Django Views

COMP 8347

Slides prepared by Dr. Arunita Jaekel
arunita@uwindsor.ca

Django Views

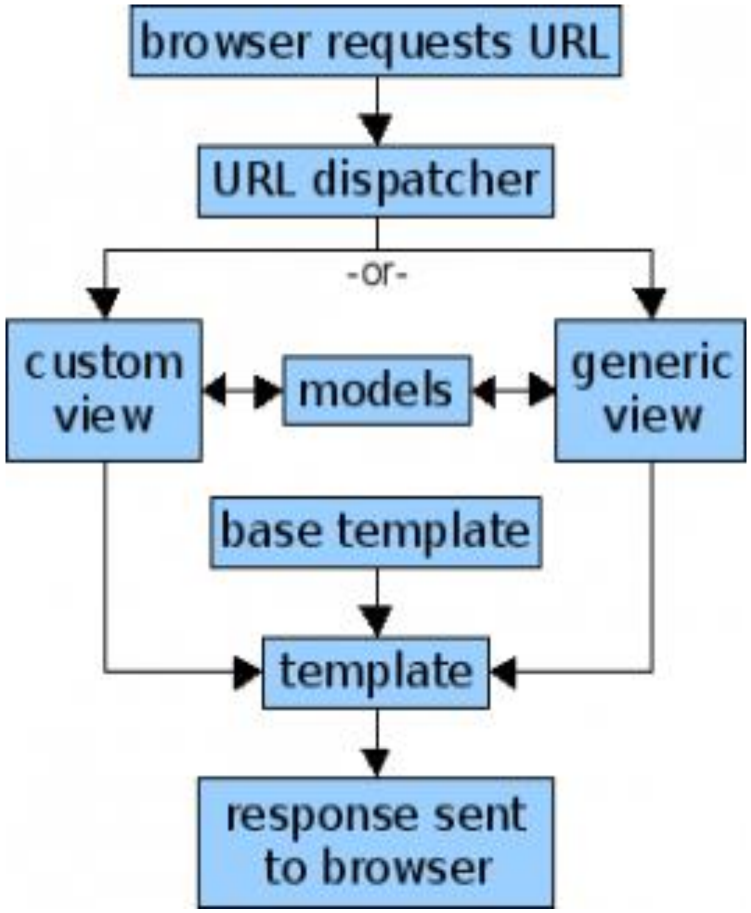
- Topics
  - URLS
  - HTTP Objects
    - Request
    - Response
  - Views
    - Custom views
    - Generic views (if time permits)

Review MTV Architecture

– Represent data organization; defines a table in a database.

– Contain information to be sent to client; help generate final HTML.

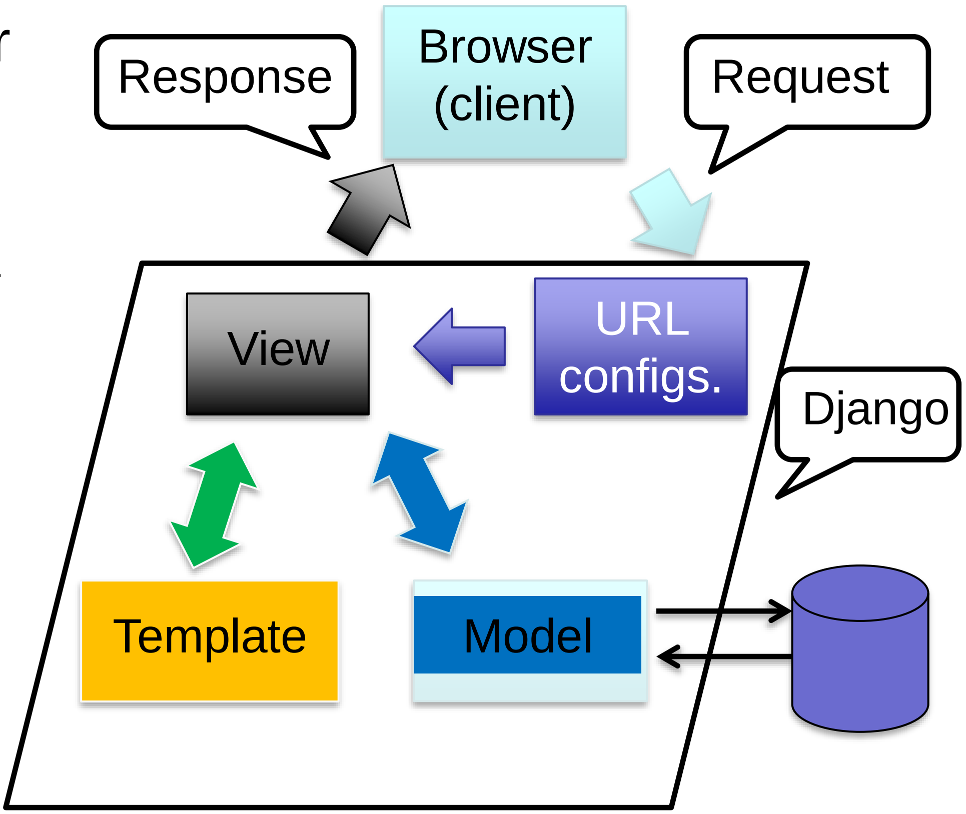
– Actions performed by server to generate data.



www.tikalk.com/files/intro-to-**django**.**ppt**

Choosing a View (Function)

- Django web pages and other content are delivered by views.
  - Each view is represented by a simple Python function (or method)
- Django chooses a view by examining the requested URL
  - Only looks at the part of URL after the domain name.
  - Chooses view that 'matches' associated URL pattern.



URLconf

- *URLconf (URL configuration): maps between URL path expressions to Python functions (your views).*
- *urlpatterns*: a sequence of Django **paths**

  Example: **path(r'', views.index, name='index'),**
- URL patterns for your app/project specified in corresponding ***urls.py*** file.

Sample urls.py

mysite/urls.py

```
from django.urls import include, path
from django.contrib import admin
urlpatterns = [
   path(r'admin/', admin.site.urls),
   path(r'myapp/', include('myapp.urls')),
]
```

myapp/urls.py

```
from django.urls import path
from myapp import views

app_name = ‘myapp’
urlpatterns = [
   path(r'', views.index, name='index'),
]
```

- include(*module*, *namespace=None*)
  - urlpatterns can “include” other URLconf modules.
  - This “roots” a set of URLs below other ones
  - When Django encounters include():
    - it chops off part of the URL matched up to that point
    - sends the remaining string to the included URLconf for further processing
  - Always use include() when including other URL patterns
    - Only exception in admin.site.urls

path()

- Syntax:
  - path(route, view, kwargs=None, name=None)
  - ***route***: a string that contains a URL pattern
    - may contain angle brackets (like **<username>**) to capture part of the URL and send it as a keyword argument to the view.
    - angle brackets may include a converter specification (like the int part of <int:section>) which limits the characters matched and may also change the type of the variable passed to the view
    - Django starts at the first **path**, compares requested URL against each route until it finds one that matches.
      - Does not search GET and POST parameters, or domain name

path()

- Syntax:
  - **path(route, view, kwargs=None, name=None)**
  - ***view***: after finding match, Django calls specified view function, with
    - HttpRequest object as the first argument and
    - any “captured” values from the regular expression as other arguments.
  - ***kwargs*** : can pass additional arguments in a dict, to view function
  - ***name***: lets you refer to URL unambiguously from elsewhere in Django

path()

- Examples:

```
from django.urls import include, path

urlpatterns = [
    path('index/', views.index, name='main-view'),
    path('bio/<username>/', views.bio, name='bio'),
    path('articles/<slug:title>/', views.article,
                  name='article-detail'),
    path('articles/<slug:title>/<int:section>/', views.section,
                 name='article-section'),
    path('weblog/', include('blog.urls')),
    ...
]
```

* A **slug** is a short label for something, containing only letters, numbers, underscores or hyphens. They're generally used in URLs.

re_path()

- Syntax:
  - re_path(route, view, kwargs=None, name=None)

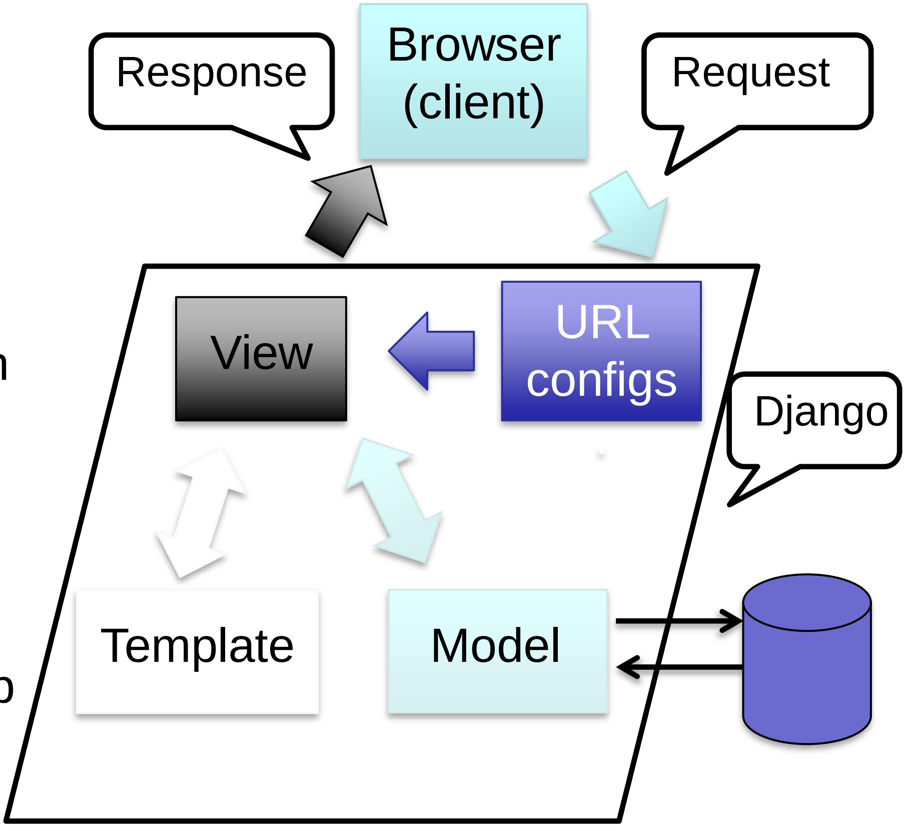
```
urlpatterns = [
    re_path(r'^index/$', views.index, name='index'),
    re_path(r'^bio/(?P<username>\w+)/$', views.bio, name='bio'),
    re_path(r'^weblog/', include('blog.urls')),
    ...
]
```

URL Matching Examples

```
from django.conf.urls import patterns, path
from myapp import views

urlpatterns = [
# ex: /myapp/

# ex: /myapp/5/

# ex: /myapp/5/results/

]
```

Web Application Flow

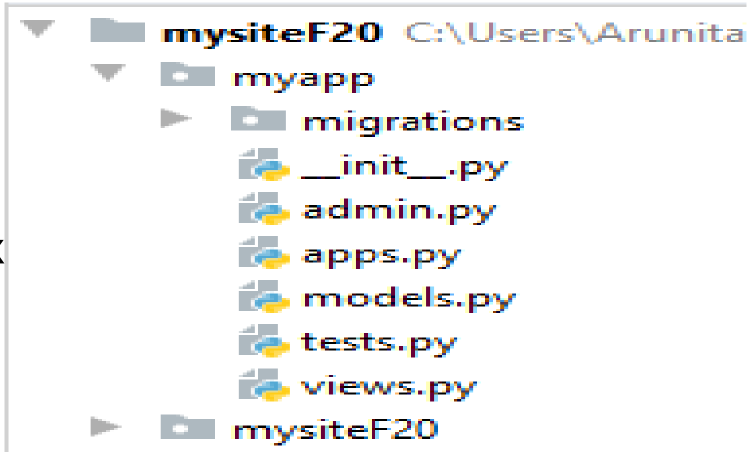
- HTTP request arrives at web server
- Web server passes request to Django
- Django creates a request object
- Django consults URLconf to find right view function
  - – Checks url against each regex/path
- View function is called with request object and captured URL arguments
- View creates and returns a response object.
- Django returns response object to web server.
- Web server responds to requesting client.

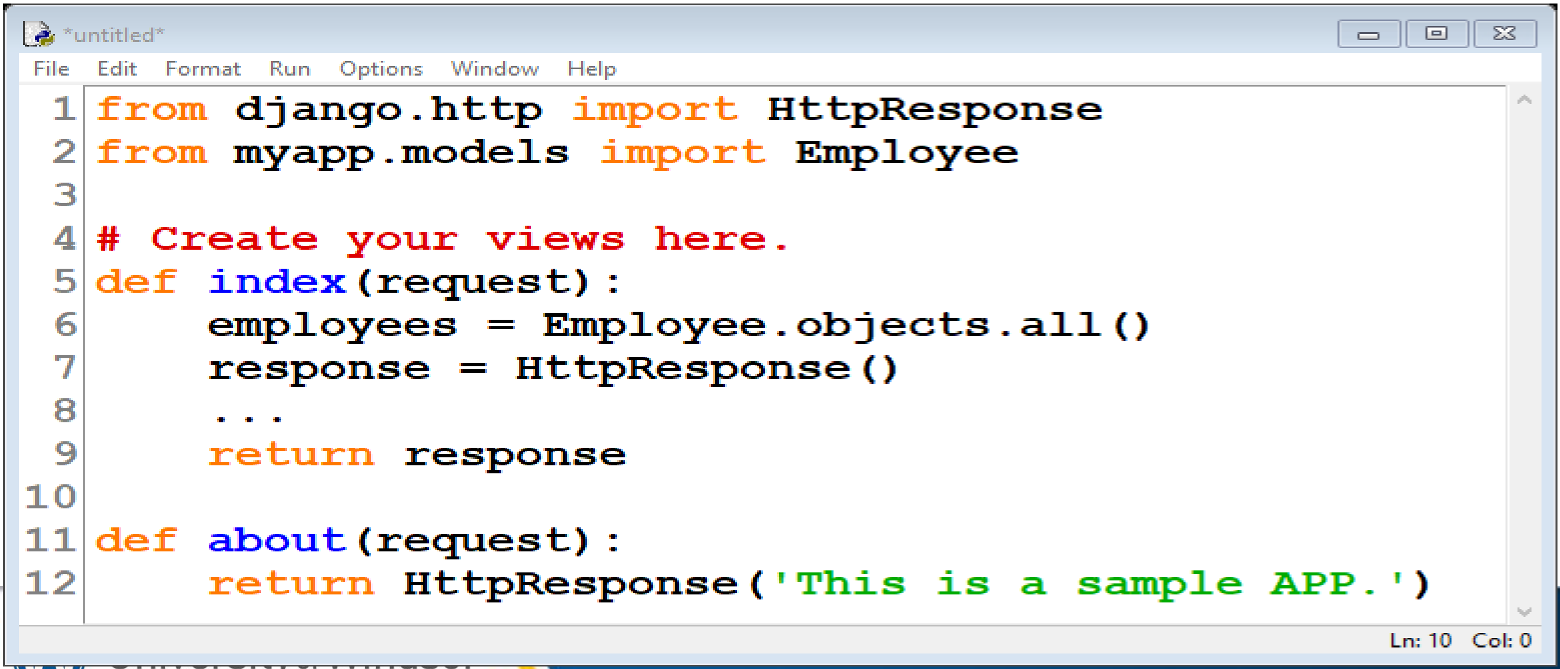


Sample urls.py

myapp/urls.py

```
from django.conf.urls import path
from myapp import views
urlpatterns = [
   path(r'', views.index, name='index
   path(r'about/', views.about,
   name=about'),
]
```


```
mysiteF20  C:\Users\Arunita
  myapp
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  mysiteF20
```



```
from django.http import HttpResponse
from myapp.models import Employee

# Create your views here.
def index(request):
    employees = Employee.objects.all()
    response = HttpResponse()
    ...
    return response

def about(request):
    return HttpResponse('This is a sample APP.')
```

Views

- ***View function*: A Python function - takes a Web request, returns a Web response.**
  - called view for short
  - response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image . . .
  - provide nearly all the programming logic
    - perform CRUD operations
  - can reside anywhere in your Python path
    - convention is to put views in a file called **views.py**, placed in your project or application directory

A Simple View

```
from django.http import HttpResponse
import datetime
def current_datetime(request):
   now = datetime.datetime.now()
   html = "<html><body>It is now {0}.</body></html>".format(now)
   return HttpResponse(html)
```

- Import the class HttpResponse and Python’s datetime library.
- define a function called current_datetime
  - view function taking HttpRequest object (typically named request) as its first parameter.
  - returns HttpResponse object with generated response
  - view function can have any name. [1]

Request Objects

- *HttpRequest*: An object with a set of attributes representing raw HTTP request
  - *GET*: An attribute of HttpRequest Object
    - represented as a Python dict subclass QueryDict.
    - GET parameters passed as URL string, but not part of URL itself;do not define a separate resource (view)
    - Example: for the URL /userinfo/ can point to specific user: /userinfo/?name=‘John Smith’

username = request.GET[‘name’]

University of Windsor

HttpRequest Attributes

– *POST*: An attribute of HttpRequest Object
  - represented as a QueryDict.
  - POST parameters are not part of URL
  - often generate by and HTML form; when user submits form, URL is called with POST dict containing form fields.
  - Example: if there is a form field ‘name’ and the user enters ‘John’

    request.POST[‘name’] will return ‘John’

– *COOKIES*: Another dict attribute; exposes HTTP cookies stored in request.

Other Attributes

- *path*: portion of URL after domain
- *method*: specifies which request method was used – ‘GET’ or ‘POST’
- *FILES*: contains information about any files uploaded by a file input form field.
- *user*: Django authentication user; only appears if Django’s authentication mechanisms activated.
- *sessions*: contains session as read from db based on users session cookie; can be written to also.
    - write saves changes back to db, to be read later.

Response Objects

- View functions return a HttpResponse object. Important attributes:
  - HttpResponse.status_code : The HTTP status code for the response
  - HttpResponse.content : A bytestring representing the content; usually a large HTML string.
  - can be set when creating a response object
    - response = HttpResponse("<html>Hello World</html")
  - can be set using write method (like a file)
    - response = HttpResponse()
    - response.write("<html>")
    - response.write("Hello World")
    - response.write("</html>")

Response Objects

- Setting HTTP headers:
  - Treat response object as a dictionary.
  - ‘key/value’ pairs correspond to different headers and corresponding values.
    - HTTP header fields cannot contain newlines.
  - Example:

```
response = HttpResponse()
response[“Content-Type”] = “text/csv”
response[“Content-Length”] = 256
```

View Functions

```
def index(request):
    books = Book.objects.all() [:10]
    response = HttpResponse()
    heading1 = '<p>' + 'List of books: ' + '</p>’
    response.write(heading1)
    for book in books:
        para = '<p>' + str(book.id) + ':  ' + str(book) + '</p>'
        response.write(para)
    return response
```

View Functions

```
def about(request):
    return HttpResponse('Sample Website')

def detail(request, book_id):
    book = Book.objects.get(id=book_id)
    response = HttpResponse()
    title = '<p>' + book.title+ '</p>'
    author = '<p>' + str(book.author)+ '</p>'
    response.write(title)
    response.write(author)
    return response
```

HttpResponse Subclasses

- Django provides HttpResponse subclasses for common response types.
  - *HttpResponseForbidden*: uses HTTP 403 status code
  - *HttpResponseServerError*: for HTTP 500 or internal server errors
  - *HttpResponseRedirect*: the path to redirect to (required 1st agrument to the constructor)
  - *HttpResponseBadRequest*: acts like HttpResponse, but uses a 400 status code
  - *HttpResponseNotFound*: acts like HttpResponse, but uses a 404 status code

Common View Operations

1. ***CRUD***: **C**reate, **R**ead (or Retrieve), **U**pdate and **D**elete
2. Load a template
3. Fill a context
4. Return HttpResponse object
   – with result of the rendered template

University of Windsor

Summary

- Choosing a view
  - URLs, patterns
- Request and response objects
  - HttpResponse subclasses
- Views
  - Common operations - CRUD

References

- [1] https://docs.djangoproject.com/en/3.0/topics/http/views/