



PYTHON / DJANGO



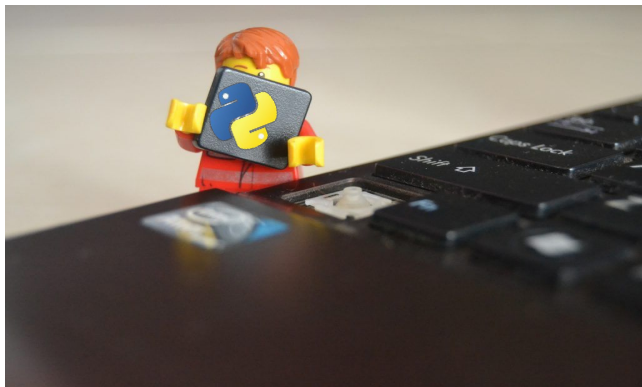
Que es PIP



PIP

Es una herramienta para administrar e instalar paquetes de python, esto permite que aislar las dependencias de nuestros proyectos.

Ambiente de Trabajo

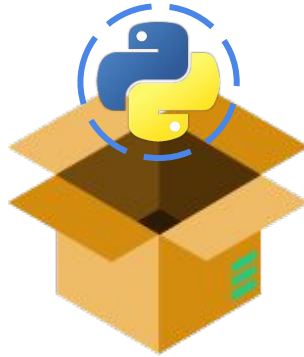


Instalar PIP en nuestra máquina

```
$ sudo apt-get install python-setuptools
```

```
$ sudo easy_install pip
```

¿Que es Virtualenv ?



El virtualenv es un programa o sistema que administra y controla múltiples instancias de entornos virtuales y que sirve para que nuestros proyectos sean independientes

Entornos de desarrollo con **virtualenv**

Instalar python 3 en Virtualenv

```
>CursoDjango$ virtualenv -p python3 env
```



Proyecto 1
python 2.7

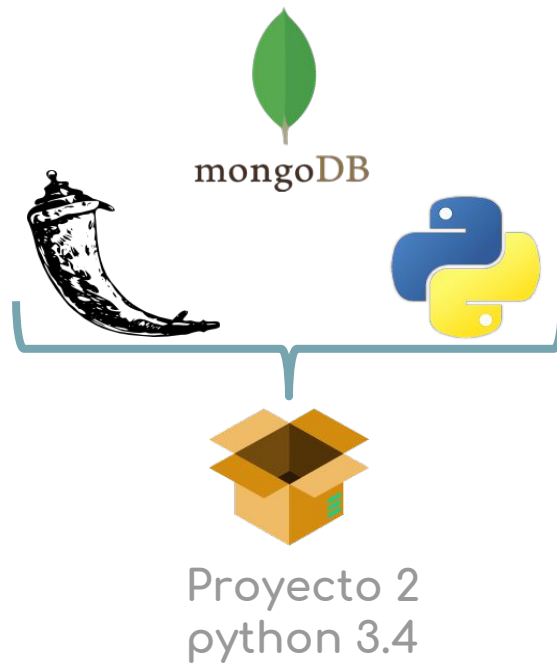


Proyecto 2
python 3.4



Proyecto 3
python 3.5

Virtualenv



Virtualenv

Crear

```
>$ virtualenv -p python3 env
```

Activar

```
>$ source env/bin/activate
```

Desactivar

```
>(env)$ deactivate
```

Instalar un paquete

```
>(env)$ pip install nombre_paquete
```

Ver paquetes instalados

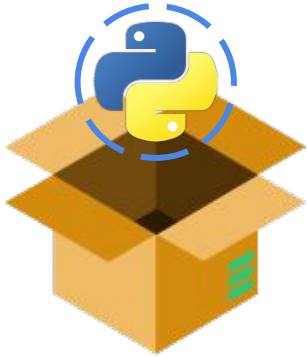
```
>(env)$ pip freeze --local
```

Ver información del paquete

```
>(env)$ pip show nombre_paquete
```



Instalación de paquetes de terceros



```
(env)CursoDjango$ pip install Pillow
```

```
(env)CursoDjango$ pip install django-geoposition
```

```
(env)CursoDjango$ pip install -U channels
```

```
(env)CursoDjango$ pip install djangorestframework
```




¿QUE ES DJANGO?



Django es un framework para aplicaciones web gratuito y open source, escrito en [Python](#).

¿ QUIENES LO USAN ?





Instalación

python 3.5

```
$ virtualenv -p python3 env  
(env) CursoDjango$ pip install django
```

INSTALACIÓN DE DEPENDENCIAS

Las dependencias se registran en el archivo
`requirements.txt`

> **(env)\$ pip install -r requirements.txt**



requirements.txt



```
# Bleeding edge Django
django==1.10.4

# Forms
django-braces==1.10.0
django-crispy-forms==1.6.1

# Images
Pillow==3.4.2
```



Linea de Comandos Django

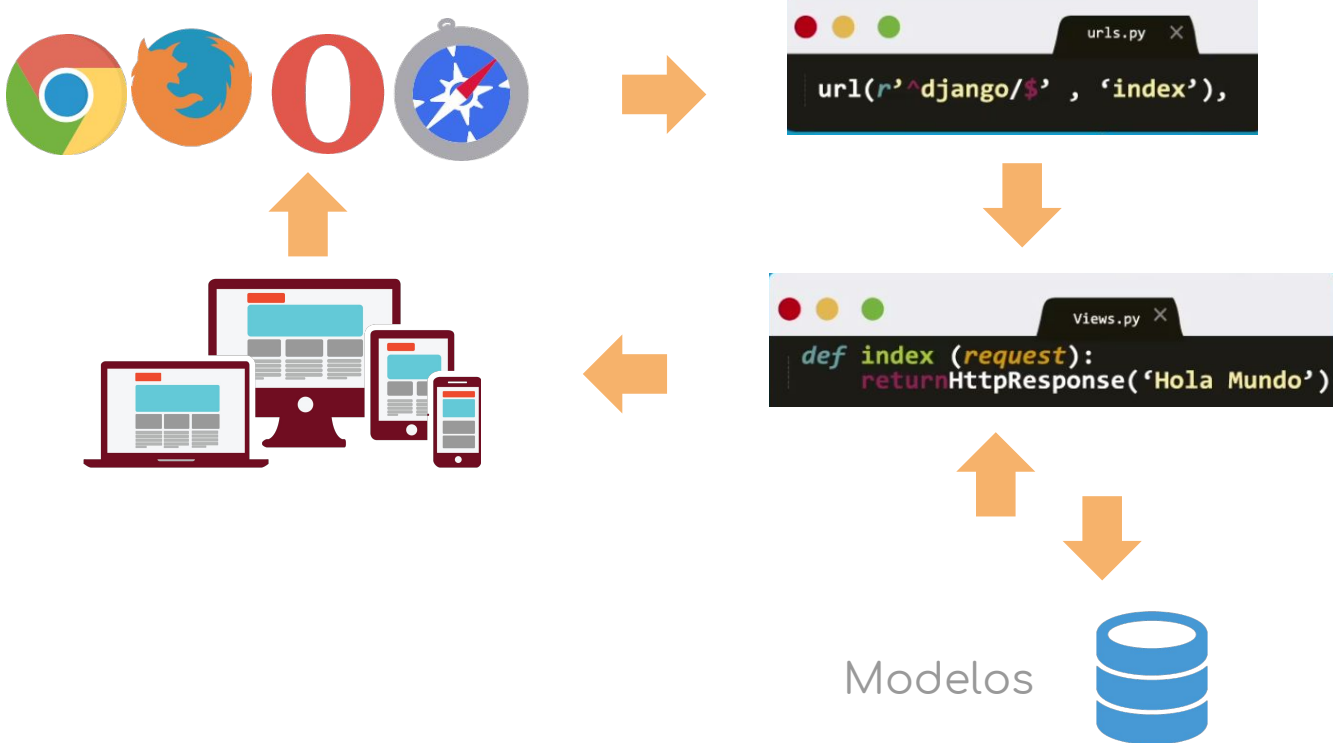
Django



Proyecto
python 3.5

```
# Crear proyecto:  
$ django-admin.py startproject ProyectoDjango  
# Iniciar servidor:  
python manage.py runserver  
# Iniciar Shell:  
python manage.py shell  
# Añadir app:  
python manage.py startapp myApp  
# Añadir migración:  
python manage.py makemigrations  
# Aplicar migraciones:  
python manage.py migrate
```

¿Como funciona Django?





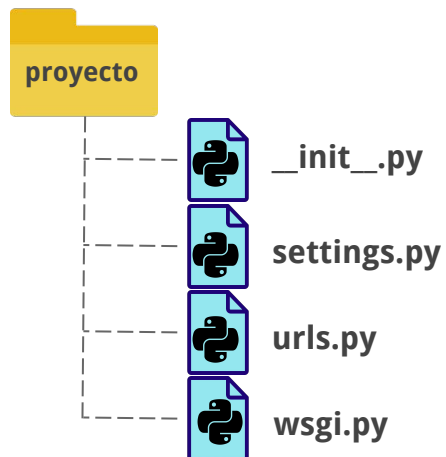
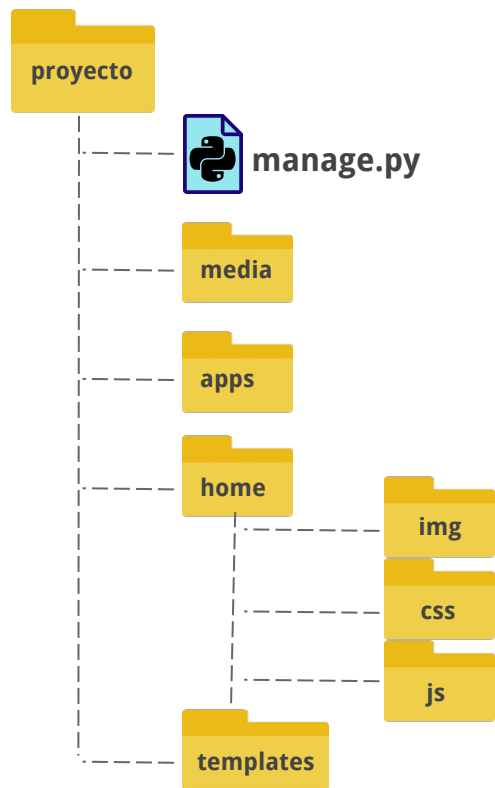
Configuración de un Proyecto Django



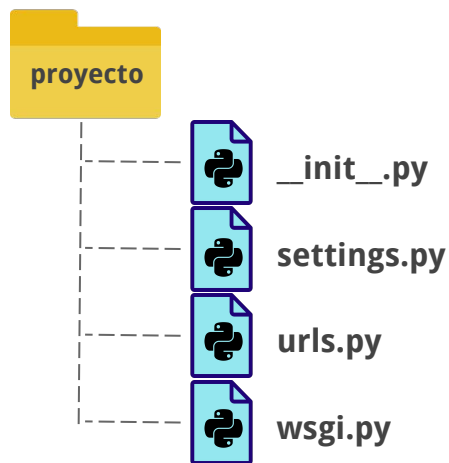
INICIAR UN PROYECTO DJANGO

(env) CursoDjango \$ **django-admin** startproject **proyecto**

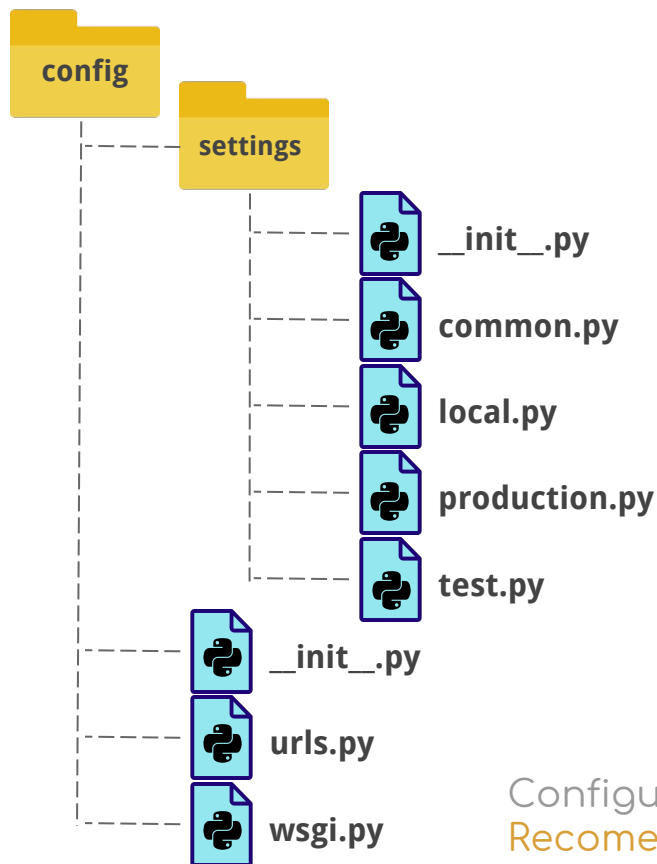
ESTRUCTURA DE UN PROYECTO DJANGO



CONFIGURACIÓN DE UN PROYECTO DJANGO



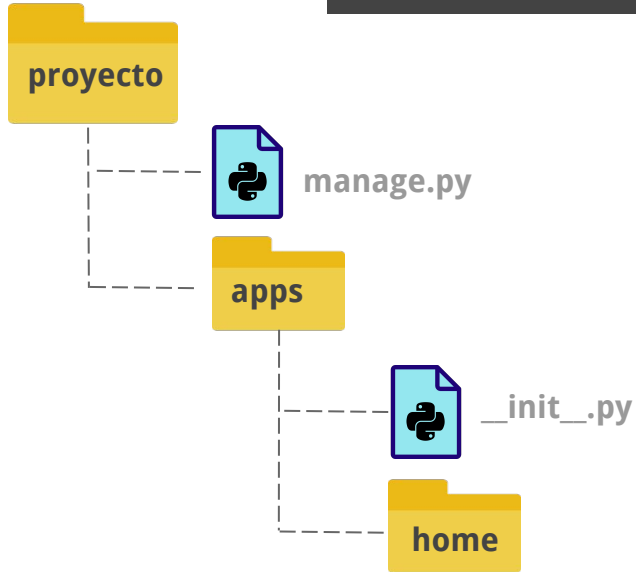
Configuración por defecto



Configuración
Recomendada

CREAR UNA APP EN DJANGO

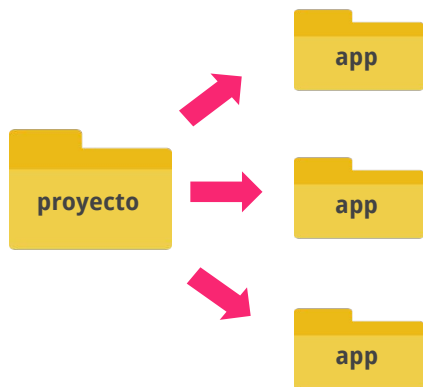
```
(env) CursoDjango/apps$ django-admin startapp home
```



ESTRUCTURA DE UNA APP DJANGO



CONFIGURACIÓN DE UNA APP DJANGO



settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'django.contrib.humanize',  
    'easy_pdf',  
    'apps.usuario',  
    'apps.cotizacion',  
    'apps.venta',  
]
```

RUTAS Y VISTAS EN DJANGO

```
from django.conf.urls.defaults import *
from mysite.views import index_view
urlpatterns = patterns('',
    (r'^index/$', index_view),
)
```

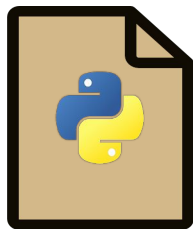
urls.py

```
from django.http import HttpResponse
def index_view(request):
    html = "Bienvenido al curso"
    return HttpResponse(html)
```

views.py



Capa de Datos



models.py

MODELOS DJANGO

"Model" (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.

Modelos

DJANGO

La definicion de `modelos` se hace en `models.py`

```
from django.db import models
class Category(models.Model):
    name = models.CharField(max_length=50)
    def __str__(self):
        return self.name

class Photo(models.Model):
    category = models.ForeignKey(Category, null=True, blank=True)
    title = models.CharField(max_length=50, default='No title')
    photo = models.ImageField(upload_to='photos/')
    pub_date = models.DateField(auto_now_add=True)
    favorite = models.BooleanField(default=False)
    comment = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.title
```

Migraciones

Las **migraciones** permiten trabajar con el schema de datos, como se haría con el código

```
# Creando un migración

(env)CursoDjango@~/apps: python manage.py makemigrations album
Migrations for 'album':
  0001_initial.py:
    - Create model Category
    - Create model Photo
```

APLICANDO UNA MIGRACIÓN

```
(env) CursoDjango@~/apps: python manage.py migrate
```

```
Operations to perform:
```

```
  Apply all migrations: admin, album, contenttypes, auth, sessions
```

```
Running migrations:
```

```
  Applying album.0001_initial... OK
```

CONSULTAS EN DJANGO

QUERY

```
books = Book.objects.all()  
books = Book.objects.all()[:100]  
books = Book.objects.all()[100:]
```

INSERT

```
book = Book(nombre='Art of war')  
book.save()
```

UPDATE

```
Book.objects.all().update(disponible=False)
```

DELETE

```
Book.objects.all().delete()
```

GET

```
Book.Objects.get(id='36')  
Book.Objects.get(nombre='Art of war')
```

FILTROS

```
Book.Objects.filter(disponible=True)  
Book.Objects.exclude(disponible=True)
```

Relaciones

Uno a Uno

```
class Autor(models.Model):  
    ...  
class Libro(models.Model):  
    autor = models.OneToOneField(Autor)
```

Clave Foránea

```
class Blog(models.Model):  
    ...  
class Post(models.Model):  
    blog = models.ForeignKey(Blog)
```

Muchos a
muchos

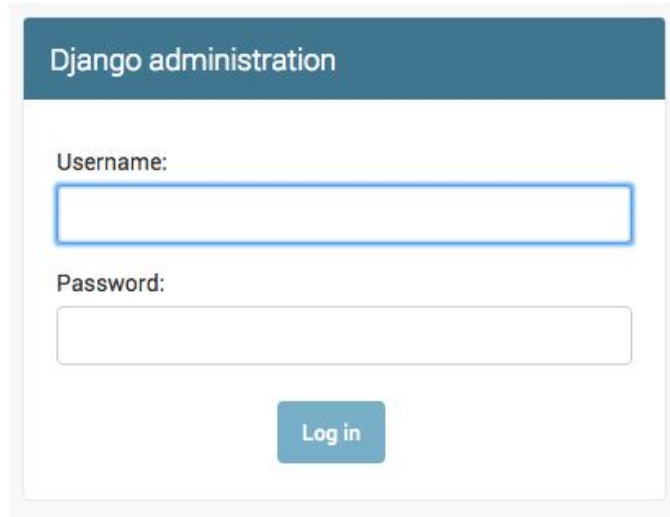
```
class tags(models.Model):  
    ...  
class Post(models.Model):  
    autor = models.ManyToManyField(tags)
```

ADMINISTRACIÓN EN DJANGO



admin.py

Django nos proporciona un panel de administración para nuestro proyecto el cual nos permite CREAR, EDITAR, MODIFICAR ELIMINAR, (CRUD) en el que podemos registrar nuestras aplicaciones.



The screenshot shows the Django administration login interface. It features a dark blue header with the text 'Django administration'. Below the header, there is a light gray box containing the login form. The form has two input fields: 'Username:' and 'Password:'. Below the password field is a blue 'Log in' button.

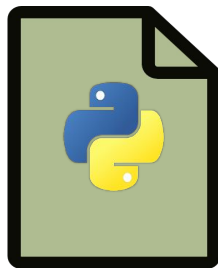
Registro de un modelo en el admin de Django

```
from django.contrib import admin
from .models import Staff

@admin.register(Staff)
class StaffAdmin(admin.ModelAdmin):
    list_display = ["user" , "position" , "cedula"]
    class Meta:
        model = Staff
```




Capa de la lógica



models.py

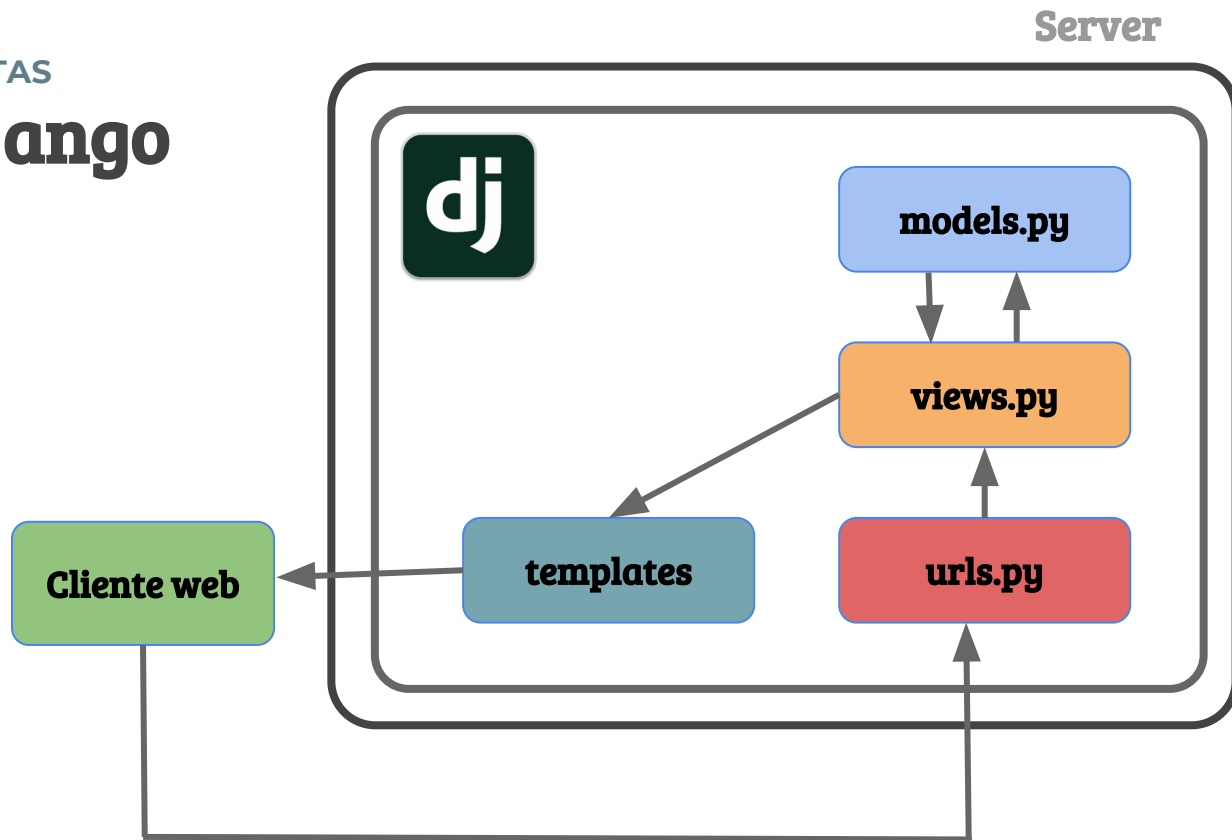
Vistas

DJANGO

"View" (Vista), esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada.

VISTAS

Django



VISTAS

Django

Vistas basadas
en funciones

```
from django.http import HttpResponse
def index_view(request):
    html = "Bienvenido al curso"
    return HttpResponse(html)
```

Vistas basadas
en clases

```
from django.views.generic import DetailView
from books.models import Publisher, Book

class PublisherDetailView(DetailView):

    context_object_name = "publisher"
    queryset = Publisher.objects.all()
```

Formularios

Django proporciona una clase auxiliar que le permite crear una clase de formulario a partir de un modelo de Django.

```
from django import forms

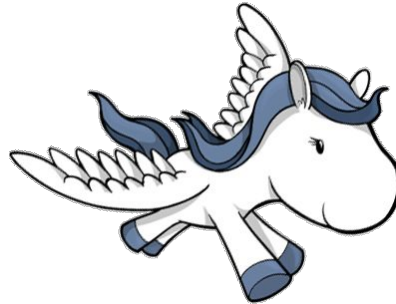
from apps.delivery.models import Product, Order

class ProductForm(forms.ModelForm):
    class Meta:
        model = Product
        fields = "__all__"
```

VISTAS BASADAS EN

Clases Django

- [django.views.generic.edit.FormView](#)
- [django.views.generic.edit.CreateView](#)
- [django.views.generic.edit.UpdateView](#)
- [django.views.generic.edit.DeleteView](#)



VISTAS BASADAS EN CLASE

FormView

```
from django import forms
class ContactForm(forms.Form):
    name = forms.CharField()
    message = forms.CharField(widget=forms.Textarea)

    def send_email(self):
        # send email using the self.cleaned_data dictionary
        pass
```

VISTAS BASADAS EN CLASE

CreateView

```
from django.views.generic.edit import CreateView
from myapp.models import Author
```

```
class AuthorCreate(CreateView):
    model = Author
    fields = ['name']
```

```
<form action="" method="post">{% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save" />
</form>
```


VISTAS BASADAS EN CLASE

UpdateView

```
from django.views.generic.edit import UpdateView
from myapp.models import Author
```

```
class AuthorUpdate(UpdateView):
    model = Author
    fields = ['name']
    template_name_suffix = '_update_form'
```

```
<form action="" method="post">{% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Update" />
</form>
```

VISTAS BASADAS EN CLASE

DeleteView

```
from django.views.generic.edit import DeleteView
from myapp.models import Author
```

```
class AuthorUpdate(DeleteView):
    model = Author
    success_url = reverse_lazy('_update_form')
```

```
<form action="" method="post">{% csrf_token %}
    <p> Are you sure you want to delete "{{ object }}"?</p>
    <input type="submit" value="Confirm" />
</form>
```



Capa de la vista

Señales

Permite enviar señales a un remitente o a un conjunto de *receptores*. Son especialmente útiles cuando muchas piezas de código pueden estar interesados en los mismos eventos.





templates.html

Plantillas

DJANGO

"Template" (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación.

PLANTILLAS

Django



Para generar plantillas se utilizan dos tipos de objetos: **Template** y **Context**.

- **Template** contiene el string de salida que queremos devolver en el `HttpResponse` (normalmente HTML)
- **Context** contiene un diccionario con los valores que dan contexto a una plantilla

```
Context {'user': 'admin'}  
Template = "Bienvenido, {{ user }}."
```



"Bienvenido, Victor."

PLANTILLAS Django



settings.py

```
TEMPLATE_DIRS = (  
    '/home/django/templates',  
)
```

```
from django.http import HttpResponse  
from django.template.loader import  
get_template  
from django.template import Context  
from datetime import datetime  
def hora_actual(request):  
    ahora = datetime.now()  
    t = get_template('hora.html')  
    c = Context({'hora': ahora})  
    html = t.render(c)  
    return HttpResponse(html)
```

ETIQUETAS Y Filtros

Los **template tags** nos permiten ejecutar código **python** dentro de las plantillas.

```
filter {{ variable|filter }}

inline tag
{% tag var1 var2 %}

block tag

{% tag var1 %}
...
{% endtag %}
```


Tags

```
{% if username == "Alfredo" %}  
    Hola Alfredo  
{% else %}  
Hola {{ usuario }},  
{% endif %}
```

```
{% for elemento in lista %}  
    <li>{{ elemento }}</li>  
{% endfor %}
```

Filtros

```
{{ username|length }}
```

```
{{ username|upper }}
```

```
{{ username|wordcount }}
```

```
{{ fecha|date:"d M Y" }}
```

```
{{ fecha|timesince }}
```

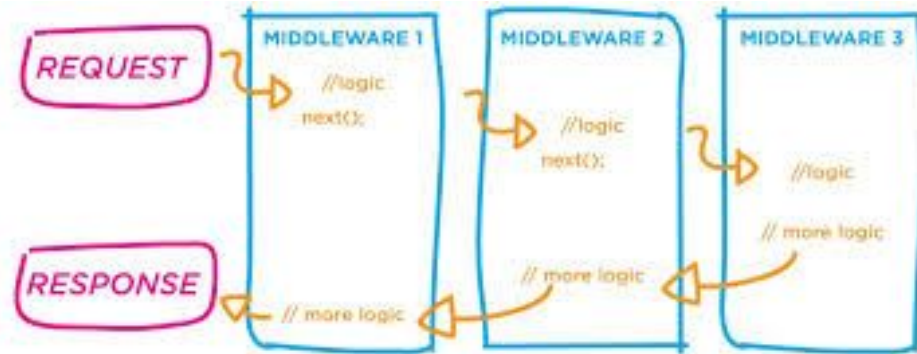


Utilidades Django

Middlewares

En ocasiones, necesitarás ejecutar una pieza de código en todas las peticiones que maneja Django.

Tu puedes hacer esto con el framework [middleware](#) de Django, que es un conjunto de acoples dentro del procesamiento de petición/respuesta de Django.



Middlewares

filter_ip_middleware.py

```
class FilterIPMiddleware(object):
    # Check if client IP is allowed
    def process_request(self, request):
        # Lista de IP's autorizadas
        allowed_ips = ['192.168.1.1', '123.123.123.123', etc...]
        # Get client IP
        ip = request.META.get('REMOTE_ADDR')
        if ip not in allowed_ips:
            # Aquí generamos un error de "forbidden site"
            raise Http403
            # Si no es ninguna, no hacemos nada
            return None
```

Middleware

El último paso sería buscar en el settings.py la línea MIDDLEWARE_CLASSES y añadimos al final de todo la llamada a nuestro middleware, tendría que quedar algo así:

```
MIDDLEWARE_CLASSES = (  
    'django.middleware.common.CommonMiddleware ',  
    'django.contrib.sessions.middleware.SessionMiddleware ',  
    'django.middleware.csrf.CsrfViewMiddleware ',  
    'django.contrib.auth.middleware.AuthenticationMiddleware ',  
    'django.contrib.messages.middleware.MessageMiddleware ',  
    # Los middleware de arriba son del settings  
    # A continuación escribimos nuestro middleware  
    'myproject.middleware.filter_ip_middleware.FilterIPMiddleware '  
)
```

Señales

Django provee un conjunto de señales integradas que permiten que el usuario reciba un aviso por Django mismo en respuesta a ciertas acciones. Estos incluyen algunas notificaciones útiles:

[Django.db.models.signals](#)

❏ `django.db.models.signals.post_save`

```
from django.contrib.auth.models import User

from django.db.models.signals import post_save

def save_profile(sender, instance, **kwargs):
    instance.profile.save()

post_save.connect(save_profile, sender=User)
```

DJANGO

Comunidad

Django cuenta con una comunidad muy grande, que constantemente actualizan los paquetes.



<https://djangopackages.org/>

<http://awesome-django.com/>