Team DataBit

Technological Feasibility Analysis

October 20, 2020

Team Members: Andrea Caviglia, Cheyenne Clutter, Samantha Rodriguez, Jensen Roe, Steven Sprouls

Sponsor: Dr. Kyle N. Winfree

Mentor: Dr. Eck Doerry

Table of Contents

1 Introduction

Cardiovascular diseases are the leading cause of death globally. In the United States alone, nearly half of adults over 20 years of age have some form of cardiovascular disease, and approximately 655,000 people die from heart disease each year – around one in every four deaths nationally. Moreover, they pose some significant concerns in light of the coronavirus pandemic, as those who suffer from cardiovascular disease are at an increased risk of severe illness and death from COVID-19. Lifestyle changes, such as engaging in physical activity and getting enough sleep, are often the best way to prevent and treat cardiovascular diseases, so being able to monitor such behaviors in some way is an important part of addressing the problem at large.

More research is needed to better understand the relationship between lifestyle changes and their impact on disease risk/outcomes. To that end, researchers require a way to collect information such as movement data from large numbers of patients. One approach is to employ wearable activity tracking monitors that can analyze physical activity and sleep behaviors throughout the day. Researchers can then conduct fitness and sleep studies and, in analyzing data collected from such devices, may aid their communities by taking action with the knowledge they extract- publishing findings, attempting their own interventions, or helping to conceive of new innovative solutions.

Often, devices called actigraphs are used in clinical settings to observe physical activity. These are small wearable devices that go on a person's wrist, much like a watch, that monitor rest and activity cycles. However, they are expensive, difficult to use, and incapable of storing much data over long periods of time, which makes them unwieldy for the purposes of long-term study with a large number of participants. Our project sponsor, Dr. Kyle Winfree, is a researcher who is interested in the utility and accessibility of Fitbits, and the potential they pose in offering "smart" feedback. As the leader of the Wearable Informatics Lab at Northern Arizona University – whose primary interest is in utilizing wearable technologies to measure and improve health care – one of his current concerns is the collection of wearable data from users. While Fitbit devices themselves provide a relatively inexpensive solution to data collection, they raise the problem of how to automatically collect data from the hundreds of participants in a typical study. [redacted]. These issues have prompted Dr. Winfree to devise the WearWare Project – a platform for evaluating

wearable fitness tracking. However, previous versions of the project's database have suffered from poor performance and an inability to handle the sheer amount of data.

Our envisioned solution is to develop a new system from the ground up that will form the backend of the WearWare concept. This will involve creating a powerful and flexible data collection, management, and delivery system that allows for the registration of studies, users, and Fitbit devices, and communicates with the Fitbit API in order to download data in real-time. We will also be providing an API for future modules of the project in order to perform various operations on the data, as well as a basic development client that connects to this API for testing. In doing so, we will be providing Dr. Winfree and his lab – as well as potentially other researchers in the future – with an innovative and affordable product for use in various health studies.

This document aims to ensure that the tool frameworks and other technical foundations of this project will be appropriate and functional in the final product. We begin by identifying the key technological challenges we expect to face, then analyze each of the major issues as well as potential alternatives, providing rationale for deciding on a solution. In the final section, we will introduce our envisioned integration of technologies and devise a preliminary sketch of the overall architecture for the WearWare system.

2 Technological Challenges

Overall, our project is heavily focused on designing a database for the WearWare system, but we will also need a front end as well as other components to support data collection. Below we have outlined the major technological elements that we anticipate needing in our project, which will ultimately serve as high level requirements.

We will need:

- A reliable database management system that is capable of secure, highly accessible storage of millions of data points.
- A reliable and secure database hosting service that can support the database management system.
- A versatile application framework that allows for relative ease of development for an administrative GUI front end and protection from a wide variety of security threats.
- A way to provide development and testing activities as well as numerical analysis.
- A way to retrieve data from participants' Fitbits on a fixed timeframe.

The core technologies of our system must address these issues in an effective manner such that they are successful both individually as well as collaboratively. In the next section, we will first analyze each challenge and propose solutions as they pertain directly to that problem.

3 Technology Analysis

On the basis of the aforementioned technological challenges, we have identified five major design decisions as critical to this project's success: the database management system, database hosting service, application framework, development client, and Fitbit API. In each of the following subsections, we will discuss the issues, introduce key characteristics of an ideal solution, and analyze various alternatives before coming to a preliminary decision on a promising technology.

3.1 Database Management System

The most central issue to our project is the database management system. It serves as the backbone of this WearWare module; without functional data management, none of the other components of our system will be effective or useful. Therefore, choosing an appropriate database system is vital to our success in this project.

- Given the volume of data that needs to be stored, it is imperative that we select a robust technology that will allow for optimization of speed and performance.
- This necessitates a need for scalability, particularly with the vision that researchers aside from our client may one day use our product.
- Because we will be storing some sensitive information of research participants, it is important that our database system is secure.
- Another major deciding factor is cost; if licensing prices are too high, it undermines the entire purpose of our project as an affordable system for conducting health studies.
- Our final metric of evaluation is ease of use, as a technology that is easier to learn and grow accustomed to is preferable over one that is difficult to understand and troubleshoot.

It is on the basis of these criteria that we will analyze and discuss five options: MySQL, MariaDB, PostgreSQL, Oracle, and MongoDB. To investigate the strengths and weaknesses of each database management system, we downloaded each and tested them using a small dataset. We also read through documentation, looked at developer blog posts, and

examined performance benchmarks in order to get a comprehensive understanding of how each technology differed.

3.1.1 MySQL

As perhaps the most popular database system, MySQL was naturally the first technology we investigated. It is currently owned by Oracle Corporation and is an open-source relational database management system (RDBMS) that is known for its performance, reliability, and ease-of-use. It has many built-in security features, including secure connections, authentication services, authorization and privilege management, as well as data encryption. However, MySQL was not built with scalability in mind, so although it is strictly possible to vertically scale a database, MySQL is generally not recommended if the database is expected to grow substantially. In the case of our project, this is almost guaranteed. Furthermore, because MySQL is dual-licensed, it can either be used under GNU General Public License or a standard commercial license purchased from Oracle; as such, some features and plugins such as thread pools are only available for proprietary editions. Given that the MySQL Enterprise Edition costs $5,000 annually, we would almost certainly default to the standard edition. We assume that none of the features exclusive to the commercial version would be strictly essential, but having them kept unavailable to us would likely limit our ability to optimize the database and query performance.

3.1.2 MariaDB

MariaDB is a fork of MySQL that was built upon the values of performance, stability, and openness. It is also the database system that was used for earlier versions of the WearWare project, which is what prompted us to consider whether it would continue to be viable or if it was, to begin with, insufficient for this system. MariaDB is an RDBMS that is similar to MySQL in most respects, but boasts better speed and scalability compared to its predecessor. It is also entirely open-source and available under the GNU General Public License. That having been said, it is lacking in some optimization features such as partial indexes and query parallelism, which somewhat detracts from the overall potential of the technology.

3.1.3 PostgreSQL

PostgreSQL is another popular open-source RDBMS and is regarded as the second most used database technology after MySQL. It is freely available, and was created with the goal of being highly extensible and standards compliant. PostgreSQL supports JSON and works well with various analysis programs (e.g. Matlab, R) and modern frameworks (Django, Ruby on Rails). Moreover, it has a reputation for high performance – generally considered faster than even MariaDB at times – as well as high scalability. In regard to security, it also

offers features such as data encryption and authentication management that make it comparable to MySQL and MariaDB. Though it does have less documentation and a slightly higher learning curve, overall PostgreSQL is still considered relatively easy to use.

3.1.4 Oracle

The Oracle database is a multi-model database management system that is perhaps generally considered the leading RDBMS. It is a proprietary system that is often used by global enterprises, only rivalled by Microsoft's SQL Server. It is highly trusted in regard to speed, scalability, and security – even more so than all three of the aforementioned technologies. However, Oracle suffers from two major downfalls: price and ease of use. The licensing costs are upwards of $5,000 per month, effectively rendering it unserviceable as far as this project is concerned. Even were that not the case, though, it is both difficult to install and maintain, and the high learning curve in itself makes Oracle an unwieldy technology for WearWare.

3.1.5 MongoDB

In the interest of thoroughness, we decided to analyze a non-relational database. MongoDB is a NoSQL document-oriented database that is used for high volume data storage. It was largely created with the intention of being intuitive for developers; rather than employing large, complicated table structures, in MongoDB, data can often be stored as a single document. As the most popular NoSQL database, it is regarded for its great speed and scalability and an overall ease of use. However, while MongoDB does have various security features expected of a modern database, unlike with SQL databases, the configuration files of MongoDB are not encrypted automatically, nor is authentication enabled by default. Likewise, although NoSQL databases are often lauded for displaying greater speed and performance compared to their SQL counterparts, some tests have shown that PostgreSQL occasionally achieves better results than MongoDB. In any case, with regard to WearWare, it is appropriate to utilize a relational schema on the data being managed, so even assuming that MongoDB was strictly faster it would not necessitate the use of a NoSQL database.

3.1.6 Chosen Approach

In general, all of the technologies we explored were strong database systems in their own right. However, in our analysis, MySQL sacrificed speed and scalability for low cost and ease of use, while Oracle was a powerful RDBMS that was simply too expensive and difficult to learn. MongoDB, similarly, suffers for its poor security implementation and simply on account of being non-relational. MariaDB and PostgreSQL became our top contenders, as they displayed a fairly even distribution of desirable qualities.

	Speed	Scalability	Security	Cost	Ease of Use	Total
MySQL	2	2	4	4	5	**16**
MariaDB	3	4	4	5	5	**21**
PostgreSQL	4	4	4	5	4	**21**
Oracle	5	5	5	1	2	**13**
MongoDB	4	5	2	3	4	**18**

Figure 1: Criteria evaluation for database management systems. Each metric is scored on a scale of 1-5, where 5 represents the ideal. The total score is out of 25 points maximum.

As shown in Figure 1 above, MariaDB and PostgreSQL achieved the same overall score. They are both freely available, and exhibit high scalability and security. Comparatively speaking, PostgreSQL trades off a slight disadvantage in ease of use for an improvement in speed. Although we did not assign formal weights to our criteria, we have determined that speed is of the utmost importance; therefore, we are opting to use PostgreSQL for our database management system.

3.2 Database Hosting Service

The database hosting service is the technology that provides and manages the database environment. As previously mentioned, scalability is an important aspect of the database system. The current software is being held on a Mac Mini, and is unfit to handle the scale needed by our client. In order to overcome this shortcoming, we are looking into using virtual servers which will allow us to host the database system on a larger scale. To that end:

- The most important challenge is deciding on a service which is highly scalable and can adapt to future growth of the system.
- We expect studies to last anywhere between six months to five years, and therefore it is critical for the host to be reliable in storing and preserving data.
- The service must be within our client's budget. Otherwise, the project could become unreasonably expensive, costing hundreds of dollars per month.
- Considering that the data itself contains personal information about participants, we must also evaluate the security of the hosting services.

- The host must connect to our database management system, and therefore compatibility is another aspect to consider. Because our proposed database system is PostgreSQL, support for that technology is imperative. It is possible that a change in database system will one day be warranted, though, so compatibility with various technologies is preferable.

Based on these criteria, we have decided to evaluate Amazon Web Services (AWS), VMware, and DigitalOcean. To investigate each, we researched the three host systems by looking at documentation and product descriptions given by their providers. We also downloaded free trials and tested them with a trivial database by performing simple queries.

3.2.1 Amazon Web Services (AWS)

AWS is a cloud computing service provided by Amazon that provides some of the most popular products for database hosting. It was developed by Amazon as an on-demand cloud computing software. As AWS is currently in use by our client, we were compelled to explore its viability. With a large clientele, AWS has nearly perfected the reliability of their system. They have servers that can frequently back up data at specified times in case of lock ups. Some services even include an extra layer of backup for database systems that need to keep a constant live collection. Meanwhile, cost and scalability are directly correlated. If some months there is more data AWS can scale to that with an increase in price, but the increase in price won't stay if the storage is not being used. This is a big advantage to using AWS, as our monthly costs should only vary between $20 to $40 in total. As the data for this project will be collected second by second, we require not only a reliable connection but a highly secure one. Though some security features are limited to the more expensive services, those within our project's budget still provide basic security encryption.

Because of their large clientele, AWS is also extremely compatible and supports an abundant array of database systems, including all of the options analyzed in the previous section, with good support for other systems and languages as well.

3.2.2 VMware

VMware is a public software company that provides cloud computing and virtual software. VMware has been used for other cloud based services as a virtual machine. Unlike the AWS cost and scalability model, however, VMware has either a one year or three year licensing purchase. While it can involve an average monthly payment as AWS does, this model is not scalable; you simply keep the amount of space that you agree to pay for. In regard to security, VMware has similar specs to AWS, but while VMware does support backups, they

have less options for this than AWS. Uniquely, it can be integrated with AWS as a hybrid cloud service. This would likely add an extra layer of complication that is unwieldy for this version of the project, but could be useful for future iterations. Similarly, for future iterations, our client wanted the ability to pull the data off the server, which is something that VMware can accomplish, though perhaps not to any major current benefit.

3.2.3 DigitalOcean

Digital Ocean is a cloud framework with data centers which provides services to run applications on multiple computers. DigitalOcean prides themselves on their reliability. The system backs up any data on a day to day basis. Unlike with AWS, this comes at no extra charge to the client. DigitalOcean also automates failovers for a trusted server in case of failure to the data. DigitalOcean runs each server on their own private network, which helps with security. The system also adds on to that by encrypting the data when it is in transit and rest, like both AWS and VMware. While DigitalOcean does scale it doesn't handle the reduction of that growth quite the same. A manual operation would have to be done to account for that. Basic droplets range anywhere from $5 to $80 per month, but they may not have enough memory for the WearWare system's needs. Another downside to this system is its compatibility. Though it is compatible with PostgreSQL, it does not support MariaDB, Oracle, or MongoDB. In the event that we need to transfer databases, this could be a potential detriment.

3.2.4 Chosen Approach

While all of the technologies have their uses, our primary concern of scalability informed our decision for this project. Though VMware had good cost and security, and DigitalOcean was very reliable, both of them suffered for their scalability. AWS, on the other hand, not only showcased the best scalability, but had the highest overall score, as shown below.

	Scalability	Reliability	Cost	Security	Compatibility	Total
AWS	5	5	4	4	5	**23**
VMware	2	3	4	4	4	**17**
DigitalOcean	3	5	5	4	3	**20**

Figure 2: Criteria evaluation for database hosting services. Each metric is scored on a scale of 1-5, where 5 represents the ideal. The total score is out of 25 points maximum.

For this reason we will use AWS as our database hosting service. One advantage of AWS is that, as mentioned before, it is already currently being used by the client so there is no

need for any major logistics. In general, it did well based on all of our selected criteria, so we have high confidence in the capability of this technology for the WearWare system.

3.3 Application Framework

For users to interact with our system, i.e. allowing participants to register their Fitbit devices into studies and for researchers to view data from their studies, we need to employ an application framework. The framework will be used to design the web application that will allow users of the device to register themselves and their Fitbit device. This will also allow studies to be set up by researchers. In choosing an application framework to use in developing the WearWare system, we are primarily focused on three important metrics:

- Security is of the utmost importance, as thousands of users' data from their wearable Fitbit devices will be stored within the system's database.
- Speed is another important metric as the system must be able to efficiently process millions of data points and quickly deliver queries to the end user, who in this case would be the researcher.
- We want to find a framework that is relatively easy to implement and has a reasonable learning curve, so that we do not end up wasting development time on start up.

With these criteria in mind, we will look at and compare the four most popular application frameworks, namely ReactJS, Angular, Django, and Ruby on Rails. To properly conduct the comparisons and give proper scores for each metric, we designed small GUI web applications, and we also took into consideration opinions that we read from online open source communities, such as the GitHub community.

3.3.1 ReactJS

ReactJS is a prewritten library of Javascript code for Javascript based web applications and it is maintained by Facebook. Some of the advantages of ReactJS are that it is easy to learn and development time is exceptionally fast. ReactJS is also easy to test with its built in native toolkit. Unfortunately, this seems to be the extent of its advantages. As only a Javascript library instead of a full-stack framework, ReactJS will only cover the frontend UI of the system. This would necessitate some additional technology for the backend that connects to the database and Fitbit API. It is also vulnerable to cross site scripting attacks due to its susceptibility to HTTP vulnerabilities, meaning a user's credentials to log in to the application could be stolen. This would potentially allow for a situation in which someone other than the researcher running a specific study could gain access to the study's data. This makes React inadequate by our security standards.

3.3.2 Angular 8.0

Angular is an open source web application framework based on Typescript which is led and continuously developed by the Angular team at Google. Compared to React, Angular has a few more features that we are interested in. In addition to being easy to learn, the time from startup to a minimum viable application is, on average, the fastest using Angular. Angular has a suite of testing tools just like React does, and it is also a full-stack framework, meaning the backend part of the application will not be dependent on another technology. With that said, Angular only provides adequate security from CSRF / XSRF attacks as it is susceptible to HTTP vulnerabilities, which would, like the situation outlined with React, allow for an unauthorized person to gain access to a study's data.

3.3.3 Ruby on Rails

Ruby on Rails (RoR) is a full-stack web application framework that was first released as an open source code in 2004. Like the previous two options, it provides speedy applications, fast development, and has a full toolkit for testing purposes. It also provides a number of features and tools for fending off the vast majority of security threats. In addition to these advantages, it also has a large online community and ecosystem where people share tools and open-source code they have made. However, many industry professionals and RoR's vast online community generally do not recommend its use for developers who are not already well versed in Ruby. This is primarily because of its steep learning curve which has to do with the variety of independent concepts that are intertwined within Rails (one must know the Ruby language, the Rails framework, and the RubyGems package manager, among other things).

3.3.4 Django

Developed by Lawrence Journal World in 2003 as a full-stack framework built on Python – something our team already has experience in – Django is easy to learn, and while development time is not ideal, it is still fast, especially given that we will not have a large learning curve. Just like Ruby, it has a very large online community with thousands of packages that are ready to be used by developers. Moreover, Django will be able to provide the level of security we need for the WearWare system. One downside to Django is that it has some difficulty in handling many requests concurrently, so it may take longer to process queries to the database and deliver data.

3.3.5 Chosen Approach

As mentioned earlier, security was an important metric we looked at when evaluating potential frameworks. Speed and ease of development were equally important criteria for

our needs. After analyzing the information about these technologies, we were able to score them based on these three metrics, as shown in the figure below. While React was the fastest framework, it had the poorest security. Angular and RoR displayed similar performance, but were hampered by lower security and ease of development, respectively. While Django was the slowest of our examined technologies, it had both the highest security and ease of development.

	Security	Speed	Ease of Development	Total
React	2	5	3	**10**
Angular	3	4	4	**11**
Django	5	3	5	**13**
RoR	4	4	2	**10**

Figure 3: Criteria evaluation for application frameworks. Each metric is scored on a scale of 1-5, where 5 represents the ideal. The total score is out of 15 points maximum.

Based on the above reasoning, we will be using Django as our web application framework. Overall, it scored the highest of the four choices, and it showcased the best security, which was our foremost critical metric. It also provides some other advantages that aren't necessarily of high importance, but will nonetheless be helpful. These include the large community that supports Django, as well as the fact that it is relatively easy to extend or scale. It is also supported by most operating systems and databases, so Django will not interfere with our choices for database type and hosting platform. We hope to address the issue of speed with database optimization, rather than by sacrificing the advantages of Django for another framework that may be slightly better in only that aspect.

3.4 Development Client

Development clients allow for users to apply algorithms, create graphs, or otherwise make sense out of data. Picking a development client will allow us to have an easy-to-use language in which we can create a cohesive library of programs that can quickly analyze our data.

In choosing a client for development and testing we have decided upon the following criteria:

- The price point is important because we want the data and data analysis to be accessible to any researcher who wants to participate in WearWare without having to subscribe to a pricey data analysis tool.
- Speed is another factor in our consideration because there can be up to millions of data points that a researcher might want to plug into a plot or run through an algorithm, which at this level can take a considerable amount of time. Paired with our database and application framework choices, we want to streamline WearWare as much as possible.
- Both data analysis and graphing capabilities are a must have for this project, which is why we have only chosen development clients with known aptitude for data analysis.
- Learning curve and client preference impact our team and client's respective abilities to use the tool effectively, so we also take those into account.

Using these criteria we will break down and score the following development clients: MatLab, Octave, Python, Julia, and R. These five languages were chosen for their popularity and use among data scientists and researchers. To determine the speed of each language we ran a parse integers and quicksort for each language. We also assessed these languages by consulting with GitHub and other online coding communities.

3.4.1 MatLab

MatLab is a popular analytical programming language that has a robust toolbox selection and powerful data analysis and graphing capabilities. The drawbacks to MatLab are that it has the second slowest performance out of our choices, and that it is the only language suite that is not free. It comes at the cost of $2,150 for a perpetual membership, or an annual subscription of $860. With over 49,000 repositories in GitHub, the large community of MatLab users and its popularity among data scientists make it an appealing choice. However, the speed and cost are too much of a drawback to make MatLab a feasible choice. For the above reasons, it is also not a preference for our client.

3.4.2 Octave

Octave is a GNU high level programming language that was based on and is compatible with MatLab. Octave has a lot of the pros associated with MatLab, but without the cost. Octave is syntactically similar to MatLab, which makes it easy for users familiar with MatLab to learn. Additionally Octave uses less RAM and computing power than MatLab. However, Octave is just as slow as MatLab and lacks many of the toolboxes that MatLab provides. With that said, Octave is what our client is currently using and he is very familiar with it.

3.4.3 Python

Python is a very versatile and easy-to-learn programming language. It has a large community (over one million coding repositories on GitHub) and many open-source tool libraries made by people active in the Python language community. Another plus is that Python is speedier than MatLab and Octave in our speed trial; however, its data analysis and graphing utilities aren't as strong. Another downside is that our client has not had much experience with Python recently, and is therefore less preferable for him.

3.4.4 Julia

Julia is a new language created to combine the data analysis abilities of MatLab, the ease-of-learning of Python, and the speed of C. Julia is capable of calling both Python and C functions. Out of all of our client choices Julia is the fastest, but it does have a longer startup time. Because it is new, however, Julia lacks an established community of developers, and graphing utilities must be borrowed from Python. Another drawback of Julia is that it is a compiled language rather than an interpreted one. While this makes it faster, it reduces the ability to make quick changes to code, which our client has identified as a necessity. Moreover, our client had previously never heard of Julia, and it is not likely that anyone in his research community is familiar with it either.

3.4.5 R

R is a very powerful programming language that is primarily used by statisticians. It was speedier than Octave in both of our program runs, but slower than MatLab at Quicksort, and slower than Python and Julia in all tests. Because it's a language developed for statisticians it has a lot of built-in data analysis capabilities. According to statmethods.net (a statistics and R learning website) R has a steep learning curve and is known to be more difficult for people coming from computer science backgrounds to learn rather than statistics. Neither our development team nor our client have a particularly strong familiarity with it.

3.4.6 Chosen Approach

While all five of the development clients we explored as potential choices could provide the data analysis capabilities needed for testing, they face different limitations. Though Matlab is a popular and robust language, it scored low because it is not free to use. Meanwhile, Python and Julia scored similarly; this is a reasonable expectation, as Julia is based on the Python language. However, our client is not familiar enough with Julia to consider using it. R and Octave also received the same total score, but our client is currently more familiar and comfortable with Octave.

	Price	Speed	Data Analysis	Graphing Capability	Learning Curve	Client Preference	Total
Matlab	1	2	4	4	3	2	**16**
Octave	5	2	3	2	2	5	**19**
Python	5	4	3	3	4	2	**21**
Julia	5	5	4	2	3	1	**20**
R	5	3	5	4	1	1	**19**

Figure 4: Criteria evaluation for development clients. Each metric is scored on a scale of 1-5, where 5 represents the ideal. The total score is out of 30 points maximum.

Based on the above table, Octave, Python, Julia, and R all present solid options, but in accordance with our client's requests we have decided to weigh his personal preference more heavily than the other criteria. While he has mentioned considering Python or parallel use of Octave and Matlab, he already has a library of his own analysis tools written for WearWare in Octave, and he has claimed it as his preferred language. Although it did not score as highly as Python or Julia, Octave is a powerful analysis tool in its own right and we see no problem moving forward with that decision.

3.5 Fitbit Data Access

This section will differ somewhat from the above technological analyses. Our project requires that we retrieve data from the Fitbits of study participants; the only way to do this is to utilize the Fitbit API. Consequently, there are no technology comparisons to be made, but there are still many aspects of the Fitbit API that we will have to be aware of and contend with.

Fitbit's API stores all user data including readings of heart rate, activity level, and sleep analysis. For the product to meet requirements, the product needs to query for updated data every minute from the participants. These queries will include requests for updated heart rate, updated activity level, and updated sleep analysis. The following subsections

detail some of the challenges and limitations we will face while working with the Fitbit API. Fitbit's system of subscriptions, request limitations, and data summarization will require us to be resourceful and efficient. Fitbit API has flexibility in the retrieval of data, and as such we need to take into consideration the requirements from our client.

3.5.1 Subscriptions

Subscriptions are Fitbit's way of keeping track of what profiles the application can pull data from. If a participant is subscribed to our application, we will have access to request intraday data on heart rate and activity. Conversely if a participant is forcefully unsubscribed, we will be unable to access any of their data. Subscriptions have some restraints with regard to response time and successful requests. Speed and reliability from our application therefore are essential to success. Requests must be successfully made on average >90% of the time. If they drop below this threshold, Fitbit states that the participant will automatically be unsubscribed from the application and data will become unable to be retrieved. Requests must also be successfully made within 3 seconds of the request. If Fitbit API sends data to our application, the application must be able to close the connection within 3 seconds reliably. Failure to close connections within 3 seconds will result in Fitbit's API unsubscribing the participant.

3.5.2 Intraday Time Series

Intraday Time Series will be the core of our application. End-of-day summaries will not be accurate enough for our client (see next section for details). The three data points that can be retrieved from the Intraday Time Series will be the user's heart rate, activity, and sleep analysis. Each of these data points can be continuously obtained through the requests as long as our permissions have been granted from Fitbit. Fitbit requires an application before they give access keys to retrieve this data, but nonprofit organizations and researchers should not have a problem obtaining these permissions.

The Intraday Time Series can also be specified by detail level. Fitbit's API allows for second-by-second detail or minute-by-minute detail. The former will attempt to recover multiple readings of heart rate, activity, or sleep analysis per second, while minute-by-minute data summarizes data points over the course of a minute before reporting back. For example, if 4 heart rates of 80, 85, 90, and 95 were recorded and minute-by-minute detail were used, the four data points would be summarized by an average of 87.5. Second-by-second data on the other hand would retain the four readings separately. For our client, second-by-second data is needed. This is the project requirement that expands our database to millions of data entries per day in a significantly large study.

3.5.3 Limitations

The Fitbit API has some limitations that our team will need to either work with or around. Request limits are one of the major restricting factors, which Fitbit has set to 150 requests per hour. This should be manageable considering that requests can be given a start and end time (this will likely be per minute). The combination of heart rate and activity level requests will amount to 120 requests per hour, with some room to spare for client messaging and failed requests.

Request speeds are also important, as they must be consistently under 3 seconds or the API will force the connection to close. In repeated cases of slow connection closes, the user will be unsubscribed from the application and data will become unobtainable. With this in mind, Our optimizations for grabbing the data from the request and then closing its connection will be essential, especially as studies scale to thousands of participants and requests.

Fitbit has stated that end-of-day summaries are not perfectly accurate in some cases. When the Fitbit device is low on battery, or the memory on the device is near capacity, Fitbit may omit some entries from the summation calculation for the end-of-day summary. As a result of this limitation, we will need to focus entirely on the intraday requests and avoid relying on data from the end-of-day summaries. This is even further emphasized by the fact that our client is using this data for scientific research, meaning that the data should be as accurate as possible.

Fitbit data unfortunately is not consistent and continuous. When requesting heart rates over a minute for example, there may be empty entries or entries at odd intervals. Our database will have to be structured in a way that deals with this challenge.

Subscriber IDs are another limitation that we must consider when linking user accounts to the database. In some cases it may be necessary to have multiple subscription IDs for each user. These different IDs would keep track of activity, heart rate, and sleep patterns, respectively. This limitation simply means we will need to ensure that user's data is stored correctly even with multiple subscriber IDs.

Fitbit API requests are returned in a JSON file format. The files are well formatted and should not be an issue regarding integration with our database management system.

3.5.4 Viability

Overall, despite having some limitations that we may have to creatively work around, the Fitbit API will be able to provide us with the data we need and the means to properly report it to the end user, namely the researcher. Assuming that we properly organize our database and optimize queries, it should be possible to effectively implement the Fitbit API within our system's architecture.

3.6 Proving Feasibility

The above proposals form a preliminary plan in addressing the major technological challenges of the project. Our client has shared with us the Bitbucket repository that contains the code for the previous version of WearWare, including Winfree's prewritten library of Octave code. Our intention is to utilize this as a resource in further testing and validating our decisions.

4 Technology Integration

Previously, we individually introduced challenges for this project and proposed solutions that directly addressed those issues, but ultimately these design decisions will need to work together collectively to form a coherent architecture that satisfies all of the product requirements. This section will provide a more cohesive contextualization of the system, outlining how all of the aforementioned major elements relate to each other. The figure below illustrates a system diagram of how we currently envision the system.

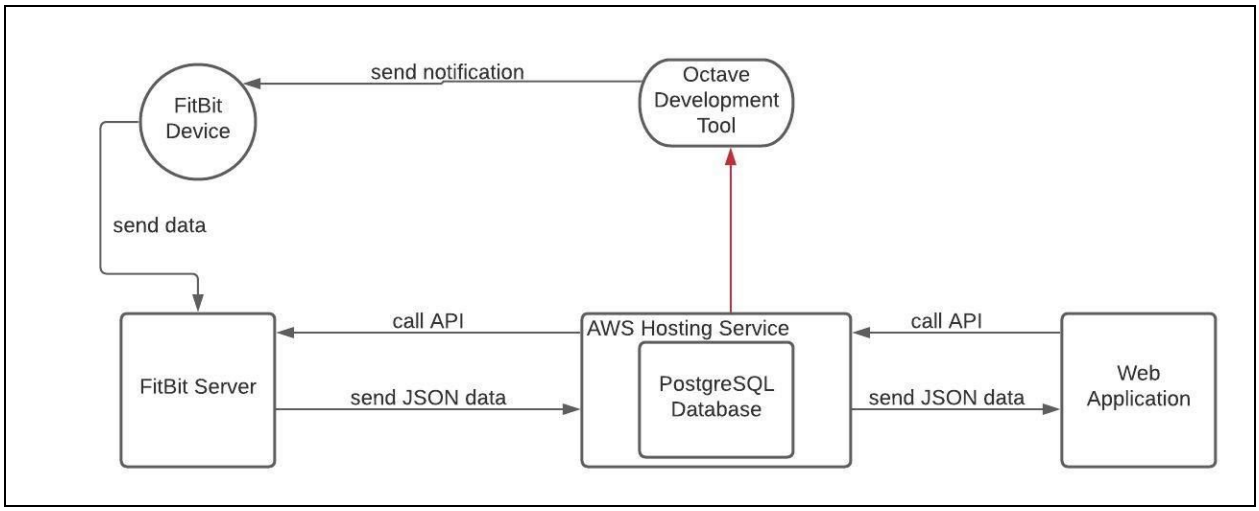

Figure 5: System diagram that shows an early sketch of the software architecture for the WearWare system.

As mentioned earlier in this document, the database system and host act as the center of the WearWare system, interacting directly with the Fitbit API, web application, and development client. The database will call upon the Fitbit API to retrieve Fitbit data, which is periodically sent from individual participants' devices, before extracting information from the JSON files it receives. This information can then be passed to the web application for the user to view; it can also be sent to the development tool component that will allow researchers to perform basic testing activities, as well as to send notifications to the specific devices of study participants.

A prototype will be developed in order to confirm that our technology choices are well suited for the final product. This prototype will prove on a small scale, that retrieving data from the Fitbit API and storing it in an efficient and speedy manner is possible when scaled to the size of a larger study. In the case that our conclusions on one or more of our technology choices prove to be false during the production of our prototype, we are confident that we can pivot from any of our selections and produce the viable finished product.

5 Conclusion

The problems posed by cardiovascular diseases are not insignificant; as such, it is important to conduct many large-scale studies that monitor participant activity in order to better understand the mechanisms and factors of these conditions. Although wearable devices pose one solution to behavioral health monitoring, researchers interested in employing them are often limited by price and the ability to conduct large, long-term studies. The WearWare system is intended to address this issue and to provide a robust, cost-effective study management platform that allows researchers to set up fitness and sleep studies while efficiently extracting Fitbit data from enrolled participants. To that end, the purpose of this document was to identify the major technical challenges and determine viable solutions that will become the core technologies of our system. We have conclusively demonstrated our rationale and decided upon the following solutions, as highlighted in the table below:

Technological Challenge	Proposed Solution	Confidence Level (1-5)
Database Management System	PostgreSQL	4
Database Hosting Service	Amazon Web Services	5
Application Framework	Django	5

Fitbit Data Retrieval	Fitbit API	5
Development Client	Octave	4

Figure 6: Final results of technological analysis and confidence levels in each, on a scale of 1-5.

Overall, we are reasonably confident that our chosen technologies will be effective in their roles and integrate well with each other. Given our analysis and attention to project requirements, we are optimistic that we will be able to develop a version of the WearWare system that improves upon past failings and meets the needs of our client.