The View layer

Web Applications and Services

Spring Term

Naercio Magaia

UNIVERSITY
OF SUSSEX

Contents

UNIVERSITY OF SUSSEX

Creating a Django app

- Setting up a virtual environment

  ```
  ...\> py -m venv d-env
  ```

- Activate the environment

  ```
  ...\> d-env\Scripts\activate.bat
  ```

- Install Django

  ```
  (d-env)...\> py -m pip install Django
  ```

- (Optional) Colored terminal output

  ```
  (d-env)...\> py -m pip install colorama
  ```

- Checking Django version

  ```
  ...\> py -m django –-version
  ```

- Creating a project

  ```
  ...\> django-admin startproject myproj
  ```

- Checking what's inside startproject directory

- ```
  ...\> tree /f myproj
  ```

```
myproj/
    manage.py
    myproj/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

UNIVERSITY OF SUSSEX
```

Creating a Django app

- Launching the development server

```
...\> py manage.py runserver
```

- Changing the port

```
...\> py manage.py runserver 8080
```

- Creating the mysite app

```
...\> py manage.py startapp mysite
```

- Checking what's inside mysite directory

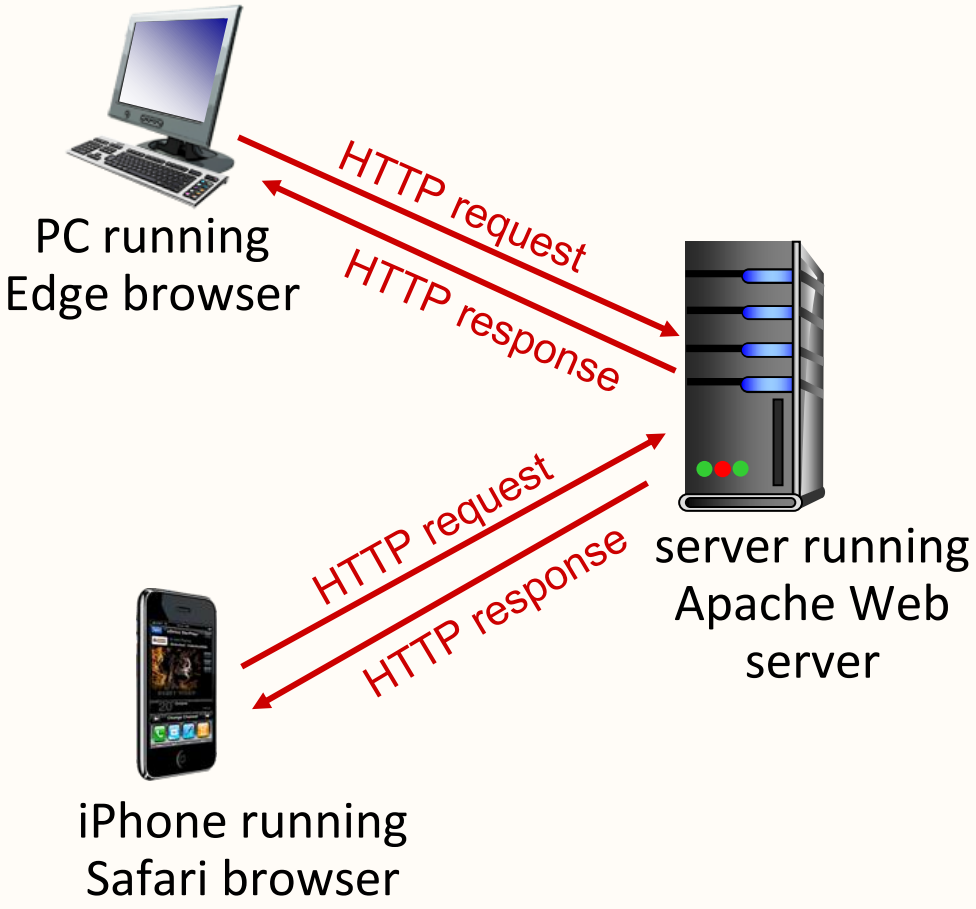
```
...\> tree /f mysite
```

```
mysite/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

UNIVERSITY OF SUSSEX

Views concept

- The concept of “views” encapsulate the logic responsible for processing a user’s request and for returning the response.
- This response can be anything. For example, the HTML contents of a web page, or a redirect, or a 404 error, or an XML document, or an image.
- The convention is to put views in a file called **views.py**, placed in your project or application directory



UNIVERSITY OF SUSSEX

Types of Views

- There are two types of views:
  - Function-based Views, which are written using functions in Python that receive as an argument an *HttpRequest* object and return an *HttpResponse* object
  - Class-based Views, which is an alternative way to implement views as Python objects instead of functions.
- Class-based views have the following advantages over function-based ones:
  - Organization of code related to specific HTTP methods (e.g., GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
  - Object oriented techniques, e.g., multiple inheritance, allow factoring code into reusable components.

A Simple View

- A *view* is just a Python function that takes an *HttpRequest* as its first parameter and returns an instance of *HttpResponse*

Importing the *HttpResponse* class of the *django.http* module

```
from django.http import HttpResponse
```

The view function *hello* takes *request* as a parameter

```
def hello(request):
    html = "<html><body>Hello World!</body></html>"
    return HttpResponse(html)
```

The view returns an *HttpResponse* object that contains the generated response.

How Django finds this function?

UNIVERSITY OF SUSSEX

URLConf

- The URL Configuration (URLConf) module is created when designing URLs for web applications.
- It is a mapping between URL path expressions to Python view functions. It can
  - be as short or as long as needed
  - reference other mappings
  - be constructed dynamically by being pure Python code
- The module *django.urls* contains functions (e.g., *path* and *include*) to use in URLConfs

UNIVERSITY OF SUSSEX

URLConf

- It searches against the requested URL, as a normal Python string.
  - This does not include GET or POST parameters, or the domain name.
- It looks for `comments/` in a request to
  - `https://www.webapps.com/comments/`
  - `https://www.webapps.com/comments/?page=2`
- It doesn't look at the request method, i.e., POST, GET, HEAD, etc., are routed to the same function for the same URL

URLConf

- Use angle brackets to capture a value from the URL.
  - For example, use `<int:name>` to capture an integer parameter.
    - If a converter isn't included, any string, excluding a / character, is matched.
- It is not necessary to add a leading slash since every URL has that.
  - For example, it's `comments` and not `/comments`.

A Sample URLConf

```
from django.urls import path
```

Importing the *include* and *path* classes from the *django.urls* module

```
urlpatterns = [
    path('comments/', views.comments),
    path('comments/<int:year>/', views.year_comments),
    path('comments/<int:year>/<int:month>/', views.month_comments),
]
```

Consider a request to **/comments/2023/01/**. What would happen?

It would match the third entry in the list.
Django would call the function views month_comments(request, year=2005, month=3).

```
from django.http import HttpResponse


def comments(request):
    return HttpResponse("Hello World!")
```

UNIVERSITY OF SUSSEX

Path converters

- *str*: it matches any non-empty string, excluding the path separator, '/'.
- *int*: it matches zero or any positive integer.
- *slug*: it matches any slug string consisting of ASCII letters or numbers, plus the hyphen and underscore characters.
  - For example: playing-with-django-for-the-1st-time.
- *uuid*: it matches a formatted UUID.
  - For example, 075194d3-6885-417e-a8a8-6c931e272f00. Returns a UUID instance.
- *path*: it matches any non-empty string, including the path separator, '/'.
  - Enables matching a complete URL path rather than a segment as with *str*.

UNIVERSITY OF SUSSEX

The `path()` function

- It returns an element for inclusion in urlpatterns

- `path(route, view, kwargs=None, name=None)`
  - The *route* argument should be a string
  - The *view* argument is a view function
  - The *kwargs* argument allows you to pass additional arguments to the view function or method.
  - The *name* argument is related to naming URL patterns, which is useful to perform URL reversing.

```
from django.urls import path

urlpatterns = [
    path('index/', views.index, name='index-view'),
    path('shortbio/<username>/', views.shortbio, name='shortbio'),
]
```

UNIVERSITY OF SUSSEX

Regular expressions

- Can be used if the paths and converters syntax isn't enough to define the desired URL patterns
  - `re_path()` should be used instead of `path()`
- Python named regular expression groups syntax is `(?P<name>pattern)`,
  - `name` is the name of the group
  - `pattern` is some pattern to match.

```
from django.urls import path, re_path


urlpatterns = [
    path('comments/<int:year>/', views.year_comments),
    re_path(r'^comments/(?P<year>[0-9]{4})/$', views.year_comments),
]
```

The `include()` function

- It takes a full Python import path to another URLconf module that should be “included” in this place

- `include(`*`module, namespace=None`*`)`
  - *`module`* is the URLconf module (or module name)
  - *`namespace`* (`str`) is the instance namespace for the URL entries being included
- `include(`*`pattern_list`*`)`
  - `pattern_list` is an Iterable of `path()` and/or `re_path()` instances.

UNIVERSITY OF SUSSEX

The `include()` function

- Including other URLconfs

```
from django.urls import include, path

urlpatterns = [
    path('register/', include('registerapp.urls')),
    path('comment/', include('commentstoreapp.urls')),
]
```

- Whenever Django encounters `include()`
  - it chops off whatever part of the URL matched up to that point
  - sends the remaining string to the included URLconf for further processing.

UNIVERSITY OF SUSSEX

The `include()` function

- Include additional URL patterns by using a list of `path()` instances

```
from django.urls import include, path

from .registerapp import views as reg_views

account_patterns = [
    path('login/', reg_views.login),
    path('register/', reg_views.register),
]

urlpatterns = [
     path('comment/', include('commentstoreapp.urls')),
     path('account/', include(account_patterns)),
]
```

Processing a request

1. Django determines the root URLconf module to use.
2. It loads that Python module and looks for the variable *urlpatterns*.
3. It runs through each URL pattern, in order, and stops at the first one that matches the requested URL, matching against *path_info*.
4. Once one of the URL patterns matches, Django imports and calls the given view, which is a Python function. The view gets passed as argument, for example, an instance of HttpRequest.
5. If no URL pattern matches, or if an exception is raised during any point in this process, Django invokes an appropriate error-handling view.

Returning errors

- Django provides help for returning HTTP error codes.
- There are subclasses of HttpResponse for a number of common HTTP status codes

```
from django.http import HttpResponse, HttpResponseNotFound

def my_view(request):
    # ...
    if foo:
        return HttpResponseNotFound('<h1>Page not found</h1>')
    else:
        return HttpResponse('<h1>Page was found</h1>')
```

The developer is responsible for defining the HTML of the resulting error page

UNIVERSITY OF SUSSEX

The Http404 exception

- There's an easier way to handle 404 errors as they are very common.
- An HTML template named 404.html can be returned to show customized HTML whenever necessary

```
from django.http import Http404

def my_view(request):
    raise Http404("File does not exist")
```

Shortcuts

- The `render()` function combines a given *template* with a given context dictionary and returns an *HttpResponse* object with that rendered text.

- `render(request, template_name, context=None, content_type=None, status=None, using=None)`
  - *request*: the request object used to generate this response.
  - *template_name*: the full name of a template to use or sequence of template names.
  - *context*: a dictionary of values to add to the template context.
  - content_type: the MIME type to use for the resulting document.
  - *status*: the status code for the response.
  - *using*: the NAME of a template engine to use for loading the template

Shortcuts: render()

- The example below renders the template *webapps/index.html* with the MIME type *application/xhtml+xml*:

```
from django.shortcuts import render

def my_view(request):
    ...
    return render(request, 'webapps/index.html', {
        'web': 'apps',
    }, content_type='application/xhtml+xml')
```

UNIVERSITY OF SUSSEX

Shortcuts

- The `redirect()` function Returns an *HttpResponseRedirect* to the appropriate URL for the arguments passed.

- `redirect(to, *args, permanent=False, **kwargs)`
  - Can receive as arguments
    - a model,
    - a view name, and
    - an absolute or relative URL
  - Temporary redirect is set by default

UNIVERSITY OF SUSSEX

Shortcuts: `redirect()`

- The `redirect()` function can be used in a number of ways

```python
from django.shortcuts import redirect

def my_view(request):
    ...
    return redirect('the-name-of-a-view', foo='bar')
```

by passing the name of a view

by passing a hardcoded URL to redirect to

```python
from django.shortcuts import redirect

def my_view(request):
    ...
    return redirect('/some/url/')
```

```python
from django.shortcuts import redirect

def my_view(request):
    ...
    return redirect('https://example.com/')
```

This also works with full URLs

Class-based views

- Enables responding to different HTTP request methods with different class instance methods
  - Differently from conditionally branching code inside a single view function (i.e., function-based views)
- In a class-based view, the code to handle HTTP GET in the simple view example would look like this:

```
from django.http import HttpResponse
from django.views import Views


class HelloView(View):
    def get(self, request):
        html = "<html><body>Hello World!</body></html>"
        return HttpResponse(html)
```

Class-based views

- The URL resolver expects to send the request and associated arguments to a callable function, not a class
- The `as_view()` class method returns a function that can be called upon a request arrival for a URL that matches the associated pattern

```
from django.urls import path
from django.urls import HelloView


urlpatterns = [
    path('hello/', HelloView.as_view()),
]
```

UNIVERSITY OF SUSSEX

View decorators

- Django provides several decorators that can be applied to views to support various HTTP features
- The decorators in `django.views.decorators.http` can be used to restrict access to views based on the request method
  - These decorators will return a django.http.HttpResponseNotAllowed if the conditions are not met
- Allowed HTTP methods
  - `require_http_methods(request_method_list)`
    - `Decorator to require that a view only accepts particular request methods`

UNIVERSITY OF SUSSEX

View decorators

- Usage of `require_http_methods`

```python
from django.views.decorators.http import require_http_methods

@require_http_methods(["GET", "POST"])
def my_view(request):
    # I can assume now that only GET or POST requests make it this far
    # ...
    pass
```

request methods should be in uppercase

- Allowed HTTP methods
  - `require_GET()`
    - Decorator to require that a view only accepts the GET method

UNIVERSITY OF SUSSEX

View decorators

- Allowed HTTP methods
  - `require_POST()`
    - Decorator to require that a view only accepts the POST method
  - `require_safe()`
    - Decorator to require that a view only accepts the GET and HEAD methods.
    - These methods are commonly considered *safe* because they should not have the significance of taking an action other than retrieving the requested resource
- The following decorators in `django.views.decorators.http` can be used to control caching behavior on particular views
  - `condition(etag_func=None, last_modified_func=None)`
  - `etag(etag_func)`
  - `last_modified(last_modified_func)`

UNIVERSITY OF SUSSEX

Next Lecture ...

- ✓ Introduction
- ✓ HTTP, Caching, and CDNs
- ✓ Views
- ➢ **Templates**
- Forms
- Models
- Security
- Transactions
- Remote Procedure Call
- Web Services
- Time
- Elections and Group Communication
- Coordination and Agreement

UNIVERSITY OF SUSSEX