Semester Project Sprint 1

Fall 2019

Group 5: Jake Alongi, Matt Hudson, Tim Kuehker, Rebecca Parker

Table of Contents

Use Cases

1.) 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.

2.) 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.

3.) 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.

Data Sources

Augur Database:

1) Data selected from key **cntrb_id** from **Contributors** table. Each cntrb_id has a one-to-many commits that have the relevant info needed: cmt_committer_date is the only required data needed and the cntrb_id to query the results.
2) From **repo** table we need the **repo_id** key, from there we can select a count of unique cmt_id keys from commits. If we want to organize by date we need the cmt_author_date as well.
3) Data would be selected from key **repo_id** from the **repo** table which would give us a set of commits tables for that repo, for each **repo_id** in the **commits** table we would match the **cmt_committer_name** with the commits from other repos to see if the same user is committing in different repos. Alternatively we could instead use the **contributors** table to query by **cntrb_id** and count the number of **commit_id** in **commits** affiliated with the **cntrb_id**. We would still have to do this for all repos. **Commits** table has the date of each commit if we wanted to display this on a timeline in cmt_author_date.

Technologies Needed

Flask - Python server framework for serving API endpoints. We chose this over NodeJS, even though some of us have Node experience, we noticed Flask is used in Augur, so we wanted to use the same framework. We will need to look into the tutorials on how to make Flask endpoints, but we have experience in serving JSON.

SQL (the Augur database) - we will need to execute SQL queries on the Augur database in order to grab the data we need for our project.

A Linux Server - these are the ones we got in class. We can use this to serve the endpoints and test our project by deploying the Flask there.

Design Overview / Endpoint Definitions

To get the data:

1) Access the database through an SQL query
2) Get the results back, pack into JSON
3) Send JSON back

Specific to each endpoint:

1) Repo timeline

- This endpoint is given a repo_id, which we can use to search for the commits associated with this repo. Then, we will examine each commit and see when it was committed. We then construct an array of datapoints, like on a line graph, that map dates to numbers of commits. For example:
  {repo_id: <repoid>,
  timeline: [{date: <date 1>, commits: 4}, {date: <date 2>, commits: 5}, etc. etc.]
  }

2) Repo Group Timeline

- This endpoint works on the same principle as [1], but is for an entire repo group rather than a single repo. This endpoint will return timelines for each repo in the group. So, we will find all repos with repo_group = the provided, and return a list of timelines:
- {repo_group_id: <repogroupid>,
  timelines: [{repo_id: <repoid>,
  timeline: [{date: <date 1>, commits: 4}, {date: <date 2>, commits: 5}, etc. etc.]},
  {another timeline here, etc}]
  }

3) Repos/commits per contributor

- For this endpoint, it will be provided the ID of a specific contributor. Then, we will use that ID to find every commit by that user, then return a list of repos the contributor has worked on, as well as how many commits per repo.

Appendix (notes we made during class)

Useful Links
ERD
Create A Metric

3 Metric Endpoints?
1. Individual commits organized in a timeline
2. Group commits with the same idea as (1)
3. How many repos a certain user has/is working on (add # of commits per repo?)

Possible methods:
1) Python server (Flask) (might want to use because tutorials are also in python for SQL)
2) Otherwise, run a node/express server - would need to research how to access SQL in this
   ^^ on this, assess background for team members; what would be best?

(Implement a first pass) - in the backlog, not relevant to this sprint

Need to design 3 use cases,

Document design decisions, plus
- Describing how features will be built from data (so, how these endpoints will take in the data and pack into JSON)
- Clarify requirements

What parts of the ERD are going to be relevant? (i.e. what data will we need to pull in?)

Simple design document (drafts) that specify what functions, etc we're going to need

1) Timeline (points on a graph) -
   - Input: start date, end date (through route parameters or get parameters)
   - Also, need repo id and repo group id
   - Query: 'select <some stuff> from commits where repo_id=provided and repo_group_id = provided'

  - Some stuff = date of commit, user who committed it
  - for each unique date, count number of commits, add number to array of points
  - So for example points = [{date: Nov 6 2018, commits: 6}, {date: Dec 6 2018, commits: 2}, etc. etc.] <= return this?

2) Same as (1), but only search by repo group, get that data instead
3) Input user id, get user id, go into commits, iterate, look for user_id, if so, add the repo to an array, also count number of commits and add that, so
   [{repo: 'rails', commits: '46'}, {repo: 'hive', commits: '2'}] <= return this