

# Github Repository Analysis Tool

## User Documentation

**Title of the Software:** Github Repository Analysis Tool

**Version Number:** 1.0

**Date:** 17 June 2024

**Authors/Contributors:** Lasse Kuipers, Daniel Danilin, Eduardo Calvo, Teun de Waal, Thijs Berende, Eloy Wijlhuizen, Marie Oldeman and Maurits van 't Hag

**Group:** 7

# Table of Contents

<a href="#">Table of Contents</a>	<a href="#">2</a>
<a href="#">Introduction</a>	<a href="#">3</a>
<a href="#">Overview of the software</a>	<a href="#">3</a>
<a href="#">Purpose and objectives</a>	<a href="#">3</a>
<a href="#">Scope</a>	<a href="#">3</a>
<a href="#">Getting Started</a>	<a href="#">4</a>
<a href="#">Installation Instructions</a>	<a href="#">4</a>
<a href="#">Starting up the Backend</a>	<a href="#">4</a>
<a href="#">Starting up the Frontend</a>	<a href="#">5</a>
<a href="#">System Requirements</a>	<a href="#">5</a>
<a href="#">User Guide</a>	<a href="#">5</a>
<a href="#">Detailed usage instructions</a>	<a href="#">5</a>
<a href="#">General Navigation</a>	<a href="#">5</a>
<a href="#">Using Specific Features</a>	<a href="#">5</a>
<a href="#">Developer Guide</a>	<a href="#">8</a>
<a href="#">Architecture Overview</a>	<a href="#">8</a>
<a href="#">System Components</a>	<a href="#">8</a>
<a href="#">Data Flow</a>	<a href="#">9</a>
<a href="#">API Documentation</a>	<a href="#">9</a>
<a href="#">Overview</a>	<a href="#">9</a>
<a href="#">Authentication and Authorization</a>	<a href="#">9</a>
<a href="#">Configuration</a>	<a href="#">10</a>
<a href="#">Security Settings</a>	<a href="#">10</a>
<a href="#">FAQ</a>	<a href="#">10</a>
<a href="#">Common Issues and Solutions</a>	<a href="#">10</a>
<a href="#">Frontend required installments</a>	<a href="#">10</a>
<a href="#">Python Virtual Environment</a>	<a href="#">10</a>
<a href="#">Appendices</a>	<a href="#">12</a>
<a href="#">References</a>	<a href="#">12</a>

# Introduction

## Overview of the software

Our client is a software developer that coordinates teams working on different projects. Over the years, it has become evident that Git repositories, and a team's ability to use them, is essential for the effective development of software projects. Hence, the problem of low quality comments, commit messages and engagement issues within a team is something that must be tackled. Our client, therefore, designed the task of creating a program that analyses GitHub repositories and measures the quality of a team's commit messages, pull request messages, and comments, as well as their level of engagement, so that it can be reflected upon and improved. The potential outcome is that GitHub repositories will be more valuable and software project managers, such as our client, can get a better overview of how well a team, or project, is developing, and have a clearer idea for how to help them. The software is intended for users who wish to improve their use of Github or to track how other users engage with a project. In both cases it can be used as an evaluation tool.

## Purpose and objectives

The purpose of the software is to give insights into the quantitative and qualitative aspects of a repository. The software evaluates this per repository, pull request, commit, comment, and user, using updated data. The objective that the software hopes to achieve is to analyse a repository as a whole and shed light on multiple perspectives in a repository. With this information the project manager (our client) can develop feedback accordingly, using not only their own view but an objective holistic one.

## Scope

This document provides comprehensive guidance on installing, configuring, and using the Github Repository Analysis Tool Software version 1.0. It is intended for end-users and system administrators who need to set up and use the software.

- **Included:**
  - Step-by-step installation instructions
  - Basic and advanced configuration options
  - User interface walkthrough
  - Common troubleshooting tips
- **Excluded:**
  - Detailed system architecture
  - Integration with third-party software not directly supported by the Software
  - In-depth programming API documentation
- **Assumptions:**
  - Readers have basic knowledge of CLI operations.

- Users are familiar with fundamental concepts of the architecture of Github Repositories.

# Getting Started

## Installation Instructions

Open the .zip file in a Source Code Editor. The developers of the software used Visual Studio Code.

## Starting up the Backend

1. For calculating the lexical density and the Flesch reading ease metrics, a few additional dependencies are required to be installed. For proper functioning of the code, it is necessary to locally install the NLTK Python (Natural Language Toolkit) library. In addition to installing the NLTK library and importing it, the following additional NLTK dependencies must be installed:

- `nltk.download('punkt')`
- `nltk.download('averaged_perceptron_tagger')`

The reasons for installing these dependencies are provided in the documentation of the code.

2. Add your own access token in the code in the following variable: `GITHUB_PERSONAL_ACCESS_TOKEN`, This token can be generated when created your GitHub account
3. Create a virtual environment, this is necessary for the code to run as it should.
4. Ctrl+Shift+P
5. Search “Python: Create Environment”
6. Click it, and follow the instructions (select ‘Venv’). You will see it will also ask you to select our requirements.txt file.
7. It will take some time installing stuff, but done! You will see the .venv folder in our root folder, and also that its name is greyed out because it’s not being tracked by git.

In the case that this does not work, see section FAQ for troubleshooting.

8. Now that you have created and started the virtual environment run the following lines in the terminal:
9. `cd backend`
10. `cd api_project`
11. `cd python manage.py runserver`
12. Now the backend should be running.

## Starting up the Frontend

13. Open a new terminal.
14. Run the following lines in the terminal:
15. `cd frontend`
16. `npm install`
17. `npm run dev`
18. Ctrl-click on the link of the local host. This should open a window in your default browser.
19. Now you can use the software using the window in your browser.
20. To exit the program ctrl-c in both terminals - the order does not matter - and then follow the instructions if relevant.

## System Requirements

To be able to run the program Python should be installed, then the virtual environment can be created. This can be done through the left hand toolbar> Extensions> Python.

## User Guide

### Detailed usage instructions

#### General Navigation

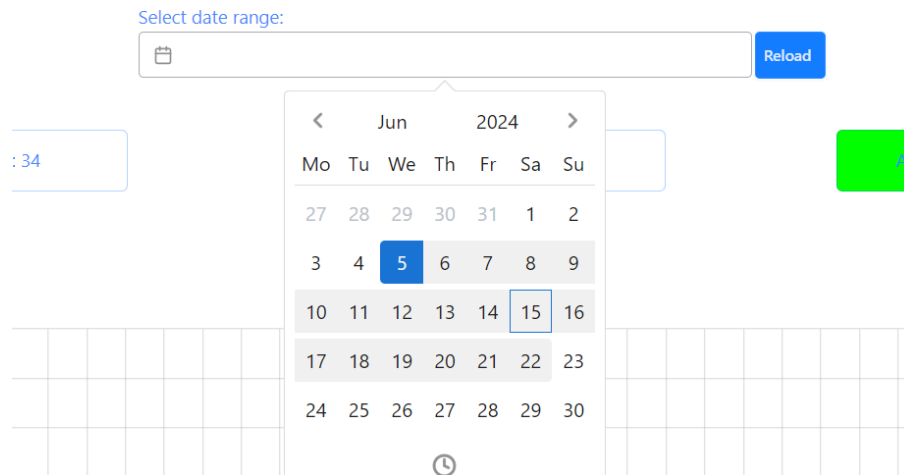
The software requires a correct username and password to log in. After these have been entered, verified and accepted you will be led to the Home Page. If this is your first time using the software then you must insert the link of the Github repository that you wish to analyse. If you have used the software before then you can follow the previous step or select a previously visited repository. A repository selection is required to view the *Repository Information*, *Pull Request*, *Commit*, and *User page*. Navigation is done through buttons on the pages. For instance, clicking on a Pull Request on the *Repository Information* page will lead you to the *Pull Request* page and clicking on a contributor on the *Pull Request* page will lead you to the *User* page. To go back to the previous page the blue button on the bottom left of each page with the text “Back” must be clicked. To navigate back to the *Home Page* the white button on the top left of each page with the text “Home” must be clicked.

#### Using Specific Features

##### **Date Range Picker**

On the *Repository Information Page* you can choose to select a date range. By selecting a certain start and end date you request data from the repository within only that range. To enter a range first select the start date and then the end date, then click on reload to update the data.

By being able to view information of a repository during a specific date range, less active months can for example be excluded from the overall data.



*Figure 1: Screenshot of software: Picking a date range.*

### **Semantic Score Colour**

The semantic score quantifies the informativeness of commit messages and comments. Per repository there exists an average general semantic score. This score also exists per pull request, commit, comment and per user. The semantic score consists of three components namely: Lexical density, Flesch reading ease and commit message length/code length ratio. All these values range from zero to 100%.

The semantic score per repository, pull request and author are computed as an average semantic score over all messages in the respective domains.

In order to easily visualise the quality of a semantic score its colour changes dynamically. As the score gets better it appears more green. As can be seen in *Figure 2* the semantic score is a relatively even blend between orange and green.



*Figure 2: Screenshot of software: Colour of an acceptable semantic score.*

### **Adjusting Semantic Score Weights**

The semantic score in this software is designed to include weights for the constituent metrics in order to grant you the possibility to choose which metrics are more important. By default, the weights in *comment* messages are distributed as:

$$0.5(\textit{Lexical Density}) + 0.5(\textit{Flesch Reading Ease})$$

In *commit* messages, the weights are by default distributed as:

$$0.33(\textit{Lexical Density}) + 0.33(\textit{Flesch Reading Ease}) + 0.34(\textit{commit message length/code length ratio})$$

To adjust the weights as required, please locate the `comment_task` and `commit_task` functions within the `API_call_information.py` file in the backend part of the web app. Within these functions, the `comment_semantic_score` and the `commit_semantic_score` variables are responsible for handling the calculation of comments and commits respectively.

As can be seen in the formula, the weight for the commit message length/code length ratio is larger than 0. It must be noted that for smaller repos, the web app functions as expected when this metric is part of the semantic score. For larger repos, it might be the case that the number of API calls exceeds the API call rate limits on the Github server side. In the future developments section of the system documentation, a possible fix for this issue is outlined. As for the current version of the web app, *it is recommended* to comment out the code related to the computation of the commit message length/code length ratio. This can be done in the `general_semantic_score.py` file, in the `calculateWeightedCommitSemanticScore` function.

The lines in this function to be commented out are as follows:

```
# Asynchronously compute code/commit message ratio. Await result
awaited_bounded_ratio = await AsyncCodeCommitMessageRatio.compute_code_commit_ratio(owner,repo,pr_num,commit_SHA,commitJSON)
bounded_ratio = sigmoid(awaited_bounded_ratio)
# Multiply metric with its respective weight
weighted_bounded_ratio = cmcl_weight * bounded_ratio
```

*Figure 3: Screenshot of software: Code that should be commented out.*

Please ensure that in the `total_weight` variable, `cmcl_weight` is commented out

## Visualisation System

The repository page contains a large central chart, which is the main source of visualisation for our repository data. This chart serves as the essential tool for understanding how a team is performing overall, and even how subsets of the team are performing, across any selected time range.

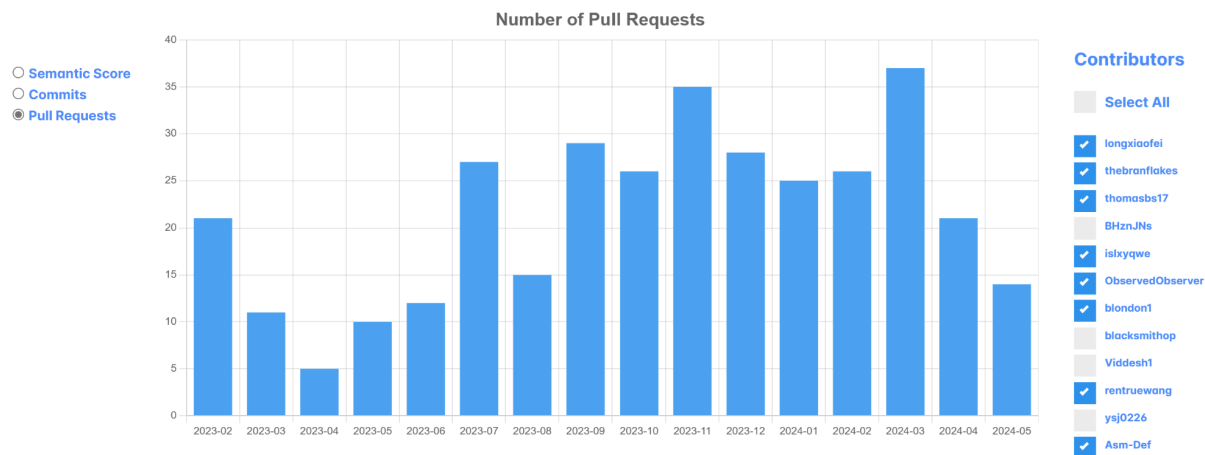


Figure 4: Screenshot of software: Graph from the frontend with contributors de-selected as an example.

It is able to display three different sources of data: the average semantic score for commit messages over time, the number of commits over time, and the number of pull requests over time. Changing this display mode is done through the radio buttons on the left sidebar.

All data is, by default, displayed on a per-month basis. This means the semantic score averages and the total counts of commits and pull requests are calculated per month. However, when a bar is clicked, the chart “zooms” into that month, showing these same calculations but on a per-day basis, and limiting the range to the specific month that was clicked. A button appears to be able to zoom out again. On the right sidebar, there is a “Contributors” checkbox list. This displays all contributors to the repository, each with a checkbox, which allows for easy filtering of all displayed data, per user or subset of users. The contributor selection also affects the list of pull requests shown below the chart, at the bottom of the repository page. The date range selector at the top of the page affects the total range of time shown in the chart.

# Developer Guide

## Architecture Overview

### System Components

The software is composed of two major components; the frontend and the backend. The two components work independently and only communicate when they need to pass information to the other. They concern different parts of the software and therefore also need to be run separately in the terminal. To put it simply, the backend communicates with the GitHub API and does most of the calculations, and the frontend manages the visualisation.



The frontend, made with Vue, is used to create the Web Application. The frontend concerns itself with the design aspect. It communicates with the backend to receive the information that it wishes to display. The frontend uses a combination of HTML, CSS and Javascript.

The backend uses Django, and works in Python and SQL when communicating with the database. As soon as the backend receives a command from the frontend it will run the necessary functions. Next it will send the requested information in a JSON package to the frontend.

## Data Flow

The communication between the two components can be seen as a circle. It starts at the frontend and as soon as the frontend requires information it sends a “message” to the backend. Then the backend sends the mandated information over to the frontend. The backend will not send over information without a request and hence the circle always starts with the frontend and is completed by the backend sending a package of information to the frontend. The buttons in the frontend are linked to different pages in the frontend which require information from the backend. For example, the submit button on the *Home Page* will send the Github Repository link to the backend which will start processing and fetching information from the repository.

The backend retrieves repository information using API calls. For every necessary component of the analysis, an API call is made to retrieve the data. Other calculations not readily available by the GitHub API are calculated in the backend code as well.

## API Documentation

### Overview

Making API calls is essential for this project as the software can therefore collect actual relevant data. There exists a sufficient amount of API calls that can be made by our software to retrieve the data required, and to store this data in a database for easier access later. The backend can for example retrieve the pull requests of a repository and store these in the allocated database. More information on this can be found in the System Documentation

### Authentication and Authorization

In order to authorise the usage of a Github Repository we use a key personal access token, generated by GitHub. When using the software you should use your own person access token to use the software.

# Configuration

## Security Settings

The system requires a username and password in order to enter the software. This is to ensure that only allocated users can use and access the contents of the software seeing as the information does concern some private data.

The inserting of a URL also contains a security constraint that ensures that the URLs inserted are in accordance with a Git URL. This is to avoid SQL injections or other malicious action. This is the only place in the software where the user can enter an input and is therefore the only place where a security measure was necessary.

## FAQ

## Common Issues and Solutions

### Frontend required installments

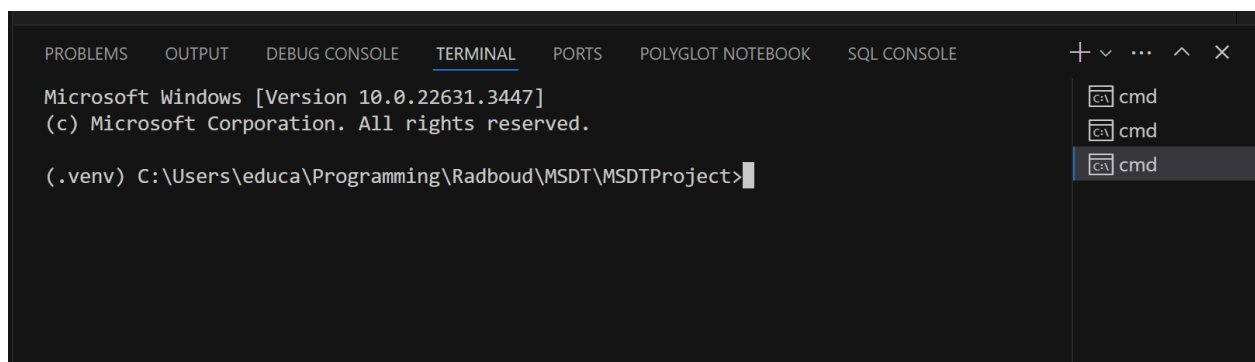
There are cases in which the frontend cannot run as there are missing installments. This can be solved by running `npm install` in the frontend terminal.

### Python Virtual Environment

In the case that the virtual environment does not create itself smoothly there are some things that you can try to fix this. First however, verify that you are running the project via Virtual Environment. This can be done by following the steps below:

#### Verify that you are running the project via Virtual Environment

1. Once the `.venv` has been created, open a new “command prompt” terminal within VSCode. The shortcut is usually `Ctrl+Shift+`` I think. It should have a “`.venv`” in front of the command line, like this:



The image shows a screenshot of a Visual Studio Code terminal window. The terminal is titled 'Microsoft Windows [Version 10.0.22631.3447] (c) Microsoft Corporation. All rights reserved.' and shows the prompt '(.venv) C:\Users\educal\Programming\Radboud\MSDT\MSDTProject>'. The terminal is part of a larger interface with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', 'POLYGLOT NOTEBOOK', and 'SQL CONSOLE'. On the right side, there is a sidebar with three 'cmd' icons, indicating multiple command prompt instances.

*Figure 5: Screenshot of software: Example in the terminal to show how if the virtual environment is working.*

2. If so, you're good! Run the project from here and install any new libraries through here (with "pip install [whatever]"). If "(.venv)" isn't showing up, try restarting VSCode, do 5 min of googling if necessary, and if it's still not working then, let me know, it can take some weird fiddling sometimes. And we can log that on Jira. Good luck!

You may get an error when trying to run your environment with `.\.venv\Scripts\activate` stating that Microsoft doesn't allow you to run the venv because of security measures. In order to circumvent this, you should start the microsoft powershell as administrator. Check your current status with ``Get-ExecutionPolicy``. Remember this status and then execute ``Set-ExecutionPolicy Unrestricted -Force``. After this, you should be able to activate the venv. **Don't forget to put your execution policy back to Restricted** → `Set-ExecutionPolicy Restricted -Force`

### Missing Dependencies

If the error is something in the scope of `OSError: [WinError 123]...` then in the case of one of the developers, they had to install all the dependencies manually. Likely because the requirements.txt did not properly automatically install them. The dependencies that they had to add were as follows and there might even be more.

1. `Pip install django`
2. `Pip install django-cors-headers`
3. `Python migrate.py migrate`
4. `Pip install requests`
5. `Pip install numpy`
6. `Pip install rest_framework`
7. `Pip install aiohttp`
8. `Pip install github`
9. `Pip install nltk`
10. `Pip install PyGithub`

These are the steps that they followed. Make sure to check the errors when running `manage.py` in between as this can tell you which dependencies need to be added in between.

# Appendices

## References

- Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.
- Loria, S. (2018). textblob Documentation. Release 0.15, 2.