Proposed Method

There are three main phases to this project, data collection and processing, and data visualization.

Data collection and processing

Available on the internet are vast sources of GitHub data, we considered the following sources because they provided data that was a good fit to what we were aiming to achieve.

- GitHub's API [13] provides per-user information (such as number of followers, number of repositories, avatar)
- GitHub's Archive [9] provides a drop of the data easily processed (however missing some required details).
- Google BigQuery [7] provides a high-level query capability atop that data.
- GHTorrent [8] another aggregation of GitHub's data, perhaps in a more complete form.
- ClickHouse query interface for GH data [1] another aggregation of GH data with fast response time in a web UI.

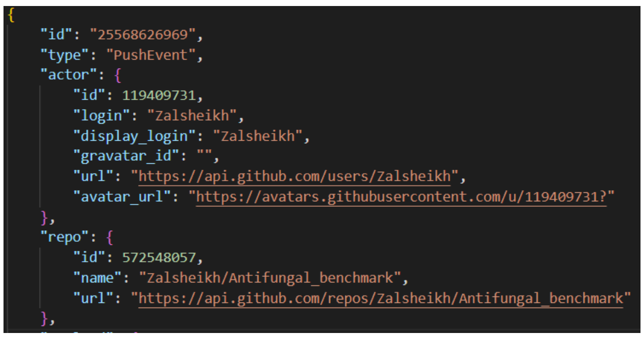
We settled on three data sources,

1. Google BigQuery (BQ) [7] for exploratory data analysis
2. GitHub's API [13] for getting additional information about a repository (repo description, repo topics, repo languages, number of watchers, forks, stars) or actor/user (location).
3. GitHub's Archive (GHA) [9] was selected mainly for the following reasons.
    - getting data from GHA was for some reason efficient than getting it from BQ.
    - GHA data is packaged in hourly files e.g data for the 15th hour of the 26th of November 2022 will can be extracted from a url with the following format https://data.gharchive.org/2022-11-26-15.json.gz .
    - downloading each hourly file was easily done using python.
    - All the relevant fields/ columns we needed were available in the hourly files with the same JSON format. The fields were as follows: type, repo, and actor. See Table Schema in Figure 1. below.

Table 1. Repository JSON Field Descriptions from Google BigQuery GitHubArchive

Field	Type	Description
id	INTEGER	Repository id
name	STRING	Repository Name
url	STRING	Repository Url

Figure 1. GitHub Archieve Sample record



```
{
    "id": "25568626969",
    "type": "PushEvent",
    "actor": {
        "id": 119409731,
        "login": "Zalsheikh",
        "display_login": "Zalsheikh",
        "gravatar_id": "",
        "url": "https://api.github.com/users/Zalsheikh",
        "avatar_url": "https://avatars.githubusercontent.com/u/119409731?"
    },
    "repo": {
        "id": 572548057,
        "name": "Zalsheikh/Antifungal_benchmark",
        "url": "https://api.github.com/repos/Zalsheikh/Antifungal_benchmark"
    },
```

Due to the total amount of data approximately 5.3 billion event records at the time of writing of this report. We decided to only focus on the PushEvent event Type which constitutes over half of all events that are in the Google BigQuery (BQ) [7] database. Table 3. below shows the count of records grouped by Event Type.

Table 2. User/ Actor JSON Field Descriptions from Google BigQuery GitHubArchive

Field	Type	Description
id	INTEGER	User id
login	STRING	User Login name
gravatar_id	STRING	Avatar id of the user
avatar_url	STRING	Avatar Url of the user
url	STRING	Url of the user

Table 3. Top 10 Events count data by Event Type

Event type	count(*)	Min CreatedDate	Max CreatedDate
PushEvent	2 694 516 446	2011-02-12	2022-11-26
CreateEvent	773 625 784	2011-02-12	2022-11-26
PullRequestEvent	423 407 350	2011-02-12	2022-11-26
IssueCommentEvent	336 755 872	2011-04-08	2022-11-26
WatchEvent	332 008 470	2011-02-12	2022-11-26
DeleteEvent	159 868 794	2011-02-12	2022-11-26
IssuesEvent	159 026 390	2011-02-12	2022-11-26
ForkEvent	120 002 813	2011-02-12	2022-11-26
PullRequestReviewCommentEvent	88 484 272	2012-02-15	2022-11-26
PullRequestReviewEvent	62 426 983	2020-08-18	2022-11-26

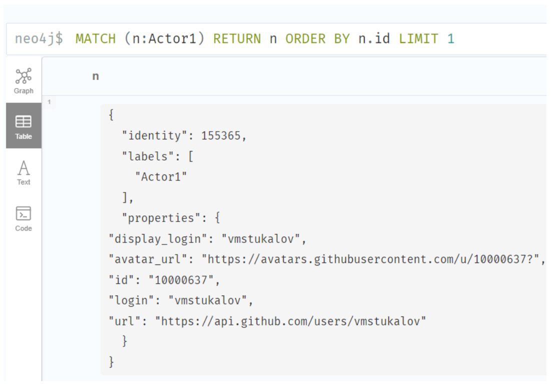
After identifying and getting the records, we wanted to get from GHA, we decided to use the Neo4j [18] database as our final storage database because it is specifically designed for storing and modelling graph-related data which made it relatively easier to Implement the shortest path algorithm when compared to doing the same in relational databases.

There are different ways of working with Neo4j databases, we opted to use a cloud-based Neo4j Aura graph database hosted on the Neo4j cloud service https://neo4j.io

We started going over the go over the events data and extracted the interaction details which allowed us to progressively build the social graph.

For each record we extracted, we noted the user/ actor and inserted this into the database as an Actor1 Node, see Figure 2 above, we also extracted the repository the action was performed on and inserted this into the database as a Repo1 Node, see Figure 3. then we created a TO relationship between the Actor1 and Repo1 nodes, see Figure 4. For example, in a push event, we noted the user who performed the action together with additional properties like login, URL, and id and the name, id and URL of the repository they performed the action.

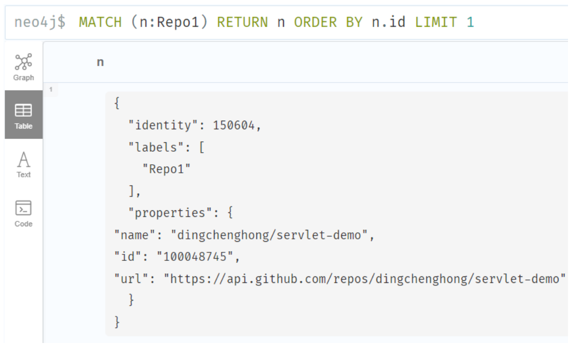
Figure 2. Single user/ Actor record in the Neo4j database



```
neo4j$ MATCH (n:Actor1) RETURN n ORDER BY n.id LIMIT 1
```

```
{
  "identity": 155365,
  "labels": [
    "Actor1"
  ],
  "properties": {
"display_login": "vmstukalov",
"avatar_url": "https://avatars.githubusercontent.com/u/10000637?",
"id": "10000637",
"login": "vmstukalov",
"url": "https://api.github.com/users/vmstukalov"
  }
}
```

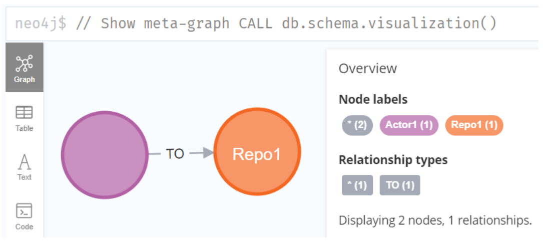
Figure 3. Single Repository record in the Neo4j database



```
neo4j$ MATCH (n:Repo1) RETURN n ORDER BY n.id LIMIT 1
```

```
{
  "identity": 150604,
  "labels": [
    "Repo1"
  ],
  "properties": {
"name": "dingchenghong/servlet-demo",
"id": "100048745",
"url": "https://api.github.com/repos/dingchenghong/servlet-demo"
  }
}
```

Figure 4. User/ Actor to Repository relationship in the Neo4j database



```
neo4j$ // Show meta-graph CALL db.schema.visualization()
```

Overview

Node labels

* (2) Actor1 (1) Repo1 (1)

Relationship types

* (1) TO (1)

Displaying 2 nodes, 1 relationships.

Visualization

For the visualization UI, we decided to use modern front-end technologies. We created a React and D3 web-app written in javascript using vite and chakra-ui.

The UI allows for searching GitHub users and observing the shortest path between them

It then provides for expanding the user or the repository nodes to view their immediate neighbors and that allows for interactive discovery.

On the side, there is an extended details pane for the currently selected user in the graph. See Figure 5. As an example,

A GitHub-login option is available for two purposes.

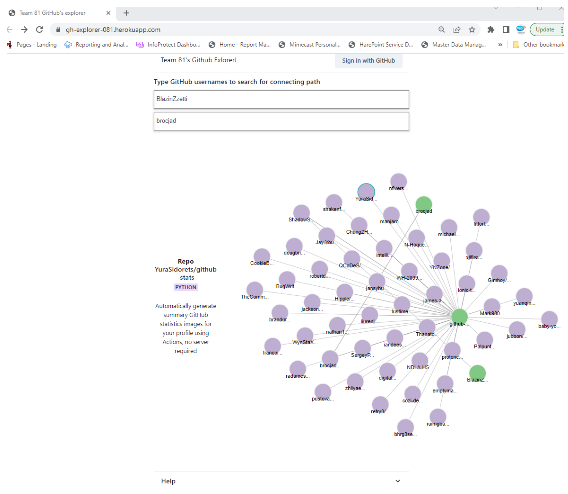
1. For the collection of sidebar details we use GH's rest API. This API is throttled after multiple calls, so logging in with a GH user allows for extra API credits.
2. Our plan was to show the currently logged-in user by default. This is possible after logging in, however, due to the sheer size of the data and processing time we decided not to load all the data, therefore, it is quite possible that the currently logged-in user is not in our database so we decided to disable this feature.

The backend is written in python and hosted on Heroku. This allows for querying into the neo4j database, connecting to GitHub via the APIs to get additional information, and implementing the Oauth2 GH login.

It is implemented using Flask. Interactive discovery is enabled in two ways.

1. by means of typeahead auto-completion which makes it simple and intuitive to look for GH users for finding the shortest path between them. (For example such as in Figure 6)
2. by clicking on nodes to expand them and iteratively discover more neighbors. (For example such as in Figure 5)

Figure 5. Sample screenshot from the User Interface



Team 81 GitHub's explorer

gh-explorer-081.herokuapp.com

Team 81's Github Exlorer!

Type GitHub usernames to search for connecting path

BlazinZzetti

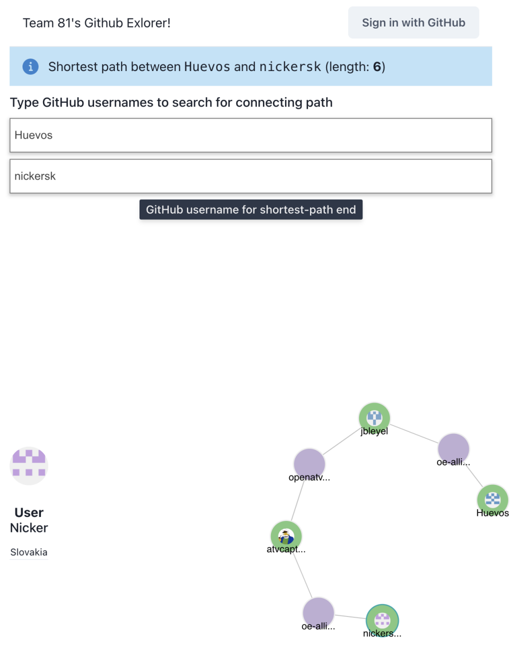
brocjad

Repo
YuraSidorets/github-stats
PYTHON

Automatically generate summary GitHub statistics images for your profile using Actions, no server required

Help

Figure 6. Example shortest path between two users



Team 81's Github Exlorer!

Shortest path between Huevos and `nickersk` (length: **6**)

Type GitHub usernames to search for connecting path

Huevos

nickersk

GitHub username for shortest-path end

User
Nicker
Slovakia

Limitations

Due to the amount of data, currently GitHub has over 53 million users and over 270 million repositories, this means that we would need to load at least 330 million nodes into the Neo4j database. Currently, our sample Neo4j database has about 180k nodes and 114 relationships consuming about 55Mb of storage, so 330 million nodes would require over 100 Gb. The cloud-based Neo4j Aura Db only allows for a maximum node count of 200k and 400k relationships/edges.

Experiments/ Evaluation

[5%] Description of your testbed; list of questions your experiments are designed to answer

[25%] Detailed description of the experiments; observations (as many as you can!)

Conclusions and discussion

[-5% if not included] Provide a statement that summarizes the distribution of team members' effort. The summary statement can be as simple as "all team members have contributed a similar amount of effort". If effort distribution is too uneven, we may assign higher scores to members who have contributed more.