Individual Requirements Analysis for Semester Project

Jeremy Breese

- Introduction

- Software product overview (hint: Augur is a good place to start http://augur.osshealth.io)
- System Use, including an actor survey
- System Requirements (including 2 use cases, a system functional specification, and a list of non-functional requirements)
- Design Constraints (at least 5)
- Purchased Components (at least 1)
- Interfaces (at least 1)

Introduction

One of the biggest challenges with large sets of data is that it is often difficult to interpret to a person who is inexperienced in the field. There can be raw data points, which are often impossible to make use of without other technology, such as the total number of commits on a repository. There can be refined data points, which can them a bit easier to use, but still not quite readable, such as being able to filter this data down to specific users, or size of the commit. Once data is taken and made into presentable graphs is when almost anybody can open that webpage and make some sense of the numbers. This is the goal of our project, as the front end team we would like to be able to take in massive amounts of data and make it presentable so that anybody is able to make use of it.

Software Product Overview

The technology stack that we decided would best fit our needs and desires is Ruby on Rails. Ruby on Rails provides an easy way to distinguish where data is being processed; on the front-end client side, or the back-end server side. With this we will be able to optimize performance by running all API calls through our back-end and then using a post request sending the data back to the client. The data is not sent in its raw form; we will sort it into models and json objects so that it will be easier to work with on a browser. The advantage to this implementation is that the possibly sparse resources on the client's machine can be saved to draw the data using javascript libraries from pre-existing objects. Although there will be some work to be done on the back-end, we feel that the product will have a very strong and well thought out front end, that was designed with user needs in mind. This entire system can be hosted on any machine, but we will be deploying it on the servers that are provided to us in class.

Feature Overview

Once a user visits our webpage, they will be presented with all of the repository groups that the augur platform supports. Once they are here, they can click on any group to see the repositories within it. At either of these first two pages the user will be able to sort them by a number of parameters, including alphabetical, most recent activity, number of commits, and number of collaborators. Once inside the group they can select a repository, and this will bring them to the data page. This section is where our webpage becomes the most useful to the end user. Using the API data, it will show meaningful statistics that are easy to understand and interpret. These statistics include but are not limited to the total number of commits, the top committers, the amount of code written by each contributor, a chart showing activity by date, active contributors (determined by last commit), and the most active branches.

Use Case

With our system the only type of user that will exist besides the developers is the end user. These users can be software engineers, project managers, and business analysts just to name a few, but all of them will have the exact same access. There is no data input necessary for our platform and that will allow us to have one standard user. If a repository wants to be added to our platform they would have to be in contact with augur and they would be the ones handling the addition to augur. Our platform would automatically accept new repositories and groups.

Actor Survey

Software Engineer Team Lead @ Cerner Corporation

“This type of software is nothing new to the industry, for as long as software has been profitable people have been creating analytic tools to find out how to best manage their team and support their ideas. The thing that I like about this though is that it gives anybody the ability to get meaningful analytics without having to pay a lot of money to a third party like we do here. With all of that said I think that this is still a useful and meaningful tool that every development group needs in one way or another.”

System Requirements

Use case 1

Actors	User
Brief Description	A software developer is looking for a new project to devote some free time to. They know that they don't have time for a leadership role, but they want to be apart of an open source project that already has a foundation.
Flow of Events	1. User connects to http://augurinformatics.com/ 2. User clicks to sort groups by most recent activity 3. User selects a group 4. User clicks to sort repositories by number of committers 5. User selects repository 6. User will get contact information for the most frequent and presumed leader of the repository
Alternate Flow of Events	1. API endpoint returns no data 2. The user is presented with a messaging informing them that augur is likely down and to try again later

Special Requirements	1. The system should use a cache to have faster loading times 2. The system should present a loading animation

Use Case 2

Actors	User
Brief Description	A team lead wants to reward a member of their community with a thank you gift. They use our platform to find a suitable candidate
Flow of Events	1. Team lead visits URL 2. Home page loads the repository groups 3. Team lead selects their group 4. Team lead selects their repository 5. Data is presented data about the amount of code written in the master branch by each contributor
Alternate Flow of Events	1. Team lead visits URL 2. Because of a data parsing error, the back end returns 500 3. An error page is loaded
Special Requirements	1. Caching should be in place to load data faster

System Functional Specification

Backend

The backend of our system will define models for repository groups, and repositories. Once a webpage is visited it will instruct the backend to either call the API to get new data, or if the data is reasonably up to date (<20 minutes old) it will use cached data. With the raw data it will parse it into the appropriate objects. The repository group model will essentially just be a collection of repositories. The reason to sort them into this higher level object though is for optimization reasons, and to get data on the entire group. Once a repository group is defined, it will go through the data provided and sort it into each repository. To lift some weight off of the resources on the front end we will use json objects inside the repository object so that it can be easily accessed by our JavaScript graphing libraries. Once all of the objects have been successfully populated it will send the objects via POST to the front end.

Frontend

Once the backend of the system sends the objects in json, the JavaScript will parse them into their own JavaScript objects. These objects will be strategically designed to easily be compatible with chartist.js, which we will be using to display data to be interpreted by the user.

Non-Functional Requirements

Usability

The webpage should be user friendly
The webpage should have clickable definitions to statistics

Reliability

The server should be available at all times
The server should check the health of the API before making calls

Performance

Use caching of data to make loading faster
Provide loading animations to show the server didn’t stall

Design Constraints

Web Technology

HTML
CSS
JavaScript
jQuery

Server Side

Ruby on Rails

Use of the augur API for data gathering

Purchased Components

The servers provided to students will be used for this at no additional cost.

Interfaces

The interface will be accessible to any user with web access and a compatible web browser.