Data Aggregation and Analysis of Wearable Devices

Aakash Sheth - Team Leader
Ben Kixmiller- Key Concept Holder
Haythem Ebrahem - Key Concept Holder
Brian Nguyen - Team Communication Leader
George Ndemi - Key Concept Holder
Jonathan Campbell - Team Webmaster
UnityPoint Health - Client
Simanta Mitra - Advisor
Dec 1610

Table of Contents

Project Overview

Purpose

The purpose of this project it to develop an application to query, store, and analyze a person's biometric data. This data includes steps, sleep, heart rate, weight, and calories. We will using FitBit's Developer API as our data source.

Background

UnityPoint Health provides care through several different provider options. This system provides data to the proper care provider based on needs such as severity, or purpose of visit, essentially a tool for less emergent situations. This model is loosely based on being able to provide generic care for many, rather than specialized care to few. Since we are focusing on the general care recipients, we also focus on general patient metrics, as mentioned above.

Problem

In order to improve patient care, UnityPoint seeks to develop an application that can be used by care providers such as nurses, or primary care physicians. The problem lies in the data source. Since UPH does not have an infrastructure to gather data, specifically from wearables such as FitBit, AppleWatch, or Garmin Vivofit, we had to develop an application using platforms that were available.

It is also required to create an application that should be accessible to the care provider, and the patient. This requires the chosen platform to be accessible from multiple locations, whether that be a mobile or web application.

Finally, the health data collected must be structured in a way that can be useful, as well as manageable. At this point, the data collected follows the format of the API provider, but it is not necessarily useful.

Solution

Currently, there are several fitness tracker applications and hardware. For ease of use, availability, and popularity, we chose to focus our attention on Fitbit. Development went through several revisions, but the general idea was to create a primary web application, followed by a secondary mobile application. Each of these applications should allow the end user, whether that be the client or care provider, an easy way to view some of their general metrics as provided by FitBit. This will include expanded metrics such as weight over a period of time, or all time steps. These metrics should provide value to a care provider to in turn provide valuable care to the patient.

Deliverables

The deliverables for this project consist of a web application as well documentation of the results of our development. Since this project is a prototype, we chose to develop on a stack that was easily accessible to developers, and not necessarily the production environment. This was mostly due to the security issues, as well as request approval to get students access to UPH's development network.

Documentation is also crucial. Our prototype runs on a different stack that may not be easily available in an enterprise environment. Documentation will help to transition to any new environments. It also important for the next group that may continue development of this project.

Implementation Details

Backend

Our application is primarily web based. It consists of querying data from one location, essentially through a proxy, saving that data, and then returning it to the user. We chose to implement this idea using the client server architecture. In this case, we have several servers, and a single client (only single user access is supported at this time). The FitBit server provides resources through REST endpoint authenticated with OAuth2. Our Node server acts as the proxy for the user to login, and then forward requests to FitBit with the authentication token provided.

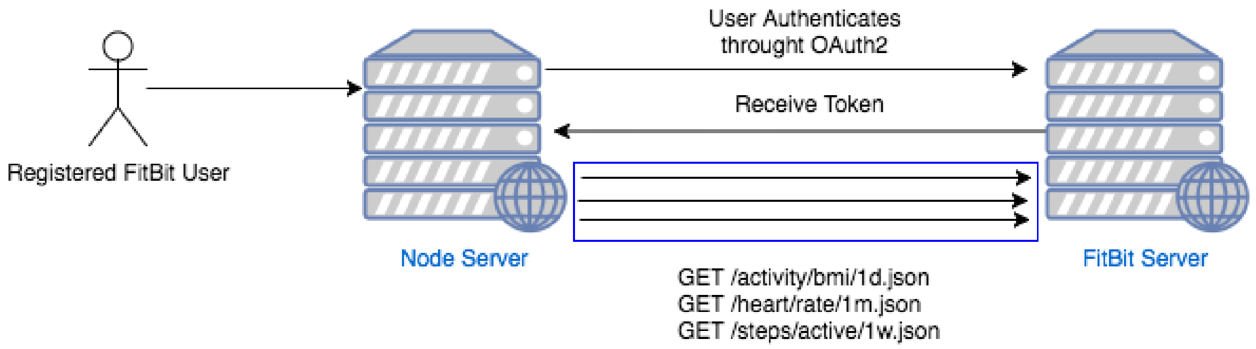
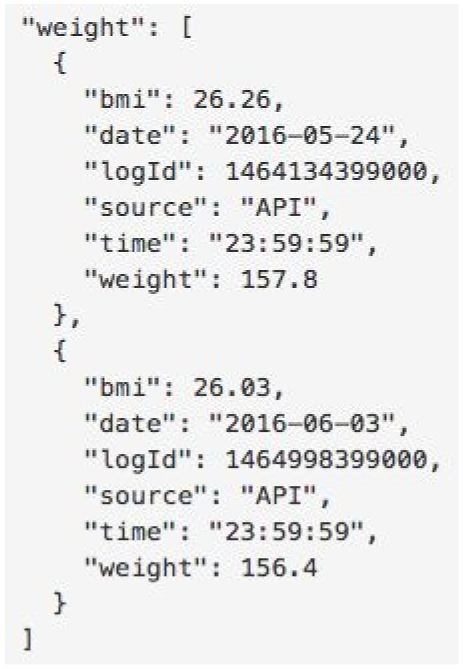

Registered FitBit User
User Authenticates throught OAuth2
Receive Token
Node Server
FitBit Server
GET /activity/bmi/1d.json
GET /heart/rate/1m.json
GET /steps/active/1w.json

Figure 1 - Typical request flow.

FitBit provides several endpoints that we found useful. To retrieve data from an endpoint, we use HTTP GET to query a resource. Each request must be accompanied by an access token. The token provides authenticity, expiration, and validity of the current logged in user. Without the token, the request will be rejected.

The response is formatted in JSON. It is returned by the FitBit server to our Node server, where data is saved, parsed, and then made available through our own REST API. This allowed us to intercept the response and then customize it to make available as a service. This way, not only could we separate the data source from the from the view, but we could also create a service that can be accessed by other services, making it modular, and extensible.



```
"weight": [
  {
    "bmi": 26.26,
    "date": "2016-05-24",
    "logId": 1464134399000,
    "source": "API",
    "time": "23:59:59",
    "weight": 157.8
  },
  {
    "bmi": 26.03,
    "date": "2016-06-03",
    "logId": 1464998399000,
    "source": "API",
    "time": "23:59:59",
    "weight": 156.4
  }
]
```

Figure 2 - Sample response

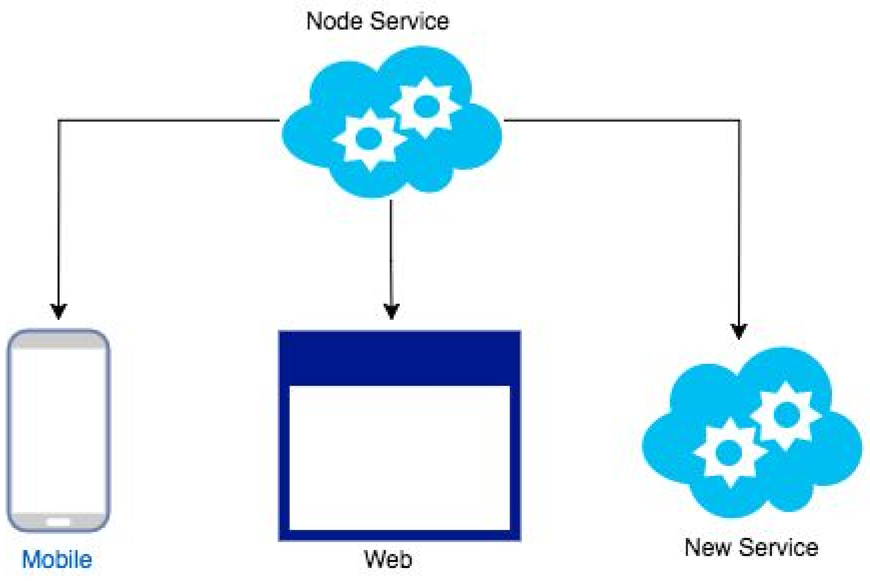


Figure 3 - Service Architecture

Front End

We decided to AngularJS for the majority of the front end for the web application. Angular allowed for rapid development, as well as a great user experience since it was so so responsive. It also allowed us to create a front end that was separable from the backend end and easily maintainable. The support community for Angular is backed by Google, which also aids in development in terms of documentation, reliability, and scalability.

We also chose to use Bootstrap as a way to make a presentable UI that was intuitive, but also simple so users could gain value from a quick glance. Bootstrap, alongside font libraries and icons, provide a generic feel and appearance to the app and kept it as simple as possible.

Testing Process and Results

Continuous Integration

We manage our code using git, specifically hosted by GitHub. By using GitHub, we were able to create pull requests that made merges much easier. It also allowed us to view code that other team members contributed.

Each of our pull requests consisted of testing before merging, but that practice slowly faded away since we were more constrained in terms of time. Moving forward, specifically for this project, we think a test driven development (TDD) plan would implement the product better. There are several requirements issues that we are unsure of, and a TDD approach would allow more transparency better engineering of these requirements.

Unit Testing

As we started developing, we first began on the iOS application and then the web application. In the iOS environment, we began to create unit tests using XCTest, but in the end proved to be more intensive than initially thought. We decided to lower the priority of unit testing on iOS.

For the server and front end, we leveraged Jasmine and Karma to create functional tests. We used these tests to verify correct response codes from various endpoints, proper bootstrapping of internal and external services, and controller verification. These tests helped to remedy several errors when integrated with new code.

Appendix I : Operation Manual

Overview

The web application is full stack Javascript using NodeJS, AngularJS, MongoDB, and ExpressJS. This allows us to employ NPM as a package manager reducing hassles for potential dependency management. In addition, we are able to manage versions of each dependency separately.

In terms of the iOS application, we chose to use CocoaPods as a package manager. In the end, we faced several complications specifically with Apple's move towards Swift 3. Many of our dependencies did not convert to Swift 3, and so required too much development time to fix. This application is not fully executable.

A REST API not yet implemented in the web application was created with Spring Boot and connected to a Microsoft SQL Server. Despite it not being implemented, it is working as expected and allows users to post and retrieve data from the MsSql database with tables formatted in the correct way. Spring Boot is an extremely well documented framework and has built in dependency management, which will allow for easy modifications in the future as well as robust performance.

Setup

To setup the web application, the host machine must have MongoDB and NodeJS installed with proper permissions. Once MongoDB and Node are installed, clone the repository, and navigate to the project root. Run the following commands:

```
$> npm install
$> node server.js
```

Demo

To visit the page, you must first authorize with FitBit. With the server running, navigate to:

```
http://localhost:3000/fitbit/authorize
```

After you authenticate with FitBit, navigate to:

```
http://localhost:3000/index.html#/sleep
```

From here, you can view sample data for a logged in user. For testing, you can using the following credential for FitBit:

Username: kaylamaeknight@gmail.com
Password: Diseaseelement1

REST Service Demo

To set up the Spring Service, the host machine must have Tomcat or Spring Tool Suite(STS) installed. For the purposes of this demo I will be showing how to set up the service with STS. Another requirement is that the host machine be white-listed by the Azure server to which it is connecting.

To run the Service, install STS(the latest version) and clone the RESTService git repository. Start up STS and Import the RESTService project as a Spring Boot project into STS (file->import). All information needed is on the application, simply run the project as a Spring Boot project.

To test the Service, some end-points can be leveraged. Each health metric we are storing (steps, heart rate, etc) have both an endpoint that will create a new entry and an endpoint that will retrieve information. To test retrieval of data, use the following url (host:port`/step/{user_id}/{startDate}/{endDate})` . Replace 'step' with 'hr', 'sleep', or 'weight' to see values returned. Start and end date must be formatted as YYYY-mm-dd.

Appendix II: Alternatives and Revisions

Initial Design

Our initial design consisted of a single mobile application accessing various endpoints from services such as FitBit, Garmin, or even the AppleWatch. Each of these had its own development exceptions. For instance, the Garmin API was only accessible after a fee was paid and a privacy disclosure created. For AppleWatch, there was no external API, and so all data would have to be forwarded from the device to a remote server in order to save patient data. For Fitbit, the main obstacle was OAuth2 authentication with iOS.

The design was originally thought to be modular by creating interfaces for each type of wearable device. Development based on this idea was rather difficult because we didn't have a clear idea of what each of the APIs could deliver. It also required us to find a user of these technologies that we could use a mock data, but were unable due to many of the restrictions mentioned above.

Before we were aware of these issues, we originally began to prototype a mobile application written in iOS. We made significant progress in the UI, but were lacking in other areas. Authentication with OAuth2 took longer than expected, team members did not have easy access to the iOS dev environment. Team members did not pick up as quickly in Swift programming as anticipated.

Failures

The iOS application was a failure due to development complications, lack of developer knowledge, and the move towards Swift 3. Although this was a failure, we did gain insight into the process we wanted to accomplish and what type of data we wanted to display.

We did not get the Spring Rest API integrated with the Node service. Fortunately, the Rest API is complete, so future integration with the web application will be relatively simple.

Revision

Since we lowered the priority of the iOS development, we increased development on a web interface. Many team members have experience with developing for the web, and so we moved towards a web application. Initially we were unsure of what tech stack to use, but it prove difficult to integrate with UPH infrastructure, mainly because of security issues.

Appendix III: Other Considerations

Other Teams

For the first semester of the project, we worked with a CyBiz team. The CyBiz team gave us input as to what type of statistics we should look into such as weight, steps, and sleep. They worked with a team that was working on this project the semester before we started working on it.

Client

We met at UnityPoint two times in Des Moines to talk to them in person. One time was during the first semester to introduce ourselves to them and what plans we had for the rest of the time we'd be working on the project. The second time was during the middle of the second semester to demo our project to them and receive input from them on what we could change.

We had weekly meetings over the phone with UnityPoint once a week. These meetings were checkups to see if both our design group and UnityPoint were on the same page as far as if what we were doing was something that UnityPoint agreed with. We would have Carey Novak, leader of setting up senior design projects at Iowa State, as well join our calls to get feedback from him.

Development

The largest obstacle to our development was lack of developer input and contribution. There were several times when we were behind our timeline due to poor communication and lack of initiative. The requirements also were not always clear, but we managed to engineer them to deliver a protypical product that can be used to further identify functional and non-functional requirements.