

Informe de Programación Web - Django 01

Tema: Modelos en Django

Nota

Estudiantes	Escuela	Asignatura
Jhonatan Benjamin Mamani Céspedes jmamanices@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Práctica	Tema	Duración
01	Modelos en Django	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 20 Mayo 2024	Al 26 Mayo 2024

1. Tarea

- Instalación, el ambiente virtual
- Crear un proyecto Django en blanco
- Configuración
- Componentes integrados
- Primer componente de aplicación
- Crear objetos en Python Shell
- Nuevos campos en el modelo

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home v 23H2 64 bits
- VIM x64 v9.1.
- Visual Studio Code x64 v1.89.1
- Git v2.45.0.
- Cuenta en GitHub con el correo institucional.
- Python v3.12.3.
- Entorno virtual.
- Django v5.0.6.

3. URL de Repositorio Github

- URL del Repositorio GitHub en donde se realiza el proyecto
- <https://github.com/JBenjamin01/pw2-django>

4. Creación del entorno virtual

4.1. Entorno virtual

- Lo primero en realizarse fue crear un entorno virtual para instalar el paquete del framework Django.
- Así que lo primero fue crear dicho entorno, se usó lo siguiente (en Windows)

Listing 1: Creación del entorno virtual

```
$ virtualenv -p python3 python_env
```

4.2. Activación

- Después de crearlo, se necesita activarlo. Para esto se usó lo siguiente:

Listing 2: Activación del entorno virtual

```
$ cd python_env  
$ .\Scripts\Activate.ps1
```

- Fuera del directorio del entorno virtual, se crea una carpeta que contendrá el proyecto:

Listing 3: Directorio de trabajo del proyecto

```
$ cd ..  
$ mkdir pw2-django
```

4.3. Dependencias

- Finalmente, solo faltaba instalar las dependencias, en este caso, solo Django, con pip freeze revisamos si se instaló correctamente.

Listing 4: Instalación del paquete Django

```
$ pip install Django
```

- Una vez instalado podemos proceder con el trabajo.

5. Creación del proyecto Django

5.1. Configuración básica

- Entonces, dentro de la ruta principal del proyecto, creo un nuevo proyecto en el cual trabajaré todo lo que se hizo en las diapositivas:

Listing 5: Inicio del proyecto

```
$ cd pw2-django
$ django-admin startproject listaContactos
```

- Tras crear el proyecto se realiza una configuración simple en listaContactos/settings.py, según las diapositivas seguidas se deben modificar el idioma y la zona horaria.

Listing 6: Ingresando a settings.py

```
$ vim listaContactos/listaContactos/settings.py
```

- Modificamos la zona horaria y el idioma:

Listing 7: Configuración de idioma y zona horaria

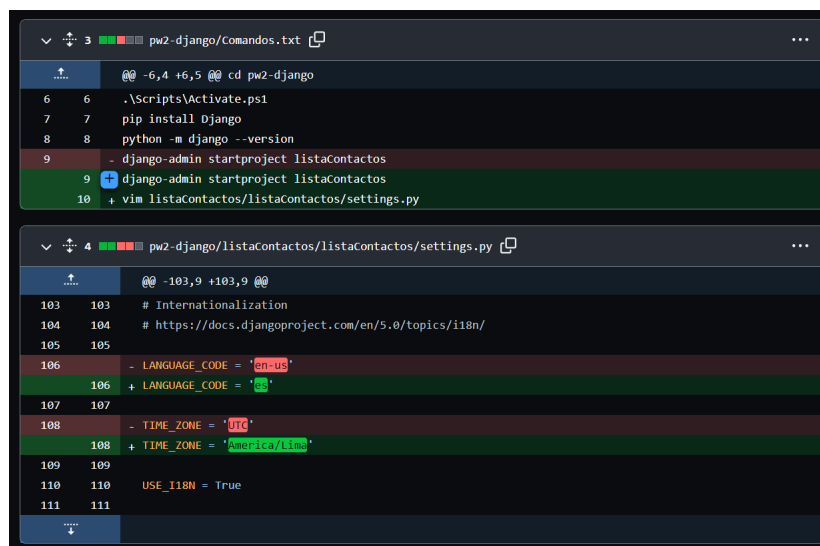
```
LANGUAGE_CODE = 'es'

TIME_ZONE = 'America/Lima'

USE_I18N = True

USE_TZ = True
```

- El commit realizado fué el siguiente:



```
pw2-django/Comandos.txt
@@ -6,4 +6,5 @@ cd pw2-django
6 6 .\Scripts\Activate.ps1
7 7 pip install Django
8 8 python -m django --version
9 - django-admin startproject listaContactos
9 + django-admin startproject listaContactos
10 + vim listaContactos/listaContactos/settings.py

pw2-django/listaContactos/listaContactos/settings.py
@@ -103,9 +103,9 @@
103 103 # Internationalization
104 104 # https://docs.djangoproject.com/en/5.0/topics/i18n/
105 105
106 - LANGUAGE_CODE = 'en-us'
106 + LANGUAGE_CODE = 'es'
107 107
108 - TIME_ZONE = 'UTC'
108 + TIME_ZONE = 'America/Lima'
109 109
110 USE_I18N = True
111 111
```

Figura 1: Commit de los cambios

5.2. Servidor

- Con este primer paso se revisa que el servidor pueda correr sin problema, entonces realizamos el siguiente comando:

Listing 8: Activación del servidor

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

- Y una vez realizado esto, accedemos al enlace que nos da el servidor para verificar su funcionamiento, en `http://127.0.0.1:8000/`

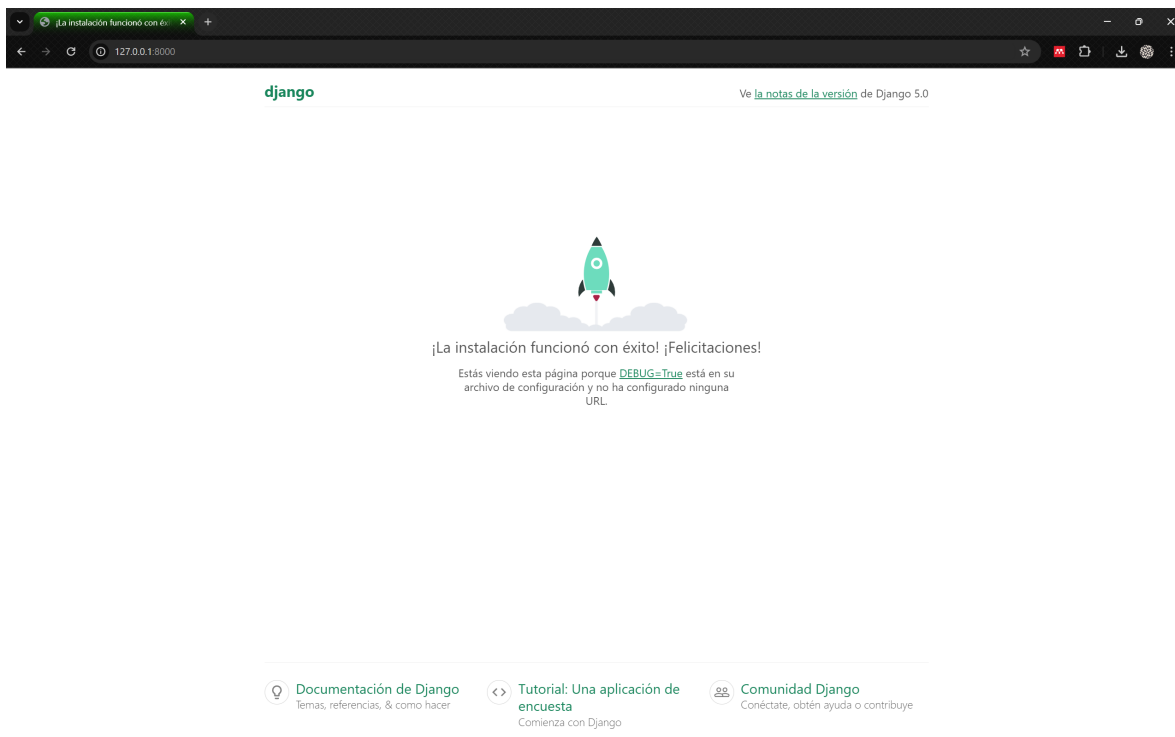


Figura 2: Página principal del servidor en Django

- Y efectivamente, parece no haber ningún problema con el servidor.

6. Creando la App 'personas'

6.1. App

- Entonces, ahora ubicados en el directorio dentro del proyecto, se procede a crear la primera app, en este caso, personas:

Listing 9: Comando para crear la app personas

```
$ django-admin startapp personas
```

6.2. Modelos

- Con la nueva aplicación, se pueden editar los archivos propios de esta, el primer paso que se sigue es la creación del modelo Persona.
- Entonces, editamos el archivo models.py de la app recién creada:

Listing 10: Ingresando a models.py

```
$ vim personas/models.py
```

- Y creamos esta nueva clase Persona para asignarla como un nuevo modelo:

Listing 11: Modelo Persona

```
from django.db import models

# Create your models here.
class Persona(models.Model):
    nombres = models.TextField()
    apellidos = models.TextField()
    edad = models.TextField()
```

- Un paso importante es modificar el archivo settings.py en el registro de las apps de nuestro proyecto, entonces nos dirigimos a este archivo:

Listing 12: Ingresando a settings.py

```
$ vim listaContactos/settings.py
```

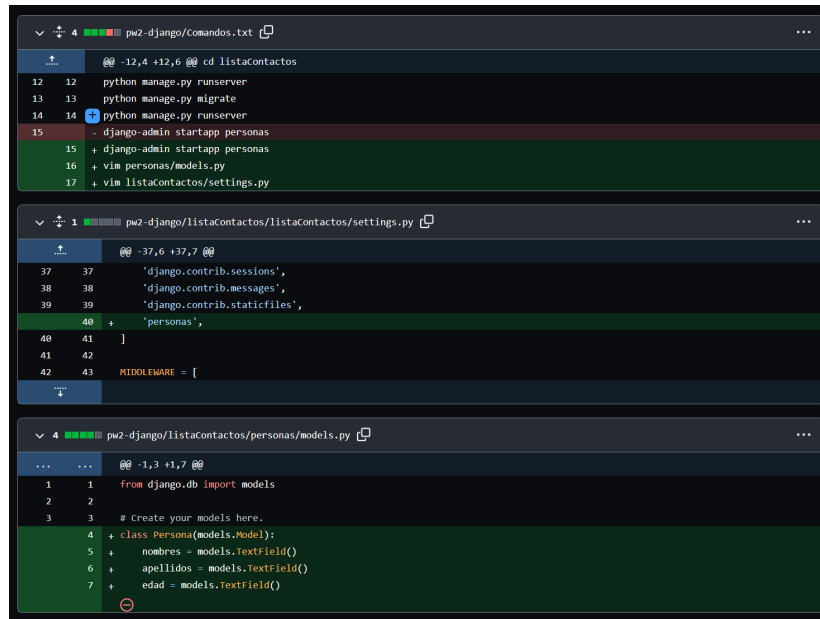
- Bajando hasta la parte de INSTALLED_APPS, agregamos la aplicación que acabamos de crear al final de la lista:

Listing 13: Aplicaciones del proyecto

```
# Application definition

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

- El commit realizado fué el siguiente:



```

pw2-django/comandos.txt
@@ -12,4 +12,6 @@ cd listacontactos
12 12 python manage.py runserver
13 13 python manage.py migrate
14 14 python manage.py runserver
15 15 - django-admin startapp personas
15 15 + django-admin startapp personas
16 16 + vim personas/models.py
17 17 + vim listacontactos/settings.py

pw2-django/listacontactos/listacontactos/settings.py
@@ -37,6 +37,7 @@
37 37 'django.contrib.sessions',
38 38 'django.contrib.messages',
39 39 'django.contrib.staticfiles',
40 40 + 'personas',
40 41 ]
41 42
42 43 MIDDLEWARE = [

pw2-django/listacontactos/personas/models.py
... -1,3 +1,7 @@
1 1 from django.db import models
2 2
3 3 # Create your models here.
4 4 + class Persona(models.Model):
5 5 +     nombres = models.TextField()
6 6 +     apellidos = models.TextField()
7 7 +     edad = models.TextField()

```

Figura 3: Commit de los cambios

6.3. Migrations

- Luego de haber realizado esta modificación a la aplicación de personas, añadiendo el nuevo modelo, se necesitan avisar sobre los cambios realizados y aplicarlos en nuestro proyecto. En los pasos de las diapositivas se siguen los siguientes comandos:

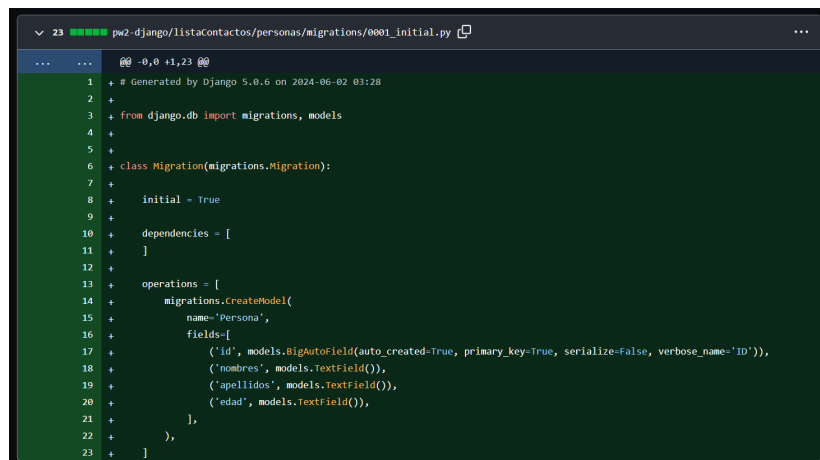
Listing 14: Makemigrations y migrate

```

$ python manage.py makemigrations
$ python manage.py migrate

```

- Al ejecutarlos, se genera un nuevo archivo '0001_initial.py' en la carpeta migrations de la app personas, esto se ve en el siguiente commit:



```

pw2-django/listacontactos/personas/migrations/0001_initial.py
... -0,0 +1,23 @@
1 + # Generated by Django 5.0.6 on 2024-06-02 03:28
2 +
3 + from django.db import migrations, models
4 +
5 +
6 + class Migration(migrations.Migration):
7 +
8 +     initial = True
9 +
10 +     dependencies = [
11 +
12 +     ]
13 +
14 +     operations = [
15 +         migrations.CreateModel(
16 +             name='Persona',
17 +             fields=[
18 +                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
19 +                 ('nombres', models.TextField()),
20 +                 ('apellidos', models.TextField()),
21 +                 ('edad', models.TextField()),
22 +             ],
23 +         ),

```

Figura 4: Migraciones aplicadas

7. Django Admin Site

7.1. Creación de nuevo usuario

- Para tener acceso al Admin Site de Django necesitamos crear un usuario administrador, en este paso se realiza el siguiente comando:

Listing 15: Comando para crear superusuario

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

- Dentro de la consola, designamos los datos que tendrá nuestro usuario, en mi caso decidí llenarlo de la siguiente forma:

Listing 16: Creando superusuario

```
$ python manage.py createsuperuser
Nombre de usuario: jhonatan
Direccion de correo electronico:
Password: jbenja123
Password (again): jbenja123
```

7.2. Acceso a Django Admin Site

- Tras este paso, se puede acceder al sitio administrativo de Django con el usuario y contraseña que se acaban de crear en <http://127.0.0.1:8000/admin/>:

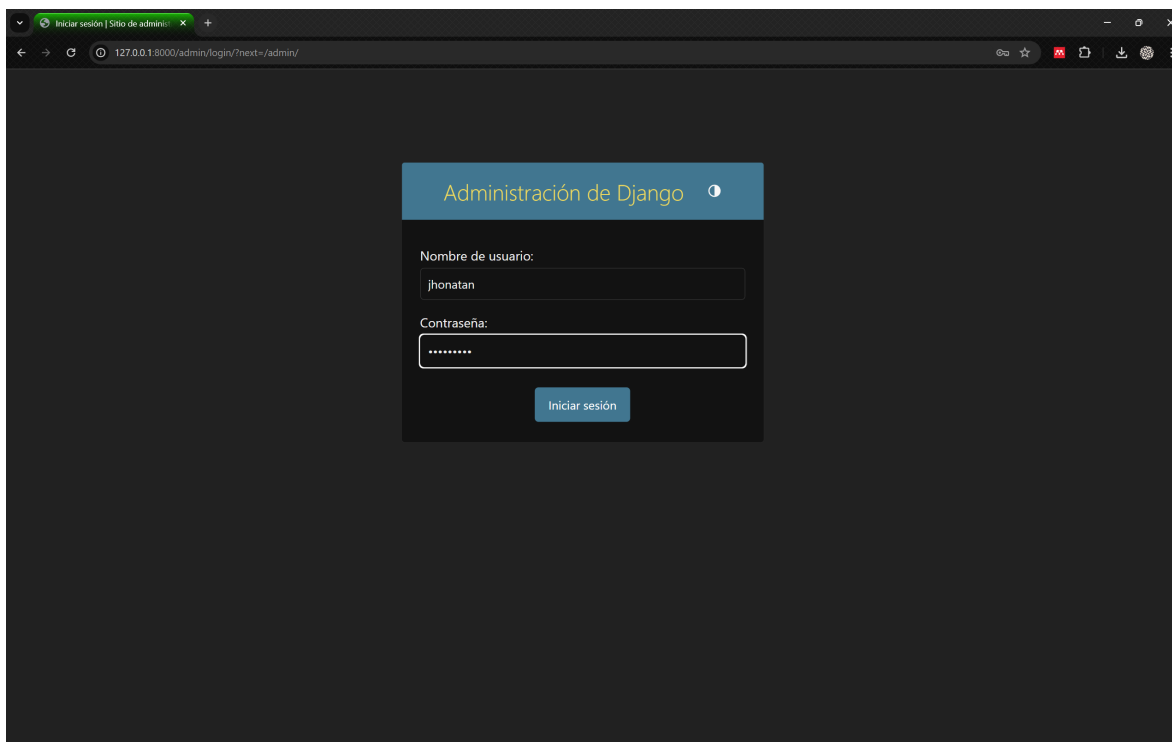


Figura 5: Inicio de sesión

- Finalmente, al acceder al sitio administrativo de Django, se muestra la información sobre los usuarios y grupos registrados hasta ahora:

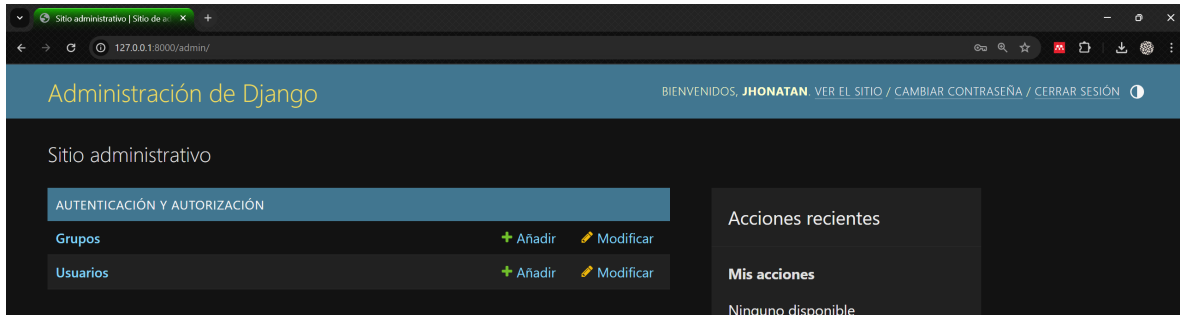


Figura 6: Django Admin Site

7.3. Registro de modelos

- El sitio administrativo es capaz de mostrar los modelos, sin embargo aquí no aparecen, en las diapositivas se sigue un paso simple para solucionarlo.
- Cuando se crean los modelos, se necesita registrarlos en el archivo admin.py de la aplicación en la que se crearon, es este caso, accedemos al de personas:

Listing 17: Ingresando a admin.py

```
$ vim personas/admin.py
```

- Aquí importamos el módulo de la clase que creamos antes en models.py de personas, luego lo registramos en el admin.site:

Listing 18: Registro del modelo Persona

```
from django.contrib import admin

# Register your models here.
from .models import Persona

admin.site.register(Persona)
```

- Así se registró el cambio del archivo en el commit realizado:

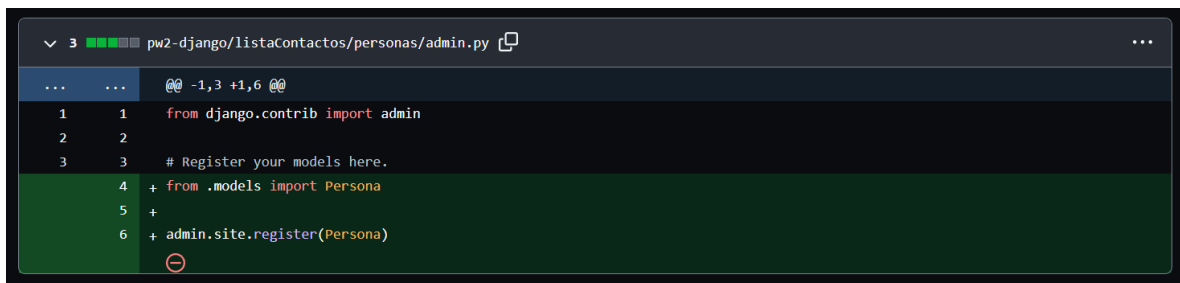


Figura 7: Commit de admin.py

- Entonces, tras esto, al reiniciar el servidor y volver a acceder al sitio administrativo de Django, encontraremos el acceso a la tabla generada desde el modelo Persona que creamos en el proyecto, curiosamente con el nombre en plural:

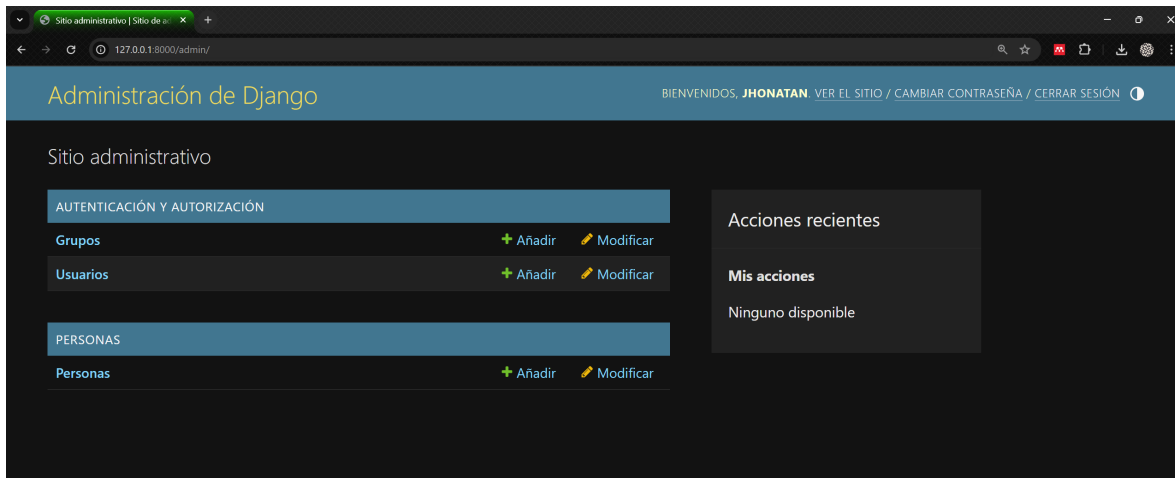


Figura 8: Django Admin Site (Con el modelo Persona)

- Desde este sitio es posible gestionar fácilmente los datos que se registran en el modelo, esto porque permite añadir, eliminar y modificar entradas en general.

8. Crear objetos en Python Shell

- Además de la facilidad de crear objetos desde el sitio administrativo de Django, es posible agregar objetos desde el Shell de Python, para acceder a este se ejecuta lo siguiente:

Listing 19: Ingresando a Python Shell

```
$ python manage.py shell
```

- Una vez dentro del shell, se procede a realizar una serie de pasos; Se importa el módulo de la clase Persona, se revisa la cantidad de objetos creados en este modelo, se crea uno nuevo con la información del profesor Alfredo Paz Valderrama y se hace un nuevo llamado para confirmar la existencia del nuevo objeto creado:

Listing 20: Creación de un nuevo objeto desde el shell

```
>>> from personas.models import Persona
>>> Persona.objects.all()
<QuerySet []>
>>> Persona.objects.create(nombres="Alfredo", apellidos="Paz Valderrama",
    edad="23")
<Persona: Persona object (1)>
>>> Persona.objects.all()
<QuerySet [<Persona: Persona object (1)>]>
>>> exit()
```

- Entonces, si se vuelve a iniciar el servidor, y nos dirigimos hacia el sitio de administración, podemos revisar su existencia también:

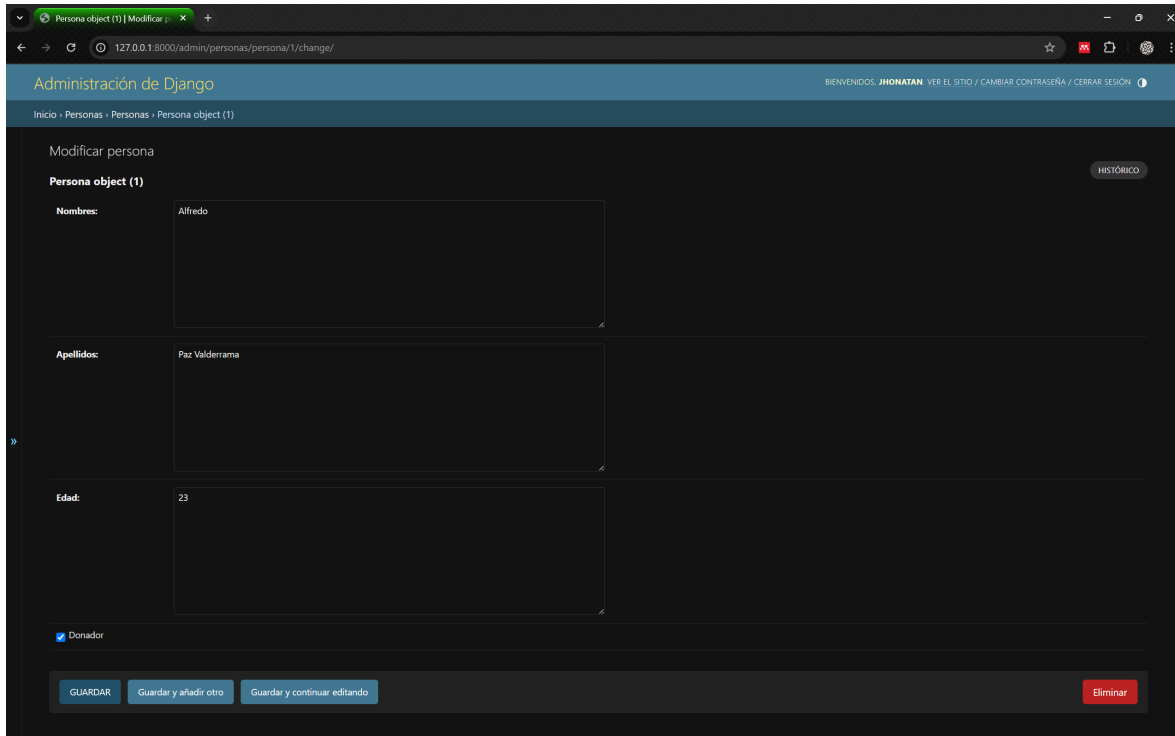


Figura 9: Django Admin Site (Persona 1)

9. Captura de pantalla de commits

- Como último paso de la actividad se ejecuta este comando para obtener el historial de los commits realizados en la actividad y le tomamos una captura:

Listing 21: Historial de commits

```
$ git log --graph --pretty=oneline --abbrev-commit --all
```

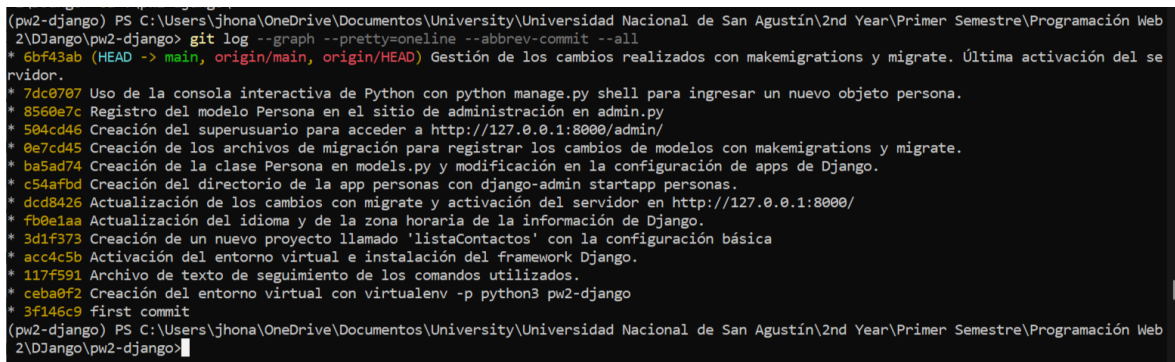


Figura 10: Captura del historial de commits

10. Estructura del trabajo

- El contenido que se entrega en esta práctica es el siguiente:

```
pw2-django/
|--- Informe01
|   |--- img
|   |   |--- Admin_login.png
|   |   |--- Admin_site.png
|   |   |--- Admin_site2.png
|   |   |--- Captura.png
|   |   |--- Commit1.png
|   |   |--- Commit2.png
|   |   |--- Commit3.png
|   |   |--- Commit4.png
|   |   |--- Django_site.png
|   |   |--- logo_abet.png
|   |   |--- logo_episunsa.png
|   |   |--- logo_unsa.jpg
|   |   |--- Personal.png
|   |--- Informe_Django01.pdf
|   |--- Informe_Django01.tex
|--- listaContactos
|   |--- listaContactos
|   |   |--- __init__.py
|   |   |--- asgi.py
|   |   |--- settings.py
|   |   |--- urls.py
|   |   |--- wsgi.py
|   |   |--- personas
|   |   |   |--- migrations
|   |   |   |   |--- 001_initial.py
|   |   |   |   |--- __init__.py
|   |   |   |   |--- __init__.py
|   |   |   |--- admin.py
|   |   |   |--- apps.py
|   |   |   |--- models.py
|   |   |   |--- tests.py
|   |   |   |--- views.py
|   |   |--- db.sqlite3
|   |   |--- manage.py
|   |--- Comandos.txt
|--- .gitignore
|--- README.md
```

11. Preguntas:

11.1. ¿Qué archivos se modificaron al hacer makemigrations y migrate?

- Se modifican los archivos dentro de la carpeta migrations.
- De manera más detallada, con el primero 'makemigrations', se detectan los cambios en los modelos de Django (definidos en models.py) y crea archivos de migración correspondientes en la carpeta migrations de cada aplicación. Estos archivos tienen nombres como 0001_initial.py, 0002_alter.py, etc., y contienen clases que heredan de django.db.migrations.Migration. Representan los pasos necesarios para aplicar (o deshacer) los cambios en la base de datos cada vez que modificamos los modelos.
- Luego, con 'migrate', se aplica o deshace las migraciones basadas en los archivos de migración. Actualiza la base de datos para que coincida con el estado actual de los modelos y mantiene un registro de las migraciones aplicadas en una tabla especial llamada django_migrations. Aunque no modifica los archivos del código fuente, es responsable de actualizar la estructura de la base de datos (por ejemplo, creando o modificando tablas).

11.2. ¿Qué archivos se modificaron al agregar personas?

- Al agregar nuevas entradas a la base de datos, como objetos del modelo Persona, no se modificaron archivos dentro del proyecto. En su lugar, aprendí que Django utiliza su Object-Relational Mapper (ORM) para generar y ejecutar consultas SQL que insertan los nuevos datos en la tabla correspondiente. Por ejemplo, con el panel de administración configurado, puedo agregar personas a través de una interfaz gráfica, y solo realiza inserciones en la base de datos sin modificar directamente ningún archivo mas que este.

12. Rúbricas

12.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

12.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	3	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		19	

13. Referencias

- <https://www.w3schools.com/django/>
- <https://docs.djangoproject.com/es/5.0/intro/tutorial01/>
- <https://github.com/mdn/django-locallibrary-tutorial/blob/main/.gitignore>
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/development_environment