Airpact-Fire Developer Documentation

Brian Soto, Luke Weber

Much of the documentation you see here is under continual construction. If you see something here You would like to change please let us know through github!

Website

Url: http://airpacfire.eecs.wsu.edu/

The website to Airpact-Fire is responsible for storing images and information uploaded by the android app and applying a specific image analysis algorithm to it to compute a visual range, and displaying the visual range results to the user. The website is also currently capable of (but not limited to) uploading its own images, searching for images, displaying timelines of images in the same area, editing picture information and perhaps most importantly, facilitating discussion of these pictures through comments and our forum powered with spirit forums.

Getting started.

Overview of technologies used

The website to Airpact-Fire uses the Django web engine to manage url redirects and html templates. It is strongly encouraged the developer reviews basic Django concepts. It is recommended the developer complete the "developing your first django app tutorial" located at this url: https://docs.djangoproject.com/en/1.11/intro/tutorial01/ Django relies heavily on Python 2.7. It is encouraged to review basic python functionalities if the developer is unfamiliar with them.

Github is used as the website/app's version manager to assure manageable, consistent version(s) between multiple developers. The master branch is currently what the server runs on. The clone url for the master branch is found at : https://github.com/AIRPACT-Fire/Website.git Please note the database file db.sqllite3 is not present on the master branch. In general, the database should be managed manually, and with care. To review basic git commands and functionality, please read https://guides.github.com/

Gunicorn (Green Unicorn) is currently used as the web server gateway interface to deploy the site onto airpacfire.eecs.wsu.edu. To review basic functionality of this technology, please read http://docs.gunicorn.org/en/stable/configure.html

Installation and setup

Step 1: pull from git

To access the most recent version of our code, pull from our master branch

```
git clone https://github.com/AIRPACT-Fire/Website.git
```

To write changes to the latest master branch please ask your administrator about being added to the contributors to the project

Step 2: installing the things

To install most of the technologies involved with this project, cd to the directory you pulled the code to and run

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

*Note: If you have a setup of python that might be compromised by adding these technologies, we recommend using a python virtual environment. More information on that here: http://python-guide-pt-br.readthedocs.io/en/latest/dev/virtualenvs/

Step 3: Run a local website

To run a local instance of the website. Cd To the directory with your manage.py file and run

```
Python manage.py runserver
```

Step 4: Deploy server

To deploy a new version on the site to *http://airpacfire.eecs.wsu.edu/.* You must SSH into the server, pull from git for the newest version and re run the website.

Please review the steps provided in the *readme given to you in the git repository. In order to SSH into and restart the server.*

Example: Adding a new algorithm to the site.

The following is some pseudo code for how to go about adding a new algorithm to the site.

Step 1: Create your class for the algorithm

Each algorithm should be represented in the form of a python class in file_upload/models.py Please read the Django models documentation; https://docs.djangoproject.com/en/1.11/topics/db/models/ for more information on the significance of models. Essentially, each class should represent a table in your database.

Each algorithm class should contain a foreign key reference to the picture table, as well as a field labeled *calculatedVisualRange.* Here's what a custom algorithm class may look like:

```
class MyAlgorithm(models.Model):
        # Foreign key reference to picture
        picture = models.ForeignKey(Picture, unique = True, on_delete=models.CASCADE)
        # Our calculated visual range
        calculatedVisualRange = models.FloatField(null=True, default=0)
        # Custom field
        customField1 = models.FloatField(null=True, default=0)
```

Now the visual range must be computed after all of the values have been retrieved. For now, we have been computing visual range in the *save* method of Django models. To override the save, the psuedo code is simply:

```
def save(self):
        self.computedVisualRange = # some crazy algorithm function
        super(CustomAlgorithm, self).save()
```

Please remember after you create a class in models.py, you must run the commands
python manage.py makemigrations
and
python manage.py migrate
before you can see the changes reflected in your database.

Step 2: Create your form(s)

Django uses models.forms to assist the developer with front-end form creation.

The functions in forms.py each produce their own form when placed in a django template in html.
If that sentence didn't make sense, please bare with me.

Now that you have created your cool new algorithm class,, go to file_upload/forms and add some functions specific to your algorithm name. Let's say *custom_algorithm_form()* and add variables in your function specific to your algorithm. For more details about how Django forms work, please read https://docs.djangoproject.com/en/1.11/topics/forms/

Below is what a custom form might look like:

```
 custom_algorithm_form():
        field1 = forms.DecimalField(widget=forms.HiddenInput())
```

Step 3: Create your own templates.

If you are unfamiliar with Django templates, please read up on them here:
https://docs.djangoproject.com/en/1.11/topics/templates/

You need to create your own html files (or templates as Django calls them) for both creating and editing your algorithm. This html file should be placed in Website/templates. This process is difficult to document since every algorithm will be vastly different (in theory), and the Django documentation does a "*decent*" job of explaining basic templated html files.

Here are a few pointers:

- The view will pass the boolean variable *has_alg* into the html, which will return True if the current html page is acting to edit the algorithm, or False if the current html page is acting to create the algorithm.
- You will probably want the image your algorithm is acting upon. The view will pass information about the image through the variable *picture* into the html. To display this picture your html may look like:

```
<div class="col-md-9" style="">
    <canvas id="canvas"></canvas>
</div>
```

  And your javascript might look like:

```
// Fill the canvas with the appropriate image
var context = $("#canvas")[0].getContext('2d');
var canvas_picture = new Image();
canvas_picture.src = "{{ picture.image.url }}";
canvas_picture.onload = function() {
        $("#canvas")[0].width = canvas_picture.naturalWidth;
        $("#canvas")[0].height = canvas_picture.naturalHeight;
        context.drawImage(canvas_picture, 0, 0);
        // Do we have an algorithm already created?
        if("{{ has_alg }}" == 'True') { // Do things
}};
```

That should successfully fill your canvas with the appropriate image.

Step 4: Views

file_upload/views.py is organized to accommodate any number of future algorithms without re-writing the *apply_algorithm* function which is associated with the url /picture/<picture_id>. To add your own algorithm, perform the following actions to file_upload/views.py. Also, don't forget to include your new form function and class at the top of file_upload/views.py

- Write your function to create your custom algorithm object. It takes in a reference to the picture object it's referencing along with the form where it will get all the data needed to create the new object. It should return true upon successfully saving a new custom algorithm object. It might look something like:

```
def create_custom_algorithm_object(Picture, form):
        if form.is_valid():
                newAlg1 = CustomAlgorithm(
                picture = Picture
                Field1 = form.cleaned_data.get("field1")
                )
                CustomAlgorithm.save()
                Return True
        Return False
```

- Write your own function to edit your algorithm, please see the edit_algorithm_one/two() functions for an example.

- Fill the following functions with your new information :
  `Retreive_algorithm_object` - should include an entry into the dictionary that looks like:
  `"CustomAlgorithm" : CustomAlgorithm.objects.filter(picture = Picture)`
  `retreive_algorithm_form()` - Please follow the example provided for algorithm's one and two. If postData is present, this function should be sending post data to the forms
  `apply_create_form()`
  `apply_edit_form()`
  `int_to_algorithm(answer)`
  `retreive_html_page(Picture)` - Please enter a new dictionary item containing your algorithm name and the html page it corresponds to

And if you manage to make it through all that you should be good to go!