WikiContrib

Release 1.0

Dec 23, 2019

Contents:

1 Intro **1**

2 Usage **3**
- 2.1 Intro to Queries 3
- 2.2 Query Creation 3
- 2.3 Viewing / Updating filters 5
- 2.4 Viewing results 5
- 2.5 Updating Queries 6
- 2.6 Deleting Queries 6

3 Internal Working **7**

4 Installation **11**
- 4.1 Backend 11
- 4.2 Frontend 13

5 Contributing **15**
- 5.1 Reporting bugs 15
- 5.2 Contributing to the repo 15

CHAPTER 1

Intro

WikiContrib is a contribution data visualization tool which can be used to view a developer's contributions from different contributing platforms to the Wikimedia Foundation, designed to help the Scholarship Committee for reviewing applications for Wikimedia events.

Currently, it fetches the contributions from

1. Gerrit
2. Phabricator

It visualizes the data in form of graphs and user contribution calendar. The tool can also be used by anyone to view the contributions of any contributor in the Wikimedia Foundation.

Note: The tool was called **Contraband** during development. After the initial development phase, the tool was renamed to **WikiContrib**.

CHAPTER 2

Usage

As specified, WikiContrib is used in visualizing the contributions of the wikimedians in the form of graphs and user contribution calendar. Let's start the discussion with how to use the tool first.

2.1 Intro to Queries

A Query is initiated whenever a user searches for the contributions of a single or a group of wikimedians. A query has a lifetime of 30 days i.e the query expires in 30 days from the creation time. Each query is uniquely identified by a **64-bit** hash. Each query has a **group of filters** associated with it. There are three filters in a query:

1. Status of the commit.
2. From time.
3. Time Range.

Status of the commit: This is used in filtering the commits based on their status like **merged**, **Abandoned** etc. The user can even provide multiple statuses once. Suppose the user add "Merged" and "Open" to the status filter. It will search for the commits that are merged and open in both the platforms.

From time: The user can get all the commits from a specified time (i.e month and year).

Time Range: From the specified **From time**, the user needs to give a range of time to which he/she want to fetch the details (i.e last one year, last one month or last 6 months etc).

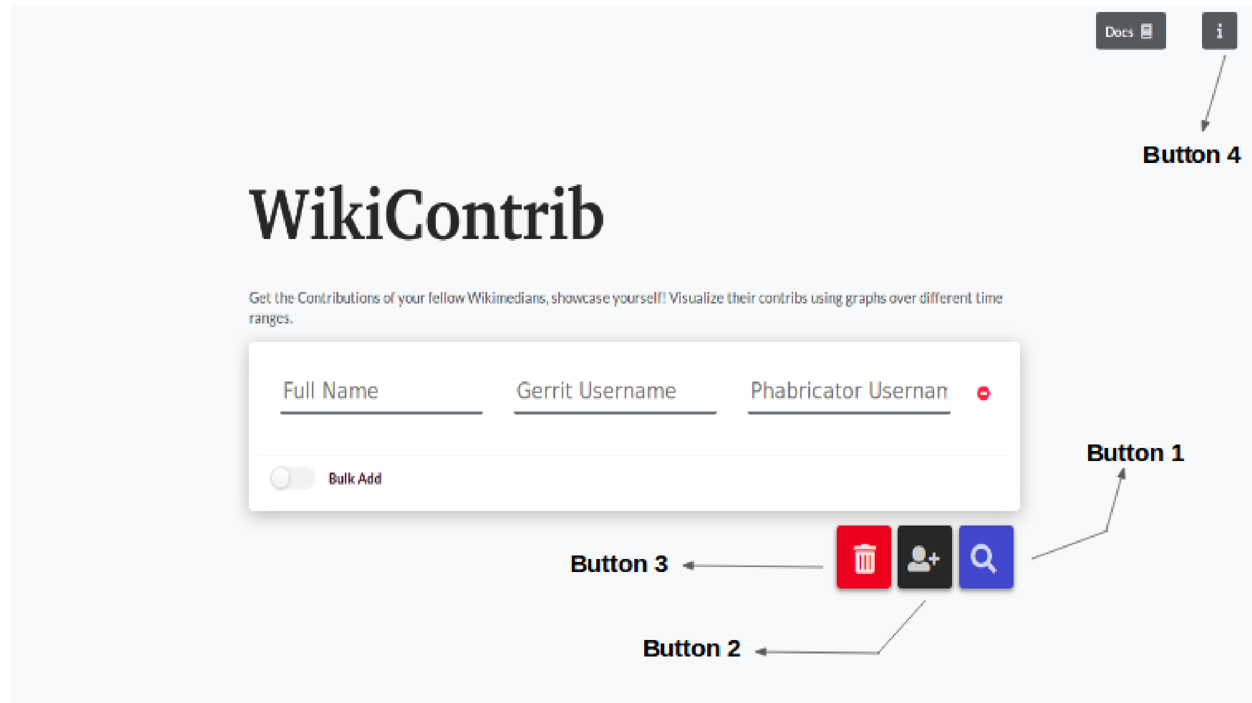
2.2 Query Creation

To create a query, the user needs to add the data (username's of Gerrit and Phabricator) to the query. The user can add the data in two different ways:

1. Entering username's manually.
2. Adding username's in bulk.

Entering usernames manually:

Home page



Whenever the user goes to the home page, the above page will be displayed. In the above above, the table has three columns. They are:

1. Fullname
2. Gerrit Username
3. Phabricator Username.

The user needs to fill the above three details of the corresponding wikimedian whom the user wants to view the contributions of. `Gerrit Username` is used to fetch the data from **Gerrit APIs** and similarly, `Phabricator Username` is used for **Phabricator APIs**. The details fetched from both the APIs are associated with a common name i.e `Fullname` of the user. Fullname is also used in searching the users, we will discuss the searching later part of the doc.

There is an option provided to add usernames of multiple wikimedians. By clicking on **Button 2**, another empty row is added to the DOM. Similarly, the user can add any number of rows and fill the usernames into them. One of the cool things of the tool is the username's, the user entered will be cached to the device. So, even if the user refresh the page or close the page, the details will not be lost! the user can fill a few username's at some time, close the page and re-open it at some other time and add few other username's. The user can also clear the cached data. Clicking the **Button 3** clears all the data he/she entered to the tool.

Adding Username's in Bulk:

Entering username's of ten's of users is easy. But what if the user wants to know the contributions of hundreds of **wikimedians**? It will take a lot of time and work to fill them manually to the tool. So, there is also an option to upload all the username's in a CSV file. On clicking the toggle bar with text "Bulk Add", the user will be provided with an option to upload a CSV file. If the user is uploading a CSV file, he/she need to fill the data in the file in a particular format.

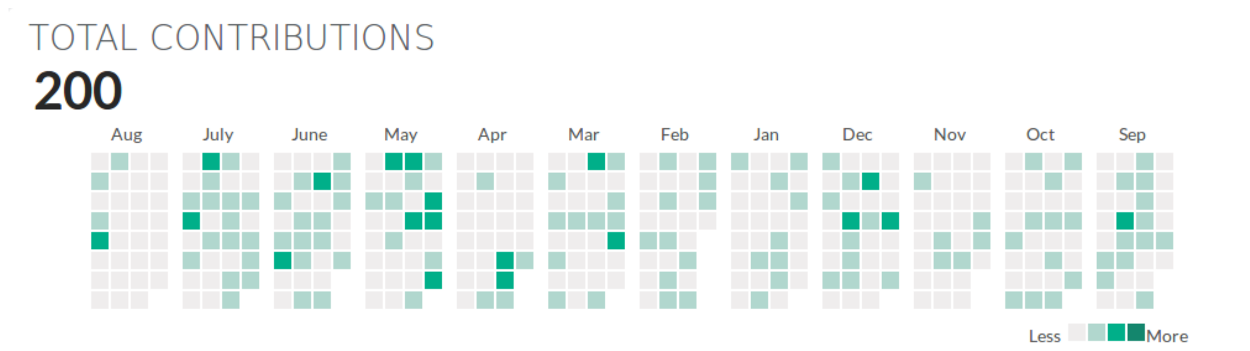
The CSV format is:

fullname	Gerrit	Phabricator
User 1	Gerrit_user_1	phab_user_1
User 2	Gerrit_user_2	phab_user_2
User 3	Gerrit_user_3	phab_user_3
User 4	Gerrit_user_4	phab_user_4
User 5	Gerrit_user_5	phab_user_5

Once the data is provided, (either entered manually or using a CSV file). The user can click the search button (i.e **Button 1**). This initiates a request to the server and starts making API requests to Gerrit and Phabricator. Once all the required details are fetched, it redirects to a URL `/query_hash_code`. Now the user can see the graphs of user contributions vs time along with a calendar that displays the contributions.

Note: `query_hash_code` is the hash-code generated by the query, the query can be accessed at any point of time using the has code.

User contribuion calendar looks like:



If the user click on a specific date in the above calendar, all the commits made by the wikimedian along with the `platform` and `status of the commit` will be displayed.

Button 4 is the Info button, hovering it gives a popup with an intro paragraph about the tool.

2.3 Viewing / Updating filters

The user can view the results by following the above process of creating a query, there are also few filters displayed along with the result. The filters can be updated. Updating the current filters performs an API request and fetches the contributions of the wikimedian according to the filters the user provided.

There is also an option to reset the filters to the default ones. Filters are associated with the Query. The contributions of all the wikimedians are fetched according to the filters the user changed!

Note: Presently, the user can see all the contributions of any wikimedian for past one year (at maximum).

2.4 Viewing results

Once the contributions of the user are fetched, there are these things displayed: 1. Graph of user contributions in **Gerrit**. 2. Graph of user contributions in **Phabricator**. 3. A simple calender that displays all the user contributions for the time span you provide (similar to **github**).

At a time, the contributions of a single user are displayed. There are arrows provided to get the details of the next and previous user to the current user. There is also an input box provided. If you want to get the contributions of a specific user, you can search the `fullname` of the user in the search box. It displays the recommendations of top 50 matching users.

2.5 Updating Queries

Once a user creates a query with the username's of a set of wikimedians and at a later point of time, if he/she wants to know the contributions of another wikimedian, instead of creating a new query for a single wikimedian, he/she can update the query and add the corresponding usernames.

There are four main different types of update's possible:

1. Initially a **CSV file** can be provided, another **CSV file** can be provided while updating the query.
2. Initially a **CSV file** can be provided, a set of **username's of wikimedians** can be provided manually while updating the query.
3. Initially a **set of username's** of wikimedians are provided manually, a **CSV file** can be provided while updating the query.
4. Initially a **set of username's** of wikimedians are provided manually, another set of **username's of wikimedians** are provided manually while updating the query.

2.6 Deleting Queries

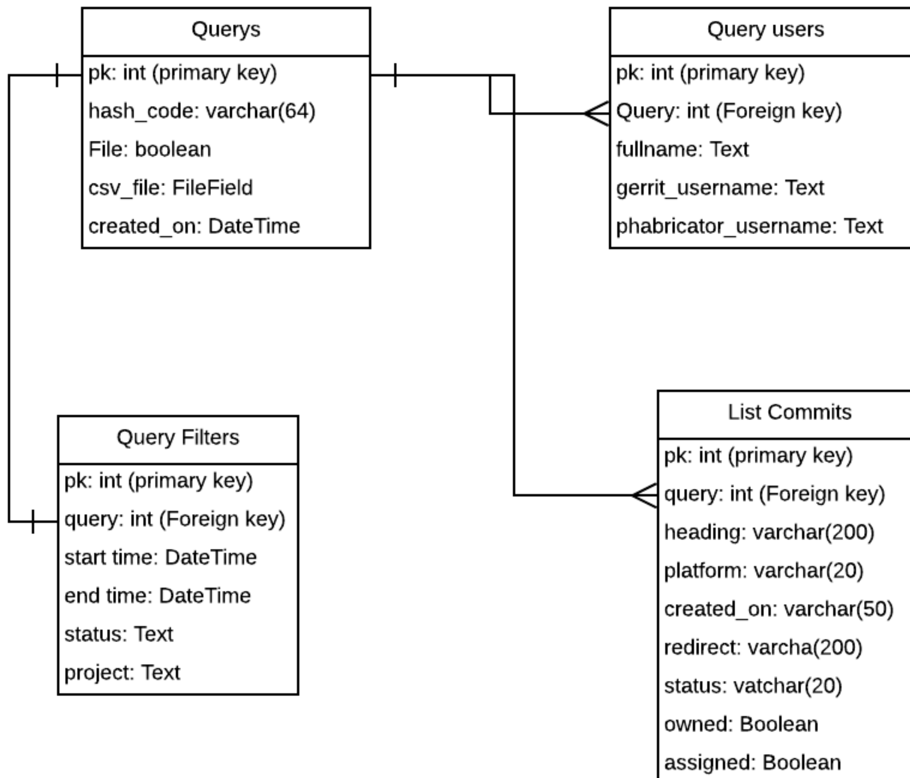
Presently, the is no feature to delete a query. The query will be automatically deleted, after 30 days of its creation.

CHAPTER 3

Internal Working

In the **Usage** section, we discussed the architecture and how to use the tool. Let's extend the discussion with a complete note of how the tool works internally.

We shall start our discussion with the schema diagram of the tool.



As you can see above, there are four tables:

1. Query.
2. Query users.
3. Query Filters.
4. List commits.

Query has the data regarding the `hash`, `created time` etc. The username's of the wikimedians that the user provided during the query creation will be stored in **Query users** table. All the filters assosiated to the query will be stored in **Query Filters** table.

Whenever a request of creating a query hits the server, initially a class named `AddQueryUser` view is triggered. The view creates a Query with a hash and adds the provided username's data to the query. This also creates a default set of filters and rturns a redirect to `/<hash>` URL.

The URL triggers `DisplayResult` view. This view performs external API requests, fetches the details and store the fetched data in the databases. It also formats the data and returns the data to the browser as an HTTP response. Let's dig deep to the working of `DisplayResult` view.

This view uses `asyncio` and `aiohttp` to perform API requests in a parallel manner. There are few constraints with the existing Phabricator and Gerrit APIs. Both of them can not return the count of contributions made by a particular user. They will return the contributions made by the user in the form of a list of JSON objects. The good thing about `Gerrit` is it returns contributions of all the users with a single API request. But in case of phabricator, it will paginate the results with a max of 100 contributions in each page Fo example, if a user performed 1000 different actions in phabricator. then 10 API requests are to be made to get all the actions performed. Another constraint is that all the API requests are to be made sequentially. The API requests can not be parallel because each page has to be requested with a reference(except the first one). The reference to a page **n** will be provided in page **n-1**. Suppose if you have to get the commits of the user in 7th page, you have to request the 6th page first to get the reference to the 7th page. To get the 6th page you have to request the 5th page and so on.

So, even if I want to get some page **n** you have to get all the details from **1 to n**.

In this tool, all the contributions of the user from Gerrit are being fetched. But in the case of phabricator, two kinds of tasks are taken into count:

1. Tasks owned by the user.
2. Tasks assigned to the user.

DisplayResult view gets all the data required to perform the external API requests and call another function `getDetails`. This function takes the data and formats it according to the requirement. It also creates a new **asyncio event loop**. This loop is given with three different co-routines. (If you are not familiar with event loops and co-routines, they are used to perform threading programmatically, you can get more information about them here).

```
async def get_data(urls, request_data, loop, gerrit_response, phab_response, phid):
    tasks = []
    async with ClientSession() as session:
        tasks.append(loop.create_task((get_gerrit_data(urls[1], session, gerrit_
↪response))))
        tasks.append(loop.create_task((get_task_authors(urls[0], request_data[0],␣
↪session, phab_response, phid))))
        tasks.append(loop.create_task((get_task_assigner(urls[0], request_data[1],␣
↪session, phab_response))))
        await asyncio.gather(*tasks)
```

The above code adds three tasks to the event loop. Each of the task fetches APIs and get information.

1. `get_gerrit_data()`: fetch contributions user from gerrit.

2. `get_task_authors()`: fetch tasks owned by a user in phabricator.
3. `get_task_assigner()`: fetch tasks assigned to a user in phabricator.

`get_gerrit_data()` perform a single API request and gets all the details of the users. `get_task_authors()` and `get_task_assigner()` gets the data but, as discussed above, phabricator APIs are paginated. So, these two co-routines has to request the data again and again, till there are no more pages left behind to request.

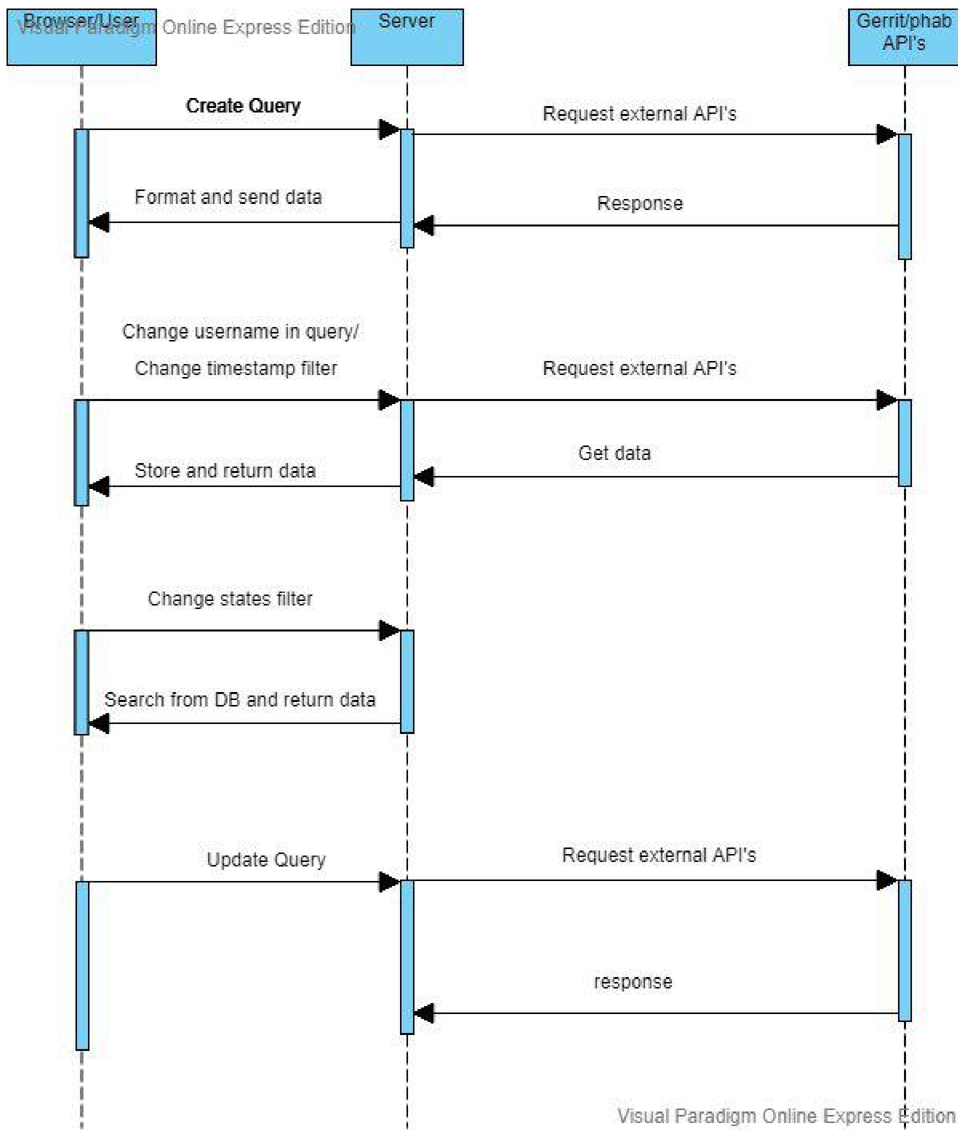
These are parallel because, let's assume there are two tasks **task1** and **task2**, initially, the loop started executing **task1**. If **task1** performs any API request, it has to wait till the response is received to proceed further. So, whenever the **task1** performs an API request, **asyncio** stores the state of **task1** and start executing **task2**. When the response to the **task1** is received, it stores the current task and executes the **task1** further.

Once the entire data are received, it is formatted and stored in the table `List Commits`. Apart from storing them in databases, the commits that meet the requirement of all the Query filters are taken and response is returned to the user. For the sake of performance, the contributions of at the max. of past one year are being requested.

Whenever the filters of a query are changed, if the filter "status of commit" is only changed, then there is no need to request external APIs, as all the commits are available here. So, databases are queried and the contributions that match to the current filters are returned. If the timestamps are changed, then the external APIs are requested again and the details are fetched.

The view `GetUserCommits` returns all the commits of a user on the particular date.

sequence daigram:



If you want to know more about the tool, you can refer the API documentation from here.

CHAPTER 4

Installation

Initially, clone the repo with the command.

`git clone https://github.com/wikimedia/WikiContrib.git`

The tool has two different components(Backend and Frontend). Each of them has their own installation instructions.

4.1 Backend

Now, if you type command `ls`, you can see a directory named **Contraband**. Go inside the directory using the command `cd Contraband.` There will be two directories in it:

1. backend
2. frontend

If you go inside `backend` directory(use command `cd backend`). You can find another directory named `Contraband`. It is the main project backend directory.

Steps to setup server locally

1. Create a virtual environment.
2. Install the required packages to run the tool.
3. Create the phabricator account and generate a Conduit API token.
4. Run the migrations.
5. Create super user.
6. Run the local server.

Creating a virtual environment:

Virtual environment isolates the entire project. The packages used by different projects will be different and few packages used in a project might not be compatible with another one. So, using virtual environment sets up packages individually to each project.

One way to create virtual environment in python is using virtualenv tool

First Install python3, type the following commands inside `backend` directory

```
sudo apt-get install python3
```

Now you need to install pip3, it manages the python packages

```
sudo apt-gt update
```

```
sudo apt-get install -y python3-pip
```

Let's Install virtualenv package.

```
pip3 install virtualenv
```

you have successfully installed `virtualenv`. Noe let's create a virtual environment.

```
virtualenv -p $(which python3) WMContraband
```

The above command creates a virtual environment named `VMContraband`. It creates a directory named `WMContraband` in the current directory. It is recommended to create virtual environment in `backend` directory. You have successfully created the virtual environment. But you need to activate it now.

To activate the virtual environment, type the following command (in the same directory where `WMContraband` is located):

```
source WMContraband/bin/activate
```

Install the required packages to run the tool:

Now go inside the `Contraband` directory (inside `backend`). Install all the packages that are required to run the project using the command.

```
pip install -r requirements.txt
```

To check if `Django` is successfully installed. Type the following command:

```
django-admin --version
```

The output is something like: `2.2.2`.

Create the phabricator account and generate a Conduit API token:

Before running the development server, you need to provide an API key to fetch the details from the phabricator. So create a phabricator account from here. Use **Log In or Register** through Mediawiki(If you don't have a Mediawiki account, you can create it from here.

Once you login to the phabricator. You can generate a Conduit API token from `https://phabricator.wikimedia.org/settings/user/{Your username}/page/apitokens/` (fill your username in the link).

Copy the API token, and paste it in the variable named `API_TOKEN` in the file `backend/Contraband/contraband/settings.py`

Run the migrations:

Now, you need to run the migration files. Type the following command:

```
python manage.py migrate
```

Create super user:

You have successfully created the schema now. Inorder to access the models you need to create a superuser. Use this command:

```
python manage.py createsuperuser
```

The above prompts for username, email and password. The command creates a user with the corresponding username and password. You can see the database tables or models using it.

Run the local server:

Finally, run the server with the command:

```
python manage.py runserver
```

Hurray!! Now you can access the API. You can get the API doc here.

To see the database tables, you can go to this url: `http://127.0.0.1:8000/admin/` and login with the above created credentials.

4.2 Frontend

If you go inside `frontend` directory(use command `cd frontend`). You can find another directory named `WikiContrib-Frontend`. It is the main project directory.

Steps to setup server locally

1. Install npm
2. Install the requirements.
3. Start the development server.

Installing npm

```
sudo apt-get update
```

```
sudo apt-get install nodejs
```

```
sudo apt-get install npm
```

You have installed npm successfully. You can check the verison of npm with the command `npm -v`

Install the requirements.

Now Inside the directory `WikiContrib-Frontend`, type the command

```
npm install
```

This installs all the requirements to the tool.

Start the development server.

Now type the following command in the same directory.

```
npm start
```

This starts a development server in a URL alike `http://localhost:3000/`. Hurray! you have successfully hosted the fronend in local environment.

You have successfully completed hosting the backend and frontend locally. You can visit the "Contributing" section to know how to contribute to the tool.

CHAPTER 5

Contributing

5.1 Reporting bugs

If you find any bugs or issues while using the app. Feel free to report them here.

If you are reporting a new issue, please provide these detials:

1. Title (Descibe the entire issue in a single sentence).
2. Description (Write more about the issue).

3. How to reproduce the issue? 3. Provide some screen shots (if possible).

5.2 Contributing to the repo

Thanks for choosing the project to contribute. You can install the repo locally through the installation instructions provided in "Installation" section. You can find the list of issues in tool here. You can choose an issue and fix it. Keep your master branch updated and pull all the changes before making a PR.

Note: If you are updating the backend after you make some change, you can check if all the tests are passing using the command:

```
python manage.py test
```