CAM2 UI Team
Cloud Editor for User Modules
Crowdsource for Metadata
Scheduled Image Saving

Prepared by:
Jin Bai
Purdue University
12/2/2015

Under Supervision of:
Professor Yung-Hsiang Lu
Purdue University

Cloud Editor for User Modules

Introduction:

The purpose of this feature is to provide CAM2 users a way to edit their modules on the web application itself, instead of redownloading it and editing it locally, then reuploading it. The Ace Cloud Editor was embedded into a new page, that could be accessed when a user wants to edit their modules. These modules are stored in the development server and can be readily downloaded. Using ACE, Django, Python, and Javascript, we enable users to edit files locally. The goal of this feature is to give increase usability for the clients by providing another more convenient way for them to edit their modules.

Background:

To get a better understanding of how ACE works, we can take a look at the documentation. A simple editor example is provided below. Figure 1 shows what the editor looks like to the user. Figure 2 is the actual UI code and how the UI code actually works.

```
1  /**
2   * In fact, you're looking at ACE right now. Go ahead and play with it!
3   *
4   * We are currently showing off the JavaScript mode. ACE has support for 45
5   * language modes and 24 color themes!
6   */
7
8  function add(x, y) {
9      var resultString = "Hello, ACE! The result of your math is: ";
10     var result = x + y;
11     return resultString + result;
12 }
13
14 var addResult = add(3, 2);
15 console.log(addResult);
16
```

Fig. 1 – Simple ACE Editor

This is what the simplest Ace Editor looks like from a user perspective.
https://ace.c9.io/#nav=about

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>ACE in Action</title>
<style type="text/css" media="screen">
    #editor {
        position: absolute;
        top: 0;
        right: 0;
        bottom: 0;
        left: 0;
    }
</style>
</head>
<body>

<div id="editor">function foo(items) {
    var x = "All this is syntax highlighted";
    return x;
}</div>

<script src="/ace-builds/src-noconflict/ace.js" type="text/javascript" charset="utf-8"></script>
<script>
    var editor = ace.edit("editor");
    editor.setTheme("ace/theme/monokai");
    editor.getSession().setMode("ace/mode/javascript");
</script>
</body>
</html>
```

Fig. 2 – Simple ACE Editor Source Code

This shows the code for the editor, without any backend components.
https://ace.c9.io/#nav=embedding

Methods in ACE can be found in the how-to guide.
https://ace.c9.io/#nav=howto

Procedure:

The main idea of this project was to utilize this basic ACE editor example and mold it into something that works for our system. Much more goes into a cloud editor than just the text editor itself. This task can be separated into two phases, front-end development of the editor and back-end development for the editor.

From a front-end perspective, a new page needed to be made that intuitively links users from looking at their modules, to giving them an option of editing existing modules. We will use HTML and CSS to do this.

From a back-end perspective, the data from existing modules must be able to be manipulated and shown to the user so that it can be edited. Then this data must be saved. We will use Django, Javascript, and Python to do this.

Below is the code for the new URLs needed.

urls.py UNREGISTERED

urls.py

```
# Nothing fancy here, normal regex to match urls
# Using the name='*' so we can change the urls whenever
# This adheres to the django DRY principle

urlpatterns = patterns('',
    # Profile
    url(r'^profile$', profile, name="profile"),
    url(r'^show_tour$', show_tour, name="show-tour"),
    url(r'^hide_tour$', hide_tour, name="hide-tour"),

    # Scheduled Saving
    url(r'^configurations/byimage$', scheduled_saving_quick_scan, name="scheduled-sa

    # Configurations
    url(r'^configurations/new$', all_cameras, name="all-cameras"),
    # disabled for current release
    url(r'^configurations/byimage$', select_by_image, name="select-by-image"),
    url(r'^configurations$', browse_configs, name="browse-configs"),
    url(r'^configurations/erase$', erase_config, name="erase-config"),
    url(r'^configurations/(?P<configuration_id>\w+)/edit$', edit_config, name="edit-
    url(r'^configurations/generate$', generate_config, name="generate-config"),

    # Modules
    url(r'^modules/submit$', submit_module, name="submit-module"),
    url(r'^modules$', browse_modules, name="browse-modules"),
    url(r'^modules/(?P<module>[0-9]+)/download$', download_module, name="download-mo
    url(r'^modules/erase$', erase_module, name="erase-module"),
    url(r'^modules/(?P<module_id>[0-9]+)/edit$', edit_module, name="edit-module"),
    url(r'^editor/(?P<system_type>\w+)/(?P<file_id>[0-9]+)$', text_editor, name="tex
    url(r'^return_file/(?P<system_type>\w+)/(?P<file_id>[0-9]+)$', return_file, name
    url(r'^save_file/(?P<system_type>\w+)/(?P<file_id>[0-9]+)$', save_file, name="sa

    # Submissions
    url(r'^submissions/new$', create_submission, name="new-submission"),
    url(r'^submissions$', browse_submissions, name="browse-submissions"),
    url(r'^submissions/erase$', erase_submission, name="erase-submission"),
    url(r'^submissions/terminate$', terminate_submission, name="terminate-submission
    url(r'^update_submission_state$', update_submission_state, name="update-submissi
    url(r'^download_file/(?P<application>[0-9]+)$', download_file, name="download-fi
    url(r'^download_module_src/(?P<submission>[0-9]+)$', download_module_src, name="

    # Alpha 1 pages
    url(r'^alpha/feedback$', alpha_feedback, name="alpha-feedback"),
    url(r'^alpha/tutorial_video$', alpha_video, name="alpha-video"),

    url(r'^datacollection/new_collection$', new_collect, name="new-collection"),
)

```

Fig. 3 – New URLs Needed (text_editor)

If we want a new page for the user to edit modules, we need to first make a new URL for it. This is done in BigData/django/system/urls.py

Below is the code for the new front end page for text editing.

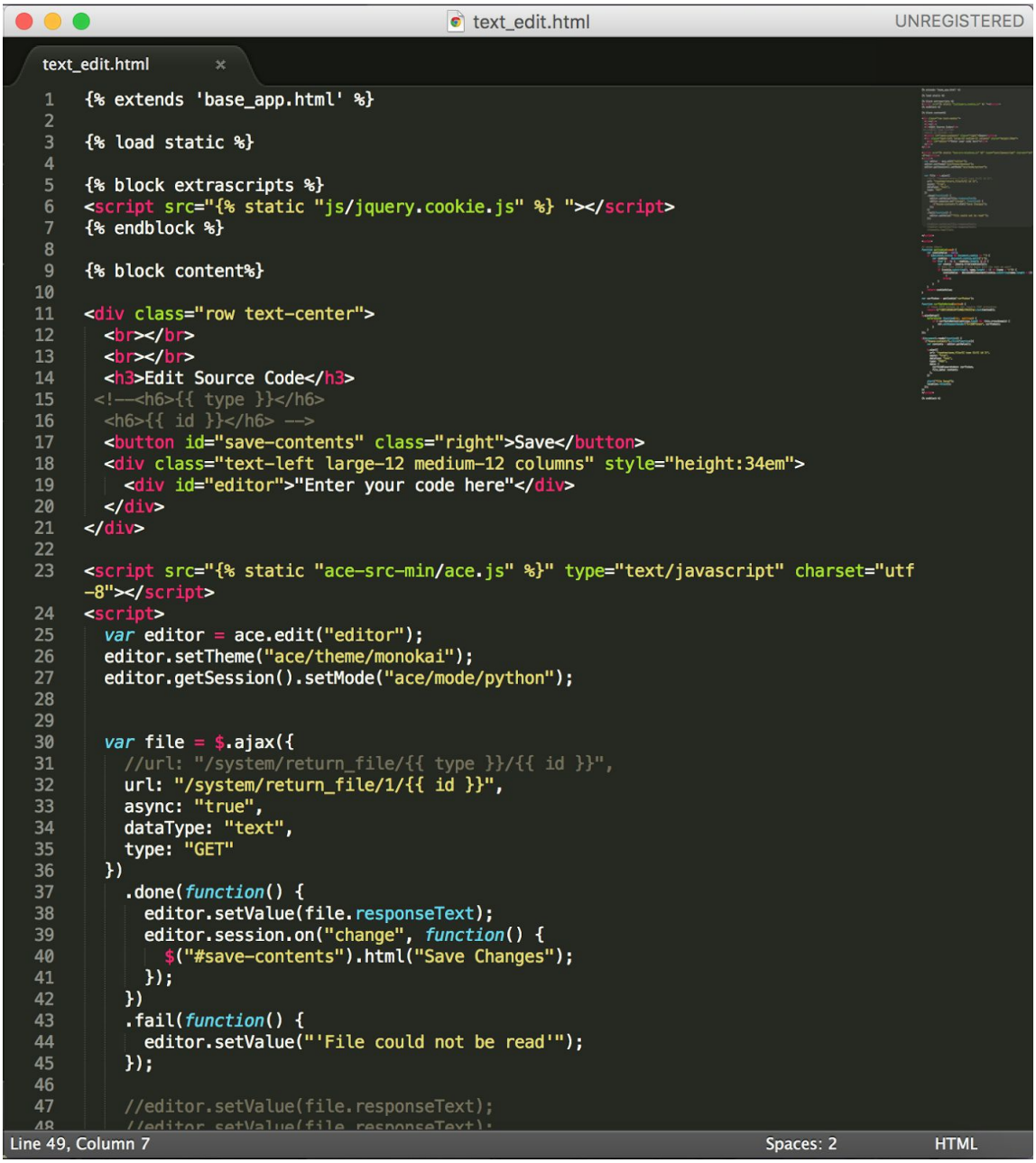

Fig. 4 – Build the Edit Source Code Page

With the URL now made, we can go ahead and build out the edit source code page using HTML and Javascript for the ACE editor.

Below is what the edit source code page looks like.

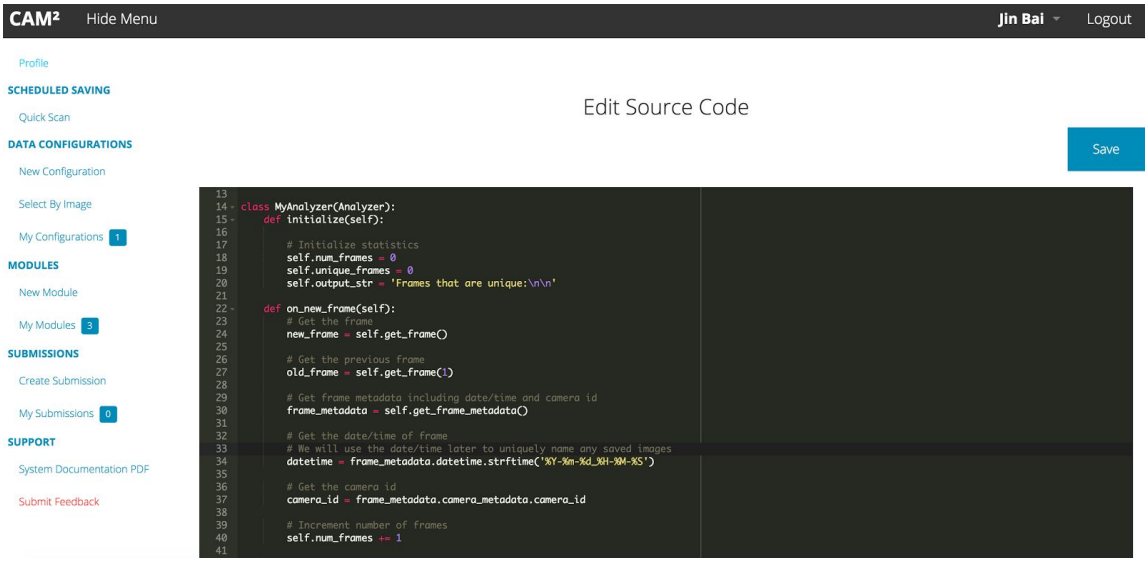

Fig. 5 – Edit Source Code (with module loaded up)

This image already has a module loaded up into it. We have not yet built that functionality from the code shown so far.

Below is how a user would access the edit module page.

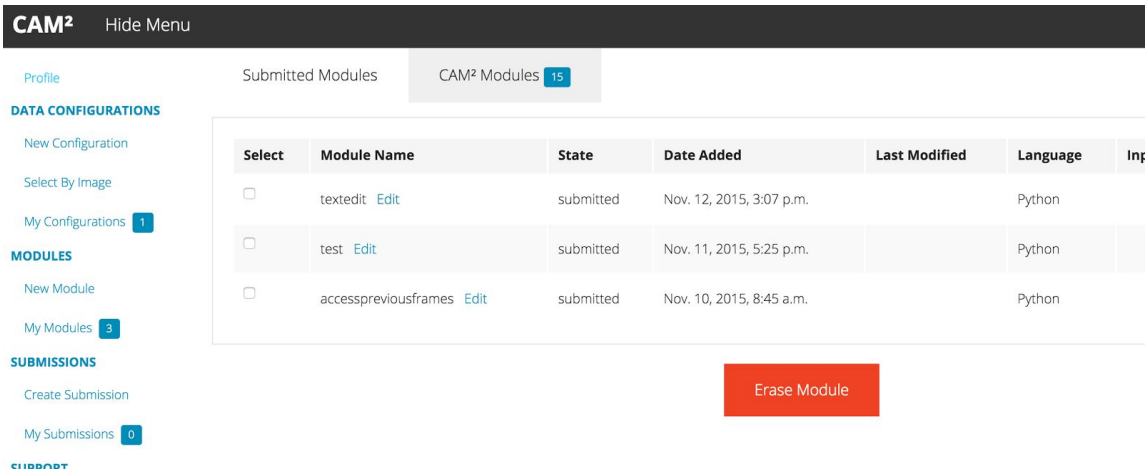

Fig. 6 – Edit button

When browsing user modules, the user now has an option to edit an existing module.

Below is the source code for editing modules.

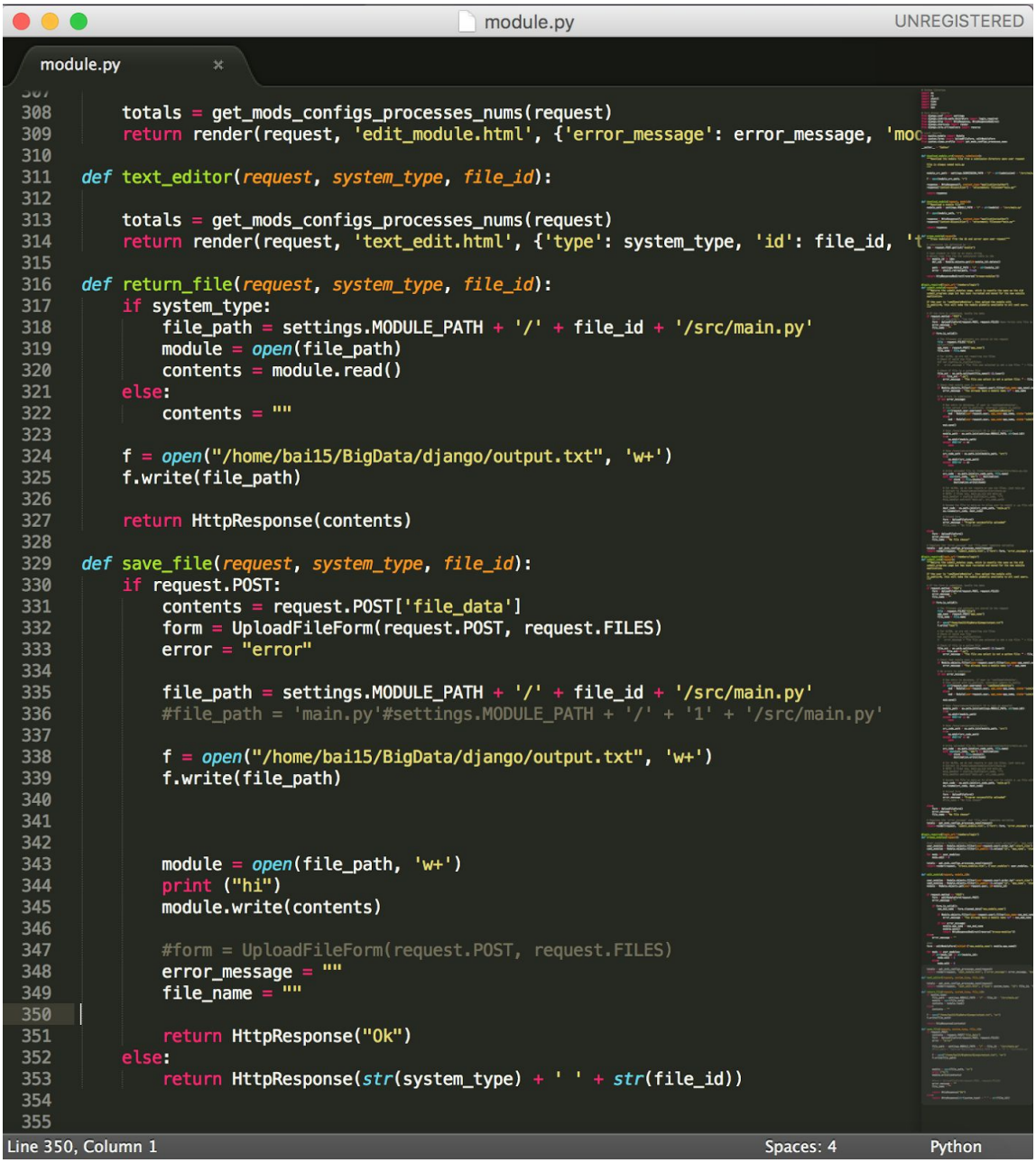

Fig. 7 – Back-end for handling data

Return file returns the file contents (sends it from our system to the editor). Save file saves the content of the editor (updates our system).

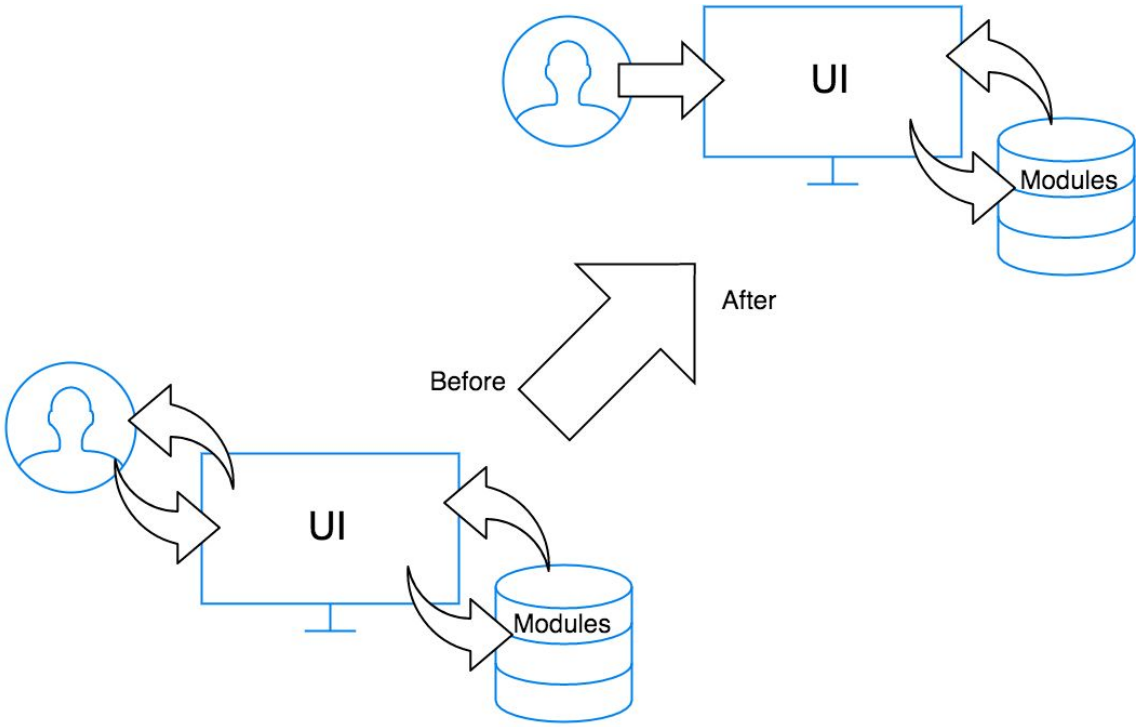

Fig. 8 – High level architecture changes

User no long edits modules locally and uploads them. Editing is done remotely and is more convenient for the user.

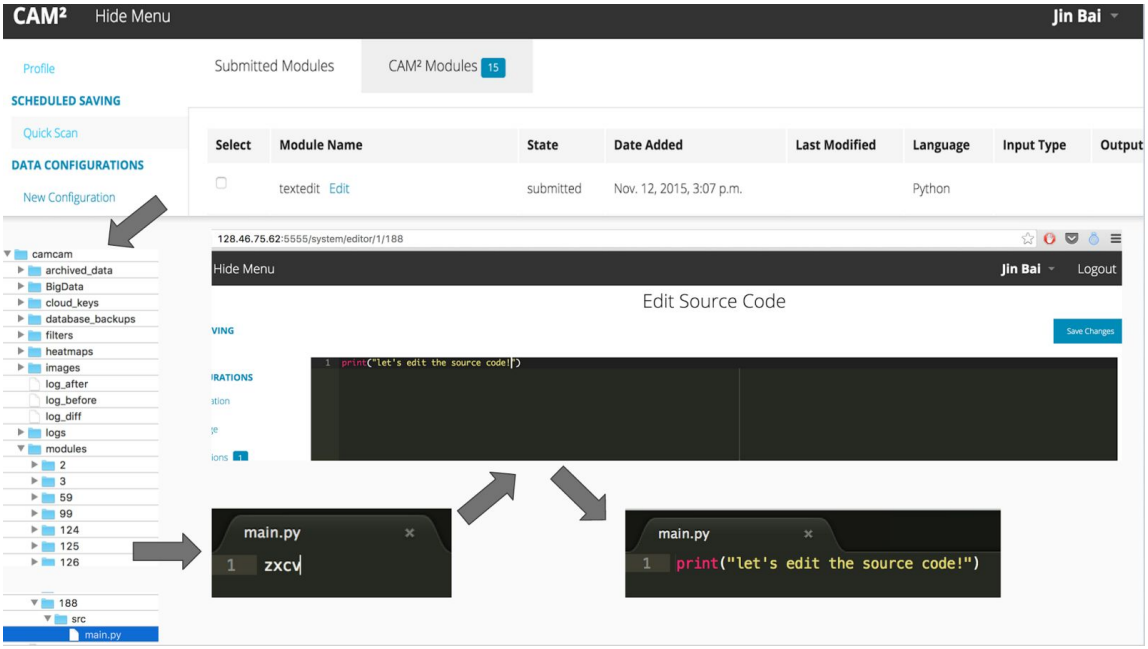

Fig. 8 – High level architecture changes

Demo of functionality. 1) edit 2)backend to frontend data 3)update system

Below is the basic front end design of crowdsourcing data, mocked up by Sam Yellin.

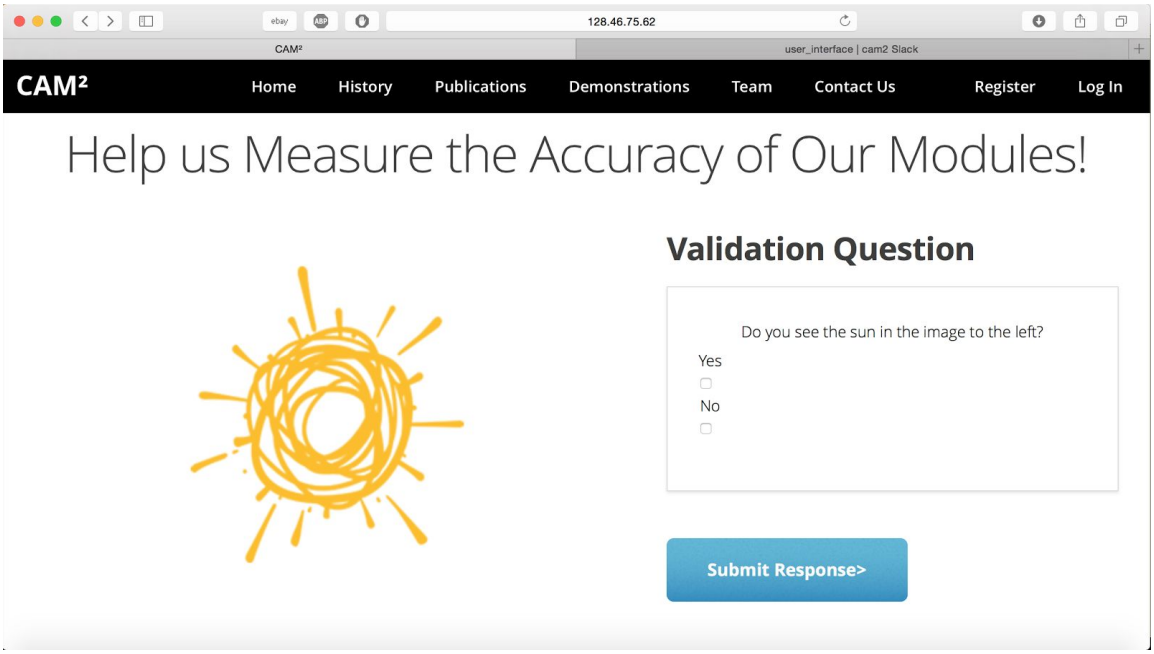

Fig. 9 – Crowdsourcing UI

The purpose of this UI is to garner metadata about our images, data that we can present to our users about the images before running any type of modules on them.

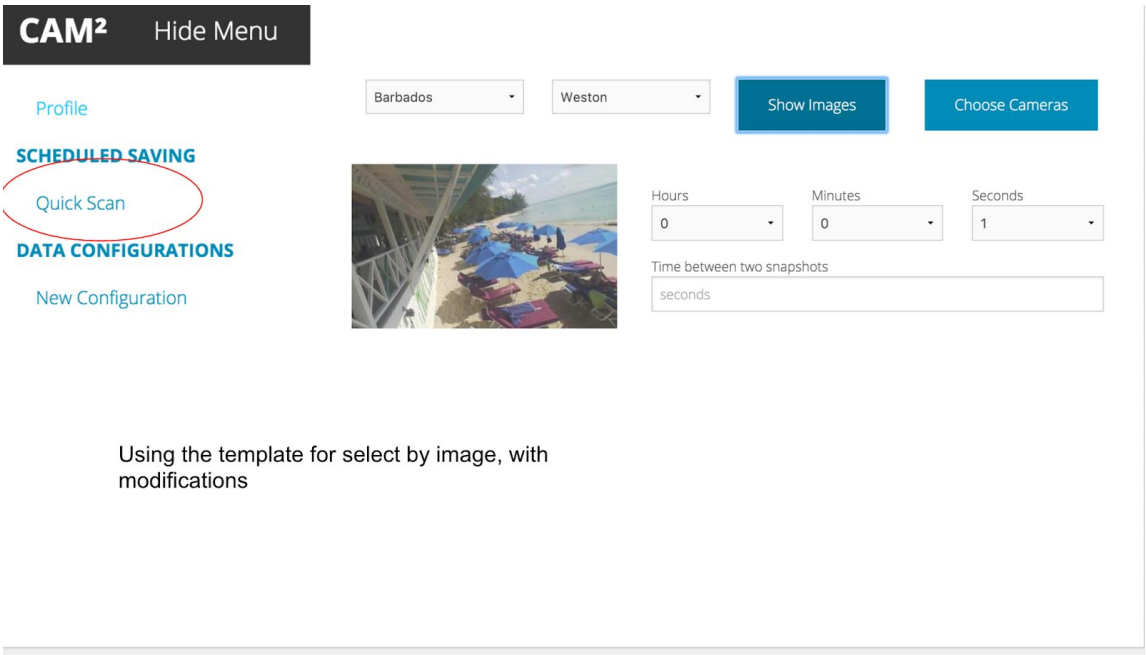

Fig. 10 – Basic Scheduled Saving UI

This UI is for users to get a quick feel for what our site can do, by quickly scanning for a few desired images.

Personal Reflection:

I learned a lot this semester from having a different role than previous semesters. I thought that this semester would be more of the same but it was different because our UI team lead had to drop early on because of other duties. I was thrown into a leadership role. I wasn't used to being the most experienced member of the team. I was used to having expert members to ask questions to whenever any little issue arose.

Being new to leading the UI team, I had my share of mistakes made. Some of my strengths, though, included communication, such as figuring out the status of the team members and making myself available for the team. I was also good at doing the work that was required of myself and finding answers to issues that arose in my personal work. I could also communicate the status of the entire team in a very coherent manner to the rest of the Cam2 team.

Some of my weaknesses showed up as well during my lead. I was overconfident that I would be able to figure out how to implement certain UI elements. This was because from the previous semesters, I was always able to figure out how to do certain things. My time management was not very good. As the leader of a team, your time management needs to be even better. I was also a little unwilling to ask for help. I'm used to figuring out things on my own, but with a team, you need to produce results to help guide the team so that things are on time and specifications aren't missed.

I can still better engage the UI team members. I could probably find a way to engage the busier members somehow as well. I need to learn how to delegate tasks and proceed to delegate them better. I can also recruit more members for this project. Being new to leading the team was hard. I was doing some senior design and taking high level 400+ courses while taking this. My interests began to diversify into other areas of computing which distracted me from this. It was also not easy being the one with the most experience which meant I couldn't collaborate as much with the senior members. All in all, it was yet another new experience which I gained much from.