Django: Views, Templates, and Sessions

CS 370 SE Practicum, Cengiz Günay

(Some slides courtesy of Eugene Agichtein and the Internets)
Warm-up project:

- Anything you are missing?
- Progress or problems? Post on Piazza.
- Still due Feb 13th
Agenda

Warm-up project:
- Anything you are missing?
- Progress or problems? Post on Piazza.
- Still due Feb 13th

Today:
- Django Views, Templates, and Sessions
django
How Django processes a request

1. **Determine** the root **URLconf** module to use. Ordinarily, this is the value of the **ROOT_URLCONF** setting, can be changed.

2. **Load URLconf** and looks for variable **urlpatterns**. This should be a Python list, in the format returned by the function `django.conf.urls.urlpatterns()`.

3. **Iterate** through each URL pattern, in order, and stop at the first one that matches the requested URL. (REGEX)

4. If one of the regexes matches, **import** and **call** the given **view**, which is a Python **function**. The view gets passed an **HttpRequest** as its **first argument** and any values captured in the **regex** as remaining arguments.

5. If no regex matches, or if an exception is raised during any point in this process, invoke an appropriate **error-handling view**.
URL Processing Example: Fixed for 1.4

- https://docs.djangoproject.com/en/1.4/intro/tutorial03/
- mysite/urls.py (main configuration file for site)

```python
urlpatterns = patterns(''
    url(r'^polls/$', 'polls.views.index'),
    url(r'^polls/(?P<poll_id>\d+)/$', 'polls.views.detail'),
    url(r'^polls/(?P<poll_id>\d+)/results/$', 'polls.views.results'),
    url(r'^polls/(?P<poll_id>\d+)/vote/$', 'polls.views.vote'),
    url(r'^admin/', include(admin.site.urls)),
)
```
• When somebody requests a page from your Web site -- say, "/polls/23/"
  – Django loads the polls Python module
  – finds variable named urlpatterns and traverses the regular expressions in order
  – First matching regex: \r'^polls/(\?P<poll_id>\d+)/$'\r
    • loads the function detail() from polls/views.py
    • calls that detail() function like as:
      detail( request=<HttpRequest object> , poll_id='23' )

Notes:
poll_id='23' comes from \(\?P<poll_id>\d+\)
regular expressions do not search GET and POST parameters
Views

from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world.")
def index(request):
    return HttpResponse("Hello, world.")

def detail(request, poll_id):
    return HttpResponse("looking at poll %s." % poll_id)

def results(request, poll_id):
    return HttpResponse("looking at results of poll %s." % poll_id)

def vote(request, poll_id):
    return HttpResponse("voting on poll %s." % poll_id)
from polls.models import Poll
from django.http import HttpResponse
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    output = ', '.join([p.question for p in latest_poll_list])
    return HttpResponse(output)
Views, cont’d 4

```python
from django.template import Context, loader
from polls.models import Poll
from django.http import HttpResponse

def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    t = loader.get_template('polls/index.html')
    c = Context({
        'latest_poll_list': latest_poll_list,
    })
    return HttpResponse(t.render(c))
```

loads the template called "polls/index.html" and passes it a context. The context is a dictionary mapping template variable names to Python objects.
{% if latest_poll_list %}
<ul>
{% for poll in latest_poll_list %}
    <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
{% endfor %}
</ul>
{% else %}
<p>No polls are available.</p>
{% endif %}
Django Templates ("Web Design")

• A Django template is a string of text to separate the presentation of a document from its data.

• A template defines **placeholders** and **basic logic** (**template tags**) that regulate how the document should be displayed.

```html
<html> <head><title>Ordering notice</title></head> <p>Dear {{ person_name }},</p>
<p>Thanks for placing an order from {{ company }}. It's scheduled to ship on {{ ship_date|date:"F j, Y" }}.</p>
```
Template basics

• **variable**: Any text surrounded by a pair of braces (e.g., ```{{ person_name }}``` ) is a *variable*. This means “insert the value of the variable with the given name.”

• **tag**: Any text that’s surrounded by curly braces and percent signs (e.g., ```{% if ordered_warranty %}``` ) is a *template tag*.

• **filter**: alter the formatting of a variable. example ```{{ ship_date|date:"F j, Y" }}``` passes the `ship_date` variable to the date filter, giving the date filter the argument "F j, Y".
  
  – The date filter formats dates in a given format, as specified by that argument. Filters are attached using a pipe character (|), as a reference to Unix pipes.

http://www.djangobook.com/en/2.0/chapter04/
Views 5: shortcut

from django.shortcuts import render_to_response
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    return render_to_response('polls/index.html',
                               {'latest_poll_list': latest_poll_list})
Detail view

• Switch to code

https://docs.djangoproject.com/en/1.4/intro/tutorial04/
Good Design: Including other URLConfs

• Your urlpatterns can "include" other URLconf modules.
  – Allows **modular** sites (good design)
    • urlpatterns = patterns(''
      (r'^wiki/', include('blog.wiki.urls'))
    )
  – Note But: (wiki/urls.py):
    • (r'^(?P<page_name>[^/]+)/$', 'blog.wiki.views.view_page')

• More details:
Design Pattern: Include URLs

• There should be one `urls.py` at the project level, and one `urls.py` at each app level. The project level `urls.py` should include each of the `urls.py` under a prefix:

• [http://agiliq.com/books/djangodesignpatterns/urls.html](http://agiliq.com/books/djangodesignpatterns/urls.html)
Coding Order

• URLpattern first and the views second, or the view first, then the URLpattern second. Which is better?

• “Top-down” design (standard OOP):
  – write all of the URLpatterns for your application at the same time, at the start of your project, and then code up the views. This has the advantage of giving you a clear to-do list, and it essentially defines the parameter requirements for the view functions you’ll need to write.

• “Bottom-up” design (standard hack):
  – write the views first, and then anchor them to URLs afterward.
Example: Custom 404 template

• **Http response 404** (page not found)
• built-in: `django.views.defaults.page_not_found()`
• Uses 404.html template in the root of your template directory.
  – `~cs370000/blog/templates/404.html`
• If **DEBUG** is set to True, 404.html template will never be rendered): **traceback** will be displayed instead.
• 404 view is also called if Django doesn't find a match after checking every regex in URLconf.
Design Pattern: Templates

• There should be one base.html at the project level, and one base.html at each of the app levels. The app level base.html should extend the project level base.html

• The templates for an app should be available as appname/template.html. So the templates should be physically located at either:
  – project/templates/app/template.html
  – project/app/templates/app/template.html

• http://agiliq.com/books/djangodesignpatterns/templates.html
HTTP Sessions

• HTTP is a stateless protocol. A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests. However web applications often implement states or server side sessions using one or more of the following methods:
  – HTTP cookies.
  – Query string parameters, for example, /index.php?session_id=some_unique_session_code.
  – Hidden variables within web forms.
Server-side Sessions

• Django provides a **session framework**

• Lets you store and retrieve arbitrary data on a per-site-visitor basis. It can store data on the server side and abstracts the sending and receiving of cookies.

• **Server-side a.k.a. Database:**
  – Cookies contain a session ID – not the data itself

• **Client-side:**
  – Data is contained in a cookie
Django Sessions Framework

- Session framework lets you store and retrieve arbitrary data on a per-site-visitor basis.
- It stores data on the server side and abstracts the sending and receiving of cookies.
- Cookies use only a hashed session ID, protecting you from most of the common cookie problems.
- [https://docs.djangoproject.com/en/1.4/topics/http/sessions/](https://docs.djangoproject.com/en/1.4/topics/http/sessions/)
Using Sessions in Views

Available via the `request.session` Dictionary:

```python
def post_comment(request, new_comment):
    if request.session.get('has_commented', False):
        return HttpResponse("You've already commented."")
    c = comments.Comment(comment=new_comment)
    c.save()
    request.session['has_commented'] = True
    return HttpResponse('Thanks for your comment!')
```
Using Sessions in Views

Available via the `request.session` Dictionary:

```python
def post_comment(request, new_comment):
    if request.session.get('has_commented', False):
        return HttpResponse("You've already commented.")
    c = comments.Comment(comment=new_comment)
    c.save()
    request.session['has_commented'] = True
    return HttpResponse('Thanks for your comment!')
```

```python
def login(request):
    m = Member.objects.get(username=request.POST['username'])
    if m.password == request.POST['password']:
        request.session['member_id'] = m.id
        return HttpResponse("You're logged in.")
    else:
        return HttpResponse("Your username and password don't match.")
```

HTTP Session Token

- A session token is a unique identifier that is sent to identify the current interaction (browser) session.
- The client usually stores and sends the token as an HTTP cookie and/or sends it as a parameter in GET or POST queries.
- Use session tokens to have all session data stored on the server (usually in a database to which the client does not have direct access) linked to that identifier.
Cookie-based sessions

• To use cookies-based sessions, set the SESSION_ENGINE setting to "django.contrib.sessions.backends.signed_cookies".

Add django.contrib.sessions to INSTALLED_APPS inside settings.py

To create database tables, do:
python manage.py syncdb
### Session Info in the Database

```sql
<table>
<thead>
<tr>
<th>#</th>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Default</th>
<th>Extra</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>session_key</td>
<td>varchar(40)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td>Change Drop More</td>
</tr>
<tr>
<td>2</td>
<td>session_data</td>
<td>longtext</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td>Change Drop More</td>
</tr>
<tr>
<td>3</td>
<td>expire_date</td>
<td>datetime</td>
<td></td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td>Change Drop More</td>
</tr>
</tbody>
</table>
```

Check All / Uncheck All With selected: Browse Change Drop Primary Unique

Index
## Session Info in the Database

### Table: Django Session Table

<table>
<thead>
<tr>
<th>#</th>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Default</th>
<th>Extra</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>session_key</td>
<td>varchar(40)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>session_data</td>
<td>longtext</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>expire_date</td>
<td>datetime</td>
<td></td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **session_key**: Stores unique session identifiers.
- **session_data**: Stores session data.
- **expire_date**: Stores the expiration date of the session.

### Session Entries

<table>
<thead>
<tr>
<th>session_key</th>
<th>session_data</th>
<th>expire_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1d33b9c3632bda5994f8e153614c15f6</td>
<td>YTQ5NzVjm2Y4MGE1MGV/3ZGU3Mjc3YQzNmE3NGZhNzhmOWU2OD...</td>
<td>2014-02-20 05:58:44</td>
</tr>
<tr>
<td>29cf1da3213bd6423e891d5207c6f</td>
<td>MzhhNJNjAjYqMgyOJbK3BbYmZwZjHgFjNDZjGjZg...</td>
<td>2014-02-18 21:59:35</td>
</tr>
<tr>
<td>4f05e8b8b42e4c6caf28296e7702d6d</td>
<td>Y2m0ZjiliMWMwZDE0Zjk2YjMwZGExM2I3NGZhNmQxYzg0Nm...</td>
<td>2014-02-18 21:59:34</td>
</tr>
<tr>
<td>6b1b78f9eb9daa1267a66e7ef25832e</td>
<td>YTQ5NzVjm2Y4MGE1MGV/3ZGU3Mjc3YQzNmE3NGZhNzhmOWU2OD...</td>
<td>2014-02-20 19:19:24</td>
</tr>
<tr>
<td>7f3df9f25be9e14c697e50274fb56ef</td>
<td>Y2m0ZjiliMWMwZDE0Zjk2YjMwZGExM2I3NGZhNmQxYzg0Nm...</td>
<td>2014-02-20 18:36:16</td>
</tr>
<tr>
<td>81ccaa4a1d2d4e6b15b314f05cc0b68</td>
<td>NDjmZDk0ZjRkNzc4MTAxEwMDA5MzhmYzc4MjUsYzAzNDQ4MW...</td>
<td>2013-02-12 17:07:57</td>
</tr>
<tr>
<td>8f456344f8ea922fd41ec98653c6f8</td>
<td>Y2m0ZjiliMWMwZDE0Zjk2YjMwZGExM2I3NGZhNmQxYzg0Nm...</td>
<td>2014-02-18 21:59:14</td>
</tr>
<tr>
<td>9f03620fd91844c9a9c0c29b9e50d0</td>
<td>NDjmZDk0ZjRkNzc4MTAxEwMDA5MzhmYzc4MjUsYzAzNDQ4MW...</td>
<td>2013-02-12 16:09:14</td>
</tr>
<tr>
<td>b04390a08f6b3845487da2590c2da0</td>
<td>YTQ5NzVjm2Y4MGE1MGV/3ZGU3Mjc3YQzNmE3NGZhNzhmOWU2OD...</td>
<td>2014-02-18 21:59:28</td>
</tr>
<tr>
<td>c19ce55c8a2359eb25bb4a3f4b661da</td>
<td>Y2m0ZjiliMWMwZDE0Zjk2YjMwZGExM2I3NGZhNmQxYzg0Nm...</td>
<td>2014-02-18 21:58:03</td>
</tr>
</tbody>
</table>

---

**CS 370, Günay (Emory) | Django Views/Templates | Spring 2014 | 7 / 7**
Cookies are SIGNED

- Can be read by client
- Can set almost any kind of data if can be serialized to a string (and parsed later)
- A web browser is expected to be able to store at least 300 cookies of four kilobytes each, and at least 20 cookies per server or domain.
Onwards and upwords...

• Django book:
  – http://www.djangobook.com/en/2.0/

• Common mistakes/gotchas:
  – https://code.djangoproject.com/wiki/NewbieMistakes

• To be continued...