



GeeksHubs
academy _

DJANGO



Django Framework



¿Qué es Django?

Django es un framework web de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles. Desarrollado por programadores para programadores.

Es gratuito y de código abierto, tiene una comunidad próspera y activa, una gran documentación y muchas opciones de soporte gratuito y de pago.



Origen

Django fue desarrollado inicialmente entre 2003 y 2005 por un equipo que era responsable de crear y mantener sitios web de periódicos.

Este código común se convirtió en un framework web genérico, que fue de código abierto, conocido como proyecto "Django" en julio de 2005.

django



Características



Características de Django:

1. Completo
2. Versátil
3. Escalable
4. Seguro
5. Multiplataforma
6. Dogmático???

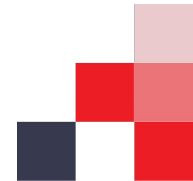


Aplicaciones que utilizan Django



Instagram

NETFLIX



Django makes it easier to build better Web
apps more quickly and with less code.

Get started with Django

Meet Django

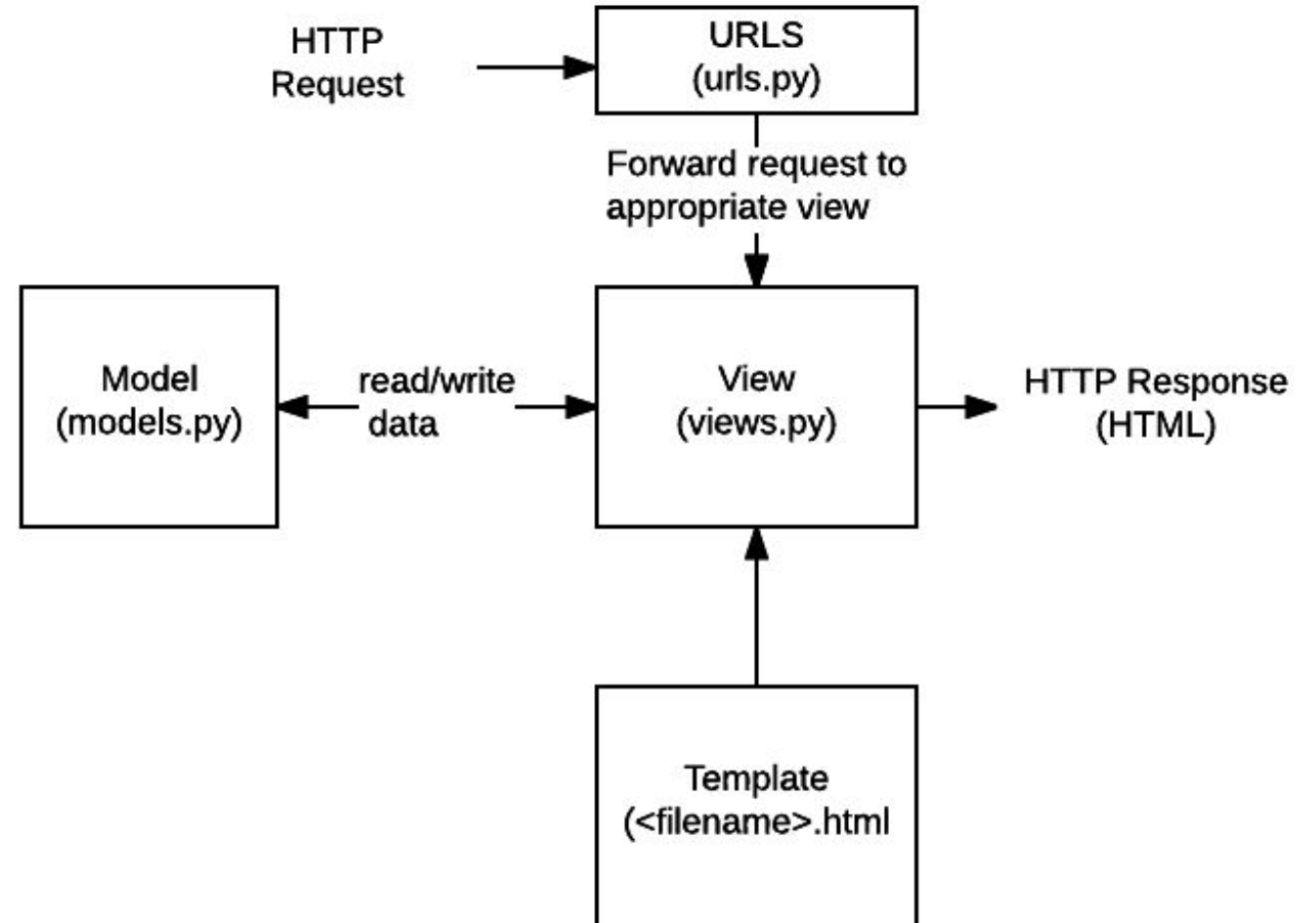
<https://www.djangoproject.com/>

Download latest release: 3.2.2

Estructura de Django

Las aplicaciones web de Django normalmente agrupan el código que gestiona cada uno de estos pasos en ficheros separados:

Patrón Modelo Vista Plantilla
MVT





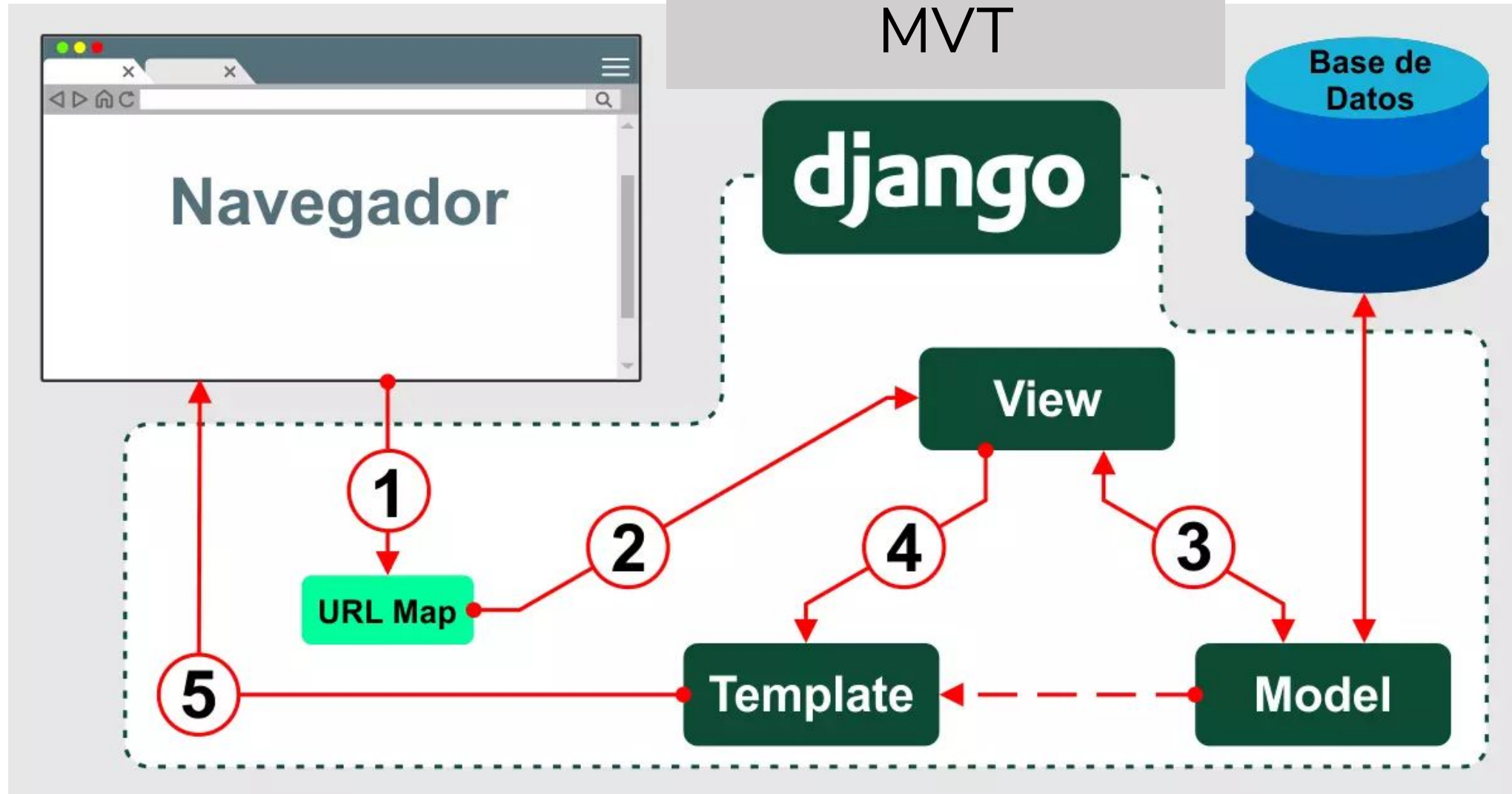
GeeksHubs
academy _

/ Desarrollamos { talento }

Funcionamiento de Django



Patrón Modelo Vista Plantilla MVT



URLs

Aunque es posible procesar peticiones de cada URL individual vía una función individual, es mucho más sostenible escribir una función de visualización separada para cada recurso.

Se usa un mapeador URL para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición.

El mapeador URL se usa para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición.

El mapeador URL puede también emparejar patrones de cadenas o dígitos específicos que aparecen en una URL y los pasan a la función de visualización como datos.



Vista (View)



Una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP.

Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de modelos, y delegan el formateo de la respuesta a las plantillas ("templates").



Modelos (Models)

Los Modelos son objetos de Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos.



django



Plantillas (Templates)

Una plantilla (template) es un fichero de texto que define la estructura o diagrama de otro fichero (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real.

Una vista puede crear dinámicamente una página usando una plantilla, rellenándola con datos de un modelo. Una plantilla se puede usar para definir la estructura de cualquier tipo de fichero; ¡no tiene porqué ser HTML!



Enviar la petición a la vista correcta (urls.py)

Un mapeador URL está normalmente almacenado en un fichero llamado `urls.py`. En el ejemplo más abajo el mapeador (`urlpatterns`) define una lista de mapeos entre patrones URL específicos y sus correspondientes funciones de visualización.

```
urlpatterns = [  
    url(r'^$', views.index),  
    url(r'^([0-9]+)/$', views.best),  
]
```



Manejar la petición (views.py)

Las vistas son el corazón de la aplicación web, recibiendo peticiones HTTP de los clientes web y devolviendo respuestas HTTP. Entre éstas, organizan los otros recursos del framework para acceder a las bases de datos, consolidar plantillas, etc.

```
## fichero: views.py (funciones de visualizacion de Django)
from django.http import HttpResponse

def index(request):
    # Obtener un HttpRequest - el parametro peticion
    # Realizar operaciones usando la infomracion de la peticion.
    # Devolver una HttpResponse
    return HttpResponse('!Hola desde Django!')
```



Definir modelos de datos (models.py)

```
# filename: models.py

from django.db import models

class Team(models.Model):
    team_name = models.CharField(max_length=40)

    TEAM_LEVELS = (
        ('U09', 'Under 09s'),
        ('U10', 'Under 10s'),
        ('U11', 'Under 11s'),
        ... #list other team levels
    )
    team_level = models.CharField(max_length=3, choices=TEAM_LEVELS, default='U11')
```



Consultar datos (views.py)

El modelo de Django proporciona una API de consulta simple para buscar en la base de datos. Esta puede buscar coincidencias contra varios campos al mismo tiempo usando diferentes criterios

```
## filename: views.py

from django.shortcuts import render
from .models import Team

def index(request):
    list_teams = Team.objects.filter(team_level__exact="U09")
    context = {'youngest_teams': list_teams}
    return render(request, '/best/index.html', context)
```



Renderización de los datos (plantillas HTML)

Los sistemas de plantillas permiten especificar la estructura de un documento de salida usando marcadores de posición para los datos que serán rellenados cuando se genere la página.

Las plantillas se usan con frecuencia para crear HTML, también pueden crear otros tipos de documentos. Django soporta de fábrica tanto su sistema de plantillas nativo como otra biblioteca Python popular llamada **Jinja2**.



<https://jinja.palletsprojects.com/en/2.11.x/>



Renderización de los datos (plantillas HTML)

```
## filename: best/templates/best/index.html

<!DOCTYPE html>
<html lang="en">
<body>

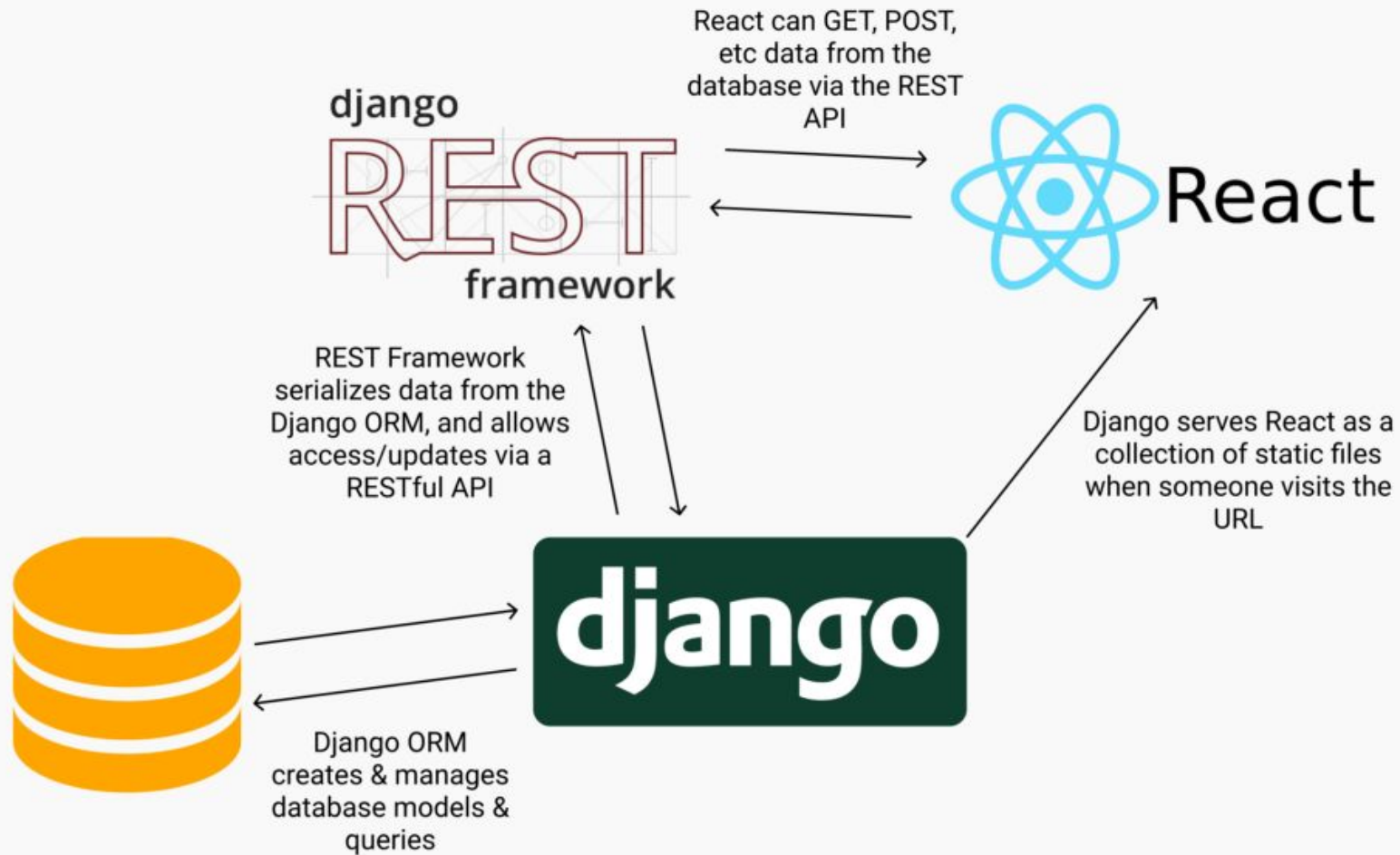
    {% if youngest_teams %}
        <ul>
            {% for team in youngest_teams %}
                <li>{{ team.team_name }}</li>
            {% endfor %}
        </ul>
    {% else %}
        <p>No teams are available.</p>
    {% endif %}

</body>
</html>
```



Mi primer proyecto Django





Creación API Django - To Do List



1. Crear/Configurar proyecto Django.
2. Crear un modelo en la base de datos que administrará Django ORM.
3. Configurar Django REST Framework.
4. Serializar el modelo de datos.
5. Crear URI endpoints para ver los datos serializados.



1. Crear/Configurar proyecto Django.

```
$ pyenv virtualenv django-rest
```

```
Looking in links: /tmp/tmpjizkdypn
```

```
Requirement already satisfied: setuptools in /home/bennett/.pyenv  
/versions/3.6.8/envs/django-rest/lib/python3.6/site-packages (40.6.2)
```

```
Requirement already satisfied: pip in /home/bennett/.pyenv/versions  
/3.6.8/envs/django-rest/lib/python3.6/site-packages (18.1)
```

```
$ pyenv local django-rest
```



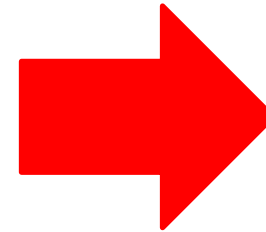
1. Crear/Configurar proyecto Django.

```
$ pip install django

$ django-admin startproject mysite

$ ls
mysite/

$ cd mysite/
$ ls
manage.py* mysite/
```



Run Server

```
$ python manage.py runserver
```



Run Server

```
$ python manage.py runserver
```

```
Watching for file changes with StatReloader  
Performing system checks...
```

```
System check identified no issues (0 silenced).
```

```
You have 17 unapplied migration(s). Your project may not work  
properly until you apply the migrations for app(s): admin, auth,  
contenttypes, sessions.  
Run 'python manage.py migrate' to apply them.
```

```
May 17, 2019 - 16:09:28  
Django version 2.2.1, using settings 'mysite.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

django

[View release notes](#) for Django 2.2

The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in
your settings file and you have not configured any
URLs.



Django Documentation
Topics, references, & how-to's



Tutorial: A Polling App
Get started with Django



Django Community
Connect, get help, or contribute



1.1. Crear API app

Podríamos construir nuestra aplicación con la estructura de carpetas tal como está ahora. Sin embargo, la mejor práctica es separar su proyecto Django en aplicaciones separadas cuando crea algo nuevo.

```
$ python manage.py startapp myapi  
$ ls  
db.sqlite3  manage.py*  myapi/  mysite/
```



1.2. Registrar app en el proyecto

Necesitamos decirle a Django que reconozca esta nueva aplicación que acabamos de crear. Los pasos que hagamos más adelante no funcionarán si Django no conoce *myapi*.

Por tanto hay que editar el siguiente archivo Python:  `mysite/settings.py`

```
INSTALLED_APPS = [  
    'myapi.apps.MyapiConfig',  
    ... # Leave all the other INSTALLED_APPS  
]
```



1.3. Migrar BBDD

Django te permite definir modelos de bases de datos usando Python.

Siempre que creamos o hacemos cambios a un modelo, necesitamos decirle a Django que migre esos cambios a la base de datos. El ORM de Django luego escribe todos los comandos SQL CREATE TABLE para nosotros.

Code First

```
$ python manage.py migrate
```

```
Operations to perform:
```

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

```
Running migrations:
```

```
  Applying contenttypes.0001_initial... OK
```

```
  Applying auth.0001_initial... OK
```

```
  Applying admin.0001_initial... OK
```

```
  Applying admin.0002_logentry_remove_auto_add... OK
```

```
  Applying admin.0003_logentry_add_action_flag_choices... OK
```

```
  Applying contenttypes.0002_remove_content_type_name... OK
```

```
  Applying auth.0002_alter_permission_name_max_length... OK
```

```
  Applying auth.0003_alter_user_email_max_length... OK
```

```
  Applying auth.0004_alter_user_username_opts... OK
```

```
  Applying auth.0005_alter_user_last_login_null... OK
```

```
  Applying auth.0006_require_contenttypes_0002... OK
```

```
  Applying auth.0007_alter_validators_add_error_messages... OK
```

```
  Applying auth.0008_alter_user_username_max_length... OK
```

```
  Applying auth.0009_alter_user_last_name_max_length... OK
```

```
  Applying auth.0010_alter_group_name_max_length... OK
```

```
  Applying auth.0011_update_proxy_permissions... OK
```

```
  Applying sessions.0001_initial... OK
```

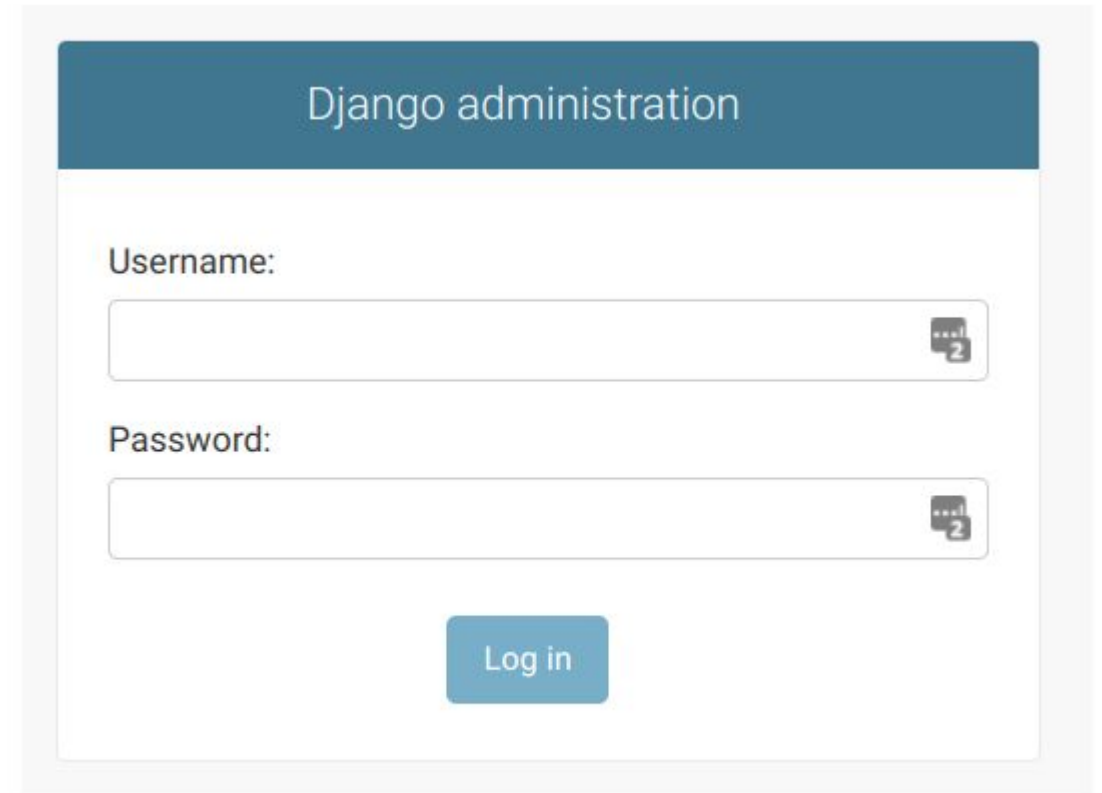

1.4. Creación de Super User

```
$ python manage.py createsuperuser
```

Run Server

```
$ python manage.py runserver
```

localhost:8000/admin



Oooo, Django Admin!!! Pretty.



Django administration

WELCOME, JOSE.

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#) [Change](#)

Users

[+ Add](#) [Change](#)

MYAPI

Heros

[+ Add](#) [Change](#)

Recent actions

My actions

[+ Black Widows](#)
Hero

[+ Storm](#)
Hero

[+ Spiderman](#)
Hero

[+ Superman](#)
Hero

[+ Batman](#)
Hero

[✗ jmarin](#)
User

[+ jmarin](#)
User

2. Crear modelo de datos

myapi/models.py

```
# models.py
from django.db import models

class Hero(models.Model):
    name = models.CharField(max_length=60)
    alias = models.CharField(max_length=60)

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



2.1. Migrar Modelo de datos

```
$ python manage.py makemigrations
```

```
Migrations for 'myapi':  
  myapi/migrations/0001_initial.py  
    - Create model Hero
```

```
$ python manage.py migrate
```

```
Operations to perform:
```

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

```
Running migrations:
```

```
  Applying myapi.0001_initial... OK
```



2.2. Registrar model en admin site

El sitio de administración de Django no sabe que existe el modelo Hero, por ese motivo, debemos añadir el modelo Hero a la definición del site.

Abrimos **myapi/admin.py** y añadimos lo siguiente:

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

admin.site.register(Hero)
```





Django administration

WELCOME, JOSE.

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add ✎ Change

Users

+ Add ✎ Change

MYAPI

Heros

+ Add ✎ Change

Recent actions

My actions

+ Black Widows
Hero

+ Storm
Hero

+ Spiderman
Hero

+ Superman
Hero

+ Batman
Hero

✗ jmarin
User

+ jmarin
User

Run Server

```
$ python manage.py runserver
```


2.3. Añadir registros

Django administration

WELCOME, JOSE. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home » Myapi » Heros » Add hero

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

MYAPI

Heros [+ Add](#)

Add hero

Name:

Alias:

Save and add another

Save and continue editing

SAVE

3. Configurar Django REST Framework.

Necesitamos serializar los datos de nuestra base de datos a través de los endpoints. Para hacer eso, necesitaremos Django REST Framework.

```
$ pip install djangorestframework
```

Una vez instalado tenemos que decirle a Django que hemos incorporado este framework a las dependencias de la aplicación. Para ello vamos a **mysite/settings.py** y añadimos los siguiente:

```
INSTALLED_APPS = [  
    # All your installed apps stay the same  
    ...  
    'rest_framework',  
]
```

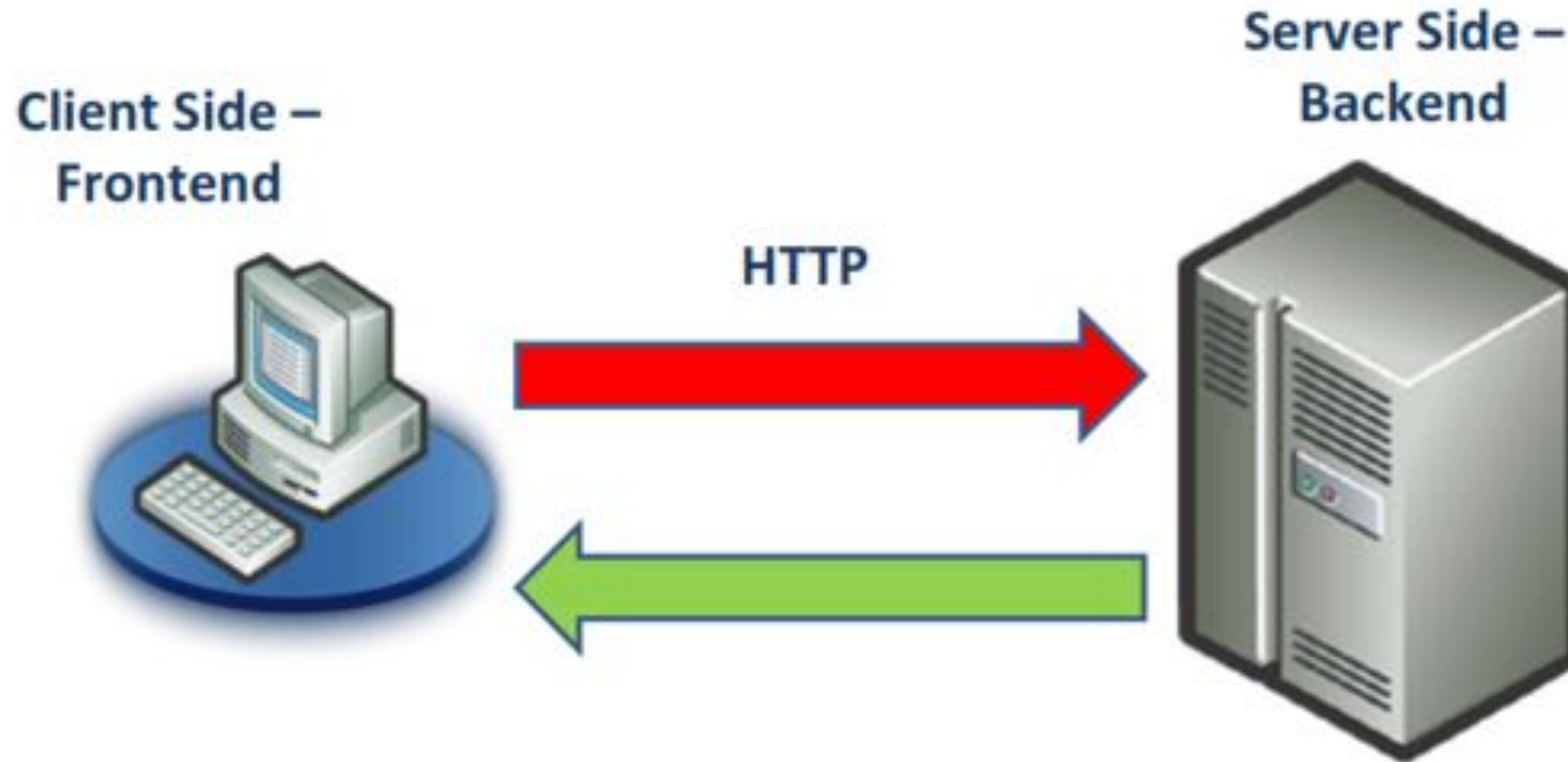


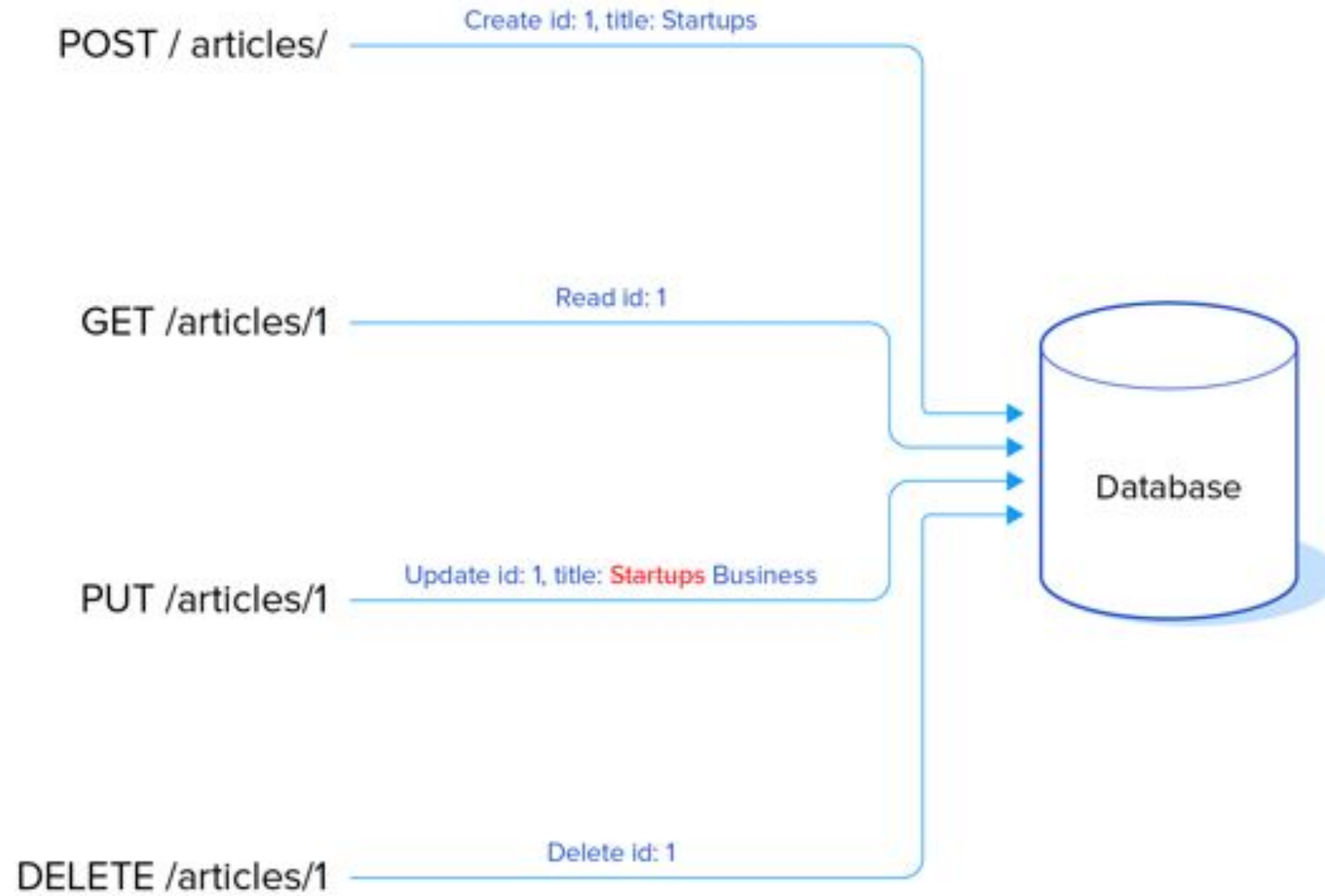
4. Serializar el modelo de datos.

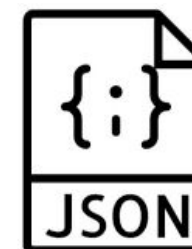


¿Qué es la serialización?









MAIN JSON OBJECT

ARRAY OF OBJECTS

```
{  
  "doorsWindows": [  
    {  
      "ID": "A",  
      "style": "roller",  
      "height": 3,  
      "width": 3,  
      "wall": "front",  
      "bay": 2,  
      "location": [0.3,0],  
      "dimensions": false  
    },  
    {  
      "ID": "B",  
      "style": "zincPA",  
      "width": 0.9,  
      "openingSide": "out",  
      "hingePost": "right",  
      "wall": "intWall_1",  
      "bay": 4,  
      "location": [4,0],  
      "dimensions": true  
    }  
  ]  
}
```

KEY ⇒ [ARRAY OF
NUMBERS]

OBJECT WITH KEYS



VALUE PAIRS



Serialización



La serialización es el proceso de convertir un objeto en una secuencia de bytes para almacenarlo o transmitirlo a la memoria, a una base de datos o a un archivo...

Su propósito principal es guardar el estado de un objeto para poder volver a crearlo cuando sea necesario. En nuestro caso el formato adoptado es el JSON



4. Serializar el modelo de datos.

La serialización es el proceso de convertir un modelo a JSON. El serializador convertirá a nuestros héroes en una representación JSON para que el usuario de la API pueda analizarlos, incluso si no están usando Python.

Para hacerlo, creemos un nuevo archivo: **myapi / serializers.py**

```
# serializers.py

from rest_framework import serializers

from .models import Hero

class HeroSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Hero
        fields = ('name', 'alias')
```



5. Crear URI endpoints para ver los datos serializados.

En primer lugar empezamos por la vista. Necesitamos renderizar los diferentes héroes en formato JSON. Para hacerlo, necesitamos:

1. Consultar la base de datos de todos los héroes.
2. Pasar ese conjunto de consultas de base de datos al *serializador* para que se convierta en JSON.
3. Representar los datos vía web.



5. Crear URI endpoints para ver los datos serializados.

En myapi / views.py:

```
# views.py

from rest_framework import viewsets

from .serializers import HeroSerializer
from .models import Hero

class HeroViewSet(viewsets.ModelViewSet):
    queryset = Hero.objects.all().order_by('name')
    serializer_class = HeroSerializer
```



5. Crear URI endpoints para ver los datos serializados.

En Django, las URL se resuelven primero a nivel de proyecto y las URI a nivel de API.

mysite/urls.py

```
# mysite/urls.py

from django.contrib import admin
from django.urls import path, include

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



5. Crear URI endpoints para ver los datos serializados.

myapi/urls.py

```
# myapi/urls.py

from django.urls import include, path
from rest_framework import routers
from . import views

router = routers.DefaultRouter()
router.register(r'heroes', views.HeroViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls',
namespace='rest_framework'))
]
```

Api Root

Api Root

OPTIONS

GET ▾

The default basic root view for DefaultRouter

GET /

HTTP 200 OK

Allow: GET, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{  
  "heroes": "http://127.0.0.1:8000/heroes/"  
}
```

Run Server

```
$ python manage.py runserver
```



Api Root / Hero List

Hero List

OPTIONS

GET ▾

GET /heroes/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
[
  {
    "id": 1,
    "name": "Batman",
    "alias": "Bruce Wayne"
  },
  {
    "id": 5,
    "name": "Black Widows",
```

5. URI endpoints

<http://127.0.0.1:8000/heroes/> GET, POST, HEAD, OPTIONS

<http://127.0.0.1:8000/heroes/{id}/> GET, PUT, PATCH, DELETE, HEAD, OPTIONS

