SweetSai - a Sweet based web app dev effort for Semantic Style Sheets

Sai Gollapudi

March 2, 2016

Contents

1 Introduction

Essentially we are trying to implement the notion of Semantic Style Sheets. Style Sheets essentially provide alternate views to one singular content. In our case we wish to provide alternate narratives to one singular narrative. The alternative narratives may be more meaningful to end users. This idea of a more meaningful narrative is what we are calling semantic. So, a semantic style sheet is a renarrating a existing page into something that has more meaning to an enduser.

The renarration of a page to a more meaningful version is a type of page transformation. We see the page transformation involving new set of parameters

- 'd' a data-structure (perhaps an ontology) to markup a page into a semantically meaningful setup. This is user-defined.
- 'v' a data-structure (perhaps an ontology) to define the various communities to which viewer belongs to. This defines which view to show to a community of user.
- 'r' this defines the renarration rules. This is also a data-structure (perhaps an ontology) which gives the various choices of renarration that are possible for a markup.

2 Requirements

2.1 overview of requirement

The overall aim is to develop a tool that would enable the following:

- an editor environment

++ a user should be able to define a document structure (as an ontology). This we are calling as 'd' in our transformation function +++ this ontology will initially be a simple data structure. Over time it will evolve into a more elaborate, user definable ontology. ++ a user should be able to import a web page by inputing a URL in the input text area ++ a user should be able to markup an input html doc with user defined tags (based on a user ontology, which we are calling 'd')

- a viewer environment

++ a viewer where a user is allowed to see a marked-up file in a newly renarrated manner +++ for this to happen, the user is identified as part of a viewer ontology called 'v'. Initially we will pre-define this ontology as a simple data-structure. Over time this will evolve into a more elaborate ontology that is user definable. +++ there is also a rules for renarration 'r' that we will be using to define how one must transform a marked-up content. This can be quite elaborate. However, for the initial round, we will fix this to be a simple structure and evolve it into a more complex ontology over time.

2.2 roadmap of activity

This Semantic Style Sheet system will be developed in releases called r0, r1, r2, r3...

- r0 - basic implementation of a python server, client side pages to input URLs, view web pages
- r1 - ability to markup a web page, adding a simple 'd'
- r2 - ability to persist the markup done in r1 (database)
- r3 - ability to input 'v' and 'r'
- etc.

2.3 r0 - release 0 requirement

- To develop a client server app using Python backend.
- The UI should be sufficient to take on a URL.
- There should be an ability to view a userinput page.

To develop a SweetApp. Aim of the app would be able to manipulate Sweets. By manipulation I mean that I should be able to create and query sweets of my own making. They should be independent of any specific Sweet App.

3 Design

This pilot, experimental appliction is designed as a client server web application. There is a server side and a client side to it. The inspiration for this work comes from looking at other Sweet based apps like Alipi, Swtr and Restory. These can be found [here : https://github.com/janastu] on github at janastu.

3.1 installation information

- assuming we have Python, Flask, Jinja2, MongoDB already installed
- tangle the SweetSai.org file to get codeblocks. Some will be in the current directory, some will be in ./app directory, yet others in ./app/static, ./app/templates, ./app/static/css, and in ./app/static/js directories.

3.2 execution information

- the app will run on local host at 127.0.0.1:5000 port
- to run we need to activate the virtual environment of sweetEnv by typing ./sweetEnv/bin/activate
- to deactivate the virutatal environment, type in deactivate
- to execute this application, type in the virtual environment, the following: python SweetSai.py. Go to a browser and look for the app running at the 127.0.0.1:5000 local address.

3.3 server side

Initial design choices are based on reference works already done for Sweets and similar apps. Some choices of technology include

- Python 2.7 for scripting
- Flask (micro web development environment)
- WTF for forms
- Requests library for creating / handling HTTP requests
- sqlite database - I am using sqlite database for my work. For this we need to have Flask-sqlalchemy and sqlalchemy-migrate packages installed. As each database is stored in a single file and there is no need to start a database server, this is a

good choice for our application.

- I am also opting to use MongoDb for persistence of information.

When working with Node.js for AngularJS, I used npm package manager.

3.3.1 CORS

In our initial part of the development we only needed to display the Sweets that we could GET via the front-end app from the existing back-end swtr-store (Sweet Store). To do this we could conceptually just rely on the fron-tend, which would be develop using JavaScript. This would make HTTP GET request to the Sweet Store and fetch us the HTML page with the Sweets, and then essential display this gathered info.

There is one challenge to this design approach - CORS. CORS security check would forbid the browser to do a cross-domain request to the Sweet Store. That is, by design a Client (browser) should be contacting the Server (Sweet Store). However, the Sweet Store server (due to CORS) will not trust our domain and not accept our HTTP requests.

To overcome this, we are actually not anymore calling the Sweet Store from the front-end web browser, but actually calling the store from this back-end or server-side code.

Due to this, we need to make our Client (Browser) talk to our own trusted Server (written up in Python/Flask) and then, in turn, have that local server be registered with Sweet Store as a trusted domain and then, finally, have our trusted local domain work with Sweet Store. The exchange between our trusted local domain Server and our Client browser can happen using JSON objects - which need to be formatted by the client for viewing on the client side. Essentially the connectivity then is from our BrowserClient to our trusted Local Server. This is in JSON. Then, our trusted Local Server then connects with Sweet Store directly using Python/ Flask based HTTP protocol (which does not incur CORS treatment).

Here a Python code makes a request to the Sweet store, gathers the Sweets from that using an API. This gathered info is in the form of a JSON object. This sweet store furnished JSON object would then be filtered on the front-end and then displayed.

If we just seek the Sweet Store Page, we get the HTML output. But, we can actually call in the API here. This is where the Sweet Store API rests. This API allows us to fetch the JSON objects.

3.3.2 SWeet Store API

Normal Sweet Store HTML page is available at http://teststore.swtr.us. The API is available at http://teststore.swtr.us/api/sweets/q?who=SaiGo Post "q?" one can query who=<userid>, or what=<context>, or where<url>. Or, have a combination of all of these. The connector is the "&" symbol.

The object that is returned will be a JSON object.

3.4 client side

??Angular JS framework is being used for doing the FrontEnd JavaScript development. For AngularJS package management (in the front-end), I use Bower. Bower is to AngularJS what "npm" is to Node.js (in the back-end). I use "Karma" the test runner for my Angular JS front-end.

3.5 Software Engineering aspects

Software Engineering design choices include

- using emacs-org-mode for Literate Programming
- using Virtual Environments
- developing the app in multiple releases r0, r1, r2...

4 implementation

4.1 basic layout & structure

Some of the coding and structure for this work has been inspired by a tutorial by Miguel grinberg. [http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iii-web-forms: Here] is that tutorial.

Directory structure for this app is like this: SweetSai/sweetEnv - for Virtual Environment SweetSai/app - for packages SweetSai/app/templates - for web based templates; html files SweetSai/app/static - for static files

The development of this work has been done in two parts: Editor and Viewer. Both have been implemented in the MVC design pattern.

4.2 Editor

Editor has been implemented as a portal using Python, Flask, Jinja2 framework. It is being executed in a virtual environment.

4.2.1 model (database work)

- database related files
  We chose sqlite database for our work. we had to update the config file with sqlite database related constants: $\text{SQLALCHEMY}_{\text{DATABASEURI}}$ and $\text{SQLALCHEMY}_{\text{MIGRATEREPO}}$.

  we intialize our database in our init.py file.

  our database is expressed as objects. This is defined in our models.py file.

- models.py file
  In the MVC architecture, this part – dealing with models – relates to the models of the database and app that i have used.

```
from app import db

# creating an object for a table named User
class User(db.Model):
   id           = db.Column(db.Integer,     primary_key = True)
   login_name   = db.Column(db.String(64),  index=True, unique=True)
   login_emailID = db.Column(db.String(120), index=True, unique=True)
   sweets       = db.relationship('Sweet', backref='author', lazy='dynamic')

   # should the user be allowed to authenticate?
   def is_authenticated(self):
       return True

   # banned users can be considered inactive
   def is_active(self):
       return True

   # fake users who are not allowed to even log on
   def is_anonymous(self):
       return False

   # returns a unique identifier for user
   def get_id(self):
       try:
           return unicode(self.id) # python 2
       except NameError:
          return str(self.id)      # python 3

   def __repr__(self):
       return '<User %r>' % (self.login_name)


class Sweet(db.Model):
   id       = db.Column(db.Integer,     primary_key=True)

   # the "s" in front represents the notion of a "sweet"; these are
   # attributes of a "sweet"
   sUsrname  = db.Column(db.String(64),  index=True, unique=True) #this is the Swe
   sUrl      = db.Column(db.String(320), index=True, unique=True)
   sContext  = db.Column(db.String(64),  index=True, unique=True)
   sAttrib   = db.Column(db.Text,        index=True, unique=True)
   sTimestamp= db.Column(db.DateTime)
   sUser_id  = db.Column(db.Integer,     db.ForeignKey('user.id'))

   def __repr__(self):
       return '<Sweet %r>' % (self.sUsrname)
```

- Table of Users
  Definition for User table

```
# creating an object for a table named User
class User(db.Model):
   id            = db.Column(db.Integer,     primary_key = True)
   login_name    = db.Column(db.String(64),  index=True, unique=True)
   login_emailID = db.Column(db.String(120), index=True, unique=True)
   sweets        = db.relationship('Sweet', backref='author', lazy='dynamic')

   # should the user be allowed to authenticate?
   def is_authenticated(self):
       return True

   # banned users can be considered inactive
   def is_active(self):
       return True

   # fake users who are not allowed to even log on
   def is_anonymous(self):
       return False

   # returns a unique identifier for user
   def get_id(self):
       try:
           return unicode(self.id) # python 2
       except NameError:
          return str(self.id)      # python 3

   def __repr__(self):
       return '<User %r>' % (self.login_name)
```

- Table of Sweets

Definition for Sweet table. creating an object for a table named Sweet. Sweet has its own ID... but it also has a link with User table one User can scribe multiple Sweets

```
class Sweet(db.Model):
   id        = db.Column(db.Integer,     primary_key=True)

   # the "s" in front represents the notion of a "sweet"; these are
   # attributes of a "sweet"
   sUsrname  = db.Column(db.String(64),  index=True, unique=True) #this is the Swe
   sUrl      = db.Column(db.String(320), index=True, unique=True)
   sContext  = db.Column(db.String(64),  index=True, unique=True)
   sAttrib   = db.Column(db.Text,        index=True, unique=True)
   sTimestamp= db.Column(db.DateTime)
   sUser_id  = db.Column(db.Integer,     db.ForeignKey('user.id'))

   def __repr__(self):
       return '<Sweet %r>' % (self.sUsrname)
```

- db Creation script file: $\text{db}_{\text{create}}$.py
  Here is a python script that creates the database

```
#!sweetEnv/bin/python

# the source for this comes from
# http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iv-database
# this is a database migration script used for moving from one to
# another version of a database

from migrate.versioning import api
from config import SQLALCHEMY_DATABASE_URI
from config import SQLALCHEMY_MIGRATE_REPO
from app import db
import os.path

db.create_all()
if not os.path.exists(SQLALCHEMY_MIGRATE_REPO):
    api.create(SQLALCHEMY_MIGRATE_REPO, 'database repository')
    api.version_control(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)
```

```
else:
    api.version_control(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO, api.vers
```

to create the database we just need to run the following python command in our virtual environment:

```
./db_create.py
```

This will create a database with the label app.db file. This will be a sqlite database. the script will also create a directory called "$\text{db}_{\text{repository}}$". This new directory wiill store the db migration files.

- db Migration script file: $\text{db}_{\text{migrate}}$.py
  migration is implemented to allow us to (in the future) to change the model of the database. Here is a script in python to facilitate that.

```
#!sweetEnv/bin/python
import imp
from migrate.versioning import api
from app import db
from config import SQLALCHEMY_DATABASE_URI
from config import SQLALCHEMY_MIGRATE_REPO

v = api.db_version(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)
migration = SQLALCHEMY_MIGRATE_REPO + ('/versions/%03d_migration.py' % (v+1))
tmp_module = imp.new_module('old_model')

old_model = api.create_model(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)
exec(old_model, tmp_module.__dict__)
script = api.make_update_script_for_model(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGR
open(migration, "wt").write(script)

api.upgrade(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)
v = api.db_version(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)

print('New migration saved as ' + migration)
print('Current database version: ' + str(v))
```

To ensure proper migration tracking, try to not rename existing fields. Limit changes to addition / deletion of fields only. Typing can also be changed. Generated migration script can also be checked to see if it is correct.

migration script can be run by executing the following python script in our virtual environment

```
./db_migrate.py
```

The script has print statements to show where the migration has been stored. version number is also displayed by this script.

- db upgradation script file: $\text{db}_{\text{upgrade}}$.py
  This python script upgrades the sqlite database to the latest revision.

```
#!sweetEnv/bin/python
from migrate.versioning import api
from config import SQLALCHEMY_DATABASE_URI
from config import SQLALCHEMY_MIGRATE_REPO

api.upgrade(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)
v = api.db_version(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)

print('Current database version: ' + str(v))
```

- db downgrade script file: $\text{db}_{\text{downgrade}}$.py
  This python script downgrades the sqlite database by one version.

```
#!sweetEnv/bin/python
from migrate.versioning import api
from config import SQLALCHEMY_DATABASE_URI
from config import SQLALCHEMY_MIGRATE_REPO

v = api.db_version(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)
api.downgrade(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO, v - 1)
v = api.db_version(SQLALCHEMY_DATABASE_URI, SQLALCHEMY_MIGRATE_REPO)

print('Current database version: ' + str(v))
```

4.2.2 view

This code represents the actions to be taken by the webapp when various views are encountered.

```
from flask import (render_template,
                   flash,
                   redirect,
                   session,
                   url_for,
                   request,
                   g,
                   abort )
from flask.ext.login import (login_user,
                             logout_user,
                             current_user,
                             login_required )
from app import app, db, lm
from .forms import get_swtIDForm, MyForm, LoginForm, InputURLform
from .models import User
import requests

@app.before_request
def get_current_user():
    g.user = None
    email = session.get('email')
    if email is not None:
        g.user = email

@app.route('/')
@app.route('/index')
def home_page():
    form = MyForm()
    return render_template('welcome.html', form=form)

@app.route('/showPg')
def showURL_page():
    return render_template('showPg.html',
                           title='user_entered_page',
                           url='www.google.com')
```

```
@app.route('/editor', methods=['GET', 'POST'])
def my_editor():
    form = InputURLform(request.form)

    # this is activated when the form is filled by user
    if request.method == 'POST' and form.validate():
         url = form.url.data
         return redirect('showPg.html',
                           title='user_entered_page',
                           url=url)
    return render_template('inputURLform.html',
                           title='input URL',
                           form=form)

@app.route('/_auth/login', methods=['GET', 'POST'])
def login_handler():
    """This is used by the persona.js file to kick off the
    verification securely from the server side.  If all is okay
    the email address is remembered on the server.
    """
    resp = requests.post(app.config['PERSONA_VERIFIER'], data={
        'assertion': request.form['assertion'],
        'audience': request.host_url,
    }, verify=True)
    if resp.ok:
        verification_data = resp.json()
        if verification_data['status'] == 'okay':
            session['email'] = verification_data['email']
            return 'OK'
    abort(400)

@app.route('/_auth/logout', methods=['POST'])
def logout_handler():
    """This is what persona.js will call to sign the user
    out again.
    """
    session.clear()
    return 'OK'

@app.route('/get_swtID', methods=['GET', 'POST'])
```

```
def get_swtID():
    form = get_swtIDForm()
    return render_template('get_swtID.html',
                           title='Sign In',
                           form=form)

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
       login_user(user)
       flask.flash('logged in successfully')

       next = flask.request.args.get('next')
       if not next_is_valid(next):
          return flask.abort(400)

       return flask.redirect(next or flask.url_for('/index'))
    return render_template('login.html', form=form)

@app.route('/submit', methods=('GET', 'POST'))
def submit():
    form = MyForm()
    if form.validate_on_submit():
        return redirect('/success')
    return render_template('submit.html', form=form)

@lm.user_loader
def load_user(id):
    # user Id from Flask-Login is unicode, thats why we need to convert
    # to int before sending it to database (SQLAlchemy) pkg
    return User.query.get(int(id))

if __name__ == '__main__':
    app.run()
```

- View for my editor
  Editor is basically a web portal. It has an ability to markup a web page. User inputs the web page by typing in an URL in a text box. The web page is marked up along the lines of the data structure 'd' of

our transformation function.

Here is the backend view for inputing the URL by the user. This URL is the page that the user will edit in the editor. It will be the one with the markup.

```
@app.route('/editor', methods=['GET', 'POST'])
def my_editor():
    form = InputURLform(request.form)

    # this is activated when the form is filled by user
    if request.method == 'POST' and form.validate():
         url = form.url.data
         return redirect('showPg.html',
                           title='user_entered_page',
                           url=url)
    return render_template('inputURLform.html',
                           title='input URL',
                           form=form)
```

User inputs the URL of the page he / she wishes to browse into this page.

```
<!-- extend from base coreLayout.html -->
{% extends "coreLayout.html" %}

{% block body %}
{% from "_formhelpers.html" import render_field %}
<form method=post action="/editor">
   <dl>
       {{ render_field(form.url) }}
   </dl>
   <p> <input type=submit value=submit>
</form>
{% endblock %}
```

This is the front-end view for showcasing the page which is to be edited for markup.

```
{% extends "coreLayout.html" %}
{% block body %}
   <iframe frameborder='0' noresize='noresize' style='position: absolute; backgrou
{% endblock %}
```

- View for showing the page given by the user's URL
  Here is the code for the showall view, which is used for seeing all the sweets.

  ```
  @app.route('/showPg')
  def showURL_page():
      return render_template('showPg.html',
                             title='user_entered_page',
                             url='www.google.com')
  ```

- View for Index
  Here is the code for the index view. It also is used for "/" view.

  ```
  @app.route('/')
  @app.route('/index')
  def home_page():
      form = MyForm()
      return render_template('welcome.html', form=form)
  ```

  Here is the html code for the welcome page. It includes a patch fix.

4.2.3 my welcome page

```
<!doctype>
<head>
  <style type="text/css" rel="stylesheet">
  #inputurl {
  width:85%;
  height:35px;
  background-color:#efd;
  }
  button {
  width:12%;
  background-color:#4d9;
  height:36px;
```

```
  font-size:20px;
  font-style:italic;
}
</style>
  <title> Renarrator </title>
  <p>
   <br> this is a Sweets based web application tool
   <br> creator: Sai Gollapudi
   <br>
   <br> the purpose is to be able to do SSS
  </p>

  <script type="text/javascript">
    function wget() {
        foruri = document.getElementById("inputurl").value;
        if(foruri.substring(0,7) == "http://") {
           if(window.location.href == "http://dev.a11y.in/server/")    {
              window.open("http://dev.a11y.in/web?foruri=" + encodeURIComponent(foruri
           }
           else {
              window.open("http://127.0.0.1:5000/?foruri=" + encodeURIComponent(foruri
           }
        }
        else if (foruri.substring(0,8) == "https://") {
           window.open("http://127.0.0.1:5000/?foruri=" + encodeURIComponent(foruri));
        }
        else {
           alert("Please enter 'HTTP' protocoled URL");
        }
    }
  </script>
</head>

<body>
  <fieldset>
    <legend> Enter a URL  </legend>
  <input id="inputurl" placeholder="http://a11y.in/" />
  <button type="submit" onClick="wget()"> Get</button>
  </fieldset>
</body>
```

```
</html>
```

4.2.4 controller

4.3 support infra

4.3.1 Config.py file

Before the app is launched we need to configure the various parameters for use by the app. A config.py file has been setup for this purpose.

In this file we are configuring the authentication done by Mozilla. the code related to Persona.js is dealing with this.

We are also configuring the mongodb.

```
#to make the WTF forms in app highly secure
WTF_CSRF_ENABLED = True  #this is for cross-site request forgery prevention
SECRET_KEY = '_Aum_JaiSaiRam,SuperDOOperSecretKey_ThatUwillNOT_b_ab1E_2_GUESS?' # need

import os
basedir = os.path.abspath(os.path.dirname(__file__))

#configuration info for the Mozilla Persona authorization work
PERSONA_JS='https://login.persona.org/include.js'
PERSONA_VERIFIER='https://verifier.login.persona.org/verify'

# sqlite database related constants
SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(basedir, 'app.db') # path to our
SQLALCHEMY_MIGRATE_REPO = os.path.join(basedir, 'db_repository') # folder where we wil
```

4.3.2 $_{init_{.pyfile}}$

When invoking the package called "app" I create my app object and initialize it.

```
import os
from flask import Flask
from flask.ext.login import LoginManager
from config import basedir
from flask.ext.sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config.from_object('config')
```

```
# creation database
db = SQLAlchemy(app) # creating a db object which represents our database

# creation of login manager
lm = LoginManager()
lm.init_app(app)

# to avoid circular references we wait till application is loaded
from app import views, models
```

4.3.3 Authentication of user

- Mozilla Persona based authentication
  I am using mozilla Persona based authentication. This requires Flask as well as "requests" libraries.

  Personas require us to do some work before any requests come in the views file.

- Persona.js file

```
$(function() {
  /* convert the links into clickable buttons that go to the
     persona service */
  $('a.signin').on('click', function() {
    navigator.id.request({
      siteName: 'SweetSai App'
    });
    return false;
  });

  $('a.signout').on('click', function() {
    navigator.id.logout();
    return false;
  });

  /* watch persona state changes */
  navigator.id.watch({
    loggedInUser: $CURRENT_USER,
    onlogin: function(assertion) {
      /* because the login needs to verify the provided assertion
         with the persona service which requires an HTTP request,
         this could take a bit.  To not confuse the user we show
         a progress box */
      var box = $('<div class=signinprogress></div>')
        .hide()
        .text('Please wait ...')
        .appendTo('body')
        .fadeIn('fast');
      $.ajax({
        type: 'POST',
        url: $URL_ROOT + '_auth/login',
        data: {assertion: assertion},
        success: function(res, status, xhr) { window.location.reload(); },
        error: function(xhr, status, err) {
          box.remove();
          navigator.id.logout();
          alert('Login failure: ' + err);
        }
      });
    },
    onlogout: function() {
      $.ajax({
        type: 'POST',
        url: $URL_ROOT + '_auth/logout',
        success: function(res, status, xhr) { window.location.reload(); },
        error: function(xhr, status, err) {
          alert('Logout failure: ' + err);
        }
      });
    }
  });
});
```

- Persona Based Authentication
  Here is the code for the Mozilla's Persona based Login view. The code
  for this is derived from https://github.com/mitsuhiko/flask/blob/master/examples/persona/perso

  ```
  @app.route('/_auth/login', methods=['GET', 'POST'])
  ```

```
def login_handler():
    """This is used by the persona.js file to kick off the
    verification securely from the server side.  If all is okay
    the email address is remembered on the server.
    """
    resp = requests.post(app.config['PERSONA_VERIFIER'], data={
        'assertion': request.form['assertion'],
        'audience': request.host_url,
    }, verify=True)
    if resp.ok:
        verification_data = resp.json()
        if verification_data['status'] == 'okay':
            session['email'] = verification_data['email']
            return 'OK'
    abort(400)
```

Here is the code for the Mozilla's Persona based Logout view. The code for this is derived from https://github.com/mitsuhiko/flask/blob/master/examples/persona/perso

```
@app.route('/_auth/logout', methods=['POST'])
def logout_handler():
    """This is what persona.js will call to sign the user
    out again.
    """
    session.clear()
    return 'OK'
```

4.3.4 Macro for rendering WTF forms

I am using a macro to render the fields in the forms. here is that macro that WTF uses. It is inspired by the user manual examples of WTF.

```
{% macro render_field(field) %}
  <dt>{{ field.label }}
  <dd>{{ field(**kwargs)|safe }}
  {% if field.errors %}
    <ul class=errors>
    {% for error in field.errors %}
      <li>{{ error }}</li>
    {% endfor %}
    </ul>
  {% endif %}
  </dd>
{% endmacro %}
```

5 Installations

6 Execution

My source file needs to be executable. So I need to change the Read,Write, Execute settings of my basic Python file. Here is where I do that.

```
chmod a+x /SweetSai.py
```

7 Server Side or Back-End Development work

7.1 Important code files

I am using Model View Control architecture for this app. The Models are contained in models.py file. The views are contained in views.py file.

The app package contains forms.py and several html forms.

Config.py file is used to initialize key variables with their values.

The initial file which launches the app is SweetSai.py file.

7.2 SweetSai.py

```
#!sweetEnv/bin/python
from app import app

app.run(debug=True)
#app.run(debug=True, host='0.0.0.0', port=5001)

# for setting up the MongoDB
from flask.ext.pymongo import PyMongo

# PyMongo connects to the MongoDB server running on port 27017
# on localhost, and assumes a default database name of app.name
# (i.e. whatever name you pass to Flask).
# This database is exposed as the db attribute.
mongo = PyMongo(app)
```

7.3 forms.py file

For Authentication I am using Flask-WTF extension. I am also creating a Forms.py

```
from flask.ext.wtf import Form
from wtforms import TextField, StringField, BooleanField, validators
from wtforms.validators import DataRequired

class get_swtIDForm(Form):
   usr_name = StringField('usr_name', validators=[DataRequired()])
   remember_me = BooleanField('remember_me', default=False)

class MyForm(Form):
   usr_name = StringField('usr_name', validators=[DataRequired()])

class LoginForm(Form):
   usr_name = StringField('usr_name', validators=[DataRequired()])
   remember_me = BooleanField('remember_me', default=False)

class InputURLform(Form):
   url = StringField('url being modified', validators=[DataRequired()])
```

7.4 my HTML files

7.4.1 core or base template

There is a core template upon which various views are built (or appended). Here is that core skeleton that is elsewhere enhanced to show various other views.

```
<!DOCTYPE html>
<html>
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <link href="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.cs
       <style type="text/css">
         .container {
           max-width: 900px;
           padding-top: 10px;
         }
```

```
      h2 {color: red;}
    </style>

    <!-- will use nav-link macro to highlight the one that we are on -->
    {% from "NavMacro.html" import nav_link with context %}

<nav class="navbar navbar-inverse" role="navigation">
 <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-targe
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="/index">Home</a>
    </div>

    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
      <ul class="nav navbar-nav">
        <li class="active"><a href="/login">Login</a></li>
        <li><a href="/editor">Editor</a></li>
        <li><a href="/showPg">ShowPg</a></li>
        <li><a href="/showall">Showall </a></li>
      </ul>

      <!-- search mechanism
      <form class="navbar-form navbar-left" role="search">
        <div class="form-group">
          <input type="text" class="form-control" placeholder="Search">
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
      </form>
      //-->

      <ul class="nav navbar-nav navbar-right">
        <li><a href="#">DefineSweet</a></li>
        <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown">User<b class="
          <ul class="dropdown-menu">
```

```
            <li><a href="/login">Login</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Action 1</a></li>
            <li><a href="#">Action 2</a></li>
            <li class="divider"></li>
            <li><a href="/logout">Signout</a></li>
          </ul>
        </li>
      </ul>
    </div><!-- /.navbar-collapse -->
  </div><!-- /.container-fluid -->
</nav>
<link rel=stylesheet type=text/css href="{{ url_for('static', filename='css/style.c
<head>
   {% if title %}
      <title> SWeeTapp - {{ title }} </title>
   {% else %}
      <title> SWeeTapp </title>
   {% endif %}
</head>

<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<script src="{{ config.PERSONA_JS }}"></script>
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.1/jquery.min.js"></script>
<script>
  /* the url root is useful for doing HTTP requests */
  var $URL_ROOT = {{ request.url_root|tojson }};
  /* we store the current user here so that the persona
     javascript support knows about the current user */
  var $CURRENT_USER = {{ g.user|tojson }};
</script>
<script src="{{ url_for('static', filename='js/persona.js') }}"></script>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

<header>
   <h1>SweetSai </h1>
   <div class="authbar">
     {% if g.user %}
       Signed in as <em>{{ g.user }}</em>
       (<a href="#" class="signout">Sign out</a>)
```

```
        {% else %}
          Not signed in. <a href="#" class="signin">Sign in</a>
        {% endif %}
      </div>
   </header>

   <body>
     <div class="container">
     {% block body %}{% endblock %}
     </div>
     <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
     <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js">
   </body>

</html>
```

Here is the html content for launching the nav bar

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css"
    <style type="text/css">
      .container {
        max-width: 900px;
        padding-top: 10px;
      }
      h2 {color: red;}
    </style>

    <!-- will use nav-link macro to highlight the one that we are on -->
    {% from "NavMacro.html" import nav_link with context %}

 <nav class="navbar navbar-inverse" role="navigation">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="/index">Home</a>
    </div>

    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
      <ul class="nav navbar-nav">
        <li class="active"><a href="/login">Login</a></li>
        <li><a href="/editor">Editor</a></li>
        <li><a href="/showPg">ShowPg</a></li>
        <li><a href="/showall">Showall </a></li>
      </ul>

      <!-- search mechanism
      <form class="navbar-form navbar-left" role="search">
        <div class="form-group">
          <input type="text" class="form-control" placeholder="Search">
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
      </form>
      //-->

      <ul class="nav navbar-nav navbar-right">
        <li><a href="#">DefineSweet</a></li>
        <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown">User<b class="car
          <ul class="dropdown-menu">
            <li><a href="/login">Login</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Action 1</a></li>
            <li><a href="#">Action 2</a></li>
            <li class="divider"></li>
            <li><a href="/logout">Signout</a></li>
          </ul>
        </li>
      </ul>
    </div><!-- /.navbar-collapse -->
  </div><!-- /.container-fluid -->
</nav>
```

Here is the macro I use to ensure that my navigator bar highlights the page that I am actively on.

```
{% macro nav_link(endpoint, name) %}
{% if request.endpoint.endswith(endpoint) %}
  <li class="active"><a href="{{ url_for(endpoint) }}">{{name}}</a></li>
{% else %}
  <li><a href="{{ url_for(endpoint) }}">{{name}}</a></li>
{% endif %}
{% endmacro %}
```

Here is the content for dealing with Mozilla's Persona based authentication

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<script src="{{ config.PERSONA_JS }}"></script>
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.1/jquery.min.js"></script>
<script>
  /* the url root is useful for doing HTTP requests */
  var $URL_ROOT = {{ request.url_root|tojson }};
  /* we store the current user here so that the persona
     javascript support knows about the current user */
  var $CURRENT_USER = {{ g.user|tojson }};
</script>
<script src="{{ url_for('static', filename='js/persona.js') }}"></script>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
```

Here is code for indicating the status of user authentication

```
<div class="authbar">
  {% if g.user %}
    Signed in as <em>{{ g.user }}</em>
    (<a href="#" class="signout">Sign out</a>)
  {% else %}
    Not signed in. <a href="#" class="signin">Sign in</a>
  {% endif %}
</div>
```

7.4.2 navigation bar template

```
{% extends "coreLayout.html" %}
{% set active_page = "index" %}

{% set navigation_bar = [
   ('/', 'index', 'Index'),
   ('/login/', 'login, 'Login'),
   ('/logout/', 'logout', 'Logout),
   ('/inputurl/', 'inputurl', 'InputURL')
] -%}
{% set active_page = active_page | default('index') -%}

<ul id="navigation">
   {% for href, id, caption in navigation_bar %}
      <li {% if id == active_page %} class="active"
          {% endif %}><a href="{{ href|e }}">{{ caption|e }}</a></li>
   {% endfor %}
</ul>
```

7.4.3 how I intend to show sweets

Here is the template on which I will show the Sweets. Pls note that it is enhancing the core or base template.

```
{% extends "coreLayout.html" %}
{% block body %}
  <ul class=entries>
  <p> Here is the Raw Sweet Array: {{ sweet_array }} </p>
  <h2> here are your sweets: </h2>
  {% for swt in sweet_array %}
     <ul> ID: {{ swt.id }} </ul>
     <ul> Who: {{ swt.who }} </ul>
     <ul> Context ID: {{ swt.context_id }} </ul>
     <ul> What: {{ swt.what }} </ul>
     <ul> Where: {{ swt.where }} </ul>
     <br>
  {% else %}
    <li><em>Unbelievable. No Sweets here so far!</em>
  {% endfor %}
  </ul>
{% endblock %}
```

7.4.4 my page for getting Sweet User ID

Authentication of the user is done by Mozilla Firefox Persona utility. Here I am registering the ID that user may want to use for composing Sweets.

```
<!-- extend from base coreLayout.html -->
{% extends "coreLayout.html" %}

{% block body %}
<form action="" method="post" name="login">
     {{ form.hidden_tag() }}
     <p> Please enter your Sweet ID: <br>
         {{ form.usr_name(size=10) }}<br>
     </p>

     <p>{{ form.remember_me }} Remember Me </p>

     <p><input type="submit" value="Sign In"></p>
</form>
{% endblock %}
```

7.4.5 Submit page

Here is my submit page in HTML

```
<!--extend from base coreLayout.html -->
{% extends "coreLayout.html" %}

{% block body %}
<form method="POST" action="/">
    {{ form.hidden_tag }}
    {{ form.usr_name.label }} {{ form.usr_name(size=20) }}
    <input type="submit" value="Go">
</form>
{% endblock %}
```

7.4.6 success page

7.4.7 Here is my success page in HTML

```
<!-- extend from base coreLayout.html -->
{% extends "coreLayout.html" %}

{% block body %}
<h1>SweetSai Success page </h1>
{% endblock %}
```

7.4.8 my logout page

Here is the code for my logout pagesources

```
<!-- extend from base coreLayout.html -->
{% extends "coreLayout.html" %}

{% block body %}
<h1> this is the logOUT page </h1>
{% endblock %}
```

7.4.9 my LoginForm page

```
<!-- extend from base coreLayout.html -->
{% extends "coreLayout.html" %}

{% block body %}
<form action="" method="post" name="login">
     {{ form.hidden_tag() }}
     <p> Please enter your Sweet ID: <br>
         {{ form.usr_name(size=10) }}<br>
     </p>

     <p>{{ form.remember_me }} Remember Me </p>

     <p><input type="submit" value="Sign In"></p>
</form>
{% endblock %}
```

7.4.10 style.css file

Here is the style sheet that I use for my Mozilla Persona authorization

```
html {
    background: #eee;
}

body {
    font-family: 'Verdana', sans-serif;
    font-size: 15px;
    margin: 30px auto;
    width: 720px;
    background: white;
    padding: 30px;
}

h1 {
    margin: 0;
}

h1, h2, a {
    color: #d00;
}

div.authbar {
    background: #eee;
    padding: 0 15px;
    margin: 10px -15px;
    line-height: 25px;
    height: 25px;
    vertical-align: middle;
}

div.signinprogress {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: rgba(255, 255, 255, 0.8) url(spinner.png) center center no-repeat;
    font-size: 0;
}
```

8 Client-side or Front-End Development work

The front-end development work is being done using Angular JS framework. It is being written in JavaScript. The page that is being displayed, or the page which triggers the JavaScript is a HTML page.

The coding details of these two are discussed below.

8.1 JavaScript code for front-end

Here is the code for the basic Angular JS APP component

```
var app = angular.module('GetPage', []);
```

This is the JavaScript code for the controller for the Angular JS app that I have created.

```
//exposing expose variables and functionality to expressions and directives in Templat
app.controller('GetPageController', function ($scope, $http) {
        $scope.inputURL = "http://teststore.swtr.us/";
        $scope.show = function() {
                var returnPromise = $http.get($scope.inputURL);
                returnPromise.then(
                    //success
                    function(data, status, headers, config) {
                           console.log("Server responded: Success in getting: ", $scop
                           $scope.expression = data;
                           },
                        //error
                    function(data, status, headers, config) {
               log($scope.inputURL);
                           console.log("Server responded: Error in getting: ", $scope.
                           },
                        //progress
                        function(data, status, headers, config) {
                           console.log("Server responded: Progress in getting: ", $sco
                           });
                log($scope.inputURL);
                console.log("I created an Asynch call and am exiting the Show() functi
                };
        });
```

This function is used for logging debug messages directly to the HTML page on the browser.

```
function log(str){
  var log = document.getElementById("log")
  if (log){
        // let's be safe...
    log.innerHTML += str + "<br/>";
    }
};
```

8.2 HTML code for front-end

Here is the HTML page that gets loaded to get / show Sweets.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <link href="//fonts.googleapis.com/css?family=Roboto:100,300"
            rel="stylesheet" type="text/css" />
      <link rel="stylesheet" href={{ url_for('static', filename='css/SwtFrntStyle.css')
   </head>

   <body data-ng-app="GetPage">
      <div class="page-container" id="log" data-ng-controller="GetPageController">
         <h2> SaiGo's Sweet Page </h2>
         <div>
            input URL: <input type="url" data-ng-model="inputURL" required>
            <button class="btn" data-ng-click="show()">Show</button>
         </div>
         <div data-ng-bind-html-unsafe="expression"> </div>
      </div>
      <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.j
      <script src={{ url_for('static', filename='js/SwtFrntApp.js')   }}></script>
      <script src={{ url_for('static', filename='js/SwtFrntCntrl.js') }}></script>
      <script src={{ url_for('static', filename='js/SwtFrntLog.js')   }}></script>
   </body>

</html>
```

Here is the CSS file for the front end tool we are developing.

```
* {
  box-sizing: border-box;
  font-family: 'Roboto', Arial;
  color: #95e2aa;
}

html, body {
  height: 100%;
  width: 100%;
}

body {
  background-attachment: scroll;
  background-clip: border-box;
  background-color: rgba(0, 0, 0, 0);
  background-image: url(http://subtlepatterns.com/patterns/stardust.png);
  background-origin: padding-box;
  background-size: auto;
}

.rebel {
  color: #ff6450;
}

.achieve {
  color: #9dc9a8;
}

.bt {
  width: 40px;
  border: none;
  cursor: pointer;
  outline: none;
}

.bt.bt-rebel {
  background: #ff6450;
  color: #fff;
}

.bt.bt-achieve {
  background: #9dc9a8;
  color: #444349;
}
```

```
.txt {
  background: transparent;
  width: 100%;
  outline: none;
  padding: 10px;
  margin: 4px 0;
  border: 1px solid #9dc9a8;
  width: 90%;
  font-size: 16px;
  position: relative;
  min-height: 100%;
}

.page-container {
  width: 90%;
  margin: 0 auto;
}

.page-container h2 {
  text-align: center;
  font-weight: 900;
}

.add-todo .txt {
  padding: 10px;
}

.todo-list, .add-todo {
  list-style-type: none;
  text-align: left;
}

.todo-list li {
  padding: 10px 80px 10px 10px;
  margin: 4px 0;
  border: 1px solid #9dc9a8;
  width: 90%;
  position: relative;
  min-height: 100%;
  cursor: pointer;
}

.todo-list li .bt {
  position: absolute;
  right: 0;
  top: 0;
  height: 100%;
  transition: width 0.4s ease-in-out;
}

.todo-list li .bt:hover {
  width: 80px;
}
```

9 clipboard