

Leveraging Wearable IoT Healthcare Devices to Save and Improve Lives

Vyas Alwar, Jason Glass, Alexander Wenzel and Yi Zhang

Motivation

In the modern world of healthcare there are an almost limitless number of devices which can track and record various biometrics, like blood glucose levels, blood oxygen levels and heart rate. For the most part this data is now being collected by devices which can communicate over WIFI, Bluetooth-Low-Energy and other newer technologies like Zigbee.

Our team wishes to leverage this data and mobility to create IoT enabled healthcare networks which could potentially save lives. Imagine if your Fitbit could alert the hospital if it noticed your heart rate dropped to zero during usage? This is effectively coopting a device not intended to be a lifesaver into one. A blood glucose monitor could forward results to a server which runs a machine learning algorithm to help the user determine which foods and activities are helping and hurting their glucose levels.

Healthcare monitoring and mobile big data could be merged into one powerful health utility that has applications all throughout the healthcare and fitness industry. Devices could be combined through the use of IoT to create entire health monitoring suites out of what used to be disjointed components. Not only is this powerful but it is the first step towards truly “open-source” healthcare.

Our team is motivated by helping the world and saving lives that would have previously gone unsaved or unassisted. We believe this project could vastly improve quality of life for hundreds of thousands of people.

Design Goals

We plan to use an IoT enabled heart rate monitoring device such, specifically a Fitbit, to continuously track heart rate data, and potentially sleep and steps. This data will be sent to a server which stores this data and also actively tracks it for abnormalities. For example if the heart rate suddenly dropped to under 20 beats a minute the server would be capable of calling 911. We also wish to apply some element of machine learning to this data in the server backend. For example if the server noticed that a person’s heart rate always spiked at 2pm everyday it could forward this data back to the user and help them determine the cause, whether it be something obvious such as going for a run or something more involved such as eating a Big Mac everyday with fries and overloading their sodium levels. We plan to also include a correlation between average resting heart rate and hours of sleep.

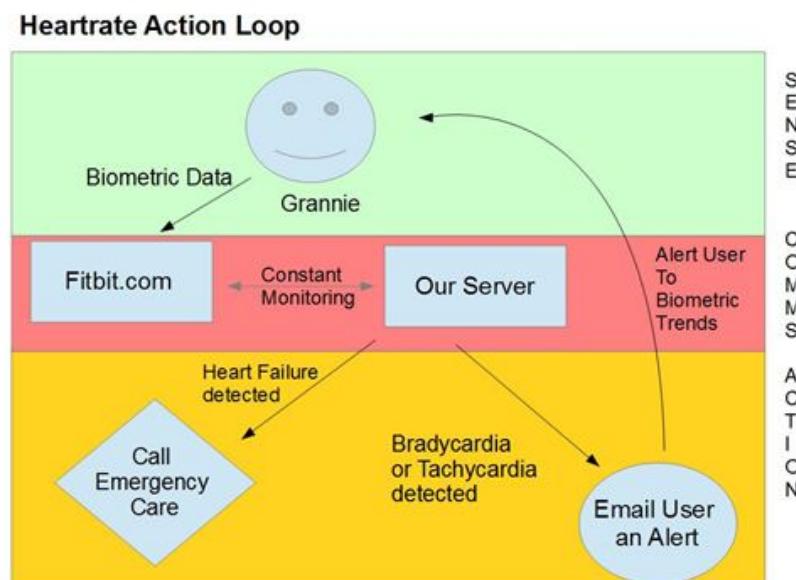
The IoT main loop is sensing->communicating->taking action. Sensing is clearly the measuring of healthcare metrics in this case and communicating depends on the metric but having your fitbit forward heart rate information to your phone or a server is communicating and applies to most metrics these days.

The taking action is largely dependent on the metric as well but in most cases involves either alerting a Doctor, the police, the user or some other stakeholder. For the heart rate case we already enumerated the ability to call an ambulance for you if your heart rate drops into cardiac arrest levels.

General Design

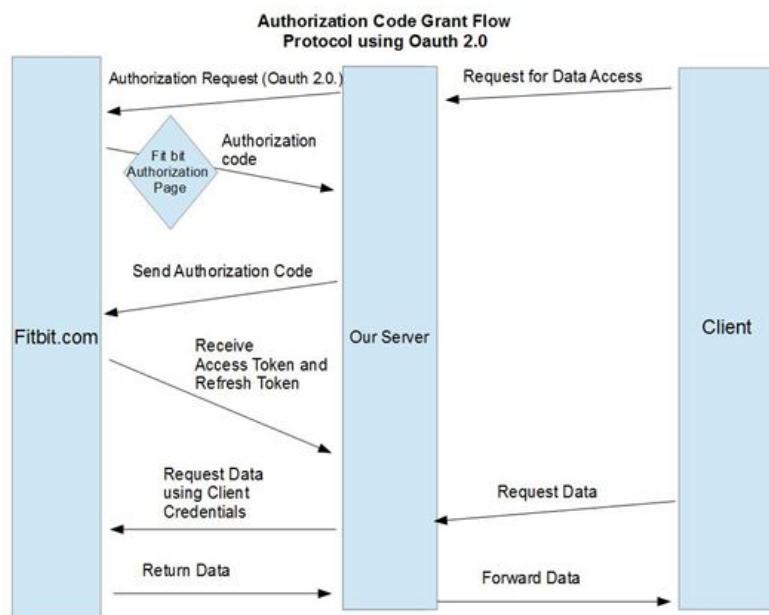
Our design is mainly focused on achieving our “Heart Rate Action Loop” (shown below) which fulfills the three main components of an IoT loop: sensing, communicating and taking action. The novelty of our system lies in continuous monitoring which only requires a wifi, bluetooth or theoretically Zigbee enabled device.

The system monitors heart rate data for abnormalities that could potentially be life threatening, such as ventricular fibrillation, a deadly arrhythmia. Upon recognition of a life threatening event the system is capable of contacting emergency services, and healthcare providers.



Our continuous monitoring system required the use of the Authorization Code Grant Flow or the Implicit Grant flow (RFC 6749) protocol which is an internet standard. This protocol is detailed in the below figure.

First the client initiates a request for data through our server, an authorization request is then passed to Fitbit.com. Fitbit redirects the requestor to an authorization page where credentials are entered. Upon successful credential recognition an authorization code is sent to our server. The server sends the authorization code back to Fitbit.com, which then sends a unique Access and Refresh token to our server. Our server uses these tokens to request data from Fitbit.com using the provided client credentials entered previously. Fitbit.com returns the data to our server which then either forwards it to the client or is displayed on our server's web page.



Final Specifications

We have fully implemented a server which provides the necessary infrastructure to allow users to authenticate with fitbit to allow our application to query their data. Fitbit allows API access via the Authorization Code Grant Flow or the Implicit Grant flow (RFC 6749), and to allow the greatest flexibility in terms of querying data, we choose the Authorization Code Grant Flow. We implement a server to complete this handshake using basic HTML and Javascript and a Python backend to perform querying and parsing operations.

To begin the process, a client loads our server's index page, which immediately redirects it to the Fitbit Oauth 2.0 webpage. The client authenticates themselves and the server receives an authorization code from Fitbit. When the server detects that it has received an authorization code from fitbit.com, it immediately sends a query for the corresponding access and refresh tokens, which allow it access to user data, completing the Authorization Code Grant Flow handshake. These codes are saved on the machine running the server where the rest of our application can use them to query user data.

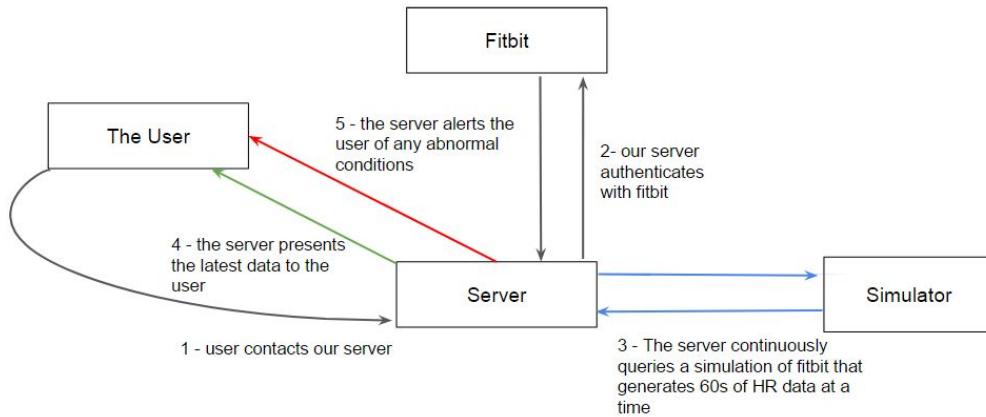
Due to an issue with Fitbit's API we were not able to have Time-domain heart rate data delivered to our server. We chose to establish our proof of concept continuous monitoring system by feeding our server simulated heart rate data that captured the 3 main events we were attempting to monitor, Premature Ventricular Contractions (PVC), no heart rate and dangerous arrhythmia; specifically atrial/ventricular flutter. A PVC is when the heart muscles do not properly fire in the correct order producing a weak, or dropped, heartbeat. While not dangerous in small quantities, they can be indicative of underlying heart conditions. Tachycardia and Bradycardia are when the heart is beating too fast or too slow respectively, medically defined as over 120 bpm or under 60 bpm. The dangerous ranges are closer to above 180 bpm or under 30 bpm for a healthy individual. During atrial/ventricular flutter the bpm will be over 200 bpm in most cases. The following section will go over our simulation details.

Simulation Details

Due to the technical issues we encountered with Fitbit, as well as the fact that none of us experience any heart conditions, it was necessary to build a simulator. To complete our IoT loop, we construct a simplified, simulated version of the Fitbit API server. The real Fitbit server sends much of its data, including sleep, steps taken, and heartbeat, as time-series data. This data takes the form of an ordered series of objects where each object has a "timeValue" timestamp in the format "HH:MM:SS" and a "value". The "value" field changes depending on the type of time-series data queried, but for heartrate data, "value" contains the heart rate that the Fitbit detected at that point in time during its continuous monitoring. With the real Fitbit API, it is possible to send queries with customized time frame and sample intervals, but in our simplified model, whenever queried, our server produces 60 seconds of time series heart rate data with a sample interval of 1 second, so 60 samples of heart rate data representing one minute of our simulated user's heartbeat. While Fitbit is able to accommodate a wider range of

queries in terms of intervals and time frames, this is still a reasonable query to the real Fitbit API.

The Simulated IoT Loop

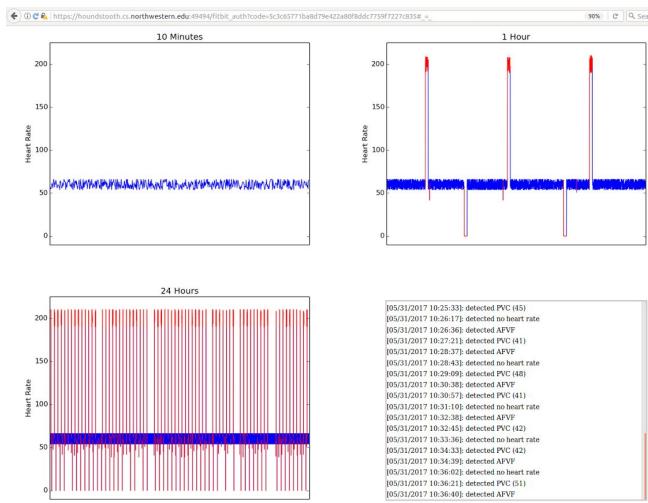


We use this structure for query responses (60 seconds of data at a time at a sample rate of 1 second) to simulate the heart conditions we described previously. A premature ventricular contraction (PVC) is a missed or weak heartbeat. To simulate this in terms of Fitbit's continuous average monitoring, we represent this as a sudden drop (by 10% or more) in the user's resting heart rate. The remaining two conditions, loss of pulse and atrial/ventricular flutter, unlike the single event PVC that should be noted, are dangerous events that can be confirmed with the detection of several consecutive abnormal samples. Accordingly, our simulator, to create data for either of these conditions, will generate between 30 and 60 seconds (samples) of consecutive data for atrial/ventricular flutter (heart rates greater than 200 bpm) or zero pulse (0 bpm). We note that the Fitbit device is capable of detecting the difference between a user with 0 pulse and no detection at all - i.e, the user is not wearing the device. The simulator generates abnormal data based on hard-coded probabilities. The majority of the time, it generates data within $\pm 10\%$ of the resting heart rate for the user it is simulating. For all of our validation, we use a resting heart rate of 60 bpm hard-coded in the simulator.

Our final IoT loop results from swapping the actual Fitbit API server for our simulated version. Once our user authenticates their real Fitbit account with the real Fitbit authentication server, our server begins sending HTTP requests to the Fitbit simulator server (which runs on the same machine) just as it would to the real Fitbit API server. Our simulator runs on an accelerated time scale - it sends a new query (which always results in 60 seconds of simulated data) every 5 real seconds and analyzes the returned data. Our server maintains a running

average of the user heart rate that it refreshes after each query, and uses it to determine thresholds for detecting abnormal heart rate conditions. To validate the simulator and the detection software on the server, we keep logs of both, and after running the simulation for 24 hours of real time, we observe that all abnormal events generated at the simulator have a corresponding detection log at the querying server.

After detecting abnormalities, the next step is to both keep the user up-to-date on their health statistics, and notify them of pressing concerns. When an abnormal event is detected, the server generates an email which it sends to the user with a timestamp and details of exactly which kind of abnormality was observed. In addition, we developed a simple website to present all monitoring results to the user. Views are available for the last 10 minutes, hour, and 24 hours, where all samples that fall within acceptable bounds (within 10% of the current resting heart rate) are plotted in blue and abnormal events are plotted in red.



While two of our extreme conditions, atrial/ventricular flutter and zero pulse events, are likely to be readily apparent to the user without having to consult the website, this constant monitoring is valuable for conditions that are not emergencies but still concerns over time, such as the PVC condition we track. The ability to monitor for these events constantly without impacting the user's quality of life is a benefit of our system.

Next Iteration Goals

The following are considered stretch goals for any further implementation of this project:

We have a long list of practical improvements we would implement on the next version of this project. Finer Granularity continuous monitoring would further improve our design, with Fitbit

API we are limited to a data request every 30 seconds. The data is also their interpretation of data from a plethysmogram, the LED devices used to measure heart rate. The lack of lower level voltage data makes it hard to perform various stochastic modeling on data.

We would also have fully integrated with Fitbit's API, we were limited to only one specific set of sleep data and could not end up retrieving the relevant time-marked heart rate data we simulated. Access to more parameters would allow for machine learning to be implemented but also for users to see more visuals concerning their biometrics.

The machine learning opportunities our team saw were the correlation of factors such as sleep, steps taken and resting heart rate. We will provide various visuals to our users to further optimize their user experience.

Lastly we would further improve the amount of personal customization to our action loops for each user. For example being able to customize when and how alerts are sent and also linking with other IoT devices.

What We Learned

We learned a valuable experience in dealing with external API and customer service. We encountered roadblocks related to not getting Fitbit's API to work properly, and a delay in help from their team. Furthermore using their API limited the ways which we could access the data we wanted and reduced the granularity of important metrics like heart rate to 30 second slices that had to be analyzed, instead of second by second chunks delivered continuously.

Overall we are happy with our project and believe the concept and motivation is very important. Going forward we would like to implement the above next steps to take our platform to another level of service.