Sugar Coded

Final “As-Built” Report

May 6th, 2018

Project: Prediabetes Intervention Mobile Application

Sponsor: Dr. Natalia Dmitrieva

Mentor: Dr. Eck Doerry

Team Members:

Chantz Spears

Julian Shak

John Bassler

Alfonso Martinez

Table of Contents

Table of Contents 1
1.0 Introduction 2
2.0 Process Overview 3
3.0 Requirements 3
4.0 Architecture and Implementation 9
4.1 Module and Interface Descriptions 11
5.0 Testing 16
6.0 Project Timeline 22
7.0 Future Work 25
8.0 Conclusion 25
9.0 Glossary 27
10.0 Appendix A: Development Environment and Toolchain 28

1.0 Introduction

Diabetes mellitus is a chronic condition characterized by elevated blood glucose levels. The reason for these elevated blood glucose levels is due to one's insulin, a hormone produced by the pancreas to filter sugar from your blood, not functioning properly. It is a disorder that currently affects over 30 million Americans. The vast majority of people with diabetes (~90%) have Type II diabetes. Those with Type II have grown resistant to their body's insulin over a long period of time (usually decades) and have a reduced capacity for blood sugar regulation. Causes of type 2 diabetes are often attributed to having a family history of diabetes, as well as a poor diet and exercise routine. Because of their high blood sugar levels, people with type 2 diabetes suffer from frequent hunger, increased thirst, fatigue, and blurred vision. Since diabetes is considered to be an irreversible condition, people with type 2 diabetes are left only to manage their condition for the remainder of their lives through various recommended strategies, including monitoring of blood sugar, carbohydrate intake, and medication use. However, type 2 diabetes prevention programs can successfully lower diabetes incidence.

One such program is the lifestyle arm of the Diabetes Prevention Program (DPP). This was designed to prevent future cases of diabetes by finding high-risk patients and offering moderate lifestyle and diet changes. These high-risk patients are usually overweight and have intermediate elevated blood sugar levels that do not yet meet the criteria for type 2 diabetes. This state describes the condition of prediabetes, and the US Department of Health and Human Services estimates that a third of US adults meet the criteria for prediabetes as of 2015. The DPP's goal is to reduce the patient's body weight by 7% and, in doing so, the DPP is successful in delaying and even preventing a future diabetes diagnoses. On average, a patient's risk of diabetes onset is reduced by 58%, as well as 45% of program participants self-identified as an ethnic or racial minority.

A problem these prevention programs are facing is the completion rates among their participants. Native Americans have the highest withdrawal rate from the DPP with approximately one-third prematurely dropping out of the 16-session program. Reasons for not completing the program or what can lead to someone dropping out are complex and hard to grasp. The current technique for finding out these reasons is through Ecological Momentary Assessment (EMA). In EMA, the methodology is to frequently ask the participants to record what they are feeling in the moment as a means for researchers to find disparities. Since it is currently implemented through phone-surveys conducted by the researchers, EMA is expensive, time consuming, and intrusive for research participants. In order to improve retention, SugarCoded is deploying a mobile app and web portal to aid with EMA research. This will allow surveys to be completed through participant's mobile phones and for researchers to prepare and deploy surveys to these participants via the web portal. By aiding researchers with our product, we can help increase the effectiveness of EMA research and in turn, help increase retention rates in diabetes prevention programs.

This report will show how our team has developed our requirements for our product as well as it's architecture and how it was implemented. We will demonstrate how produced and tested our product, the timeline we followed during production, and goals for future work on SCHEMA. Through our report, we will show you the final outcome of the final product as well as sharing the Capstone experience our team went through to create our final product.

2.0 Process Overview

Our project development was divided into two main components, a mobile application and a web portal. Due to the short period of time that we had for the capstone, we used a very rapid development process which involved attacking different modules within each of our separate components according to priority. The initial phases of development included establishing the base functionality of each component (i.e., setting the bare bones skeletons). After ensuring our components were fully functional, we started UI design. This stage of development involved iterative meetings with our Capstone sponsor to make sure that our project met our given requirements/sponsor's expectations. Chantz and Alfonso worked together to develop the Web Portal, Julian developed the Mobile Application, and John helped provide some necessary functions to both components. Our team used GitHub as our version control system, to ensure all members of our team had access to the code base. The team met once a week to delegate tasks and discuss progress on current ones. If more meetings were needed for code merging or presentation practice we would find a common time that worked for everyone. Slack was the main form of intrateam communication and served its purpose well. The team met all deliverable dates and worked cohesively as a unit throughout the entire school year.

3.0 Requirements

To develop the following requirements the team has been through multiple client meetings since the start of the project, bouncing ideas off of Dr. Dmitrieva until we found feasible requirements that satisfy her needs. After our technical feasibility was laid out we discussed among the team requirements that can be implemented using the technologies we chose to work with. To start of the section below we go into the domain requirements which lay out the general specifications the client expected out of SCHEMA.

Domain Level Requirements

The following are the features from the domain perspective:

DR1 **Keeping Track of Participant's Data**

Each research participant will need to have their information easily identifiable, and stored and accessible separate from everyone else.

<u>DR2</u> **Participant Self Reporting**

Users need to be able to report on key behaviors such as when they have eaten something, and how they were feeling at the time.

<u>DR3</u> **Send reminders / direct users to questionnaires**

The administrator needs to be able to set push notifications to be delivered to users at certain intervals and direct users to additional questionnaires.

<u>DR4</u> **Question Delivery Protocols**

Must support creating, editing, and delivering complex question protocols including the ability to specify timing, events, or user initiated protocols.

<u>DR5</u> **Administrative Abilities**

The researcher requires the ability to setup and manage study participants, and download datasets from the research for analysis.

<u>DR6</u> **Fitbit data collection**

Collect and store individual users Fitbit information which contains data such as steps taken and heart rate at any given time.

<u>DR7</u> **Hybrid application**

The mobile application should work for both Android and IOS as to not alienate the majority of research participants.

The following requirements sections will take a deeper look into the finer points of our project and relate them back the domain level requirements. Specifically, each requirement is attributed to a single domain level requirement, where '**DR#**' stands for domain requirement, '**FR#**' stands for functional requirement, and so forth.

Functional Requirements

The functional requirements entail every function that our product will need to have/perform. The functions will be presented initially at high level and gradually increase into more detailed lower level functionality.

<u>DR1.FR1</u> **Secure login**

- Users will each be set up with a single account that will have a secure login to protect their information, this information will be kept confidential and is for research staff eyes only.

- *App must allow for an ID interface for users to securely log in and out of the account they are given to participate in the study*
- Allow users to stay logged in even after exiting the application

DR1.FR1.1 Store usernames and passwords in a secure database
- Our database will need columns for both usernames and passwords
- The database needs to be secured and only accessible to the research team

DR1.FR1.2 HIPAA compliance
- Login process and data management must be HIPAA compliant
- Use a HIPAA compliant cloud data storage solution such as AWS
- Safeguard PHI data

DR2.FR1 **Log a user's health states**
- User consistently logs events such as eating and exercise through the mobile app
- Alongside each event log the user will be asked questions to determine their current momentary stressors, social support, physical context, food availability, etc.

DR3.FR1 **Notify users of questionnaires via Push notifications**
- Push notifications will be used to:
  - Remind users to log events in time based questionnaires
  - Navigate the user to the questionnaire

DR4.FR1 **Question Delivery Protocols**
- Participants should be able to skip certain questions and actions depending on their subgroup, previous questions answered, and factors chosen by the researcher.
- Participants must be placed into specific subgroups depending on their answers to questionnaires and baseline questions.
- Individuals can specify interruption times to a certain degree, namely their sleeptime

DR4.FR1.1 Branching logic
- Conditional statements will determine what a user is asked, what they are allowed to skip, etc. based on information stored about this user in the database.
- Conditionals determine interruption windows

DR5.FR1 **Create an administrator web portal**
- Utilize web2.0 ideology to create a web application for the research administrator(s) creating the studies

<u>DR5.FR2</u> **Set up and manage research participants**

- Through the web portal the researcher will set up each research group with a ‘study-specific’ to be admitted into the study
  - On the participants end they will need to enter this code to finish their registration
  - The code will be secured in our database and checked against the participants input to insure only verified users are participating in the study.

<u>DR5.F3</u> **Data viewing**

- Researcher can view data gathered from the participants in near real time
  - This data will be pulled from our cloud hosted database
  - The data will be displayed using a chart based javascript tool such as D3 or chart.js

DR5.F3.1 **Downloading data sets**

- The ability to download .csv files generated from our cloud data storage such as Excel files

<u>DR5.F4</u> **Manually group participants**

- Researcher can select users based on how they have answered certain questions or individually to assign them to certain sub pools
  - We will query the database to find who the researcher is looking for, then update the individual’s ‘group’ field

<u>DR5.F5</u> **Send out notifications**

- Researcher can send out notifications to all or specific individuals to alert them of anything they should need
  - Push notifications will be sent to the mobile applications detailing the message the researcher selects
  - Have the researcher select which participants they would like the notification to reach via various selection boxes
    - Query the database to determine which users to send the notifications to.

<u>DR6.FR1</u> **Gather users Fitbit data**

- Collect users steps, heart rate, and more to deliver to the researcher web portal.
  - Fitbit data collected from Dr. Winfree’s web portal and placed into our cloud database. From the database we relay the Fitbit info back to the researcher.

<u>DR7.FR1</u> **Mobile application runs on Android and IOS**

- The mobile portion of our project needs to be able to seamlessly run on either operating system, Android or IOS.

    - Developing the application in a hybrid framework such as Meteor or Ionic allows for easy integration into either OS.
- The application will be backwards compatible with recent OS versions

Performance Requirements

The performance requirements detail the conditions in which out product is supposed to function. These functions are measurable and testable, unlike the functional requirements and include some quantitative qualities.

DR1.PR1 **Quick account creation**

- Users will receive a Unique ID from researcher
- Estimated time to fill out single use form
    - Unique ID - ≤20 seconds
    - Button submission - ≤2 seconds
- Process should take ≤ 30 seconds

DR1.PR2 **Fast log on and quick turnaround for users to log events**

- Application will keep users logged into their accounts for quick use
    - Pressing re-login - ≤ 2 seconds

DR2.PR1 **Entering Participant-initiated logs and time based logs**

- User logs emotions, positive experiences, stressors, and behaviors after an activity or responds to a time based query
- Time to enter form respectively
    - Type of log - ≤ 3 seconds
    - Log Data - ≤ 3 minutes
    - Open ended voice to text box ≤ 20 seconds
    - Button submission - ≤ 2 seconds
- Process should take ≤ 3.5 minutes

DR3.PR1 **Mitigate Intrusive Interruptions**

- Application will take in user's initial sleep hours, and any other busy hours in order to create an interruption time for that specific user, which determines the frequency of the application's push notifications
- User's will be choosing a time to take "End-of-Day" questions
- Time to enter form respectively
    - Work hours entry - ≤7 seconds
    - Sleep hours entry - ≤7 seconds

    - Other hours entry - ≤7 seconds
    - “End-of-Day” entry - ≤7 seconds
    - Button submission - ≤2 seconds
- Process should take ≤ 23 seconds
- Interruption time determines frequency of the application’s push notifications

__**DR4.PR1**__ **Minimal need for application usage training**

- Simple and elegant design so user is able to quickly and efficiently understand the sections of the application
- Should take user no more than 3-8 minutes to fully understand UI if they are comfortable with modern mobile applications, otherwise 12 - 20 minutes

__**DR5.PR1**__ **Ability to add participants**

- Any researcher will be able to add any questions to all participants
- Time to enter form respectively
    - Enter Unique ID - ≤5 seconds
    - Enter name - ≤5 seconds
    - Enter dob - ≤5 seconds
    - Enter gender - ≤3 seconds
    - Submit - ≤ 2 seconds
- Process should take ≤ 20 seconds

__**DR5.PR2**__ **User data operations**

- GET, PULL, PUSH Requests: Should take no more than 5-10 seconds to retrieve and send User data

Environmental Requirements

__**ER.1**__ **Fitbit integration**

- Allow further data collection through our app by integrating data collected by from a Fitbit worn by the app user.
- Require participants to connect to the internet at least once a week to allow data from Fitbit to be offloaded

__**ER.2**__ **The app will be developed for both Android and iOS**

- App functionality must be consistent across both platforms to ensure app users have similar experiences

- Development of the app will be done in a hybrid environment

<u>ER.3</u> **Prevent researchers from seeing personal info of participants**

- Allow for app users to be anonymous to the researchers tracking a participant's progress

The following section, Architecture and Implementation will examine how the project requirements were realized into a working product.

4.0 Architecture and Implementation

Delving into a high level architectural overview of the project can provide a solid foundation for understanding how the system produces the desired behavior. In figure 1 featured below the systems four main components can be seen: the Mobile Application, the Web Portal, the Database, and the Wear Ware Portal.

Figure 1: Architectural Diagram: The yellow components represent modules that Sugar Coded is building from the ground up. The red component is denoting the database that will link SCHEMA together and allow data collection and protocol administration between modules. The grey components are comprised of third party software provided by Dr. Kyle Winfree to collect participants Fitbit data.

<u>Responsibilities and Features</u>

The initial component that SCHEMA's users will interact with is the Web Portal. At a high level the Web Portal will allow researchers to:

- Create and publish studies
- Add participants to studies
- Create questions to be administered

- Control all time based protocols attached to the studies
- View study results
- Download results to a .csv

The Web Portal will control and provide all necessary functions a researcher would require to successfully administer a study. The Mobile Application will be what attaches study participants to a particular study and allows them to provide researchers with study results.

The Mobile Application will be utilized by study participants to:

- Attach themselves to studies
- Answer study questions (time based or user initiated responses)
- Receive study notifications

The Mobile Application will feature a question backlog section so if a user is unable to answer a time based question right away they can return at a convenient time to answer their questions. Starting the application for the first use will require participants to fill out a baseline questionnaire should the researcher desire and set up suitable times for the application to send them notifications.

The Wear Ware Portal will be used to gather participants' Fitbit data. Should a study issue participants Fitbits this portal will keep track of all Fitbit related data which can then be sent to the Web Portal for researchers to view.

The Firebase database will be the main connection for all of SCHEMA's individual modules, storing and pushing data between them when needed. The following section describes in finer detail the communication mechanisms the project utilizes that are heavily reliant on this centralized database.

Data Flow

Figure 1, above, shows how data will flow into and out of the database over the course of a study, the following describes a step by step of the process. After a study's creation its components are saved in the database and pulled into the Mobile Application. The answered questions in the Mobile Application and Fitbit data are uploaded to the database whenever a participant is connected to a network. These participant provided results are then delivered back to the researcher's dashboard on the Web Portal for their viewing needs. This is a cyclic process that continues through the duration of a study; questions are administered to the participants, they provide answers, and the researcher views the results. As mentioned earlier upon a studies completion the collected results may be downloaded to a .csv file.

SCHEMA’s architecture is influenced heavily by the client server and component based architectural styles. Clients make requests to the server, where in this case the server is a database with application logic represented as stored procedures, that then directs the client how to proceed. SCHEMA is also sectioned into reusable functional and logical components that exhibit well-defined communication interfaces.
In section 4.1 an in depth view of each module and the roles associated with it are broken down, this will help provide greater clarity of the project in its entirety.

4.1 Module and Interface Descriptions

Section four, Module and Interface Descriptions, will provide an in depth overview of how each facet of SCHEMA will be composed, that is the classes each module will need, and the functions and variables within each class. The need and purpose of each class and how it furthers SCHEMA’s abilities will be explained as well.

Web Portal Module

Below are UML diagrams depicting the different components of SCHEMA and the classes they are composed of. The UML diagrams are accompanied with a description of the components responsibilities and services each provide.

Figure 2: Web Portal UML

Figure 2 is a UML for the Web Portal module of SCHEMA which is composed of 6 actors/classes: User, Participant, Subject, Study, Module, and Questions. The Web Portal will be the user's main hub for setting up studies and viewing the results. Below explanations will be given for any important variables and functions involved with the classes.

- User: This is the researcher or entity that will hold the most power in SCHEMA's hierarchy and the intended audience of Sugar Coded's product. Users will be able to create and manage multiple studies, view results in realtime, and export data after studies have concluded. The createStudy() function allows a user to set up a new study and fill in all the necessary information such as title, abstract, etc. ManageParticipants() allows a user to add, delete, or edit any participants information and change their status in the study. ViewData() allows a user to view all the currently gathered results from the study while exportData() allows the user to get the results in a .csv file. The createModule() and createQuestions() functions are what allows a user to customize a study to their liking. With these creation functions a user sets up what types of questions users will be asked, whether they will be notification based or self reported questions and with the setTimes() function they will be able to set the times for these questions to be asked. The following will describe any variables that are not implicitly inferred:
    - Title: How the user would like to be labeled i.e. Lead Researcher
    - Role: Whether or not the user is the owner of the study or just assisting with it.
    - Institution: The company or institute the user is associated with.
    - Department: The specific department within the users institution that they belong to.
- Study: A study is what the user creates and what participants are enrolled in. Studies have start/publish() and end() functions to mark when a study will begin pushing questions the participants and collecting data and when the study concludes and the participants are done answering questions. The study stores modules which control the protocols for questions, that is whether they are self reported questionnaires, or time based. The following will describe any variables related to a study that are not implicitly inferred:
    - Owner: The user in charge of this particular study.
    - Abstract: A description of what the study entails.
    - Modules: As mentioned above these are what hold the studies questions, protocols, and timings.
    - Participants: Studies keep track of which participants are associated with them.

- Module: Modules are able to addQuestions() to themselves or removeQuestions(). Adding a question allows a user to choose question type (multiple choice, slider, etc.), and if a time based question the user may use setupTimings() to schedule when the participants should receive notifications to answer the question. Each module is individually set up by the user to either be a self reporting module or time based module, which then would only contain the corresponding questions.
- Questions: Questions are the actual data gathering tools setup by the user for participants to interact with. The Questions class will collectAnswers() of users after each their answers are submitted, sending them back to the database which the Web Portal will then pull from to display to the user. The following will describe any variables related to a study that are not implicitly inferred:
  - Type: Whether the question is requiring an answer in the form of a textbox, radio button, slider, etc.
  - Choices: If a question should be multiple choice for example the choices will be the options a participant has to choose from.
- Participant: These are who will be providing all the research data for a study via answering questions or providing information of their own accord. Participants will be able to answerQuestions() and pickInterruptTimes(). Picking interrupt times allows a participant to do things like set their typical sleep time so that no notifications will be sent to their mobile device in that time block (discussed further in the Mobile Application Module section below). The following will describe any variables related to a study that are not implicitly inferred:
  - Answers: All of the responses a user gives will be momentarily stored and sent to the database.
  - Editable Record: Each participant has a record that contains all of the demographic information they entered, they may change this at a later date should something change.
- Subject: This class is essentially all the identifying parts of a participant. Should a user whish to do a blind study simply hiding the subject class or breaking the link to participants will completely anonymize participants. The variables in this class are customizable as they could contain anything a user wishes to collect on participants.

The following section detailing the Mobile Application Module, goes in depth on the classes and functions that compose the mobile portion of SCHEMA and the goals it hopes to accomplish.

Mobile Application Module

SCHEMA's mobile portion will be the main form of communication study participants have that enables them to provide results to a researcher. Filling out self reported and time based questionnaires provide a variety of ways researchers can collect data and SCHEMA's notification system will help keep the participants participating.

Figure 3 displays a UML diagram for the mobile application that we will be developing. There are four main actors/classes within our mobile application framework: Modules, Notifications, Users, and Studies. A study contains user participants which each have question modules associated to them. Each module in turn has researcher defined notifications associated with it. In the list below we go more in depth on the variables and functions that are associated with these four classes.

- Module: an admin defined questionnaire/information gathering survey that is configured through the study admin web portal. Within this module class, our mobile application will be able to query data from our database and with the displayModule() function it will display the module according to the settings configured by the study owner. A module includes:
    - Questions: admin defined questions
    - Time Intervals: time intervals in which modules need to be completed
    - Module Type: self initiated or time interval
    - Responses: the user's response to questions in the module
- Notification: we will be utilizing a local notification system to alert the research participants of any modules that need to be completed. Upon initial login, our mobile application will be gathering all of the time intervals in which each module

needs to be completed and we will be storing these time intervals into the notification system of the mobile device. Our notifications will include two functions, the first function is sendNotification(), which will trigger based on the time intervals stored in the notification system. The second function is the onClick(), which is triggered once the notification is clicked on. This function will take the user directly to the correct module by looking at the variables associated to every notification listed below:

  - ID: each notification will have an ID, which we will be using to associate different notifications to the type of module that they will be directed to upon click.
  - ScheduledTime: this is the admin defined time interval in which this notification will be sent to the user.
  - CallBack: this variable contains any additional information that we would like to associate to a notification to encapsulate any other information that will be sent to the module.
  - Title: this is the bolded text at the top of the notification, with a short description of the action required.
  - Message: this is a more detailed message to help the user identify which kind of module they are being requested to finish.

- User: a study participant is described in the above UML as a User. They are the main means of producing data for the researchers to collect through completing modules.
  - UserUniqueID: this generated ID will identify users while keeping their anonymity intact. This will also allow users to participate in multiple studies without creating new accounts.
  - logMeal(): the User uses the mobile app to complete a basic questionnaire for any meal they have during their time on the study. Some examples of information
  - submitModule(Module): upon completing a Module, a User can submit their completed Module through the mobile app to be stored in the database.
  - delayModule(Module): when the user receives a Push Notification to complete a module and they are currently unavailable to answer the Module (driving, receiving phone call, etc), they can delay answering the Module for a more convenient time.
  - editInterruptTimes(): a User will be allowed to edit the times the app will send notifications for their module. As each user will have different sleep patterns, customizing interrupt times will allow for better user interaction for logging data through the app.

- Study: Studies are created by researches within the web portal. The Study has an ID and the Module(s) that execute while the study is active.
    - ID: each created Study will be given an ID that uniquely identifies it. An ID can be generated for a Study with no modules or study participants.
    - Modules: each study created by a researcher will contain one or more modules pertaining to their study participants (Users). Modules define the study, allowing the study creator(s) to edit and customize the questions to put in a module.

The team was fortunately able to stick to our implementation plan fairly close and built a product to meet almost all of the client's requirements. The WearWare third party web portal for grabbing Fitbit data was not functional during our capstone so the Fitbit features of the project had to be put off for future work when the proprietary data can be gathered. Aside from Fitbit the final product closely resembles our original design, going far beyond initially planned features in some areas. In the next section, testing, the team has proven that SCHEMA is not just for show but functions as well.

5.0 Testing

To ensure SCHEMA's modules and procedures perform correctly the team has designed unit tests for each of our application's main functions. Since many of our functions make use of minor helper functions we opted to focus our tests on main subsets of our web and mobile applications. We have written the majority of our code in TypeScript and to help expedite our unit testing we will be leveraging the following tools, Mocha a testing framework, and Chai an assertion library. These tools both have first-class support for TypeScript which is our main motivation for choosing them, they will make writing unit tests far easier than without using any external libraries or testing functions. The modules listed below will be accompanied by tables providing the name of the module or function being tested, the equivalence partitions, boundary values, sample output, and expected output for multiple unit tests. To start off we will examine the web portal's tests before detailing the mobile application's tests later on.

Unit Testing

To ensure SCHEMA's modules and procedures perform correctly, the team has designed unit tests for each of our application's main functions. Since many of our functions make use of minor helper functions, we opted to focus our tests on the main subsets of our web and mobile applications. To help expedite our unit testing, we will be leveraging Mocha, a testing framework, and Chai, an assertion library. Both these tools have first-class support for TypeScript, which is our main motivation for choosing them, and will make writing unit tests far easier than without using any external libraries or testing functions.

Our full unit testing module results is too long to list within this report, but we can list below the major aspects we wanted to test through unit testing (namely our web portal's functionality to create, update, add, and delete modules, the branching logic of our questionnaires, our mobile app connecting to studies and the notification system). The modules listed below will be accompanied by tables providing the name of the module or function being tested, the equivalence partitions, boundary values, sample output, and expected output for multiple unit tests.

Study, Question, and Module Creation, Updates, and Deletion -
User's may add, update and delete numerous studies, modules, and questions within the web portal. When a user creates a new study, module, or question or updates an existing one all fields must be filled out with valid alphanumeric characters.

Unit Test	**Equivalence Partitions**	**Boundary Values**	**Sample Input / Action**	**Expected Output / Effect**
Valid Creation	Filling out all fields before completing creation	Any character can be used for all the fields except for dates which must be filled using given tool	Clicking create study having filled out every field	A new study is created in the database with included fields
Invalid Creation	Failing to fill out all fields before trying to complete creation	Empty strings	Leaving fields blank and clicking create study	User is notified all fields must be filled out
Update	All fields must still be filled out upon clicking update	Any character can be used for all the fields except for dates which must be filled using given tool	Clicking update study having made changes to fields	The database is updated with the new information for the study
Deletion	There is a study stored in the database	N/A	Clicking the delete button after selecting the study	The study is found in the database and removed

Branching -
After questions are added to a module their order and branching paths need to be determined. If the question a user is branching is a radio button they will need to provide branch paths for each possible choice rather than just one path for a text question.

Unit Test	**Equivalence Partitions**	**Boundary Values**	**Sample Input / Action**	**Expected Output / Effect**
Valid Branching	Filling out a path for every question inside the module	The path value needs to either be another question or “end” from a drop down menu which signifies the end of that questionnaire	Clicking on the add branching button after selecting a branch path for each option on the current question	The branch paths are added or updated within the database and the user is taken back one page to the module screen
Invalid Branching	Leaving branching paths blank	Leaving drop down fields empty	Clicking on the add branching button after leaving any dropdown blank	User is prompted to fill each field

<u>Mobile Application</u>

<u>Connecting to studies </u>-
This module handles the validation of unique User IDs that are auto-generated by our database for each participant that is registered within a study. Upon downloading our app, end-users are presented with a login page that prompts them for the unique User ID, which should be distributed by the study administrator. Any input is accepted into this field, however only when a valid unique User ID has been entered will the end-user be connected to a study. Invalid User IDs will present the user with an “Invalid Login” alert.

Unit Test	**Equivalence Partitions**	**Boundary Values**	**Sample Input / Action**	**Expected Output / Effect**
Valid Log In	Any valid unique User ID that is registered to a study.	Any typed input	User ID: 1234	User’s unique ID checked against database and let inside app
Invalid Log In	Any input that is not a unique User ID registered to a study.	Any typed input	User ID: 456	This ID is not a unique User ID registered to any study in our database, so an “Invalid Login” alert appears.

Notifications -
This module creates all of the notifications that will be associated with the Time-Initiated Modules (questionnaires) of a study. The Time-Initiated Modules that we are allowing the study administrator to configure are daily recurring Modules, such as “every 30 minutes”. Upon initial login, the mobile application will prompt the user to enter their approximate “sleep times”. These are approximate times in which this participant sleeps and wakes up every day. The notifications that we configure should not trigger between these times.

Unit Test	**Equivalence Partitions**	**Boundary Values**	**Sample Input / Action**	**Expected Output / Effect**
Create Time-Initiated Module (every 30 min)	N/A	N/A	A Module is created on the web portal and is configured to trigger every 30 minutes.	The mobile application will notify the participant to complete this Module every 30 minutes, except when they are sleeping.

Integration Testing

This application will be using a one time connection that begins with the web portal and goes to the Mobile application and a one way connection from Mobile application to the web portal. The researcher creates a study that will be downloaded by the user by logging into application with their unique id. Once the study has been downloaded the user will begin answering modules, and the answers to that module will be sent to the Firebase database, where the researchers can access using the web portal. Another application that will connect to our web portal is a third party application called Wearware. A researcher will be able to query Wearware in order to get fitbit data for a specific user. This next section will go into the different tests to make sure that data is passed around correctly.

Study Upload/Mobile Download - As mentioned previously, a researcher will create a study. The researcher will give the users their unique id to login into the study. Once the user has logined to study using their unique id they will download the study to their mobile device.

Integration Test	**Sample Input / Action**	**Expected Output / Effect**
Study Download; Participant Login	Create Study and participant; Have Participant login to study	Participant will be welcomed to study; Baseline Module will be given to Participant if one exists

Answers Upload and Download - Users will answer modules throughout the study and those answers will be uploaded to Firebase. Researchers will have access to the answers of any module in the study.

Integration Test	**Sample Input / Action**	**Expected Output / Effect**
Answer Module; Download Answers of Module	Participant answers module; Researcher downloads the module answers	Researcher will receive a CSV file of the answered module

Ownership/Display Permissions - The web portal includes ownership of each study. This is to prevent from anyone seeing seeing certain studies without the correct permissions. Studies, Modules, Questions, and Participants all have ownership permissions, which means only those with the correct permissions will be able to see these specific collections.

Integration Test	**Sample Input / Action**	**Expected Output / Effect**
Study permissions	Create two Researchers, one creates the study; Have the other look through their own studies	Researcher without the permission will not be allowed to view that specific study

Wearware Fitbit Query - The study that will use this application will be using Fitbit to gather more data from the users. The data received from fitbit will include data such as steps, heart rate, distance walked, etc. Wearware will be used to get the data from Fitbit, and the web portal will query Wearware for that data.

Integration Test	**Sample Input / Action**	**Expected Output / Effect**
Query Wearware data	Researcher will send query from web Portal to Wearware	Researcher will receive a .CSV file with fitbit data

The above integration tests will ensure our data is being grabbed and transferred correctly between modules and our database as well as make sure all interactions between our modules are correct. Section four, Usability Testing, will discuss our plan for ensuring a smooth end user experience.

Usability Testing

Our goal with usability testing is to ensure users of both of our applications, the web portal and mobile application, can effectively use and access the functionality we have provided. We need to make sure that our interfaces are understandable and actually perform how the end users desire them to perform. Before any serious testing begins we are planning some rapid iteration sessions with Dr.Dmitrieva where we present her everything we have in its extremely rough state without any user interface optimizations. Dr. Dmitrieva will then hone in on facets of our applications we can change relatively quickly to better suit her needs or provide end users with a more understandable experience. After the application has been smoothed out some from these sessions we can jump into usability testing for the web portal and mobile application.

WEB PORTAL USABILITY TESTING

Before usability testing can begin for our mobile application, we will start by testing our web portal. The target users for the web portal are researchers hosting studies, so both Dr. Dmitrieva and her students will be interacting with the portal for us to gather their feedback. Usability testing for the web portal will begin with moderating Dr. Dmitrieva and her students' use of the portal.

To moderate over the web portal usability testing, we will provide Dr. Dmitrieva and her students with a list of tasks asking them to perform various tasks from our integration testing, such as creating an account, creating a study, and creating questions. These tasks will be described somewhat vaguely to avoid over-instructing the researchers. Some task examples would include:

- Register an account on the web portal
- Create a study titled “My Research Study” along with questions, and modules
- Add the modules into the study and questions into the modules

By moderating over the researchers completing tasks, we can ask for their feedback on how they liked or disliked interacting with the interface. How they would prefer pages oriented, how buttons are mapped, and how it may compare to software they have previous experience with. We would also encourage the testers to “break” our code in edge cases we may not be aware of, such as certain data values that researchers wouldn't use or formatting rules that must be followed within scientific studies.

After moderated testing and continuous improvements to our product based on feedback we began letting testers use the products on their own in any way they see fit and report back to us any critiques or findings. Simultaneously we conducted mobile testing, detailed below.

MOBILE USABILITY TESTING

For unmoderated mobile testing, the user was on a “dummy” study. This will be a dry run of hosting users on a study created in our webportal. The users will also be given a task sheet with intentionally general/vague objectives to complete to interact with the app. For example, they will only be prompted “Visit the main screen and log your most recent meal” as opposed given step-by-step instructions on how to select, begin, complete, and submit a user initiated log.

The users will be invited to the dummy study and given a code. The user will download the app and be accepted into the study via their generated code. After the account setup, the only information users will have will be on their task sheet. During their testing period, they will act as if they are research participants

For in-person testing, we gave the testing participant a few small goals and oversaw how they interacted with our UI. Having each of these users go through the registration process is too time-consuming and we wished to focus on the app functionality, so we gave the user a phone to use with a test account attributed to it. We intended to moderate users by prompting them to do a task within the app and record their reactions and comments. Some questions that could give much-needed input from the end users may include the following:

- Is it always clear how the app wants you to complete a question? Why or why not?
- On a scale of one to five, how likely are you to select a push notification as soon as it arrives?
- Are the user initiated questionnaires easy to access?
- Would you be interested in seeing the results of previous questionnaires?
- Are any pages hard to navigate to?

Usability testing is a major part of discovering how functional and usable our product is, and as such, our testing methodology should reflect how in-depth and robust our product should be. Through our usability testing methods of both in-person moderated testing and hands-off user interaction over multiple days, we are confident both our mobile app and our web portal have the usability required to assist the researchers and the research participants with their needs.

6.0 Project Timeline

The projects timeline consisted of four main facets, research / tech feasibility, requirement acquisition and prototyping, core development, and testing. This section will provide some insights into each block.

Research and Tech Feasibility

Our team started out by making sure we knew exactly what our client wanted. We researched and had meetings with our client in order to achieve this goal. Once we had an idea on what to build we looked into the different frameworks we could use to build the application. Once we found what we want to use we began to look at what data we will be handling.

Requirements and Prototype

After finishing research, we looked into creating the requirements of our mobile and web application. After creating the requirements we began on implementing the prototype for the web application. During this period we were using Meteor JS to build the application and we were having difficulties with creating our applications.

Testing Framework and UI Implementation

The team continued with implementation in the beginning of second semester, changed the framework from Meteor JS to Ionic. We tested the framework to ensure that it would be able to meet the requirements for our application.

Mobile and Web Development

The team was divided into two groups:

Mobile Development	- Julian Shak and John Basler
Web Development	- Alfonso Martinez and Chantz Spears

This is where most of the time was spent during the second semester. The team would meet weekly so we can coordinate and work on tasks that will connect both applications together.

Testing
As detailed in section five, this is where the team conducted all forms of testing with the majority of the time focused on rapid iterative usability testing. The team conducted multiple usability test with the client and outside students during the months of April and May making constant changes, fixes, and improvements to SCHEMA all the way up to and including the final week of the semester to cram as much of the clients evolving requirements in as possible.

Section seven will briefly go over the plans for SCHEMA's future and what can be expected in upcoming development.

7.0 Future Work

The team was successful in creating an Ecological Momentary Assessment Tool that can expedite and lower cost for numerous researchers, while being less intrusive and easy to operate for study participants. Moving forward the client is looking to implement social functions to the application so participants have a greater sense of community and desire to complete diabetes prevention programs in particular. Beyond our current client, the application will start making its way into other researchers hands so they can begin utilizing it and requesting additions or modifications of their own.In the coming years the product is poised to have a widespread effect in the ecological research community as a whole. Once Fitbit data can be gathered wither by means of the WearWare portal or otherwise, the client will once again look into integrating Fitbit devices into her research. Fitbits could provide valuable data such as length of time sedentary and step count, so this will be a closely followed feature moving into the future.

8.0 Conclusion

As stated previously, approximately 10% of Americans have diabetes mellitus (32.3 million), with the highest rates observed among individuals who self-identify as American Indian and Alaska Native (15.1%). A large body of research has begun through examining effective approaches that may help reduce diabetes incidence in this population. Currently there is an effective 16-session program called Special Diabetes Program for Indians Diabetes Prevention (SDPI-DP). However, retention rates within these programs is problematic, with only 74% of participants remaining in the program by the 16th session, and only 33% remaining by the three-year follow-up.

To increase retention of among American Indian patients with prediabetes in SDPI-DP, the long-term goal of this work is to develop a mobile application that will support attendance and engagement in SDPI-DP. Our Capstone project will create a prediabetes intervention mobile application that will be used to research psychosocial experiences and health behaviors among American Indians who are enrolled in SDPI-DP. This application will collect momentary data on various psychological, social, dietary, and physical activity experiences, while making sure that the data management is HIPAA compliant. Our web portal will aid our client by allowing her to create, edit, and submit studies to research participant’s mobile application. Our product allows for customized EMA studies to be published cheaply, deployed quickly, and overall reducing the cost and workload of researchers.

This as-built report clearly details our project's requirements, both functional and non-functional, and forms the contractual basis for the expectations to be fulfilled by our Capstone team. This document helped our developmental team line up our expectations with our project sponsor as well as identify key future milestones that we will face in the months ahead. We are excited to finalize the key requirements for our project and continue any additional research to further prove the feasibility of our proposed implementation.

In conclusion, this Capstone experience for team Sugar Coded has required a large amount of work, but certainly a fruitful one. Us members can move forward from this experience with great confidence in our abilities in learning and understanding a framework, how to work and fulfill roles within a team, and to be productive in all steps needed to develop functional software as requested by a client.

9.0 Glossary

Firestore - Store and sync data between users and devices - at global scale - using a cloud-hosted, NoSQL database. Cloud Firestore gives you live synchronization and offline support along with efficient data queries.
Fitbit - Activity trackers, wireless-enabled wearable technology devices that measure data such as the number of steps walked, heart rate, quality of sleep, steps climbed, and more.
HIPAA - The Health Insurance Portability and Accountability Act of 1996 is United States legislation that provides data privacy and security provisions for safeguarding medical information.
Ionic - A complete open-source SDK for hybrid mobile app development. The original version was released in 2013 and built on top of AngularJS and Apache Cordova. The more recent releases, known as Ionic 3 or simply "Ionic", are built on Angular.
PHI - Protected health information (PHI) under US law is any information about health status, provision of health care, or payment for health care that is created or collected by a Covered Entity (or a Business Associate of a Covered Entity), and can be linked to a specific individual.
User Interface (UI) - The junction between a user and a computer program. An interface is a set of commands or menus through which a user communicates with a program.
Web2.0 - Websites that emphasize user-generated content, usability (ease of use, even by non-experts), and interoperability (this means that a website can work well with other products, systems, and devices) for end users.

10.0 Appendix A: Development Environment and Toolchain

This section will detail how to get your computer ready to continue development for the project. The applications were made using all of the different platforms (Windows, Mac, and Linux) so there is no constraint to the environment being used. Here are the steps that you will need to take in order to get ready.

*Note: Drop sudo if running the commands on Windows

Install Git

First, install Git on the computer by going to the website https://git-scm.com/downloads and download the necessary version needed for your environment. (The windows version will install Git Bash)

Install Node JS

The next thing to do is to install Node JS by going to their website https://nodejs.org/en/download/ and download the latest stable version needed for your environment.

Install Ionic Framework

In order to install Ionic you'll need to run a couple of commands.

1. Install Cordova

   Using the command line or Git Bash, run:

   ```
   $ sudo npm install -g cordova
   ```

2. Install Ionic

   Next using the command line or Git Bash, run:

   ```
   $ sudo npm install -g ionic
   ```

Now that we have Ionic installed, we will clone the repository and install the dependencies.

Clone Project

Go to the Github for the project: https://github.com/NauSugarCoded and choose which repository you are going to work on (The Web or Mobile Application).

You will click on:

and copy the link that is shown. Make sure that you are in the directory that you want the project to be stored. Type in the command without the quotations.

```
$ git clone “Copied Link goes here”
```

This will download a local copy of the project to the computer.

Install Dependencies

In the command line or Git Bash, make sure to navigate to the working directory of the repository.

Web application working directory is located in:

../capstone/WebPortal_prototype

Mobile application working directory is located in:

../Mobile-Application

Once in the working directory run the command:

```
$ sudo npm install
```

This will install all the features that the app uses, rather than having to install each one individually.

Run Project

In order to run the Ionic Application run the command

```
$ ionic serve
```

This will run the project locally on your machine to use for development and testing purposes. After this you are ready for hosting or creating a deployable .APK, both of these processes are documented in the install section of the user manual, however where you host the portal may change the process.