Team DataBit

Software Design Document

Version 2

February 19, 2021

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

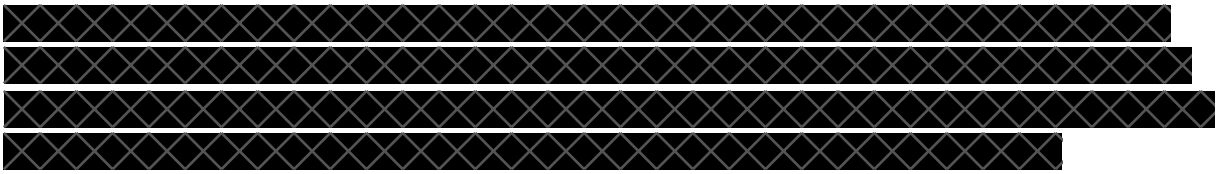
Cardiovascular diseases (CVDs) are the leading cause of death globally. In the United States alone, nearly half of adults over 20 years of age have some form of cardiovascular disease, and approximately 655,000 people die from heart disease each year – around one in every four deaths nationally. Moreover, CVDs pose 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. Typically, researchers enroll hundreds of participants in a study and observe them over a certain period of time. The nature of these observations vary, but in the case of health and fitness studies, researchers are often interested in gathering information about participants such as heart rate, activity levels, and so on. This data is collected and recorded over weeks, months, or even years, and is analyzed for patterns that support a given hypothesis.

To enable and support such large-scale, long-term studies, researchers require a way to efficiently maintain and collect exertion and 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. For example, devices such as Actigraphs™ are used in clinical settings to capture and record physical activity. These are small wearable devices that go on a person's wrist, much like a watch, and that monitor rest and activity cycles. However, Actigraph devices are expensive (around $250 per device), 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: a cheap and highly usable consumer version of an activity monitor as a potential alternative to Actigraphs in exercise and movement studies. 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, it is not immediately obvious

that they could be useful for large-scale studies. In particular, a major obstacle is collecting and accessing data collected by Fitbits.



near-real-time data access, or in communicating with and receiving feedback from the scientific community. These issues have prompted Dr. Winfree to devise the WearWare Project – an informatics platform for evaluating wearable fitness tracking. This would allow researchers to create and manage studies, enroll participants, and access their Fitbit data for further analysis. A first prototype was developed and proved the basic concept, but 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 to create a powerful data management cornerstone that we call DataWrangler as 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, participants, 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 serves as a blueprint for our final product. We will begin by going over our implementation, including our solution vision and the tools we will be utilizing to build DataWrangler. Then, we will outline our architectural design along with detailed descriptions of each of the major modules. Finally, we will provide an implementation timeline for our project in accordance with the design decisions given throughout the document.

2 Implementation Overview

Our main task in designing our DataWrangler module is to address problems from the current version of the project, namely concerning database optimization and other missing functionality that hampers interaction between researchers and participants. DataWrangler will be a high-performance data management foundation for WearWare that consists of the following:

- A well-designed database that stores all study participant Fitbit data, including activity levels, heart rate, sleep data, and other information at both the minute and sub-minute level. This will be optimized for performance such that queries are more efficient and requests take a reasonable amount of time.
- A flexible API that provides a powerful array of key data management functions to allow other WearWare modules to interact with the database.
- A simple test harness GUI that, in absence of the front-end WearWare modules that will be added in the future, provides a means to verify our API and database functionality and performance.
- Additional functionality that streamlines patient-researcher interaction. Namely, participants should be allowed to enroll themselves in a study through a unique, one-time sign-up link, while researchers should have the ability to send messages to participants via email.

Ultimately, the goal of our project is to provide database services for the WearWare system through an API. To accomplish this task, we will be utilizing various tools and technologies:

- **PostgreSQL**, an open-source relational database management system that will store all study and participant data. Given that database performance is a major priority in this project, we opted for PostgreSQL, which has a strong reputation in that category.
- **Amazon Web Services (AWS)**, a cloud computing service for hosting the PostgreSQL database. AWS is a relatively affordable option that can accommodate our system's storage needs while providing various scalability options, in anticipation of future growth. Specifically, we will be working with an EC2 instance that our client already uses.
- **Django**, an open-source web framework based on Python. Namely, Django REST framework will allow us to build the Web API needed to pull data from the database.
- **Celery**, a task queue implementation for Python web applications that is well supported by Django. This will allow our system to make real-time requests from the Fitbit API at the minute and sub-minute level for each participant.
- **RabbitMQ**, an open-source message-broker software which will construct a message queue for scheduling, since Celery itself lacks this functionality. It is the default broker for Celery.

For this document, we will continue to discuss these components, including their functionality and how they interact with one another.

3 Architectural Overview

In this section we will introduce the WearWare system architecture. This will give a high-level overview of our system and the components that will be further detailed in subsequent sections.

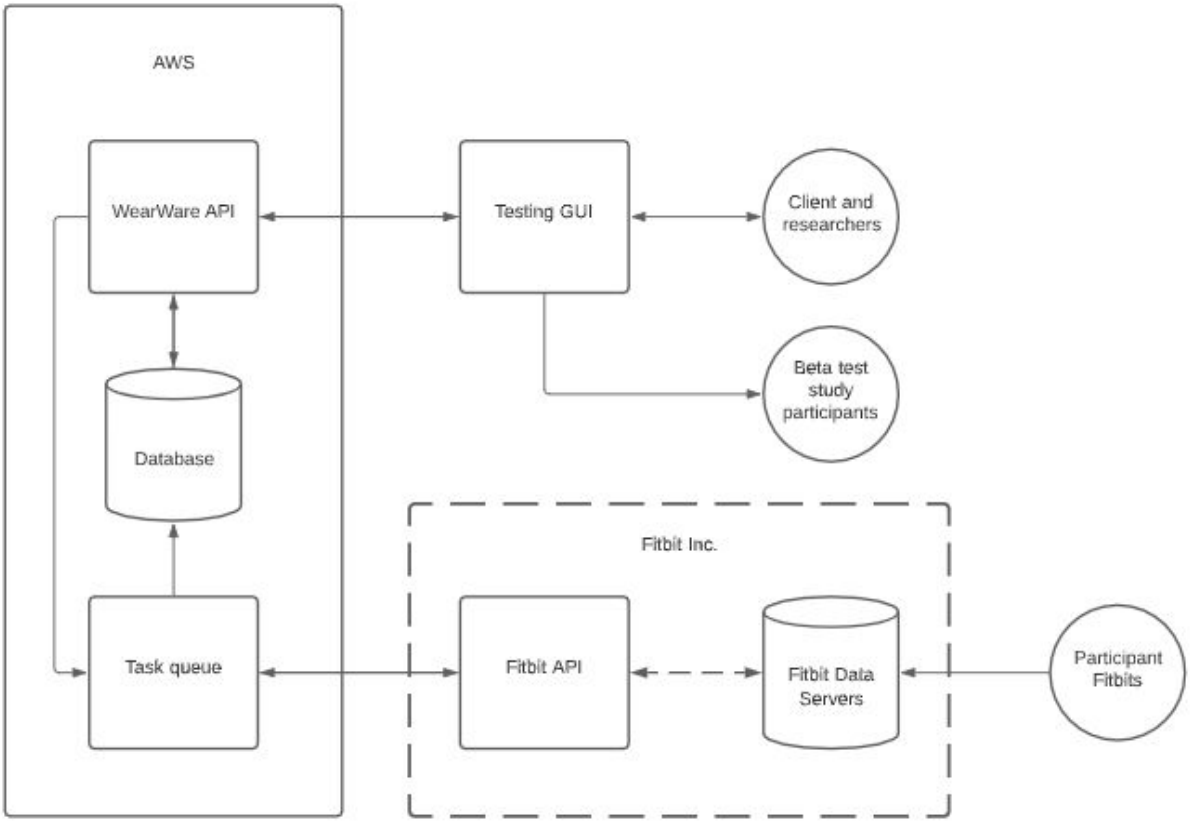


***Figure 1**: High-level architectural diagram.*

As shown in Figure 1 above, the WearWare system resides on an EC2 instance of AWS. The backbone of this system is the PostgreSQL database, which holds all study information and participant Fitbit data. The API is primarily responsible for providing researchers with access to this data, with various end-user flows in mind: creating studies, adding participants, retrieving slices of data, and so on. For the time being, these functions will be executable through the use of a rudimentary testing GUI, which serves as the direct intermediary between researchers and the WearWare system. It will also support a messaging system such that researchers can create email templates to send to participants

when a problem has been noticed (e.g., the participant has not logged any data in several days).

Fitbit itself provides data servers which directly collect and store data from Fitbit devices, as well as an API for accessing these servers. A task queue will be utilized for scheduling requests to the Fitbit API for participant data and storing it on the WearWare database. Requests will be scheduled through the task queue several times a day per participant and collected data will be stored in the appropriate table (outlined in section 4.1). Meanwhile, a researcher can connect to the WearWare API and call functions to access that database.

4 Module and Interface Descriptions

As indicated in Figure 1, we have identified three main modules in our system: the database, the API, and the task queue. The following subsections will describe each of these components and their interface functions in more detail.

4.1 Database Module

A well-designed database is paramount to our project's success. As the backbone of the WearWare system, the database is responsible for storing all data concerning studies and participants, including raw data that has been gathered from participant Fitbit devices at both the minute and sub-minute level. Figure 2 below illustrates our proposed design.

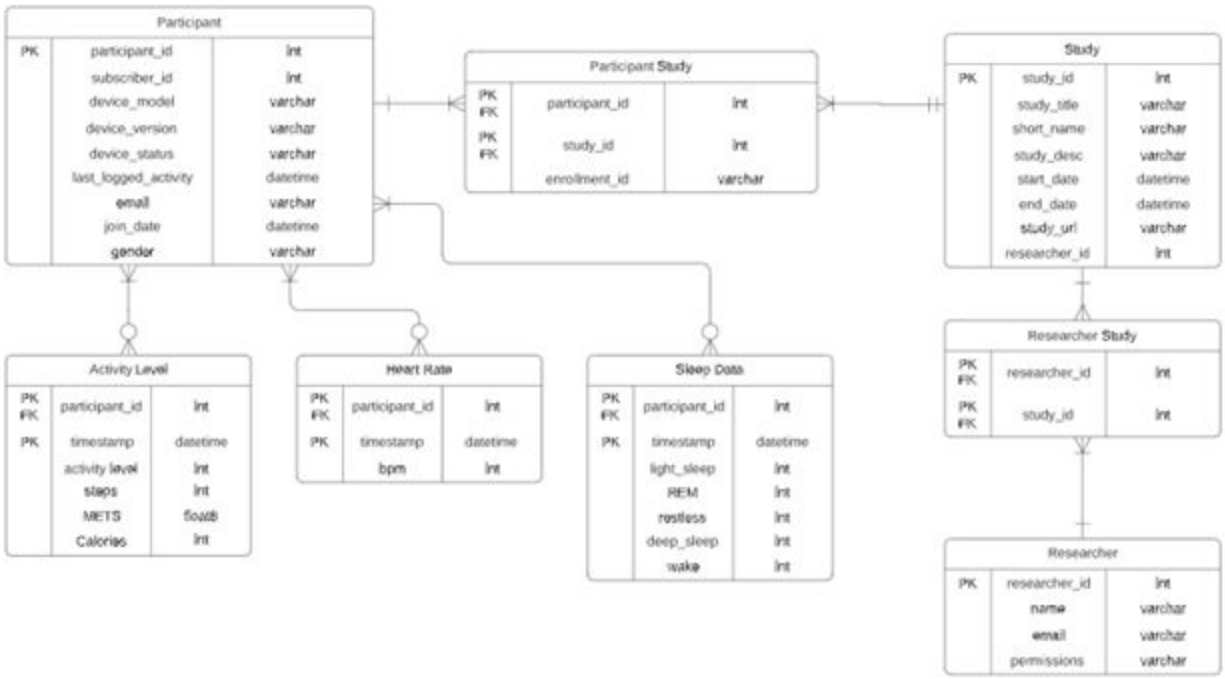


***Figure 2**: ERD of PostgreSQL Database*

4.1.1 Participant Table

This table contains a list of all participants registered in the WearWare system, which would number upwards of 20,000 entries. It stores information concerning the participants themselves, as well as their Fitbit device metadata:

- **participant_id (PK)**: A unique integer identifier for each participant. As the primary key, this is not nullable.
- **subscriber_id**: An integer used by the Fitbit API to identify devices. This is necessary for making requests to retrieve a participant's Fitbit data, and is not nullable.
- **device_model**: The varchar(20) model of the participant's Fitbit device.
- **device_version**: The varchar(20) current version of the participant's Fitbit device.
- **device_status**: A varchar(20) sync status of the Fitbit device to the WearWare system. A researcher should have updated access to this information at any given time, so this is not nullable.
- **last_logged_activity**: A timestamp that shows the last time a participant's data was logged into the database.
- **email**: The participant's varchar(255) email address. Allows a researcher to send a message to a participant in case of difficulties, such as if the participant has not logged any data recently. This is required for registration and not nullable.
- **join_date**: The date the participant registered to the WearWare system, which is filled when the participant is first entered into the database.

4.1.3 Researcher

This table contains all the researchers and research assistants entered in the WearWare system. Depending on permissions, these users can create/edit studies and pull raw Fitbit data gathered from enrolled participants.

- **researcher_id (PK)**: A unique integer identifier for each researcher. As the primary key, this is not nullable.
- **name**: The varchar(70) name of the researcher. This is not nullable.
- **email**: The varchar (254) email address for the researcher. This is not nullable.
- **permissions**: A char(3) field that explains whether a user is an admin (SU), a researcher (PI), or a researcher assistant (GRA, RA). Each user type has a permission set that allows them different levels of privileges to view/edit studies, and this field cannot be null.

4.1.4 Participant Study

This is a join table that resolves the many-to-many relationship between the Participant and Study tables. It establishes which participants are enrolled in each study. It also

includes an enrollment_id field which defines a participant by study (e.g., a participant in study ABC might be given the id ABC012).

4.1.4 Researcher Study

This is a join table that resolves the many-to-many relationship between the Researcher and Study tables. It establishes which researchers have access to which studies.

4.1.6 Activity Level

This table holds all of the raw activity data from each participant, accounting for millions of entries total. Each row is defined by the id of the participant in question as well as the time of the record, generally at the minute level. It provides the following data:

- **activity_level**: The integer result of a proprietary Fitbit formula based on exercise and daily steps (sedentary, mildly active, moderately active, heavily active, extremely active).
- **steps**: The integer amount of steps the wearer of the Fitbit device walked in a 24 hour period.
- **METS**: Metabolic equivalents, or the amount of energy used by a person at rest, represented by a double.
- **kcalories**: The integer number of Calories a Fitbit user logs to their device each 24 hour period.

4.1.7 Heart Rate

This table holds all of the raw heart rate data from each participant, accounting for millions of entries total. Each row is defined by the id of the participant in question as well as the time of the record, generally at the sub-minute level. It provides the following data:

- **bpm**: An integer value of the participant's heart rate at the given time.

4.1.8 Sleep Data

This table holds all of the raw sleep data from each participant, accounting for millions of entries total. Fitbit provides summary data of the minutes spent in each stage of sleep, but the data behind its calculations is not given. Each row is defined by the id of the participant in question as well as the time of the record. It provides the following data:

- **light_sleep**: The integer number of minutes spent in light sleep.
- **REM**: The integer number of minutes spent in REM.
- **restless**: The integer number of minutes spent restless.
- **deep_sleep**: The integer number of minutes spent in deep sleep.
- **wake**: The integer number of minutes spent awake.

4.2 API Module

A well designed API is what will allow the user of WearWare to communicate with the EC2 database on AWS storing all of the participant data. The WearWare API will use CRUD operations (create, read, update, and delete) to manage the information related to studies, participants, and researchers. Below, in Figure 3, is a representation of the WearWare API explaining how it will act when it accepts queries from a user. If there is some kind of error in the query, it will return a corresponding error code and the user will have to try again. If the query is correct and the results are available in the database, it will provide the results and complete the request, or if there is an error on the end of API, it will return an error and direct the user to try again.

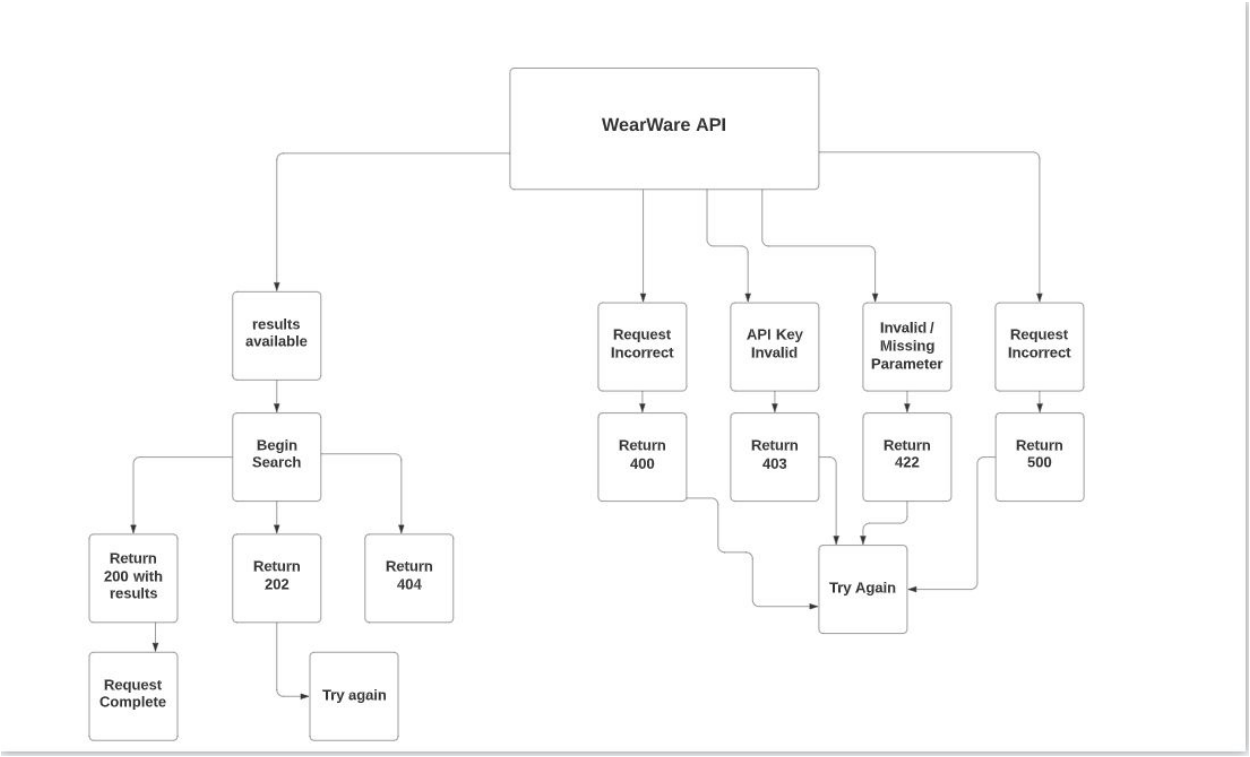


***Figure 3**: WearWare API flow chart*

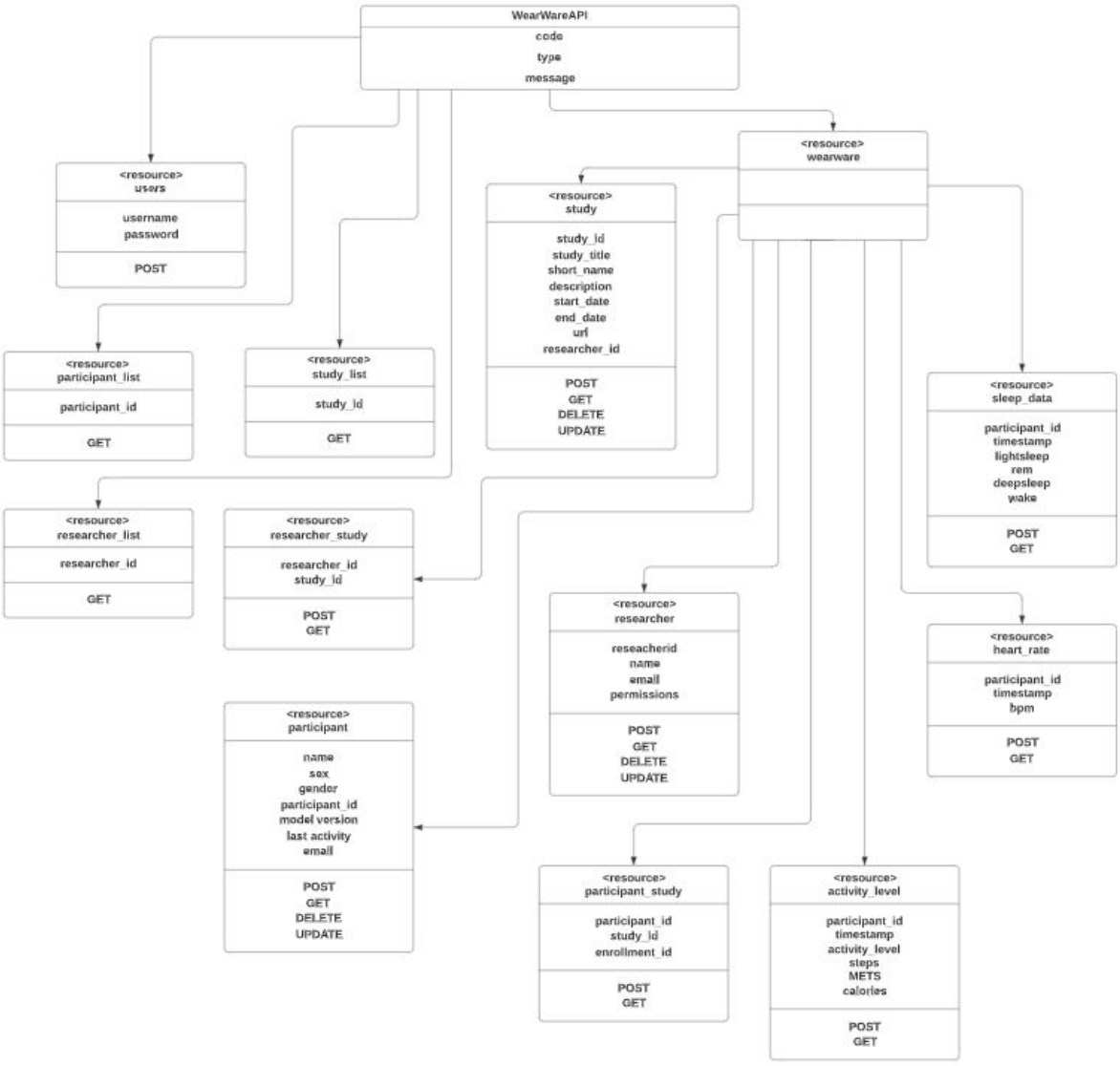


Figure 4*: WearWare API UML.*

Figure 4 is a visual representation of WearWare API usage and its interaction with the EC2 PostgreSQL database. The API features GET, POST, UPDATE, and DELETE methods, which differ depending on what data the user of the API is wanting to interact with. Registered users of the system can log in via a POST operation.

GET operations include allowing the user to retrieve a list of study data, researcher data, and participant data via their corresponding primary keys. The tables containing information related to sleep data, heart rates, activity levels, and participant studies (all studies in which one individual participant is part of) can be interacted with via GET and POST operations. A user of the system can use GET to retrieve this data related to either a

specific participant using their participant ID, or with a date range in order to gather large quantities of data relating to multiple participants.

The POST operation in these cases will be used by Celery and RabbitMQ to retrieve data from participants' Fitbit devices. The study, researcher, and participant tables will be accessible through all of the CRUD operations, as objects in these tables will need to be created using data from Fitbit, read by researchers, potentially updated by researchers (or Fitbit in the case of backfilling data), and deleted by researchers. They will be accessed through their primary keys, which are all depicted in Figure 2 of the database ERD.

4.2.1 Examples & Use Cases

In the case of a user of the wearware API wanting to create a study, they would input the required information in JSON format and send a POST request. Below is an example of an acceptable request.

```
POST /wearware/study/ HTTP/1.1
Host: WearWare.com
Content-Type: application/json
Accept: application/json
Accept-Charset: utf-8

  {
      "study_id": 1,
      "study_title": "title",
      "short_name": "ex01",
      "study_desc": "Example description",
      "start_date": "2021-02-13T14:20:00Z",
      "end_date": "2021-02-14T14:20:00Z",
      "study_url": "test",
      "researcher_id": 1
  }
```

Should the user want to send a GET request to the API to retrieve the information related to this study, the request is different from the one above in that it only requires the study_id field, as this is a unique identifier for each study.

Another use case would be if a user wanted to create a participant. It would be done in the following way.

```
POST /wearware/participant/ HTTP/1.1
Host: WearWare.com
Content-Type: application/json
Accept: application/json
Accept-Charset: utf-8

{
   "participant_id": 1,
   "subscriber_id": 1,
   "device_model": "ChargeHR",
   "device_version": "1",
   "device_status": "Synced",
   "last_logged_activity": "2021-02-13T15:27:00Z",
   "email": "sms968@nau.edu",
   "join_date": "2021-02-13T15:28:00Z"
}
```

Now, after creating this participant in the system, the next logical step would be to enroll them into a specific study. This is done by using their participant_id field and then the study_id field from a previously created study. An example of doing so using the test study and the test participant in the previous examples is shown below.

```
POST /wearware/participantstudy/ HTTP/1.1
Host: WearWare.com
Content-Type: application/json
Accept: application/json
Accept-Charset: utf-8

{
   "participant_id": "1",
   "study_id": 1
}
```

This would enroll the participant with the id “1” into the study having the id of “1”. The GET request for retrieving data related to a participant is done in the same manner as the GET request for a study, by using just the participants unique identifier, participant_id. Making a researcher and assigning them to a certain study is done essentially the same way

as making a participant and adding them to a study, just with the different parameters that relate to researchers. The sleep data, heart rate, and activity level fields are not directly interacted with by users of the API, instead activity is only PUT to these fields by connection to the Fitbit API via participants personal Fitbit devices. However, users can use GET requests to retrieve this data. An example of a request to retrieve heart rate data for a specific participant in a specific date range is as follows.

```
GET /wearware/heartrate/ HTTP/1.1
Host: WearWare.com
Content-Type: application/json
Accept: application/json
Accept-Charset: utf-8

{
   "participant_id": "1",
    "start_date": "2021-02-13T14:20:00Z",
    "end_date": "2021-02-14T14:20:00Z"
}
```

There are a few additional rules related to some requests in the API that only those with WearWare administrator privileges can make. These include making studies, making researchers, and then adding these researchers to specific studies. Those with researcher permissions are able to enroll participants into whatever studies they have authorization over.

4.3 Task Queue Module

The task queue serves as the core data collection module for WearWare to gather raw data from participants' Fitbit devices. It is responsible for servicing requests to the Fitbit API for intraday data, as well as storing the collected data back in the PostgreSQL database. Figure 5 below illustrates the subprograms that provide this functionality.

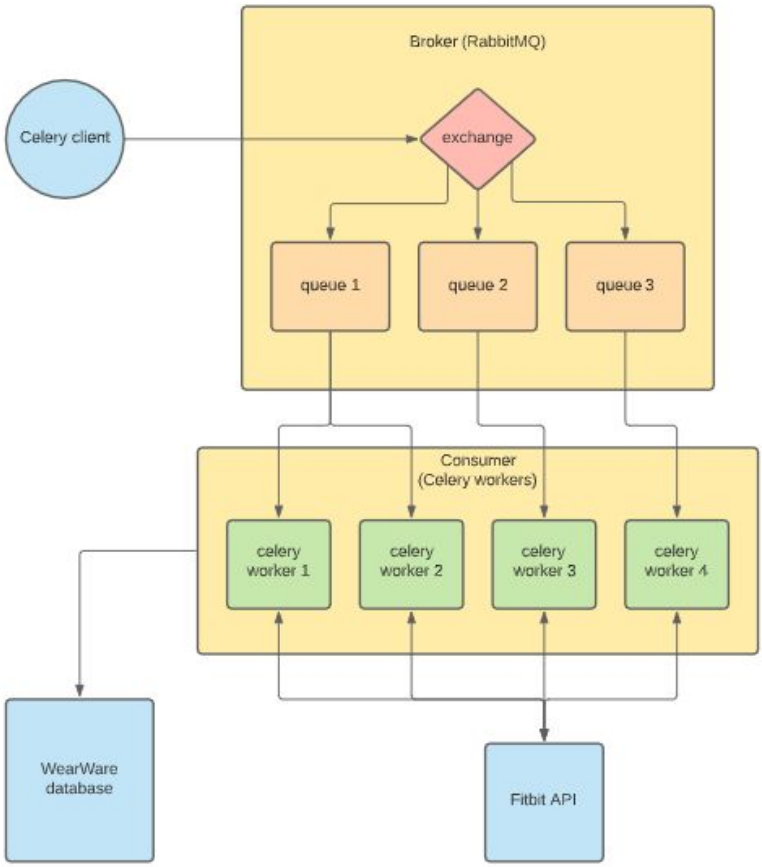


***Figure 5**: Scheduling subprograms for collecting raw Fitbit data.*

For example, requests to the Fitbit API for sleep data may only be necessary once a day per participant, since it can generally be assumed that most participants only sleep once at night. Even so, to periodically collect this data for each participant, the WearWare system needs to individually connect to the Fitbit API multiple times (upwards of thousands, in order to support large-scale studies) throughout the day, every day.

To handle this workload, an asynchronous task queue such as Celery can be used to schedule each call to the Fitbit API and execute these tasks in the background. The Celery client submits jobs to a message broker, RabbitMQ, which creates a queue based on job requests and distributes the tasks to Celery workers. Each worker is responsible for making one request to the Fitbit API to collect raw data from a certain participant device, using an Access Token that authorizes access to the Fitbit user's data. If the request goes through, the Fitbit API returns the relevant data, where it can then be sent to the WearWare PostgreSQL database.

5 Implementation Plan

We have broken our project down into tasks and drafted an implementation schedule for completing each of the major components in the system. These components are given from the modules detailed in previous sections: the database, API, and task queue. We will conclude with a testing and refinement phase where we make final adjustments to confirm that we have met all of our project's requirements. Figure 6 below illustrates this implementation plan.

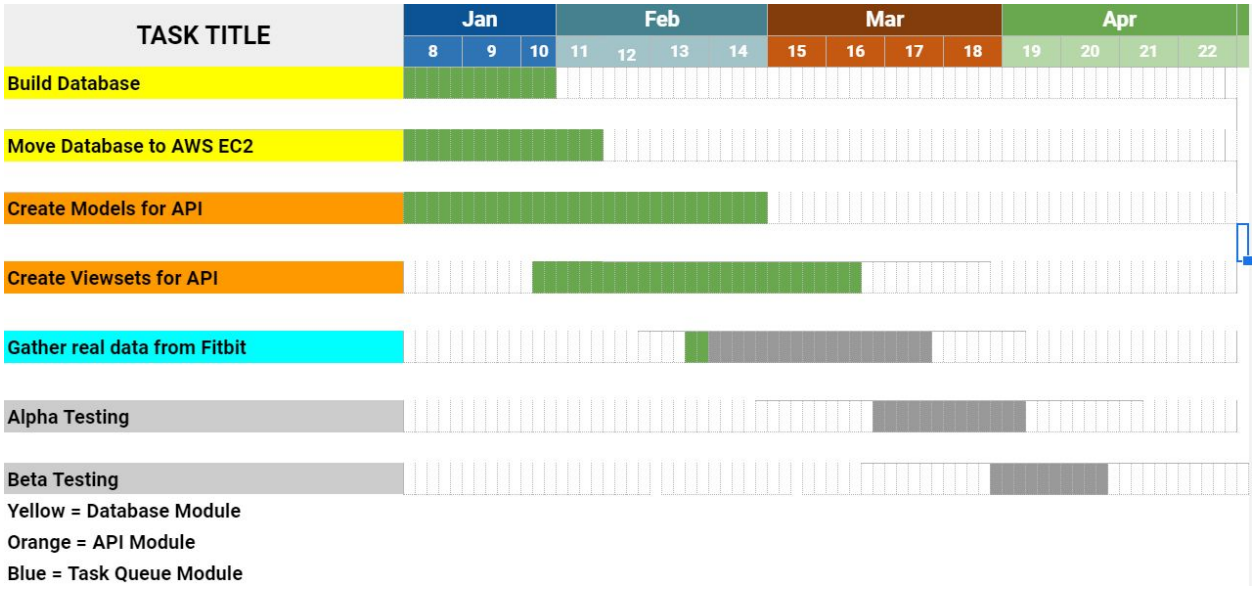


Figure 6: *Gantt chart of implementation schedule*

Our first goal is to build the database according to the ERD using PostgreSQL, including all of the necessary tables and fields. For preliminary testing and querying, we will need to populate the database with fake but realistic data that follows a similar format to the raw data returned by the Fitbit API. We will also need to gain access permissions and move this database onto our client's Amazon EC2 instance.

The next step in implementation is to actually create the API. APIs created using Rest Framework can be divided into two main sections: models and views. Models are just that; models of the tables in the database and their respective data fields. The data fields are then accessed via model serializers. Views are what allows the API to actually send and receive HTTP requests to and from the database. There are multiple different kinds of views provided by Rest Framework, but we chose to use the API viewset as this provides delete(), post(), get_object(), get_list(), and update() requests, which are what we need to provide the necessary use cases of WearWare.

Once our database and API are set up, we can start gathering real data from Fitbit. Our first step is to define a way to authenticate with the Fitbit API to retrieve one user's raw data. After this is accomplished, we will need to connect the API to Celery and RabbitMQ to schedule and distribute tasks to workers that can retrieve data from multiple participants asynchronously. This will allow us to conduct our final tasks of testing and refinement.

6 Conclusion

Given the prevalence of CVDs, which kill 655,000 people in the US each year, it is important for researchers to be able to conduct large scale health studies that can gauge the impact of physical activity and sleep on disease risk and outcomes. The WearWare system is intended to address this concern, by allowing researchers to utilize inexpensive wearable devices, namely Fitbits, and collect information such as:

- Heart rate, which can be tracked on a one-second interval.
- Activity level, which gives a range of a brisk walk to a full exercise routine.
- Sleep data, which provides in-depth analysis such as restless sleep, R.E.M. cycles, and light sleep.

An existing prototype of the WearWare system proves the underlying concept of feasibility, but has significant performance and feature limitations. The aim of this project is to build a solid, high performance database module to serve as the data management core of a future redesigned WearWare system.

In this document, we presented an overview of the system design that will allow us to accomplish this goal. We have identified the main modules - the database, API, and task queue - as well as how they fit within the overall system architecture and the essential functionalities that each must provide. We are confident that our design will be sufficient to meet the requirements of our client, and that we have an appropriate schedule for implementing all of the necessary components that will make WearWare a useful tool for conducting large-scale fitness and sleep studies.