

Notas de Django.

Iniciando el proyecto.....	1
Instalando python.....	1
Creando un entorno virtual.....	1
Instalar django.....	2
Django en sí.....	2
¿Qué es lo que generó?.....	2
El servidor de desarrollo.....	3
Apps.....	4
Vistas.....	5
Bases de datos.....	7
Migraciones.....	9
Activar los modelos.....	10
Usando la API.....	11
Sitio administrativo default.....	14

Iniciando el proyecto

Instalando python

Primero que nada hay que instalar python, en www.python.org se halla el código, sino podemos instalar en windows 11 usando en el cmd:

winget install python.python.3.12

o en linux seria:

sudo pacman -Syu python

sudo apt install python

Creando un entorno virtual

Para crear un entorno virtual con python usamos el comando:

python -m venv .venv

para activarlo usamos:

.\.venv\bin\activate

o en linux:

source ./venv/bin/activate.sh

Instalar django

A este punto ya podemos hacer:

code .

para iniciar el editor de texto con las librerías de python cargadas.

dentro del editor usamos al terminal y ponemos:

pip install django

para chequear si django está bien instalado debemos usar:

python -m django --version

y finalmente para inicializar un proyecto de django usamos:

django-admin startproject caritas

Django en sí

¿Qué es lo que generó?

Cuando usamos el comando "startproject" django creo una carpeta con el nombre del proyecto, que contiene:

mysite/

manage.py

mysite/

__init__.py

settings.py

urls.py

asgi.py

wsgi.py

1. **manage.py**: Es una utilidad de linea de comandos para interactuar con el proyecto, ver en:
<https://docs.djangoproject.com/en/5.0/ref/django-admin/>
2. **mysite/mysite/**: es el directorio que contiene el paquete de tu proyecto. El nombre del mismo es el que vamos a usar para importar algo dentro de él: "mysite.urls".

3. **mysite/mysite/__init__.py**: Esto le dice a python que esto es un paquete.
4. **mysite/mysite/settings.py**: Configuración del proyecto de Django. Por ejemplo:
ALLOWED_HOSTS = ["www.example.com"]
DEBUG = **False**
DEFAULT_FROM_EMAIL = "webmaster@example.com"
Ver más en:
<https://docs.djangoproject.com/en/5.0/topics/settings/>
5. **mysite/mysite/urls.py**: Declaración de URL 's del proyecto, acá, a lo que el usuario va a poder acceder.
Ver más en:
<https://docs.djangoproject.com/en/5.0/topics/http/urls/>
6. **mysite/mysite/asgi.py**: Un punto de entrada para servidores que use ASGI. Ver más en:
<https://docs.djangoproject.com/en/5.0/howto/deployment/asgi/>
7. **mysite/mysite/wsgi.py**: Un punto de entrada para servidores que use WSGI. Ver más en:
<https://docs.djangoproject.com/en/5.0/howto/deployment/wsgi/>

El servidor de desarrollo

Para iniciar el servidor usamos:

python manage.py runserver

Y debería mostrar esto:

March 04, 2024 - 16:37:02

Django version 5.0.3, using settings 'caritas.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CTRL-BREAK.

Y probablemente un par de avisos más, pero si vas a la dirección y ves una página con links a explicaciones, entonces anduvo bien.

<https://docs.djangoproject.com/en/5.0/ref/django-admin/#django-admin-runserver>

Nota: El servidor se carga automáticamente cuando hay cambios en el código, pero si hay un error crashea.

Para elegir el puerto donde va el server podemos usar:

```
python manage.py runserver 8080
```

Para hacer que el servidor escuche en todas las IPs públicas (para que se pueda ver en otras computas en la red) usamos:

```
python manage.py runserver 0.0.0.0:8080
```

Apps

Un proyecto puede tener múltiples aplicaciones, las aplicaciones definen aplicaciones web, y un proyecto sería una colección de configuraciones y aplicaciones.

Ejemplo: Sistema de blog web.

Para crear una app usamos:

```
python manage.py startapp polls
```

Usamos este comando se crea una carpeta del nombre que ponemos, con las siguientes cosas:

polls/

__init__.py

admin.py

apps.py

migrations/

__init__.py

models.py

tests.py

views.py

Vistas

Una vista es lo que va a ver un usuario cuando va a una dirección dentro de nuestro sitio.

En nuestro archivo "views.py" agregamos el siguiente código:

```
def index(request):  
    return HttpResponse("Hello, world. You're at  
the polls index.")
```

Esta es la vista más simple que se puede crear.

Una vez registrada la vista, debemos registrarla en un archivo llamado "urls.py" (que debemos crear en la carpeta de nuestra app).

Debería contener el siguiente código:

```
from django.urls import path  
from . import views  
  
urlpatterns = [  
    path("", views.index, name="index"),  
]
```

Esencialmente, urlpatterns es una lista que contiene paths, el primer argumento de la función path() en este ejemplo es a qué dirección responde (en este caso "/polls/"), luego que vista será renderizada cuando vayamos ahí, y luego nombre.

Despues debemos registrar la vista en urls.py de nuestro proyecto:

```
from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path("polls/", include("polls.urls")),

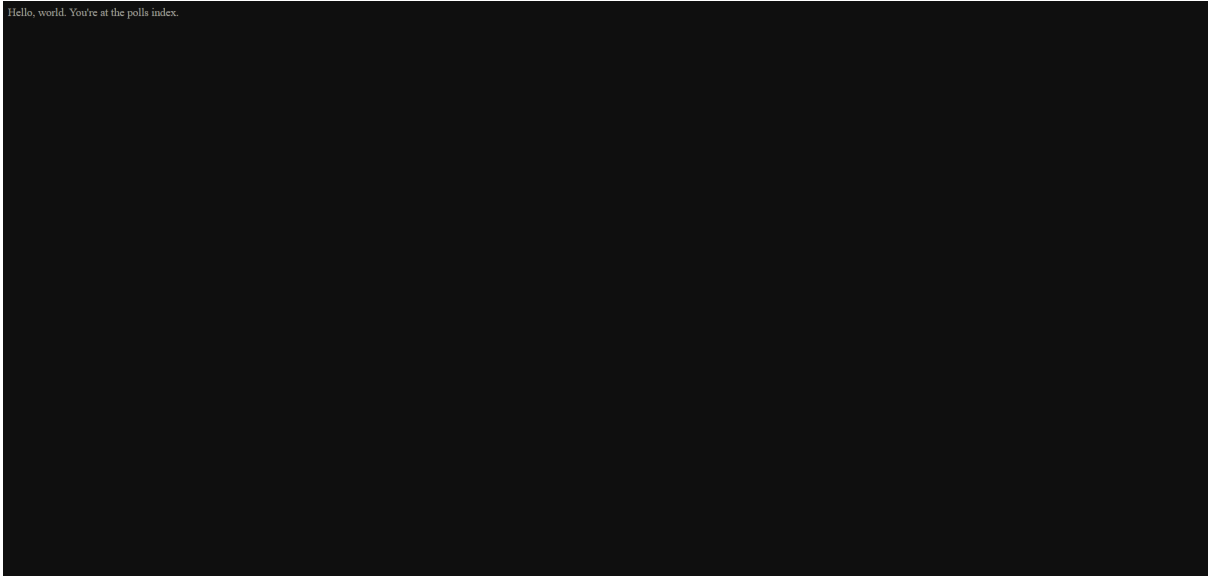
    path('admin/', admin.site.urls),

]
```

Ahora si vamos a donde está nuestro proyecto:

<http://127.0.0.1:8000/polls/>

Veremos lo siguiente:



Hello, world. You're at the polls index.

“La función `include()` permite hacer referencia a otros `URLconfs`. Cada vez que Django encuentra `include()` corta cualquier parte de la URL que coincide hasta ese punto y envía la cadena restante a la `URLconf` incluida para seguir el proceso.

La idea detrás de `include()` es facilitar la conexión y ejecución inmediata de las URLs. Dado que las encuestas están en su propia `URLconf` (`polls/urls.py`) se pueden ubicar en «`/polls/`», «`/fun_polls/`», «`/content/polls/`» o en cualquier otra ruta raíz, y la aplicación todavía seguirá funcionando.

Siempre debe usar `include()` cuando incluye otros patrones de URL. `admin.site.urls` es la única excepción a esto.”

Bases de datos

Django viene por defecto con la base de datos SQLite. Que se puede cambiar pero no nos calienta ahora.

Si queremos cambiar la base datos tenemos que buscar `database` en el archivo de `settings.py` de nuestro proyecto:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Y lo podemos cambiar a postgres así:

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": "mydatabase",
        "USER": "mydatabaseuser",
        "PASSWORD": "mypassword",
        "HOST": "127.0.0.1",
        "PORT": "5432",
    }
}
```

Como vemos cada base de datos requerirá diferentes configuraciones. Podemos ver más en <https://docs.djangoproject.com/en/5.0/ref/settings/#databases>

Importante: Cambiar el timezone de la configuración a nuestra zona horaria por defecto UTF.
(America/Argentina/Buenos_Aires en Argentina).

INSTALLED_APPS contiene los nombres de todas las aplicaciones Django que están activadas en esta instancia de Django. Las aplicaciones se pueden usar en diversos proyectos

y se pueden empaquetar y distribuir para que otras personas las utilicen en sus proyectos.

Por defecto, `INSTALLED_APPS` contiene las siguientes aplicaciones:

1. `django.contrib.admin`: El sitio administrativo. Usted lo utilizará dentro de poco.
2. `django.contrib.auth`: Un sistema de autenticación.
3. `django.contrib.contenttypes`: Un framework para los tipos de contenido.
4. `django.contrib.sessions`: Un framework de sesión.
5. `django.contrib.messages`: Un framework de mensajería.
6. `django.contrib.staticfiles`: Un framework para la gestión de archivos estáticos.

Estas aplicaciones se incluyen de forma predeterminada como una conveniencia para el caso común.

Algunas de estas aplicaciones utilizan al menos una tabla de base de datos, por lo que necesitamos crear las tablas en la base de datos antes de poder utilizarlas. Para ello, ejecute el siguiente comando:

python manage.py migrate

Esto crea las tablas necesarias de nuestra base de datos. Si no necesitamos las aplicaciones default, está bien sacarlas.

Migraciones

Django utiliza la filosofía DRY (don't repeat yourself), por tanto deriva la información de una sola fuente, lo que definimos en nuestro archivo `"models.py"`. Incluido las cosas

para actualizar la base de datos (migración = actualización de db).

De ejemplo del sitio usamos estas 2 tablas para el sitio ejemplo de encuestas:

```
class Question(models.Model):
    question_text =
models.CharField(max_length=200)
    pub_date = models.DateTimeField("date
published")

class Choice(models.Model):
    question = models.ForeignKey(Question,
on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Que lo ponemos en polls/models.py.

Cada **campo** que definimos (por ejemplo pub_date), será una columna en nuestra db y tendrá el nombre de la variable en nuestro código de python. Como vemos no definimos primary key.

Cada **campo** es de tipo Field, y algunos tipos de Field requieren argumentos.

Para hacer **relaciones** usamos models.ForeignKey, soporta conexiones de uno a muchos, uno a uno, muchos a muchos.

Activar los modelos

Para activar nuestros modelos debemos agregar nuestra aplicación a la lista de INSTALLED_APPS.

```
INSTALLED_APPS = [
```

```

    "polls.apps.PollsConfig",
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
]

```

Para commitear las migrations usamos:

python manage.py makemigrations polls

Y luego para hacerlas usamos:

python manage.py migrate

Para ver la migración que va a hacer usamos:

python manage.py sqlmigrate polls 0001

Usando la API

Para usar la api de Django en la shell podemos usar:

python manage.py shell

Y tenemos este ejemplo de como usarlo:

```

>>> from polls.models import Choice, Question #
      Import the model classes we just wrote.

      # No questions are in the system yet.
>>> Question.objects.all()
<QuerySet []>

      # Create a new Question.

```

```
# Support for time zones is enabled in the default
settings file, so
# Django expects a datetime with tzinfo for
pub_date. Use timezone.now()
# instead of datetime.datetime.now() and it will
do the right thing.
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?",
pub_date=timezone.now())

# Save the object into the database. You have to
call save() explicitly.
>>> q.save()

# Now it has an ID.
>>> q.id
1

# Access model field values via Python attributes.
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217,
tzinfo=datetime.timezone.utc)

# Change values by changing the attributes, then
calling save().
>>> q.question_text = "What's up?"
>>> q.save()
```

```
# objects.all() displays all the questions in the
database.

>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

Para generar representaciones utiles de nuestras tablas, debemos generar el "toString" de nuestra clase, sobrecargando el operador `__string__`:

```
class Question(models.Model):
    question_text =
models.CharField(max_length=200)
    pub_date = models.DateTimeField("date
published")
    def __str__(self):
        return self.question_text
```

Es importante añadir los métodos `__str__()` a los modelos, no solo para la línea de comandos, sino también porque las representaciones de objetos se usan en todo el sitio administrativo generado automáticamente de Django. Podemos agregar métodos con lógica los objetos así:

```
class Question(models.Model):
    ...
    def was_published_recently(self):
        return self.pub_date >=
datetime.datetime.now() -
datetime.timedelta(days=1)
```

Sitio administrativo default

El sitio administrativo default que da django se usa para modificar la base de datos. Primero tenemos que crear una cuenta de admin:

python manage.py createsuperuser

Para el cual nos pedirá el nombre, email y contraseña.

Cuando inicializamos el server, podemos ir a:

<http://127.0.0.1:8000/admin/>

Para entrar a la administración.

Una vez entramos, vamos a ver (si tenemos la aplicación de autenticación default puesta), 2 tablas a agregar (users, groups). Para poder modificar las tablas custom en el panel de administrador agregamos a "admin.py":

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