Software Engineering - CS 4320 Fall ’19

Assignment 8: Individual Requirements Analysis - Augur

Matt Hudson - Group 5 (API)

Table of Contents

Introduction

1.1 Purpose

This document is a high-level overview of the Augur system. These include an overview of the system's usage, requirements, design, and constraints. This document is not intended as a complete overview of the Augur system; it focuses on the overall goals and architecture of the Augur system, and how the current implementation uses this system. It also details the purpose of Augur and its parent organization CHAOSS in open source software development.

1.2 Scope

Augur is a system that studies the health and other associated metrics of open-source remote code repositories (known colloquially as *repos*). The primary hosts of these repos use **Git/GitHub**, the most popular repository hosting services available. In line with the philosophy of open-source software development, these metrics and repository information are available publicly, and intended to assess the health of these repositories, and to aid in research related to open source project health.

Additionally, these metrics assess the health and development of software communities that develop these open-source projects. Analyzing this data is critical to understanding what makes software project development inclusive, productive, and healthy. In addition, this data is used to assess problems in the software development lifecycle, and what can be done to prevent similar situations in other communities.

Augur is open source software and maintained by the CHAOSS community. Members of this community work to design the Augur software, and also to design implementation-agnostic metrics the system measures. The Augur project is freely available on GitHub and is open to additional contributors not affiliated with CHAOSS. The CHAOSS Metrics Committee identifies new metrics that can be measured, and the Software Committee works on developing software products such as Augur to collect and present these metrics.

1.3 Assumptions and Dependencies

A) Augur utilizes data provided by GitHub to assess the health of repositories hosted by this service. Therefore, Augur depends on the functionality of GitHub and the free availability of repository metrics.

B) Augur is open source and freely distributed, and is optimized to run on Linux systems. Individuals can freely use the metrics produced by Augur, and contribute to the project.

C) Augur is a Python Library that uses Flask to host a web server. Therefore, Augur depends on Flask and Python.

Software Product Overview

Augur is a Python library and web server that implements the CHAOSS project for open-source software repository health metrics. Specifically, Augur focuses on *human-centered* metrics and data science strategies. Augur is an implementation of the CHAOSS project's metrics on software health and sustainability, and provides three important components for these metrics:

1) Data collection - Augur allows collection of repository data, as well as creating the defined metrics on software health.
2) Data access - Augur runs a backend API that serves metric data in JSON format, that enables applications to use and present this data.
3) Data presentation - Augur additionally runs a front-end data visualization that presents the accumulated data in an easy to understand and relevant format.

To this end, Augur provides a backend and frontend. The backend is responsible for collecting and maintaining repository data in a centralized data model; in this case, a relational database model through MySQL. This data is accessed and exposed through a RESTful API. The frontend utilizes Vue to display sample visualizations of this data, although other front-end applications can make use of the Augur API and the data it provides.

The purpose of Augur is to present this data in many data formats, such as .csv files, JSON, and to provide visualizations in .svg format.

System Use

The use of the Augur system can be generalized to its three components: database, back end, and front end. In the database, automated programs called *workers* collect data from repositories being measured by the system. This data is then inserted into the Augur database. An example workflow would be that a worker will fetch commit data of a particular repo and update the database to reflect this new data.

In the back end, end users or applications make requests to the API. Typically, other applications access the back end since the data is raw and not visualized. The back end server then extracts data from the database, packages the data into a well-defined JSON format, and returns this data through a RESTful API. The data is handed off to the requester, where it can be manipulated in a multitude of different ways.

The front end is typically accessed by end users, not other applications. Users go to the front end address to receive visualizations of the data returned by the back end API.

Actor Survey

Workers - automated processes spawned by Augur that retrieve data from GitHub and insert it into the database. They interact solely with the database, and are independent from the backend API and frontend web server.

Housekeepers - automated processes that maintain existing data in the database.

GitHub Repositories/Contributors - contributors who work on the repos Augur surveys. Contributors indirectly feed Augur data on the repos they work on and by working on open-source software projects.

End Applications - End applications that access Augur's API endpoints in order to obtain metric health data. These applications can range from further visualization of the data, or further processing of the data.

End Users - the interested users who want to examine data Augur has compiled on open-source projects. End Users are more likely to access the front-end application.

System Requirements

Use Cases

1. Access commit data to visualize commit proportions for a repository

Actors: End Applications / End Users

Description: This use case specifies an application or user using the Augur system in order to visualize a proportion of commits for a specific repository.

Basic flow of events: A user such as a project manager is interested to view a summary of their project's commits. They need to visualize how many commits each contributor is making. The **end user** initializes the connection to Augur's Front End service. This service (the **end application**) accesses Augur's API endpoint providing the specific *repo_id* and *repo_group_id* needed to access this data. The API endpoint accesses the database and returns the results. Then, the resulting JSON is used by the end application to generate a graph showing proportions of commits for the repository.

2. Access and store repository data for commits

Actors: Workers / GitHub Repositories/Contributors

Description: This use cases specifies the Augur system pulling in data from GitHub for a specific repository in order to assess its health and produce metrics.

Basic flow of events: Contributors in an open-source software project in GitHub commit to various branches in the repository. After a scheduled time (for example, once a day) or when triggered by an end user, the worker accesses the repository and retrieves the number of commits during the period between now and the last time it checked. The worker utilizes the provided SQL credentials to add this data to the database.

System Functional Specification

Database Storage
GET - provides access to specific tables, assuming proper authentication credentials are provided.
PUT - provides an interface for workers and housekeepers to add and update data in the database.

Back-End API
GET(endpoint) - a request from an end user/application for data for a specific endpoint. (there are many endpoints in Augur, for example `top-committers`, etc.). For each endpoint, the server should return a response with the desired information. The GET Method (like HTTP's GET method) should **not** alter the data in the database.

Front-End Application
GET(endpoint) - similar to the backend API, the end user requests an endpoint to visualize. The server should send a response for the web page for the user to examine.

GET_API(endpoint) - this function calls the back-end API to obtain its JSON results. The front end application will then present the JSON in a more readable and relevant format such as in a graph.

Non-Functional Requirements

1. **Usability** - the front-end visualization of data must be familiar and navigable by users not familiar with Augur's internal workings. Additionally, the front-end shall provide visualizations that show important relations between data sets.
2. **Reliability** - each service should be reliable in returning a result, even in the cases of error. i.e. the server should never respond with an HTTP 500 error in cases of insufficient data; it should respond with a legible response that the data is not found, for example. Additionally, using GET methods should return the same result for each run of the application, assuming the data is still the same.
3. **Performance** - workers should update the database on a regular basis, and at times as to not interfere with other system functions. Endpoints should serve data as fast as possible, and provide an asynchronous handling of requests.

Design Constraints

1) The database model shall be accessible by Workers and associated maintenance programs, and by the internal API server.

2) The three major components of the system (database, API, Web) shall function separately, and not tightly coupled with one another. One should be able to replace implementations with others that utilize the same interfaces.

3) The front-end server should not directly communicate with the database, and vice-versa, in accordance with common software architectural patterns such as MVC.

4) The metrics and data gained through the Augur system shall be independent of the specific languages or frameworks used in repositories Augur analyses. In other words, the specific contents of repositories should not affect these metrics, in order for the metrics to be implementation-agnostic.

5) The system shall have as much uptime as possible, in order to serve data and visualizations. Server maintenance that affects uptime should be scheduled and minimal.

6) The database should avoid holding redundant data, and shall not hold the same information twice, unless to relate separate tables with each other.

7) The API server shall consistently return relevant results, even in the case of error. The server should not response with an error code such as an HTTP/500 error except in cases of actual server error. In other words, the API server should handle edge cases such as insufficient data gracefully.

8) The front-end service shall use HTTPS with an SSL certificate in order to provide standard data encryption and security.

Purchased Components

Augur's code is freely hosted on GitHub, so no additional cost is necessary to host the program code. Augur requires a Linux server (ideally running Ubuntu) to host the database, expose the data through a back-end API server, and host the front-end web application to visualize the data.

Hardware needed:
Ubuntu Server - components include CPU, RAM, Hard Disk Storage, and Network access. Provisioning of these computational resources could be acquired by a cloud hosting service such as Amazon EC2 or Microsoft Azure; a benefit of this approach is that hardware resources can be increased. Pricing for this service varies depending on system usage and size. A medium Linux server begins at **$18.67/month** with 100% usage. Depending on the accessing needs of the project, additional resources can be acquired, but would increase the monthly cost.

Alternatively, the project can invest in permanent hardware to serve the application; however, this risks system outages, and cloud services such as EC2 are generally reliable and handle configuration of the system's IP address.

Software needed:
The software needed to develop and deploy Augur is open-source and freely available. This includes Python, MySQL, Apache Web Server, and Flask.

The system will also require a reserved domain and HTTPS SSL certificate to ensure security and promote user acceptance of the project. Domains can be acquired with an SSL certificate for **$20/year** and require a small amount of configuration. Other DNS and SSL certificate services are feasible, but incur additional costs.

Interfaces

Hardware Interfaces

The server will bind to the DNS address of augur.osshealth.io and will serve the back and front ends of the application. The back-end API will be accessible through port **5000** through HTTP(S), and the front-end will be accessible through port **80/443** (HTTP(S)). If a cloud hosting service is used to host the application, a CNAME record will be needed to redirect augur.osshealth.io to the address of the instance.

Software Interfaces

The server will host the following servers: Web Server, API Server, Database Server. The single Ubuntu system will host these servers, and they shall communicate internally. To follow standard software architectural models, the front-end shall not directly communicate with the database model. Therefore, the database shall communicate to the API server, and the front-end web server shall communicate to the API server.