A Diabetes Management Device With an Incentivized Reward Feature to Promote Healthier Patient Behavior

Citation

Permanent link
https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37365033

Terms of Use
This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story
The Harvard community has made this article openly available. Please share how this access benefits you. Submit a story.

Accessibility
A Diabetes Management Device with an Incentivized Reward Feature to Promote
Healthier Patient Behavior

Azzitta Rahmani

A Thesis in the Field of Biotechnology
for the Degree of Master of Liberal Arts in Extension Studies

Harvard University
May 2020
Abstract

The past decade marks the rise of the mobile health revolution; mobile applications (apps) have entered the field of medicine and numerous apps have been developed to manage various diseases, such as diabetes, obesity, lung disease and heart disease. The advent of mHealth is empowering patients to adequately manage their health. Currently, Apple’s AppStore contains over 40 thousand health related apps to assist patients with their chronic conditions, with diabetes management leading. Of these, nine are FDA approved and are considered a medical device. There is a great demand for medical apps to help patients manage their health while following the guidelines of a medical device.

This case aims to prototype a medical device by making use of the iPhone to encourage patients with Type II diabetes mellitus (T2DM) to develop consistent day-to-day self-care habits to raise medication adherence, healthy eating, weight loss, reduce the advancement of diabetes complications, improve quality of life, advocate favorable lifestyle behaviors, and mitigate diabetes-related emotional distress and fatigue. Simultaneously, users can earn and exchange reward points for retail gift cards.
Dedication

This dissertation is dedicated to the memory of my beloved mother, Fatemeh Jafari, who passed away on November 29, 2010 due to Type II diabetes mellitus complications. Her enormous sacrifices, continuous support and unconditional love have inspired me to pursue and complete this research.
# Table of Contents

List of Tables ........................................................................................................... V

List of Figures .......................................................................................................... VI

I. Introduction ........................................................................................................... 1

   Background of Medical Mobile Applications ................................................. 4

   FDA’s Regulatory Approach ........................................................................ 5

   The Medical Mobile App Market ................................................................. 8

II. Materials and Methods ....................................................................................... 10

   Swift for iOS .................................................................................................... 11

   Development Environment .......................................................................... 12

   HealthKit SDK .............................................................................................. 13

   Amazon Elastic Compute Cloud (EC2) ...................................................... 14

   JSON ............................................................................................................ 15

   GoLang – Server Software ......................................................................... 16

   PostgreSQL – Server Database .................................................................. 17

   System Architecture and Design .............................................................. 21

   System Architecture .................................................................................. 21

   System Design ............................................................................................ 24

   System Functionality and Prototype .................................................... 26

   Application Testing ..................................................................................... 28

III. Results ............................................................................................................... 32

   Application Features – Screen Mocks and Description ......................... 3
IV. Discussion

45
List of Tables

Table 1. HbA1c Levels and Corresponding Blood Glucose……………………………………2

Table 2. FPG, OGTT and RPGT Levels Linked to Diabetes…………………………………3

Table 3. Mobile Medical Apps for Which the FDA Will Enforce

Regulatory Oversight……………………………………………………………………….6
List of Figures

Figure 1. The UML Diagrams, Representing Each Data Model…………………………20
Figure 2. Model View Controller Architecture…………………………………………22
Figure 3. Registration Screen Mock……………………………………………………34
Figure 4. Sing In Screen Mock…………………………………………………………34
Figure 5. Homepage Onboarding Screen Mock………………………………………..37
Figure 6. Standard Homepage Screen Mock…………………………………………..37
Figure 7. Points Screen Mock………………………………………………………….39
Figure 8. Add Entry Screen Mock……………………………………………………..39
Figure 9. Adding a Tag to an Entry Screen Mock………………………………………40
Figure 10. Setting a Reminder to an Entry Screen Mock……………………………..41
Figure 11. Adding a Note to an Entry Screen Mock……………………………………41
Figure 12. Health Screen Mock………………………………………………………..42
Figure 13. Inbox Screen Mock………………………………………………………….43
Figure 14. Other Screen Mock………………………………………………………….43
Figure 15. Profile Screen Mock………………………………………………………….44
Figure 16. Settings Screen Mock………………………………………………………..44
Chapter I
Introduction

Type II diabetes mellitus (T2DM) is a chronic metabolic disease that affects over 30 million Americans, representing 9.4 percent of the population (ADA, 2018). The prevalence of diabetes mellitus is steadily rising and each year 1.5 million are diagnosed with this condition (ADA, 2018). Diabetes continues to be the 7th leading cause of death (ADA, 2018) and poses a considerable economic burden. Since last year, the total cost of diabetes has been estimated to be $327 billion: $90 billion in reduced productivity and $237 billion in direct medical costs (diabetic medication and supplies, drugs to treat complications of diabetes, hospital outpatient care, office visits, stays in nursing facilities), (ADA, 2018). As follows, T2DM patients’ medical expenses are on average 2.3 times higher than those without the disease (ADA, 2018).

Diabetes can be life threatening if not managed adequately and poor glycemic control is a continual battle for patients with T2DM. To measure for blood sugar levels, patients test for haemoglobin A1c (HbA1c) levels. Glycated haemoglobin, also known as HbA1c, is an accurate marker for the prediction of complications. This biomarker is produced by a nonenzymatic reaction with haemoglobin (which is a red blood cell protein that carries oxygen throughout the body) and glucose. HbA1c is a reliable indicator of long-term glycemic control because the average life of a red blood cell is two to three months and the amount of glucose that combines with hemoglobin is directly

1
proportional to the total amount of sugar that is in the body. According to Karnchanasorn et al., “the measurement of HbA1c has become a standard in the care of diabetic patients and for monitoring glycemic control over a three month period”. HbA1c levels are expressed in terms of percentage, as illustrated in Table 1.

<table>
<thead>
<tr>
<th>HbA1c %</th>
<th>Status</th>
<th>Blood Glucose</th>
<th>mmol/L</th>
<th>mg/dL</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Normal</td>
<td>5.4</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Normal</td>
<td>7.0</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Pre-Diabetes</td>
<td>8.6</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Diabetes</td>
<td>10.2</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Diabetes</td>
<td>11.8</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Diabetes</td>
<td>13.4</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Diabetes</td>
<td>14.9</td>
<td>268</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Diabetes</td>
<td>16.5</td>
<td>297</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. HbA1c levels and corresponding blood glucose (Sherwani, 2016)

Additionally, patients can be tested for fasting plasma glucose (FPG) levels. This test reveals fasting blood sugar levels and diabetes is diagnosed at FPG levels of greater than or equal to 126 mg/dL. There are two more methods to diagnose T2DM, the Oral Glucose Tolerance Test (OGTT) and Random Plasma Glucose Test (RPGT). Table 2 shows the FPG, OGTT, and RPGT levels that are linked to diabetes. OGTT shows how the body processes sugar by checking glucose levels before and two hours after drinking
a sweet beverage. RPGT is a blood test that is taken when the patient is experiencing severe symptoms.

<table>
<thead>
<tr>
<th>Diabetes Screening Methods</th>
<th>Level</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fasting Plasma Glucose (FPG)</td>
<td>126 mg/dL or higher</td>
<td>Diabetes</td>
</tr>
<tr>
<td></td>
<td>100 mg/dL – 125 mg/dL</td>
<td>Prediabetes</td>
</tr>
<tr>
<td></td>
<td>Less than 100 mg/dL</td>
<td>Normal</td>
</tr>
<tr>
<td>Oral Glucose Tolerance Test (OGTT)</td>
<td>200 mg/dL or higher</td>
<td>Diabetes</td>
</tr>
<tr>
<td></td>
<td>140 mg/dL – 199 mg/dL</td>
<td>Prediabetes</td>
</tr>
<tr>
<td></td>
<td>Less than 140 mg/dL</td>
<td>Normal</td>
</tr>
<tr>
<td>Random (or Casual) Plasma Glucose Test</td>
<td>200 mg/dL or higher</td>
<td>Diabetes</td>
</tr>
</tbody>
</table>

Table 2. FPG, OGTT and RPGT levels linked to diabetes (ADA, 2018)

With effective treatment and management, diabetes and subsequent complications can be mitigated. The ADA advises glycosylated hemoglobin (HbA1c) target of 7% or less to prevent diabetes-related complications. Diabetes self-management education and support (DSME/S) are prerequisites to achieving the target HbA1c; by this route, patients gain the fundamental knowledge, skill and ability as well as support necessary for diabetes self-care. Funnell et al., report that the “objectives of DSME are to support informed decision-making, self-care behaviors, problem-solving and active collaboration with the health care team and to improve clinical outcomes, health status, and quality of life”. Customarily, a health care provider administers DSME and a member of the health care team provides DSMS. One of the challenges patients face with this curriculum is
accessibility to the program. With the adoption of medical mobile apps for diabetes self-care, this challenge may be alleviated.

Background of Medical Mobile Applications

The first iPhone was released on June 29, 2007. Nearly two years later, during the World Wide Developer’s Conference, an executive from Johnson & Johnson introduced the potential of mobile medical apps by connecting a blood glucose meter to an iPhone. In 2010, Free et al., described health care via a mobile app as “the use of mobile computing and communication technologies in health care and public health”. Specifically, mHealth, a subdivision of eHealth, describes the use of mobile devices in healthcare. As of January 2018, the number of available iPhone apps is 2.2 million (Statista, 2017) and 1.85 percent, just over 40 thousand, are categorized as medical apps (Statista, 2018). A general search on the App Store on an iPhone, in one sitting on November 28, 2018, using the term “diabetes” resulted in 198 apps. Accordingly, the total percentage of apps for diabetes management is only 0.5%. The rate of mobile phone usage is steadily increasing; Statista reports that in 2016, sixty-three percent of the population worldwide owned a mobile phone and 38 percent of this group consisted of smartphone users, which is expected to reach 2.7 billion by 2019 (Statista, 2016). Additionally, in 2016, the number of mHealth app downloads was 3.2 billion (Statista, 2017) and the healthcare field with the highest global market potential through mHealth is diabetes (Statista, 2018). The rate of mobile device adoption combined with the market value of mobile devices makes their application in medicine more and more evident.
FDA’s Regulatory Approach

According to the Federal Food, Drug, and Cosmetic Act (FDA) section 201(h), a mobile medical app is an app with the purpose of 1) transforming a mobile platform into a regulated medical device or 2) operating as an accessory to a regulated medical device (FDA, 2015). “A regulated medical device is defined as a product that meets the definition of device in section 201(h) of the FD&C Act and that has been cleared or approved by the FDA review of a premarket submission or otherwise classified by the FDA” (FDA, 2015). Health and medical related apps have become so prevalent recently that the FDA specifically defined to which operational use qualifies them as a medical device. Table 3 outlines existing examples of cases in which mobile apps have been used as a medical device, and for which the FDA will enforce its oversight.

<table>
<thead>
<tr>
<th>Case &amp; Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile apps that are: 1) an extension of one or more medical device(s) or 2) display, store, analyze or transmit patient specific medical device data</td>
<td>1) Remote display of data from bedside monitors</td>
</tr>
<tr>
<td></td>
<td>2) Display of previously stored EEG waveforms</td>
</tr>
<tr>
<td>Mobile apps that transform the mobile platform into a medical device</td>
<td>1) Attachment of a transducer to a mobile platform to function as a stethoscope</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>1) by using attachments, display screens, or sensors or 2) by including functionalities similar to those of currently regulated medical devices</td>
<td></td>
</tr>
<tr>
<td>2) Attachment of a blood glucose strip reader to a mobile platform to function as a glucose meter</td>
<td></td>
</tr>
<tr>
<td>3) Attachment of electrocardiograph (ECG) electrodes to a mobile platform to measure, store, and display ECG signals</td>
<td></td>
</tr>
<tr>
<td>Mobile apps that allow the user to input patient specific information and through the use of formulae or processing algorithms, output a patient specific result, diagnosis, or treatment recommendation to be used in a clinical practice or to assist in making clinical decisions</td>
<td>1) Compute the prognosis of a particular condition or disease</td>
</tr>
<tr>
<td>2) Perform calculations that result in an index or score</td>
<td></td>
</tr>
<tr>
<td>3) Calculate dosage for a specific medication or radiation treatment</td>
<td></td>
</tr>
<tr>
<td>4) Provide recommendations that aid a clinician in making a diagnosis or selecting a specific treatment for a patient</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Mobile Medical Apps for which the FDA will enforce regulatory oversight (US Food and Drug Administration, 2015)

The FDA has classified a subset of software apps that meet the definition of a device, through this classification; it has identified specific regulatory requirements that apply to these devices (FDA, 2015). Thus, the agency does not regulate every medical
app; if an app does not meet the terms outlined in section 201(h) it will not be regulated because it is not treated as a medical device. Additionally, if an app does meet the requirements to be considered a medical device but does not cause risk to the user, it will not be regulated as strictly. The level of regulatory control varies upon the risk presented by the device to public health. There are three different classifications; Class I, II and III that can be assigned to a regulated medical app. A medical app with a classification of I is considered low risk whereas a medical app with a classification of III is treated as a high-risk device. Class I devices include general controls, Class II devices require general controls as well as special controls (special controls are specific to the device, which include performance standard, postmarket surveillance, patient registries, special labelling requirements, premarket data requirements, and guidelines according to FDA, 2015), and class III devices include general controls and premarket approval. “The FDA intends to apply its regulatory oversight to only those mobile apps that are medical devices and whose functionality could pose a risk to a patient’s safety if the mobile app were to not to function as intended. This subset of mobile apps the FDA refers to as mobile medical apps” (FDA, 2015). In order to be competitive in the medical application market, and reputable amongst consumer groups, it's essential for the business to obtain FDA approval. It also creates additional exposure to the product because it may be prescribed by physicians and covered under medical insurance.
The Medical Mobile App Market

In 2010, the FDA cleared and approved the use of mobile apps as medical devices to manage diabetes, which implies that mobile health apps are considered effective in the management of diabetes, as stated by Jo In-Young et al., 2017. In fact, a study by Mao et al., shows that the use of a mobile medical app is an effective tool in helping patients manage diabetes; patients in this study successfully reduced their body weight and blood pressure using digital therapy in conjunction with tailored health programs. A recent study by Conway et al., demonstrates that 71 percent of smartphone users favored mHealth, however, user engagement at seven percent needs to be improved. We propose that in order to combat the low engagement other apps face, a reward system must be implemented in which patients are not just encouraged to use it, but financially benefit as well.

Currently, there are various resources and gadgets available to help patients advance their role in managing their diabetes. Such devices include blood glucose meters and strips, continuous glucose monitors (CGMs), flash glucose monitoring, injection pens for insulin and GLP-1 agonists, insulin pumps, automated insulin delivery systems, and data downloading software. Supplementary to this index is a list of mHealth apps specific to diabetes care. A feature that distinguishes this app from other diabetes-related apps in the current market is a reward system where patients can earn points for each healthy behavior such as exercising, healthy eating and logging, blood glucose logging, medication adherence, etc. Mango Health, a gamification health app available today, implements the incentive reward to encourage patients to practice daily healthy behavior.
Patients using the Mango Health app can earn points every time they take their medication, and their earned points can be redeemed for a retail gift card. What makes Mango’s business model attractive is their partnership with Express Scripts which is one of the largest pharmacy benefit management organization, allowing 85 million clients direct access to Mango Health mobile app. In an interview with Mobi Health News, CEO Jason Oberfest reported that the company has 180 days clinical data showing that their technology has raised 89 percent adherence to anti-hypertensives, 84 percent adherence to statins and 85 percent adherence to diabetes medication (Comstock, 2017).

Additionally, John Hancock, a $400 billion life insurance company revolutionized their 156-year-old business model to join Vitality. Vitality is the first insurance company that offers a behavior change platform to reward customers for the everyday actions they take to live healthier lives. This collaboration stems from the desperate shift in American’s health where lifestyle diseases due to physical inactivity, unhealthy diet, excessive alcohol, and smoking account for the leading causes of death (Senior, 2018). Vitality’s smart tech, data, incentives, and behavioral science show promising results; to date, their customers live 13 to 21 years longer and hospitalization costs are reduced by 30 percent compared to other insured populations. Additionally, John Hancock Vitality policyholders take twice as many steps as the average American, recorded over three million healthy activities such as walking, swimming and biking, and user retention is approximately 576 times per year (Senior, 2018). The shift for insurance giants to adopt mobile app technology as a preventative care measure to reduce costs creates an opportunity for app developers to fill that demand. The attention will be on those which are both FDA approved and recognized by insurance companies as preventative. As a
result, there will be immense growth in the mobile medical app space, and we will see them compete on user retention.
Chapter II.
Materials and Methods

This chapter focuses on the technologies that are required for both front-end and back-end infrastructures to communicate with one another. The technology stack section proposes and lists at a high level various third-party hardware, APIs (Application Program Interface), SDKs (Software Development Kit), programming languages, and database formats that are required for the application to function in its entirety. The specific technologies have been decided based on current trends, costs to deliver the service (web hosting), speed of scaling (how many active users do I expect to have in Q1, Q2, etc.) as well as determined by listing out each feature requirement. For example, in order to store user’s information in a format that allows it to be optimally indexed, queried, easily scalable and cheap, the decision has been made to use a relational database (SQL) because the technology is proven (used by industry leaders), well documented (it has been around for at least 15 years), and will be relatively easy to find engineers familiar with its use, which means development will be cheaper. There are also cross-platform technologies that allow engineers to create a single application that can be installed on both iOS and Android devices, however those solutions can be severely limited when integrating with device hardware such as Bluetooth. Examples include Xamarin and React Native, both of which are very respected options, well supported and documented, however since the proposal requires that the app be able to integrate with third party wireless glucose meters, the decision has been made to develop two separate
iOS and Android apps, written in their native languages, which is Swift for iOS and Java for Android. The proposal is to build an iOS prototype that integrates with at least one wireless glucose meter with the intent that additional meters will be added in the future. Today, glucose reader manufacturers have created separate frameworks (SDK) for both iOS and Android software developers, which is also why it makes sense to develop them separately. For the purpose of creating an iPhone app prototype, I list specific technologies and procedures that iOS may require, however the back-end technology stack must be designed so that it also functions on both Android and mobile web platforms, such as web browsers in the future. This will be considered a “mobile-first” company since the software will be created for use only on a mobile device and not a desktop computer. The following section breaks down each technology by describing how it fits into the overall tech stack, why it's needed, as well as any drawbacks or considerations that need to be taken into account.

**Swift for iOS**

Creating applications for the Apple iPhone requires using a specific programming language, which is either Objective-C or Swift. Objective-C is the original language and is still widely used today, however it is being replaced by the Swift language. Swift offers more advanced features over Objective-C and is a much cleaner and easier to work with. It generally forces the engineer to write “safer code” or code which is less likely to cause the app to unexpectedly crash. Apps that are buggy or frequently crash are more likely to receive poor ratings on the AppStore which will have a negative impact on the number of downloads, so there is also a business reason for making this choice. There may be some
thirty party frameworks that use Objective-C, so developers need to be familiar with it, but for the majority of the application, such as the user interface, it makes sense to use the Swift programing language. Swift is also highly adopted by the app development community, well documented, and is the preferred language, especially when creating a new app from the ground up. For these reasons, I choose Swift as the development language.

Development Environment

The IDE (Integrated Development Environment) that is used to create iPhone apps is called Xcode and requires running on an Apple computer with the OSX operating system. I propose using the latest version of Xcode, which at the time of this writing is Xcode 11.3.1 using Swift 5.1, and a minimum iOS device version of 11. Choosing the minimum iOS version is an important business decision because it determines which devices will be supported. If I choose too high of a minimum version, then it forces the user to own the latest version of the iPhone, which reduces the number of potential users. Older devices aren't fast enough to run the latest version and supporting too old of a version means extra work for the engineer, and that can be costly, so it's best to choose a version that is maybe 2-3 years old at most. By setting the minimum to iOS 11 means to support the iPhone 5s and newer, which is a 6-year-old device, a requirement that our target demographic should satisfy. It is expected that any device older than that would lack the Bluetooth hardware requirements and be too slow.
HealthKit SDK

Apple has released HealthKit to developers, which provides a way to store and share health related data across apps. It allows data collected by one app; like heart rate or sleep pattern information, to be viewed by another app, say one with a better chart or algorithm to automate health recommendations, as an example. Since the SDK now supports glucose tracking data, it will be an import technology the app can leverage. It would allow any user with existing glucose data to import that data and use it towards establishing historical blood sugar levels. Data collected by this app can also be exported to the SDK so it can be used by other apps. Since HealthKit is a relatively new platform, there's a limited number of apps that use it, so there is a possibility that Apple could recognized the app on the AppStore for utilizing this feature, which is also a great way to get free publicity, ultimately leading to more downloads and more active users.

Amazon Elastic Compute Cloud (EC2)

Data that is entered into the app, such as account information, must be accessible over the Internet, and stored using a central database, known as the cloud, back-end, or server. The reason why we force the user to register an account, as opposed to simply storing data locally on the device, is for easy back-up and restore of log data, access to online features such as rewards that can be redeemed for gift cards, and to facilitate the use of relevant targeted ads.

The system will support different types of users such as caregivers, physicians, and patients, who can interact in the app, but for the initial prototype, the only type of supported user will be the patient, or the user with T2DM. This requires storing specific
data on the user such as email, password, age, etc. into the cloud, or server running software that acts as an Application Programming Interface (API). To achieve this, a virtual server instance running on Amazon EC2 listens to incoming communications to each device over the Internet. The system itself is designed in a way that allows dynamically increasing compute speed on-demand whenever there's a sudden spike in traffic. In the same way it also allows scaling up resources as the platform grows its user base. It is essentially itself a computer with the sole responsibility of processing incoming and outgoing Internet traffic to the app with the benefit of not having to maintain a physical server. It acts as a gatekeeper between the application running on the phone and the central database running on the server. The server instance runs software that accepts incoming HTTP requests using a commonly used controller/action plus parameters API design format. For example, the API to get a user's profile with unique identifier “3” would be in the format GET /user/profile?user_id=3. To save a user's profile to the server would be in the format POST /user/profile?user_id=3 with a JSON payload that contains the data to be saved for the user. Data collected by the application running on the user's device is encrypted and sent to the server over a Secured Socket Layer (SSL) to the back-end API. In this case, it looks up the user profile for user id “3” and sends the profile information back to the phone. It's important to note that the server itself doesn't define the API specification, it's the back-end engineers who define it by writing custom tailored software that is then uploaded and run on the server. This is collectively referred to and viewed from the app's perspective as the “back-end API” because it simply exposes methods that the client (app) may use to store and retrieve data from the database over the Internet. This method for integrating with the server is commonly used throughout the
Internet by many apps and is not unique or propriety, which means it's relatively easy to find documentation on how to create and maintain similar systems. The product will instead leverage specific features to differentiate itself rather than utilizing any unique technologies.

**JSON**

Java Script Object Notation (JSON) is the preferred data format that the front-end uses to transmit and receive data to the backend, and visa-versa. It's a structured key-value format that is commonly used to represent every variation of common primitive data types, such as strings, integers, numbers, dictionaries, and arrays. It acts as an intermediate data format between the device and server that is human readable and easy to understand. JSON is only ever used external to both the client and server when transacting over the Internet. Data from the app gets transformed from byte code into JSON, sent to the server over the Internet where it's converted back into another model that the server understands in order for the payload to be used. This process is called JSON serialization and deserialization, or the conversion into and out of JSON format. It acts as a common standard for computers to transfer data between one another since they most likely run different operating systems and languages.

**GoLang – Server Software**

Go is a relatively newer language developed by Google that is extremely fast, scalable, and has a proven track record for use with mobile first applications that process
a large number of transactions. It's a compiled language which means that each time you make a change you must re-compile and re-deploy, and that can consume developer time, however compiled languages execute requests much quicker than scripted languages like PHP. The tradeoff is beneficial from a business perspective if the goal is to adequately support a large number of simultaneous users sending small and large amounts of data, rather than just a few users sending large amounts of data. Also, Go has been in existence for several years and has a strong developer community behind it. Go will be the language used to create the software that runs on the server (EC2), processing each API request, does any JSON serialization, and interacts directly with the database.

**PostgreSQL – Server Database**

A structured query language (SQL) such as PostgreSQL is an industry leading solution for database management systems and is used for many different types of software applications. The database itself can be thought of as a number of separate spreadsheets that are linked together using a query language. It supports the storage of data into tables that can be indexed using unique identifiers, known as primary keys. Tables are separated and grouped into logical buckets of data, for example user data into a user table, where each row represents a user entry, and each column represents a value, like first name, last name, etc. A unique primary key in one table becomes the foreign key in another table, providing the structure in structure query language. For example, you might have a table containing all medications for all users in the system within a single medication table. Each medication entry (row) has a unique primary key (medicationId) that auto-increments whenever a medication is added, 1, 2, 3, etc. In order to link a medication to a
particular user or to view all medications for a user, it must also contain the unique user identifier, the userId, which is added as a foreign key in the medication table. To query the database and lookup the medication of a particular user, you would query the medication table by userId. If you wanted to display data contained across multiple tables in the same API response, you can join the tables using the primary and foreign keys to be able to access the data that's required. For example, if you want to show a user's information and all medications for that user within the same API response (same page), you would join the user and medication tables using the globally unique userId. SQL uses what is known as a “select” statement, that is used to limit the number of columns of data to include in the response. A where clause reduces the results to a particular row, and the from command indicates what table to query. For example, to list all users in the system, a query may include user id, email, first and last names:

```
SELECT userId, email, firstName, lastName FROM User
```

If you just want to look up the results for a particular user 35, use the where clause:

```
SELECT email, firstName, lastName FROM User WHERE User.userId = 35
```

To automatically include every column in the table, an asterisks (*) is used. To link two or more tables together, a join command is required followed by their primary/foreign key equality. An example SQL query to display all user information and all medications for user 35:

```
SELECT * FROM User INNER JOIN Medication ON User.userId=Medication.userId
WHERE User.userId=35
```
SQL also supports much more complicated queries and ways to answer very specific questions, like “How many users are older than 50 years old?” or “How many medications does Joe take?”

There is another database technology called NoSQL but that specializes in larger distributed sets of data not needed for this type of application. SQL is commonly used in large scale enterprises like insurance companies and other medical related services, so it's easier to find software engineers familiar with its use, meaning it will be less expensive to maintain in the long run. It also works very well when exporting to other formats like XML (Extensible Markup Language) or CSV (Comma Separated Values) making it easy to sync the data across third party medical related databases. Each table structure and their interconnected relationships can be visually represented using a Unified Modeling Diagram (UML). The diagram includes each individual table with the column name and data type, as well as if it's primary or foreign key (PK vs. FK).

The minimum tables required for this application to function are User, Medication, Activity, Reminder, Glucose, Meal, Keytones, Log, Tags, Insulin, Mood, BloodPressure, Water, and Weight. Most of the tables include a user id as the foreign key to allow an entry to be tied to a particular user. Additional columns may be added at a later date, but usually the data type can never be changed. For example, the Amount column in the Weight table is a Float data type and can never be changed to a String later because the API and application expect it to be a Float (decimal point numb
Figure 1. The UML diagrams, representing each data mode
System Architecture and Design

The mobile application runs on either an Apple iPhone or iPad that operates a minimum version of iOS11. End users have the option to either manually enter in their health data, import it using Apple's HealthKit, or connect wirelessly to a device that can read it automatically. Connecting to a Bluetooth peripheral device requires the user to explicitly grant permission to the application. There are several wireless Bluetooth glucose meters in the market that will work with the application. For the prototype, the app will support one type of glucose meter with the idea that more will be added in the future. There is a rewards feature that allows the user to collect points by interacting with the app and reaching their health goals, which can be redeemed for discounts to products and services in the form of gift cards. To ensure accountability and to prevent people from gaming the system by entering in false blood sugar levels to earn points, only verifiable methods used to collect the data can count towards redeemable points. This is made possible by using a Bluetooth glucose meter.

System Architecture

A common design pattern used by mobile software development is what's called MVC, or Model View Controller. It separates the code into three distinct roles and responsibilities allowing reusability, separation of logic, testability, and maintenance. It enforces that one component within the software operates independently and without knowledge of another component. While its concept originated from UI design pattern principles, it can also be used for other systems like application development and back-end software development. Developing mobile applications for the iPhone and iPad
requires adopting this design pattern because it is actually enforced by the way Apple frameworks are designed out of the box.

![Model View Controller Architecture](image)

Figure 2. Model View Controller Architecture

The model represents common re-usable data objects that contain properties, or variables, which store values. An example would be the customer model. Since there are many customers, they can be represented using a common structure, a customer model that has several properties, such as first and last name, both of which can be represented using string value types. Another value type is a number or integer, and can represent things like a user's age, or number of points they have accumulated. There are many different models used by the system that work together and generally are represented similarly by the app, database, and in JSON format. Typically, a single model in a UML diagram is used for all purposes, even if their implementations vary across platforms. A model may also contain other nested models, which is known as a one-to-many relationship. For example, a single user may have many addresses. A single address is represented by an address model, while a user's profile model could contain multiple
addresses, hence a one-to-many relationship, one user model, many address models.

The controller acts as the gatekeeper between the model and view, but also the business logic, managing what gets displayed when, what data is needed, and how to respond to user input. Its primary responsibility is to load the data models, choose the view to use, and fill the view with values that come from the model. On mobile apps, this means making a network request to the backend to fetch the models and render the results to the user interface. On the server side, this could require querying the database to load the models and then send them back to the client, in JSON or HTML format. Controllers tend to be heavy weight in that they hold a lot of logic required for the application to function, so often there will be many separate controllers, usually separated by screens in the app. For example, you may have a user profile controller or a glucose level input controller. Additionally, controllers contain the business logic of the application, deciding how to respond to events based on user input, dictating what screen to present to the user, when to refresh data, or what errors to show to the user.

The view is typically what's rendered on screen and presented to the user, its responsibility is primarily how it should layout itself. It may have to resize to fit text on screen, or listen to gesture events, respond to screen rotation, etc. The views in an application are thought of as “dumb” in that they should never contain business logic and they never tell other parts of the application what to do. They don’t know about the controllers or the models, they just contain the layout logic. For example, if there is a business requirement to hide a label in a view whenever a value in the data model doesn't exist, then the controller contains that logic to hide the view, the view doesn’t hide itself. On the server side, a view can be thought of as how the data over the network should be
formatted depending on what system requests it, the mobile app itself or a web browser. For example, you may want to use HTML for mobile web (Safari, Chrome, etc.) and JSON for the native mobile app. Both languages may use the same underlying data model, but the server response would look different depending on the type of client used.

System Design

The server is configured to use a particular domain name and defines a set of back-end APIs that the client may use. While the server it is hosted on Amazon, it doesn't require any specific Amazon SDK to function since networking libraries are built into the GO programming language. It mostly maintains the database that stores all of the user’s information, health and profile data, as well as system configurations and acts as a public interface to the database used by the mobile application so that it may read and write to it. The iOS application uses a built-in networking framework that allows communication to the server and SDK to integrate with the wireless glucose meters, called Agamatrix. There will be a screen in the app to configure and control that specific brand of device for the prototype. In the future more devices will be added, which would require implementing multiple SDKs, one for each manufacturer.

The built-in iOS libraries that will be used are called UIKit, NSURLSession, Foundation, and CoreBluetooth. The parsing of data to and from the server in JSON format is handled by the NSJSONSerialization framework and a method in Swift called Codable.

Data sent to and from the server is done by using a representational state transfer, or REST design, as opposed to a web socket which pushes data in real-time to the client. Since the features in the application don't require real-time updates, REST is the
preferred choice. Networking requests are created and sent to the server by the mobile app based on user interaction as the user navigates between views. For example, if the user taps a button that causes a transition to another screen in the app, there is an underlying controller that knows what data model is needed to display to the user and what API to use. It establishes a network connection to the server, requests the data it needs, then terminates that connection. The downside to using REST over sockets is that the data on the server may change and the app has no way of knowing until it reloads the data again. For example, a system administrator could update a user's last name in the database while the user is looking at the profile page. The last name won’t automatically update on that screen, the user needs to navigate away and back to load the page again. This limitation is acceptable because that situation is quite rare and to implement a real-time socket system to solve that issue is very complicated and the costs outweigh the benefit to the user, and the company.

System Functionality and Prototype

The first screen the user sees when launching the app is the on-boarding screen, which is a view that describes the platform, some of its features, and how the user will benefit, also known as the value proposition. It may contain a button that presents the user with an alert to allow the app to access. The user is brought to a registration page where they can create a new account using their email address, which is also their username. On successful registration their information is stored in the user table of the SQL database. Password credentials are also stored on the device in their keychain, so that they are automatically logged in each time they launch the app. They can navigate to
a user profile page where they can store personal information like gender, weight, height, medications, etc., also stored in the database within the user table. For the prototype, the application will require an Internet connection, either through WiFi or by using cellular data. There is the possibility to support “offline” mode for users without Internet, like when on an airplane, but that is a feature that can be rolled out as an update since it's assumed the majority of users will meet this requirement.

There is a page to enter in your current blood glucose level, both manually and automatically by using a Bluetooth connected meter. The data collected on that screen is the glucose level (integer value), an optional comment (string value), and the choice to specify a date and time, or use the device's current date and time, with a submit button. This data is stored as a row in the glucose log table in the database, both on the device and on the server. It can also get these values by connecting to one of the supported wireless glucose meters. Initially, there will be support for a single device and other devices will be added in the future. This is because each manufacture must have an available iOS SDK in order for the app to integrate with it and access its data, which for most meters isn’t yet available. The first device to be supported is called the AgaMatrix Jazz Wireless 2. At the time of this writing it's listed for sale at $20.99 and the company has an SDK that allows developers to easily integrate it into the application. Users can setup and configure the meter under the device configuration screen. In order for it to find and connect to a Bluetooth device, the user must explicitly grant the app access during the launch and/or on-boarding process. It's not clear if the company offers their SDK free of charge, or if there is a flat or recurring fee. For the purpose of this proposal, I am assuming they will provide it free of charge because the device will be promoted as a
preferred meter to use with the app.

A very common feature amongst diabetes tracking apps is the ability to track caloric intake. The reason is because the data can be helpful to generate an overall health profile, providing indicators to which type of diet causes unusual fluctuations in insulin levels. There is a separate screen in the app to allow users to input each meal throughout a day. The information collected about each meal is its category, a description, number of carbohydrates, calories, and fat. The data will be stored in the database under the meals table. Not all the information will be required, except choosing a category and entering the estimated calories and description.

Exercise is another very important component to managing diabetes, so there is a page for the user to input their exercise routine. Data collected will be the type of exercise (running, walking, swimming, etc.), the duration, calories burned (reading the value from a treadmill), time of day, and an optional note. Some of the fields will be optional so that it doesn’t prevent the user from entering in information. This data will be stored in the exercise database table. By integrating with the Apple HealthKit SDK, some of this information can be automatically read into the system from the operating system itself or other fitness apps. For example, the device already tracks the number of steps and altitude climbed throughout the day which can help get a general sense of the user’s activity. So even without any user input, the app will be able to gather some fitness information.
Application Testing

This section describes the type of testing required when building a mobile application so that the features function as required and that it doesn't unexpectedly crash. There are several different methods used to reduce the chance that a bug could make it into a publicly available release. Also, there is a potential to inadvertently introduce an issue, completely unrelated to a feature being worked on because sections of code typically depend on other parts, so modifying one line can sometimes affect a completely different area of the app. That is why each release requires considerable manual testing by running through the same scenario for each version to detect when something doesn’t work the same way as the prior release.

Unit testing is a concept in software development where engineers write code that tests the code they've just written. It's basically a way to prevent someone from writing code that accidentally breaks other parts of the codebase and is typically used when there are more than one person working on the same project at a time. An example would be a typical function that uses input and output values, can be tested by passing in all the variations of inputs and then checking that the outputs are what would be expected given each input. Such unit tests are usually run automatically each time an engineer attempts to make a change to a shared codebase. The tests run using the new code and alert the user if something occurs that is unexpected. If the tests fail for any reason, then their proposed code changes are blocked from going in, forcing the engineer to investigate why, and make appropriate changes where needed. When implementing unit testing, there is a metric commonly used called code coverage, or the percentage of lines of code that have tests written for them. Managers or technical leads often set goals for their
organization to reach and maintain a certain percentage. As an example, a mobile application that has 60,000 lines of code and a test coverage of 80%, which is considered a high percentage, equals 48,000 lines tested. Unit tests aren’t a replacement to manual tests at all because they aren’t customer facing and cannot test everything a user experiences. They are simply there to help an engineer understand how a piece of code should work when they weren’t the one who wrote it.

Regression testing is a process by which the application is manually tested for quality assurance (QA) before it is released to the public. QA staff can have either a technical or non-technical background, but their job is to manage the releases by checking a number of features throughout the application. They execute what are called test suites, or repeatable actions that should always work as expected. For example, there may be a suite to test the new user registration flow. The test itself lists specific actions a user needs to take, in what order, and what conditions would be considered a failure. What happens when an email address already exists, does the strong password validation still work, was the account created successfully, etc. The definition of a regression is when an issue or bug arises in the new version that didn't exist in the prior one. If the issue is severe enough and negatively impacts the user experience, then it's considered a release blocker in that you cannot release the version until the issue is fixed. If, however, a bug exists in both the prior and new versions, then it's not classified as a regression; it's considered a known issue, still a bug, but would be considered lower priority.

A lightweight regression test is often referred to as a “smoke test” which means to validate that a basic change has been made, or that the build functions at a high level. It doesn't go into nearly as much depth in terms of the number of areas to test with all the
variations that go along with them. Smoke testing usually occurs only after a build has already passed a full regression test but since then, a very simple and safe change has been made, and you just want to verify that it's made it into the release. An example of this would be updating an icon or changing the wording on a translated string after the build has already passed regression tests. What is considered a safe or unsafe change is determined by the software engineer who did the work and sometimes the work is verified by a manager or technical lead.

Integration testing is similar to unit testing but instead tests the interoperability of two discrete software systems that exchange data between one another. The mobile app relies on the software running on the server to return data to the app in a specified format, while the server requires that the data sent by the app also be in the correct format. To achieve this, schemas are used to represent each data model that contain the property names, data types, and weather or not the value is optional (can be left out). The tests run automatically against the schemas during a back-end deployment to verify that data integrity isn't compromised.

User acceptance testing (UAT) is the process by which a particular version of the software is manually tested to verify that particular features meet their intended business requirements. It is the very last phase before a public deployment (production release), and the last opportunity to catch any bugs that slipped through any previous testing steps. This type of testing focuses more on the business impact, so the build is usually shared amongst a larger non-technical group of people, such as business stakeholders, investors, and sometimes even those outside of the organization to get feedback. A UAT build in the software development industry may also be called a beta release.
This section describes the various features that the app will contain and why they will be critical in order for the user to successfully manage their diabetes, reach their overall health goals, unlock points throughout their achievements and trade those points in for retail gift cards toward purchasing health-related products. Additionally, the user experience is demonstrated by mock screens with a description of functionality that is included in each mock. The purpose of this section is to make the app structure evident in terms of screen flow. The screen details document the functionality called for by system users as well as the input and output requirements of the app for a given feature. The application will initially support the English language since that is the common language of the target demographic, however, Spanish and French will be added later. Measurement values on the server will be stored using metric units, and then automatically converted to whichever preference the device is set to use. For example, users who have configured their device's settings to use feet instead of meters will see feet and inches, while the actual values sent to and read from the server are in meters or centimeters. The conversion from metric to imperial will take place within the app for display purposes only.

One of the primary features of this application is to collect data on blood sugar levels from the user over time, whether that be by manually entering in each reading, or by connecting to a wireless glucose meter to read the data. Since this feature is likely to
be one of the most frequently used one, it should be easily and obviously accessible, by placing a large plus icon at the top right of the screen. As part of this feature, the app will support syncing with a wireless Bluetooth glucose meter to import blood sugar levels.

Another important feature is meal tracking or caloric tracking, which provides a way for the user to monitor the meals they eat throughout the day. The argument for why this is important for a person with diabetes is that it forces them to pay closer attention to their overall food consumption and sheds light as to areas of improvement. It provides additional data points to the user that can be helpful and shed light on how certain foods cause undesirable fluctuations in blood sugar level, so that modifications to their diet can be made. This feature will be accessible via the same plus icon mention above on the top right side of the screen. The user can choose from a list of already entered-in meals or to create a new one. Adding a new meal requires picking a category, description, expected calories, optional fat and carbohydrates, the time of day, as well as any other optional notes. There is also the ability to take a photo of the meal and save it as a reference.

Exercise is another very important component known to help control diabetes from getting worse and even reverse it all together, so it's also a very important feature for the application. The plus icon allows the user to begin entering a new log, allowing the user to enter-in details regarding their fitness activity. Here they will also have the ability to integrate with Apple HealthKit and import data from the device itself or other apps like fitness trackers and wearable devices such as the Apple watch or Fitbit. Integrating with HealthKit is an important feature because it also includes data that the phone collects such as steps walked per day or number of stairs climbed. It is used as a way to paint an overall picture of how active a user is on a given day. If a heart rate
device is synced to your phone, then it will share the data with the app which can be used to determine the number of calories burned.

A typical use case of the application is also whenever a caregiver administers the glucose tests and checks on the person's progress. To support this ability, there is also a feature where the patient may grant read access to the caregiver and share the data collected by the app with them. This allows family members not living in the same household to be able to check on a loved one trying to manage their diabetes.

The app will allow the user to set reminders for certain activities like logging glucose level or going to the gym. This feature is called reminders and will be accessible when logging a new entry for a ritual. If applicable, reminders can also be viewed and edited by a caregiver, and they will be notified when those tasks are completed. Apple push notifications will be used to send the reminders to the device from the server, along with other alerts such as whenever they've reached a milestone or if they've unlocked a redeemable reward, like a product coupon or gift card. The user has to agree to receive notifications when they first launch the app via a pop-up alert and can disable them in the device's settings. The user has the ability to control which types of notifications they receive under their profile settings, so that they only receive the ones that they care about. Push notifications can also be used to increase user engagement because it's very common for someone to tap on the notification as soon as they see it, which directly launches the app. The message can contain meta-data within the payload that the app uses to provide additional context to the user, such as linking them to a particular page within the app, like the glucose page for example. It's an opportunity to enhance the user
experience, which often is thought of as a differentiating factor amongst the competition.

Application Features – Screen Mocks and Description

1. Registration Screen
   - Access to account registration via email and password
   - Access to Terms and Services Agreement
     - Sign In access

2. Sign In Screen
   - Access to sign in via email and password

Figure 3 & 4. Registration and Sign In Screen Mocks
3. Homepage Onboarding and Standard Homepage Screens

Feature description starting from the top to bottom of the screen:

- Large plus icon allows users to log any ritual
- Top summary bar which includes:
  - Date of the latest log entry
  - Summary of health metrics which includes:
    - Glucose
    - HbA1c
    - Insulin
    - Calories
    - Mood
- Analysis Chart which summarizes glucose levels over time
  - For new, first time users a popup window will be displayed to communicate how to get started. This brief tutorial will include a series of five screens.
- Access to ‘My Points’ progress bar
  - Shows the level that the user is at; there are a total of three levels
  - Displays the user’s current points on the bottom left of the progress bar
  - Includes the remaining points the user has to earn to reach the next level at the bottom right side of the progress bar
- Access to the Tab bar at the bottom of the screen which includes access to the following screens:
  - Home
  - Points
Figure 5 & 6. Homepage Onboarding and Standard Homepage Screen Mocks

4. Points Screen

Feature description starting from the top to bottom of the screen:
• Large plus icon allows users to log any ritual

• Points progress bar, starting at level 1 to level 3
  o Level 1 consist of 60 points which represents a retail gift card of $5
  o Level 2 consist of 90 points which represents a retail gift card of $10
  o Level 3 consists of 110 points which represents a retail gift card of $20
  o In this mock, the user has earned a total of 20 points. The progress bar easily notifies the user that they are at level 1

• Points serve as indicators of achievement and progress

• Levels serve to recognize a user’s accomplishments

• List of rituals and potential points for each ritual
  o Users can earn 5 to 20 points for logging a new entry for 10 different rituals, which are:
    ➢ Medication; 20 points
    ➢ HbA1c; 20 points
    ➢ Glucose; 20 points
    ➢ Keytones; 10 points
    ➢ Blood Pressure; 10 points
    ➢ Carbs; 10 points
    ➢ Activity; 5 points
    ➢ Weight; 5 points
    ➢ Water; 5 points
    ➢ Mood; 5 points

• Access to the Tab bar at the bottom of the screen
5. Add Entry Screen

- Users are notified of their latest log entry which includes the date and time
- Notification message at the top of the screen lets users know how many points
they are away from reaching the next level, in this mock the user is 40 points away from reaching level 2

- After selecting one out of the 10 rituals to log an entry for, the entry is automatically date and time stamped

- Before saving the entry, users can:
  - Tag the entry
    - List of tags include symptoms and other labels such as cravings, helplessness, illness, tiredness, palpitation, anxiety, headache, nausea, slowness, after meal, holidays, menstruation, sober, etc.
  - Set a daily or weekly reminder for the next time they need to do the same ritual
  - Add notes
Figure 9. Adding a tag to an entry screen mock
6. Health Screen

- Large plus icon allows users to log any ritual
- Analysis
  - Users can select one or more rituals to view in the chart over the course of a
    - Day
    - Week
    - Month
Within a given timeframe which the users can specify

- This data can be exported via
  - PDF
  - CSV
  - Excel

- Access to the Tab bar at the bottom of the screen

Figure 12. Health screen mock
7. Inbox Screen

- Large plus icon allows users to log any ritual
- Users can view promotional, progress and update messages in their inbox
- Access to the Tab bar at the bottom of the screen

Figure 13 & 14. Inbox and Other screen mocks

8. Other Screen

- Large plus icon allows users to log any ritual
- Profile access
- Settings page access
- Help access
- Invite Caregiver
- Invite Doctor
- Access to the Tab bar at the bottom of the screen

Figure 15 & 16. Profile and Settings screen mocks
Chapter IV.
Discussion

The aim of this study was to prototype a medical device for T2DM patients to better manage their diabetes by reminding and motivating them to take and record daily rituals such as monitoring blood glucose levels and HbA1c, healthy eating and physical activity to name a few. In exchange, patients can earn reward points that can be used toward gift cards at retailers such as Whole Foods, CVS, Walgreens, etc. The underlying benefit to the reward program is that patients are motivated to practice daily self-care. As mentioned previously, FDA approval warrants this application as a medical device which differentiates this application from all the other diabetes management apps in the market today. Additionally, with FDA approval and recognition of a medical device, patients can benefit from paying lower insurance premiums. Finally, it has been shown that patients who use medical apps to manage diabetes develop better self-care behavior. Research conducted by Kebede and Pischke shows that one third of respondents (with type 2 diabetes) reported using diabetes apps for self-management. The self-care behavior score among diabetes app users was drastically higher compared to non-users. Additionally, MySugr app offers gamified solutions for managing diabetes with over 1.5 million users. Their gamification element allows users to earn reward points while managing their diabetes. A study by Debong et al. explores the influence of MySugr app usage on patient’s overall diabetic health. Their finding concludes that MySugr apps approach to helping patients manage their diabetes correlates to significant improvement of blood
glucose levels. Specifically, the mean blood glucose levels had decreased by 17.9 percent at six months (Debong, 2019). While this study is not conclusive and cannot be generalized to other medical apps, it does provide insight into real-world use of a health app for diabetes management. At this time, it is difficult to find any available studies that directly link financial incentives to better diabetes outcomes, however, there is research available for diet management apps that use similar motivational structures as what we’re proposing. The assumption is that people who use a diet and exercise management app respond to similar motivating factors as someone who has diabetes and uses a diabetes management app, since both require health related behavioral changes. Purnell et al. identified that financial incentives are strongly tied to having a positive short-term effect on dietary outcomes. Research by Bachireddy, et al. similarly concluded that using financial incentives as motivation to make behavioral changes to a patient’s physical activity had a significant positive effect when the payment was constant rather than increasing or decreasing. It reassures that there’s no argument whether or not financial incentives lead to better health outcomes; the debate is how to determine the amount and frequency which leads to a lasting positive effect, for which additional research is needed.
VIII. References - Works Annotated

   • Article describes hosting the server code and backend API
   • Analysis of financial incentive to promote physical activity
   • Review summarizes seven strategies for evaluating and selecting health-related apps
   • Reviews include apps for: weight loss, chronic medical conditions, mental health, medication self-management, cancer prevention and management, HIV prevention, sports injury prevention, and smoking cessation
   • Analysis of diabetes app user feedback
   • Comparison of user preference to current available diabetes apps
   • Provides a real-world assessment of mySugr app
   • Article describes the programming language which is swift to build the app
   • Article describes how the views are constructed
   • Article describes the way to sync health data across apps
   • Review provides statistics about diabetes
    • Review provides overview of a1c, FPG, and OGTT
    • Review offers a comprehensive view of impact of mobile phones in health care
    • Presents strength and limitations in mHealth research
   - Systemic review on the effectiveness of mobile technology adoption in health care and public health globally
   - Lists patient reported outcomes for medical device intervention
   - Review of diabetes self-management education standards
   - Article describes the installation and configuration of server side programming language
   - Articles describes what JASON is and how it works
   - Report on how effective a mobile app is for two patients with diabetes
   - Discuss issues and strategies of mobile app intervention for diabetes management
   - Article describes John Hancock new business model
   - Review of popular diabetes apps
   - Investigate the association of diabetes app use and cumulative self-care behavior
   - Offers a comparison between HbA1c and FPG in diagnosing diabetes
   - Evaluation of the early impact of digital health coaching service on weight loss and blood pressure management
   - Coaching services provided for the management of cardiometabolic conditions: diabetes, hypertension, hyperlipidemia and prediabetes
• Article describes Mango Health’s partnership with pharmacy benefit management organization
• Presents two initiatives designed to enhance the science of mHealth
• Address the growing need for high quality mobile health research
• Provides a systemic review of financial incentives to change dietary behavior
• mHealth statistics and facts
• Offers the number of mobile phone users worldwide from 2015 to 2020
• Provides the number of available apps in the Apple App store from July 2008 to January 2017
• Indicates the number of mHealth app downloads.
• Shows the most popular categories in the Apple App store in May 2018
• Definition and classification of Mobile Medical Applications
• FDA’s most current thinking on Mobile Medical Applications
• Articles describes how to query the database