Software Design

Git OSS-um

Gary Baker
Van Steinbrenner
Stephen White

February 18, 2019

Project Sponsor: Dr Igor Steinmacher,
Assistant professor, SICCS, Northern Arizona University

Faculty Mentor: Ana Paula Chaves Steinmacher
Version 2.0

Table of Contents

1. Introduction

Open Source Software (OSS) is a type of computer software in which source code is released under a license, where the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose. OSS may be developed in a collaborative public manner using a version control system named Git and hosting the project itself on a site known as GitHub[1]. This version control system tracks changes made to a project and easily allows projects to have multiple contributors. It also allows for a project owner or manager to accept or decline contributions depending on their opinion of the quality of the change through what is called a pull request.

Unfortunately, a substantial amount of developers do not contribute to open source projects. We have all been there: we do not know where to start, what files are important, or what needs to be changed. The community may be friendly and quick to provide assistance to a newcomer, or the contributor may get berated for submitting a pull request that does not adhere to the project's standards. It is so daunting that the newcomer gives up and moves on. This experience shows that there is a need in the open source community for a tool that will allow a novice passing through to quickly and efficiently determine the newcomer "friendliness" of a repository.

Our client, Dr. Igor Steinmacher, is a researcher and assistant professor at Northern Arizona University. Dr. Steinmacher and his colleagues have created a web application called FLOSScoach[2] to address newcomers' needs, where a user can investigate the skills needed to contribute to open source. Although FLOSScoach has been helpful, it has its shortcomings, as listed below.

[1] GitHub: https://github.com/
[2] FLOSScoach: http://www.flosscoach.com

- Data is manually fed.
- Contant manual maintenance.
- Data is not up to date.

Due to these shortcomings, we will be spearheading a solution, independent from FLOSScoach. Our job is aimed at giving power back to the newcomer by creating an autonomous process where relevant data of a requested repository is presented to the user. This will allow them to decide if the specific OSS project will support their needs and help them feel at ease when picking their first open source project.

In order to develop a successful newcomer meter for open source dynamics, we will create a web application that will take a URL leading to a GitHub project page, which will then be accessed via the GitHub API. Desired data such as newcomer acceptance rates and newcomer retention, among various other data points will be collected. This data will be used to assist users in finding projects, and interactive graphics will demonstrate to the newcomer what OSS projects are best for them.

Within this document, we will break down our envisioned implementation of features that meet the requirements associated with our client's vision. We will be discussing our implementation and architectural overviews, modules, interfaces, and finally end with a brief recap of our design. In the following section, we will describe the implementation overview in order to give the "big picture" of our product and the technologies that we plan on using to make this product come to life.

2. Implementation Overview

As stated before, we will be creating a web application to solve the problem of newcomers to OSS being discouraged to start contributing due to their lack of knowledge on

how, as well as the inherent intimidation that comes along with contributing. Our application will retrieve GitHub data for a specified repository and display it to the user in a tabular form, accompanied by around 3-4 interactive graphics that will help the user visualize the data.

The way that this application will work is mostly simple. The user will first request information for a specific repository on our application frontend. The request is sent to the application backend where the data is retrieved from our database and sent back to the front end (most likely via JSON). Then, at the frontend, the utility Plotly takes that data and translates it into a table of raw data points as well as interactive graphics that are all displayed to the newcomer to help them form their own opinion about whether or not the repository is right for their first contribution or not.

Our application will include a few different technologies, but it mainly revolves around the web framework Django. Django is a high-level Python web framework that consists of Python, HTML, CSS, and JavaScript to create both great frontend user interfaces as well as fast and reliable backend functionality. We have chosen for our backend logic as well as our frontend user experience to be done using Django due to our familiarity with Python in general and the existing support and libraries that can be utilized when using Python and Django. Django is a very powerful framework and will allow us to do everything that is needed for this product while doing so with high efficiency and a streamlined development experience.

We will be storing a lot of data for our application in order to provide the newcomer with an efficient and reliable experience. In order to store the data mined from GitHub, we will be using MongoDB due to its high efficiency, ease of access, and its seamless integration with Python and Django. MongoDB is a document based database solution in which the documents consist of JSON-like format. This makes the importing of data from GitHub especially easy since the data is retrieved in JSON format from the API calls to GitHub.

There has been a lot of talk about displaying the data to the user via interactive graphics. This will be done using a utility called Plotly which is a library created for Python that is a toolbox to create graphics based on the mined repositories stored in our database. It will allow us to create dynamic, interactive graphs of our data quickly and elegantly to help give the newcomer an easier time reading the data that is returned for their requested repository.

The last main portion of this product is hosting, this is a web application after all. In order to make this application available to the newcomers of OSS, we will be using a service called Digital Ocean. This service has a lot of features to offer our client in terms of scaling up the application as it gets bigger and also provides first class customer support to make sure that very few problems arise and that when they do, they are taken care of.

Now that we have a general idea of what technologies are at play here, it is time to take a bit more detailed look at the architecture of the product, which will be explained in the following section.

3. Architectural Overview

In the previous section, we discussed implementation overview of Git-OSS-um. We will now be taking a high-level look at the architectural of our service, to better visualize what services will comprise the end product. The core of our software will be written using the Django Python web framework, whilst making use of MongoDB to store and access files that we mine from the GitHub API. While there are many other components, Django will be the driver for all interaction between these components. This is visualized in Figure 1 below.

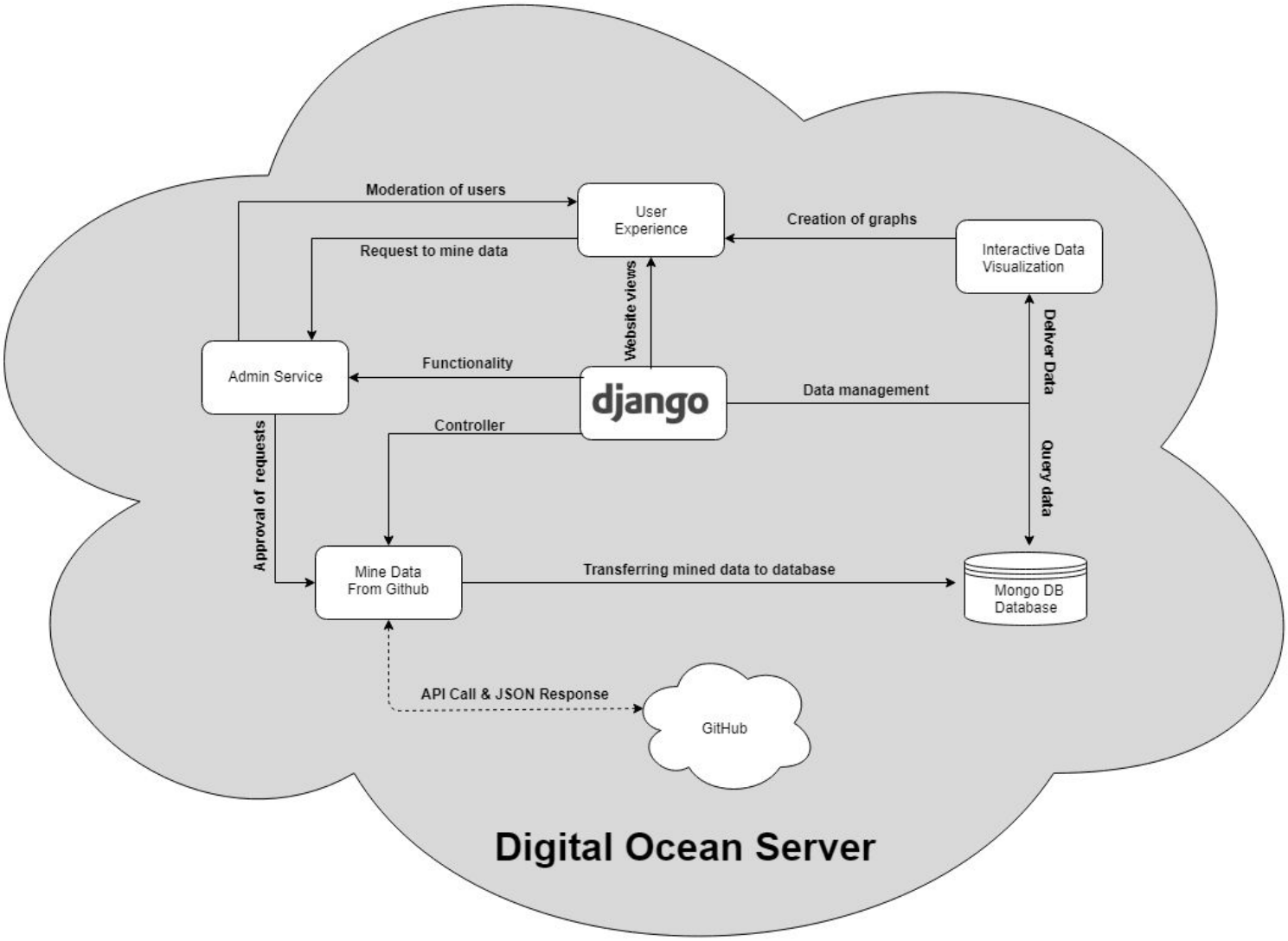

Figure 1: Architectural Overview of System Components

As explained briefly above, Django will be used to design, navigate, and operate all the necessary functions of our application. The team has decided to make use of Django 1.11 as it is a long term support (LTS) version of the framework that will be supported into 2020. It is at

the center of our application, offering such necessities as admin functionality, login security, form validation, and much more.

Our users will make use of the “User Experience” module to create an account, sign into the web application, and access all of the content Git-OSS-um strives to provide. Through this module, a user will be able to submit a request to the administrator to mine data from GitHub, filter repository results, view visualizations of a selected repository, and cross compare up to three repositories at a time in an effort to assist the newcomer in choosing a repository that fits their needs.

The “Administrator Service” module is provided by the Django Web Framework and will contain all necessary functionality for an admin to oversee Git-OSS-um. Within this module, the administrator will be able to manage users who have created and validated their accounts, manage mining requests submitted by the user, enforce a policy for when mined repositories shall be updated, and even delete selected repositories from our system. When a user submits a request to mine data, the administrator may approve selected repositories for mining (therefore placing those repositories in a queue where they will sit until their content is fully mined from the GitHub API), delete a request (allowing the user to submit an identical request for the same repository afterward), or blacklist an irrelevant repository (such as a repository containing textbook PDF’s) therefore preventing the user from requesting that repository in the future.

The “Mining Data from GitHub” module will be triggered by the administrator approving selected repositories for mining. This module is made possible through the use of an open source library known as PyGitHub which provides functionality for making requests to the GitHub API and returning the JSON responses. The three core pieces of its functionality which we will be making use of in our application are it’s OAuth token validation for making up to 1,000

requests per hour to the API, its ability to mine the “landing page” JSON which describes a repository, and it’s ability to retrieve the JSON files of every pull request of a given repository.

Our team selected MongoDB as our database of choice to store and retrieve the data that we mine from the GitHub API. It is a document-based, NoSQL database that is optimized to store and locate large sets of JSON data with minimal hassle for the developer. Using this technology, Django will be able to quickly and effectively query JSON data to build interactive data visualizations for any repository. In order to interact with MongoDB, the team will be making use of the open source library PyMongo, which allows us to store and access the data that we mine in a simple query format through Python.

Lastly, the “Interactive Data Visualization” module will be responsible for creating graphs in which the user can manipulate, explore, and interpret. For this module, we will be making use of the open source library known as Plotly. All information needed to produce the interactive graphics will be queried from our MongoDB database, and passed into Plotly.

All extra functionality that is needed will be written in Python, making it simple to plug into Django. This allows the team to quickly and efficiently write the software needed to mine data from GitHub, and store this data in a MongoDB collection. As stand alone Python code, Django can handle when this code will be used, and who has the access rights to call it. Now with an understanding of the architectural overview of Git-OSS-um, we will dive further into the specific modules that comprise our software.

4. Module and Interface Descriptions

Within the previous section, we discussed the architectural overview of Git-OSS-um, and briefly explained its main components. We will now describe these components in greater detail in order to understand our web application in greater detail.

4.1. Mining Data From GitHub

The mining.py component is accessible only by Django itself, when a the following conditions are met: GitHub data has been successfully validated (it matches a predefined regular expression, the repository exists on GitHub, it has not been blacklisted by the admin, has already been mined, or is currently being mined), the form has been cleaned of any potential security threats (such as HTML, or SQL injection), and the administrator has approved the request. This script is contains functions written by the team that make use of several libraries in order to mine and store JSON data from GitHub, and to send an email notification to the user when this is finished. We make use of PyGithub to quickly and efficiently mine JSON data from GitHub, Pymongo to interact with our MongoDB database collections, and SMTPLib to send email notifications to the user via the Gmail server. An example of a mining request in the form of a SQL model is shown below as Figure 2 and the organization of our mining script is shown as Figure 3.

```
class MiningRequest(models.Model):
    repo_name               = models.CharField(max_length=240, null=False, blank=False)
    requested_by            = models.CharField(max_length=240, null=False, blank=False)
    email                   = models.EmailField(null=False, blank=False)
    send_email              = models.BooleanField()
    timestamp               = models.DateTimeField(auto_now_add=True)
    updated                 = models.DateTimeField(auto_now=True)
```

Figure 2: Mining Request SQL Model

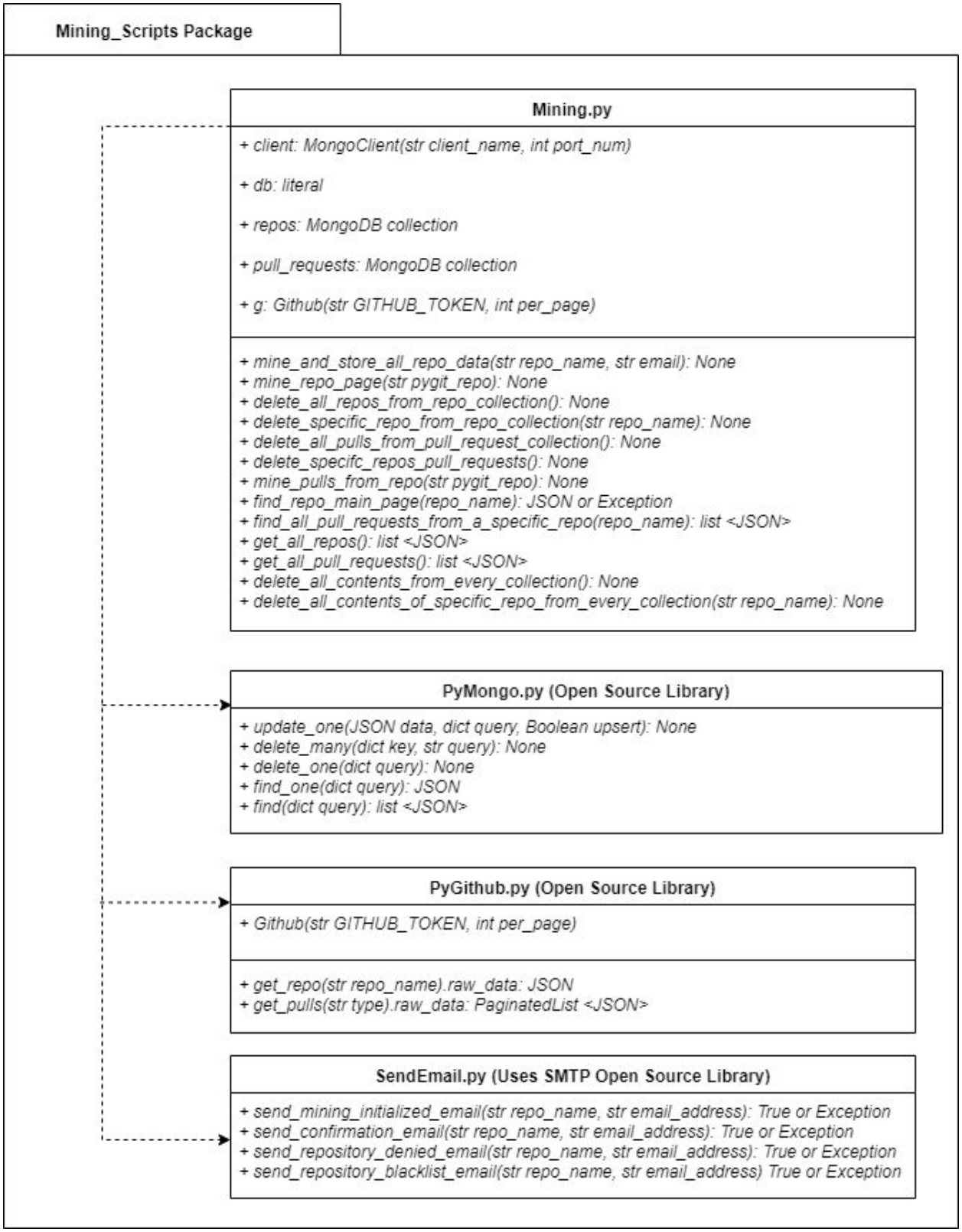

Figure 3: Mining Scripts Package Diagram

4.2. MongoDB

MongoDB will be running as a process on our DigitalOcean server on port 27017, and will never be directly accessible by the user. As mentioned in the previous section, all interaction with MongoDB will be done via the use of Pymongo within the mining.py component of our web application. We have designed our NOSQL database in way that it is simple, intuitive to learn, and accessible by our Python scripts. The database is made up of two separate collections for storing JSON files from GitHub: Repos, and PullRequests. The Repos collection is responsible for containing all JSON files that represent the landing page of a GitHub repository, while the PullRequests collection will contain all pull requests that comprise any repository that we mine. Figure 4 illustrates what the database will look like.

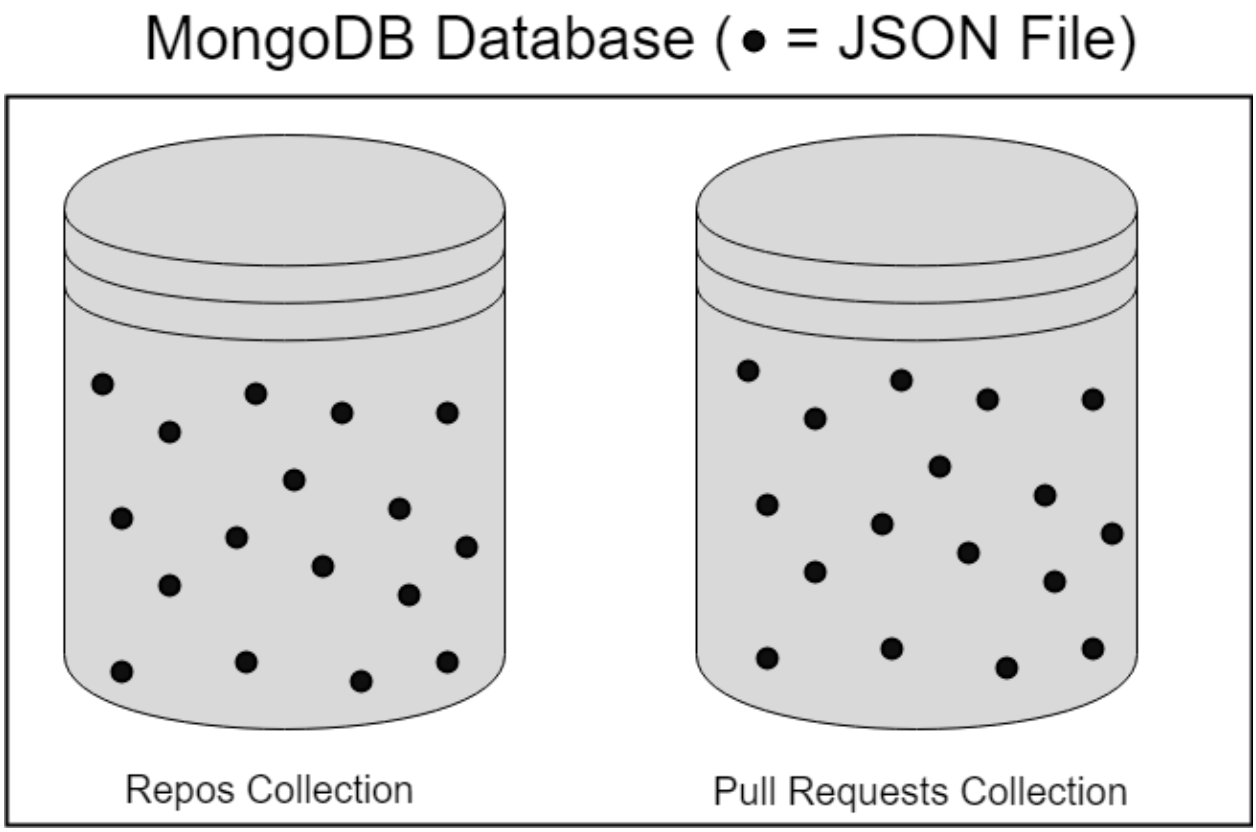

Figure 4: MongoDB Database Overview

It is important to note that Git-OSS-um will not be managing the fields of the JSON data that have been mined from GitHub. Rather, we store the files as they are in the proper collection, and extract what data is necessary for visualization. Figures 5 and 6 display partial examples of actual data that will be stored in our MongoDB database.


```
// 20181110140906
// https://api.github.com/repos/rails/rails

{
  "id": 8514,
  "node_id": "MDEwOlJlcG9zaXRvcnk4NTE0",
  "name": "rails",
  "full_name": "rails/rails",
  "private": false,
  "owner": {
    "login": "rails",
    "id": 4223,
    "node_id": "MDEyOk9yZ2FuaXphdGlvbjQyMjM=",
    "avatar_url": "https://avatars1.githubusercontent.com/u/4223?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/rails",
    "html_url": "https://github.com/rails",
    "followers_url": "https://api.github.com/users/rails/followers",
    "following_url": "https://api.github.com/users/rails/following{/other_user}",
    "gists_url": "https://api.github.com/users/rails/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/rails/starred{/owner}{/repo}",
    "subscriptions_url": "https://api.github.com/users/rails/subscriptions",
    "organizations_url": "https://api.github.com/users/rails/orgs",
    "repos_url": "https://api.github.com/users/rails/repos",
    "events_url": "https://api.github.com/users/rails/events{/privacy}",
    "received_events_url": "https://api.github.com/users/rails/received_events",
    "type": "Organization",
    "site_admin": false
  },
```


Figure 5: Example JSON Representing a Repository Landing Page

```
// 20181110141021
// https://api.github.com/repos/rails/rails/pulls/9523

{
  "url": "https://api.github.com/repos/rails/rails/pulls/9523",
  "id": 4411988,
  "node_id": "MDExOlB1bGxSZXF1ZXN0NDQxMTk4OA==",
  "html_url": "https://github.com/rails/rails/pull/9523",
  "diff_url": "https://github.com/rails/rails/pull/9523.diff",
  "patch_url": "https://github.com/rails/rails/pull/9523.patch",
  "issue_url": "https://api.github.com/repos/rails/rails/issues/9523",
  "number": 9523,
  "state": "closed",
  "locked": false,
  "title": "ActiveSupport::Notifications::Instrumenter#instrument should yield",
  "user": {
    "login": "stopdropandrew",
    "id": 6681,
    "node_id": "MDQ6VXNlcjY2ODE=",
    "avatar_url": "https://avatars1.githubusercontent.com/u/6681?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/stopdropandrew",
```


Figure 6: Example JSON Representing a Pull Request

4.3. Front-end Components with Django

We are implementing a full Django web application, which condenses our web application into one centralized package, instead of having separate front and back-end applications. Below are the components of our front-end with Django and the functionality of the components.

4.3.1. Logging Into The Application

Users will be able to sign in to our application, however, it is not necessary for browsing data. Users are encouraged to create an account so that they can post mining requests to administrators. For user sign up and login to our application, we will use Django forms. Django comes with built-in form functionality so we can build forms with ease and less code. Our login form asks users to input their username and password, we have built error handling to check our database whether a username has been registered to our product. For our sign up form for users, we ask for their name and email as a required field and their GitHub Oauth token as an optional field. If a user wants to request mining data, then they would have to register their Oauth token to mine the data.

```
81  class LoginForm(forms.Form):
82      username = forms.CharField(max_length=30, required=True)
83      raw_password = forms.CharField(widget=forms.PasswordInput())
84
85      def clean_username(self):
86          username = self.cleaned_data['username']
87          users = list(User.objects.values_list('username', flat=True))
88          if not username in users:
89              raise ValidationError("Incorrect username.")
90
91          return username
```

Figure 7: User Login Form Model

```
class SignUpForm(UserCreationForm):
    first_name = forms.CharField(max_length=30, required=True, help_text='*Required.')
    last_name = forms.CharField(max_length=30, required=True, help_text='*Required.')
    email = forms.EmailField(max_length=254, required=True, help_text='*Required.')
    github_oauth = forms.CharField(max_length=254, required=False, help_text="*Optional.")
```

Figure 8: User Sign Up Form Model

4.3.2. Admin Portal

One of the requirements of our web application is to have a admin portal where administrators can approve or decline mining requests made by users and to manage database collections. Django offers a template interface, but can be altered and edited to accommodate our needs. The main functionality of our admin portal is a database management system where administrators can manage data in our MongoDB collections and to accept or decline mining requests made by the user. This central interface will show the administrators what requests are pending the mined JSON data that represents a repository in GitHub. This interface will also allow administrators to implement custom updates to the database, which means that an admin can see if a JSON file is up-to-date with the information that resides within it, and update that file as needed.

```
139  # Register all of the admin panels to their respective models
140  admin.site.register(MiningRequest, MiningRequestAdmin)
141  admin.site.register(QueuedMiningRequest, QueuedMiningRequestAdmin)
142  admin.site.register(BlacklistedMiningRequest, BlacklistedMiningRequestAdmin)
143  admin.site.register(MinedRepo, MinedRepoAdmin)
144  admin.site.register(OAuthToken, OAuthTokenAdmin)
```

Figure 9: Admin Portal Registers Model

4.3.3. About Us Page

As a landing page, we will be creating an “About Us” page that would describe the product and team. This page will describe our web application and the core functionality and who the development team are. This page will briefly describe the main functionality such as viewing of mined data of a repository with interactive visualizations to explore the data. Alongside this, users can also cross compare up to three different repositories to further explore the data. A description of how to get a specific repository data will be explained on the page as

well to clarify the process to new users. The “About Us” page will briefly introduce the development team as well to give users a background of who developed the application. The “About Us” page will briefly introduce the team’s sponsor, Igor Steinmacher, and explain how the project would be used for his research. Figure 10 below is what the About Us page looks like at this moment, it is subject to change as the product moves forward.

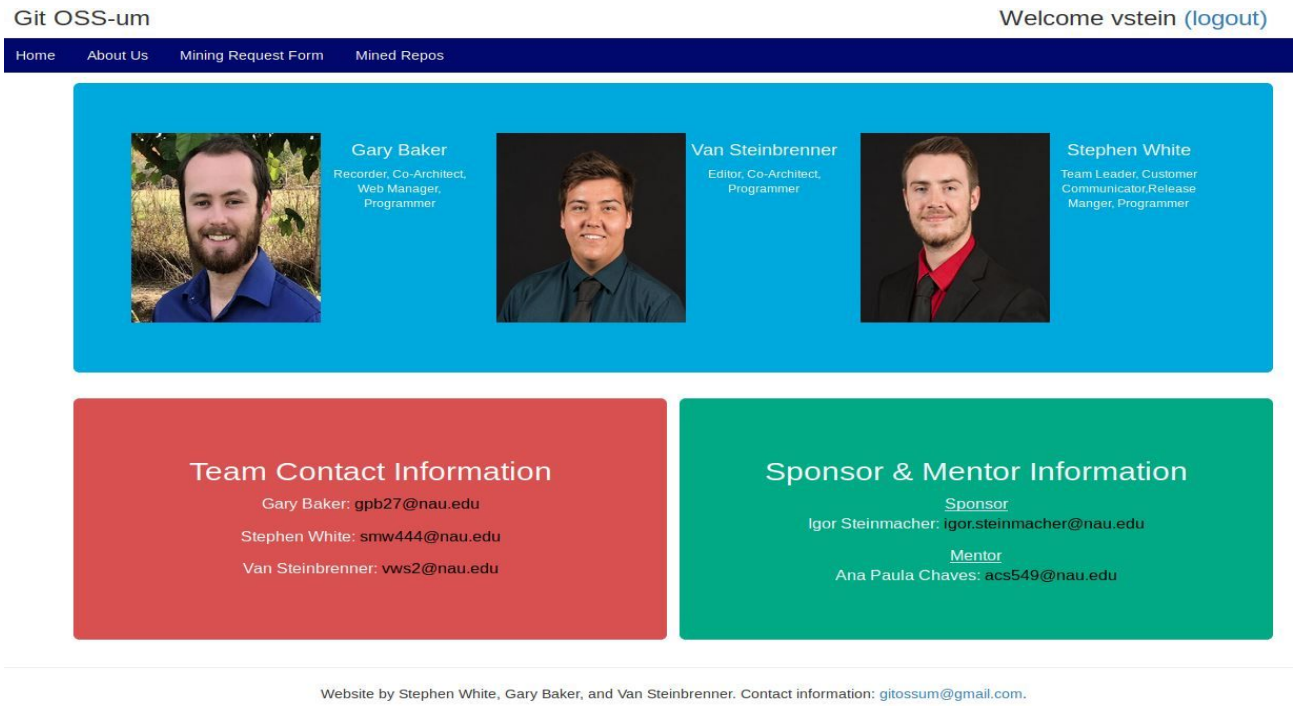

Figure 10: About Us Page Model

4.3.4. Requesting Data

In order to request data from a specific repository, a user will have to fill out a form with specific fields. Two main parts of this form would be the repository name and a user’s email address so that they can get a message saying that their request of that repository has been sent. This form will be able to be accessed in our Django admin portal, where administrators can either accept or decline the request for mining. After a decision is made, a user will get an email notification of the decision. If the request has been approved, a user’s Oauth token will be

used to mine the data. After the data is mined, user's can see the data and interact with visualizations.

```
class MiningRequestForm(forms.Form):
    repo_name               = forms.CharField(max_length=120, required=True, label="Repository",
                              widget= forms.TextInput(attrs={'placeholder':'owner/repository'}))
    email                   = forms.BooleanField(required=False,
                                                 label="Send Me Email Notifications About This Request")
```

Figure 11: Mining Request Form

4.3.5. Data Visualization

The Plotly framework is an open source Python framework for data visualization. We will be using Plotly along with other Python frameworks such as Numpy and Pandas to quickly iterate through our large data sets that we have mined. With Plotly, we can create interactive data visualizations that allow our users to explore certain data points within a visualization. We will also have multiple graphics to show different parts of a repository to further enlighten the information. In the snippet below, we have created a Plotly chart based off of the number of pull requests over a span of time. We separate requests based on "Closed-Merged", "Closed-Unmerged", and "Open" on the x- axis of our visualization and the value of each of these values represented on the y-axis. This snippet shows the simplest form of a visualization, as time goes on, we will have more informative visualizations that explore information more in depth.

```
80  def pull_request_charts(repo_name):
81      mined_repo_sql_obj = MinedRepo.objects.get(repo_name=repo_name)
82      num_closed_merged_pulls = getattr(mined_repo_sql_obj, 'num_closed_merged_pulls')
83      num_closed_unmerged_pulls = getattr(mined_repo_sql_obj, 'num_closed_unmerged_pulls')
84      num_open_pulls = getattr(mined_repo_sql_obj, 'num_open_pulls')
85
86      trace2 = go.Bar(
87          x=["Closed-Merged", "Closed-Unmerged", "Open"],
88          y=[num_closed_merged_pulls, num_closed_unmerged_pulls, num_open_pulls],
89          name='Pulls Bar Chart',
90          marker=dict(
91              color=['rgba(255,0,0,1)',
92                     'rgba(0,94,255,1)',
93                     'rgba(8,154,105,1)']
94          )
```

Figure 12: Data Visualization Snippet with Plotly

4.3.6. Cross Comparison of Repositories

The ability to compare up to three different repositories is a feature that can be used to further show what repository may be the best for the user to contribute to. The comparison will display both data and visualizations to further show users the differences of each repository. Overall, the cross comparisons of repositories and the visualizations will provide greater clarity to users.

4.4. Plotly Framework

Plotly is our main framework for data visualization in our product, because we are using Django as our web framework, a similar Python framework such as Plotly would make sense to use. We can import our data for visualizations from our database as a SQL object and create attributes to place in our charts. Once the Python scripts have been created, Plotly charts can return HTML tags which means that we can place the outputs directly into our HTML template files. The output of HTML tags from the Plotly charts allow different iterations of each chart, which means that we can use the same data from different repositories but use the same charts for interactivity and data exploration. Our charts with Plotly will be placed in our visualizations.py

file, generated with the views.py file, and be displayed in our mined_repo_display.html file. Overall, Plotly will be used to create interactive visualizations and will be generated and displayed with the assistance of our Django web framework.

5. Implementation Plan

Beginning the Spring semester of 2019 in mid January, the team had a working prototype as a proof of concept in which checked off the core functionality of the product outlined in the Requirements Specification document, signed by our team lead as well as our client.

As one of the most important tasks in the early stages of implementation, the team went about creating a checklist of all features and major milestones that needed to be met along with tentative due dates and team members responsible for each task. This checklist will be given to the client as well as a way to outline exactly what the team will be doing and what to expect by the final product deployment.

The checklist is dated up to the last week before Spring Break, in which we plan to have not just the alpha build, but a mostly finished product in order to utilize the remaining portion of the semester for testing of the application and preparation for the final release of the product in which the newcomers can use. The following image displays our checklist featuring all of the tasks and major milestones with their due dates and responsible team member. This checklist may be subject to change but is a mostly complete timeline of the entire implementation period.

Done	Task Description	Week Due	Who is Working on it?
	Django About Us page and mock-ups for the rest.	1/29	Gary, Van
	Mining.py script that will mine information from github and interact with MongoDB via python 3.6	1/29	Stephen
	Mining_tests.py to ensure the quality/integrity of the code is high. This will be used to simulate interacting with the script, so when we move to RESTful practices, we already know the functionality works as intended	1/29	Stephen
	Software Design Document outline	1/29	All members
	Software Design Document Rough	2/5	All members
	python-nvd3 Bar Graph and Line Graph Prototypes (at least 1 or 2)	2/5	Van
	Software Design Document Final	2/12	All members
	All front end pages layout complete with navigation	2/12	Gary
	Post requests for Mining (Front End)	2/19	Gary, Stephen
	Admin Portal layout completed	2/19	Gary, Van
	Connect Application to https://gitossum.com domain which will be actually at https://138.68.40.43:PORT_NUM	2/19	Stephen, Gary
	Admin Portal completed	2/26	Gary, Van
	Post request data visualized with python-nvd3 on front end	2/26	All members
	Design review II Presentation	2/28	All members
	Tech Demo (Alpha) before Spring Break	3/12	All members

Figure 13: Team checklist for tasks leading up to Spring Break

6. Conclusion

Open source software is not the easiest field to be involved in, with our product, we wish to make the contribution process easier to newcomers. Our web application will be able to mine data straight from GitHub, store the data in a database for future use, and visualize that data in the form of interactive graphics which are displayed to the newcomer. In this document we have discussed the “big picture” associated with our application, implementation and architectural overviews of the product, the modules and interfaces that will be used to implement our solution, and a detailed plan of attack that maps out all tasks and milestones that must be

completed by mid-March in order to deliver a mostly complete product early to leave excess time for testing as preparation for deployment to the newcomers of OSS.

By the time spring break arrives, we are confident that we will have a minimum viable product ready to present to our sponsor as both a research tool and a product to assist newcomers into contributing to open source software. As described in our document, we will create a web application that will mine data from GitHub repositories and provide the data and visualizations to our users. Our product will deliver satisfaction to the user, which will make the decisions for newcomers and the contribution process easier in open source.