

Introduction

Installation

Settings Module

Requests and Response

Running development server

Django admin site introduction

Installation

- ▶ Python Installation
- ▶ Django Installation

```
pip install django
```

Settings Module

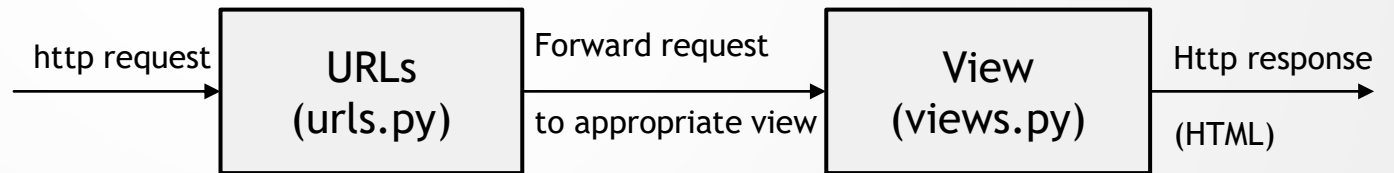
- ▶ settings.py - why these variables and values are used

Separating dev, prod, test environments

- ▶ Create a folder called settings or config
- ▶ Move the settings.py to that dir
- ▶ Create a __init__.py
- ▶ Create a dev.py, import settings to it, overwrite the values in dev.py
- ▶ Repeat the same for prod.py
- ▶ In settings.py, point the base dir one step up
- ▶ In wsgi.py and manage.py, point the settings to dev
- ▶ Run the server

Requests & Response

- ▶ When a url is striked in the web browser, *urls.py* is involved to map the path to a view (views.py)
- ▶ *views.py* is responsible to send a response HTML



Requests & Response

urls.py

```
from .views import index

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', index),
]
```

views.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse('<b>Hello world<b>')
```

Running development server

- ▶ Make a new project directory
- ▶ Start the project

```
django-admin startproject mysite
```

- ▶ Run the server

```
python manage.py runserver
```

Django admin site introduction

- ▶ Apply / enable the default app(s) by migrate

```
python manage.py migrate
```

- ▶ Create a super user

```
python manage.py createsuperuser
```

- ▶ Run the server and play around admin site

```
python manage.py runserver
```

Template Layer

Overview of template language

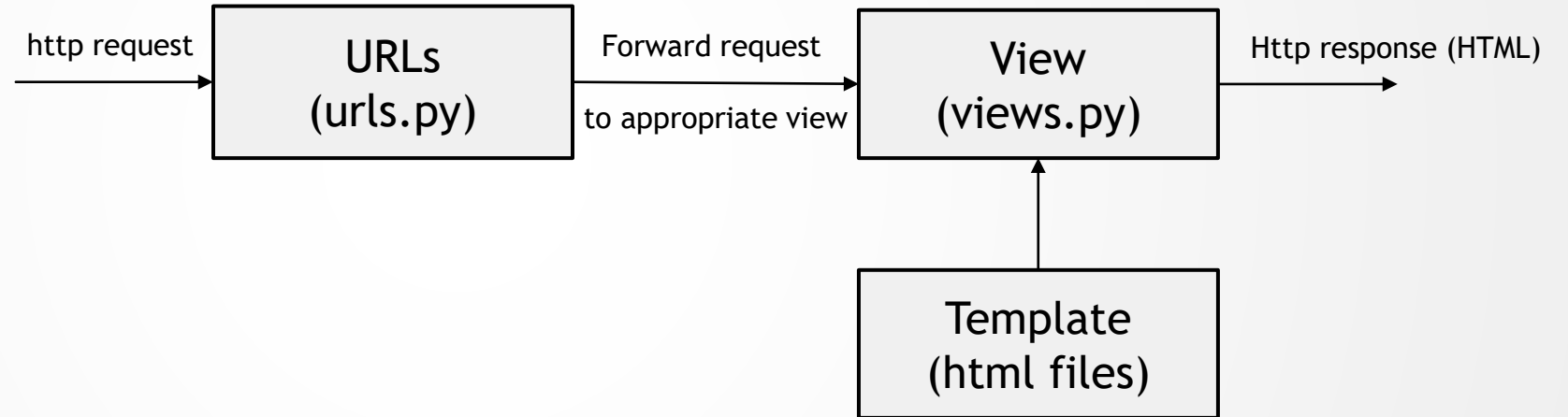
Built-in tags and filters

Humanization

custom tags and filters

csrf token

Overview of template language



Overview of template language

- ▶ Start an app

```
>python manage.py startapp members
```

- ▶ Add app to Installed Apps
- ▶ In project directory, in settings.py, add the newly created members

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'members'  
]
```

Overview of template language

- ▶ In app directory, create a folder called **templates** → backend → index.html
- ▶ Edit index.html

index.html

```
<html >

<style>
body {
    background-color: lightblue;
}
</style>
<body>
    hey
</body>
```

Overview of template language

- ▶ Edit members/views.py

views.py

```
def url1(request):  
    return render(request,  
        'backends/blog.html')
```

- ▶ Add path to to project's urls.py

urls.py

```
from members.views import page  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('members/', page)  
]
```

Overview of template language

▶ Makemigrations

```
>python manage.py makemigrations
```

▶ Migrate

```
>python manage.py migrate
```

▶ Run server

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

Built-in tags and filters

- ▶ For

- ▶ If

- ▶ Block

```
{% block content %}
```

```
{% endblock %}
```

- ▶ Extends

```
{% extends 'backend/base.html' %}
```

Built-in tags and filters

► Passing context from views.py

views.py

```
from django.shortcuts import render

# Create your views here.

context = {'data': [

    {'name': 'jessie', 'department': 'IT', 'count': 1},
    {'name': 'malini', 'department': 'Chem'}

]}

def page(request):
    return render(request, 'backend/index.html',
context)
```

Built-in tags and filters

► Using for and if

index.html

```
<h1> my app </h1>
<b> data is </b>

{{ data }}

<b> elements in data </b>

{% for student in data %}
{% if student.name == 'jessie' %}
{{ student.count }}
{% endif %}
{% endfor %}
```


Templates

Built-in tags and filters

- ▶ Add
- ▶ Capfirst

Additional Reference - [official doc](#)

Templates Built-in tags and filters

► Using add

index.html

```
<b> elements in data </b>
```

```
{% for student in data %}
```

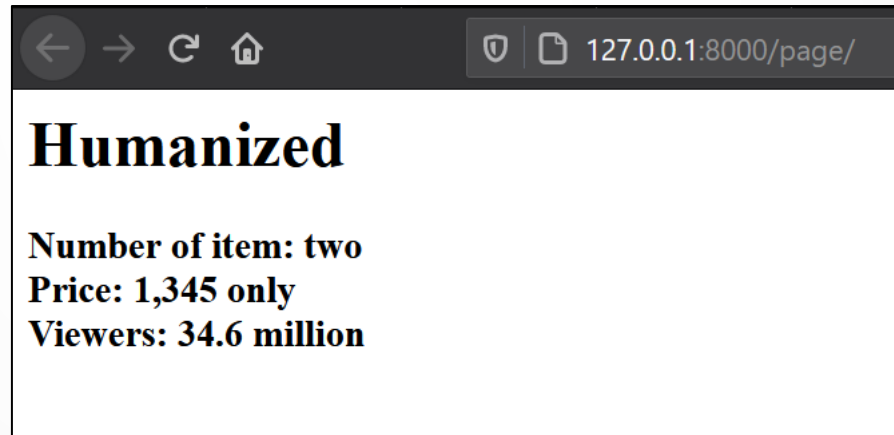
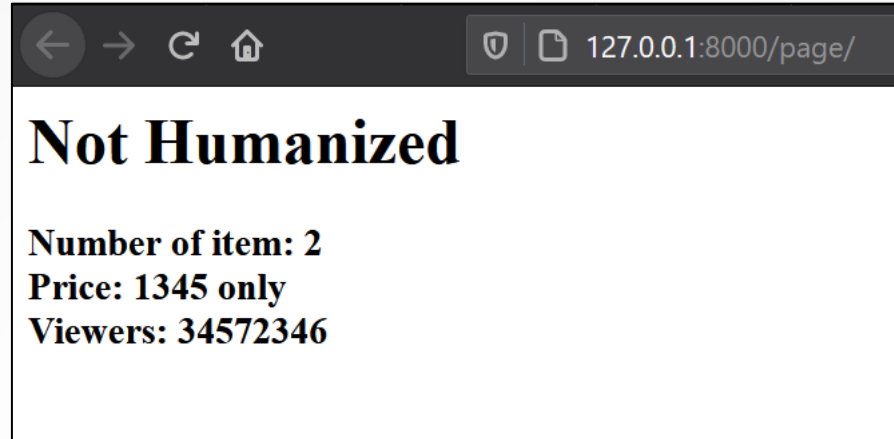
```
{% if student.name == 'jessie' %}
```

```
{{ student.count|add:1 }}
```

```
{% endif %}
```

```
{% endfor %}
```

Humanization



Ref:

<https://docs.djangoproject.com/en/3.1/ref/contrib/humanize/>

Humanization

- ▶ In Installed apps add → *django.contrib.humanize*
- ▶ Add context in *views.py*

views.py

```
from django.shortcuts import render

# Create your views here.
context = {
    'num_of_item': '2',
    'price': 1345,
    'viewers': 34572346
}

def page(request):
    return render(request,
        'backend/index.html', context)
```

Humanization

► Edit *index.html*

index.html

```
<h1> My Page </h1>

{% load humanize %}

<h3>
    Number of item:
    {{ num_of_item|apnumber }}
    <br>

    Price:
    {{ price|intcomma }} only
    <br>

    Viewers:
    {{ viewers|intword }}
    <br>

</h3>
```

custom tags and filters

- Tags

- ▶ Create a directory called *templatetags* in app directory
- ▶ Create an empty *__init__.py* in templatetags to treat the directory as a python package
- ▶ Create any py file in templatetags, in this slide we are taking it as *mycustomtags.py*

```
myproject
├── db.sqlite3
├── manage.py
├── myapp
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   ├── views.py
│   └── __init__.py
├── migrations
├── templatetags
│   ├── mycustomtags.py
│   └── __init__.py
```

custom tags and filters

-

Tags

- ▶ Edit mycustomtags.py

mycustomtags.py

```
from django import template

register = template.Library()

@register.simple_tag
def count_list(lst):
    return len(lst)
```

- ▶ Add to `INSTALLED_APPS` →
`'myapp.template_tags.mycustomtags'`

custom tags and filters

- Tags

► Context in *views.py*

views.py

```
context = {  
    'data' : [1,2,3,4]  
}
```

► In *index.html*

index.html

```
<h1> Custom Tags </h1>  
  
{% load mycustomtags %}  
  
{% count_list data %}
```


custom tags and filters

- filters

- ▶ Create a new python file for a custom filter - example *mycustomfilters.py*

mycustomfilters.py

```
from django import template

register = template.Library()

@register.filter
def cut(value, arg):
    return value.replace(arg, '')
```

- ▶ Add the custom filter to *INSTALLED_APPS* → *'myapp.template_tags.mycustomfilters'*

custom tags and filters

- filters

- ▶ Make sure you have a string value in *context* in *views.py*

views.py

```
context = {  
    'data' : [1,2,3,4],  
    'name' : 'malini'  
}
```

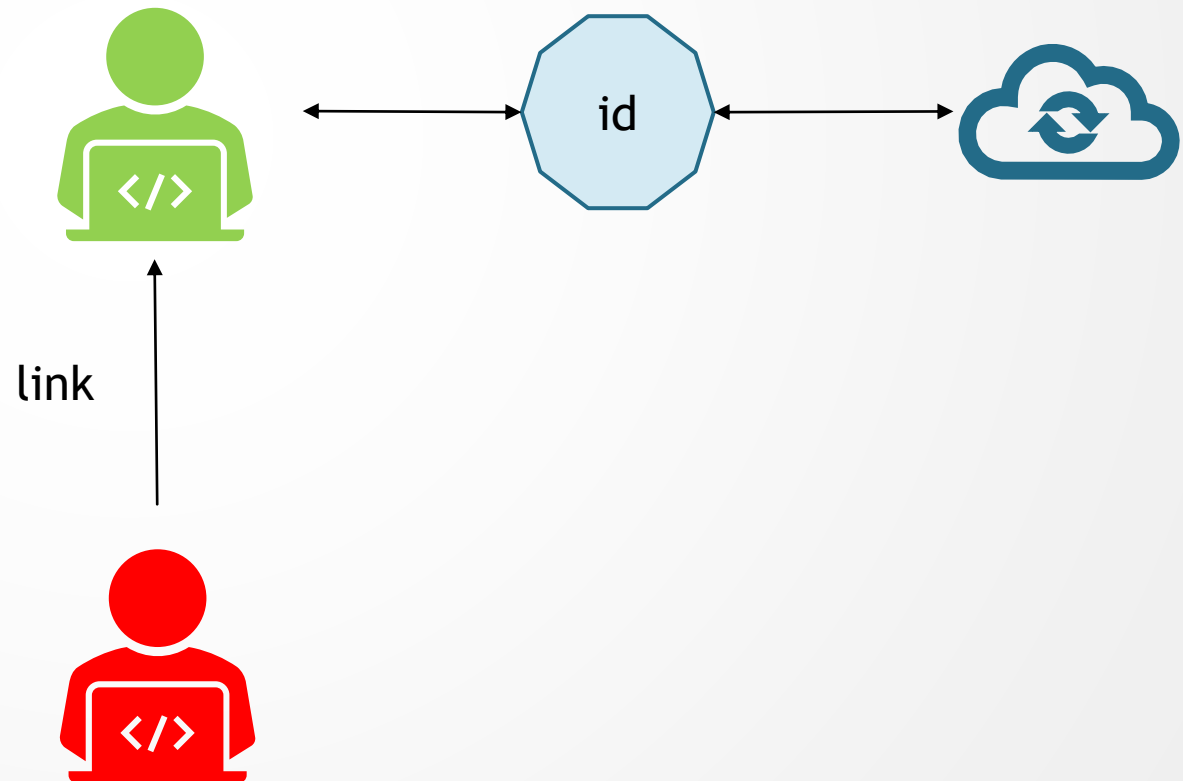
- ▶ In *index.html*

index.html

```
<h1> Custom Filters </h1>  
  
{% Load mycustomfilters %}  
  
{{ name|cut:'a' }}
```

Csrf token

- ▶ Csrp - Cross site request forgery
- ▶ To know what's csrf - <https://www.youtube.com/watch?v=hW2ONyxAySY>



View Layer

View functions

URL confs

Shortcuts and decorators

Request and response objects

File upload

Class based views

Mixins

Generating csv and pdf

View functions

- ▶ In project directory, in *views.py*

views.py

```
from django.http import HttpResponse

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

URL confs

- ▶ In project directory, in *settings.py*, ***ROOT_URLCONF*** where Django sees the urls patterns
- ▶ In app directory, create a new py file to store the app urls. Example: ***myappurls.py***

myappurls.py

```
from django.urls import path

from .views import page

urlpatterns = [
    path('page/', page),
]
```

URL confs

- ▶ In project directory, *urls.py*

urls.py

```
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.myappurls'))
]
```

- ▶ Try editing project dir settings.py urlconf
- ▶ Undo ! 😊

Shortcuts and decorators

Shortcuts

- ▶ `render()`
- ▶ `redirect()`
- ▶ `get_object_or_404()`
- ▶ `get_list_or_404()`

Official Doc Ref:

<https://docs.djangoproject.com/en/3.1/topics/http/shortcuts/>

Shortcuts and decorators

- ▶ Using redirect
- ▶ In app dir, *views.py*

views.py

```
from django.shortcuts import render, redirect

def page2(request):
    return redirect('../page/')
```

- ▶ In *urls.py*

urls.py

```
from .views import page, page2

urlpatterns = [
    path('page/', page),
    path('page2/', page2)
]
```

Shortcuts and decorators

Decorators

`django.views.decorators.http`

- ▶ `require_http_methods(request_method_list)`
- ▶ `require_GET()`
- ▶ `require_POST()`
- ▶ `require_safe()`

Official Doc Ref:

<https://docs.djangoproject.com/en/3.1/topics/http/decorators/>

Shortcuts and decorators

Decorators

- ▶ Normally a view function would use get method
- ▶ Try to restrict it to only POST method using *require_POST()*
- ▶ You should get *Method Not Allowed* error

views.py

```
from django.views.decorators.http import  
require_POST
```

```
@require_POST  
def page(request):  
    return HttpResponse('<h1> Hello </h1>')
```

Request and response objects

Request Objects

- ▶ `HttpRequest.scheme`
- ▶ `HttpRequest.path`
- ▶ `HttpRequest.method`
- ▶ `HttpRequest.is_secure()`
- ▶ `HttpRequest.user` >> *`request.user.is_authenticated`*

More:

<https://docs.djangoproject.com/en/3.1/ref/request-response/>

Request and response objects

► Few request and response objects

views.py

```
def page(request):  
  
    userauth = request.user.is_authenticated  
    secure_flag = request.is_secure()  
    curr_path = request.path  
    method_used = request.method  
  
    html_content = '''  
    <html>  
    <body>  
    user authenticated: {} </br>  
    connection uses https: {} </br>  
    current url path: {} </br>  
    method used: {} </br>  
    </body>  
    </html>  
    '''.format(userauth, secure_flag, curr_path,  
method_used)  
  
    return HttpResponseRedirect(html_content)
```

Class based views

- ▶ Without doing regular `request.method=='POST'`

views.py

```
from django.shortcuts import render, redirect

from django.contrib.auth import authenticate

from django.views import View
# Create your views here.

class Login(View):
    context = {'error_msg': ''}
    context = {'username': ''}

    def post(self, request):
        username = request.POST.get('username')
        password = request.POST.get('passcode')

        user = authenticate(request,
username=username, password=password)
```

Class based views

► Contd...

views.py

```
        if user is not None:
            self.context['username'] =
username
            return render(request,
'home.html', self.context)
        else:
            self.context['error_msg'] = '''
Error, Try again.
username and password mismatch
or
user doesnot exists.
'''
            return render(request,
'login.html', self.context)

    def get(self, request):
        return render(request, 'login.html',
self.context)
```

Class based views

- ▶ In urls.py

urls.py

```
from django.urls import path, include
from .views import login

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

- ▶ Makemigrations
- ▶ Migrate
- ▶ runserver

Class based views

List View

- ▶ Let's look at how not to use `render()` yet use template html files using the simple *listView*

views.py

```
from .models import member

from django.views.generic import ListView

class myclass(ListView):
    template_name='home.html'
    queryset=member.objects.filter(first_name__exact='hephzi')
```

Class based views

List View

- ▶ In urls.py and template and html file

urls.py

```
from django.urls import path, include
from .views import myclass

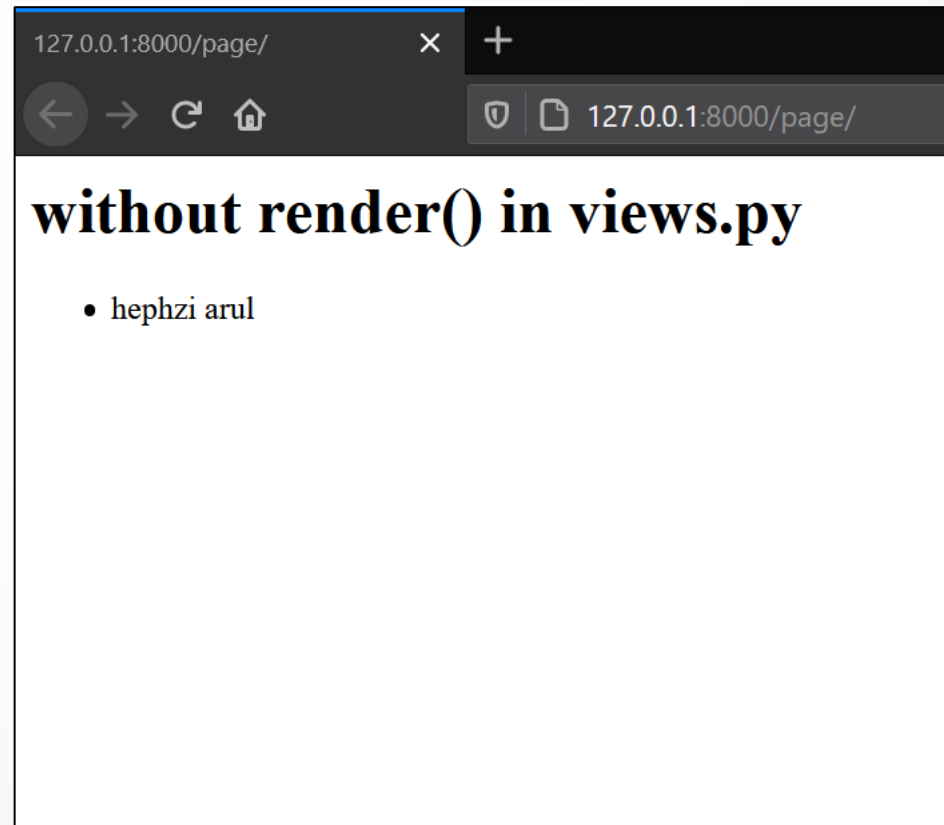
urlpatterns = [
    path('page/', myclass.as_view()),
]
```

home.html

```
<h1>without render() in views.py</h1>
<ul>
{% for i in object_list %}
    <li>{{ i.first_name }} {{ i.last_name }} </li>
{% endfor %}
</ul>
```

- ▶ Make migrations
- ▶ Migrate
- ▶ Runserver

▶ Output:



Class based views

List View

Generating csv and pdf - CSV

- ▶ Csv - Comma Separated Values
- ▶ Example:

```
hephzi, arul  
malini, surya  
jessy, karthik
```

- ▶ In templates dir,

my_template_name.txt

```
{% for row in data %}"{{ row.0|addslashes }}", "{{  
row.1|addslashes }}", "{{ row.2|addslashes }}", "{{  
row.3|addslashes }}", "{{ row.4|addslashes }}"  
{% endfor %}
```

Generating csv and pdf

- CSV

► In app dir, views.py

views.py

```
from django.http import HttpResponse
from django.template import loader

def page(request):
    response =
    HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment;
    filename="somefilename.csv"'
    csv_data = (
        ('First row', 'Foo', 'Bar', 'Baz'),
        ('Second row', 'A', 'B', 'C',
        '"Testing"', "Here's a quote"),
    )

    t = loader.get_template('my_template_name.txt')
    c = {'data': csv_data}
    response.write(t.render(c))
    return response
```

Generating csv and pdf - pdf

- ▶ Install reportlab for generating pdfs

```
>pip install reportlab
```

- ▶ In views.py

views.py

```
import io
from django.http import FileResponse
from reportlab.pdfgen import canvas

def page(request):
    buffer = io.BytesIO()

    p = canvas.Canvas(buffer)

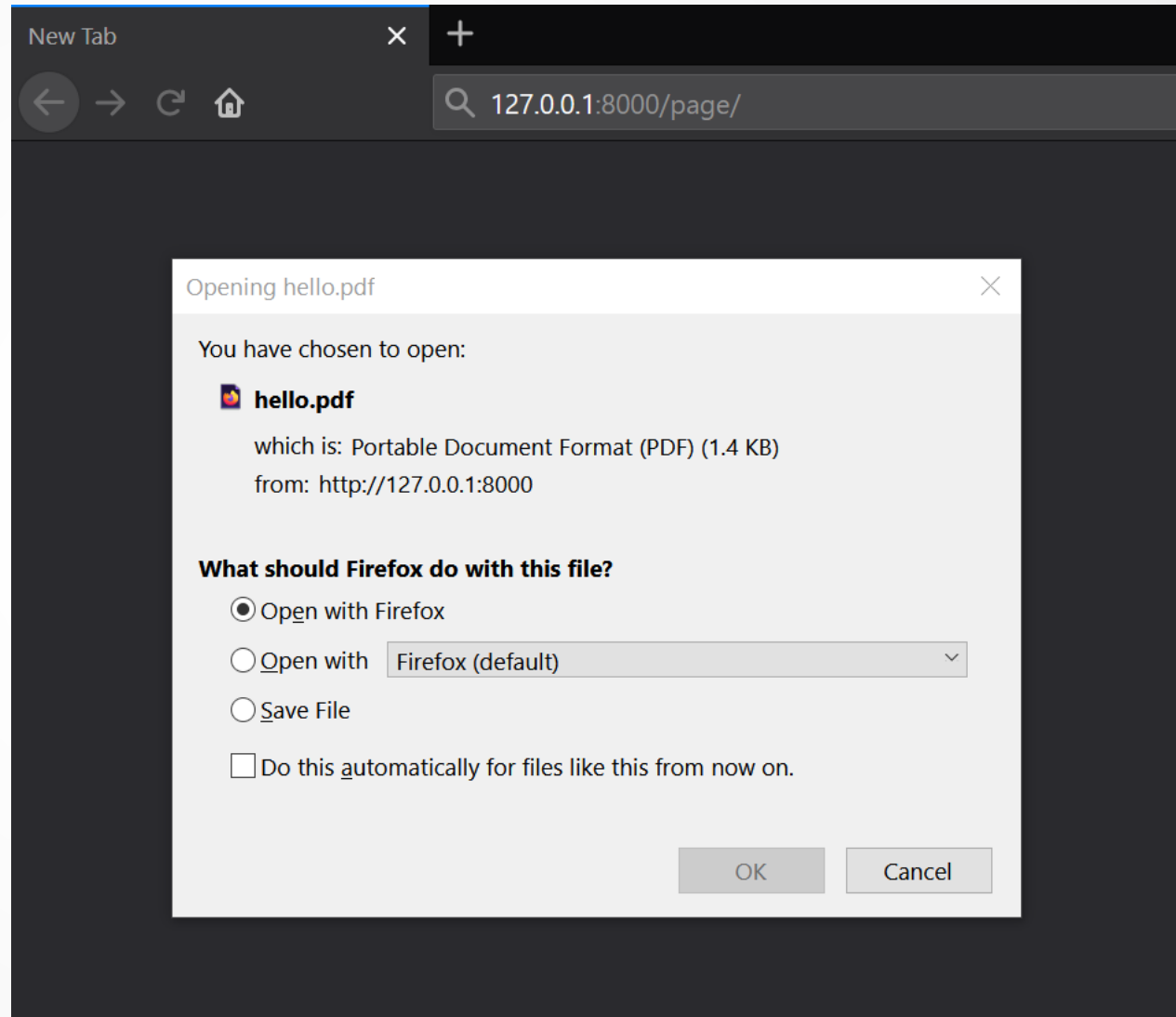
    p.drawString(300, 300, "Hello world.")

    p.showPage()
    p.save()

    buffer.seek(0)
    return FileResponse(buffer, as_attachment=True,
filename='hello.pdf')
```

Generating csv and pdf - pdf

► Runserver



Models Layer

Model Introduction

Field types and customization

Making queries

Accessing related objects

Django migrations

Raw sql, search

Model Introduction

- ▶ In app dir, in *models.py*

models.py

```
from django.db import models

# Create your models here.
class member(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Model Introduction

- ▶ Check the sqlite3 DB to see whether a new table with columns got created

```
myproj>C:\Users\Downloads\sqlite-tools-win32-x86-3340000\sqlite-  
tools-win32-x86-3340000\sqlite3.exe db.sqlite3
```

```
sqlite> .tables  
auth_group                                django_admin_log  
auth_group_permissions                   django_content_type  
auth_permission                          django_migrations  
auth_user                                django_session  
auth_user_groups                         myapp_member  
auth_user_user_permissions  
sqlite> pragma table info(myapp member);  
0|id|integer|1||1  
1|first_name|varchar(30)|1||0  
2|last_name|varchar(30)|1||0  
sqlite>
```

Field types

- In app dir, in *models.py*

models.py

```
from django.db import models

# Create your models here.
class member(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)

    packages = (
        ('a', 'annual'),
        ('h', 'half_yearly'),
        ('q', 'quarterly')
    )

    packages =
models.CharField(blank=True,max_length=1, choices=packages)
```

- ▶ Check the sqlite3 DB to see whether a new column got created

```
myproj>C:\Users\Downloads\sqlite-tools-win32-x86-3340000\sqlite-  
tools-win32-x86-3340000\sqlite3.exe db.sqlite3
```

Field types

```
sqlite> pragma table_info(myapp_member);  
0|id|integer|1||1  
1|first_name|varchar(30)|1||0  
2|last_name|varchar(30)|1||0  
3|packages|varchar(1)|1||0  
sqlite>
```

Field types

- In app dir, in *models.py*

models.py

```
from django.db import models

# Create your models here.
class member(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    updated_dt = models.DateTimeField(auto_now=True)

    packages = (
        ('a', 'annual'),
        ('h', 'half_yearly'),
        ('q', 'quarterly')
    )

    packages =
models.CharField(blank=True,max_length=1, choices=packages)
    def __str__(self):
        return self.first_name
```

Field types

- ▶ Check the sqlite3 DB to see whether a new column got created

```
myproj>C:\Users\Downloads\sqlite-tools-win32-x86-3340000\sqlite-  
tools-win32-x86-3340000\sqlite3.exe db.sqlite3
```

```
sqlite> pragma table_info(myapp_member);  
0|id|integer|1||1  
1|first_name|varchar(30)|1||0  
2|last_name|varchar(30)|1||0  
3|packages|varchar(1)|1||0  
4|updated dt|datetime|1||0
```

- ▶ More:
<https://docs.djangoproject.com/en/3.1/ref/models/fields/>

Making queries

► Open shell

```
myproj>python manage.py shell
```

```
(InteractiveConsole)
```

```
>>> from myapp.models import member
```

```
>>>
```

```
>>> b = member(first_name="jessy", last_name="tvm")
```

```
>>> b.save()
```

```
>>>
```

► Output

```
sqlite> select * from myapp_member;
```

```
1|hep |||2021-02-07 21:05:20.147456
```

```
4|hep |||2021-02-07 21:15:02.733587
```

```
5|jessy|tvm||2021-02-07 21:22:15.489916
```

Making queries

▶ Retrieving objects

```
>>> member.objects.all()[4].first_name  
'jessy'  
>>> member.objects.all()[3].first_name  
'hep '
```

▶ Filter objects

```
>>> member.objects.all().filter(first_name__exact = 'jessy')  
<QuerySet [<member: jessy>]>
```

▶ Try __gt, __lt, __contains, __startswith, __endswith

▶ More:

<https://docs.djangoproject.com/en/3.1/topics/db/queries/>

Making queries

- ▶ Create a new row in DB through *views.py*
- ▶ Retrieve a row from DB through *views.py*
- ▶ Retrieving human readable name of choices

```
def page(request):  
  
    obj = member.objects.all()[0]  
    b = b.packages  
  
    b = b.get_packages_display()  
  
    return HttpResponse(b)
```

Raw sql, search

- ▶ Raw sql, search - in Django shell

```
>>> from myapp.models import member
>>> for i in member.objects.raw('select * from myapp_member'):
...     print(i.first_name)
...
hephzi
```

- ▶ Though particular column is not selected in raw sql search, it does not throw error but load it *on-demand*

```
>>> for i in member.objects.raw('select id, first_name from
myapp_member'):
...     print(i.last_name)
...
arul
```

Forms Layer

Introduction

Forms API

Validating forms

Built-in fields, built-in widgets

Model form

Form sets

Form Introduction

► Creating a form using Django

views.py

```
from .forms import myForm

from django.http import HttpResponse
from django.views import View
from django.shortcuts import render

class page(View):
    def post(self, request):
        form = myForm(request.POST)

        if form.is_valid():
            return HttpResponse('thanks')

    def get(self, request):

        form = myForm()
        context = {'form': form}

        return render(request, 'home.html', context)
```

Form Introduction

- ▶ In app dir, create a file *forms.py*

forms.py

```
from django import forms

class myForm(forms.Form):
    full_name = forms.CharField(label='full name',
                                max_length=100)
    sender = forms.EmailField()
```

- ▶ In template html file

home.html

```
<form method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

Form Introduction

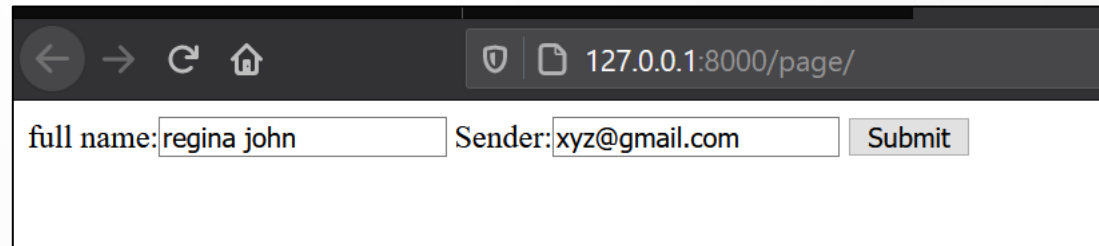
► In urls.py

urls.py

```
from django.urls import path, include
from .views import page

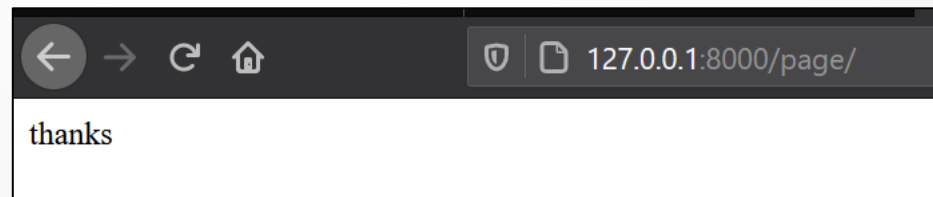
urlpatterns = [
    path('page/', page.as_view()),
]
```

► Runserver



127.0.0.1:8000/page/

full name: Sender:



127.0.0.1:8000/page/

thanks

To cover

- ▶ Inclusion tag - on completion of models
- ▶ Connect to another DB
- ▶ Model field customization
- ~~▶ Csrf - after forms or post get methods~~
- ~~▶ Views decorators - after get post methods~~