Git Statistics (Rachana’s Honors Feature)

Changes/Redesign to feature
As I began to actually implement the Github API into my feature, I saw what kinds of data could easily be retrieved. The Github API has a function that returns two arrays: the number of commits a *user* has put into a project every week for the last year, and the *total* number of commits a group has put into a project. While it is was relatively straightforward to retrieve both arrays of data, it was much more difficult to store the data for individual users in our database.

Before I began connecting with the API, I restructured the git data as requested and placed individual user data within the user object. Each user had an array of statistics objects, and each object contained the number of commits each user put into a project, the x-axis labels and a color for the bars. But I quickly discovered that it was trickier to update the user's data, as data had to be mapped to the stats object that referred to the project being graphed.

Due to time constraints, I have opted to remove user stats entirely and simply display the total contributions that have been made to a project over the last 10 weeks. This version allowed me to create a cleaner feature that used the required entities (server methods, database queries, etc.) and would be mostly functional. I also opted to remove the specific member labels to the group stats for 2 reasons: (a) this would involve another call with the GitHub API and mixing data, which could get complicated (b) If I was working on a project, I would rather not have have every individual contribution I’m making tracked. I know this is the way Github has designed their graphing statistics, but I personally believe it is a feature that portrays too much data.

UI Additions
In each project, I have opted to add some github related fields under the ‘settings’ tab. Not every project is going to require github, so I figured the user should be able to decide if they want to use the functionality or not. If the team members are using a github repo for their project and want to see their effort on display, they simply have to enter the name of the project repo and the owner of the repo into the appropriate fields and save the changes. The repo and owner names are then updated in the project via the ‘project update’ functionality and thus, the graphs can be rendered for the project. Authorization is running on this functionality because only users assigned to the project can actually update the settings of the project.

Http Routes Implemented

Client Side:

Under the Statistics React Component (app/views/statistics/index.js), I created a promise function that would send a server request to the appropriate server method every time a user tries to click on the 'Statistics' page on the webpage. The Github API allows a maximum of 60 calls to its servers every hour, so a user is bound by this number for the number of times they can access the 'Statistics' page every hour. This route is a simple 'get' request as we are asking the server to speak to the github server to update the data.

Server Side:

GET /api/statistics/gitStats

(route lies under src/server/api/statistics/index.js in order to separate my work from the team's)

This route fetches data from the Github servers for the git stats feature. I installed a node github library in order to make coding this server route easier. A variable storing some of the properties that the API request needs is created at the start of the of the router request. Then, through a series of callbacks and database functions, every project has its github repo and repo owner name extracted in order to update its data in the database. Although the API call returns an array with 52 members, I store only the last 10 weeks worth of commits in order to make the display less cluttered. Data is now stored in the database and the stored data is now used to render the graphs.

Mongo DB Queries Implemented

The database queries that I used to implement this feature were relatively simple. I store the db.collection.find() query in a variable in order to then apply an 'each' call on the collection. This query allowed me to examine each project in the collection and extract and update the data I needed. The other primary query that I used was the db.collection.update() query so that I could update the data array for each project each time a call was made to the API. Overall, straightforward but necessary processes.

Outstanding Issues

1)The main issue that my feature has is lack of error checking. If a user inputs the bad data into the repo or owner name into the github connection fields, a generic error is thrown. If they try to access the page too many times in an hour, the user wouldn't particularly understand why. Better error handling would make this feature a little easier to use.

2)There is no way to navigate around a project without a github repo hooked up to it. Currently, when a new project is created, I have a random repo from one of our group members attached to the project. Users have the option to change the repo, but they don't have the option to forgo the stats entirely.

3) Due to issues with github authorization on private repos, users can only input public repositories for statistics. There is a way to work around authorization, but for the sake of general functionality, I opted to forgo authorization for private repos.