Project

Writing an Python module for the Open Humans API & a self-contained, modular Django app

Synopsis:

There are numerous projects which connects to Open Humans through the API like oh-data-uploader, etc. The functionalities generally used by the projects are upload, message, download and delete. Extension of python API library needs to be done. Some of the features are yet to be added in the API library are getting public data files, details of activities a member has joined, details of members who have joined each activity.

While many people working together on various functionalities - there are possibilities for anything to break and go undetected for a long time. Detecting could be difficult. A lot of tests have been added during the pre-outreachy period. Tests for some functions such as download_member_shared,download_member_project_data,
download_all,upload_member_from_dir are yet to be added.

A lot of django applications need to interface with the Open Humans API. Since there is no existing django utility to make this easier, a lot of code is rewritten in a lot of applications. These involve functionality such as logging in and out of Open Humans, getting project information, messaging other members of their project, getting shared Adata from the Open Humans API, etc.

From the user perspective, a major problem is to manually refresh his access token every 10 hours, which is a cumbersome task. Thus it would be extremely beneficial to streamline the application development process, by providing reusable components, forms, and functionalities which are of plug-and-play nature. Writing tests for the django application is crucial - the test suite needs to be up-to-date and should have a good code coverage

Proposed Project

Extension of API

API library lacks the following functions which are present in Open Humans api. The following functions which needs to be implemented are :

1. get_public_data
2. get_public_data_sources_by_member
3. get_public_data_members_by_source
4. get_members
5. get_project

The pseudocode for **get_public_data** function is here. Code implementation of the other function will be done in a similar manner.

Testing:

Unittest has to be done for each function. vcrpy is used for testing and cassettes are generated. Pseudocode for **get_public_data** is here.
The unittesting for **get_public_data_sources_by_member, get_public_data_members_by_source, get_members, get_project** will be done in a similar fashion.

CLI functions

Message and outh_token_exchange functions which have not yet been added to be used directly by command line. These functions will be added.
Pseudocode for message function to be implemented for directly being used by CLI is given here.
Code implementation for outh_token_exchange will be done in a similar manner.

Django application

Requirements

To build a reusable, pluggable django application which utilizes the extended ohapi client library, to provide developers a simple way to interface with the Open Humans API.

Making django app reusable

The core focus of the django application is to provide reusable components, which can be imported and extended by other developers to suit their needs, or else, used without modification. Thus the application must be be open for extension, but closed for modification. Generally a view after processing redirects to a particular template. I will give template as an argument so that it can be customized by the reusable app user. Any url can be redirected to other url on user's choice.
I will be providing default templates for all the major functionalities like refreshing the token, OAuth, login, logout, etc which can be used or be overriden by user's own templates depending on their choice.
The user can add new functionality built on top of the existing reusable app functionality by extending models and forms. The subclasses are substitutable for the base classes. This will ensure that the developer has to take care only of the part of feature they want to customize while reusing the functionality provided by the base classes.

I will implement the following reusable components:

1. Forms and models which the user can subclass
2. Views that are generic and take template names as arguments
3. Functions which will work with the base classes of forms.

Django application built on open-human-api

I will be importing open-human-api package in the django application. Every open-human-api functionality will be implemented as form so that the user can render the form for any function with just basic templates if they want. Any open-human-api function can be called with the appropriate parameters by a form. All the request handling will be done by open-human-api functions itself.

Features of the app

Automation of refreshing access token

Presently, users have to manually refresh User Access Token (OAuth2) every 10 hours. This process will be automated, thus reducing the hassle for them. Every view which might requires the access token will check whether or not the access token is valid. If it is expired, the application will prompt the users to fill in their refresh token, client id and client secret. These will be fields/attributes to the form implementing refresh token functionality which will generate a new token.

Implementation

There are two ways to implement the automation of refreshing access token.
1 class based views: Mixins are used for class based views.
2. Function based views : Decorators are used for function based views.
I will write a decorator/mixin which will check whether the token is valid or not. If not, the url will be redirected to a form to generate the new access token, after which the page will redirect back to complete the original request. This will be used to decorate all the views that require access token.
(Writing a decorator or mixn will be discussed with the mentors)

Directory Structure

The directory structure for the open-humans-api django application will be as follow:

```
django-tiny-open-humans/
   settings.py
   urls.py
   projects/
       dev_example/
       docs/
   static/
       css/
       js/
       images/
```

```
tests/
    Project level tests
templates/
    Overridable templates
```

Open Human API Functions

I will implement the standard features of Open-human-api in the django application.

Message

USE: Send an email to individual users, or in bulk.
API ENDPOINT https://www.openhumans.org/api/direct-sharing/project/message/
Implementation details are given here.

Upload Files

USE: Simple file upload to member account
API ENDPOINT https://www.openhumans.org/api/direct-sharing/project/files/upload/
Implementation details are given here.
A form similar to that for MessageUsersForm will be written. It will have two types of upload functions which will be called on the basis of file size to be uploaded.

Delete Files

USE: A form class will be created in forms.py which will have the following fields.
Implementation details are given here.

Other functions which will be implemented are **Getting Public Data**, **Getting Members by Source, Getting Sources by Member, Getting Members** and **Getting exchange member.**

Testing

tox

Tox is a tool for running tests in different virtual environments. The application will include a **tox.ini** that automates checks. Unittesing will also be done.

Documentation

Sphinx will be used to generate documentation automatically from code docstrings written in reStructuredText (reST).

Sample Application And Usage

The reusable django application will be used by other applications in the following way.

1. Install the open-humans-api with pip install open-humans-api
2. Pip install django-tiny-open-humans (which would have been published in PyPI¶)
3. To the INSTALLED_APPS variable in settings.py, they add our reusable app's name.

A sample application will be provided for easy usage of the Django application.
It will feature a clean interface demonstrating all the functions provided by the Django app.

Pip Package

The Django application will be distributed as a reusable application. The following steps will be required.

1. Add a LICENSE file
2. Add dependencies on install_requires on setup.py
3. Prefix name with django
4. Check for name clashes
5. Create a python wheel
6. Distribute on PyPI
7. Publish on Django Packages

Timeline

Pre-Outreachy period:

1. Extend the existing ohapi library

Outreachy:

1. Extend the existing ohapi library (**1 week**)
   I will write functionality for adding the remaining functions to the client library. The work in this phase will mainly depend on how much I am able to finish before the official outreachy internship period.
2. Writing test for the library (**1 week**)
   For the functions that were added in the previous week, and also including any other functions whose tests are not written, unit tests will be written using vcrpy.
3. Develop the django app - oauth and decorator formation(**1 week**)
   In this week, I shall setup the django application, and add the oauth token refreshing functionality after finalizing the exact technical design by discussing with mentor.
4. Develop the django app - Open Humans API functions(**2 weeks**)
   For the next two weeks, I will focus on each of the functions exposed by the Open Humans API, and implement that in the django application.

5. Writing test for django app (**1 week**)
   I will write unit tests for the functionalities implemented.
6. Unit testing + Integration Testing (**1 week**)
   I will do integration testing, add the tox file, and ensure that the application is able to run in various environments.
7. Write documentation (**2 weeks**)
   Apart from finishing off any work that might have been delayed due to unforeseen issues, I will write the documentation for django application using Sphinx.
8. Provide sample app for use (**2 weeks**)
   A sample application will be provided for users. The sample application will be published along with the reusable app.
9. Package and publish (**1 week**)
   This is the final phase of the project. I will tie up all loose ends, package the application, and finally publish it on PyPi.

Code snippet

get_public_data

```
OH_BASE_URL = os.getenv('OHAPI_OH_BASE_URL', 'https://www.openhumans.org/')
def get_public_data(base_url=OH_BASE_URL):
    url = urlparse.urljoin(base_url, ‘/api/public-data/’)
    response = requests.get(get_public_data(url))
    if not response.status_code == 200:
        err_msg = 'API response status code {}'.format(response.status_code)
        if 'detail' in response.json():
            err_msg = err_msg + ": {}".format(response.json()['detail'])
        raise Exception(err_msg)
    return response
```

Code implementation of the other function will be same.

Tests for get_public_data

```
class APIGetPublicData(TestCase):

    def setUp(self):
        pass

    @my_vcr.use_cassette()
    def test_get_public_data_with_results(self):
        OH_BASE_URL = os.getenv('OHAPI_OH_BASE_URL',
'https://www.openhumans.org/')
        response = get_public_data(OH_BASE_URL)
        self.assertEqual(response.status_code, 200)
```

```
    @my_vcr.use_cassette()
    def test_get_page_invalid_result(self):
        OH_BASE_URL = os.getenv('OHAPI_OH_BASE_URL',
'https://www.openhumans.org/')
        self.assertRaises(Exception, get_public_data, url)
```

CLI function

```
import ohapi
@click.command()
@click.option('-s', '--subject, help='subject', required=True)
@click.option('-m', '--message', help='compose message', required=True)
@click.option('-at', '--access_token', help='access token', required=True)
@click.option('--all_members', help='all members',
                      default=False, show_default=True)
@click.option('--project_member_ids', help='show project_member_id',
                       default=None, show_default=True)
@click.option('-v', '--verbose', help='Show INFO level logging',
is_flag=True)
@click.option('--debug', help='Show DEBUG level logging.', is_flag=True)
def message(subject, message, access_token, all_members=False,
                      project_member_ids=None, base_url=OH_BASE_URL,
                      verbose=False, debug=False):
      project_member_ids = re.split(r'[ ,\r\n]+', project_member_ids)
      ohapi.api.message(subject, message, access_token, all_members=False,
                                  project_member_ids=None,
base_url=OH_BASE_URL)
```

Message form for django application

```
from django import forms
from ohapi.api import message as api_message

class MessageUsersForm(forms.Form):
      subject = forms.CharField(label='Subject')
      Message = forms.CharField(label = 'Message')
      Access_token = forms.CharField(required = True, label = 'Access
      Token')
      all_members= forms.BooleanField(required=False, initial = False,
      label='All Members')
      Project_member_ids = forms.CharField(label = 'Project IDs (Comma
      Separated) ')
```

```
def clean_project_member_ids(self):
      raw_ids = self.data.get('project_member_ids', '')

      # the HTML form is a comma-delimited string; the API is a list
      if not isinstance(raw_ids, basestring):
      raw_ids = ','.join(raw_ids)
      project_member_ids = re.split(r'[ ,\r\n]+', raw_ids)

      # remove empty IDs
      project_member_ids = [project_member_id for project_member_id
                                  in project_member_ids if
project_member_id]
      # check for unallowed IDs
      if any([True for project_member_id in project_member_ids
            if len(project_member_id) != 8 and
            len(project_member_id) != 16]):
            raise forms.ValidationError(
                  'Project member IDs are always 8 digits long.')

      return project_members
def clean(self):
      cleaned_data = super(MessageUsersForm, self).clean()

      all_members = cleaned_data.get('all_members')
      # get this from the raw data because invalid IDs are cleaned out
      project_member_ids = self.data.get('project_member_ids')

      if not all_members and not project_member_ids:
            raise forms.ValidationError(
                  'You must specify either all members or provide a list
of '
                  'IDs.')

      if all_members and project_member_ids:
            raise forms.ValidationError(
                  'You must specify either all members or provide a list
of IDs '
                  'but not both.')

def send_message(self):
      project_members = self.cleaned_data['project_member_ids']
      #Returning the output of the message function so that the user can
use it       #appropriately
      return api_message(self.subject, self.message, self.access_token,
      self.all_members,project_member_ids)
```

Delete Files form for django application

```
Class DeleteFiles(forms.Form):
      project_member_id = forms.CharField(
      label='Project member ID',
      required=True)

      file_id = forms.IntegerField(
      required=False,
      label='File ID')

      file_basename = forms.CharField(
      required=False,
      label='File basename')

      all_files = forms.BooleanField(
      required=False,
      label='All files')

      Access_token = forms.CharField(required = True, label = ‘Access
Token’)
```

Upload files from django application

```
class UploadDataFileForm(forms.Form):

      project_member_id = forms.CharField(
      label='Project member ID',
      required=True)

      metadata = forms.CharField(
      label='Metadata',
      required=True)

      data_file = forms.FileField(
      label='Data file',
      required=True)

      Access_token = forms.CharField(required = True, label = ‘Access
Token’)
```