Constructiveness & Inclusiveness Evaluator

Team:
Giulio Cappellani,
Olga Kondratenko,
Milton Carreno,
Yuan Qian
aka GOMY Bears Team

Sponsors:
Malini Bhandaru (Intel)
Anna Jung (VMware)

Instructor:
Doug Halperin

CS690 Fall 2021
December 15th 2021

Project Overview

Description of Project

We created a user-friendly tool that allows anyone to evaluate the health of an open source project, according to two criteria: constructiveness and inclusiveness. We gathered, cleaned and formatted data with numpy and pandas; derived sentiment analysis with natural language processing from NLTK and Vader; created, trained, tested and validated machine learning models with sklearn and tensorflow/keras.

List of Features

1. Containerized Application
    a. 3 Docker Containers, Ready to Deploy to GCP
2. New labelled Dataset
3. ML
    a. Voting Classifier
        i. Decision Trees
        ii. Random Forests
        iii. K Nearest Neighbours
    b. Neural Network
    c. Vader
        i. Sentiment Analysis
    d. Vectorization
        i. TF/IDF Scores
        ii. Metadata Calculations
4. UI:
    a. Welcome page
    b. Evaluate page
        i. PR evaluation block
        ii. Project evaluation block
    c. PR evaluation
        i. Amount of Evaluated Comments
        ii. Constructiveness
        iii. Inclusiveness
        iv. Negative Sentiment
        v. Neutral Sentiment
        vi. Positive Sentiment

d. Project evaluation
    i. Amount of Evaluated Comments
    ii. Constructiveness
    iii. Inclusiveness
    iv. Negative Sentiment
    v. Neutral Sentiment
    vi. Positive Sentiment
e. About
f. Error pages and Same Page Error Handling

Next Steps

This project is ready for the following features and improvements:

1. Store evaluation results in DB,
2. Load Evaluated projects from DB to the web page,
3. Add more labelled data to extend project's dataset,
4. Add more new separate features to be evaluated by like merged PR,
5. Deploy the application to GCP/Amazon/etc.,
6. Add new ML algorithms to voting classifier,
7. Add new neural networks,
8. Tune more models' hyperparameters,
9. Add comparison classical ML vs Deep learning evaluation results

and much more.

Design and Software Architecture

Updated Section 6

Diagrams/Explanation of what was built

Design

1.1. Design Considerations

1.1.1. Describe the most difficult problems – understood to date – that must be solved to successfully complete the Project.

There are 3 problems. For the first is time consuming, and involves labeling data and it may be hard to make a labeling decision in some cases.

The second pertains to actual machine learning – exploring different model algorithms and training them to eventually select the best. There are multiple model classes, those that use high level engineered features such as decision trees and clustering and those that require large volumes of training data but less

feature engineering, or the neural network class of models. We will need to prepare the data and train and retrain. To avoid the tedious error prone nature of this repetitive search task, we shall use Machine Learning operation tools such as Kubeflow. A group of problems contains two of them: selecting the best model and building a pipeline. Best model part will include selecting performance and other metrics to measure results of different models as well as tumin models' hyperparameters in order to get the best one or more. For the pipeline it is unclear how to do it as none of us have done it before. It is going to be challenging.

The third problem is the integration of different components of the project, how to feed the data to the trained model in real time. How to interact with the user and return the result the user expects.

1.1.2. Did any of the above or did other constraints influence your choice of technical architecture? How?

For annotation we are using google spreadsheet to help with the annotation process, we annotate the data manually. We chose docker and flask as this combination is usually used for projects such as this one to streamline the development process.

1.2. Technical Architecture

1.2.1. If the Project is a component of a larger System provide a diagram showing how the Project fits into the other components of the System.

Not applicable since this project is a standalone application.

1.2.2. Provide a high-level overview of how the functionality and responsibilities of the Project are partitioned between subsystems, components, or modules. The main purpose here is to gain a general understanding of how and why the Project was decomposed, and how the individual parts work together to provide the desired functionality.

Diagram 1 shows how our project starts with training of a model on the particular open-source project from GitHub. We are going to extract data from GitHub, clean it and form a dataset. Then we are going to choose a model out of a list of models by the best results in testing metrics.

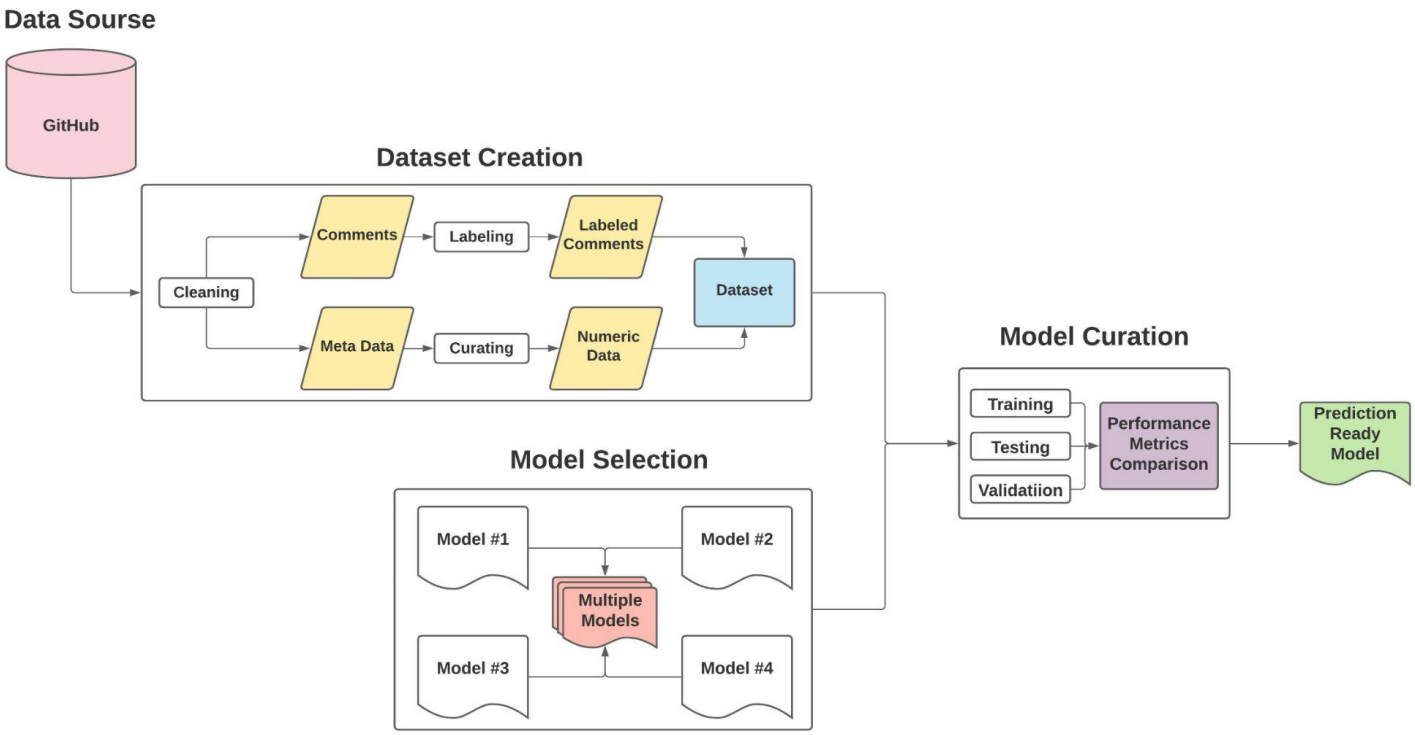


Diagram 1

To use our project after the first part is completed please refer to the diagram 2 below.

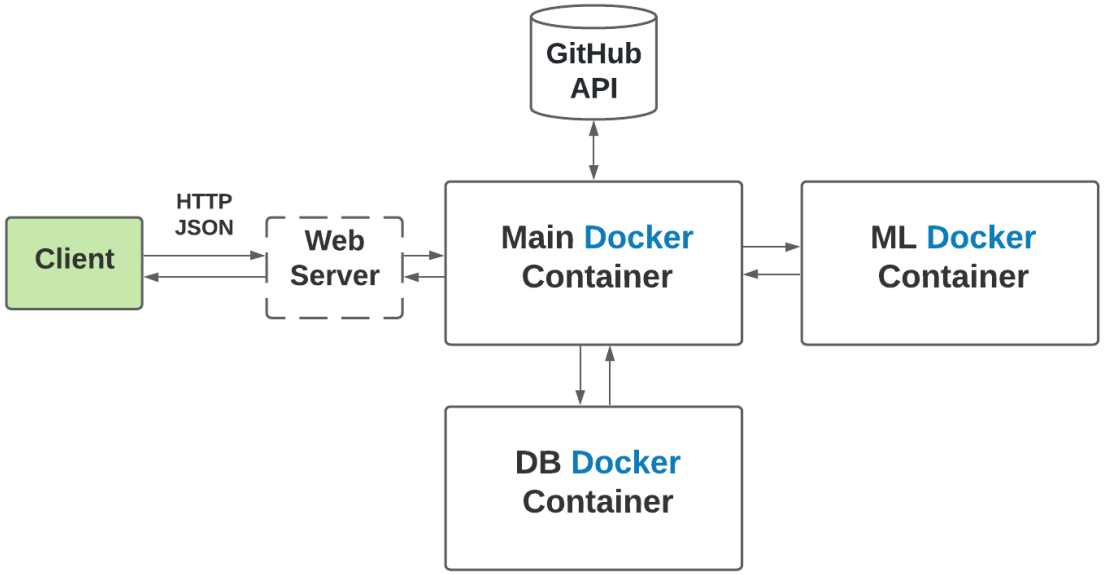


Diagram 2

We also create a website with a user interface for evaluating a project by the link for a specific year or one PR by the link from GitHub and showing the score for the entire project.

1.3. Project Modules & Interfaces

1.3.1. Describe each of the major software modules planned to be implemented for the Project.

Pipeline:

1. data extraction
2. data cleaning
3. feature engineering
4. training model

Diagram 1 is a short description of the Pipeline that we are going to implement using all tools that were described above. Pipeline will contain scripts to gather all needed data and integration of data. Data extraction is a list of searching queries to fetch data from specified GitHub repositories using API calls to GitHub GraphQL, data cleaning is a part of work with the data to remove

1. user identifying information e.g. login, id, user name, user email, signature, etc.
2. not relevant data for the model e.g. sha, urls, html links, personal data, unique ids. html links worth extracting but drop for ML training etc. Why extract, so you can review the original online and see if what the model predicts makes sense.

After data was cleaned we may need to add more features from existing data, split or combine some data, perform feature analysis to determine what features have impact on the models' training.

Website (time permitting):

1. UX/UI
2. front-end
3. back-end
4. API
5. DB

We are going to use figma for designing UI, HTML5, CSS,potentially Vue.js for the front-end, Flask, SQLAlchemy, Docker for the back-end, Postgres as the website's database. Our simple database schema is shown on diagram 3 and site architecture on diagram 4.

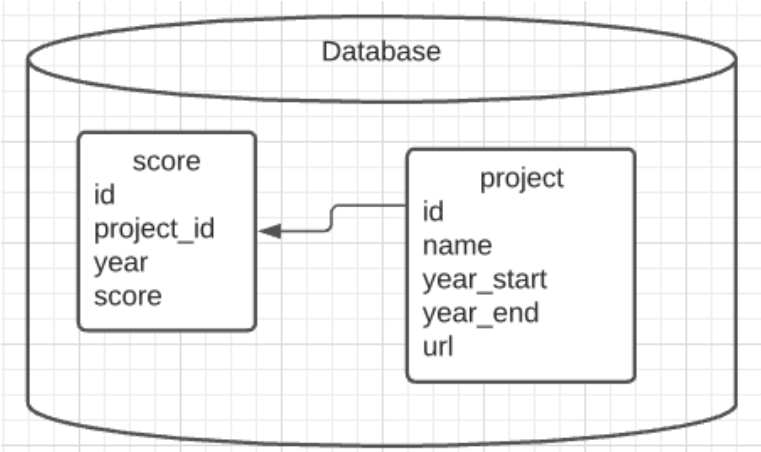


Diagram 3

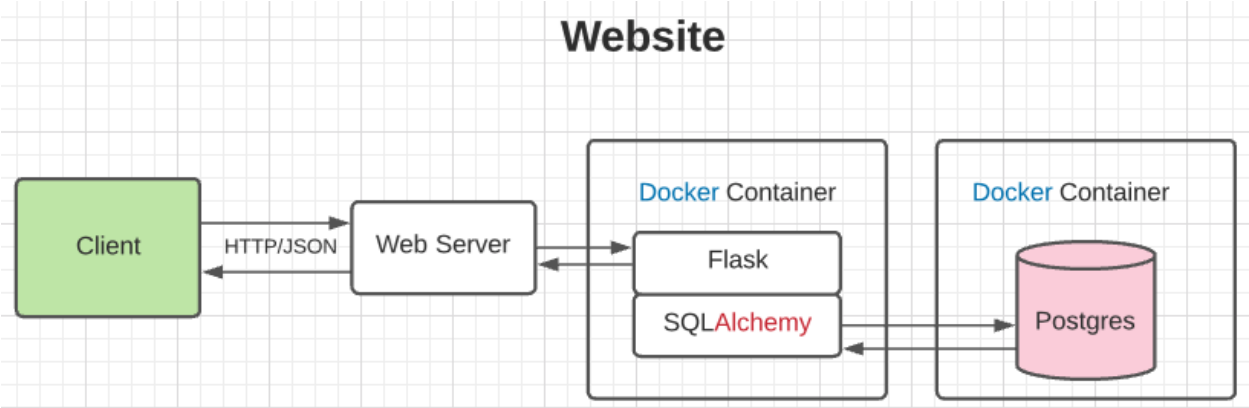


Diagram 4

1.3.2. *If there are any APIs that the Project connects to, describe these and provide references to any available documentation for these APIs.*

Our project will connect to GitHub API to retrieve data from open source project repositories.

1.3.3. *Are there any other systems (e.g., an ftp server) that the Project must rely upon?*

Docker

1.3.4. *If there is an API being developed in the Project (e.g., a RESTful interface to the server-side of an application), list and described planned calls to be implemented as part of the API.*

Any user:

HTTP POST to get a score of the project

HTTP POST to get PR evaluation

Admin (by providing another url):

HTTP POST to store the score into the DB

1.4. Data

1.4.1. What are the key groups of data in the Project? For example, user data, product information, etc.

PRs' comments from the projects' repositories with timestamp and other related metadata.

1.4.2. Provide a flowchart of how data will move through the components of the Project. Additional flowcharts may be appropriate for additional/different types of data in the Project.

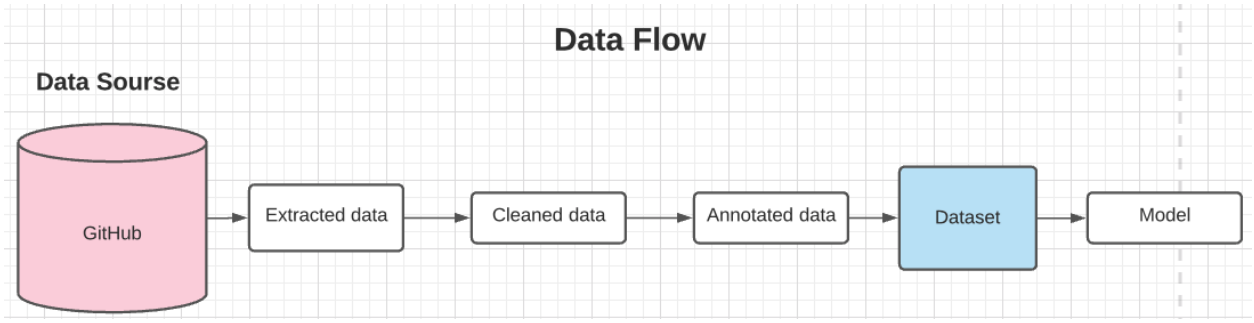


Diagram 5

1.4.3. If there is a database planned for the Project, what type of database will it be? (e.g., MySQL)

Postgres

1.4.4. If there is a database planned, provide a list of tables planned with some description of the fields expected for each of the tables. Alternatively, provide a full database schema.

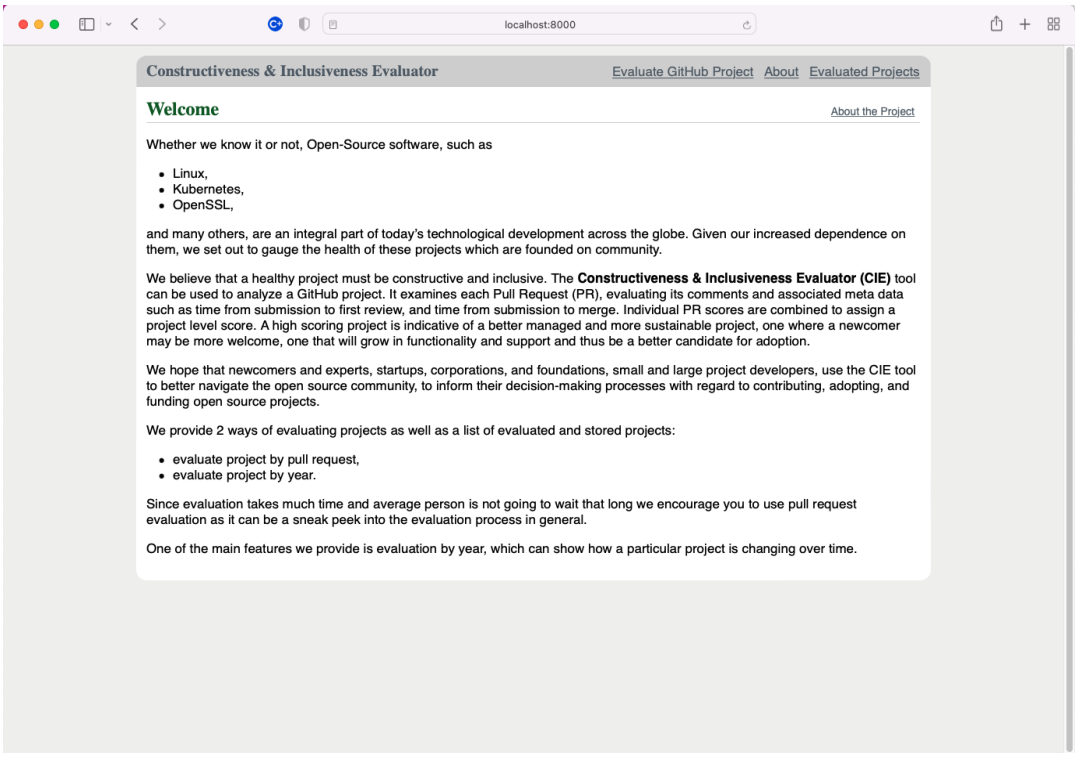
1.5. User interface (UI)

1.5.1. Through what mechanism will users interact with the Project? For example, Browser, command-line, etc. Will it be different for different types of users?

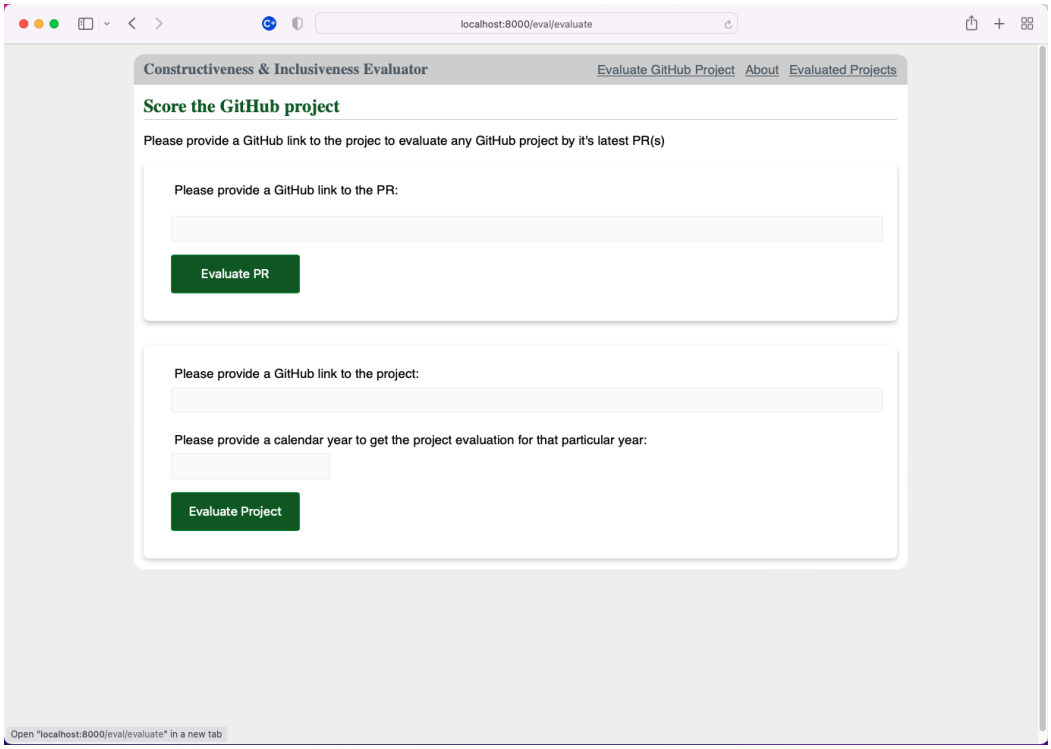
Website.

1.5.2. Unless the Project is an internal tool, provide a series of sketches showing how each of the key pages/views of the application will look for each major function/feature supported by the Project.

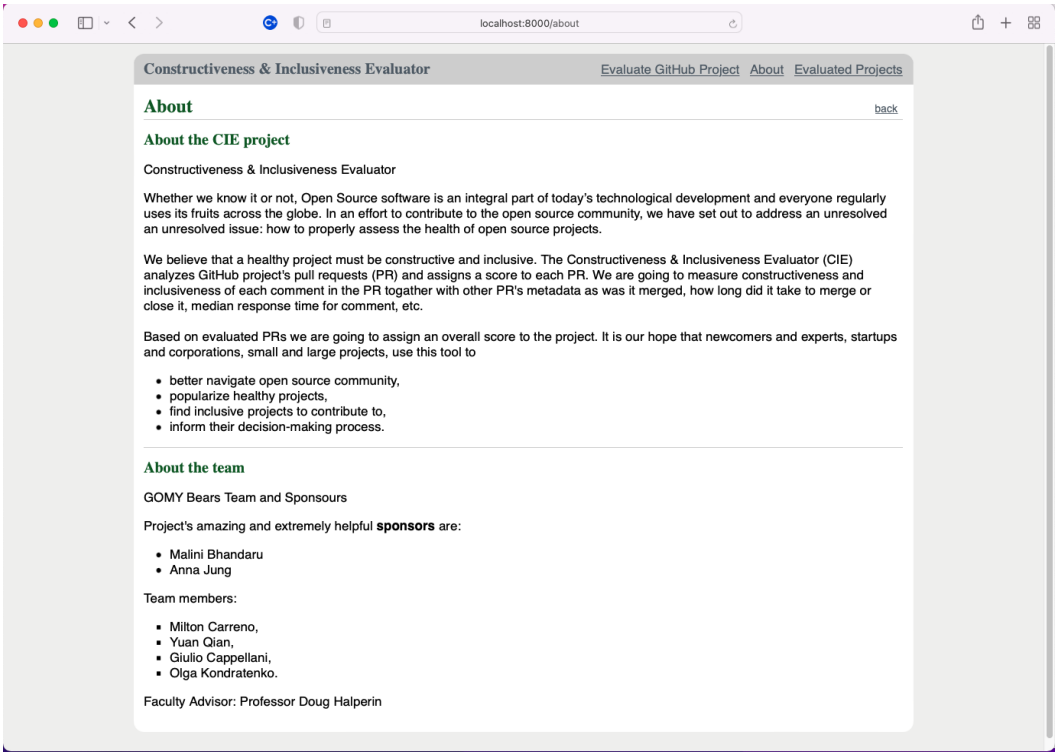
Our project ended up having the following pages included below.



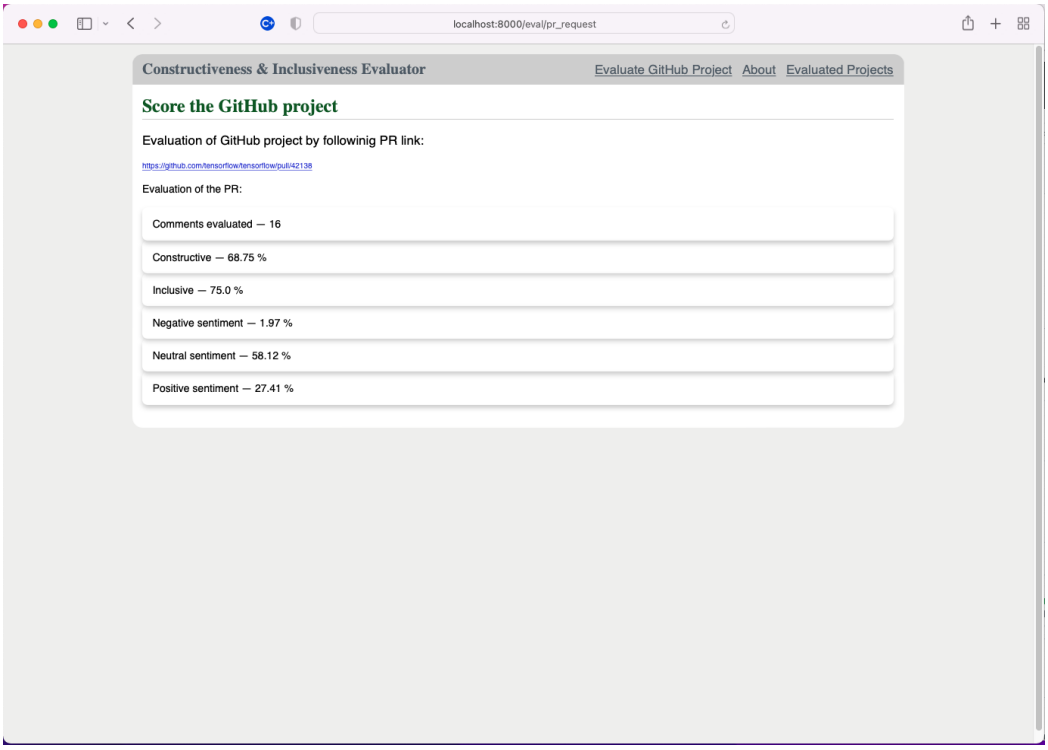
Screenshot 1: Welcoming page



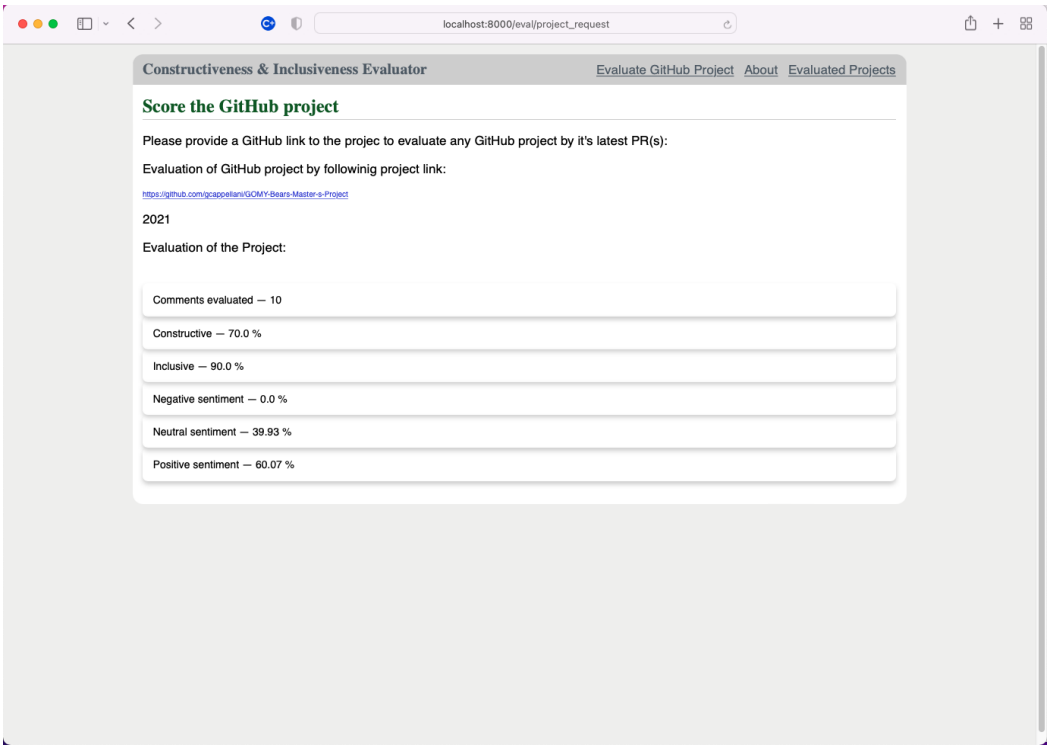
Screenshot 2: Evaluation page



Screenshot 3: About page



Screenshot 4: PR evaluation results page



Screenshot 5: Project evaluation results page

Build Instructions

In Docker

To get the project up and running in Docker

1. Clone the repository https://github.com/gcappellani/GOMY-Bears-Master-s-Project into a new folder
2. Make sure that Docker is installed and running
3. In the console run the following command to build the application:
   docker-compose build
4. In the console run the following command to start the application:
   docker-compose up
5. When you are done using it and want to shut down just press **CTRL + C** and close terminal and docker application.

When you are done using it and want to remove images run the following command:
docker-compose down -v

Locally

To get the project up and running locally

1. Clone the repository https://github.com/gcappellani/GOMY-Bears-Master-s-Project into a new folder
2. Make sure that Docker is installed and running as DB is configured to Docker
3. Run DB in Docker
4. Under /back/flask/extract/graph_ql_request.py, replace {YOUR_ACCESS_TOEKN} with your own access_token
5. Create a virtual environment:
   python3 -m venv .venv
   source .venv/bin/activate
6. After the environment created, in the console open 'back' folder by **cd <cloned folder path>/back** and run the following commands to install requirements:
   pip install -r requirements.txt
7. Now open second console window and go to the ml folder **cd <cloned folder path>/ml**
8. Run the following commands to install requirements:
9. After all required packages installed, in the console run the following commands to build the main flask application:
   pip install -r requirements.txt
10. In the console run the following commands to start the ml application:
    export FLASK_APP=service
    flask run --host 0.0.0.0 --port 5001
11. Make sure it is up and running by visiting localhost:5001/health in the browser, if it shows response then close this page, no need to keep it open.

12. Switch to the first console and run the following commands to start the main application:
    export FLASK_APP=flaskr
    flask run
13. If everything went well by opening localhost:5000/ in the browser you see the welcome page.

When you are done using the application and want to shut it down:

1. Open both console windows and just press **CTRL + C** in both consoles.
2. In any console close the virtual environment by:
   deactivate
3. To close DB image run the following command:
   docker-compose down