Flastic – Specifications

Definition & Purpose

In with words, Flastic is a pico-framework for static website design inspired by Flask micro-framework. It provides tools for designing simple static website project using Flask-like syntax and project architecture as well as leveraging Jinja2 templating system and Bootstrap “beautyfying” capability. Additionally, Flastic aims to easy the porting to Flask if extra functionality becomes needed further down your website life cycle. Basic knowledge of Jinja2 templating and Flask1.0 microwebframework will be assumed from here on (see http://flask.pocoo.org/docs/1.0/ and http://jinja.pocoo.org/docs/2.10/). Flastic will be python 3.6+ compatible.

Project Architecture/Template

The Project Architecture Template is essentially a light version of the Flask project architecture plus a “project_builder.py” file usually named “__init__.py” in Flask. The “build-folder” is the resulting static website once “website.build()” is ran (see “Coding Pattern” below):

Project-Folder: Development phase

```
|_project_builder.py
|_views.py (optional, “view” functions can be defined in the project_builder.py too)
|_templates(folder)
|_style.css (CSS style sheet)
|_bootstrap (folder. Optional, could be replaced by a central)
|       |_bootstrap widget, java snippets
```

Build-Folder: Post-build/Pre-deployment phase

```
|_build(folder)
        |_index.html
        |_website-folder-architecture...(containing *.html and optionally static files)
        |_static(folder)
                |_staticfiles (folder, optional)
                |       |_static files and downloadables
                |_style.css (Copy from above, optional)
                |_bootstrap (folder. Copy from above, optional)
```

The idea here is that, once built, the website deployment would be reduced to copy the “build-folder” to your NGINX serving folder (e.g. /net/moli.local/export/moli20/htdocs/).

Classes, Decorators, Functions and Environment variables

They are essentially wrappers with the sole purpose of near-seamless compatibility with Flask syntax and framework.

Builder Class

Roles:
- set-up the Jinja templating environment
- create the “website-folder-architecture” and *.html files (see above)
- avoid duplicate folder and *.html by keeping track of every new instance in a dict like so {“function_name”+”var-1”+...+”var-n”: “path/filename.html”}...
- ...by leveraging the @website.route decorator (see below)

Features:
- Based on a “factory” pattern
- Flexible build: can build pages independently, deploy at different locations, point to its own bootstrap folder or a shared one, similarly for CSS style sheet.

Static Class

Roles:
- move/copy static files to their destination (“staticfiles” folder by default or anywhere else if specified) during building phase.
- avoid duplication by keeping track of every new instance in a dict like so {ii: “path/to/static_file.*”}

Features:
- based on a “factory” pattern
- all paths relative to “build-folder” for deployment flexibility
- each Static instances will have at least its path as attribute to facilitate the templating in Jinja.

@website.route Decorator

Roles:
Inspired by “@app.route” decorator, @website.route decorator is an elegant way to specify which *.html file(s) will be created and where should it (they) go.
Features:
They are two patterns for this decorator. The first partner create one *.html file:

```
@website.route(‘path/pattern1/’)
def view_1():
   # python code
   ...
   return render_templa2te("template_2.html", **context)
```

The second pattern generates len(var-1) *...* len(var-n) *html templates:

```
@website.route(‘another/path/pattern/<type:var1>/.../<type:varn>/’, var1=[val1, …, valn],
…,varn=[val1, …, valn])
def view_2(var1,…,varn):
   # Logic/Mining/etc
      …
   return render_template("template_2.html", **context)
```

url_for() Environment Method

Similar to Flask ‘static’ environment variable, the url_for is an indexing method designed to facilitate the templating by using calls like {{ url_for('static', filename='style.css') }} or {{ url_for('view_1') }} directly in the html templates.

render_template function

Similar to Flask’s render_template function, it fetches the requested template in the “templates” folder and renders it.

Coding Pattern/Template

Purposely inspired by standard Flask __init__.py file, here is the “project_builder.py” template:

```
from swask import Builder, Static, render_template

website = Builder(”build”,
                  templates=/path/to/templates/folder,
                  bootstrap=/absolute/or/relative/path,
                  css_style_sheet=/absolute/or/relative/path)

@website.route(‘path/pattern/’)
def simple_view():
   return render_template("template_1.html")

@website.route(‘other/path/pattern/’)
def view_with_data_mining():

   # Logic/Mining/etc
   context = {}
   context[‘title’] = blabla
   context[‘body’] = blabla
   …
   im1 = Static(/path/to/image, dest=/path/to/go/otherwise/goes/to/static)
   context[‘im1’] = im1
   ...
   return render_template("template_2.html", **context)
```

```
@website.route(‘another/path/pattern/<type:var1>/.../<type:varn>/’, var1=[val1, …, valn],
…,varn=[val1, …, valn])
def view_with_data_mining_and_variables(var1,…,varn):
    from random import random_function
    # Logic/Mining/etc
    context = {}
    context[‘title’] = blabla
    context[‘body’] = blabla
   context[‘value’] = random_function(var1, var2)
    …
    im = Static(“/path/to/image”+ var1, dest=”/path/to/go/otherwise/goes/to/static”+ var1)
    return render_template("template_3.html", **context)

if __name__ == "__main__":
    …

    if debug
         website.build(dest=”./”)
    else:
         website.build(dest=“/path/to/where/build/folder/goes”)

    if args[‘pages’]:
         website.build(pages=[‘simple_view’, ‘view_with_data_mining_and_variables’])
```

To be continued...