

## Contenido

1.	Creación del proyecto .....	2
2.	Creación de la base de datos .....	2
3.	Arrancar el servidor .....	2
4.	Test .....	2
5.	Creación de aplicaciones. ....	2
6.	Registro de la aplicación .....	2
7.	Checkeo .....	3
8.	Creación de la referencia a la carpeta de templates.....	3
9.	Creación de las vistas .....	3
10.	Creación de los paths .....	3
11.	Creación de los documentos html.....	4
12.	Ejecución parcial.....	4
13.	Uso de elementos estáticos .....	4
14.	Creación de los modelos de datos .....	4
15.	Generación de la persistencia .....	5
16.	Panel de administración .....	7
17.	Insertar la entidad .....	7
18.	CRUD.....	8
	Create .....	8
	Lectura de una entidad por la clave:.....	8
	Lectura de una entidad por el valor de columna:.....	8
	Lectura de todas las entidades: .....	8
	Update.....	8
	Delete .....	8
	Read con filter y like:.....	9
	Eliminación utilizando el método filter .....	9
	Modificación con filter .....	9
19.	Session.....	9
20.	Detección de método. ....	9

# CREACIÓN DE UN PROYECTO DJANGO

## 1. Creación del proyecto

```
django-admin startproject gestor_videojuegos
```

## 2. Creación de la base de datos

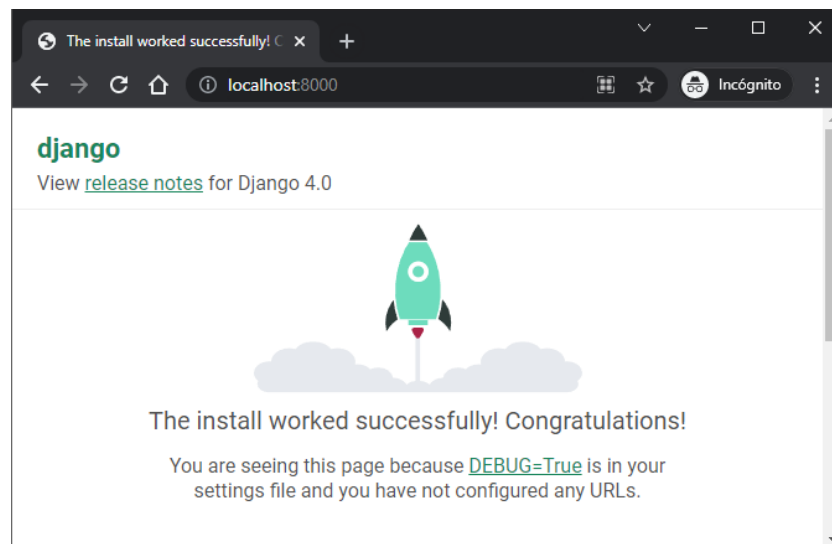
Desde la carpeta del proyecto:

```
python manage.py migrate
```

## 3. Arrancar el servidor

```
python manage.py runserver
```

## 4. Test



## 5. Creación de aplicaciones.

Un proyecto puede estar compuesto por varias aplicaciones. Una aplicación puede utilizarse en varios proyectos.

```
python manage.py startapp nombre_app
```

## 6. Registro de la aplicación

En el fichero settings.py, agregar al elemento INSTALLED\_APP la aplicación:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',
```

```

'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'nombre_app',
]

```

## 7. Checkeo

python manage.py check **nombre\_app**

## 8. Creación de la referencia a la carpeta de templates

En el fichero *settings.py* modificar la entrada *TEMPLATES* añadiendo las rutas de las plantillas a la clave *DIRS*:

```
'DIRS': ['. / nombre_app / templates', ],
```

## 9. Creación de las vistas

Modificar el fichero *views.py* de la aplicación: crear una función que genere el renderizado del *template* que se desee.

```

def home(request):
    return render(request, "home.html")

def crear_videojuego(request):
    return render(request, "crear_videojuego.html")

def mostrar_videojuegos(request):
    return render(request, "mostrar_videojuegos.html")

```

## 10. Creación de los paths

Modificación del fichero *urls.py*:

- Importación de las vistas de la aplicación:

```
from nombre_app import views
```

ó

```

from nombre_app.views import home, crear_videojuego,
mostrar_videojuegos

```

- Creación de los paths:

```

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', home),
    path('crear_videojuego/', crear_videojuego),

```

```
        path('mostrar_videojuegos/', mostrar_videojuegos),
    ] + static(settings.STATIC_URL,
               document_root=settings.STATIC_ROOT)
```

## 11. Creación de los documentos html

Dentro de la aplicación, crear la carpeta *templates* y dentro crear las páginas.

## 12. Ejecución parcial.

Desde el navegador, acceder a <http://localhost:8000/registro>

## 13. Uso de elementos estáticos

En entorno de desarrollo:

**En el fichero urls.py agregar los siguiente s import:**

```
from django.conf import settings
from django.conf.urls.static import static
```

A continuación de las urlpatterns, agregar el siguiente código:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('registro/', views.mostrar_formulario_registro),
    path('registrar/', views.registrar),
    path('verusuarios/', views.ver_usuarios)
] + static(settings.STATIC_URL,
           document_root=settings.STATIC_ROOT)
```

**En la app:**

Crear la carpeta nombre\_app/static y las subcarpetas que se deseen (css, images, etc.)

**En el template:**

Incluir al principio del documento:

```
{% load static %}
```

Incluir las referencias:

```
<link rel="stylesheet" href="{% static
'app_catalogo/css/estilo.css' %}">

```

## 14. Creación de los modelos de datos

Modificación del fichero models.py de la aplicación.

Crear una clase por cada entidad (tabla).

Ver tipos de datos en: <https://docs.djangoproject.com/en/4.0/ref/models/fields/>

```
class Credenciales(models.Model):  
    email=models.CharField(max_length=50)  
    password=models.CharField(max_length=50)
```

Algunos tipos (<https://docs.djangoproject.com/en/4.0/ref/models/fields/>):

- AutoField (Integer autoincremental)
- BinaryField
- BooleanField
- CharField
- DateField y DateTimeField
- DecimalField
- EmailField
- FilePathField
- FloatField
- IntegerField
- JSONField
- TextField
- URLField

Más parámetros de los tipos de datos:

- max\_length = longitud\_máxima
- verbose\_name = "Nombre de la columna para el panel de administración"
- blank=True → Puede ser vacío
- null=True → Puede ser nulo
- choices → Selección de un conjunto
- default → Valor por defecto
- primary\_key = True → Si no se utiliza, Django crea su propio ID PK.

## 15. Generación de la persistencia

Ver: <https://docs.djangoproject.com/en/4.0/topics/migrations/>

**python manage.py makemigrations**

Genera la base de datos vacía:

```
Migrations for 'app_catalogo':  
    app_registro\migrations\0001_initial.py  
        - Create model Credenciales
```

**python manage.py sqlmigrate app\_catalogo 0001**

Genera los scripts de base de datos. Hay que indicar el número de migración.

```

BEGIN;
--
-- Create model Credenciales
--
CREATE TABLE "app_registro_credenciales" ("id" integer NOT NULL
PRIMARY KEY AUTOINCREMENT, "email" varchar(50) NOT NULL,
"password" varchar(50) NOT NULL);
COMMIT;

```

### python manage.py migrate

Ejecuta los scripts y crea las tablas.

Operations to perform:


Apply all migrations: admin, app\_registro, 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 app_registro.0001_initial... 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 auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK

```

▼  Tablas (12)

▼  app_registro_credenciales	CREATE TABLE "app_registro_credenciales" ("id"
 id	integer "id" integer NOT NULL
 email	varchar(50) "email" varchar(50) NOT NULL
 password	varchar(50) "password" varchar(50) NOT NULL
>  auth_group	CREATE TABLE "auth_group" ("id" integer NOT N

La tabla se llama: *nombre\_aplicación\_nombre\_tabla*

Crea una clave primaria autoincremental *id(integer)*.



**Después de cada cambio del esquema (de los modelos) hay que ejecutar:**

```
python manage.py makemigrations
Python manage.py migrate
```

## 16. Panel de administración

Crear usuario:

Desde la carpeta del proyecto:

```
python manage.py createsuperuser
```

Registrar los modelos de cara a la administración:

En el fichero **admin.py**, importar los modelos:

```
from nombre_app.models import NombreModelo
```

Registrar los modelos:

```
# Register your models here.
```

```
Admin.site.register(NombreModelo)
```

Entrar en la consola de administración con el usuario creado:

<http://localhost:8000/admin>

## 17. Insertar la entidad

En la vista, importar las clases del modelo.

Crear las instancias del modelo e invocar al método `save()`.

```
from django.shortcuts import render
from app_registro.models import Credenciales

# Create your views here.

def mostrar_formulario_registro(request):
    return render(request, "registro.html")

def registrar(request):
    usuario = request.GET["email"]
    password = request.GET["password"]
    c = Credenciales(email=usuario, password=password)
    c.save()
    return render(request, "registro.html")
```

Alternativamente a crear el objeto e invocar al método `save()` se puede utilizar la siguiente notación:

```
Credenciales.objects.create(email=usuario, password=password)
```

## 18. CRUD

Para agilizar el desarrollo se pueden realizar las ejecuciones de los accesos a la base de datos desde un terminal. Para ello, hay que arrancar una shell Python con el siguiente comando:

```
python manage.py shell
```

Una vez en la Shell, se debe importar la entidad:

```
from app_registro.models import Credenciales
```

### Create

```
c = Credenciales(email="email", password="password")
c.save()
```

### Lectura de una entidad por la clave:

`credencial = Credenciales.objects.get(pk=1)` → Provoca un error de tipo `DoesNotExist` si no existe la pk.

```
Credenciales.objects.get(id=3)
```

o

```
Credenciales.objects.get(pk=3)
```

### Lectura de una entidad por el valor de columna:

```
c = Credenciales.objects.get(email="fernando.paniagua@gmail.com")
```

### Lectura de todas las entidades:

```
credenciales = Credenciales.objects.all()
```

### Update

Se debe recuperar el objeto, modificar los atributos e invocar al método `save()`.

```
c = Credenciales.objects.get(email="fernando@gmail.com")
c.password = "k4$_askd"

c.save()
```

### Delete

Se debe recuperar el objeto e invocar al método `delete()`.

```
c = Credenciales.objects.get(email="fernando@gmail.com")
c.delete()
```



Read con filter y like:

```
cs = Credenciales.objects.filter(email__contains="nan")
```

Eliminación utilizando el método filter

```
Credenciales.objects.filter(email="fernando@gmail.com").delete()
```

Modificación con filter

```
Credenciales.objects.filter(email__contains="nan").update(password="38_$x48")
```

## 19. Session

```
if ('contador' not in request.session):  
    request.session['contador']=1  
else:
```

```
request.session['contador']=request.session['contador']+1
```

## 20. Detección de método.

En las vistas de views.py:

```
if request.method=="POST"
```

Permite discriminar si el método de acceso ha sido GET o POST (primera vez, se muestra el formulario, segunda vez, se ha recibido los datos).