1. Introduction

1.1 Purpose

This document was made to specify the requirements and details of the Software to make the API endpoints for Augur. This SRS will describe the intended useage and internal software design.

1.2 Scope

Group 5's API endpoints are intended for the use of health and management monitoring of active projects. Each API taking a look at individual's contributions, group contributions and individuals across multiple groups to see if they are having their work divided. The master overseer Goggins has requested the need for more endpoints for more functionality to benefit management, so the whip may be cracked so to say. Having these endpoints will be easy for the front end team to make beautiful charts that highlight the difference in work quality amongst peers so they may be better scrutinized. Having these endpoints will save the front end team countless hours in the scraping of data and connecting the data from the servers.

The API will be built in several sprints. Firstly, deciding and designing on the most deprecating ways to monitor the employee's contributions to the development of Augur. The term sprint will indicate an agile method to production of the API endpoints. There will be several days of design and communication from the master overseer. The second sprint will consist of understanding and implementing basic functions to connect to the already established infrastructure of the database and begin building off of the web framework. The third sprint will consist of the core web application modules being build. After this phase 80 percent of the final product will be complete. The fourth and final phase will consist of unit testing, advancing the software to improved states and simplifying the overall components.

1.3 Assumptions and Dependencies

A) Given the agile nature of the development process it is assumed that the master overseer will be willing to help through the projects pipeline.

B) because this application is web-based, the entire system will be available on any web browser, but with limited views to the entirety of the database content.

2 Software product overview (hint: Augur is a good place to start http://augur.osshealth.io)

Augur is a github dependent open source application that monitors all activities of a/several repositories on github. Its stores relevant data that can be easily accessed and made into info-views that help understand the health of an individual repository/any sub component of a repository that is desired for health statistics.

3 System Use

3.1 Actor Survey

System features:

Generate / View reports

Administrative manager:

Administrative manager is the main user of the system. This actor is responsible for entering in the desired name of the employee or repo they are interested in seeing the workflow of.

System Features:

- Enter name of employee and repository
- Enter name of Repo
- Enter name of employee

Design features are extremely simplified so that all data will be populated with only the entry of the desired entity that they wish to be queried. The API endpoints and front end will be developed later to provide more smart decisions for the administrative user, so that they will not need to make uninformed queries with complicated constraints.

4 System Requirements (including 2 use cases, a system functional specification, and a list of non-functional requirements)

4.1 Use Cases

Repo contributor:

A repo contributor wants a personal “GitHub Report.” They gather data about their personal commits over time to produce a graph to visualize their commits over time.

Repo manager:

A manager of a repo identifies points of interest in the life of the repo to identify peak productivity. They want to see times of peak productivity in workers, so they use the endpoint to gather data to visualize the commits over time.

Repo manager:

A repo manager wants to see the distribution of work for an employee among multiple repos. They are able to gather data on which repos an employee commits most and/or least to, and use this insight to assign work to multiple developers among multiple repos.

4.2 System Function Specification

A search bar that when the data field has complete is hidden and populates the screen with a loading bar to indicate that it is being process.

Proper feedback for the User if the results return no values, as well as suggested results that may be found in the database, similar to the data in the data field.

Simplified JSON file with only the required data for the API endpoints to limited potential errors cause from loading in unnecessary data

Simplified view that show smart queries of the desired search

4.3 Non-functional requirements

Usability:

The design process will be completed in such a way that users will not require and training on how to request the data. Simply putting in the name of what they request will query desired results

Reliability:

The Queries should be designed with high data integrity as to not cause poor API Jsons return that do not populate the desired views with the proper info. Or worse cause the entire view to crash.

Performance

Queries will be structures and optimized to not require long load times, due to querying from the entire database, but from data tables with tight scopes around that desired search.

5 Design Constraints

This system will run as a web application with access via an encrypted SSL Login. It will run on all major web browsers – Internet Explorer, Fire Fox, Chrome, and Safari. Each we browser adheres to web standards differently and Augur must run seamlessly on any platform.

The system shall be developed using Python/Flask, SQL(Augur Database), Apache and windows

Good coding practices like cohesion and loose-coupling should be used as to allow for proper scaling and maintenance of the cod in the future

6 Purchased Components

Nothing was purchased, everything was given for free. This is an open source project.

Augur – Open source Database

Python – PyCharm free IDE compatible with flask and free access to students.

7 Interfaces

Interfaces are to be designed by the front end team of the project. The intended views of the data would be timeline based charts like a line graph to monitor usage by whatever portions of time specified. If there are many returned values from an individual or repo. The values will be put on shorter timeframes to make the number easier to understand and more meaningful. Conversely the data will be put on much longer timeframes if there is low returned values from the request. Meaning the user has not made many commits or the repository has not had much activity.

The primary interface for a user is a web based browser that is HTTP over SSL. Traffic is directed at the Augur database server. The application server will internally call services using API's with Python over HTTP protocols. Access of the database will be performed using Flask with standard POSTGRES ports.

Currently all user interfaces will be available publicly. The Application must use Pythons API and Asynchronous JavaScript and XML will be used for server communication