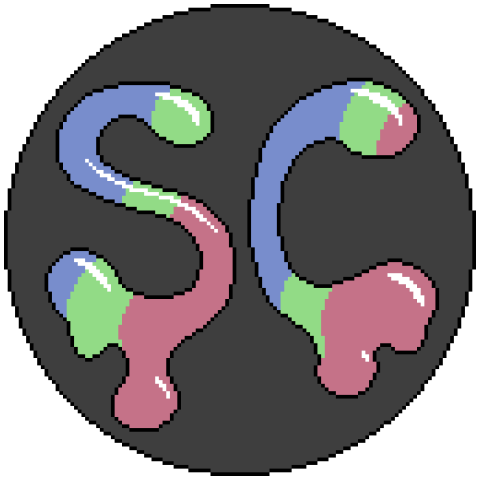
Sugar Coded



Technological Feasibility Report

November 7, 2017

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

1.0 Introduction

As of 2014, approximately 10% of Americans are diabetic, a disease defined by having a dangerously high blood sugar level. There exists two types of diabetes: Type I and Type II. Type I diabetics have a genetic disease that limits the amount of insulin they produce, making it difficult for the body to remove excess blood sugar. The vast majority of diabetics have Type II, often referred to as “adult-onset diabetes”. Type II diabetics have built a resistance to their body’s insulin, which creates a similar effect on the body as Type I. Diabetes is considered to be an irreversible condition, so diabetics are usually left to manage their condition for the remainder of their lives through constant blood sugar monitoring. There are organizations and programs that raise awareness about diabetes and others that try preventative measures to lower the rate of diabetes.

One such program is called the Diabetes Prevention Program (DPP). They attempt to prevent future cases of diabetes by finding high-risk patients and offer some moderate lifestyle and diet changes. The DPP’s goal is to reduce the patient’s body weight by 7% and the DPP claim some success in delaying and even preventing future diabetes diagnoses. The program includes patients of all ethnicities with about 45% of program participants are an ethnic minority. Those who stay on the program all have similar. However, there is a concerning anomaly among the Native American community in the program. Native Americans have the highest withdrawal rate from the DPP with approximately two-thirds prematurely dropping out of the program. Native Americans have the highest disposition for becoming diabetic and as such require more targeted support in order for the DPP

Our client Natasha Dmitrieva is an Assistant Psychology Professor here at NAU. She received her PhD at Pennsylvania State University for Human Development and Family Studies in 2011. Dr. Dmitrieva’s research focuses on identifying vulnerable subgroups that can benefit from targeted intervention. To discover reasons for those falling off the program, the current research method is to conduct phone surveys with the research participants. These would last for around a week and a survey participant is called every 60-90 minutes to fill out a paper survey over the phone.

In order to assist Dr. Dmitrieva research this disparity, Team Sugar Coded has been asked to develop a mobile application that can aid research on diabetes prevention program retention rates. The application would be considered an improvement to the current phone-survey method for collecting data from research participants. This project will also include a web portal where Dr. Dmitrieva can review the data on the study participants for her research. The mobile application will be used to identify how the patients are participating and provide information on their diet, exercise, and emotional patterns to give researchers a better way to provide retention support.

1.1 Document Purpose

The technology feasibility document will give a brief overview of the major design decisions we have to sort out in order to complete this project. This document will first introduce our Capstone project including a quick glance at the problem our application is trying to solve. Our Technological Challenges explains what important hurdles in our project we will have to overcome, such as database integration or app development environments. Following this section is the Technological Analysis which breaks down each challenge and we give our best solutions to each problem so they can work as a cohesive and wellbuilt project. After which is a brief overview of how we intend to integrate the components of our project together. The document comes to a conclusion with a summary of our findings and the technology we are choosing to move ahead with at this point.

2.0 Technological Challenges

The prediabetes research application is intended to be a mobile application that can collect momentary data on various psychological, social, dietary, and physical activity experiences; with a web application for viewing collected data and administering new inquiries. The project's goal is to provide information on targeted users for intervention support in a future study. To accomplish this, Sugar Coded will need to meet the following high level requirements and solve the technical hurdles that come along with them.

Baseline Questions: Administer a cross-sectional survey at a single point in time. Use participants answers from these surveys and consecutive time-based questions to tailor the application to uniquely fit each individual.
Time-Based Questions: Administer key questions about psychological/social context at both regular predetermined intervals as well as at specific times dependent on the user. The application should identify "interruptible" times based on each individual's waking hours, work schedule, etc.
Participant-Initiated Logs: Allow participants to log an eating event and all the characteristics of said eating event.
Sync with Fitbit: The app should be able to sync with fitbit data to collect exercise and sleep information.
Provide Admin Web Portal: The researcher can view real time data from the research participants and push new questions to the mobile app.

Having laid out our main high level requirements we can now discuss our technological hurdles to overcome.

Technological Hurdles:

- Database Integration - We need each application, the web portal and multitude of concurrently running mobile applications, to be able to access a single database. This database should be capable of storing large amounts of user input, including Fitbit data, demographic information, and event logs.
- Native Versus Hybrid Development Environment - The team ideally wants the mobile application to be available for both Android and IOS, which would require a hybrid application development process. If this is costly or unfeasible however the team needs to consider which native environment to go with.
- Frameworks - The application needs to provide a reactive interface for users to log events, as well as having an easy time with database integration.
- Data Visualization - The researcher needs to be able to easily view data from the database at any given time in a digestible manner. Our data visualization solution needs to be able to display data using multiple graphs and be easy to incorporate into our framework.

These hurdles will be further analysed and given a chosen solution in our following technological analyses section.

3.0 Technological Analysis

3.1 Database Integration

Introduction

The back-end database will be the core of our Capstone project, as it will be the central location of all of our project data such as:

- User information: demographic user data (name, age, contact information, etc.).
- Collection of survey questions: collection of baseline questions (first-time login surveys) and time-based questions (questions querying for emotion regulation, momentary stressors, etc.).
- Gathered research data: individual's responses to the questions described above.
- Storage for study meta data for studies being designed/deployed.

The desired characteristics we will be looking for in our back-end database include:

- Low cost: a low cost solution as our back-end database is desired for the amount of data that we will be utilizing.
- Capability to store large data sets: we will need to store large amounts of data, preferably with minimal performance degradation.
- User authentication: a built-in user authentication would allow us to quickly register and authenticate our application users.

- Ability to export to Excel: Dr. Dmitrieva would like the ability to export all of collected research data to put into an Excel spreadsheet.
- Ability to store images: a feature we may implement into our application is the ability to allow users to take pictures of meals, rather than manual input.

Upon preliminary research, we have compiled together three alternative options for our back-end database.

- MySQL is one of the most well known relational databases which is often used in projects that require storage of structured data.
- MongoDB and Firebase are two NoSQL databases that provide users with more flexibility in storing data, without restrictions on integrity constraints that you will face with MySQL.

In the subsequent section we will be going more in depth on each of these database solutions and comparing each approach with respect to our project needs.

Alternatives

1. MySQL

MySQL is a popular open-source relational database management system (RDBMS) that is developed, distributed and supported by Oracle Corporation. Like other relational systems, MySQL stores data in tables and uses structured query language (SQL) for database access. In MySQL, you pre-define your database schema based on your requirements and set up rules to govern the relationships between fields in your tables. In MySQL, related information may be stored in separate tables, but associated through the use of joins. In this way, data duplication is minimized. In the section below, we layout the pros and cons of MySQL with respect to our project.

Pros

- Cost: depending on what you plan to use it for, a MySQL implementation could range in price from free to $10,000 or more. Either way, it's significantly less expensive than most other database options on the market.
- Familiarity: all group members in the team have worked with the MySQL database.
- Complex transactions: a database transaction is one or more database statements that must be executed together, or not at all. A bank account transfer is a good example of a complex transaction.
- Tables: data is stored in separate tables, and tables can be related with foreign keys.
- Structured data: the relationship between tables and field types is called a **schema**. In a relational database, the schema must be clearly defined before any information can be added.

Cons

- Performance degradation: works better when you have a low write/read ratio, and offers low scalability as the read/write ratio grows.
- Not designed for qualitative data: designed to manage relational data. Though MySQL has implemented measures to better handle bodies of text and images, there are NoSQL solutions much better suited for this task, such as MongoDB.

- Dependent on add-ons: although MySQL is relatively easy to set up, it tends to have less out-of-the-box functionality than many other database systems on the market. Certain features – such as text search and ACID compliance – are dependant not on the core engine but on applications and add-ons.
- No built-in user authentication: MySQL does not provide built-in authentication, as some NoSQL databases do.

2. MongoDB

MongoDB is an open-source database that stores data in JSON-like documents that can vary in structure. Related information is stored together for fast query access through the MongoDB query language. MongoDB uses dynamic schemas, meaning that you can create records without first defining the structure, such as the fields or the types of their values. You can change the structure of records (which are called documents) simply by adding new fields or deleting existing ones. This data model give you the ability to represent hierarchical relationships, to store arrays, and other more complex structures easily. Documents in a collection need not have an identical set of fields and denormalization of data is common. MongoDB was also designed with high availability and scalability in mind, and includes out-of-the-box replication and auto-sharding. In the section below, we have the pros and cons of MongoDB with respect to our project.

Pros

- Cost: pay only for the capacity that is actually used
    - Free up to 512 MB of storage
    - Charged hourly: starting at $0.013/hr for 2 GB, $0.035/hr for 5 GB, etc.
- Qualitative data: storage of categorical measurements expressed in terms of natural language descriptions (unstructured).
- Ability to store large amounts of data: MongoDB's use of dynamic schemas allows dissimilar data sets to be stored together. NoSQL databases, focus on servicing highly concurrent requests while exhibiting low latency for responses operating on highly selective access criteria.
- Collections: A *collection* may store a number of documents and is analogous to a table of an RDBMS. A *collection* may store documents those who are not same in structure.
- Flexible: does not require a set formula (static schema) defining structure of data. So, it is possible to store documents of varying structures in a collection.
- Built-in roles: grants access to data and commands through role-based authorization and provides built-in roles that provide the different levels of access commonly needed in a database system.
- Integrated with meteor framework: each meteor application has a built-in integration with a MongoDB database.
- Built-in security: provides various features, such as authentication, access control, encryption, to secure your MongoDB deployments.

Cons

- Does not support transactions: you cannot perform multiple data manipulation operations atomically. If there's a failure or interruption, you will have to detect and clean up the mess yourself. This originates with the fact that Mongo was designed to be a document object-store.

- No data validation: does not validate data like MySQL does. This has to be done in the application using a library like Mongoose or Meteor's simple schema. This chews up the Node processes' CPU time - the most precious system resource in a Node application.
- Fairly new product: less up to date information available/fast evolving product.
- No JOINS of collections: less flexibility with querying.

3. Firebase

Firebase is a technology that allows you to make web applications with no server-side programming so that development turns out to be quicker and easier. With Firebase, we don't have to stress over-provisioning servers or building REST APIs with just a little bit of configuration; we can give Firebase a chance to take every necessary step: storing data, verifying users, and implementing access rules.

Firebase supports the web, iOS, OS X, and Android clients. Applications using Firebase can use and control data, without having to think about how data would be stored, and synchronized across various examples of the application in real time. There is no need to write server side code, or to deploy a complex server framework to get an app started with Firebase. In the section below, we layout the pros and cons of Firebase with respect to our project.

Pros

- Built-in authentication: provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app.
- Real time: NoSQL database that lets you store and sync data between your users in real time.
- Built-in security: provides a full set of tools for managing the security of your app. These tools make it easy to authenticate your users, enforce user permissions, and validate inputs.
- File storage backed by Google Cloud Storage: The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content. On the server, you can use Google Cloud Storage, to access the same files.
- Cloud hosted: Stored in the cloud so readily available everywhere.

Cons

- No JOINS of collections: less flexibility with querying.
- Does not support transactions: you cannot perform multiple data manipulation operations atomically. If there's a failure or interruption, you will have to detect and clean up the mess yourself. This originates with the fact that Mongo was designed to be a document object-store.
- No data validation: does not validate data like MySQL does. This has to be done in the application using a library like Mongoose or Meteor's simple schema. This chews up the Node processes' CPU time - the most precious system resource in a Node application.

Chosen Approach

In the table below, we have outlined the most prominent aspects of a back-end database, as it pertains to our project. As mentioned above, the desired characteristics we will be looking for in our back-end database include: low cost, capability to store large data sets, user authentication, ability to export to Excel, and ability to store images. We rate each database alternative on a scale of 1-5 in regards to each of these desired characteristics.

Scaling:

- 1 = database does not provide support for feature
- 3 = provides a work around/slight implementation of the feature
- 5 = feature is fully supported

Features	**MySQL**	**MongoDB**	**Firebase**
Built-in User Authentication	1	5	5
Image Storage	1	5	5
Social Media Authentication	1	1	5
Cost	4	4	4
Dynamic Schema (Flexible)	1	5	5
Large Data Sets	3	5	3
Can Export to Excel	4	5	1
Total	15	31	28

As you can see if the table above, MongoDB provides us with all of the desired characteristics we are searching for in our back-end database, other than social media authentication. However, Meteor which is the framework that uses a built-in integration of MongoDB, includes social media authentication with an account based plugin.

Based on our extensive research and analysis, we have decided to choose MongoDB as our back-end database because it is a very flexible relational database which provides a built-in user authentication and is also fully integrated with the Meteor framework. MongoDB's flexibility is a key feature that will enable our data storage to be highly configurable.

Proving Feasibility

To prove the feasibility of MongoDB as our back-end database, we will acquire a few test data sets from our client to prove that we can efficiently store and retrieve this data from our database. This demo will not be incorporated into a fully functional application, however it will be able to

show the communication between our front-end user interface and back-end MongoDB database.

3.2 Native vs Hybrid App Development

Introduction

In order to create an application for as many potential users as possible, our team's best option is to have the app available for both Android and iOS. However, these two platforms require their own development process, programming language, and development tools in order to run on their native devices. An important decision to make early in the app development process is whether to create the app natively or on a hybrid development platform. Native applications are considered to be an ideal approach. They allow for better in-app performance by taking advantage of the OS features and utilize the device's hardware. On the other hand, a hybrid app can roughly be described as a webpage on your phone. The programming is done in HTML5 and JavaScript, which allows the app to run on both Android and iOS. In the following two tables, we will demonstrate some major benefits and pitfalls to both Native and Hybrid apps.

Alternatives

1. Native App

An app developed to be used on a specific platform makes use of the specific platform's native features and hardware. Features would include picture in picture mode, custom keyboards, interaction with push notifications, and gestures to name a few. Hardware interaction would include integration of devices: GPS, accelerometer, camera, Native apps allow for the best app security by taking advantage of all security that comes with the hardware and OS of the platform native to the device. Hybrid apps generally only allow basic password protection, so they are often best used in applications where sensitive data protection isn't a necessity. Navigating through a natively developed app which makes use of the platform's buttons, menus, and formatting. This allows for a more familiar interface for phone users on that specific platform. What follows is a list of pros and cons in regards to developing a native app for our project.

Pros

- Best in-app performance: Since the application is developed for a specific platform it can be optimized to a greater degree.
- Best security: Since the app is relegated to a single platform security holes, bugs, and hacks are fewer and farther between
- Standardized user experience: Many users are used to seeing applications with, for example, Android Studio assets, and they have been proven effective by popularity.
- Easier feature integration: Plentiful amounts of documentation available for all kinds of features.

Cons

- Long development period: Development is very standardized, no real shortcuts.
- Expensive and time consuming to maintain
- Greater coding complexity: In depth knowledge of the specific native development environment is required to produce any type of robust application.

It seems quite clear there are very attractive benefits to develop a native app as seen in the pros above. However, with each benefit that Native apps offer, there is a great increase in the workload for the developer. Creating a native app for both Android and iOS requires the developer to have a knowledge base for developing apps in both environments. The time and effort required to complete such a project must be considered against the obvious benefits of having a native application in both environments. This topic leads into the opportunity to develop a hybrid application which will allow development over both platforms.

2. Hybrid App

Hybrid apps are web-based applications that run on both Android and iOS. They are written in web programming languages such as JavaScript and HTML5. These web languages offer all the potential for a great UI that are available on any website while also requiring fewer lines of code and less overall maintenance than Native apps. Because the project can be developed in one environment and requires only one programming language, the development period is much shorter. The following table displays the advantages and disadvantages of a hybrid application. The following is a breakdown of the pros and cons of a hybrid app in relation to our project.

Pros

- Cheaper development cost: No need to buy, for example, Xcode just to create an IOS application.
- Code portability to other platforms and web apps: Projects can more easily be transferred to different environments / frameworks opposed to native apps.
- Fewer overall code length
- Easier to maintain: More general language knowledge rather than specific development environment knowledge.

Cons

- Slower performance: Since the apps are not optimized for a single platform there is a decrease in performance speed.
- Less security: With a more open development structure and outside libraries being used comes the risk of security holes.
- Often requires online connectivity: Many of these apps require constant connection to the server they are being hosted on.

Hybrid applications have the advantage of code portability, they can work on both IOS and Android platforms most of the time. Since developing a hybrid application is much like developing a web application the code base is easier to maintain and less complex in the long run.

Chosen Approach

In the table below, we compare hybrid and native applications in terms of development metrics, performance, UI, and security. This table rates hybrid and native applications on a scale of 1 to 5, 5 being the highest score.

Scaling:

- 1 = app type does not provide support for feature / extremely weak in this category
- 3 = provides a decent implementation of feature
- 5 = feature is fully supported / strongly represented

Features	**Native App**	**Hybrid App**
Time to develop	2	5
Ease to Develop	2	4
Performance	5	3
UI	4	4
Security	5	2
Development Costs	2	4
Total	20	22

Our chosen approach is to create a hybrid application. Having to develop a single web-application on a phone that will work for both Android and iOS is the most attractive advantage. Developing two complete and separate apps with two different programming languages and for two different platforms for a minor to moderate performance boost and user familiarity with an app is less feasible to us than developing on one platform for both. As described in the table above, the recent additions to JavaScript make it possible for hybrid apps to access phone features, which a few years prior would be a major reason to forgo hybrid in favor of native app development.

As our main choice for a hybrid app development platform is Meteor, (which has MongoDB integrated) our choice to use MongoDB for database management will be a great aid for developing our app's database. By creating a hybrid app, our app will reach a larger audience more quickly without much of a sacrifice to its performance or the app's overall utility.

Proving Feasibility

To demonstrate the feasibility of a hybrid application for a project of this scale, we will begin by showing the app's ability to send basic push notifications on both Android and iOS devices to collect data on key behaviors such as eating, exercise, or mood. The app must also be able to register accounts for the study participants. These accounts must be able to transfer basic information from the device to the web-portal for the administrator to collect the data for their research.

3.3 Frameworks

Introduction

In order to create this mobile application we have to chose what framework we want to use to structure our mobile application. Frameworks are used to efficiently create applications by giving the developer resources for implementation. They help alleviate the bulk in creating modern day applications with starting templates and easy to implement features. There are many frameworks to choose from so we narrowed our research down to three frameworks that have some of the specific functionalities that we need. These functionalities are:

- Cross-Platform Integration - Is able to be used to develop both Android and iOS application
- Database Integration - Has database management built-in within Framework

The following section is an analyzation of three frameworks, recommended by multiple peers and forums, to see which can fulfill our needs best.

Alternatives

1. Android Studio

Android studio is Android's official IDE for creating Android apps. Android Studio has integration for different modules (Android app modules, Library modules, and Google App Engine modules). It has built-in integration with Google's Firebase and Cloud.

Android Studio uses a Gradle Build system. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process - This means we are able to change our application by changing certain configurations within the build.
- Create multiple APKs for your app, with different features using the same project and modules - We can create multiple versions of our application and add/remove features for testing purposes.

Here we can see the advantages and disadvantages for using Android Studio as our framework:

Pros

- Optimized for all Android devices: We are able to decide what what version of Android will support the Application we are creating
- Android Studio has Google's Firebase and Cloud Integration: A very important aspect for our application to have
- In-line Debugging: Has a built in debugging tool for quick fixing
- All members have experience with Android Studio
- Can run application within the IDE using an Emulator - Can easily test application from within computer

Cons

- UI is very bulky (Very difficult to use)
- App will only be developed for Android Devices
    - Would have to rebuild application in order to have an iOS version
    - IOS applications are usually made using Swift
    - Requires a Mac to develop on Swift

As we can see, Android Studio's biggest disadvantage is its ability to only develop Android applications, whereas our other frameworks are multi-platform capable.

2. Oracle Mobile Application Framework (MAF)

Oracle Mobile Application Framework (Oracle MAF) is a hybrid mobile framework that allows users develop single-source applications and deploy to Apple's iOS, Google's Android, and Microsoft Windows 10 platforms. Oracle MAF leverages Java, HTML5 and JavaScript to deliver a complete MVC framework with declarative user interface definition, device features integration and built-in security. This framework does not have database integration but it does give us the cross-platform functionality that we need to port this application to Android and iOS.

Here are some of the advantages and disadvantages of this framework:

Pros

- Cross-platform: Can easily build application for multiple platforms such as Android, iOS, and Windows
- Choose what language to develop: Oracle MAF gives us the option to develop the application, using either Java or Javascript
- Uses standard technologies (HTML5, Java, JavaScript, CSS)
    - Platforms that the team knows and are comfortable with

Cons

- No Concrete Database Integration
    - Needs extra development in order to incorporate a database within the application
- Team does not have experience with Framework

Overall this framework would be a prime candidate for developing this application, but since it does not have built-in database integration it will cause extra unnecessary work.

3. Meteor

Meteor is a full-stack JavaScript platform for developing modern web and mobile applications. Meteor includes a key set of technologies for building connected-client reactive applications, a build tool, and a curated set of packages from the Node.js and general JavaScript community. Meteor is able to update the application in real time with sever data (Web application information/adjustments) that is sent to the client. Meteor also has MongoDB integration, which is one of our wanted features for the framework.

Meteor integrates with Cordova, a well-known Apache open source project, to build mobile apps from the same codebase you use to create regular web apps. With the Cordova integration in Meteor, you can take your existing app and run it on an iOS or Android device with a few simple commands.

Here are some of the advantages and disadvantages of using this framework:

Pros

- Meteor allows you to develop in one language for all environments: application server, web browser, and mobile device.
- Cross-platform: Can easily build application for multiple platforms such as Android, iOS, and Web
- MongoDB integration: It uses the Distributed Data Protocol and a publish–subscribe pattern to automatically propagate data changes to clients without requiring the developer to write any synchronization code.
- Meteor uses data on the wire
    - This means the server sends data, not HTML, and the client renders it.
- Meteor provides full stack reactivity
    - Allowing your UI to reflect the true state of the world with minimal development effort.

Cons

- Smaller Community compared to other frameworks
- Team does not have full experience with framework

Chosen Approach

In the following table desired features for our framework our outlined, and the three frameworks are scored on a 1 - 5 scale in regards to how well they implement that feature.

Scaling:

- 1 = framework does not provide support for feature
- 3 = provides a work around/slight implementation of the feature
- 5 = feature is fully supported

Features	**Android Studio**	**Oracle MAF**	**Meteor**
Database Integration	4	1	5
Supports both platforms	1	4	4
Easy Debugging	4	3	5
Web App compatible	2	5	5
Realtime Rendering	4	3	4
Total	15	16	23

We decided to go with Meteor to create our application. It allows us to use the same source code for developing the application for either Android or iOS. If we use Android Studio we will only be able to develop the application for Android devices. This means that if we want any iOS functionality we would need to rebuild our application from scratch. Oracle MAF has no concrete database integration so it will add more overhead with creating the database for the application. With Meteor we will not have to worry about these issues. The following section discusses our plans for proving Meteor's feasibility in our project.

Proving Feasibility

In order to demonstrate feasibility, we will create a prototype to test the data transferring functionality between both the mobile application and the web application. We will also have this prototype implemented into both Android and iOS device to check that data transferring can be done between both platforms.

3.4 Data Visualization

Introduction

While our client, Dr. Dmitrieva, has a degree in statistics and is more than capable of easily reading through and drawing conclusions from raw data, our mentor advised us that other researchers may not be as statistically fluent. Our desired characteristics for any method of data visualization include:

- Ease of integration - How easy can we incorporate it into our chosen framework and database, will there need to be significant consideration in our code to allow the data visualization option to be feasible.
- Easy for researchers to draw conclusions from - Intuitive displays, no over complicated or crowded graphical displays. The visuals need to be simple and informative.
- Free - Ideally we do not want to pay for any software, free and open source is our desired route.
- Capable of portraying multiple graph types - Depending on the research being conducted different visuals will be needed to succinctly convey gathered data.

The following section goes through three possible choices for data visualization, each with unique advantages and disadvantages. These options were selected after we saw them being mentioned repeatedly in top comments on forums such as Stack Exchange and Blogspot.

Alternatives

1. Excel

Most researchers are familiar with Microsoft Excel as it has been the primary Electronic Spreadsheet Program used in research and business for decades now. Great for storing, organizing and manipulating data for businesses, and academia, easy to no setup, and many databases can export straight to Excel or .csv files. The following pros/cons list highlights Excel's strengths and weaknesses.

Pros

- Easiest to integrate: can export data from a database straight to an excel document.
- Familiarity: Natasha, and the majority of researchers, are used to seeing, and manipulating data within Excel.
- Compatibility: Excel is compatible with old technology, new technology, large data sets, and small data sets, more so than many of its competitors.
- Free / cheap (.csv files can be opened with google sheets if necessary)

Cons

- Drawing Conclusions: Conclusions from data will most likely not be gleaned as quickly as other, more visual data visualization methods.
- Manual: Excel can not produce graphs or visuals inside our application without someone manually creating the graphs.
- Visuals: Excels graphs are not as interactive or visually stimulating as many other options in the space.

While Excel is simple and easy to use, it may not be robust enough for our needs. Graphs can be made in Excel but they often are not visually stimulating and require some manual tweaks to come out correctly. The largest weakness Excel exhibits is the inability to display any type of visuals in our application without manual tinkering, for this reason we find excel to be a weak contender.

2. D3 - Data Driven Documents

D3.js is a JavaScript library for manipulating documents based on data. D3 helps bring data to life using HTML, SVG, and CSS, all of which are pretty common to our team. D3's emphasis on web standards gives us the full capabilities of modern browsers without tying us to a proprietary framework, combining powerful visualization components and a data-driven approach to data object model manipulation. In short, no matter what we decide to use as our framework / database d3 should be compatible for displaying quick and insightful data visualizations. This pros/cons list will further highlight any advantages or disadvantages D3 may hold.

Pros

- Team familiarity: D3 is a JavaScript library that uses HTML, SVG, and CSS and the team is already fairly familiar with these languages, so no extensive learning should be required.
- Diversity: Displays graphs of all types, shapes and sizes so researchers can quickly glean any information needed.
- Flexible: Avoids proprietary representations, data representations are extremely flexible.
- Free

Cons

- Time consuming: Will assuredly take longer to incorporate smoothly into the admin portal rather than just excel downloads, or a simpler visualization option.
- Researcher preference: Users may want different graph types than what D3 offers, or they may just want to make any graphical representations themselves for fine tuning purposes.

D3 is looking to be a solid contender for our data visualization needs, it seems to easily, albeit time consuming, integrate into frameworks like Meteor and offers a robust array of graphical options.

3. Metabase

An open source business intelligence tool that allows easy sharing of data insights across a company or group. Allows you to 'ask questions' about your data and displays answers in a convenient but technical way. Metabase is a fairly new option in this space and seems to cater towards larger businesses rather than small projects. The following illustrates the pros and cons discovered with Metabase in relation to our project.

Pros

- Shareable: Can run on AWS to let others log into accounts, share reports, etc. fuss free.
- Provides SQL interface if we desired a more conventional method of grabbing the data
- Customizable: Visualizations can be customized in very few clicks.
- Free and opensource

Cons

- Restricted: Fewer graph types, and the visualizations that can be made with the graphs are limited

- Difficult to implement: Little to no documentation on including these visualizations inside of an application, getting the dashboards into our app would require a lot of time and trial and error.
- Pulling data: Acquiring the data from a database and turning that into a graph or data visualization is much more difficult than competitors due to lengthier more obfuscated commands.
- Specialized for metadata: Originally designed for metadata much of the functionality is contingent on this, and while we may use some metadata we need data visualization that can work for all types of data.

Since Metadatas graphs are rather restricted and there is not a whole lot of application implementation documentation due to its recent appearance, it would not be in our best interest to move forward with Metabase.

Chosen Approach

In the table below, we have outlined desired features we were looking for in data visualization software and compared the three aforementioned options. The table follows the 1 - 5 rating conventions, with 5 being the highest score.

Scaling:

- 1 = the data visualization does not provide support for feature
- 3 = provides a work around/slight implementation of the feature
- 5 = feature is fully supported

Features	**Excel**	**D3**	**Metabase**
Easy to Integrate	5	4	2
Different Graph Types	3	5	4
Free	4	5	4
Visually Appealing	1	5	3
Works With Our Data Types	5	5	2
Total	18	24	15

Having reviewed the above information we decided to choose D3 - Data Driven Documents, as many forums discussed how they had integrated it with meteor and MongoDB with great success and it offered the cleanest and least confusing way to visualize data when compared to competitors. We will still offer the capability for a researcher to export their data straight to a .csv, so if they choose to draw their own conclusions in a program like Excel they are free to do so. The following section discusses how we plan on proving feasibility for D3 in our project.

Proving Feasibility

To prove the feasibility of D3 as data visualization library, we will acquire a few test data sets from our client and store them into our database, then create a dew sample graphs from the data. These graphs will display what we will be capable of showing in our end product.

Now that we have decided which technologies to move forward with in our project we need to take a moment and discuss how we plan to integrate these many moving parts together for a complete product.

4.0 Technological Integration

Our project will consist of two or possibly three (explained below) applications, with one central back-end database to store all data pertaining to our project. As we move forward into the initial development of our project we will face two main integration challenges. The first integration challenge that we will face is the configuration of all separate applications to the same back-end database. The second integration challenge will involve the integration of Fitbit user data, which is proprietary. We will be going more in depth on these integration challenges and our proposed solutions in the sections below.

System Diagram

The diagram on the following page is a visualization of our project's overall structure and the integration of all major components.

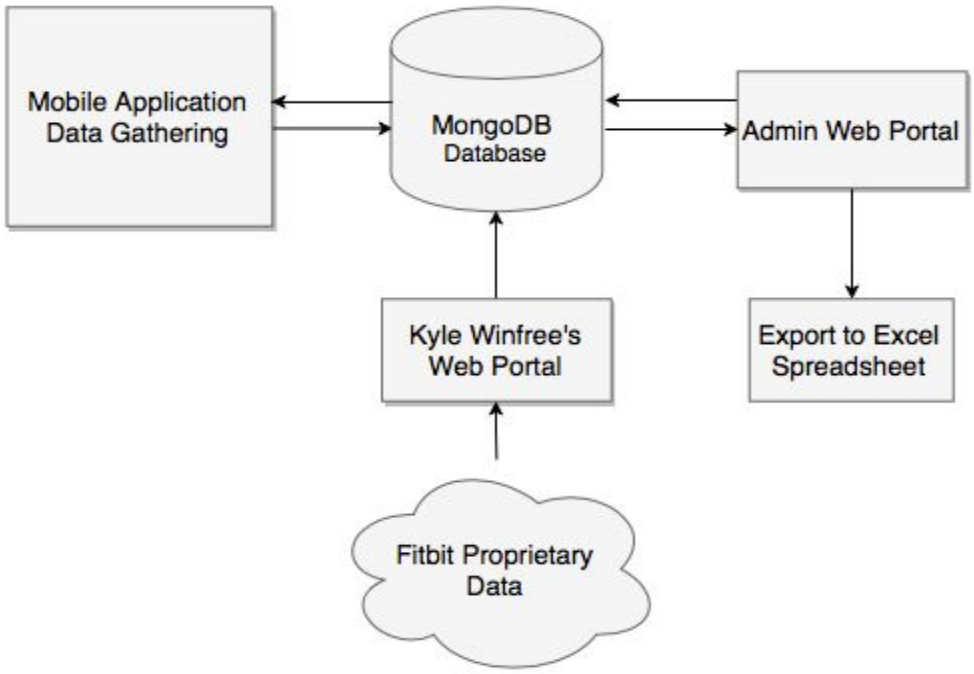


Figure 1: System Diagram

Integration Issue 1 : Central Database for Mobile and Web Application

The first problem we will face during implementation is the configuration of one central MongoDB database for both our mobile and web application. Fortunately, Meteor which is the web and mobile framework that we are using, includes built-in integration with MongoDB. The main challenge that comes with this central database solution is ensuring that both our mobile and web application are storing and retrieving data from the same MongoDB database. After initial research, the solution to this integration challenge seems pretty trivial, as long as the applications use the same database (as identified by the MONGO_URL), they will respond reactively to changes in the data.

Integration Issue 2: Integration of Proprietary Fitbit Data

As displayed in Figure 1 above, not only will our mobile and web application be communicating with the MongoDB database, but we will also be looking into retrieving individual user's Fitbit data. The issue with this is that all of Fitbit's data is proprietary, which means we can not obtain this data without the consent of Fitbit along with a fee. This is the second integration challenge that we will face, however we will be collaborating with a SICCS Assistant Professor, Kyle Winfree who has done extensive research with Fitbit integration. We will be collaborating with him on his portal that has Fitbit integration access to possibly connect Fitbit data to our project.

5.0 Conclusion

As stated previously, approximately 10% of Americans are diabetic (32.3 million). The question is "What do we do to decrease this number?" One solution is to provide support for specific groups that have a huge amount of high risks for diabetes. Currently there is a program that is called "Special Diabetes Program for Indians Diabetes Prevention". The issue with this is that there a 67% withdrawal rate within the American Indian community. We are building this mobile application to research the causes of these withdrawal rates. With our application, researchers will be able to better study and understand why people quit DPP.

Project Summary

Overall, our project is to create a prediabetes intervention mobile application that will be used to research behaviors among American Indians with prediabetes. Our goal for this project is to create an application that will collect momentary data on various psychological, social, dietary, and physical activity experiences, while making sure that the data management is HIPAA compliant. Below is a summary table of our confidence levels in our chosen technologies.

Tech Challenges	Proposed Solutions	Confidence Level
Storing Data	MongoDB	High
Hybrid vs Native Application	Hybrid	High
Web-Application Framework	Meteor	Very High
Mobile-Application Framework	Meteor	Very High
Data Visualization	D3	High

We are confident in our proposed solutions for our tech challenges. MongoDB meets our criteria and is built-in within our selected framework. We are going with a hybrid application over a native application so we can have our application be deployed for both Android and IOS devices. Meteor is the winner when it comes to our desired features for our needed framework, since it has MongoDB integration and is able to be used for both our mobile and web application. D3 is able to build graphs and data charts with the pulled information from our database to make information much more readable.

With these selected technologies we will be able to implement a mobile application that will give researchers a data collection tool that will simplify their current methods of gathering information. Through constant contact and communication with our client we can ensure we are making satisfactory progress and if all the basic necessities are met in a timely manner, we may be able to add any additional features that our client wants incorporated into the application.