data.
- To compare the insulin dosage predictions to what users would normally take and to note any discrepancies.
- Near the end of testing, to retrain the model after entering 20 – 30 more logs and try to assess whether improvement in accuracy in results was achieved.

Before commencing the testing, the users were given a brief introduction to the app's functionality and how it can be leveraged. Two of the users were required to use it daily for one week, while the third user agreed to use it for two weeks. During testing, only the mobile application was tested. This was due to the fact that none of the users used Linux as an operating system which the meter tool requires. Additionally, all the users used their meters to only record their blood glucose results and did not fully utilize the exercise, carbohydrate counting and insulin dosage logging functionality that their meters provided making the meter tool redundant in this case.

Initial setup required users to add between 20 and 30 diabetes logs from the preceding two weeks. This was a guided process to ensure that accurate data was logged so that accurate models could be generated. Once enough data had been logged, a model was generated for each user.

User progress was tracked by querying their data through MySQL workbench. If problems occurred during testing, the user could provide the error and it would be fixed as soon as possible. This was important, as the APK given to users did not contain functionality to edit historic data records.

Users were asked to make a note of the prediction returned by the app next to where they recorded their actual insulin log in their paper based logbook. This allowed for predictions to be easily compared when testing had concluded. This data could also be extracted from the database at any time if needed.

User Acceptance Testing Results

User 1 and 2 had used the application for 1 week and user 3 had used the application for 15 days. After testing had completed, a considerable amount of data had been generated by each of the users. In total, 288 rows of data for generating models had been logged.

- User 1 had logged 84 rows of data over 7 days plus 2 weeks of previous data.
- User 2 had logged 88 rows of data over 8 days plus 2 weeks of previous data.
- User 3 had logged 116 rows of data over 15 day plus previous data.

After testing had completed on the 30th march, the results were analyzed and feedback was collected.

User 1 had found the insulin suggestions to be very accurate and had closely matched their insulin taken in many cases. User 1 was described as being lightly active and took exercise two nights per week (Figure 55). After the exercise, the user would typically reduce the insulin they took by half. For example, they would typically use 500 kilojoules of energy which would result in an insulin reduction of four units. This effect was identified and used for making decisions by Glucose Coach. As a result, the amount of insulin suggested to the user was reduced from 9 units to 4 units which is what they would have normally done in that situation.

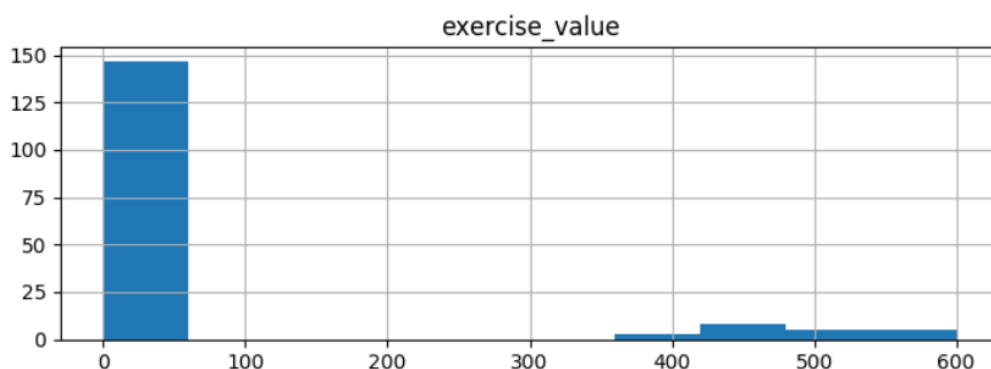


Figure 55: Chart of exercise taken by user 1

User 2 had similarly found the insulin predictions to be accurate except for the morning time when they had to add an extra unit of insulin for the first 3 days of testing. After research, this effect was identified as the “dawn phenomenon”. The natural rise in blood sugar between 4 and 8 am is the result of hormone changes in the body [15]. Glucose Coach was able to identify this trend, and after additional logs were entered for the morning time, the system was able to add an additional 0.5 units of insulin to the prediction (Table 11). Overtime, it is expected that this will become more accurate.

Actual	7	6	6	8	8.5	9.5	8	4
Prediction	5.5	5	5	7	8	9	7.5	3.5

Table 11: 8-day trend of morning insulin predictions for user 2 becoming more accurate

User 3 had used the system for two weeks. The user is described as being very active during weekends but inactive during weekdays. The user found the predictions to be accurate in most cases. However, they found that the suggested insulin dosage given after having completed a large amount of strenuous exercise to be inaccurate.

The inaccuracy was that the extended effect of exercise on blood glucose levels was not provided for. Typically following strenuous exercise, the user would reduce their insulin by half for approximately a period of 24 hours. However, Glucose Coach was recommending that they should take a reduced amount of insulin only immediately after having done the exercise and not for the next few hours.

All users found the application to be easy to use and intuitive. Users remarked that the design was easy to understand and to navigate through. The action button was found to be very useful as users frequently accessed it from the logbook screen. Users highlighted the fact that it would be useful if tapping cards on the home screen brought them to the respective add log screen. No application crashes were reported by users. The necessity to sign in every 24 hours was seen to be problematic.

User Acceptance Testing Feedback

The following is a list of suggestions for improvements received directly from users:

- Need to be able to edit previous entries.
- Would be nice to see chart listed by date rather than a number.
- Tapping on home screen cards should bring up the data entry screens. Interface looks nice!
- Not sure what exactly what retraining data does, maybe add an explanation of some kind?
- Would like the ability to search logs by date in the logbook screen.
- Most of the insulin predictions are not accurate for hours after exercising. Normally if I've finished my run on a Sunday afternoon, I would reduce my insulin by half of what I would normally have during dinner and before bed. However, the before bed prediction does not take this into account.
- Would like to add multiple food items at once rather than just entering carbs for the meal.
- Add a label to the popup icons as to their meaning.

Most of the feedback received related to feature updates in the app. These were not originally implemented due to time constraints. Users were generally happy with the design and performance of the app. The insulin predictions were mostly accurate in a high proportion of cases, as mentioned above. The issue of exercise having long lasting impacts on blood sugar levels and thus insulin dosage would be the next major problem that would need to be addressed.

b. Demonstration

A GitHub linked is provided below to a master repository for this project. It consists of the various components which make up the Glucose Coach system. Each component has been merged from their own separate repository using the Git subtree command. This URL is active as of 04-04-2017.

<https://github.com/Zontzor/glucose-coach>

A YouTube link to a demonstration of the Android app is provided below.

https://youtu.be/zI5K-lH6_Ok

Conclusion and Reflection

a. Project Summary

The development of Glucose Coach was prompted by my personal interest in the management of type 1 diabetes. Various factors influence insulin dosages daily. My idea was to produce a prediction system that made use of personal data entered by users. An additional requirement was the facility to upload data from a user's blood glucose meter.

In the introduction section, an overview of the system was described. This overview detailed why this project was undertaken, existing applications and their perceived limitations and how this project would address these issues. This section also detailed the objectives of the project and potential challenges.

The research section describes the background research performed. Alternative existing solutions were looked at and compared to Glucose Coach. Various technologies were evaluated as to their potential usefulness in this project. A plan for initial development was then devised based on the research done.

The design section explained the design methodology chosen for the implantation phase of the project. It also detailed the user interface prototypes and final designs, the various use-cases possible and design decisions made for the database.

Then architecture and tools section detailed the system architecture that was used. It explained the various technologies used for the client and server and their various advantages and disadvantages. It also detailed the development tools that were utilized and the version control system used.

The development section describes the development iterations. Each of the project components is given an overview, description of the initial implementation and further details about development. This section goes into detail about the machine learning systems evaluated in early iterations and their respective advantages and disadvantages.

The system validation section discusses the testing phase. The method of deployment for the Glucose Coach API is described. User acceptance testing is detailed, including the aims of testing, how testers were briefed and the results found. Problems that were found during testing are identified and explained with plans detailing how they would be fixed in the future.

b. Future Work

Glucose Coach has the potential to be an extremely useful tool for people with type 1 diabetes. With the addition of new features and data analysis, the system could prove to be a helpful part of a diabetic's daily management routine. To achieve this, further work and more extensive testing would be necessary.

Android Application Architecture

The Android applications underlying architecture would need to be updated to a more suitable architecture, such as model-view-presenter (MVP). This is due to the fact that the Android framework does not encourage developers to write clean code and makes it very difficult to create unit tests. Updating to an architecture such as MVP would allow for:

- Clearer separation of concerns which would allow for source code to be easier to understand and reason with as well as making it more maintainable. As it is, the Glucose Coach app has different bits of business and view logic mixed into the same classes. This can cause confusion when trying to understand exactly what a particular classes function is, particularly if bugs occur.
- Increased modularity would allow for application logic to be easily added or removed. At present, there is a high amount of coupling between code in the various activities. If changes need to be made, multiple parts of the app would be affected and would need to be updated. The presenter/controller in MVP would act as an intermediary between the UI code and the model. This would allow the view and the model to evolve independently of each other and thus be loosely coupled.
- Easier testing of components. The existing architecture makes it very difficult to unit test components effectively. Using a MVP architecture would establish a clear boundary between components. This would allow for the easier creation of mock objects to make testing of code simpler.
- Hiding of the data access layer. Using the MVP architecture forces the use of design patterns that move the data access code into a data access layer where it belongs. Network requests in Glucose Coach are currently made in the same area as application logic. This makes code look bloated and unreadable, particularly when chains of calls need to be made.

Additional Testing

During the testing phase of Glucose Coach, interesting patterns emerged in user's data that was different from their normal pattern. These patterns resulted in inconsistent insulin predictions. If more testing was performed, a better understanding would be gained into the causes of these inaccurate predictions. Input from a medical professional would be of great assistance in this regard.

Additional testing would allow for more user trends to be identified and would demonstrate more accurately how these trends are handled by the machine learning system. This data could then be used to see how many records it takes for the

personalized model to become more accurate. Examples of this could be that a person who is older would require only 2 weeks of data to generate an accurate model whereas a more active and younger person would require 4 weeks of data as well as an additional time dimension, such as an 'after sports' log.

Additional Features

Based on feedback received from testers and additional feedback received from a medical professional during the initial requirements phase, numerous additional features were proposed for the Glucose Coach system. Listed below are some of the most important of these planned features:

- The ability for a patient to send a weekly email to a medical professional listing their users logs as well as an account of how the actual insulin taken compared to the prediction. This would provide a facility for the accuracy of predictions to be checked before they were ever used in practice. The proposed plan for this feature is the addition of a database column containing the actual insulin taken by a user. Functionality in the app and API would need to be implemented to record it. The user could then send their weekly data as a spreadsheet to their doctor.
- A drawback cited by each of the users was the limited food log recording in the app. Currently, the Glucose Coach application only supports the logging of a single item of food for each meal. Users need to be able to add multiple items for each meal and possibly be able to sync information from third party APIs, such as MyFitnessPal. This would require changes to the database tables and triggers as well as implementing the functionality in the mobile app.
- A web based application had been proposed initially for this project. Due to time constraints, it was decided to focus on the mobile application and testing. The web app would give users detailed insights into the information they have logged. Graphical charts would be used to break down the various logs submitted by users, such as their blood glucose levels overlaid with the insulin they took. The web app would also be able to show comparisons of insulin the user took to the predictions they requested. An additional feature that would prove useful is allowing doctors to sign up and track a user's progress. They could then submit feedback to the user.

c. Reflection

The past six months has been a very challenging but rewarding experience. During this time, I have developed numerous organizational and technical skills. I am grateful to DIT for affording me the chance to develop a project which is of huge personal interest to me. I believe that if this project were fully realized, it would have the potential to be of great assistance to people living with type 1 diabetes.

I believe the project encompasses the major features initially required. I believe that choosing to expand the testing phase early in the project at the expense of developing a web app proved to be the correct decision. It gave me enough time to deploy my app on AWS to be used publicly and allowed users to thoroughly assess the performance of the application. The results from testing proved that the system is able to accurately identify trends in a user's daily routine.

This project allowed me to experiment and implement new technologies in which I had an interest but had never had the opportunity to use. Examples of the new technologies I had used in this project include; AWS, machine learning, Docker and reactive programming. Implementing these technologies successfully has given me greater confidence in my abilities as a software developer and made me less hesitant in looking at new ways to solve problems in the future.

Using the personal extreme programming software methodology and sticking to a tight timeframe has greatly improved my own project management skills. At the beginning of the process, a project roadmap had been defined. I succeeded in meeting most of the weekly iteration deadlines. In the few instances where I would slip behind on meeting a deadline, I would alter my schedule accordingly. I found that doing this provided an additional impetus to meet subsequent deadlines. The process of working on a strict timeline and performing weekly retrospectives on performance gave me a better appreciation for planning and time management in the delivery of software.

If I was to start this project again, I would use different approaches in a number of instances. In early iterations of the project, I would ensure to have a minimal working system at the end of every iteration. This would allow users to thoroughly test the system to identify potential problems early on in development. In addition, I would use a more maintainable and efficient Android system architecture rather than using Androids default model-view architecture.

I have an ongoing interest in further developing this project. During testing, users were impressed by the accuracy of the system and its potential to identify trends. After consulting with a medical professional about the possibilities of the app, an additional use case was identified which no other system currently provides. In addition to the predicting of insulin dosages, the system would enable constant monitoring of a user's progress. Based on the above, this system has the potential to be commercially viable and would be a valuable tool for users with type 1 diabetes.

Bibliography

- [1] Diabetes Ireland, "Managing Diabetes," October 2016. [Online]. Available: <https://www.diabetes.ie/living-with-diabetes/living-type-1/managing-diabetes/blood-glucose-levels-targets>.
- [2] American Diabetes Association, "Standards of medical care in diabetes," *The Journal of Clinical and Applied Research and Education*, vol. 37, March 2014.
- [3] V. C. Woo, "New Insulins and New Aspects in Insulin Delivery," *Canadian Journal of Diabetes*, vol. 39, no. 4, pp. 335-343, 2015.
- [4] P. J. Watkins, ABC of diabetes. 5th ed., London: BMJ Publishing Group, 2002.
- [5] M. Patricia Halfon, "Correlation between amount of carbohydrate in mixed meals and insulin delivery by artificial pancreas in seven IDDM subjects," *Diabetes Care*, no. 12, p. 427-429, 1989.
- [6] H. S. Warshaw, ADA Complete Guide to Carb Counting 2nd ed, Alexandria, VA: American Diabetes Association, 2004.
- [7] A. Hess-Fischl, "What is Insulin?," [Online]. Available: <http://www.endocrineweb.com/conditions/type-1-diabetes/what-insulin>. [Accessed October 2016].
- [8] A. D. Association, "Blood Glucose Control and Exercise," American Diabetes Association, [Online]. Available: <http://www.diabetes.org/food-and-fitness/fitness/get-started-safely/blood-glucose-control-and-exercise.html>. [Accessed October 2016].
- [9] Numenta, "Nupic GitHub," [Online]. Available: <https://github.com/numenta/nupic/wiki>. [Accessed November 2016].
- [10] MySugr, "Paper is for origami, not diabetes logbooks," [Online]. Available: <https://mysugr.com/paper-is-for-origami-not-diabetes-logbooks>. [Accessed November 2016].
- [11] M. K. MD, "UCI Machine Learning Repository," [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/Diabetes>. [Accessed December 2016].
- [12] Numenta, "Swarming Algorithm," [Online]. Available: <https://github.com/numenta/nupic/wiki/Swarming-Algorithm>. [Accessed November 2016].
- [13] Walac, "PyUSB 1.0 Tutorial," [Online]. Available: <https://github.com/walac/pyusb/blob/master/docs/tutorial.rst>. [Accessed January 2017].
- [14] B. Jones, "Experimental tool to dump data from a Bayer Contour Next USB meter," [Online]. Available: <https://github.com/bnjones/contourtool>. [Accessed December 2016].
- [15] F. Porcellati, "Thirty Years of Research on the Dawn Phenomenon," *Diabetes Care*, vol. 12, no. 36, 2013.
- [16] B. A. P. G. e. a. Laurenzi A, "Effects of carbohydrate counting on glucose control and quality of life over 24 weeks in adult patients with type 1 diabetes," *Diabetes Care*, no. 34, p. 823-827, 2011 .

Appendix

a. Project Plan

FYP Checklist		Weeks	Completed	Notes	Time Spent	Status		Weeks		
Task						Tests	Passed			
Android	Add BG Activity	1	✓		1d			23/01/17	1	
	Add Insulin Activity	1	✓		1d			30/01/17	2	
	Add Carbs Activity	2	✓		1d			06/02/17	3	
	Add Exercise Activity	2	✓		1d			13/02/17	4	
								20/02/17	5	
								27/02/17	6	
								06/03/17	7	
	User Sign In Activity	2	✓	Included planning	3d			13/03/17	8	
	User sign up activity	3	✓		4h			20/03/17	9	
	User Session Handling	2	✓	Token -> SharedPrefs	2h			27/03/17	10	
								03/04/17	11	Submission date - 04/04
	Logbook Activity	6	✓		2h					
	Charts Activity	7	✓							
	Settings Activity	7	✓							
	Main Screen Activity	2	✓		1d					
	Add main screen last data display	4	✓							
	Cleanup UIs	8	✓		1d					
	Refactor network callback hell	8	✗							
	bg reading post and validation	3	✓		3h					
	ins value post and validation	3	✓		2h					
REST API										
	Setup User Endpoint	2	✓		2h					
	Setup BG Endpoint	2	✓		2h					
	Setup Insulin Endpoint	2	✓		1h					
	Setup Carbs Endpoint	4	✓							
	Setup Exercise Endpoint	4	✓							
	Setup User Authentication	3	✓		1w					
	Add config file for setup	4	✓		1h					

Upload Tool					
	Pull Data from Device	2	✓		
	Auto Upload to DB	6	✓		
	Create UI	7	✓		
	Sync from last sync data	7	✓		
	Update UI	8	✓		
Machine Learning					
	Create a basic dataset for testing in NuPIC	1	✓		
	Adapt NuPIC to swarm over the dataset on Ubuntu	1	✓		
	Verify the model created by the swarm has accuracy, when running the dataset though NuPIC	5	✗		
	Research alternative machine learning system	4	✓		
	Complete Amazon ML Tutorial	4	✓		
	Create basic blood sugar amazon model	4	✓		
	Get accurate predictions from model	4	✓		
	Get ML endpoints talking to app	5	✓		
	Research regression algos	4	✓		
	Implement scikit regression algorithm	5	✓		
	Compare regression algorithms	5	✓		
	Implement endpoints for training and predicting	5	✓		
	Implement mysql databases	5	✓		
	Add unique model support for each user	6	✓		50%
	Add data pull from db	5	✓		
Data Generation					
	Generate test blood sugar levels	3	✓		
	Generate training user data: bg, ins, carbs, exercise	4	✓		2h
	Option to generate csv or sql	5	✓		
	Create foods and exercises inserts	5	✓		

b. API Endpoints

* Denotes authentication required

Available API endpoints:

GET /users/{username}	- List a user *
POST /users	- Add a user
PUT /users	- Update a users info *
GET /users/usernames/{username}	- Get a username
GET /users/{username}/bgreadings	- Get a users bg logs *
GET /users/{username}/bgreadings/{bg_id}	- Get a users bg log *
POST /users/{username}/bgreadings	- Add a user bg log *
PUT /users/{username}/bgreadings/{bg_id}	- Alter a users bg log *
GET /users/{username}/insdosages	- Get a users insulin log values *
GET /users/{username}/insdosages/{ins_id}	- Get a users insulin log value *
POST /users/{username}/insdosages	- Add a user insulin log value *
PUT /users/{username}/insdosages/{ins_id}	- Alter a users insulin log value *
GET /users/{username}/exerciselog	- Get a users exercise logs *
GET /users/{username}/exerciselog/{elog_id}	- Get a users exercise log *
POST /users/{username}/exerciselog	- Add a user exercise log *
PUT /users/{username}/exerciselog/{elog_id}	- Alter a users exercise log *
GET /users/{username}/foodlogs	- Get a users food logs *
GET /users/{username}/foodlogs/{flog_id}	- Get a users food log *
POST /users/{username}/foodlogs	- Add a user food log *
PUT /users/{username}/foodlogs/{flog_id}	- Alter a users food log *
GET /sync/{username}	- Get a users last sync date *
POST /sync/{username}	- Add a users last sync date *
GET /token	- Get a token, requires username and password

c. Partial Dataset

pf_time_of_day	bg_value	food_value	exercise_value	ins_value
1	5.1	26	0	2
2	11.3	100	0	12
3	4.6	105	0	10
4	5.7	10	0	1
1	15.9	34	0	7.5
2	11	45	0	6
3	5.1	78	0	7
4	5.9	8	0	0
1	2	31	227	0
2	16.9	39	0	8
3	5.6	45	0	4
4	5.1	14	0	1
1	2.8	26	0	0
2	12.5	30	150	5
3	5.2	77	0	7
4	5.1	94	0	9
1	5.6	20	0	2
2	12.6	49	0	7
3	5.5	67	0	6
4	5.9	87	49	8
1	6.7	98	0	9
2	10.3	87	0	9.5
3	5	74	0	7
4	5.4	13	0	1
1	5	38	0	3
2	16.6	65	0	11
3	15.6	63	0	10.5
4	5.1	8	35	0
1	5.8	25	0	2

d. Amazon Machine Learning

Try real-time predictions

You submitted 4 out of 4 data values for this prediction. ? X

Try generating real-time predictions for free using the web browser on this page. To request a real-time prediction, complete the following form or provide a single data record in CSV format. To provide a data record, choose the **Paste a record** button. Paste a record

Items per page: 10 << < 1 - 5 of 5 > >>

	Name	Type	Value
1	bg_value	Numeric	6.7
2	carbs	Numeric	50
3	exercise	Numeric	120
4	insulin_dosage	Numeric	Target
5	timestamp	Categorical	2

<< < 1 - 5 of 5 > >>

Clear data

Create prediction

Prediction results

Target name insulin_dosage
ML model type NUMERIC
Predicted value 2.7145893573760986

```
{
  "Prediction": {
    "details": {
      "Algorithm": "SGD",
      "PredictiveModelType": "REGRESSION"
    },
    "predictedScores": {},
    "predictedValue": 2.7145893573760986
  }
}
```

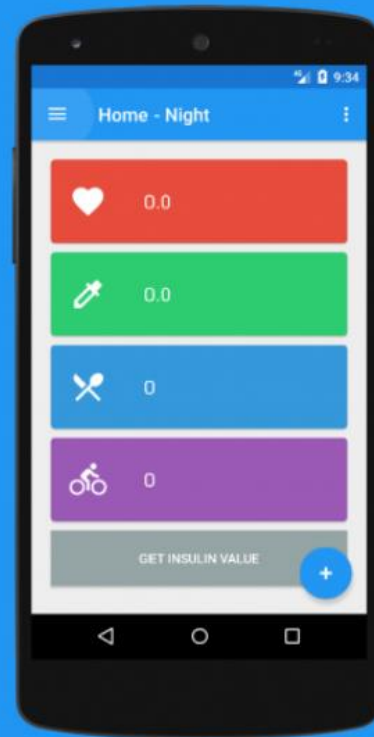
Next steps

To enable real-time predictions for your application, create a real-time endpoint. (Capacity and use charges apply. [Learn more.](#))

Create endpoint

Glucose Coach

Diabetes management
made easier using
machine learning



- Mobile application to log daily data
- Desktop tool to upload data from meters
- Personalized insulin predictions