Server Side Languages

Web Design & Development

Day 7

Calendar Overview

Day 7, - Python Flask Forms, Uploads, and Hashing
Day 8, - Login and Sessions
Day 9, - Python CRUD
Day 10, - API, Review and Catch-up
Day 11, - Practical

Lecture Overview

- 7.1 Student Topic 2 Presentations
- 7.2 Lab Questions
- 7.3 Flask

Student Presentations

Topic 2

Python Language

Review of Python basics:

Whitespace (and tabs) matter - indents determine levels. No {} or ;

Import libraries to include other functions and classes

"class" to create user-defined classes

"def" for user-defined functions

Loops "for *iterator* in *collection*:" *iterator.property*

(especially, watch at the end of loops - tabs matter inside/outside loop)

Conditionals "if *truecondition*:" *statement* "else:" *otherstatement*

#comments

Flask Micro Framework

Features:

- built in development server and debugger
- integrated unit testing support
- RESTful request dispatching
- uses Jinja2 templating
- support for secure cookies (client side sessions)
- 100% WSGI 1.0 compliant
- Unicode based
- extensively documented

Framework Vocabulary

Routing - Determine which Controller to run when a URL is called.

http://site.com/login -> Login Controller, http://site.com/ -> Default Controller

Data Validation - Library to sanitize and validate user input at server side.

Ex: validate($form->useremail, EMAIL_FILTER)

Scaffolding - Creates basic structure in an application, which may include database fields and basic page with form fields.

$ rails generate controller welcome index
Would create "http://site.com/welcome" route
"welcome" controller and "index" method inside that controller
"views/welcome/index.html" view inside

Page Templating - Takes a template page and merges dynamic data at runtime. Can also often include additional templates (header, nav bar, footer).

Ex: <head><title>**<%= title %>**</title></head>, Title variable is passed in.

Web Frameworks

Pre-built programs that allow for quick development of applications by using a standard set of methods and conventions.

Frameworks allow us to standardize

- Database abstraction
- Mapping URLs (RESTful Clean URL structure)
- Validate user input
- Session management
- Reuse commonly-used functionality
- Render responses
- Prebuilt plugins, packages (such as authentication)
- Handle requests and responses in a secure way (to minimize vulnerabilities)
- Package management (installing packages or resolving dependencies)
- Data scaffolding (creating the data schema automatically in a standard way)

Content Management Systems (CMS)

Publishing, editing content

Built for easy content creation and updating by users

Usually stores content in a database

Many use templates to format pages - merge content with template to present to users

- Advantages: Easy to use and customize, SEO, Low cost, plugins
- Disadvantages: Latency (slow pages, load from DB), hard to scale, poorly-designed plugins
- Examples: WordPress, Joomla!, Drupal, SilverStripe, MediaWiki, Magnolia

Micro Frameworks

A micro framework is a simple, lightweight and extensible web framework that provides a common structure and page routing, but often by default does not include page templating, database abstraction, data input validation, scaffolding, file uploading or file manipulation, built-in authentication, and other common framework functionality. This additional functionality must be included through external libraries or through the base language itself.

Advantages: Lightweight, only load what you need when you need it. Scalability with more traffic (in theory, faster).

Disadvantages: Provides basic structure, but you have to everything you need.

Examples: Slim (PHP), Silex (PHP), Flight (PHP), Flask (Python), Bottle (Python), Express (Node), Sinatra (Ruby), Camping (Ruby)

Full Stack Frameworks

Full stack frameworks include the most common functionality (database abstraction, authentication, templating, scaffolding) without having to include or declare additional libraries or plugins.

Advantages: Most common items included, so you just call method, or use existing class. Easy to build quick applications. Scaffolding can create basic page and data structure.

Disadvantages: May consume more resources, loads a lot of extra things you might never use, or only use for one part of your application. Forces developer into framework-specific way of programming which may differ from native language.

Examples: FuelPHP (PHP), CakePHP (PHP), Laravel (PHP), Symfony2 (PHP), Django (Python), Web2Py (Python), Ruby on Rails (Ruby), Sails (Node)

Routing

Most frameworks will have a single script that handles routing users to the correct controller and view.

Ex: index.php?controller=welcome&method=index

In this example, index.php will parse the querystring parameters and then pass the script to the appropriate controller and method.

Example: User Login
//Presents user with login form with a POST action of "**validate**"
index.php?controller=login&method=**form**

//Script that handles POST from login form, and redirects to user profile
index.php?controller=login&method=**validate**

Clean URLs

The URL "http://site.com/index.php?controller=login&method=form" isn't very friendly or easy to read, so we can make it simpler.

Clean URL - http://site.com/login/form - is much easier to read.

URL Segments - Each additional "/" is a segment. Usually in the form of:
http://site.com/**controller**/**method**/data/data/slug

URL Rewriting

URL Rewriting - We can force the web server to parse Clean URLs and rewrite in a form the framework can understand.

Rewrite Rule - Take "http://site.com/**segment1**/**segment2**" incoming request, and internal to webserver only transform into: index.php?controller=**$1**&method=**$2**

Sample Apache .htaccess file

```
RewriteEngine On
RewriteBase /
RewriteRule ^(.*)$ /index.php?controller=$1&method=$2 [L]
```

Apache .htaccess - Directory-level override file that instructs Apache rules for handling incoming requests.

Install Flask on Mac

Under your Class Exercises directory, create a subdirectory "flask"
Open Terminal prompt, cd to flask directory

```
sudo easy_install pip

//pip is a package manager that downloads and installs Flask
pip install Flask
```

This has installed the libraries for Flask that we will use in our project.

Exercise 7.1: Create Hello World Flask Application

Open up your text editor, and create a file named "hello.py" in the flask project directory. Make sure to pay attention to tabs (remember Python whitespaces).

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello World!'

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

Save the file, and launch the application, then browse http://localhost:5000

From Terminal prompt in flask directory:

```
python hello.py
 * Running on http://127.0.0.1:5000/
127.0.0.1 - - [26/Feb/2014 17:48:37] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [26/Feb/2014 17:48:37] "GET /favicon.ico HTTP/1.1" 404 -
```

Importing Libraries

Flask imports Python libraries to extend the functionality of the framework.

```
from flask import Flask
  Flask #Default Flask library
  url_for #Build a URL
  request #Handle HTTP Requests
  make_response #Handle sending HTTP responses, cookies
  render_template #Jinja2 Templates
  Markup #Generate escaped HTML, or wiki-like markups
  escape #Generate escaped HTML (it not using templates)
  redirect #HTTP Redirects
  session #Support for sessions
  os #Filesystem or OS functions
  current_app #app context functions
  sqlalchemy #Database access, ORM
```

URL Routing

As in other frameworks, we need to define routes based on the URI requested (http://localhost/ or http://localhost/user/profile/10)
and HTTP Request method (GET, POST, PUT, DELETE)

We need to import the "request" library to access any request methods.

```
from flask import request
```

Route HTTP Request Example: (implied GET if not specified)

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        do_the_login()
    else:
        show_the_login_form()
```

URL Variables

Add variable segments to your URL.

Delimit variables with <variable>

Optional Convert Variable Types: int, float, path (accepts slashes)

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id
```

Request GET Parameters

To access GET parameters

Forms posted with GET method or querystring, http://localhost/login?user=bob

Use request.args.get('key')

```
searchword = request.args.get('user')
```

Request POST Parameters

To access POST form fields or GET parameters

Use request.form['keyname'] to access POST fields.

```
@app.route('/login', methods=['POST', 'GET'])
def login():
    error = None
    if request.method == 'POST':
        if valid_login(request.form['username'],
                       request.form['password']):
            return log_the_user_in(request.form['username'])
    else:
        error = 'Invalid username/password'
# the code below is executed if the request method
# was GET or the credentials were invalid
return render_template('login.html', error=error)
```

Uploads

Flask File Upload

```
from flask import request

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['the_file']
        f.save('/var/www/uploads/uploaded_file.txt')
```

Framework Vocabulary

Plugins - Modules that add specific functionality. May be included in framework or added as additional libraries.

Middleware - Software that works in conjunction with core code to add additional layers of functionality, for instance caching, interfacing to remote applications, etc. Works in conjunction with plugins (not to replace them.)

Dependency Injection - Insert objects into an app, or override default objects. In framework use, can be used to override built-in functionality, such as automatically passing views through a default templating engine.

Questions?

Lab 7: Forms, uploading, and hashing

Refer to FSO for Lab7 and Screencast 7 for this assignment