Intro to Django and MVC

An overview to help make learning Django easier.

Topics

1. Model-View-Controller (MVC) Design Pattern
   most web apps use this pattern.

2. How Django Processes a Web Request

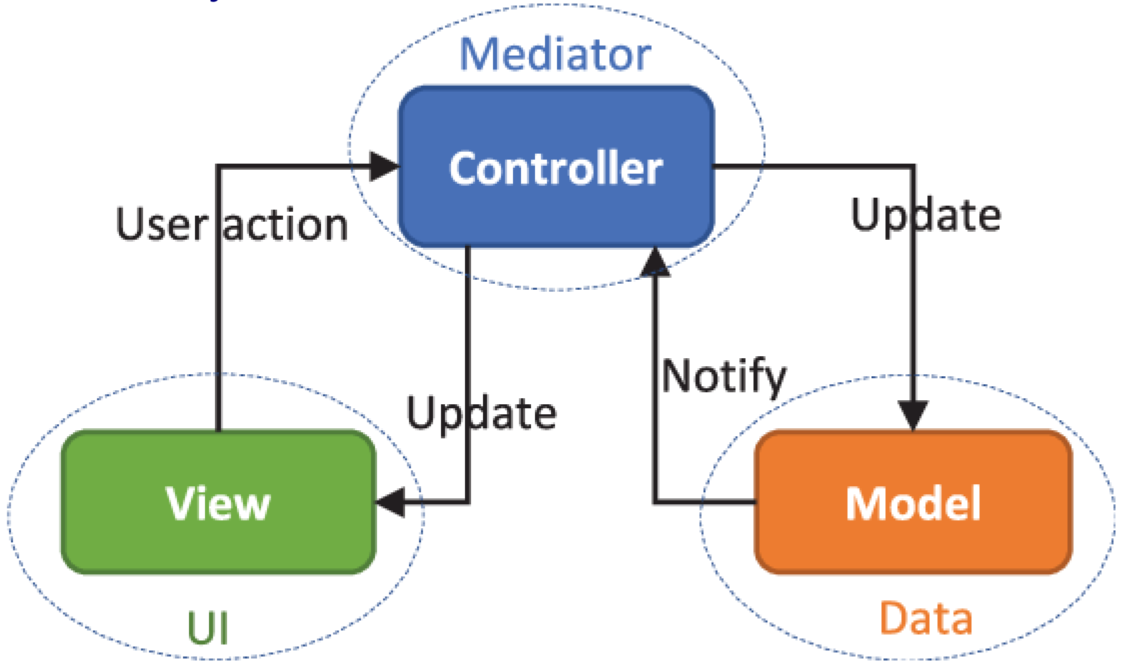
3. Structure of a Django Project
   ...and tip so your configuration directory always has the same name

Model-View-Controller Pattern

There are many ways to implement MVC, with different interactions between M-V-C. This is just one of them:

Handles requests or events from UI; converts them into commands for model or views.
May receive notifications from models.

Displays info to user and accepts interactions.

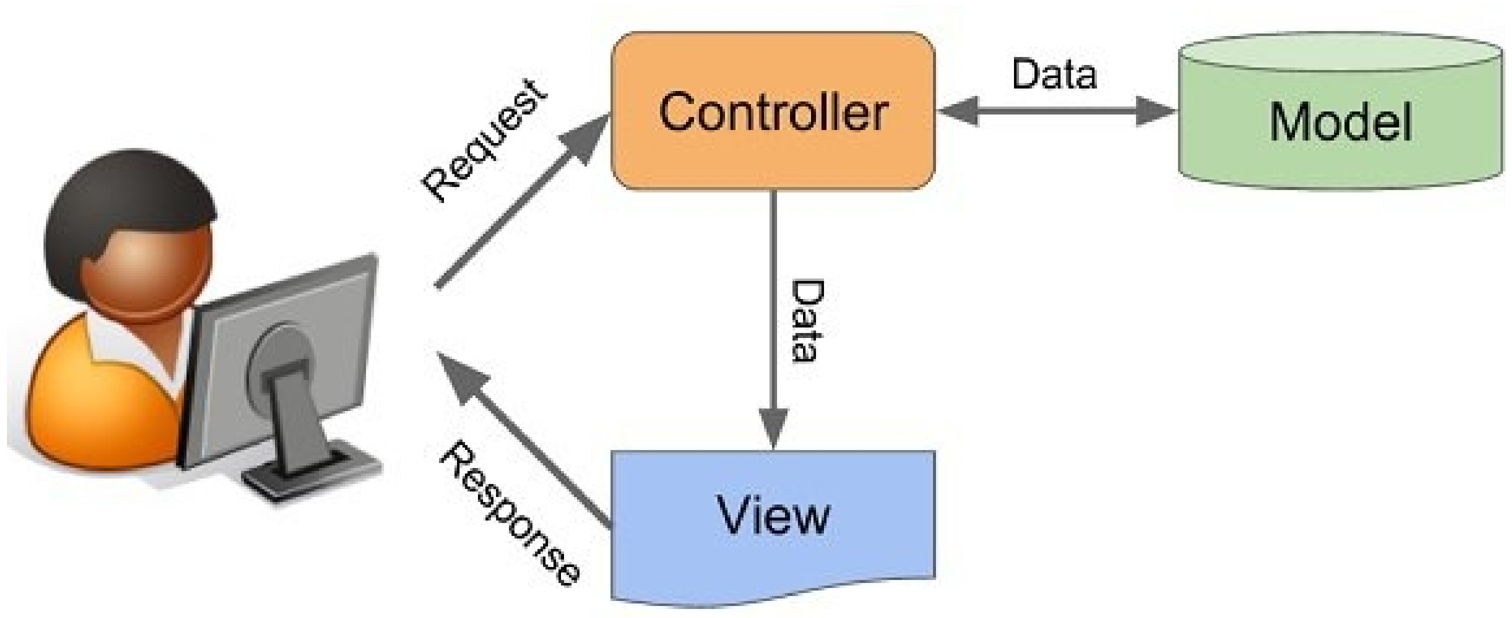


Application logic, data, and rules.

Model usually doesn't know about the UI (View).

Simple MVC for Web Apps

Shows flow of request-response; the label "Data" is misleading. Controller makes requests of Application Layer (including Model) to handle user requests.



Source: https://www.tech101.in/streamline-your-system-with-the-mvc-model/

MVC in Web Apps

Views - web pages or code that generates web pages. Views may be passive (HTML) or interact with user via code such as Javascript in web pages.

Controllers - code that receives user's request. Usually the first thing after the "router" (part of framework that assigns URLs to methods).

Model(s) - responsible for application data and logic. Often involves handling *persistent* data.

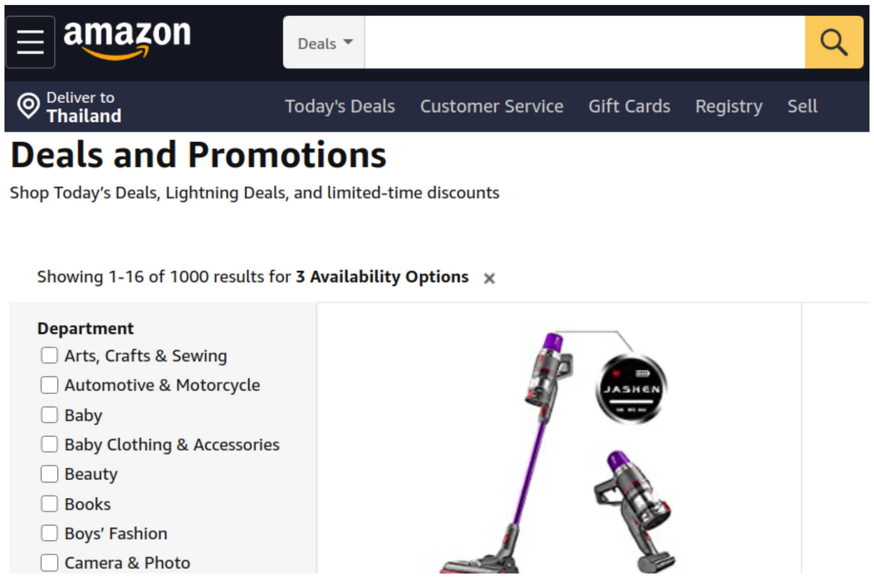
Why MVC?

A dynamic web page (like pages from amazon.com) are created from:

Layout - how the page should be structured

Data - source of info that goes in the page

Business Logic & Application Logic - how to handle user requests, managing flow of the application



Deals and Promotions

Shop Today's Deals, Lightning Deals, and limited-time discounts

Showing 1-16 of 1000 results for **3 Availability Options** ×

Department
- ☐ Arts, Crafts & Sewing
- ☐ Automotive & Motorcycle
- ☐ Baby
- ☐ Baby Clothing & Accessories
- ☐ Beauty
- ☐ Books
- ☐ Boys' Fashion
- ☐ Camera & Photo

Separation of Concerns

Separate these three concerns into different components.

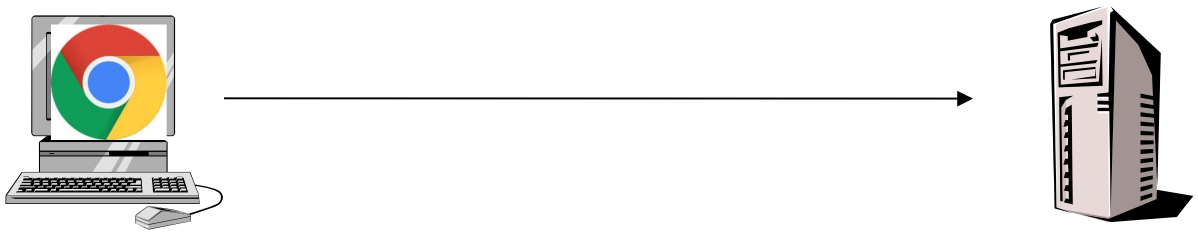
Controller - handles requests & performs application logic

Model - handles data and business logic

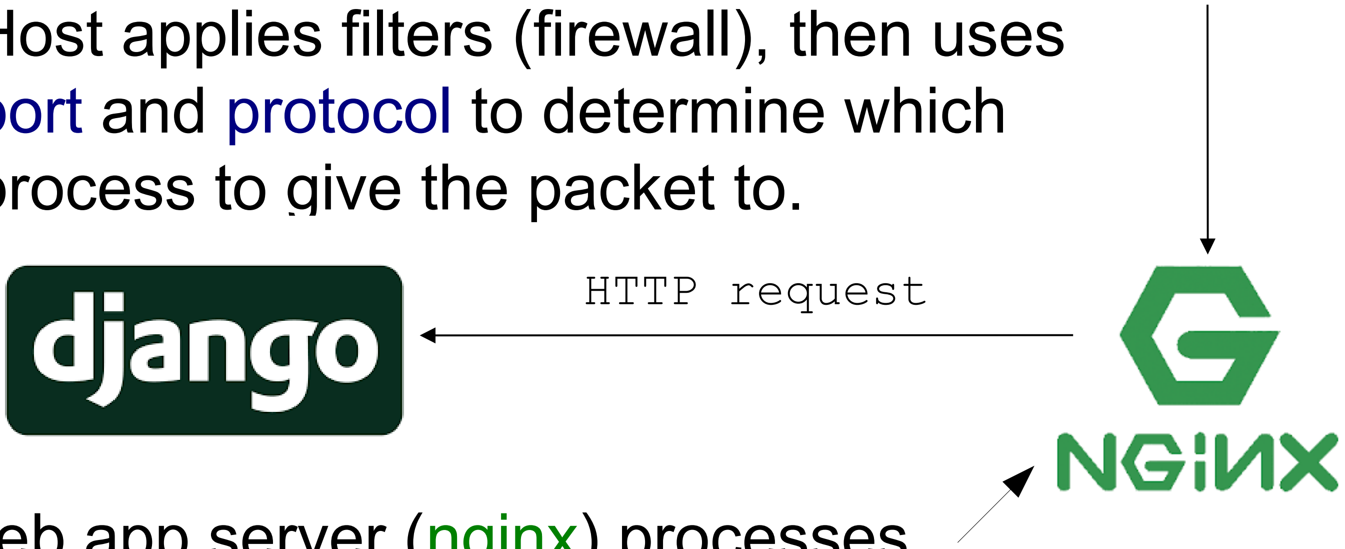
View - handles layout of pages

Handling a Web Request

1. TCP/IP packet is received by a server (host).



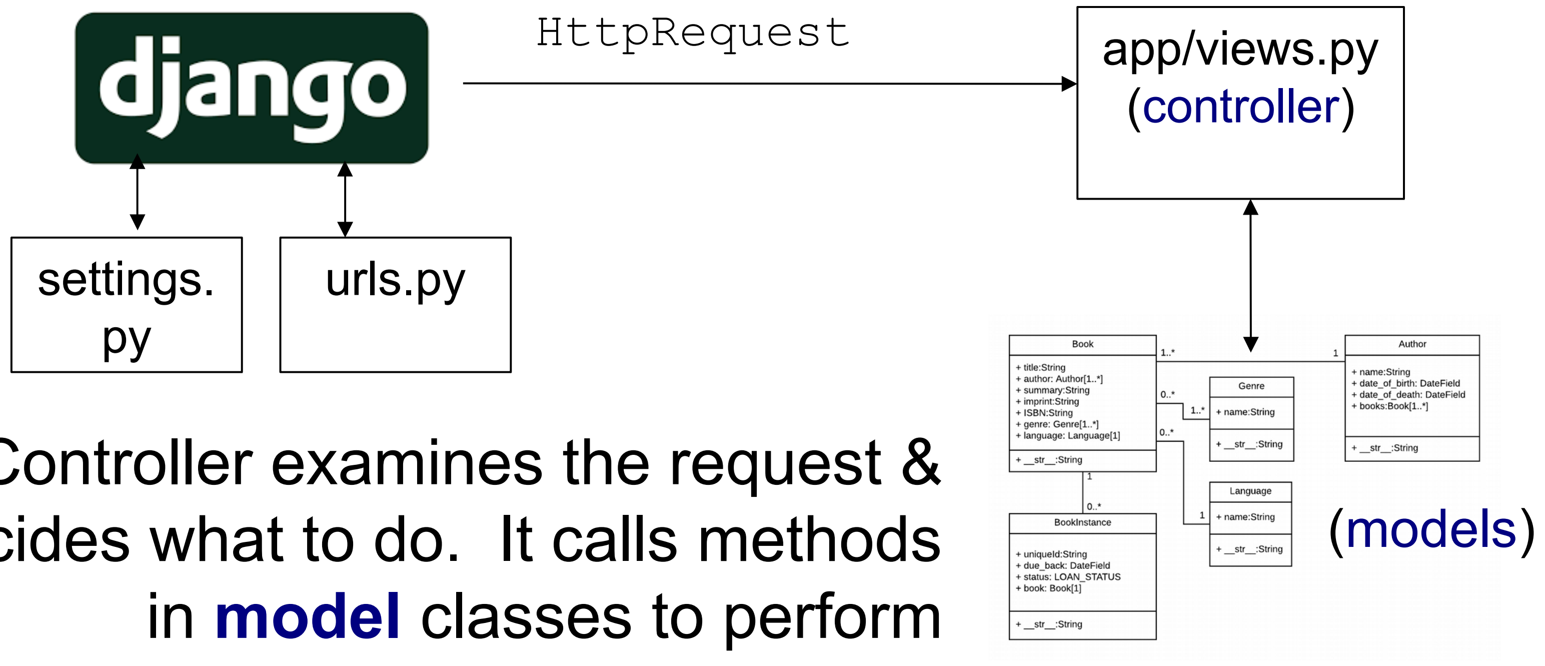
2. Host applies filters (firewall), then uses port and protocol to determine which process to give the packet to.



3. Web app server (nginx) processes HTTP request and gives it to the web application.

Web App Handles the Request

4. Django (framework code) parses the request to create an `HttpRequest` object.
5. It uses the app's *router* (Django "URLconf") to decide which controller code should *handle* the request.

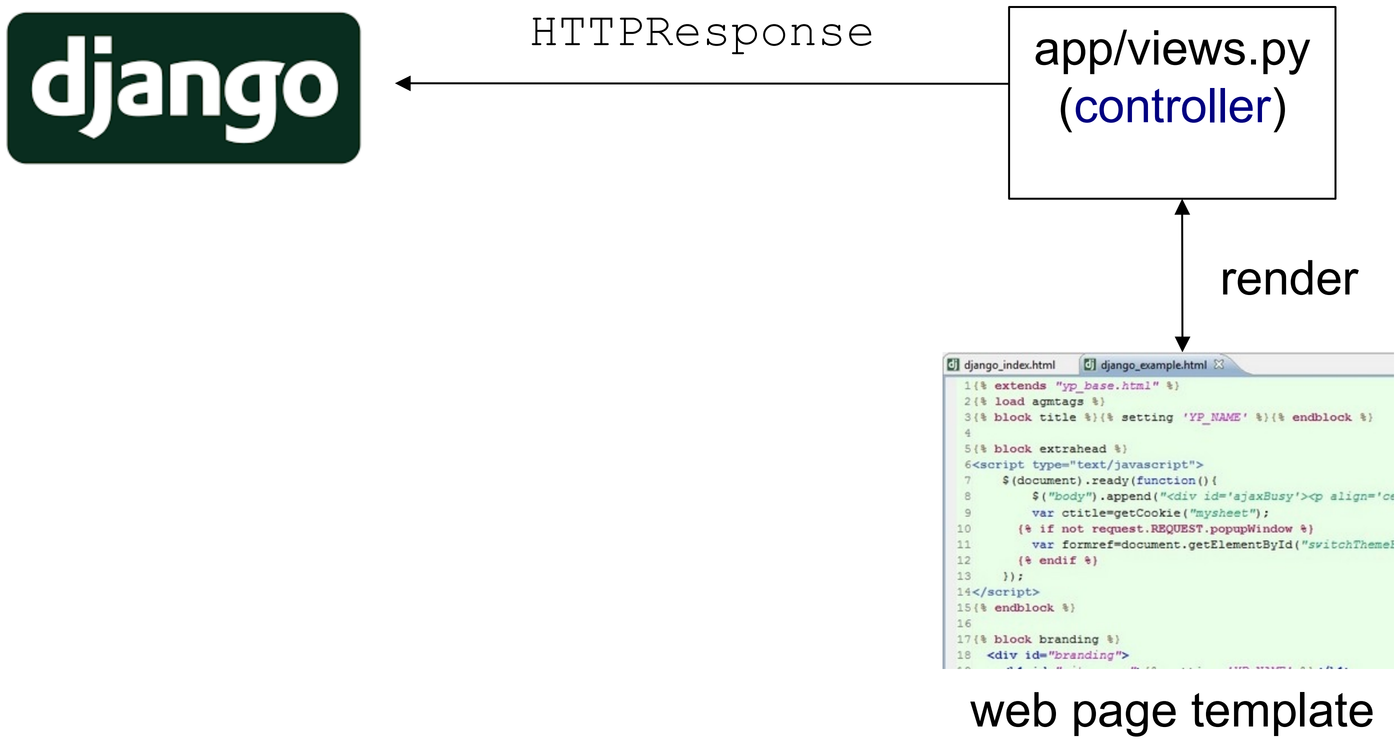


6. Controller examines the request & decides what to do. It calls methods in **model** classes to perform business logic & access app data.

The App Returns a Response

7. The controller creates and returns an `HttpResponse`.

- controller (Django "view") can create response itself, or
- render a template to create a web page

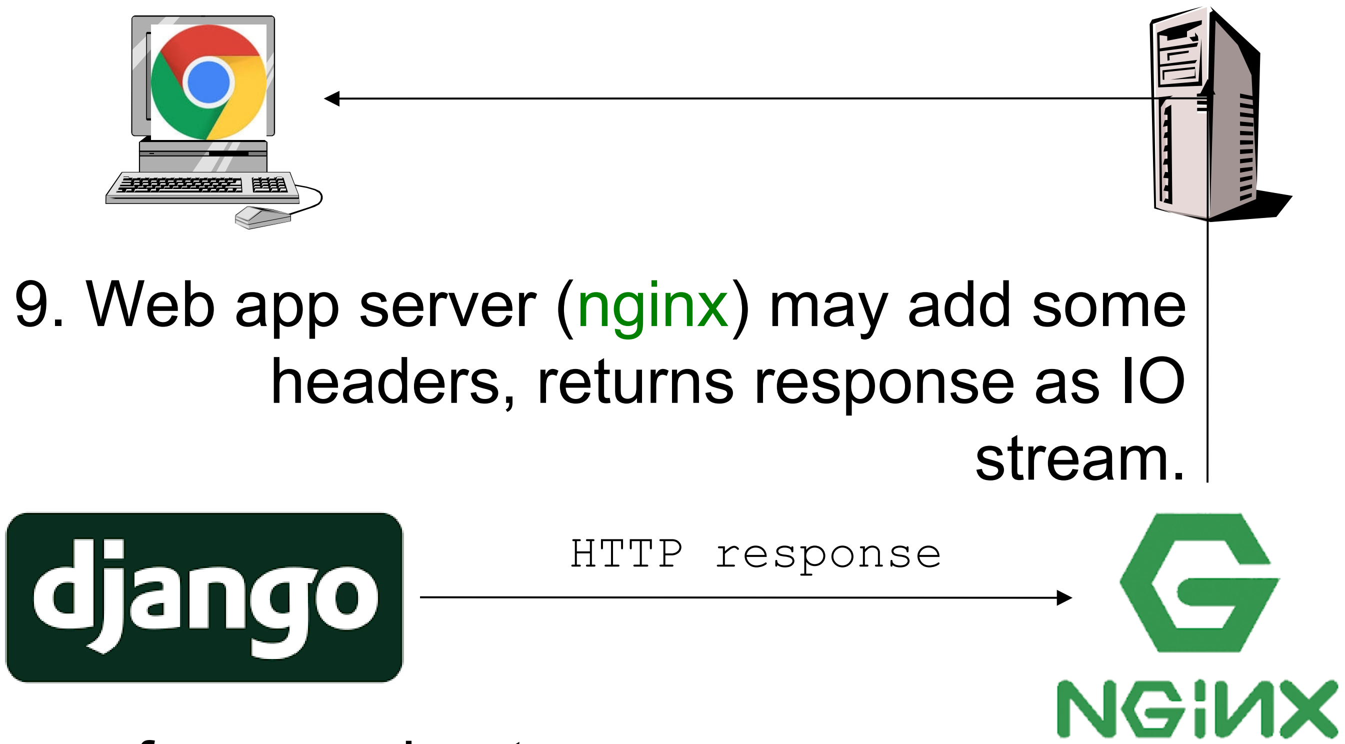


```
{% extends "yp_base.html" %}
{% load agmtags %}
{% block title %}{% setting 'YP_NAME' %}{% endblock %}

{% block extrahead %}
<script type="text/javascript">
    $(document).ready(function(){
        $("body").append("<div id='ajaxBusy'><p align='ce
        var ctitle=getCookie("mysheet");
      {% if not request.REQUEST.popupWindow %}
        var formref=document.getElementById("switchTheme
      {% endif %}
    });
</script>
{% endblock %}

{% block branding %}
  <div id="branding">
```

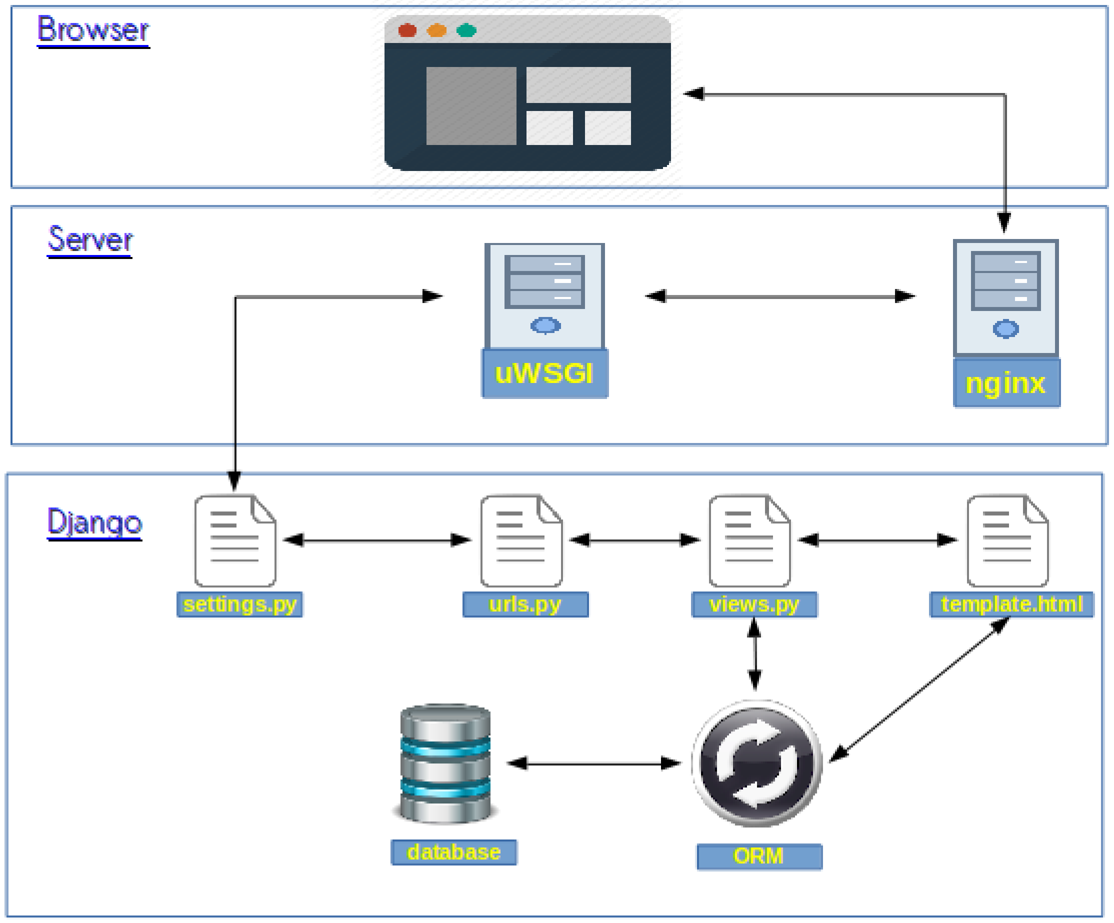
Send the Response!



9. Web app server (nginx) may add some headers, returns response as IO stream.

8. Django framework returns `HttpResponse` to web app server as a standard HTTP response.

Another Illustration of Request Lifecycle



From http://django-easy-tutorial.blogspot.com/2017/03/django-request-lifecycle.html

MVC or MVT?

Django says the framework uses "Model-View-Template" not MVC design pattern.

What Django calls "views" are similar to what other frameworks call "controllers".

Intro to Django MVT with explanation:

https://www.youtube.com/watch?v=GGkFg52Ot5o

(5 minutes)

Structure of a Django Project

Create a project named "mysite".

```
cmd> django-admin startproject mysite
```

Creates:

```
mysite/
    manage.py

    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

script to start, stop, test, or update the project

subdirectory for project settings and configuration

"`mysite`" configuration directory

Every Django project has a project configuration dir.

settings.py - names of apps and "middleware" you use.
- database location and credentials
- variables used by your apps and Django
- a project "secret" key

urls.py - defines which URLs should be sent to which methods.
Used to "route" requests to your code, e.g.
`GET /polls/1` -> polls.views.detail(1)

Demo: real settings.py and urls.py

View an actual `settings.py` and `urls.py` file.

Demo: start the built-in server

```
cmd> python manage.py runserver
```

You can view your Django app at http://localhost:8000

Django is a web application framework.

Its <u>not</u> a "web server", but includes a web server for development. Its not a production-level server.

Demo: add static content

While the development server is still running!

1. Edit `mysite/settings.py`. At the end of file add:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]
```

2. In the project base dir, create a subdir "`static/`". Then create `static/greeting.html`

3. You can view this file without restarting server!

```
http://localhost:8000/static/greeting.html
```

Create an "app" for your code

Inside your django project, create an "app" for actual code:

```
cmd> cd mysite
cmd> python manage.py startapp polls
```

Creates:

```
mysite/
    manage.py
    polls/
        admin.py
        apps.py
        migrations/
        models.py
        tests.py
        urls.py
        views.py
```

subdirectory for your application code.

`urls is optional`

admin.py

Used to "register" your models with Django middleware.

Can also be used to customize the "admin" panel for your app.

```
# admin.py
from django.contrib import admin
from .models import Question, Choice


# Register your models here.
admin.site.register(Question)
admin.site.register(Choice)
```

Our Model classes

apps.py

Define a Class for app configuration and a name for your app.   It inherits everything from AppConfig, so you don't need to write any code.

This is used in `settings.py` (in project config dir).

```
# apps.py
class Polls(AppConfig):
    name = 'polls'
```

models.py

Define Model classes containing data and application logic. Model objects are saved to a database.

This is one of the most important parts of your app!

```
# models.py
from django.db import models
class Question(models.Model):
    question_text = models.CharField(
                 'question', max_length=100)
    pub_date = models.DateTimeField(
                 'date published')

    def isPublished(self):
        return datetime.now() > self.pub_date
```

migrations/

A directory containing "SQL migrations".

When you change the structure of models, the structure of the database tables (*schema)* must be updated to match.

Django creates an "SQL migration" in this directory whenever you run:

```
python manage.py makemigrations
```

```
migrations/
    0001_initial.py
    0002_add_closing_date.py
```

tests.py

A file for unit tests of your app.

Putting all your tests in one file is not a good idea.

We will later replace this file with a **`tests/`** directory.

```
# tests.py
from django.test import TestCase

class QuestionModelTest(TestCase):
    def test_create_question(self):
        q = Question(question_text="...")
        q.save()
        self.assertTrue(q.id > 0)
        ...
```

Use django.test.TestCase

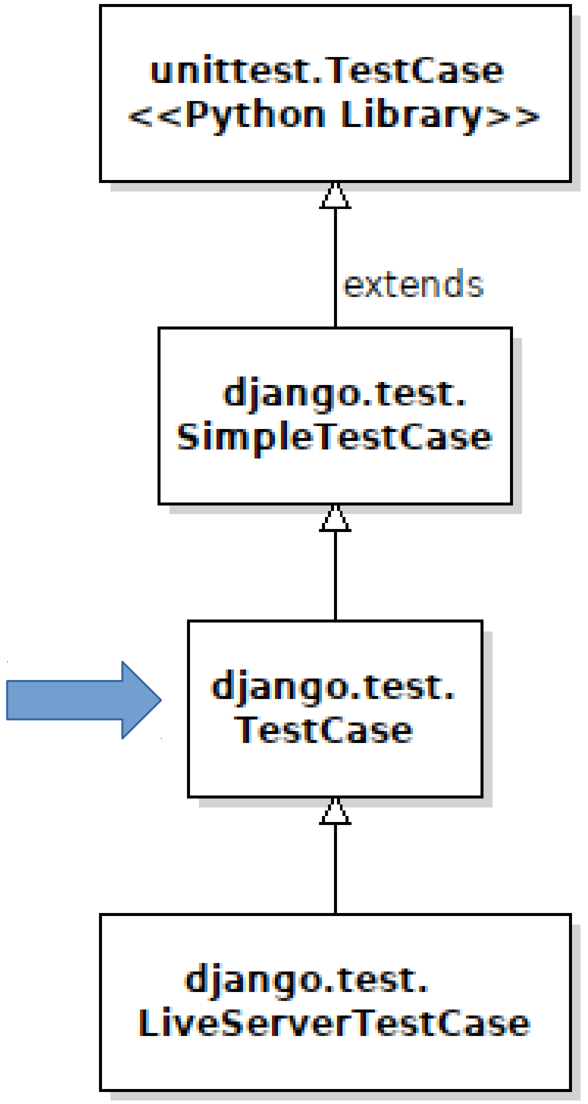
Use django.test.TestCase instead of Python's TestCase.

Django's TestCase adds important features.

- creates a "test" database in-memory before each test.

- extra assert methods, like assertInHTML, assertRedirects

- provides a Client class for testing views.



views.py

A file for your "view" methods that handle requests from the user. views may also be *classes*.

Views often provide data for an HTML "template" and tell Django to **render** it, as in example below.

```
# views.py

def index(request):
    """show index of recent poll questions"""
    questions = Question.objects.order_by('id')[:10]
    return render(request, 'polls/index.html',
           {'question_list':questions})

...
```

views, requests, and responses

Django creates an HttpRequest object from the data in the HTTP request received from the web.

It gives this request object to the view.

A view returns an HttpResponse that Django returns.

HttpRequest object

```
# views.py

def index(request):
    """show index of recent poll questions"""
    questions = Question.objects.order_by('id')[:10]
    return render(request, 'polls/index.html',
                  {'question_list':questions})

...
```

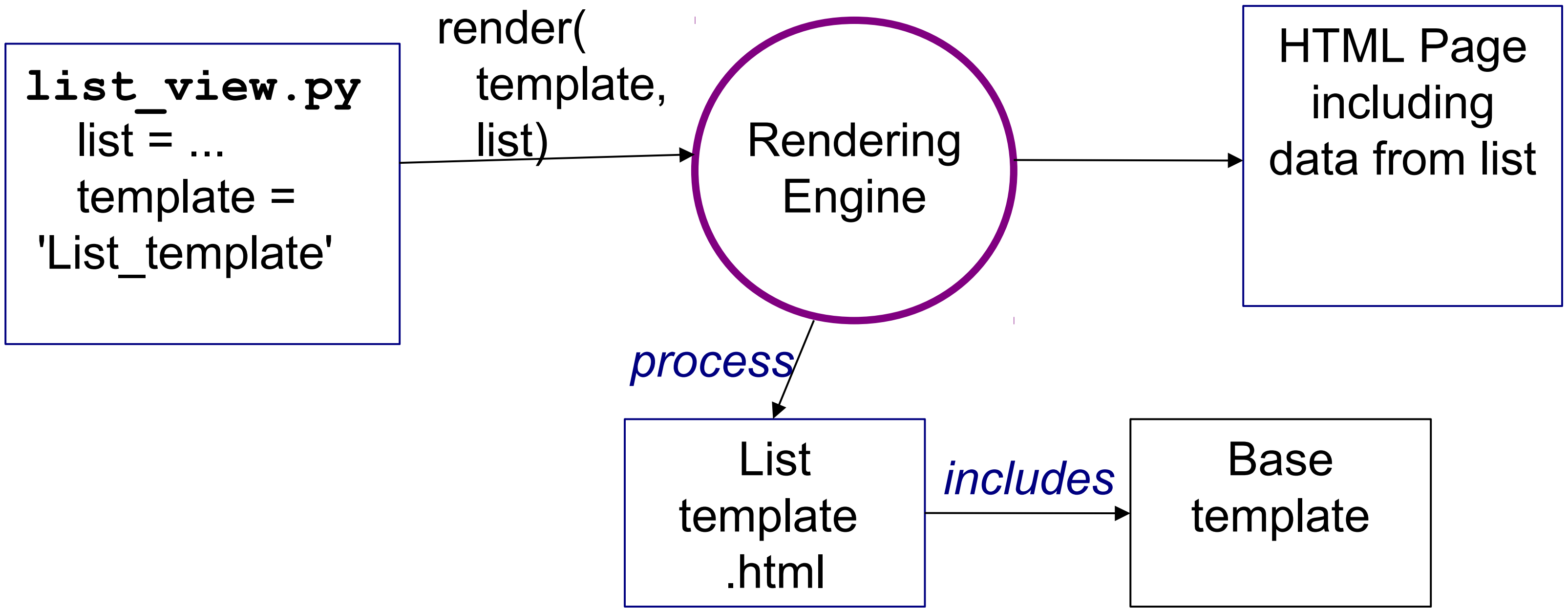
render() creates **HttpResponse** object

Templates

Web apps return customized HTML pages.

Apps inject data values into a "template" for an HTML page. A "rendering engine" processes the template.

Templates may include other templates.



`templates/`

You create this directory for your HTML templates.

Django recommends an extra subdirectory so that references to files are *unambiguous*.

```
mysite/
    manage.py
    polls/
        admin.py
        apps.py
        ...
        templates/polls/
                    index.html
                    poll_detail.html
                    poll_results.html
        views.py
```

Template to show a List of Questions

{%...%} are commands, {{ name }} are for data values.

```
{% extends 'base.html' %}
{% block content %}
<h1>List of Polls</h1>
<table>
    {% for question in question_list %}
    <tr>
      <td><a href="{% url 'polls:detail'
                       question.id %}">
                 {{question.question_text}}
          </a>
      </td>
    </tr>
    {% endfor %}
</table>
```

template can access attributes of an object

Django Project Creation

By default, the project config directory has the same name as the project main directory.

```
cmd> django-admin startproject amazon
```

Creates:

```
amazon/
    manage.py
    amazon/    ←    This is CONFUSING
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

I want "`mysite`" !!

I want the project config directory to **always** be named "`mysite`" ... or "`config`" or (whatever you prefer).

We should have a standard name for the config dir for **all** our projects!

```
amazon/
    manage.py
    mysite/          ← I want my settings here!
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Method 1: Rename project

Always create a project with name "`mysite`",
then rename the top-level project directory.

```
cmd> django-admin startproject mysite
```

```
cmd> rename mysite amazon
```

```
amazon/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Method 2: Create project in "."

Create project directory yourself, "cd" to that directory, and <u>then</u> run "startproject" with an extra parameter:

```
cmd> mkdir amazon

cmd> chdir amazon

cmd> django-admin startproject mysite .
```

"." means *create the project in the <u>current</u> directory*.

Resources for MVC

Too many! Everyone has their own interpretation of the MVC Pattern. A useful place to start is:

Wikipedia page for "MVC Design Pattern"

JavaFX uses MVC. Views are templates written in FXML, or generated by the controller.