Jeremy Breese
Zach Snyder
Brad Schinker
Henry Sills

Goals:

- Create a simple and intuitive UI that allows a user to view and understand Augur usage data
- Create a project that can read an API endpoint to obtain the data
- Display a variety of Augur usage data to meet the users needs
- Use chartist js to provide meaningful and easily interpreted graphics

Endpoints:

Repo Groups: http://augur.osshealth.io:5000/api/unstable/repo-groups/

Repos: http://augur.osshealth.io:5000/api/unstable/repo-groups/ (group #) /repos

Sub Project Count : http://augur.osshealth.io:5000/api/unstable/repo-groups/ (group #) /sub-projects

Abandoned Issues : http://augur.osshealth.io:5000/api/unstable/repo-groups/ (group #) /abandoned_issues

Contributors: http://augur.osshealth.io:5000/api/unstable/repo-groups/ (group #) /repos/ (repo #) /contributors/

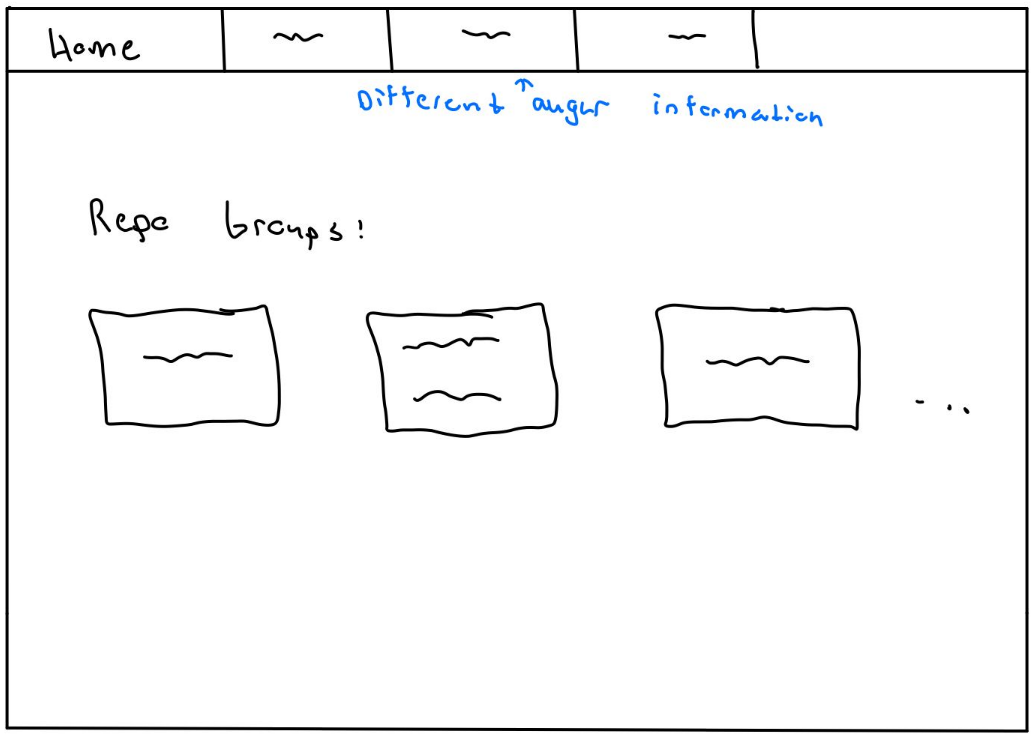
Pull request acceptane rate: http://augur.osshealth.io:5000/api/unstable/repo-groups/ (group #) /repos/ (repo #) /pull-request-acceptance-rate/

Active issues: http://augur.osshealth.io:5000/api/unstable/repo-groups/ (group #) /repos/ (repo #) /issues-active/

Wireframes:

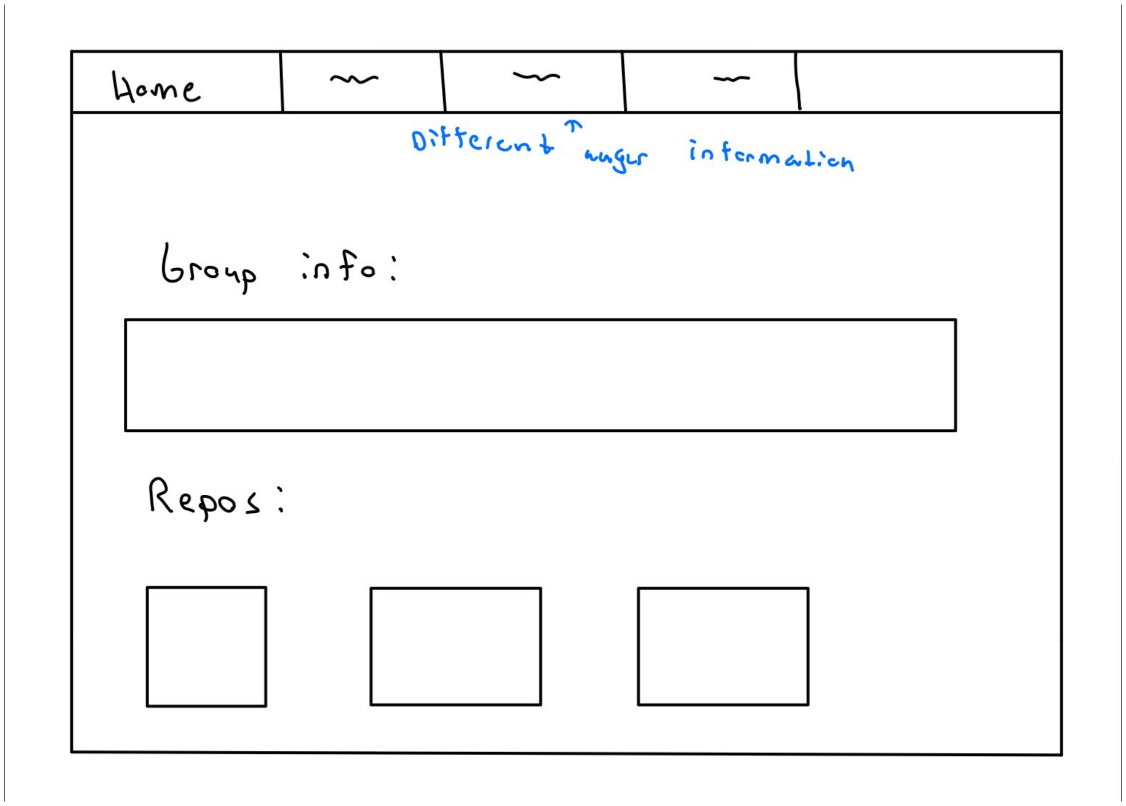
The following drawings are a simple sketch for the layout of a possible user interface implementation. The several different sketches demonstrate the different pages that could make up the project.

Wireframe of home/repository group selection page:



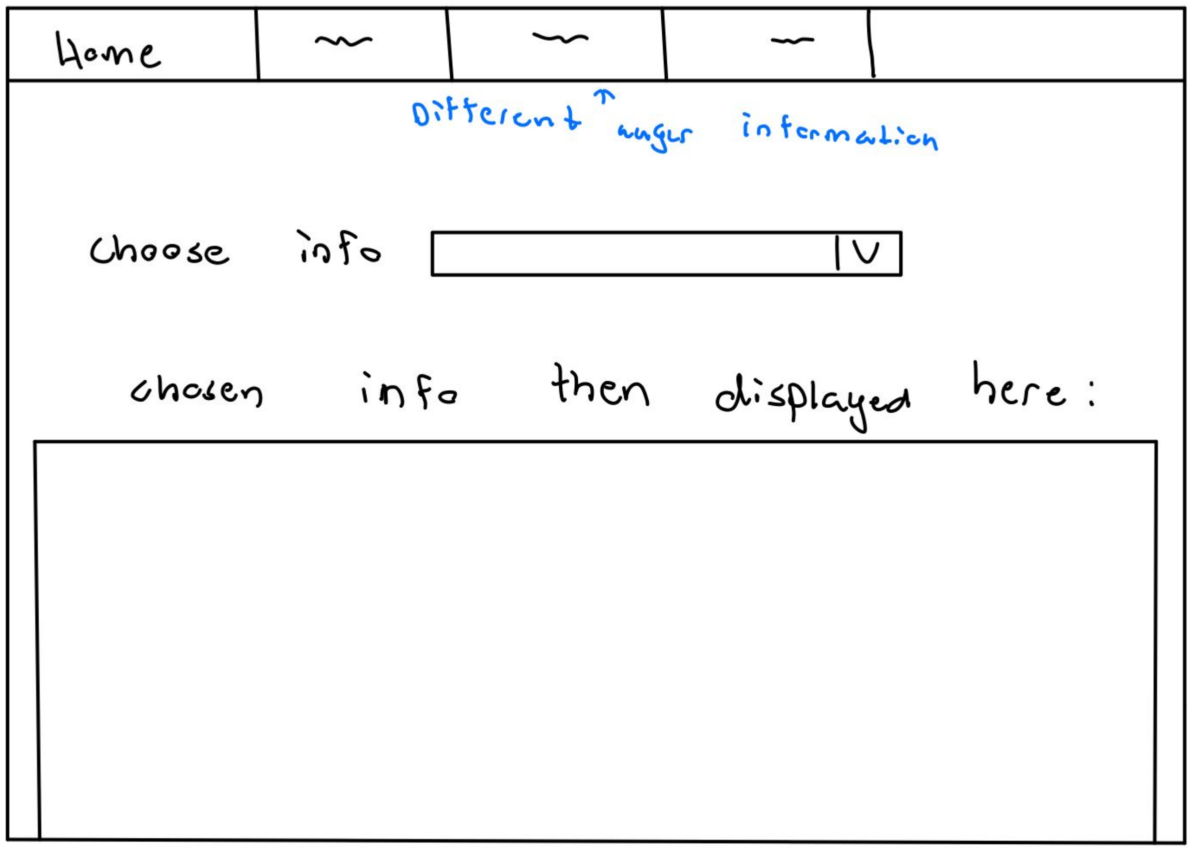
This page will allow you to view the different repository groups that are stored within Augur. By clicking on the different groups, it will navigate you to the repository's within that group. Each card contains details about that group, such as its name or potentially how many repositories are contained within the group. At the top of the page, there could potentially be a navbar that contains other Augur information that is not linked to the repos/repo groups, as well as a link back to this page. This could be visible across all pages.

Wireframe of repository group page:



This page is a sketch of format for the selected repository group. This page could display more specific information about the selected repository group, as well as displaying all of the repositories in the group in a similar way to the previous page. Also similar to the first page, if you click on the repository card, it will navigate you to another page featuring more detailed information about that repository.

Wireframe for the detailed repository page:


Home
~
~
~
Different ↑ augur information
Choose info
|V
chosen info then displayed here:

This page will contain a main portion of the information for the project. Augur is structured in such a way that the endpoints require a repo group and repo to be selected. Once you have selected these two pieces of data, there is a wide variety of endpoint data that can be accessed. On this page, there potentially be a selector for the types of data that are contained in Augur about that particular repo. Once you select something, the page could then be populated by the data associated with that selector from the API endpoint. The method of when/where to display that data could be subject to change.

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.