Django

Sears Merritt
CSCI 5448: Object-Oriented Analysis and Design
What is Django?

“Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.” – Django Website
History

• Began as a framework for The World Company, used to manage their news sites.

• Developers needed an efficient method for managing and publishing news articles.

• Open-sourced under BSD license in July 2005

• Django Software Foundation founded in June 2008 to continue the management and support of the framework
Django Driven Sites:
www.lawrence.com
Django Driven Sites:
www.washingtonpost.com
Django Driven Sites:
www.tabblo.com
Primary Concept and Design Pattern: Model-View-Controller
Django Model-View-Controller Overview

- **Model:** Holds data, typically in a database. Superclass is Model. Models for applications are created in Django by sub-classing the Model class. A variety of databases are supported ranging from sqlite3 to mysql that are used to store tables of model objects.

- **View:** Presents data to users using Django templates and template language. Accessed by creating html pages with Django template language. Content is dynamically rendered using a wide range of Django methods.

- **Controller:** Controls which data and views are presented. Accessed via routing requests to particular view routines. Each view routine has access to the models and additional logic to create and manipulate data. Request routing is handled through urls.py while view logic is handled through functions defined in views.py
Model

• Application models stored in models.py

• New models are subclassed from django.db.models.Model

• Models are composed of different types of fields:
  – Characters
  – Integers
  – Floats
  – Dates
  – And more!

```python
from django.db import models

class Symptom(models.Model):
    symptom = models.CharField(max_length=200)
    def __unicode__(self):
        return self.symptom

class Location(models.Model):
    city = models.CharField(max_length=200)
    state = models.CharField(max_length=200)
    latitude = models.FloatField()
    longitude = models.FloatField()
    pop = models.PositiveIntegerField(default=1)
    def __unicode__(self):
        return ','.join((self.city, self.state))

class UserReport(models.Model):
    symptom = models.CharField(max_length=2400)
    latitude = models.FloatField()
    longitude = models.FloatField()
    timestamp = models.DateTimeField()
    location = models.ForeignKey(Location)
```
View Controllers

• View controllers are invoked by routing requests to them via urls.py.
• Views.py defines the set of controllers associated with each rule in urls.py
• Each view can:
  – Accept HTTP messages
  – Handle input and output
  – Generate dynamic content for rendering by the template language
View Controller Routing

1. URL sent to urls.py
2. Matched with the blog regex
3. Routed to the mapped function, blog, in views.py

```
def about:
def index:
def blog:
```

views.py

```
urls.py
```

1. `www.test.com/blog`

2. `r('^$', 'index')`
3. `r('^about/$', 'about')`
   `r('^blog/$', 'blog')`
View Controllers – urls.py

- List of regular expressions matching urls with view routines

- Left hand side is regex, right side is function in views.py
View Controllers – views.py

• Each view is defined as a function
• Each function receives a request
• A request contains dictionaries of data
• Using requests, data can be accessed easily for saving or generating dynamic content
• Each view returns an HTTP response ranging from redirects to dynamically rendered HTML pages
View Controllers – views.py

```python
from django.template import Context, loader
from django.template import RequestContext
from django.shortcuts import render_to_response, get_list_or_404, redirect
from website.socialhealth.models import Symptom, UserReport, Location

def index(request):
    return render_to_response('index.html', context_instance=RequestContext(request))

def report(request):
    sympList = get_list_or_404(Symptom)
    return render_to_response('report.html', {'sympList': sympList}, \
                               context_instance=RequestContext(request))

def results(request):
    return render_to_response('results.html', context_instance=RequestContext(request))

def result(request):
    if request.POST:
        r = UserReport()
        r.email = request.POST.get('email')
        r.symptom = r.symptom.lstrip(',')
        r.timestamp = datetime.datetime.fromtimestamp(time.time())
        r.longitude = request.POST.get('longitude')
        r.latitude = request.POST.get('latitude')
        r.location = Location.objects.get(city=cluster((float(r.latitude), float(r.longitude))))
        r.save()
        return redirect('/')
    else:
        return redirect('/')
```
Views – Templates and Template Language

• Views are essentially html pages with embedded template language

• Template language is Django specific

• Django renders template language using data received from the view controller
Template Language Syntax

Template language consists of intuitively tagged python statements

{% for item in list %}
    {{ item }}
{% endfor %}

{% if item %}
    {{ item }}
{% endif %}

This language is embedded in html pages and gets rendered when a view controller passes data and the page to the renderer
Views – Template and Template Language

```html
if sympList %}
  form id="form" action="../result/" method="post">
    csrf_token %}
    for symptom in sympList %}
      <input type="checkbox" name="symptom{{ forloop.counter }}" id="symptom{{ forloop.counter }}" value="{{ symptom }}" />
      <label for="symptom{{ forloop.counter }}">{{ symptom }}</label><br />
    endfor %}
    <input type="hidden" name="latitude" id="latitude" value="" />
    <input type="hidden" name="longitude" id="longitude" value="" />
    <input type="hidden" name="email" id="email" value="null" />
    <input type="submit" name="submit" id="submit" value="Submit" />
  form>
else %}
  No symptoms are available.</p>
endif %}
```
Django Projects and Applications

• Django organizes MVC implementations into projects.

• Projects aggregate applications.

• Each application has its own models, views, and controllers.
Project Configuration – settings.py

• Holds the configuration settings for a project
  – Database configuration
  – Media directory (css, javascript, etc.)
  – Template directory (html files)
  – Django middleware (such as authentication and CSRF)
  – Django modules
  – Project applications
Creating a Django Website

• Create a project
  – django-admin.py startproject myproject
• Create an app
  – django-admin.py startapp myapp
• Configure settings.py
• Configure the MVC implementation for myapp
  – urls.py
  – views.py
  – models.py
• There is an excellent tutorial here.
The Life of a Request

1. HTTP packet arrives at web server
2. WSGI routes packet to Django core
3. Django core:
   1. parses message
   2. routes the message to the appropriate view handler via urls.py
   3. Creates a request object
4. Controller handler parses request and performs appropriate logic via handlers in views.py
5. Controller hands resulting data to the template system
6. Template system pulls HTML, fills with content from view handler, sends response
Django in Action
References

• Django Website

• Django Wikipedia

• Social Health Website