

Django

Capítulo 1: Introducción a Django



Django es un framework de desarrollo web de código abierto en Python que sigue el patrón de diseño Modelo-Vista-Controlador (MVC). Fue diseñado para facilitar la creación rápida y el despliegue de aplicaciones web complejas y escalables.

Django hace énfasis en el principio de “baterías incluidas”, lo que significa que incluye una gran cantidad de funcionalidades listas para usar, como autenticación de usuarios, administración de bases de datos, formularios y más, lo cual acelera el desarrollo sin necesidad de reinventar la rueda en cada proyecto. Es ampliamente utilizado tanto por desarrolladores individuales como por grandes empresas debido a su robustez y la comunidad activa que lo respalda.

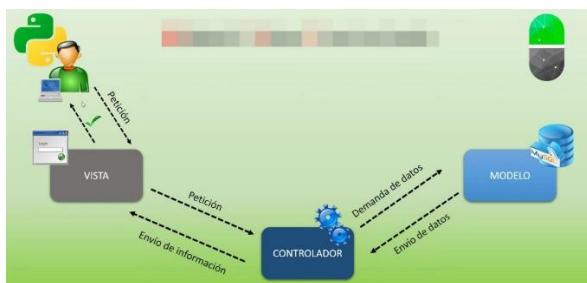
¿Qué es un framework?

Un **framework** es un marco de trabajo que nos ofrece un conjunto de herramientas y bibliotecas que proporcionan una estructura y una metodología para desarrollar software.

Estos están diseñados para ayudar a los desarrolladores a evitar la repetición de código común y a seguir buenas prácticas de programación.

¿Qué es el Modelo- Vista-Controlador (MVC)?

El **Modelo-Vista-Controlador** es un patrón de diseño arquitectónico utilizado comúnmente en el desarrollo de software, incluyendo aplicaciones web.



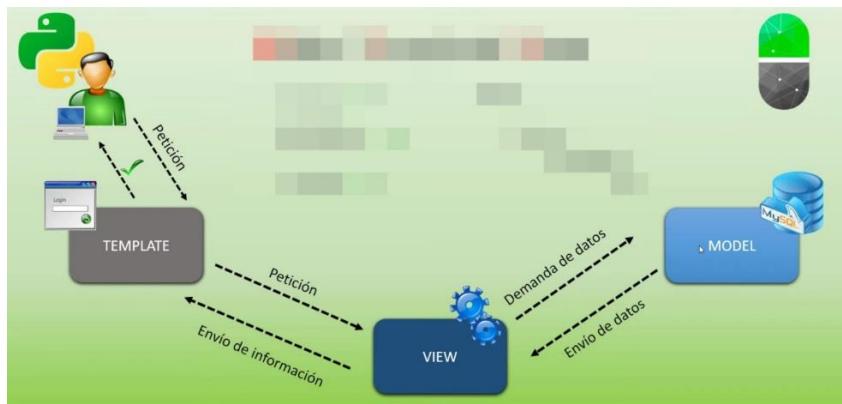
1-. Modelo: Representa los datos y la lógica de negocio de la aplicación. Es responsable de gestionar el acceso a los datos, su validación, y cualquier lógica relacionada con la manipulación de esos datos.

2-. Vista: Es la interfaz de usuario que presenta los datos al usuario y con la cual el usuario interactúa. La vista se encarga de la presentación visual de la información, como HTML en el contexto web.

3-. Controlador: Actúa como intermediario entre el modelo y la vista. Responde a los eventos generados por el usuario desde la interfaz de usuario y realiza las peticiones adecuadas al modelo para actualizar el estado de los datos. Además, actualiza la vista cuando los datos cambian.

Este enfoque también permite que diferentes equipos de desarrollo trabajen en paralelo en distintas capas de la aplicación sin interferir en las otras, lo que facilita la escalabilidad y el desarrollo colaborativo.

Nota: Django utiliza un patrón similar al MVC llamado **MTV**, que significa **Modelo-Template-Vista**. La principal diferencia radica en que el template en Django se centra más en la presentación de la información al usuario, mientras que la vista (o controlador, en el contexto de MVC) maneja la lógica de negocio y la interacción con el modelo.



Proceso de instalación local

1-. Ir a la página de Django: <https://www.djangoproject.com/download/>

2-. Introduce el comando adecuado para tu sistema operativo en la consola del sistema.

```
C:\Windows\system32>py -m pip install Django==5.0.6
Collecting Django==5.0.6
  Downloading Django-5.0.6-py3-none-any.whl.metadata (4.1 kB)
  Collecting asgiref<4,>=3.7.0 (from Django==5.0.6)
    Downloading asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
  Collecting sqlparse==0.3.1 (from Django==5.0.6)
    Downloading sqlparse-0.5.0-py3-none-any.whl.metadata (3.9 kB)
Requirement already satisfied: tzdata in c:/users/52556/appdata/local\programs\python\python312\lib\site-packages (from Django==5.0.6) (2023.3)
Downloaded Django-5.0.6-py3-none-any.whl (8.2 MB)
----- 8.2/8.2 MB 2.0 MB/s eta 0:00:00
Downloaded asgiref-3.8.1-py3-none-any.whl (23 kB)
Downloaded sqlparse-0.5.0-py3-none-any.whl (43 kB)
----- 44.0/44.0 kB 2.1 MB/s eta 0:00:00
Installing collected packages: sqlparse, asgiref, Django
Successfully installed Django-5.0.6 asgiref-3.8.1 sqlparse-0.5.0
[notice] A new release of pip is available: 24.0 -> 24.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
C:\Windows\system32>
```

3-. Para verificar si la instalación fue exitosa, utiliza los siguientes comandos:

```
C:\Windows\system32>python
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]
n win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION
(3, 0, 6, 'final', 0)
```

Django soporta varios tipos de bases de datos relacionales y no relacionales, permitiendo una flexibilidad considerable en la elección del motor de base de datos. Los principales sistemas de gestión de bases de datos (SGBD) compatibles con Django incluyen:

SQL Relacionales

- * PostgreSQL
- * MySQL
- * SQLite (predeterminada de Django)
- * Oracle
- * SQL Server

No SQL (No Relacionales)

- * MongoDB (a través de bibliotecas como Django MongoDB Engine)
- * Firebase (a través de bibliotecas como Django Firebase)

Django utiliza su ORM (Object-Relational Mapping) para proporcionar una capa de abstracción sobre estos motores de base de datos, lo que facilita la gestión y manipulación de los datos sin tener que escribir SQL directamente en la mayoría de los casos. Esta flexibilidad hace que Django sea adecuado para una amplia gama de aplicaciones y entornos de desarrollo.

Primer proyecto en local:

1-. Crear una carpeta y abrir la consola del sistema para pegar su dirección correspondiente y colocar el siguiente comando:

```
>cd C:\Users\52556\Desktop\Django
```

2-. A continuación, introduce el siguiente comando, añadiendo el nombre que deseas para tu proyecto:

```
>django-admin startproject Proyecto1
```

3-. Se generará una subcarpeta con todos los archivos necesarios para que el proyecto funcione:



4-. Algunas de las aplicaciones instaladas por defecto necesitan una base de datos. Por lo tanto, crearemos una utilizando los siguientes dos comandos:

```
>cd Proyecto1
```

```
>python manage.py migrate
```

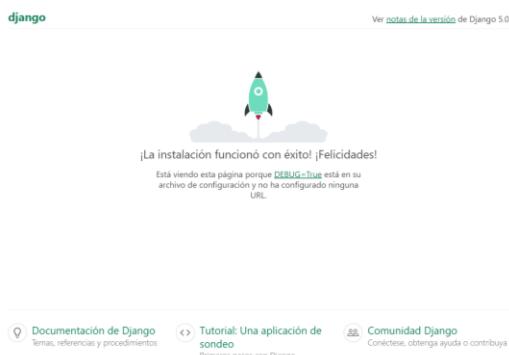
5-. Se generará una base de datos en SQLite para activar el proyecto:

Proyecto1	30/06/2024 07:19 p. m.	Carpeta de archivos
db.sqlite3	30/06/2024 07:19 p. m.	Archivo SQLITE3 128 KB
manage	30/06/2024 06:56 p. m.	Python File 1 KB

6-. Para comprobar si hemos construido correctamente nuestro proyecto, introduciremos el siguiente comando:

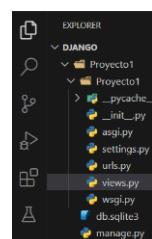
```
>python manage.py runserver
```

Después, abriremos un navegador web en donde introduciremos el link otorgado por el comando anterior; esto iniciará el servidor de desarrollo de Django para que puedas ver y probar tu aplicación web localmente.



Práctica 1: Primera página web

Paso 1: Agregar la carpeta del proyecto creado a Visual Studio Code y crear en la subcarpeta un archivo llamado views.py



Paso 2: En esta carpeta crearemos nuestras primeras “vistas”

```
views.py
from django.http import HttpResponse
def saludo(request): # A cada función perteneciente a este archivo se le llamará vista
    return HttpResponse("Primera página con Django")
def despedida(request):
    return HttpResponse("Hasta luego Django")
```

Paso 3: Luego, tendremos que crear una URL para cada una de estas vistas. Para hacerlo, iremos al archivo correspondiente.

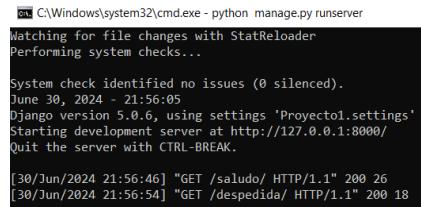
```
urls.py
from django.contrib import admin
from django.urls import path
from Proyecto1.Views import saludo
from Proyecto1.Views import despedida
urlpatterns = [ # Dentro de esta tupla estableceremos las URL
    path('admin/', admin.site.urls),
    path('saludo/', saludo),
    path('despedida/', despedida),
```

Nota: Cabe mencionar que, por cada función que utilicemos en este archivo, tendremos que importar su respectivo módulo.

Paso 4: A continuación, introduciremos los siguientes 2 comandos para “correr” el servidor.

```
>cd C:\Users\52556\Desktop\ Django\Proyecto1
```

```
> python manage.py runserver
```



```
C:\Windows\system32\cmd.exe - python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
June 30, 2024 - 21:56:05
Django version 5.0.6, using settings 'Proyecto1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[30/Jun/2024 21:56:46] "GET /saludo/ HTTP/1.1" 200 26
[30/Jun/2024 21:56:54] "GET /despedida/ HTTP/1.1" 200 18
```

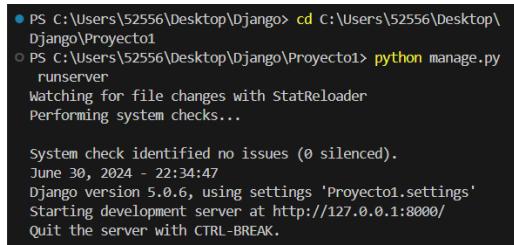
Paso 5: Por último introduciremos en cualquier servidor la siguiente URL
localhost:8000/nombre_de_la_función/

Primera página con Django

Hasta luego Django

Práctica 2: Parámetros en URL

Paso 1: Como primer paso, ejecutamos el servidor, pero esta vez a través de la consola de Visual Studio Code.



```
PS C:\Users\52556\Desktop\ Django> cd C:\Users\52556\Desktop\ Django\Proyecto1
PS C:\Users\52556\Desktop\ Django\Proyecto1> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
June 30, 2024 - 22:34:47
Django version 5.0.6, using settings 'Proyecto1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Paso 2: Crearemos dos funciones como ejemplo, una para introducir parámetros en una URL y otra para que la página tenga un forma dinámica al momento de darnos la hora.



```
def fecha(request):
    fecha_actual = datetime.now()
    documento = "<html><body><h1> Fecha y hora actuales <s> </h1></body></html>" %fecha_actual
    return HttpResponse(documento)

def edad(request, edad, agno):
    # edad_actual = 22
    periodo = agno - 2024
    edad_futura = edad + periodo
    documento = "<html><body><h1> En el año <s> tendrás <s> años</h1></body></html>" %(agno, edad_futura)
    return HttpResponse(documento)
```

Paso 3: Crearemos una URL para cada una de estas vistas.

```

from django.contrib import admin
from django.urls import path
from Proyecto1.views import saludar
from Proyecto1.views import despedida
from Proyecto1.views import fecha
from Proyecto1.views import edad

urlpatterns = [ # Dentro de esta tupla estableceremos las URL
    path('admin/', admin.site.urls),
    path('saludar/', saludar),
    path('despedida/', despedida),
    path('fecha/', fecha),
    path('edad/<int:edad>/<int:agno>', edad),
]

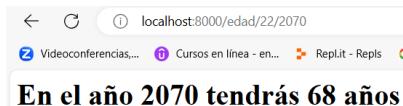
```

Paso 4: Por último introduciremos en cualquier servidor la siguiente URL
localhost:8000/nombre_de_la_función/

Nos actualizará la hora cada vez que recarguemos la página.

Fecha y hora actuales 2024-07-01 01:05:18.883056

En este caso, introduciremos manualmente los parámetros correspondientes para que nos calcule la edad.



Capítulo 2: Plantillas

En el contexto del desarrollo web, las **plantillas** son archivos que definen la estructura y el diseño de las páginas web. En Django, las plantillas se utilizan para generar dinámicamente contenido HTML.

Aquí hay algunas características clave de las plantillas:

1-. Separación de contenido y presentación: Permiten separar el contenido de la lógica de presentación, facilitando el mantenimiento y la reutilización del código.

2-. Etiquetas y Variables: Utilizan un sistema de etiquetas y variables para insertar dinámicamente datos en el HTML. Por ejemplo, `{% if user.is_authenticated %}` o `{{ user.username }}`.

3-. Herencia de Plantillas: Permiten la herencia, lo que significa que puedes crear una plantilla base con elementos comunes (como encabezado y pie de página) y extender esta plantilla en otras, reutilizando así el código.

4-. Filtros y Funciones: Pueden usar filtros y funciones para transformar y mostrar datos de manera específica. Por ejemplo, `{{ date | date:"D d M Y" }}`.

Práctica 1: Plantillas

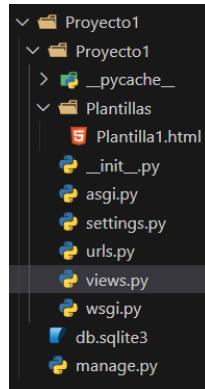
Paso 1: Ejecutamos el servidor, y crearemos un nuevo archivo en Visual Studio Code en donde copiaremos y pegaremos el código HTML de nuestra primera función.

```

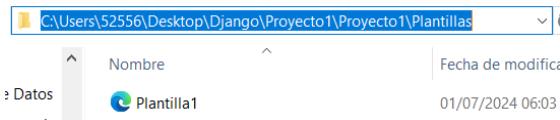
views.py          <html> Untitled-1  urls.py
1   <html>
2     <body>
3       <h1> Primera página con Django </h1>
4     </body>
5   </html>

```

Paso 2: Guardaremos el archivo HTML en una carpeta llamada “Plantillas”.



Paso 3: Copiamos la ruta donde tengamos el archivo HTML y la asignamos a una variable.



Paso 4: Después, modificaremos esa ruta convirtiendo las barras invertidas en barras diagonales y agregaremos al final de la misma el nombre del archivo HTML con su extensión.

```
doc_externo = open("C:/Users/52556/Desktop/Django/Proyecto1/Proyecto1/Plantillas/Plantilla1.html")
```

Paso 5: A continuación, crearemos las partes faltantes para que la plantilla funcione correctamente.

```

from django.template import Template
from django.template import Context

def saludar(request): # A cada función perteneciente a este archivo se le llamará vista

    doc_externo = open("C:/Users/52556/Desktop/Django/Proyecto1/Proyecto1/Plantillas/Plantilla1.html")

    plantilla = Template(doc_externo.read())

    doc_externo.close()

    contexto = Context()

    documento = plantilla.render(contexto)

    return HttpResponseRedirect(documento)

```

- Lee el contenido del archivo abierto (doc_externo.read()) y lo pasa al constructor Template de Django para crear un objeto Template. Este objeto representa la plantilla cargada desde el archivo.

- Crea un objeto Context. El contexto es un diccionario que contiene las variables y sus valores que se utilizarán en la plantilla. En este caso, el contexto está vacío.
- Renderiza la plantilla (plantilla) utilizando el contexto (contexto). El método render reemplaza las variables en la plantilla con sus valores del contexto y devuelve el resultado como una cadena de texto HTML (documento).
- Crea una respuesta HTTP con el contenido del documento renderizado (documento) y la devuelve al navegador del usuario. HttpResponseRedirect es una clase de Django que se utiliza para enviar datos al cliente.

Paso 6: Cargaremos la página para ver el resultado.



Práctica 2: Plantillas

Paso 1: Crearemos tres nuevas variables en la misma vista que antes.

```
def saludo(request): # A cada función perteneciente a este archivo se le llamará vista
    nombre = "Sebastián"
    apellido = "Solis"
    fecha_actual = datetime.now()
```

Paso 2: Agregaremos esas variables al contexto.

```
contexto = Context({"nombre_persona":nombre, "apellido_persona":apellido, "ahora":fecha_actual})
```

Paso 3: Vincularemos ese contexto a la plantilla para que pueda ser visualizado.

```
Proyecto1 > Proyecto1 > Plantillas > Plantilla1.html > ...
1 <html>
2   <body>
3     <h1 style = "color: red">
4       Primera pagina con Django
5     </h1>
6     <p> El nombre del profesor es: {{nombre_persona}} {{apellido_persona}}</p>
7     <p> Este tema se esta viendo el dí;a: {{ahora.day}} / {{ahora.month}} / {{ahora.year}}</p>
8   </body>
9 </html>
```

Paso 4: Ejecutamos el servidor para ver los resultados.

Primera pagina con Django

El nombre del profesor es: Sebastián Solis

Este tema se esta viendo el dia: 1 / 7 / 2024

Paso 5: Otra forma de ver este tema es mediante la POO, por lo que agregaremos una clase con su respectivo constructor y atributos.

```
class Persona(object):
    def __init__(self, nombre, apellido):
        self.nombre = nombre
        self.apellido = apellido
```

Paso 6: Crearemos una instancia en la vista “saludo”.

```
def saludo(request): # A cada función perteneciente a este archivo se le llamará vista
    persona1 = Persona("Miguel", "Galindo")
```

Nota: Convertiremos en comentarios las variables nombres y apellido.

Paso 7: Modificaremos el contexto de la vista y sustituiremos los valores del diccionario por las propiedades de la clase definida.

```
contexto = Context({"nombre_persona":persona1.nombre, "apellido_persona":persona1.apellido, "ahora":fecha_actual})
```

Paso 8: Cargaremos la página para ver el resultado.

Primera pagina con Django

El nombre del profesor es: Miguel Galindo
Este tema se está viendo el día: 1 / 7 / 2024

Práctica 3: Plantillas

Paso 1: Agregaremos una lista al contexto de la vista “saludo” y la relacionaremos en la plantilla HTML.

```
"ahora":fecha_actual, "temas":["Plantillas", "Modelos", "Formularios", "Vistas", "Despliege"])
```

```
<html>
    <body>
        <h1 style = "color: red">
            Primera pagina con Django
        </h1>
        <p> El nombre del profesor es: {{nombre_persona}} {{apellido_persona}}</p>
        <p> Este tema se está viendo el día: {{ahora.day}} / {{ahora.month}} / {{ahora.year}}</p>
        <p> Temas del curso: {{temas}}</p>
    </body>
</html>
```

Paso 2: Cargaremos la página para ver el resultado.

Primera pagina con Django

El nombre del profesor es: Miguel Galindo
Este tema se está viendo el día: 1 / 7 / 2024
Temas del curso: ['Plantillas', 'Modelos', 'Formularios', 'Vistas', 'Despliege']

Nota: Si queremos que solo aparezca un elemento de la lista, debemos poner un punto seguido del número de su índice en el nombre del espacio correspondiente a esta.

Nota: Otra forma sería agregando toda la lista a una variable y luego poniendo en el contexto el nombre de esa variable.

Paso 3: Otra forma de presentar esta lista es mediante un bucle for por medio de la plantilla.

```
<html>
  <body>
    <h1 style = "color: red">
      Primera pagina con Django
    </h1>
    <p> El nombre del profesor es: {{nombre_persona}} {{apellido_persona}}</p>
    <p> Este tema se esta viendo el dia: {{ahora.day}} / {{ahora.month}} / {{ahora.year}}</p>
    <p> <strong> Temas del curso: {{temas.1}} </strong></p>
    <p>
      <ul>
        {% for variable in temas %}
          <li>{{variable}}</li>
        {% endfor %}
      </ul>
    </p>
  </body>
</html>
```

Paso 4: Cargaremos la página para ver el resultado.

Primera pagina con Django

El nombre del profesor es: Miguel Galindo

Este tema se esta viendo el dia: 1 / 7 / 2024

Temas del curso: Modelos

- Plantillas
- Modelos
- Formularios
- Vistas
- Despliege

Nota: Para mejorar o personalizar nuestra lista, podemos agregarle un bucle o un condicional.

```
<html>
  <body>
    <h1 style = "color: red">
      Primera pagina con Django
    </h1>
    <p> El nombre del profesor es: {{nombre_persona}} {{apellido_persona}}</p>
    <p> Este tema se esta viendo el dia: {{ahora.day}} / {{ahora.month}} / {{ahora.year}}</p>
    <p> <strong> Temas del curso: {{temas.1}} </strong></p>
    <p>
      <ul>
        {% if temas %}
          {% for variable in temas %}
            <li>{{variable}}</li>
          {% endfor %}
        {% else %}
          <p> No hay elementos que mostrar </p>
        {% endif %}
      </ul>
    </p>
  </body>
</html>
```

En este caso, si la lista existe y tiene información, la plantilla la mostraría; de lo contrario, tendría que mostrar la frase “No hay elementos que mostrar”.

Nota: Si queremos agregarle métodos a nuestros valores, debemos seguir la misma notación que se usa con un atributo o índice.

```
<p> El nombre del profesor es: {{nombre_persona.upper}} {{apellido_persona.upper}}</p>
```

¿Qué son los filtros?

Los **filtros** son herramientas que permiten modificar el valor de una variable antes de que se muestre en el HTML. Se usan para realizar transformaciones de datos directamente en la plantilla, lo que permite mantener la lógica de presentación separada de la lógica de negocio.

Para aplicar un filtro a una variable en una plantilla, usa la siguiente sintaxis:

```
{ { variable | filtro } }
```

Puedes encadenar varios filtros uno tras otro:

```
{ { variable | filtro1 | filtro2 } }
```

Ejemplos de filtros comunes:

1-. **date**: Formatea una fecha de acuerdo a un formato especificado.

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

2-. **length**: Devuelve la longitud de una secuencia.

```
{ { lista | length } }
```

3-. **default**: Devuelve un valor por defecto si la variable es vacía o no está definida.

```
{ { variable | default:"Valor por defecto" } }
```

4-. **upper**: Convierte una cadena a mayúsculas.

```
{ { nombre | upper } }
```

5-. **lower**: Convierte una cadena a minúsculas.

```
{ { nombre | lower } }
```

6-. **title**: Convierte la cadena a formato título (primera letra de cada palabra en mayúscula).

```
{ { nombre | title } }
```

7-. **truncatechars**: Trunca una cadena a un número específico de caracteres.

```
{ { texto | truncatechars:10 } }
```

Ejemplo de encadenamiento con varios filtros:

```
<ul>
  {% if temas %}
    {% for variable in temas %}
      <li>{{variable|first|lower}}</li>
    {% endfor %}
  {% else %}
    <p> No hay elementos que mostrar </p>
  {% endif %}
</ul>
</p>
</body>
</html> |
```

Temas del curso: Modelos

- p
- m
- f
- v
- d

¿Qué son los cargadores?

Los **cargadores** de plantillas son componentes responsables de encontrar y cargar plantillas desde diversas fuentes. Los cargadores de plantillas buscan los archivos de plantillas en ubicaciones específicas y los devuelven como objetos de plantilla que pueden ser renderizados

con datos del contexto. Estos cargadores permiten flexibilidad en cómo y dónde se almacenan las plantillas, facilitando la organización y reutilización de plantillas en una aplicación Django.

Práctica 4: Plantillas y cargadores

Paso 1: Importaremos la librería correspondiente para usar los cargadores.

```
from django.template import loader
```

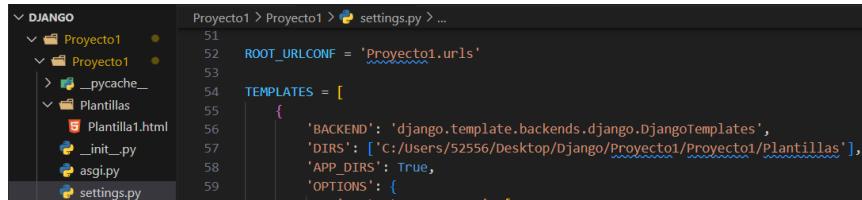
Paso 2: Comentaremos las líneas correspondientes a la liga que introducimos anteriormente.

```
# doc_externo = open("C:/Users/52556/Desktop/Django/Proyecto1/Proyecto1/Plantillas/Plantilla1.html")

# plantilla = Template(doc_externo.read())

# doc_externo.close()
```

Paso 3: Ahora debemos especificar en nuestro proyecto cuál es la carpeta donde se encuentran nuestras plantillas, y esto se debe hacer en el archivo “settings.py”.



Paso 4: Cargaremos esa plantilla externa en el archivo “views.py”.

```
doc_externo = loader.get_template('Plantilla1.html')
```

Nota: Si quisiéramos cargar cinco o más plantillas, lo único que tendríamos que hacer es utilizar esta instrucción.

Paso 5: Comentaremos la línea del contexto debido a que, en este caso, el método “template” es diferente al anterior, ya que no admite un contexto como parámetro. En su lugar, acepta un diccionario directamente. En el caso anterior, teníamos que crear un contexto y pasarle un diccionario a ese contexto.

```
# contexto = Context({"nombre_persona":persona1.nombre, "apellido_persona":persona1.apellido, "ahora":fecha_actual, "temas":["Plantillas", "Modelos"]})
documento = doc_externo.render({"nombre_persona":persona1.nombre, "apellido_persona":persona1.apellido, "ahora":fecha_actual, "temas":["Plantillas", "Modelos"]})
```

Nota: Una forma de simplificar más la instrucción load es mediante la siguiente importación, ya que con esto puedes ahorrarte escribir la palabra “load” en el código.

```
django.template.loader import get_template
```

Práctica 5: Plantillas

Paso 1: Importaremos la librería correspondiente para simplificar más el código en la sección “views”.

```
from django.shortcuts import render
```

Paso 2: Comentaremos las líneas de las variables “doc_externo” y “documento” y modificamos la linea de la variable “return”.

```
# doc_externo = loader.get_template('Plantilla1.html')

# documento = doc_externo.render({"nombre_persona":persona1.nombre, "apellido_persona":persona1.apellido, "hora":fecha_actual}

return render(request, "Plantilla1.html", {"nombre_persona":persona1.nombre, "apellido_persona":persona1.apellido, "hora":fecha_actual,})
```

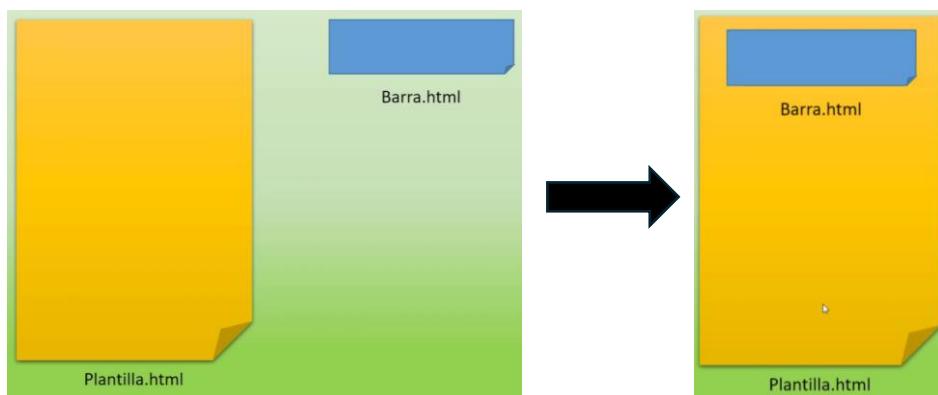
Nota: De esta forma simplificaremos aún más el código anterior dejándolo solo con 4 líneas.

¿Qué son las plantillas incrustadas?

Las **plantillas incrustadas**, también conocidas como plantillas anidadas o subplantillas, son plantillas que se incluyen dentro de otras plantillas para promover la reutilización y organización del código en aplicaciones web. En Django, esto se realiza mediante la etiqueta `{% include %}`, que permite insertar una plantilla dentro de otra.

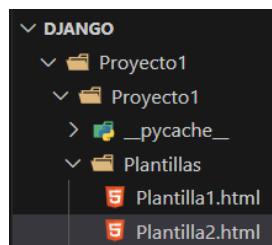
Beneficios de las plantillas incrustadas

- 1-. Reutilización de código:** Puedes crear componentes reutilizables, como encabezados, pies de página o barras de navegación, y usarlos en varias páginas.
- 2-. Mantenimiento más fácil:** Al separar las secciones comunes en plantillas independientes, se facilita la actualización y el mantenimiento del código.
- 3-. Organización del código:** Mejora la legibilidad y estructura del código, haciendo que sea más fácil de entender y gestionar.



Práctica 6: Plantillas incrustadas

Paso 1: Crearemos una segunda plantilla en su carpeta correspondiente.



Paso 2: Crearemos una barra de búsqueda en la nueva plantilla.



```
<html>
  <body>
    <ul>
      <li>Home</li>
      <li>Servicios</li>
      <li>Quiénes somos</li>
      <li>Contacto</li>
      <li>Acerca de</li>
    </ul>
  </body>
</html>
```

Paso 3: Incrustaremos esta segunda plantilla en la primera (en el lugar que quieras que aparezca) mediante la siguiente etiqueta.

```
{% include "Plantilla2.html" %}
```

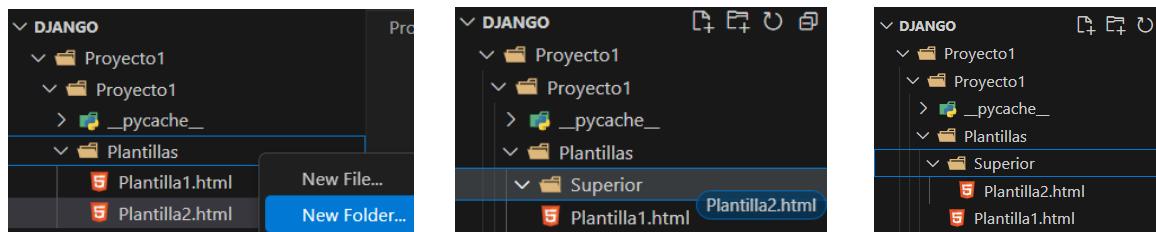
Paso 4: Le daremos unos toques personales a la segunda plantilla.



```
<html>
  <style>
    #Barra{
      margin: 0;
      padding: 0;
      list-style: none;
      text-transform: uppercase;
    }

    #Barra li{
      display: inline;
      margin: 0 20px;
    }
  </style>
  <body>
    <ul id = "Barra">
      <li>Home</li>
      <li>Servicios</li>
      <li>Quiénes somos</li>
      <li>Contacto</li>
      <li>Acerca de</li>
    </ul>
  </body>
</html>
```

Paso 5: En el caso de que tengamos demasiadas plantillas, deberíamos crear subdirectorios dentro de la carpeta general “plantillas” para mantenerlas organizadas.



Nota: Para evitar errores, especificaremos el subdirectorio en la etiqueta `{% include %}`.

```
{% include "Superior/Plantilla2.html" %}
```

Paso 6: Cargaremos la página para ver el resultado.

Primera pagina con Django

El curso lo imparte el profesor titular

El nombre del profesor es: MIGUEL GALINDO

Este tema se esta viendo el dia: 5 / 7 / 2024

Temas del curso: Modelos

- p
- m
- f
- v
- d

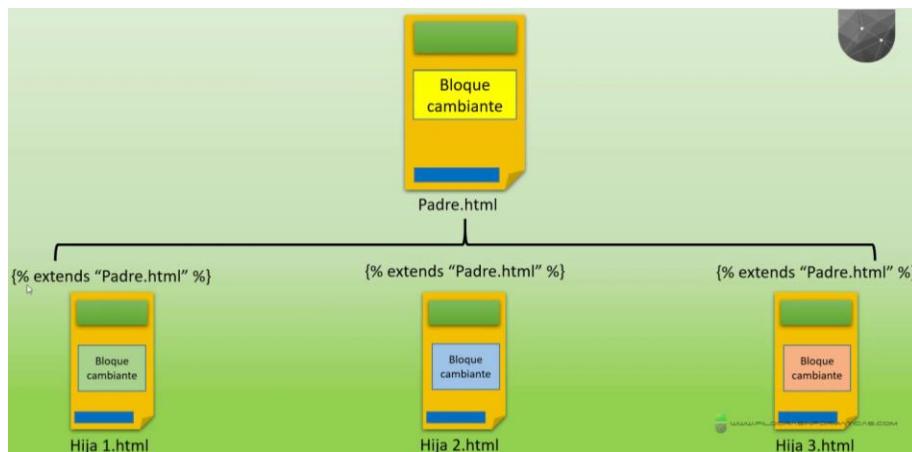
Herencia de plantillas

La **herencia de plantillas** en Django es una característica poderosa que permite crear una estructura base para el diseño de tus páginas web y reutilizar esa estructura en múltiples plantillas. Esto ayuda a mantener la coherencia del diseño y facilita el mantenimiento del código.

Este enfoque se basa en dos conceptos principales: la plantilla base y las plantillas que heredan de ella.

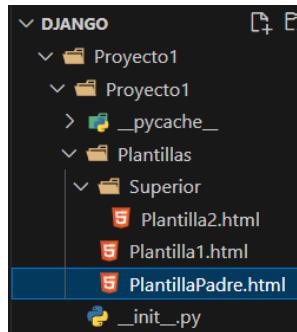
1-. Plantilla base (o padre): Define la estructura general de tu sitio web, como el encabezado, el pie de página y las secciones de contenido principal.

2-. Plantillas secundarias (o hijas): Heredan de la plantilla base y pueden sobrescribir o añadir contenido en secciones específicas definidas por bloques en la plantilla base.



Práctica 7: Herencia de plantillas

Paso 1: Crearemos la plantilla principal y su estructura en la carpeta "plantillas" para especificar qué elementos queremos mantener constantes y cuáles queremos cambiar.



```
PlantillaPadre.html ×
Proyecto1 > Proyecto1 > Plantillas > PlantillaPadre.html > html
1  <html>
2   <head>
3     <title>
4       | {& block title %} Plantilla padre {& endblock %}
5     </title>
6   </head>
7   <body>
8     <h1 style = "background-color: #darkolivegreen; color: white; text-align:center">Píldoras Informáticas</h1>
9     <h2>Cursos de programación, ofimática y diseño web</h2>
10    {& block content %} {& endblock %}
11   <p style = "background-color: #darkolivegreen; color: white; text-align:center">Cursos gratuitos. Gracias por su visita</p>
12 
13 </body>
</html>
```

Paso 2: Ahora crearemos una plantilla hija en la carpeta "plantillas" para heredar la estructura de la plantilla principal y especificar qué elementos cambiarán (en este caso, el título y el contenido).

```
PlantillaPadre.html      PlantillaHija1.html ×
Proyecto1 > Proyecto1 > Plantillas > PlantillaHija1.html > ...
1  {% extends "PlantillaPadre.html" %}
2
3  {% block title %} Plantilla hija 1 {& endblock %}
4
5  {% block content %}
6
7  <p>Estamos a día: {{dameFecha}}</p>
8
9  <iframe width="560" height="315" src="https://www.youtube.com/embed/d1-UX6aR...
10
11  {% endblock %}
```

Paso 3: El siguiente paso será registrar una vista para poder ver la plantilla hija y, a continuación, registrar la URL que queremos utilizar para acceder a esta página.

Views:

```
def plantilla_hija_uno(request):
    fecha_actual = datetime.now()
    return render(request, "PlantillaHija1.html", {"dameFecha":fecha_actual})
```

Urls:

```
from Proyecto1.views import plantilla_hija_uno
path('plantillahija/', plantilla_hija_uno),
```

Paso 4: Cargaremos la página para ver el resultado.



Paso 5: Si creáramos algunas plantillas adicionales y quisieramos agregar un nuevo elemento a cada una de ellas, solo tendríamos que modificar la plantilla principal (padre).

Agregamos la segunda línea de código (que ya fue creada anteriormente) en la parte que deseemos.

```
<h1 style = "background-color: #darkolivegreen; color: white; text-align:center">Píldoras Informáticas</h1>
{% include "Superior/Plantilla2.html" %}
```

Cargaremos la página para ver el resultado.



Capítulo 3: Bases de datos

Django facilita el trabajo con bases de datos gracias a su sistema ORM (Object-Relational Mapping). El ORM de Django permite interactuar con bases de datos usando código Python en lugar de SQL, simplificando la creación, lectura, actualización y eliminación de datos.

Práctica 1: Como crear una aplicación con Django



1-. Nos iremos a la consola de Visual Studio e introduciremos el siguiente comando, añadiendo el nombre que deseas ponerle a tu proyecto:

```
>django-admin startproject TiendaOnline
```

2-. Se generará una subcarpeta con todos los archivos necesarios para que el proyecto funcione:

Proyecto1	30/06/2024 07:19 p. m.	Carpeta de archivos
TiendaOnline	08/07/2024 03:37 p. m.	Carpeta de archivos

3-. Luego, introduciremos el primer comando para entrar en la carpeta del proyecto y poder utilizar el archivo “manage.py” con el segundo comando (añadiéndole el nombre que deseas ponerle a tu aplicación), de esta manera iniciaremos nuestra aplicación.

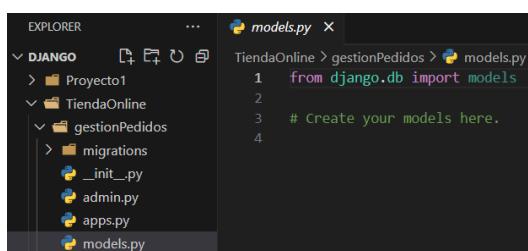
```
>cd TiendaOnline
```

```
>python manage.py startapp gestionPedidos
```

4-. Se generará una carpeta correspondiente para nuestra primera aplicación

gestionPedidos	08/07/2024 03:47 p. m.	Carpeta de archivos
TiendaOnline	08/07/2024 03:44 p. m.	Carpeta de archivos
manage	08/07/2024 03:37 p. m.	Python File 1 KB

5-. Después, abriremos la carpeta principal del proyecto en Visual Studio y accederemos al archivo “models” dentro de la subcarpeta “gestionPedidos”.



Nota: En este archivo se trabajará con la bases de datos.

6-. A continuación, construiremos nuestra base de datos especificando los campos y los tipos de datos correspondientes para cada tabla.

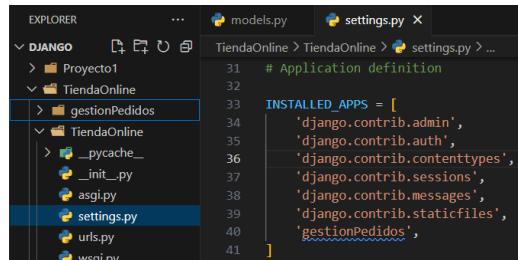
```

models.py
1  from django.db import models
2
3  # Create your models here.
4
5  class Clientes(models.Model):
6      Nombre = models.CharField(max_length=30)
7      Direccion = models.CharField(max_length=50)
8      Email = models.EmailField()
9      Telefono = models.CharField(max_length=7)

11 class Articulos(models.Model):
12     Nombre = models.CharField(max_length=30)
13     Seccion = models.CharField(max_length=20)
14     Precio = models.IntegerField()
15
16 class Pedidos(models.Model):
17     Numero = models.IntegerField()
18     Fecha = models.DateField()
19     Estado = models.BooleanField()

```

Nota: Debemos informar a nuestro proyecto de Django que tenemos una nueva aplicación llamada “gestiónPedidos”. Esto se hace mediante la configuración en el archivo settings.py (de la subcarpeta TiendaOnline).



Esto se hará por cada aplicación que introduzcamos al proyecto.

Nota: Para comprobar si todo va bien con nuestra aplicación, pondremos el siguiente comando.

>python manage.py check gestionPedidos

```

● PS C:\Users\52556\Desktop\Django\TiendaOnline> python manage.py startapp gestionPedidos
● PS C:\Users\52556\Desktop\Django\TiendaOnline> python manage.py check gestionPedidos
System check identified no issues (0 silenced).
○ PS C:\Users\52556\Desktop\Django\TiendaOnline>

```

7- Crearemos nuestra base de datos (con 3 tablas) mediante el siguiente comando.

>python manage.py makemigrations

```

● PS C:\Users\52556\Desktop\Django\TiendaOnline> python manage.py makemigrations
Migrations for 'gestionPedidos':
  gestionPedidos\migrations\0001_initial.py
    - Create model Articulos
    - Create model Clientes
    - Create model Pedidos
○ PS C:\Users\52556\Desktop\Django\TiendaOnline>

```

8- Se generará una base de datos vacía en la carpeta del proyecto.

gestionPedidos	08/07/2024 04:42 p. m.	Carpeta de archivos
TiendaOnline	08/07/2024 03:44 p. m.	Carpeta de archivos
db.sqlite3	08/07/2024 04:45 p. m.	Archivo SQLITE3 0 KB
manage	08/07/2024 03:37 p. m.	Python File 1 KB

9- Tendremos que indicarle a Django que dentro de esa base de datos vacía debe insertar las tablas que ya hemos creado. Para ello, generaremos el código SQL (de las 3 tablas) utilizando el siguiente comando.

>python manage.py sqlmigrate gestionPedidos 0001

```

● PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py sqlmigrate gestionPedidos 0001
BEGIN;
-- 
-- Create model Articulos
-- 
CREATE TABLE "gestionPedidos_articulos" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "Nombre" varchar(30) NOT NULL, "Seccion" varchar(20) NOT NULL, "Precio" integer NOT NULL);
-- 
-- Create model Clientes
-- 
CREATE TABLE "gestionPedidos_clientes" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "Nombre" varchar(30) NOT NULL, "Direccion" varchar(50) NOT NULL, "Email" varchar(254) NOT NULL, "Telefono" varchar(7) NOT NULL);
-- 
-- Create model Pedidos
-- 
CREATE TABLE "gestionPedidos_pedidos" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "Numero" integer NOT NULL, "Fecha" date NOT NULL, "Estado" bool NOT NULL);
COMMIT;
○ PS C:\Users\52556\Desktop\ Django\TiendaOnline>

```

Nota: El valor 0001 es el número de migración que nos permite rastrear las diferentes modificaciones realizadas en la base de datos.

Ya después utilizaremos ese código SQL para meter esas tablas en nuestra base de datos vacía mediante el siguiente comando.

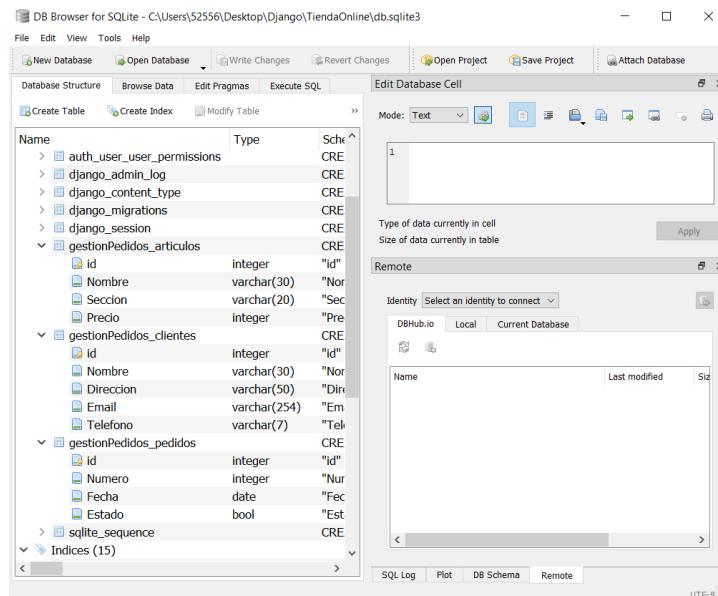
>`python manage.py sqlmigrate`

```

● PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestionPedidos, 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 auth.0012_alter_user_first_name_max_length... OK
  Applying gestionPedidos.0001_initial... OK
  Applying sessions.0001_initial... OK
○ PS C:\Users\52556\Desktop\ Django\TiendaOnline>

```

10-. Meteremos el archivo SQLite3 en el visualizador Db Browser.



Nota: Podremos ver que se han generado más tablas por defecto, lo cual es necesario para que Django funcione correctamente.

Nota: Otro aspecto a destacar es que Django agrega un campo ID por defecto a todas las tablas.

Práctica 2: Insertar, actualizar, eliminar y seleccionar registros desde consola

1-. Nos iremos a la consola de Visual Studio mediante el siguiente comando.

```
>python manage.py shell
```

```
PS C:\Users\52556\Desktop\Django\TiendaOnline> python manage.py shell
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
• IPython 8.17.2 -- An enhanced Interactive Python. Type '?' for help.

○ In [1]:
```

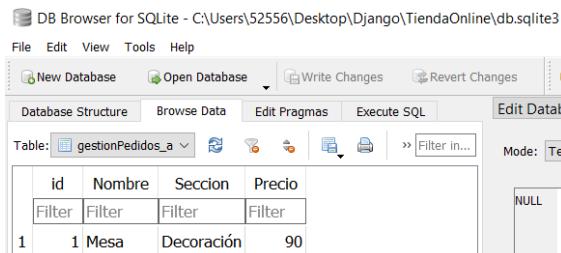
2-. Importamos el modelo con el que vamos a trabajar (en este caso, realizaremos las consultas con el modelo o la clase “Artículos”).

```
In [1]: from gestionPedidos.models import Articulos
```

3-. Introduciremos nuestro primer registro a la tabla de la siguiente manera.

```
In [2]: art=Articulos(Nombre='Mesa', Seccion='Decoración', Precio=90)
In [3]: art.save()
```

4-. En DB Browser podremos ver nuestro registro.



Nota: Otra forma de insertar un registro, sería de la siguiente forma.

```
In [7]: art3=Articulos.objects.create(Nombre='Taladro', Seccion='Ferretería', Precio=65)
```

5-. Actualizaremos nuestro primer registro a la tabla de la siguiente manera.

```
In [10]: art.Precio=95
In [11]: art.save()
```

6-. En DB Browser podremos ver el cambio que le realizamos a nuestro primer registro.

	id	Nombre	Sección	Precio
	Filter	Filter	Filter	Filter
1	1	Mesa	Decoración	95
2	2	Camisa	Confección	75
3	3	Taladro	Ferretería	65

7-. Borraremos nuestro primer registro.

```
In [12]: art4=Articulos.objects.get(id=3)
In [13]: art4.delete()
Out[13]: (1, {'gestionPedidos.Articulos': 1})
```

8-. En DB Browser podremos ver el cambio que le realizamos a nuestro tercer registro.

	id	Nombre	Sección	Precio
	Filter	Filter	Filter	Filter
1	1	Mesa	Decoración	95
2	2	Camisa	Confección	75

9-. Por último, crearemos una consulta en la consola.

```
In [14]: Consulta=Articulos.objects.all()
In [15]: consulta
Out[15]: <QuerySet [Articulos: Articulos object (1)], <Articulos: Articulos object (2)>>
In [16]: Consulta.query._str_()
Out[16]: 'SELECT "gestionPedidos_articulos"."id", "gestionPedidos_articulos"."Nombre", "gestionPedidos_articulos"."Sección", "gestionPedidos_articulos"."Precio" FROM "gestionPedidos_articulos"
```

Nota: En el último output, podemos ver la instrucción de tipo SELECT que ha generado Django para obtener el resultado anterior (es decir, el del output 15).

Práctica 3: Configurar Django con PostgreSQL en local

1-. Instalaremos una interfaz que nos permita gestionar el DBMS (el link para descargar esa interfaz es el siguiente: [PostgreSQL: Instaladores de Windows](#)).

Nota: En la instalación necesitaremos descargar algunas categorías para un mejor funcionamiento (esto dependerá de tus necesidades).

2-. A continuación, buscaremos el panel de administración, pgAdmin4, que se utiliza para gestionar todas las bases de datos que vayamos a crear con PostgreSQL.



3-. PostgreSQL crea una base de datos por defecto, pero nosotros crearemos la nuestra de la siguiente manera.

Haremos clic en “Query Tool” para habilitar el panel donde podemos introducir instrucciones SQL. Después crearemos nuestra base de datos mediante la tecla F5.



4.- Para poder ver la base de datos nos vamos a la sección de “Databases” de la parte izquierda y seleccionaremos la opción de “Refresh”.



5.- Conectaremos nuestro proyecto de Django en Visual Studio con la base de datos que acabamos de crear y para eso necesitamos una librería de Django llamada “psycopg2”.

> pip install psycopg2

```
PS C:\Users\52556\Desktop\DJANGO> cd TiendaOnline
PS C:\Users\52556\Desktop\DJANGO\TiendaOnline> pip install psycopg2
Collecting psycopg2
  Downloading psycopg2-2.9.9-cp312-cp312-win_amd64.whl.metadata (4.5 kB)
  Downloading psycopg2-2.9.9-cp312-cp312-win_amd64.whl (1.2 MB)
    1.2/1.2 MB 2.0 MB/s eta 0:00:00
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.9

[notice] A new release of pip is available: 24.0 -> 24.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\52556\Desktop\DJANGO\TiendaOnline>
```

Después cambiaremos la configuración en el apartado de “databases” en “settings”.



Luego, introduciremos en la consola el siguiente comando:

> python manage.py migrate

Por último le daremos clic en “refresh” a la base de datos y en la parte de “tables” podremos ver que se han cargado 13 tablas con éxito (entre ellas, las tablas que ya habíamos creado anteriormente).

Tables (13)
> auth_group
> auth_group_permissions
> auth_permission
> auth_user
> auth_user_groups

6-. Insertaremos un registro en la tabla “clientes” (mediante la consola) de PostgreSQL.

```
PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py shell
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.17.2 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from gestionPedidos.models import Clientes
In [2]: cli=Clientes(Nombre='Sebastián', Direccion='Avenida de la flores', Email='sebastiansolvil@gmail.com', Telefono='1234567')
In [3]: cli.save()
```

Después de eso, le daremos clic en “Refresh” a la tabla “artículos” y luego a “View/Edit Data”.

auth_user_user_permissions	django_admin_log	django_content_type	django_migrations	django_session	gestionPedidos_articu	gestionPedidos_client	gestionPedidos_pedid

Data Output					Messages	Notifications
	id [PK] bigint	Nombre character varying (30)	Direccion character varying (50)	Email character varying (254)	Telefono character varying (7)	
1	1	Sebastián	Avenida de la flores	sebastiansolvil@gmail.com	1234567	

Práctica 4: Consultas con PostgreSQL

1-. Crearemos los siguientes registros en la consola de Visual Studio.

	id [PK] bigint	Nombre character varying (30)	Seccion character varying (20)	Precio integer
1	1	Mesa	Decoración	90
2	2	Lámpara	Decoración	70
3	3	Pantalón	Confección	45
4	4	Destornillador	Ferretería	35
5	5	Balón	Deportes	25
6	6	Raqueta	Deportes	105
7	7	Muñeca	Juguetería	15
8	8	Tren eléctrico	Juguetería	135

2-. Para que el resultado de la consulta sea legible tendremos que usar el método `__str__()` en la tabla deseada. Cabe mencionar que por cada cambio que le hagamos a un modelo, tendremos que volver a realizar las migraciones correspondientes.

```
In [19]: exit()
PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py makemigrations
● No changes detected
PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py migrate
● Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestionPedidos, sessions
  Running migrations:
    No migrations to apply.
PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py shell
○ Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.17.2 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from gestionPedidos.models import Articulos
In [2]: Articulos.objects.filter(Seccion='Deportes')
Out[2]: <QuerySet [, <Articulos: El nombre es Raqueta la sección es Deportes y el precio es 105>]>
```

3-. Realizamos más consultas.

Nota: Devuelve el artículo mesa de la sección “decoración”.

```
In [4]: Articulos.objects.filter(Nombre='Mesa', Seccion='Decoración')
Out[4]: <QuerySet [<Articulos: El nombre es Mesa la sección es Decoración y el precio es 90>]>
```

Nota: Si queremos utilizar signos como < o >, debemos emplear una notación diferente, ya que la consola utiliza estos símbolos para otros propósitos.

```
In [7]: Articulos.objects.filter(Seccion='Deportes', Precio__gte=100)
Out[7]: <QuerySet [<Articulos: El nombre es Raqueta la sección es Deportes y el precio es 105>]>
```

```
In [8]: Articulos.objects.filter(Seccion='Deportes', Precio__lte=100)
Out[8]: <QuerySet [<Articulos: El nombre es Balón la sección es Deportes y el precio es 25>]>
```

Nota: Devuelve los artículos de la sección “deportes” en base a un determinado rango (50 – 150).

```
In [12]: Articulos.objects.filter(Seccion='Deportes', Precio__range=(10, 150))
Out[12]: <QuerySet [<Articulos: El nombre es Balón la sección es Deportes y el precio es 25>, <Articulos: El nombre es Raqueta la sección es Deportes y el precio es 105>]>
```

Nota: Enumera los artículos (mediante el precio) ascendentemente.

```
In [14]: Articulos.objects.filter(Precio__gte=50).order_by('Precio')
Out[14]: <QuerySet [<Articulos: El nombre es Lámpara la sección es Decoración y el precio es 70>, <Articulos: El nombre es Mesa la sección es Decoración y el precio es 90>, <Articulos: El nombre es Raqueta la sección es Deportes y el precio es 105>, <Articulos: El nombre es Tren eléctrico la sección es Juguetería y el precio es 135>]>
```

Nota: Enumera los artículos (mediante el precio) descendentemente.

```
In [15]: Articulos.objects.filter(Precio__gte=50).order_by('-Precio')
Out[15]: <QuerySet [<Articulos: El nombre es Tren eléctrico la sección es Juguetería y el precio es 135>, <Articulos: El nombre es Raqueta la sección es Deportes y el precio es 105>, <Articulos: El nombre es Mesa la sección es Decoración y el precio es 90>, <Articulos: El nombre es Lámpara la sección es Decoración y el precio es 70>]>
```

Capítulo 4: Panel de administración

Un **panel de administración** es una interfaz web que permite a los administradores de un sistema gestionar y supervisar diferentes aspectos de una aplicación o sitio web. En el contexto de Django, el panel de administración permite:

- 1-. Gestión de Usuarios:** Crear, editar y eliminar usuarios, así como gestionar sus permisos y roles.
- 2-. Gestión de Modelos:** Crear, editar y eliminar instancias de los modelos definidos en la aplicación (por ejemplo, productos, órdenes, artículos).
- 3-. Monitoreo del Sistema:** Ver estadísticas y registros (logs) del sistema para asegurar que todo funcione correctamente.
- 4-. Configuración del Sitio:** Ajustar configuraciones del sitio y de la aplicación desde una interfaz gráfica sin necesidad de modificar directamente el código fuente.

Características del Panel de Administración de Django

- 1-. Interfaz Amigable:** Proporciona una interfaz intuitiva y fácil de usar.

2-. Personalizable: Permite personalizar la apariencia y la funcionalidad según las necesidades específicas del proyecto.

3-. Seguridad: Incluye características de seguridad como autenticación y autorización para proteger el acceso a datos sensibles.

4-. Extensible: Permite la creación de funcionalidades adicionales mediante la definición de acciones personalizadas y la integración de plugins.

En Django, el panel de administración ya está habilitado en ciertos archivos como “admin.py”, “settings.py” y “urls.py”. Esto significa que, al ejecutar el servidor con el comando correspondiente (python manage.py runserver), la consola nos proporcionará una dirección IP y un puerto para acceder al sitio web de Django. Si agregamos “/admin/” a la URL del servidor, podremos acceder al panel de administración de Django.



Práctica 1: Crear un super usuario.

1-. Para poder acceder al panel de administración de Django tendremos que crear un super usuario (para ser el administrador), por lo que primero detendremos el servidor mediante la combinación de teclas Ctrl + c.

2-. Después tendremos que ejecutar el siguiente comando para crear el super usuario.

```
PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py createsuperuser
Username (leave blank to use '52556'): Sebastián
Email address: mac.sebas.acatlan@gmail.com
Password:
Password (again):
```

```
Superuser created successfully.
PS C:\Users\52556\Desktop\ Django\TiendaOnline>
```

Nota: Cabe mencionar que tu contraseña debe tener al menos 8 caracteres alfanuméricos, incluyendo mayúsculas, minúsculas y algún guión.

Nota: Volvemos a ejecutar el servidor.

3-. Ahora ingresaremos al panel de administración con los datos que hemos creado antes.



La interfaz principal es la siguiente:

The screenshot shows the Django Admin dashboard. At the top, there's a header bar with the title "Administración de Django". Below it, the main content area has a sidebar titled "AUTENTICACIÓN Y AUTORIZACIÓN" containing "Grupos" and "Usuarios" sections, each with "Agregar" and "Cambio" buttons. To the right, there's a sidebar titled "Acciones recientes" with "Mis acciones" and "Ninguno disponible".

Nota: En este panel podemos modificar, nombrar, agregar y eliminar usuarios.

4.- Cabe destacar que al unir nuestros 3 modelos con PostgreSQL, se generaron 30 tablas en total (3 de ellas corresponden a los modelos que creamos anteriormente). Esto nos permite comprender mejor la función de algunas tablas generadas por defecto. Por ejemplo, la tabla “auth_user” almacena los usuarios registrados, “auth_user_groups” almacena los grupos que podemos crear desde el panel de administración, y también hay tablas para permisos, migraciones, entre otras.

Tables (13)	
>	auth_group
>	auth_group_permissions
>	auth_permission
>	auth_user
>	auth_user_groups
>	auth_user_user_permissions
>	django_admin_log
>	django_content_type
>	django_migrations
>	django_session

5.- Le daremos clic derecho al apartado de “auth_user” y seleccionaremos la opción de “all rows”.

The screenshot shows the "auth_user" data table in the Django Admin. The "View/Edit Data" context menu is open, and the "All Rows" option is highlighted. Other options visible include "Search Objects...", "PSQL Tool", "Query Tool", and "Properties...".

Aquí podemos ver toda la información de los usuarios registrados hasta el momento.

Data Output		Messages	Notifications
id [PK] integer	password character varying (128)	last_login timestamp with time zone	is_superuser boolean
1	pbkdf2_sha256\$72000\$kmrEgoOuYg83e3gj13mrkk\$eJjmvmDm8YcD9ccNtH+muuBV1q4HMYqo/rxSuTqRFCw...	2024-07-10 00:38:06.178856-06	true
			Sebastián

Práctica 2: Manipulación de tablas desde el panel de administración.

1- Si queremos añadir a nuestro panel de administración una tabla o modelo en específico, debemos de importarlo desde el archivo “admin.py”.

```
TiendaOnline > gestionPedidos > admin.py
1   from django.contrib import admin
2
3   # Register your models here.
```

```
TiendaOnline > gestionPedidos > admin.py
1   from django.contrib import admin
2   from gestionPedidos.models import Clientes
3
4   # Register your models here.
5
6   admin.site.register(Clientes)
```

2- Iniciamos el servidor para poder visualizar la tabla “clientes” en el panel de administración.

The screenshot shows the Django Admin dashboard. At the top, it says "Administración de Django". Below that, "Administración del sitio". Under "AUTENTICACIÓN Y AUTORIZACIÓN", there are links for "Grupos" and "Usuarios" with "Agregar" and "Cambio" buttons. Under "GESTIONPEDIDOS", there is a link for "Clientes" with "Agregar" and "Cambio" buttons. To the right, there's a sidebar titled "Acciones recientes" with "Mis acciones" and "Ninguno disponible".

3- En ese apartado, podemos guardar, cambiar o eliminar los datos de cualquier cliente. Cualquier cambio realizado se reflejará en la base de datos de PostgreSQL.

The first screenshot shows the "Seleccionar clientes para cambiar" page with a dropdown for "Acción" and two checkboxes: "CLIENTES" and "Objeto Clientes (1)". A large black arrow points down to the second screenshot. The second screenshot shows the "Cambiar clientes" page for "Objeto Clientes (1)", displaying fields for "Nombre" (Sebastián), "Dirección" (Avenida de la flores), "Correo electrónico" (sebastiansolvil@gmail.com), and "Teléfono" (1234567). Buttons at the bottom include "SALVAR", "Guarda y agrega otro", "Guardar y continuar editando", and "Borrar". A second black arrow points down to the third screenshot. The third screenshot shows a "Data Output" table with columns: id [PK] bigint, Nombre character varying (30), Dirección character varying (50), Email character varying (254), and Telefono character varying (7). The table has one row with values: 1, Sebastián, Avenida de la flores, sebastiansolvil@gmail.com, and 1234567.

Nota: Cabe mencionar que desde un inicio todos los campos son obligatorios.

4- Si queremos hacer que un campo dentro del panel sea opcional (es decir, que no esté en negritas), necesitaremos modificar su respectivo campo en el archivo “models.py”, realizar las migraciones necesarias y luego reiniciar el servidor.

```

class Clientes(models.Model):
    Nombre = models.CharField(max_length=30)
    Direccion = models.CharField(max_length=50)
    Email = models.EmailField(blank=True, null=True)
    Telefono = models.CharField(max_length=7)

PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py makemigrations
Migrations for 'gestionPedidos':
• gestionPedidos\migrations\0002_alter_clientes_email.py
  - Alter field Email on clientes
PS C:\Users\52556\Desktop\ Django\TiendaOnline> python manage.py migrate
• Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestionPedidos, sessions
  Running migrations:
    Applying gestionPedidos.0002_alter_clientes_email... OK
○ PS C:\Users\52556\Desktop\ Django\TiendaOnline>

```

Cambiar clientes

Objeto Clientes (2)

Nombre:	María
Dirección:	Calle de la Amargura 123, Col. San Ángel, CDI
Correo electrónico:	marian@gmail.com
Teléfono:	765 432

SALVAR **Guarde y agregue otro** **Guardar y continuar editando**

Práctica 3: Personalización del panel de administración.

Nota: El panel siempre muestra la primera letra de cualquier campo en mayúscula.

Nota: El panel siempre eliminará cualquier tipo de guión que este en el nombre de un campo.

1.- Si queremos modificar el nombre de un campo en el panel de administración (sin que afecte el nombre verdadero en Visual Studio) tendremos que agregarle (en “models.py”) un segundo parámetro al modelo correspondiente (en este caso el de clientes).

```

class Clientes(models.Model):
    Nombre = models.CharField(max_length=30)
    Email = models.EmailField(blank=True, null=True)
    Direccion = models.CharField(max_length=50, verbose_name="La dirección")
    Telefono = models.CharField(max_length=7)

```

La dirección: Avenida de la flores

2.- Cuando un modelo tiene el método `__str__` implementado, la información en el panel de administración es más legible. Si un modelo no tiene esta característica, cada registro se mostrará como un objeto genérico. Para evitar esto, es necesario modificar el modelo para implementar el método `__str__`.

```

class Clientes(models.Model):
    Nombre = models.CharField(max_length=30)
    Email = models.EmailField(blank=True, null=True)
    Direccion = models.CharField(max_length=50, verbose_name="La dirección")
    Telefono = models.CharField(max_length=7)

    def __str__(self):
        return self.Nombre

```

Seleccionar clientes para cambiar

Acción: ----- Ir 0 de 2 seleccionados

CLIENTES
 María
 Sebastián

2 clientes

3-. Si deseamos mostrar más de un campo al seleccionar un modelo, necesitaremos crear una clase en el archivo admin.py para poder modificar o manipular el modelo correspondiente mediante un método en el archivo “admin.py”.

```
TiendaOnline > gestionPedidos > admin.py > ...
1  from django.contrib import admin
2  from gestionPedidos.models import Clientes
3  from gestionPedidos.models import Articulos
4  from gestionPedidos.models import Pedidos
5
6  # Register your models here.
7
8  class ClientesAdmin(admin.ModelAdmin):
9      list_display=("Nombre", "Direccion", "Telefono")
10
11 admin.site.register(Clientes, ClientesAdmin)
12 admin.site.register(Articulos)
13 admin.site.register(Pedidos)
```



Seleccionar clientes para cambiar AÑADIR CLIENTES +

Acción:	NOMBRE	LA DIRECCIÓN	TELÉFONO
<input type="checkbox"/>	María	Calle de la Amargura 123, Col. San Ángel, CDMX	765 432
<input type="checkbox"/>	Sebastián	Avenida de la flores	1234567

2 clientes

4-. Para agregar una casilla de búsqueda al panel de administración, debemos añadir una nueva instrucción similar al paso anterior. En este caso, deseamos habilitar la búsqueda por nombre y teléfono.

```
class ClientesAdmin(admin.ModelAdmin):
    list_display=("Nombre", "Direccion", "Telefono")
    search_fields=("Nombre", "Telefono")
```



Seleccionar clientes para cambiar AÑADIR CLIENTES +

Buscar

Acción:	NOMBRE	LA DIRECCIÓN	TELÉFONO
<input type="checkbox"/>	María	Calle de la Amargura 123, Col. San Ángel, CDMX	765 432
<input type="checkbox"/>	Sebastián	Avenida de la flores	1234567

2 clientes

Nota: Si deseamos eliminar varios registros a la vez, debemos seleccionarlos y luego utilizar la barra de acciones con la opción correspondiente.

Acción:

NOMBRE Eliminar clientes seleccionados

María Calle de la

Sebastián Avenida d

2 clientes

Práctica 4: Filtros y cambio de idioma.

1-. Si queremos filtrar los registros almacenados de una tabla (en este caso, filtraremos los registros por sección de la tabla "artículos"), procederemos a crear otra clase en el archivo del panel de administración correspondiente.

The diagram shows a code snippet on the left and a filter interface on the right, connected by a large black arrow pointing from left to right.

Code Snippet:

```
class ArticulosAdmin(admin.ModelAdmin):
    list_filter = ("Seccion",)

    admin.site.register(Clientes, ClientesAdmin)
    admin.site.register(Articulos, ArticulosAdmin)
    admin.site.register(Pedidos)
```

Filter Interface:

FILTER

- Show counts
- ↓ By Sección
- All
- Confección
- Decoración
- Deportes
- Ferretería
- Juguetería

Nota: Se nos generó por defecto (en la respectiva tabla) un cuadro de filtros con los campos que le especificamos en el archivo.

2-. Ahora insertaremos varios registros en la tabla “pedidos”.

Cambiar pedidos

Pedidos objeto (1)

Número: 1005

Fecha: 2024-07-11 Hoy | Notas: Tiene 6 horas de retraso en la hora del servidor.

Estado

SALVAR Guardar y agregue otro Guardar y continuar editando

Nota: Dado que el panel de administración está configurado en inglés, debemos agregar un formato de fecha americano para evitar errores.

Observamos los registros en PostgreSQL:

Data Output Messages Notifications

	id [PK] bigint	Número integer	Fecha date	Estado boolean
1	1	1005	2024-07-11	true
2	2	1006	2024-11-15	false
3	3	1007	2024-11-19	true
4	4	1008	2024-09-15	true

3-. Si deseamos filtrar por un campo de tipo fecha, necesitaremos crear otra clase en el mismo archivo. Además, agregaremos la opción de visualizar algunos de sus campos como en las tablas anteriores.

The diagram shows a code snippet on the left and a curved arrow pointing down to the right.

Code Snippet:

```
class PedidosAdmin(admin.ModelAdmin):
    list_display = ("Número", "Fecha")
    list_filter = ("Fecha",)
    date_hierarchy = "Fecha"

    admin.site.register(Clientes, ClientesAdmin)
    admin.site.register(Articulos, ArticulosAdmin)
    admin.site.register(Pedidos, PedidosAdmin)
```

NÚMERO	FECHA
1008	15 de septiembre de 2024
1007	19 de noviembre de 2024
1006	15 de noviembre de 2024
1005	11 de julio de 2024

4 pedidos

Nota: Estos 2 filtros (el de arriba y el de la derecha) nos ayudarán a simplificar nuestras búsquedas mucho más.

4-. Si deseamos traducir nuestro panel al idioma inglés, debemos ir al archivo “settings.py”, buscar la sección “LANGUAGE_CODE” y modificar el idioma según nuestras preferencias.

```
LANGUAGE_CODE = 'en-us'
```

En Django (en la parte de **autenticación y autorización**), los permisos “activo”, “staff” y “superusuario” tienen diferentes implicaciones y permisos:

1-. Activo:

Descripción: Un usuario marcado como "activo" puede autenticarse en el sitio.

Permisos: Puede iniciar sesión en el sitio web y acceder a las partes del sitio que requieren autenticación. Si un usuario no está marcado como "activo", no puede iniciar sesión en absoluto.

2-. Staff:

Descripción: Un usuario marcado como "staff" (personal) puede acceder al panel de administración de Django.

Permisos: Puede acceder al panel de administración, pero los permisos específicos dentro del panel dependen de los permisos adicionales asignados al usuario. Por ejemplo, podrían tener permisos para agregar, cambiar o eliminar ciertos modelos, pero no necesariamente todos.

3-. Superusuario:

Descripción: Un usuario marcado como "superusuario" tiene todos los permisos posibles en Django.

Permisos: Tiene acceso total a todas las partes del panel de administración, puede realizar cualquier acción (agregar, cambiar, eliminar) en cualquier modelo, y tiene acceso a todas las funcionalidades administrativas. Un superusuario también está automáticamente marcado como "activo" y "staff".

Práctica 5: Agregar permisos a usuarios y a grupos.

1-. Crearemos un usuario personal en el apartado de “autenticación y autorización”.

AUTENTICACIÓN Y AUTORIZACIÓN

Grupos [Agregar](#) [Cambio](#)

Usuarios [Agregar](#) [Cambio](#)

Agregar usuario

Primeramente, ingrese un nombre de usuario y una contraseña. A continuación, podrá editar más opciones de usuario.

Nombre de usuario: Obligatorio. 150 caracteres o menos. Letras, dígitos y @//+/-_, solamente.

Contraseña: Tu contraseña no puede ser muy similar a otra información personal. Su contraseña debe contener al menos 8 caracteres. Su contraseña no puede ser una contraseña de uso común. Su contraseña no puede ser completamente numérica.

Confirmación de contraseña: Introduzca la misma contraseña que antes, para la verificación.

[SALVAR](#) [Guarde y agregue otro](#) [Guardar y continuar editando](#)



2-. A continuación, completaremos las características de ese usuario, asignándole los permisos correspondientes (en este caso le daremos el permiso de usuario “activo”).

Cambiar de usuario

Germain

HISTORIA

Nombre de usuario: Obligatorio. 150 caracteres o menos. Letras, dígitos y @//+/-_, solamente.

Contraseña: algoritmo: pbkdf2_sha256 iteraciones: 720000 sal: jgpTlI***** hash: whbMtz***** Las contraseñas sin procesar no se almacenan, por lo que no hay forma de ver la contraseña de este usuario, pero puede cambiar la contraseña mediante este formulario.

Información personal

Nombre:

Apellido:

Dirección de correo electrónico:

Permisos

Activo Designa si este usuario debe ser tratado como activo. Anule la selección de esta opción en lugar de eliminar cuentas.

Situación del personal Designa si el usuario puede iniciar sesión en este sitio de administración.

Estado de superusuario Designa que este usuario tiene todos los permisos sin asignarlos explícitamente.

Nota: Durante este proceso, observaremos que la contraseña ingresada se encripta automáticamente para mayor seguridad.

3-. Guardaremos los cambios y podremos visualizar quién es parte del personal administrativo y quién no.

Seleccione el usuario que desea cambiar

Buscar

Acción: 0 de 2 seleccionados

NOMBRE DE USUARIO	DIRECCIÓN DE CORREO ELECTRÓNICO	NOMBRE	APellido	SITUACIÓN DEL PERSONAL
<input type="checkbox"/> Germain	germaingarcia@gmail.com	Germain	García	
<input type="checkbox"/> Sebastián	mac.sebas.acatlan@gmail.com			

2 usuarios

Por lo tanto, cuando intentemos ingresar al panel de administración, obtendremos un error ya que no somos parte del personal administrativo.



The screenshot shows the Django Admin login interface. At the top, it says "Administración de Django". Below that is a red error message box containing the text: "Ingrese el nombre de usuario y la contraseña correctos para una cuenta de personal. Tenga en cuenta que ambos campos pueden distinguir entre mayúsculas y minúsculas." Below the message are two input fields: "Nombre de usuario:" with "Germain García" typed in, and "Contraseña:" with a masked password. At the bottom is a blue "Inicia sesión" button.

4- Volveremos a ingresar al panel de administración con el superusuario que creamos y haremos clic en el apartado de “situación del personal”.



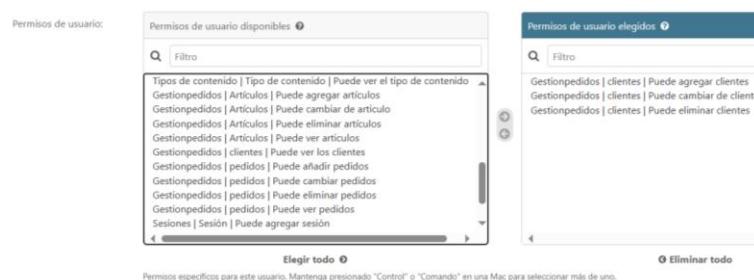
The screenshot shows the "Permisos" (Permissions) section in the Django Admin settings. It includes three checkboxes: "Activo" (Active) which is checked, "Situación del personal" (Personal status) which is also checked, and "Estado de superusuario" (Superuser status) which is unchecked. Each checkbox has a descriptive subtitle below it.

5- Al ingresar nuevamente con el nuevo usuario, podemos notar que ahora nos permite entrar al sitio, pero sin ningún permiso asignado.



The screenshot shows the Django Admin site index. At the top, it says "Administración de Django" and "BIENVENIDO, GERMAIN / VER SITIO / CAMBIAR CONTRASEÑA / CERRAR SESIÓN". Below that is a message: "Administración del sitio" followed by "No tienes permiso para ver ni editar nada.". To the right is a sidebar titled "Acciones recientes" (Recent actions) with the sub-section "Mis acciones" (My actions) showing "Ninguno disponible" (None available).

Por lo tanto, volveremos a acceder al panel con nuestro superusuario y asignaremos los permisos adecuados.



The screenshot shows the "Permisos de usuario" (User permissions) configuration screen. It has two main sections: "Permisos de usuario disponibles" (Available user permissions) on the left and "Permisos de usuario elegidos" (Selected user permissions) on the right. Both sections list various permission entries such as "Gestionpedidos | Artículos | Puede agregar artículos", "Gestionpedidos | Artículos | Puede eliminar artículos", etc. At the bottom, there are buttons for "Elegir todo" (Select all) and "Eliminar todo" (Delete all), with a note: "Permisos específicos para este usuario. Mantenga presionado 'Control' o 'Comando' en una Mac para seleccionar más de uno."

Nota: Con la tecla control podremos seleccionar más de un permiso.

Y como podemos ver, los permisos han sido registrados.

Nota: Si queremos asignar el mismo permiso a varios usuarios, podemos crear un grupo para facilitar la gestión.

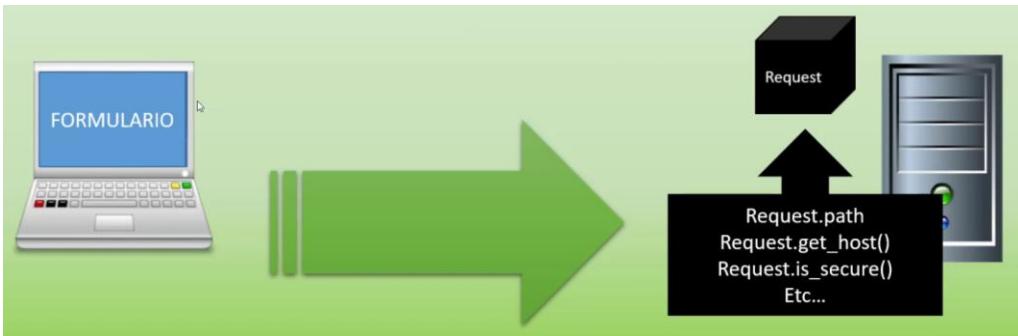
6-. Agregaremos un nuevo grupo en la misma sección en donde creamos un usuario y le asignaremos los permisos adecuados.

7-. Por último, asignaremos a cada usuario el grupo adecuado correspondiente.

Nota: Si queremos agregarle permisos adicionales a un usuario, le daremos clic a la parte de “permisos de usuario”.

8-. Podremos ver los permisos asignados en la cuenta creada anteriormente.

Capítulo 5: Formularios



Los **formularios** son una parte fundamental de las páginas web que permiten a los usuarios interactuar y enviar datos al servidor. Permiten recopilar información como nombres, contraseñas, comentarios y más, utilizando campos y controles como cajas de texto, botones de opción, menús desplegables, entre otros.

En Django, los formularios son representados mediante clases de Python que heredan funcionalidades de `django.forms.Form` o `django.forms.ModelForm`. Estas clases permiten definir los campos que un formulario debe contener y especificar validaciones y comportamientos adicionales. Django facilita la creación, validación y procesamiento de formularios, además de proporcionar herramientas para renderizarlos en plantillas HTML de manera segura y eficiente.

Métodos GET y POST:

Las diferencias entre los métodos HTTP GET y POST son fundamentales en el desarrollo web, especialmente al trabajar con formularios y solicitudes en aplicaciones Django u otros frameworks web. Algunas diferencias son:

1-. Método GET:

- Se utiliza para solicitar datos del servidor que están contenidos en una URL específica.
- Los parámetros enviados con GET son visibles en la URL del navegador.
- Menos seguro que POST porque los parámetros son visibles en la URL y pueden quedar registrados en logs del servidor.
- Puede ser almacenado en caché por el navegador o proxies, lo que puede llevar a que los parámetros sean visibles incluso después de que el usuario haya cerrado la sesión.
- Limitado a la longitud máxima de la URL, que puede ser un problema para grandes cantidades de datos.

2-. Método POST:

- Se utiliza para enviar datos al servidor para ser procesados, generalmente a través de un formulario HTML.
- En Django, se maneja típicamente en vistas usando el objeto `request` para obtener los datos enviados por el usuario.

- Los parámetros no son visibles en la URL del navegador, lo que hace que sea más seguro para enviar datos sensibles como contraseñas.

- No es almacenado en caché por el navegador o proxies.

- No tiene restricciones en la longitud de los datos enviados, lo que permite enviar grandes cantidades de información.

En Django, un **request** (petición) es un objeto que representa la información enviada por un cliente web al servidor. Contiene detalles sobre la solicitud HTTP realizada por el cliente, como la URL solicitada, los parámetros de la solicitud, encabezados HTTP, tipo de método HTTP, y más.

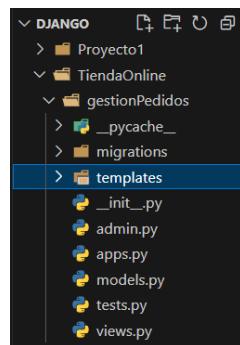
Consideraciones:

1-. Selección del Método: Escoge GET cuando solo necesitas recuperar datos del servidor y no se envían datos sensibles. Usa POST cuando necesitas enviar datos sensibles o cuando realizas cambios en el servidor que afectan los datos.

2-. Seguridad: Asegúrate de usar POST para operaciones que modifican o actualizan datos importantes para evitar que los datos se expongan a través de la URL o sean almacenados en caché.

Práctica 1: Creación de un formulario HTML de forma manual y envió de datos al servidor.

1-. Este formulario debe de estar en una carpeta llamada “templates”.



2-. Crearemos un archivo llamado “búsqueda:productos.html” y generaremos nuestro primer formulario.

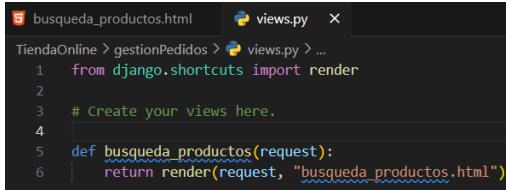
3-. Crearemos una casilla de búsqueda, un cuadro de texto de búsqueda y un botón de enviar.

```
<html>
  <head>
    |   <title> </title>
  </head>
  <body>
    |   <form action="/buscar/" method="GET">
    |       <input type="text" name="prd">
    |       <input type="submit" value="Buscar">
  </body>
</html>
```

The image shows a code editor window with the file 'busqueda_productos.html' open. The code is written in HTML. It starts with a basic structure: <html>, <head>, and <body>. Inside the body, there is a form with an action attribute set to '/buscar/' and a method attribute set to 'GET'. Inside the form, there is an input field of type 'text' with the name attribute set to 'prd'. Below it, there is an input field of type 'submit' with the value attribute set to 'Buscar'. The code is numbered from 1 to 11 on the left side.

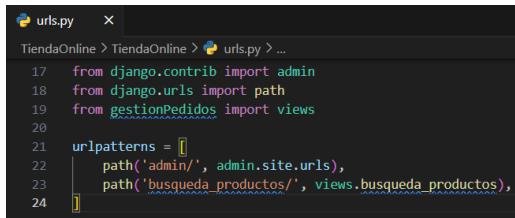
Nota: El objetivo de la línea 5 es que, al presionar el botón de enviar del formulario, este nos dirija inicialmente a una URL llamada “buscar”, utilizando el método especificado para enviar la información del formulario.

4- Crearemos una vista que nos permita abrir el formulario previamente creado.



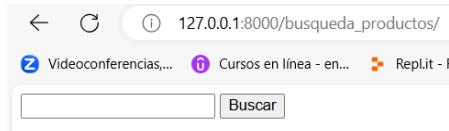
```
busqueda_productos.html views.py
TiendaOnline > gestionPedidos > views.py > ...
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def busqueda_productos(request):
6     return render(request, "busqueda_productos.html")
```

Y después debemos de registrar esa URL o path:



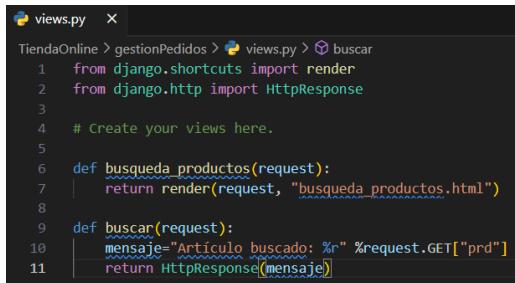
```
urls.py
TiendaOnline > TiendaOnline > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path
19 from gestionPedidos import views
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('busqueda_productos/', views.busqueda_productos),
24 ]
```

5- Observamos el resultado al ejecutar en la barra de búsqueda la siguiente URL:
127.0.0.1:8000/busqueda_productos/



Nota: Al pulsar el botón de buscar, debería llevarnos a la URL: /buscar/. Sin embargo, esta aún no está definida.

6- Debemos de crear una vista para la URL de /buscar/ y después registrarla en el path.



```
views.py
TiendaOnline > gestionPedidos > views.py > buscar
1 from django.shortcuts import render
2 from django.http import HttpResponseRedirect
3
4 # Create your views here.
5
6 def busqueda_productos(request):
7     return render(request, "busqueda_productos.html")
8
9 def buscar(request):
10    mensaje="Artículo buscado: %r" %request.GET["prd"]
11    return HttpResponseRedirect(mensaje)
```

Nota: Aquí creamos la vista que nos dirá si la información que hemos ingresado en el formulario llega efectivamente al servidor o no.

```

urls.py  X
TiendaOnline > TiendaOnline > urls.py > ...
17   from django.contrib import admin
18   from django.urls import path
19   from gestionPedidos import views
20
21   urlpatterns = [
22       path('admin/', admin.site.urls),
23       path('busqueda_productos/', views.busqueda_productos),
24       path('buscar/', views.buscar),
25   ]

```

7-. Por último, ingresaremos el nombre de un producto en la barra de búsqueda para obtener un resultado por parte del servidor.



Práctica 2: Búsqueda de información en una base de datos.

1-. Construiremos una condición if en la vista “buscar” que nos mostrará diferentes opciones dependiendo de los casos que puedan surgir al ingresar o no ingresar información.

```

def buscar(request):
    if request.GET["prd"]:
        mensaje="Artículo buscado: %r" %request.GET["prd"]
    else:
        mensaje="No has introducido nada"
    return HttpResponseRedirect(mensaje)

```

2-. Ahora recuperaremos lo que hay en “prd”, que es el cuadro de texto, y haremos que se busque en la base de datos.

```

TiendaOnline > gestionPedidos > views.py > buscar
1  from django.shortcuts import render
2  from django.http import HttpResponseRedirect
3  from gestionpedidos.models import Articulos
4
5  # Create your views here.
6
7  def busqueda_productos(request):
8      return render(request, "busqueda_productos.html")
9
10 def buscar(request):
11     if request.GET["prd"]:
12         producto=request.GET["prd"]
13         articulos=Articulos.objects.filter(Nombre__icontains=producto)
14         return render(request, "resultados_busqueda.html", {"articulos":articulos, "query":producto})
15     else:
16         mensaje="No has introducido nada"
17     return HttpResponseRedirect(mensaje)
18

```

Nota: El método “icontains” funciona como la cláusula “LIKE”, permitiendo buscar coincidencias parciales en la base de datos.

Nota: Una vez hecha la consulta, nos dirigiremos a un archivo HTML que aún no tenemos (pero que llamaremos “resultados_busqueda.html”), el cual debe ser capaz de mostrar la información con sus características.

3-. Crearemos la página HTML llamada “resultados_busqueda.html” para relacionar y estructurar los resultados que vayamos a obtener de la base de datos.

```

TiendaOnline > gestionPedidos > templates > resultados_busqueda.html > html
1  <html>
2   <body>
3     <p>Estás buscando: <strong>{{query}}</strong></p>
4     {% if articulos %}
5       <p>Artículos encontrados: {{articulos|length}} articulos</p>
6       <ul>
7         {% for articulo in articulos %}
8           <li>
9             | {{articulo.Nombre}} &nbsp; {{articulo.Seccion}} &nbsp; {{articulo.Precio}}
10            </li>
11          {% endfor %}
12        </ul>
13      {% else %}
14        <p>No está el artículo encontrado:</p>
15      {% endif %}
16    </body>
17  </html>

```

- 4-. Introduciremos un artículo en el formulario y comprobaremos si existe o no dentro de la base de datos de PostgreSQL.

The image contains four screenshots of a web browser window. The top-left screenshot shows a search bar with 'serrucho' and a 'Buscar' button. The top-right screenshot shows a search bar with 'muñeca' and a 'Buscar' button. The bottom-left screenshot shows the search results for 'serrucho' with the message 'Estás buscando: serrucho' and 'No está el artículo encontrado'. The bottom-right screenshot shows the search results for 'muñeca' with the message 'Estás buscando: muñeca', 'Artículos encontrados: 1 artículos', and a list item '• Muñeca Juguetería 15'.

Si introducimos un nombre muy largo en una barra de búsqueda, pueden ocurrir varias cosas dependiendo de cómo esté configurada la aplicación y su manejo de entradas:

- 1-. Limitación del Campo de Texto:** Si el campo de texto tiene un límite de caracteres, solo se permitirá ingresar hasta esa cantidad de caracteres.
- 2-. Rendimiento del Servidor:** Consultar con una cadena de búsqueda extremadamente larga puede afectar el rendimiento del servidor, especialmente si no se han implementado optimizaciones o restricciones adecuadas.
- 3-. Errores de la Base de Datos:** Dependiendo de cómo se construyan las consultas, una entrada muy larga podría causar errores en la base de datos, especialmente si no se han tomado precauciones contra inyecciones SQL.
- 4-. Seguridad:** Aceptar cadenas de búsqueda muy largas sin validación adecuada puede abrir vulnerabilidades, como ataques de inyección de código o denegación de servicio.

Práctica 3: Limitar el número de caracteres en una búsqueda en la base de datos.

- 1-. Nos vamos al archivo “views” y añadiremos en la vista “buscar” el siguiente condicional:

```

def buscar(request):
    if request.GET["prd"]:
        producto=request.GET["prd"]
        if len(producto)>20:
            mensaje="Texto demasiado largo"
        else:
            articulos=Articulos.objects.filter(Nombre_icontains=producto)
            return render(request, "resultados_busqueda.html", {"articulos":articulos, "query":producto})
        else:
            mensaje="No has introducido nada"
    return HttpResponseRedirect(mensaje)

```

- 2-. Actualizaremos el servidor y pondremos un nombre demasiado largo.



Práctica 4: Formulario de contacto (parte 1).

1-. Crearemos una nueva vista para el formulario.

```
def contacto(request):
    return render(request, "contacto.html")
```

2-. Construir una URL en base a esa vista.

```
path('contacto/', views.contacto),
```

3-. Generaremos la página HTML que estará relacionada a esa vista.

```

TiendaOnline > gestionPedidos > templates > contacto.html > html
1 <html>
2   <head>
3     |   <title>Contáctanos</title>
4   </head>
5   <body>
6     <h1>Contáctanos</h1>
7     <form action="/contacto/" method="POST">
8       <p>Asunto:<input type="text" name="asunto"></p>
9       <p>Email:<input type="text" name="email"></p>
10      <p>Mensaje:<textarea name="mensaje" rows="15" cols="45"></textarea></p>
11      <input type="submit" value="Enviar">
12    </form>
13  </body>
14 </html>
```

4-. Observamos el resultado al ejecutar en la barra de búsqueda la siguiente URL:

127.0.0.1:8000/contacto/

El **CSRF token** (Cross-Site Request Forgery token) es un mecanismo de seguridad utilizado en aplicaciones web para prevenir ataques de falsificación de solicitudes entre sitios (Cross-Site Request Forgery, CSRF).

¿Qué es un ataque CSRF?

Un ataque CSRF ocurre cuando un atacante engaña a un usuario autenticado para que realice una acción no deseada en una aplicación web en la que el usuario está autenticado. Por ejemplo, el atacante puede hacer que el usuario haga clic en un enlace malicioso que envía una solicitud al servidor de la aplicación web para realizar una acción (como cambiar la contraseña del usuario) sin el conocimiento o consentimiento del usuario.

¿Cómo funciona el CSRF Token?

El CSRF token funciona de la siguiente manera:

1-. Generación del Token: El servidor genera un token único y lo envía al cliente (navegador del usuario) cuando se carga la página o el formulario.

2-. Incorporación del Token en la Solicitud: El token se incluye en todas las solicitudes que el cliente envía al servidor, generalmente como un campo oculto en los formularios o en el encabezado de la solicitud.

3-. Verificación del Token: Cuando el servidor recibe una solicitud, verifica que el token proporcionado en la solicitud coincide con el token generado y almacenado en la sesión del usuario. Si los tokens coinciden, la solicitud se considera legítima y se procesa; de lo contrario, se rechaza.

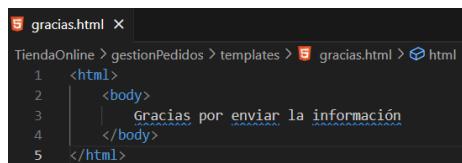
5-. Introduciremos el token en el formulario de la página HTML.

```
<h1>Contáctanos</h1>
<form action="/contacto/" method="POST">{% csrf_token %}
```

6-. Para comprobar si el formulario está funcionando utilizando el método POST, se agregará un condicional a la vista “contacto”.

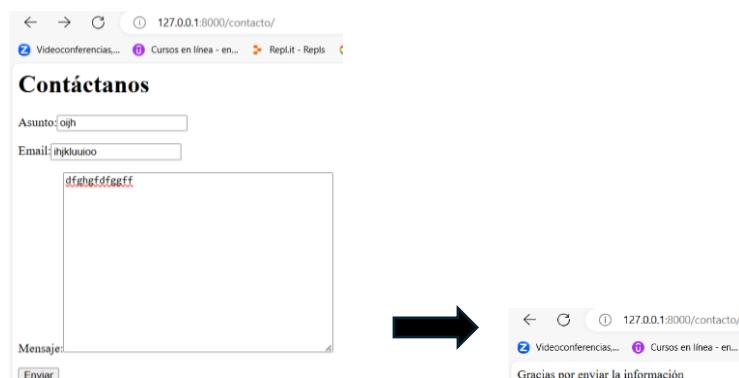
```
def contacto(request):
    if request.method=="POST":
        return render(request, "gracias.html")
    return render(request, "contacto.html")
```

Nota: Se tendrá que crear una página HTML para relacionarla con la vista.



```
gracias.html x
TiendaOnline > gestionPedidos > templates > gracias.html > html
1  <html>
2   <body>
3     Gracias por enviar la información
4   </body>
5 </html>
```

7-. Visualizaremos el mensaje que nos devuelve el formulario al enviar la información introducida en la lista de contacto.



Práctica 5: Envío de emails (parte 2).

1.- Para enviar correos electrónicos con Django, necesitamos configurar un servidor de correo y establecer los parámetros correspondientes en el archivo “settings.py”.

```
settings.py < TiendaOnline > settings.py > ...  
133 EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"  
134 EMAIL_HOST = "smtp.gmail.com"  
135 EMAIL_PORT = 587  
136 EMAIL_USE_TLS = True  
137 EMAIL_HOST_USER = "sebastiansolvil@gmail.com"  
138 EMAIL_HOST_PASSWORD = "llcg eqey sakt tmds"
```

Nota: Para poder enviar correos electrónicos mediante Gmail, es necesario configurar nuestra cuenta para permitir el acceso de aplicaciones de terceros.

Nota: La cuarta línea indica el protocolo de seguridad que utiliza Google para enviar correos.

Nota: Para comprobar si estamos bien, ejecutaremos en la consola los siguientes comandos.

```
In [2]: from django.core.mail import send_mail  
...  
...: send_mail(  
...:     'Asunto',  
...:     'Mensaje',  
...:     'sebastiansolvil@gmail.com',  
...:     ['mac.sebas.acatlan@gmail.com'],  
...:     fail_silently=False  
...:  
Out[2]: 1
```

Asunto ➔ Recibidos x
sebastiansolvil@gmail.com para mí ▾
Mensaje
Responder Reenviar

2.- La vista “contacto” maneja solicitudes POST para enviar un correo electrónico con el asunto y mensaje proporcionados por el usuario, y luego muestra una página de agradecimiento. Si no es una solicitud POST, muestra la página de contacto.

```
from django.shortcuts import render  
from django.http import HttpResponseRedirect  
from gestionPedidos.models import Articulos  
from django.core.mail import send_mail  
from django.conf import settings
```

```
def contacto(request):  
    if request.method=="POST":  
        subject=request.POST["asunto"]  
        message=request.POST["mensaje"] + " " + request.POST["email"]  
        email_from=settings.EMAIL_HOST_USER  
        recipient_list=["mac.sebas.acatlan@gmail.com"]  
        send_mail(subject,message,email_from,recipient_list)  
        return render(request,"gracias.html")  
    return render(request,"contacto.html")
```

3.- Introduciremos la información en el formulario y haremos clic en el botón de enviar. Luego, revisaremos el correo del destinatario para verificar el mensaje enviado.

Contáctanos

Asunto: Asunto de prueba
Email: sebastiansolvil@gmail.com
Mensaje:
Este es un mensaje de prueba
Enviar

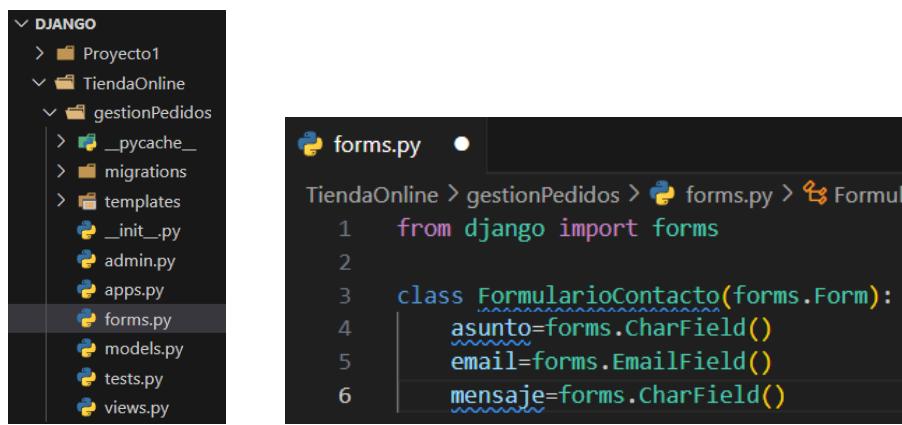
Asunto de prueba ➔ Recibidos x
sebastiansolvil@gmail.com para mí ▾
Este es un mensaje de prueba sebastiansolvil@gmail.com
Responder Reenviar

Gracias por enviar la información

Práctica 6: Creación de formularios con API forms.

La API de formularios en Django, conocida como **django.forms**, es un componente que permite crear, validar y manejar formularios web de una manera sencilla y estructurada. Es una herramienta poderosa que facilita la creación de formularios, desde los más simples hasta los más complejos, integrándose perfectamente con el modelo de datos de Django y su sistema de validación.

1.- Comenzaremos creando un archivo llamado “forms.py” (preferentemente ubicado junto al archivo “views.py”). Dentro de este archivo, crearemos una clase para construir un formulario. Es importante recordar que, si nuestra aplicación Django necesita cinco formularios, deberemos crear una clase por cada formulario dentro de “forms.py”.



2.- Para observar la simplicidad con la que Django construye formularios simplemente al definir una clase, ingresaremos los siguientes comandos en la consola.

```
PS C:\Users\52556\Desktop\DJANGO\TiendaOnline> python manage.py shell
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.17.2 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from gestionPedidos.forms import Formulario
In [2]: miFormulario=Formulario()
```

Imprimiremos en pantalla la instancia llamada “miFormulario”. Al ejecutar la instrucción, observamos que se genera automáticamente una serie de etiquetas HTML, y que, por defecto, cada uno de esos campos se establece como obligatorio, lo que implica una primera validación.

```
In [3]: print(miFormulario)
<div>
    <label for="id_asunto">Asunto:</label>

    <input type="text" name="asunto" required id="id_asunto">
</div>
<div>
    <label for="id_email">Email:</label>

    <input type="email" name="email" maxlength="320" required id="id_email">
</div>
<div>
    <label for="id_mensaje">Mensaje:</label>

    <input type="text" name="mensaje" required id="id_mensaje">
</div>
```

3.- También podemos construir un formulario donde, además de definir los campos pertinentes, introducimos información en cada uno de ellos.

```
In [4]: miFormulario=Formulario({'asunto':'prueba', 'email':'sebastiansolvil@gmail.com', 'mensaje':'mensaje de prueba'})
```

4-. Si escribimos la siguiente instrucción, podremos verificar si el formulario previamente escrito es válido o correcto.

```
In [5]: miFormulario.is_valid()
Out[5]: True
```

Y si es válido, podremos utilizar el método “cleaned_data” para que nos devuelva un diccionario con la información que se ha enviado en el formulario.

```
In [6]: miFormulario.cleaned_data
Out[6]:
{'asunto': 'prueba',
'email': 'sebastiansolvil@gmail.com',
'mensaje': 'mensaje de prueba'}
```

5-. Ahora haremos lo mismo, pero esta vez modificando la vista “contacto”.

```
def contacto(request):
    if request.method=="POST":
        miFormulario=Formulario(request.POST)
        if miFormulario.is_valid():
            infForm=miFormulario.cleaned_data
            send_mail(infForm['asunto'], infForm['mensaje'], infForm.get('email', 'sebastiansolvil@gmail.com'), ['mac.sebas.acatlan@gmail.com'],)
        else:
            miFormulario=Formulario()
    return render(request, "formulario_contacto.html", {"form":miFormulario})
```

6-. Nos dirigiremos a “templates” para crear un nuevo archivo llamado “formulario_contacto.html” y luego escribiremos el código HTML correspondiente para crear las validaciones, como hicimos en la consola.

```
TiendaOnline > gestionPedidos > templates > formulario_contacto.html > html
  1  <html>
  2      <body>
  3          <h1>Contáctanos</h1>
  4          {% if form.errors %}
  5              <p style="color: red;">Por favor revisa este campo</p>
  6          {% endif %}
  7          <form action="" method="POST">{% csrf_token %}
  8              <table>
  9                  </table>
 10          </form>
 11      </body>
 12  </html>
```

Nota: También generaremos la etiqueta “form” y la etiqueta “table” (ya que como vimos en la consola, esta no nos lo genera).

7-. Por último, le indicaremos a nuestro código que nos genere automáticamente el formulario (como en la consola).

```

<html>
  <body>
    <h1>Contáctanos</h1>
    {% if form.errors %}
      <p style="color: red;">Por favor revisa este campo</p>
    {% endif %}
    <form action="" method="POST">{{ csrf_token }}
      <table>
        {{ form.as_table }}
      </table>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>

```

8-. Al cargar la página, podemos visualizar el formulario generado mediante API forms.

Contáctanos

Asunto:	<input type="text"/>
Email:	<input type="text"/>
Mensaje:	<input type="text"/>
<input type="button" value="Enviar"/>	

Si no ingresamos información en ninguno de los campos y presionamos enviar, el formulario nos mostrará un aviso indicando que es obligatorio completarlos, validando cada uno de ellos automáticamente.

Contáctanos

Asunto:	<input type="text"/>
Email:	<input type="text"/>
Mensaje:	<input type="text"/> ! Rellene este campo.
<input type="button" value="Enviar"/>	

Una vez introducida la información, daremos clic en enviar.

Contáctanos

Asunto:	<input type="text" value="Asunto de prueba"/>
Email:	<input type="text" value="sebastiansolvil@gmail.com"/>
Mensaje:	<input type="text" value="Esto es un mensaje de prueba"/>
<input type="button" value="Enviar"/>	



Gracias por enviar la información





Proyecto

Parte 1: Vistas yUrls

1-. Creamos una carpeta principal para el proyecto.

```

DJANGO
> Projeto1
> ProjetoSebas
> TiendaOnline

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

● PS C:\Users\52556\Desktop\ Django> django-admin startproject ProjetoSebas
○ PS C:\Users\52556\Desktop\ Django>

```

2-. Creamos la aplicación del proyecto.

```

DJANGO
> Projeto1
> ProjetoSebas
> ProjetoSebas
> ProjetoSebasApp
  manage.py
> TiendaOnline

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

PS C:\Users\52556\Desktop\ Django> django-admin startproject ProjetoSebas
● PS C:\Users\52556\Desktop\ Django> cd ProjetoSebas
● PS C:\Users\52556\Desktop\ Django\ProjetoSebas> python manage.py startapp ProjetoSebasApp
○ PS C:\Users\52556\Desktop\ Django\ProjetoSebas>

```

3-. Realizamos las migraciones correspondientes y arrancamos el servidor.

```

PS C:\Users\52556\Desktop\ Django\ProjetoSebas> 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_user_email_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
Performing system checks...

System check identified no issues (0 silenced).
July 16, 2024 - 18:22:43
Django version 5.0.6, using settings 'ProjetoSebas.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

```

4-. Construimos las vistas en el archivo “views.py”.

```

from django.shortcuts import render
from django.shortcuts import HttpResponseRedirect

# Create your views here.

def home(request):
    return HttpResponseRedirect("Inicio")

def servicios(request):
    return HttpResponseRedirect("Servicios")

def tienda(request):
    return HttpResponseRedirect("Tienda")

def blog(request):
    return HttpResponseRedirect("Blog")

def contacto(request):
    return HttpResponseRedirect("Contacto")

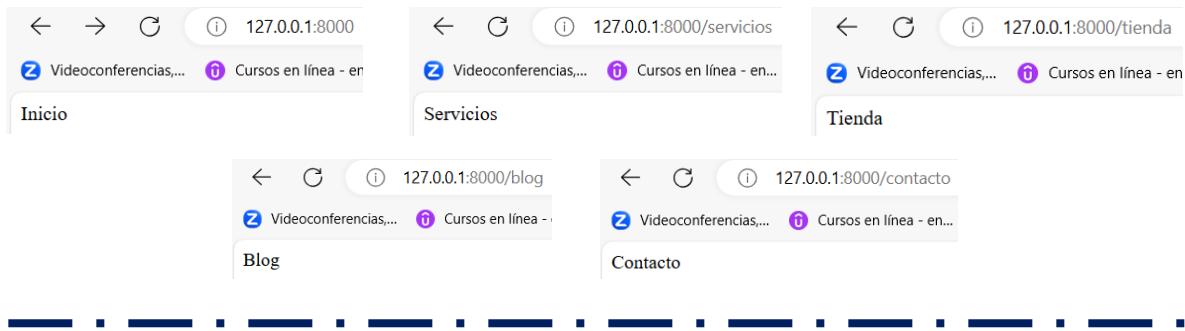
```

5-. Importamos las URLs de cada una de las vistas.

```
from django.contrib import admin
from django.urls import path
from ProyectoSebasApp import views

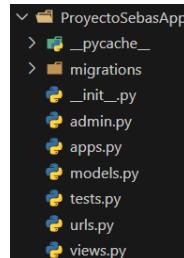
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.inicio, name="Inicio"),
    path('servicios', views.servicios, name="Servicios"),
    path('tienda', views.tienda, name="Tienda"),
    path('blog', views.blog, name="Blog"),
    path('contacto', views.contacto, name="Contacto"),
]
```

6-. Comprobamos en la URL del servidor cada una de nuestras vistas.



Parte 2: Organizar de mejor manera lasUrls y los templates

1-. Crearemos un archivo “urls.py” dentro de cada aplicación para que cada una contenga su propio archivo de URLs, lo que ayudará a organizar mejor todas las URLs que usaremos en el proyecto.

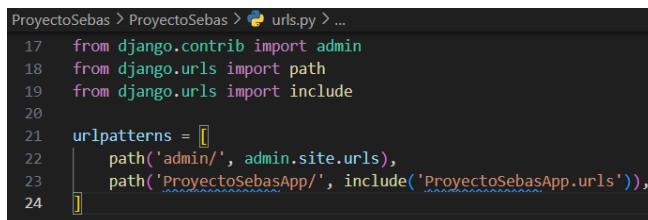


2-. Importaremos la función “path” y las vistas, así como el archivo de “urlpatterns”, en el archivo “urls.py” de la aplicación. Luego eliminaremos estos elementos de la carpeta principal del proyecto.

```
ProyectoSebas > ProyectoSebas > urls.py > ...
17   from django.contrib import admin
18   from django.urls import path
19
20   urlpatterns = [
21       path('admin/', admin.site.urls),
22   ]
```

```
ProyectoSebas > ProyectoSebasApp > urls.py > ...
1  from django.urls import path
2  from ProyectoSebasApp import views
3
4  urlpatterns = []
5  path('', views.inicio, name="Inicio"),
6  path('servicios', views.servicios, name="Servicios"),
7  path('tienda', views.tienda, name="Tienda"),
8  path('blog', views.blog, name="Blog"),
9  path('contacto', views.contacto, name="Contacto"),
10 ]
```

3-. Enlazaremos el archivo “urls.py” del proyecto con el archivo “urls.py” de la aplicación importando tanto la función “include” como el path correspondiente.



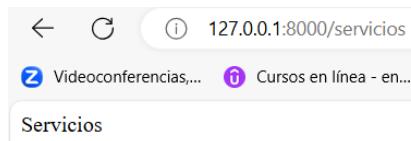
```
ProyectoSebas > ProyectoSebas > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path
19 from django.urls import include
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('ProyectoSebasApp/', include('ProyectoSebasApp.urls')),
24 ]
```

4-. Ahora podremos acceder a todas las URLs de la siguiente manera.

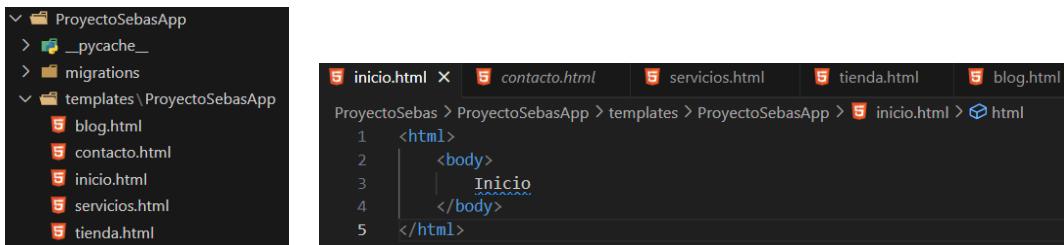


Nota: Si queremos simplificar la URL como antes, haremos una pequeña modificación en el path del archivo “urls.py” de la carpeta principal del proyecto.

```
path('', include('ProyectoSebasApp.urls')),
```



5-. Nos dirigiremos a nuestro archivo “views.py” y sustituiremos cada una de las vistas por los archivos HTML pertinentes. Por eso, trabajaremos con templates, creando una carpeta con este nombre y, dentro de ella, una subcarpeta con el nombre de nuestro proyecto, donde crearemos los archivos HTML.



6-. Para que cada archivo HTML funcione, tendremos que registrar nuestra aplicación. Nos dirigiremos al archivo “settings.py” en la carpeta principal y añadiremos nuestra aplicación a la lista de INSTALLED_APPS.

```

ProyectoSebas > ProyectoSebas > settings.py > ...
31     # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'ProyectoSebasApp',
41 ]

```

```

def inicio(request):
    return render(request, "ProyectoSebasApp/inicio.html")

def servicios(request):
    return render(request, "ProyectoSebasApp/servicios.html")

def tienda(request):
    return render(request, "ProyectoSebasApp/tienda.html")

def blog(request):
    return render(request, "ProyectoSebasApp/blog.html")

def contacto(request):
    return render(request, "ProyectoSebasApp/contacto.html")

```

Parte 3: Formatear el sitio web con Bootstrap y modificar los templates para trabajar con la herencia de estos.

Bootstrap es un framework front-end de código abierto que facilita el desarrollo de sitios web y aplicaciones web responsivas. Fue creado por Mark Otto y Jacob Thornton en Twitter y lanzado en 2011. Bootstrap proporciona una colección de herramientas y componentes predefinidos, como botones, formularios, menús de navegación, y cuadros de diálogo, que pueden ser utilizados para crear interfaces de usuario coherentes y atractivas.

Características principales de Bootstrap:

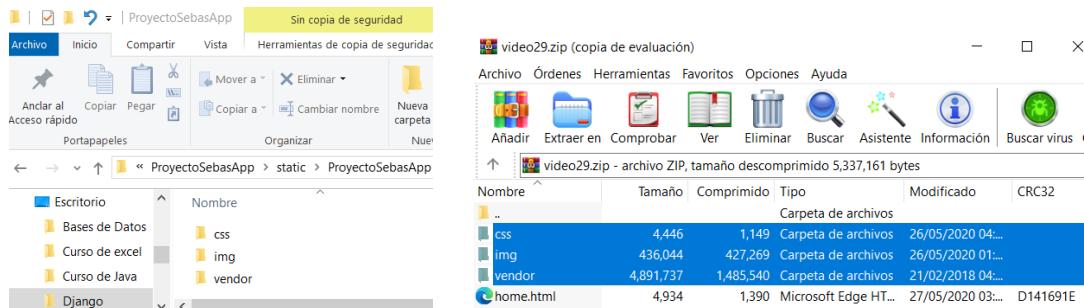
1-.Grid System (Sistema de rejilla):

- Permite crear diseños responsivos de manera sencilla utilizando un sistema de 12 columnas.
- Facilita la creación de diseños adaptables a diferentes tamaños de pantalla (dispositivos móviles, tabletas, computadoras de escritorio, etc.).

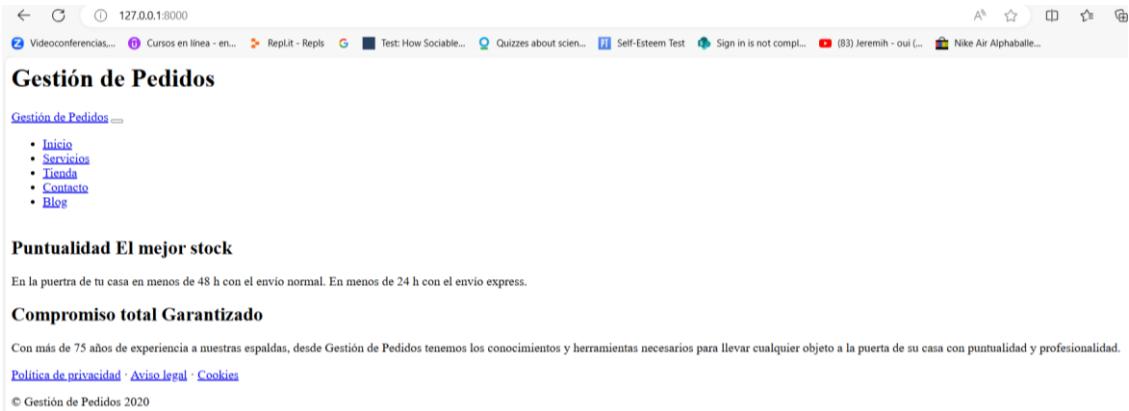
2-.Componentes:

- Ofrece una amplia variedad de componentes de interfaz de usuario como botones, formularios, alertas, modales, menús de navegación, tarjetas, etc.
- Estos componentes son personalizables y fáciles de integrar en cualquier

1-. Crearemos una carpeta llamada “static” y una subcarpeta dentro de la estructura de nuestra aplicación “ProyectoSebasApp”. En esta subcarpeta almacenaremos un archivo comprimido que contendrá diversas bibliotecas de Bootstrap, incluyendo CSS, JavaScript, entre otras, que utilizaremos a lo largo del proyecto.



2-. Eliminaremos la página HTML llamada “inicio” y la reemplazaremos por la carpeta del archivo ZIP denominada “Home”. Por ende si volvemos a cargar la página principal tendremos el siguiente resultado.



Gestión de Pedidos

Gestión de Pedidos

- Inicio
- Servicios
- Tienda
- Contacto
- Blog

Puntualidad El mejor stock

En la puerta de tu casa en menos de 48 h con el envío normal. En menos de 24 h con el envío express.

Compromiso total Garantizado

Con más de 75 años de experiencia a nuestras espaldas, desde Gestión de Pedidos tenemos los conocimientos y herramientas necesarios para llevar cualquier objeto a la puerta de su casa con puntualidad y profesionalidad.

Política de privacidad · Aviso legal · Cookies

© Gestión de Pedidos 2020

3-. Modificaremos los enlaces del archivo “inicio” para darle formato a nuestra página principal, que actualmente carece de formato. Le indicaremos al archivo que cargue todo el contenido de la carpeta “static” y luego estructuraremos correctamente los enlaces mediante la etiqueta “static”.

```
{% load static %}  
<!-- Bootstrap -->  
<link href="{% static 'ProyectoSebasApp/vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">  
  
<!-- Styles -->  
<link href="{% static 'ProyectoSebasApp/css/gestion.css' %}" rel="stylesheet">  
  
  
  
<!-- Bootstrap -->  
<script src="{% static 'ProyectoSebasApp/vendor/jquery/jquery.min.js' %}"></script>  
<script src="{% static 'ProyectoSebasApp/vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>
```

4-. Al realizar los cambios pertinentes, cargaremos la página principal y veremos la verdadera funcionalidad de Bootstrap.



5- Copiaremos el archivo “**“inicio”**” y lo pegaremos en la subcarpeta “**“ProyectoSebasApp”**”, renombrándolo como “**“base.html”**” para que funcione como nuestra plantilla base.



En esa copia eliminaremos las secciones de “**“heading”**” y “**“message”**” y pondremos una etiqueta “**“block content”**”.

```
<!-- Contenido cambiante -->
{% block content %}
{%
endblock %}
```

6- Ahora, eliminaremos gran parte de la plantilla “**“inicio”**”, dejando solo el encabezado y el mensaje. Renombraremos la plantilla a “**“base”**” y la utilizaremos como plantilla principal, ya que contiene gran parte del código que heredarán las demás secciones. Además, cargaremos todos los archivos necesarios desde la carpeta “**“static”**” y abriremos el bloque de contenido en el código restante.

```
ProyectoSebas > ProyectoSebasApp > templates > ProyectoSebasApp > inicio.html > section.page-section.clearfix > div.container > div.intro
1  {% extends "ProyectoSebasApp/base.html" %}
2
3  {% load static %}
4
5  {% block content %}
6
7  |<!-- Heading -->
```

7- Ahora conectaremos todas las plantillas de la barra de navegación a nuestra página de inicio, agregando sus respectivas URLs (de su respectivo archivo) en la plantilla “**“base”**”.

```
ass="navbar-brand text-uppercase text-expanded font-weight-bold d-lg-none" href="{% url 'Inicio' %}">>
on class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-c
an class="navbar-toggler-icon"></span>
ton>
class="collapse navbar-collapse" id="navbarResponsive">
class="navbar-nav mx-auto">
li class="nav-item active px-lg-4">
<a class="nav-link text-uppercase text-expanded" href="{% url 'Inicio' %}">Inicio</a>
/li>
li class="nav-item px-lg-4">
<a class="nav-link text-uppercase text-expanded" href="{% url 'Servicios' %}">Servicios</a>
/li>
li class="nav-item px-lg-4">
<a class="nav-link text-uppercase text-expanded" href="{% url 'Tienda' %}">Tienda</a>
/li>
li class="nav-item px-lg-4">
<a class="nav-link text-uppercase text-expanded" href="{% url 'Contacto' %}">Contacto</a>
/li>
li class="nav-item px-lg-4">
<a class="nav-link text-uppercase text-expanded" href="{% url 'Blog' %}">Blog</a>
/
```

8- Ahora copiaremos y pegaremos el código que heredarán las demás secciones (servicios, tienda, contacto y blog) de la página principal.

```

blog.html x
ProjetoSebas > ProjetoSebasApp > templates > ProjetoSebas
1   <html>
2     <body>
3       Blog
4     </body>
5   </html>

blog.html x
ProjetoSebas > ProjetoSebasApp > templates > ProjetoSebas
1   {% extends "ProjetoSebasApp/base.html" %}
2
3   {% load static %}
4
5   {% block content %}
6
7   <div><p></p></div>
8
9   {% endblock %}

```

9-. Podremos ver los resultados una vez que le demos clic a cada sección.



Nota: Para resaltar cada una de las secciones, agregaremos la siguiente etiqueta en el menú de la plantilla “base”.

```

<li class="nav-item {% if request.path == '/' %} active {% endif %} px-lg-4">
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Inicio' %}">Inicio</a>
</li>
<li class="nav-item {% if request.path == '/servicios' %} active {% endif %} px-lg-4">
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Servicios' %}">Servicios</a>
</li>
<li class="nav-item {% if request.path == '/tienda' %} active {% endif %} px-lg-4">
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Tienda' %}">Tienda</a>
</li>
<li class="nav-item {% if request.path == '/contacto' %} active {% endif %} px-lg-4">
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Contacto' %}">Contacto</a>
</li>
<li class="nav-item {% if request.path == '/blog' %} active {% endif %} px-lg-4">
  <a class="nav-link text-uppercase text-expanded" href="{% url 'Blog' %}">Blog</a>
</li>

```

Podremos ver los resultados una vez que le demos clic a cada sección.



Parte 4: Creación de la aplicación “servicios”.

1-. Creamos una segunda app para la parte de servicios.

```
● PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py startapp servicios
```

2-. Registraremos la app en el archivo “settings”.

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

3-. Buscaremos en la app de servicios el archivo “models.py” y empezaremos a construir los modelos correspondientes.

```
from django.db import models

# Create your models here.

class Servicio(models.Model):
    Titulo=models.CharField(max_length=50)
    Contenido=models.CharField(max_length=50)
    Imagen=models.ImageField()
    Created=models.DateTimeField(auto_now_add=True)
    Updated=models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name='servicio'
        verbose_name_plural='servicios'

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

Nota: Esta clase Servicio define un modelo para almacenar información sobre servicios, incluyendo el título, contenido, imagen, y las fechas de creación y actualización. Los metadatos y el método `__str__` proporcionan una mejor legibilidad y gestión en la administración de Django.

Nota: Cuando miremos en el panel de administración, veremos la palabra "servicio" para un solo elemento y "servicios" para varios.

Nota: Para poder manipular imágenes necesitamos de un comando especial.

```
● PS C:\Users\52556\Desktop\ Django\ProyectoSebas> pip install Pillow
Requirement already satisfied: Pillow in c:\users\52556\appdata\local\programs\python\python312\lib\site-packages (10.3.0)

[notice] A new release of pip is available: 24.0 -> 24.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

4-. Ahora realizaremos las migraciones correspondientes.

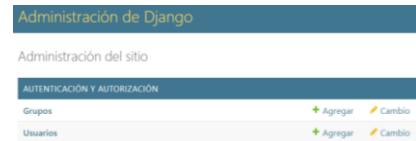
```
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py makemigrations
● Migrations for 'servicios':
  servicios\migrations\0001_initial.py
    - Create model Servicio
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py migrate
● Operations to perform:
  Apply all migrations: admin, auth, contenttypes, servicios, sessions
  Running migrations:
    Applying servicios.0001_initial... OK
○ PS C:\Users\52556\Desktop\ Django\ProyectoSebas>
```

5-. Podremos visualizar en DB Browser los modelos que vayamos creando.



6-. Crearemos un super usuario.

```
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py createsuperuser
• Username (leave blank to use '52556'): Sebas
Email address: sebastian.solis.mac@gmail.com
Password:
Password (again):
Superuser created successfully.
PS C:\Users\52556\Desktop\ Django\ProyectoSebas>
```



7-. Registraremos esta aplicación en el panel de administración mediante el archivo “admin.py” de nuestra aplicación.

```
admin.py •
ProyectoSebas > servicios > admin.py
1  v from django.contrib import admin
2  from servicios.models import Servicio
3
4  # Register your models here.
5
6  admin.site.register(Servicio)
```



8-. Si seleccionamos el respectivo apartado, podremos ver que tres de los cinco campos son visibles. Para hacer visibles los campos restantes (Created y Updated), modificaremos nuestro archivo anterior.

```
admin.py X
ProyectoSebas > servicios > admin.py > ...
1  from django.contrib import admin
2  from servicios.models import Servicio
3
4  # Register your models here.
5
6  class ServicioAdmin(admin.ModelAdmin):
7      readonly_fields=('Created','Updated')
8
9  admin.site.register(Servicio, ServicioAdmin)
```

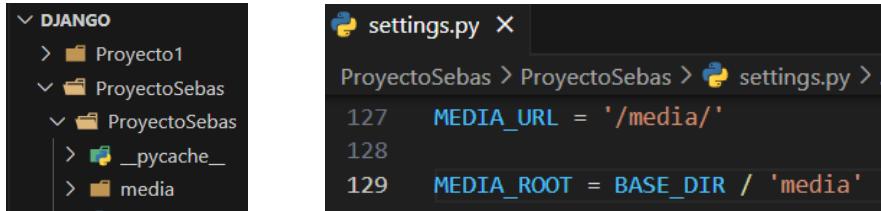
9-. Modificaremos el idioma del panel en el archivo “settings.py”.

```
LANGUAGE_CODE = 'es-es'
```

Nota: Al agregar un servicio, podremos notar que la imagen seleccionada (en el panel de administración) en su respectivo campo no se puede visualizar. Esto se debe a que Django, por defecto, no es capaz de mostrar imágenes u archivos multimedia debido a su configuración. La instrucción “DEBUG = True” (ubicada en el archivo “settings.py”) permite desarrollar el proyecto de esta manera. Una vez que hayamos terminado el desarrollo y el proyecto esté en el servidor, tendremos que cambiar esta instrucción a “DEBUG = False”.

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

10-. Por lo tanto, para organizar y visualizar el contenido multimedia, generaremos una carpeta llamada “media” (en donde crearemos una jerarquía de subcarpetas para guardar las imágenes de cada aplicación de manera ordenada) dentro del proyecto web. Luego, deberemos indicarle a Django, mediante el archivo “settings.py” (de la carpeta del proyecto general), que busque los archivos multimedia en esa carpeta específica.



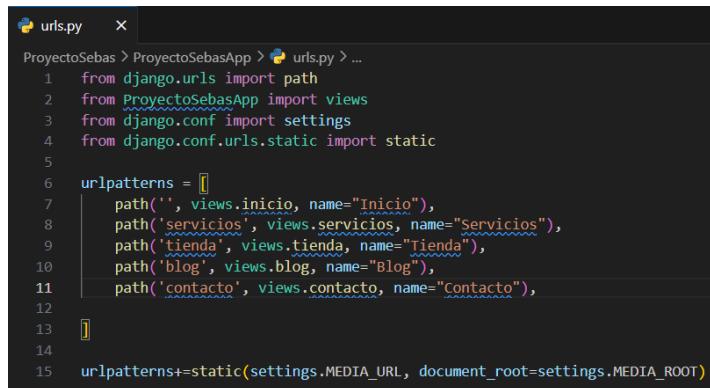
Entonces, nos dirigiremos al archivo “models.py” y, mediante un parámetro en la función “ImageField”, haremos que Django genere automáticamente una subcarpeta dentro de la carpeta “media” llamada “servicios”.

```

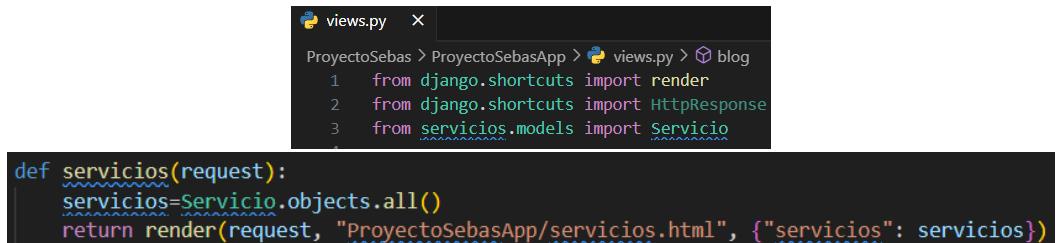
class Servicio(models.Model):
    Titulo=models.CharField(max_length=50)
    Contenido=models.CharField(max_length=50)
    Imagen=models.ImageField(upload_to='servicios')
    Created=models.DateTimeField(auto_now_add=True)
    Updated=models.DateTimeField(auto_now_add=True)

```

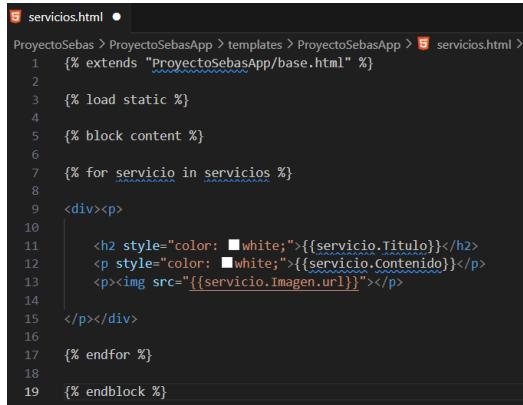
Por último, registraremos (en el archivo “urls.py” de la aplicación) las URLs de los archivos media (de esta manera podremos visualizar la imagen cuando le demos clic en el panel de administración).



11-. Ahora, importaremos desde el archivo “views.py” de la aplicación web, la clase que hemos creado para generar y mostrar los servicios, la cual está definida en el archivo “models.py”.



12-. Modificaremos el template correspondiente a los servicios para que muestre en pantalla los tres campos principales de la clase “Servicio” del panel de administración.

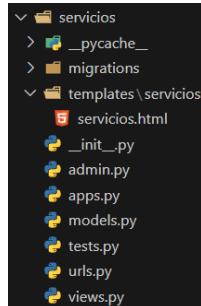


```
servicios.html
1  {% extends "ProyectoSebasApp/base.html" %} 
2
3  {% load static %} 
4
5  {% block content %} 
6
7  {% for servicio in servicios %} 
8
9    <div><p>
10      <h2 style="color: black;">{{servicio.Titulo}}</h2>
11      <p style="color: black;">{{servicio.Contenido}}</p>
12      <p></p>
13    </p></div>
14
15  {% endfor %} 
16
17  {% endblock %}
```

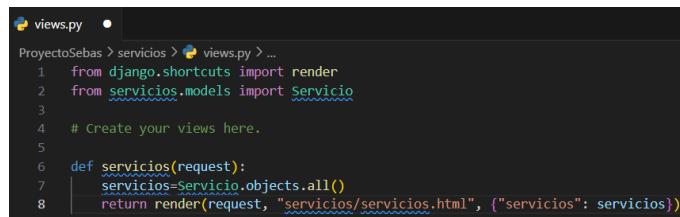
Nota: La línea 13 del código sirve para que la imagen (en la pantalla principal) se nos muestre sin generar ningún tipo de error.

13-. Ubicaremos y acomodaremos cada archivo y cada plantilla perteneciente a la sección servicios a la aplicación del mismo nombre.

Crearemos una carpeta llamada “templates” en la aplicación de servicios. Dentro de esta carpeta, crearemos una subcarpeta llamada “servicios” para trasladar el template con el mismo nombre que habíamos creado anteriormente.



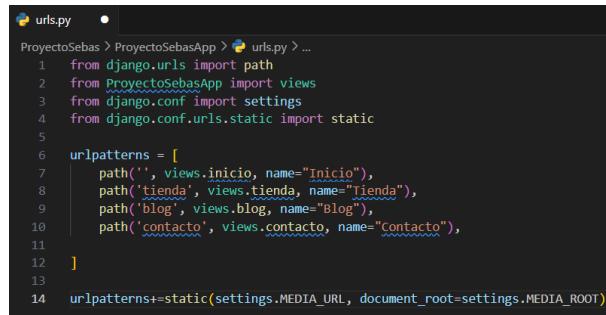
14-. También moveremos la vista de servicios (con su respectiva importación) al archivo “views” de la aplicación de servicios.



```
views.py
1  from django.shortcuts import render
2  from servicios.models import Servicio
3
4  # Create your views here.
5
6  def servicios(request):
7    servicios=Servicio.objects.all()
8    return render(request, "servicios/servicios.html", {"servicios": servicios})
```

15-. Modificaremos el archivo “urls.py” de nuestra aplicación principal y crearemos un archivo con el mismo nombre en la aplicación de “servicios”.

Aplicación principal:

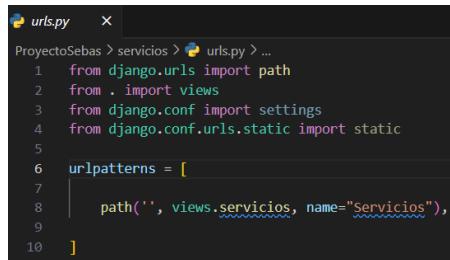


```

urls.py •
ProyectoSebas > ProyectoSebasApp > urls.py > ...
1  from django.urls import path
2  from ProyectoSebasApp import views
3  from django.conf import settings
4  from django.conf.urls.static import static
5
6  urlpatterns = [
7      path('', views.inicio, name="Inicio"),
8      path('tienda', views.tienda, name="Tienda"),
9      path('blog', views.blog, name="Blog"),
10     path('contacto', views.contacto, name="Contacto"),
11 ]
12 ]
13
14 urlpatterns+=static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

Servicios:

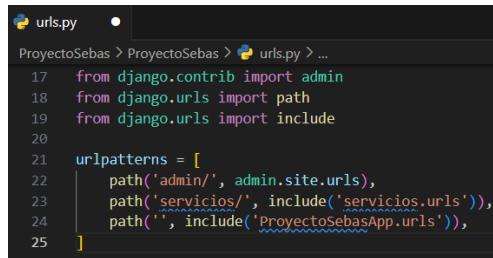


```

urls.py •
ProyectoSebas > servicios > urls.py > ...
1  from django.urls import path
2  from . import views
3  from django.conf import settings
4  from django.conf.urls.static import static
5
6  urlpatterns = [
7      path('', views.servicios, name="Servicios"),
8 ]
9
10 ]

```

16-. Registraremos la URL de nuestra aplicación de servicios en el archivo “urls.py” de “ProyectoSebas”.

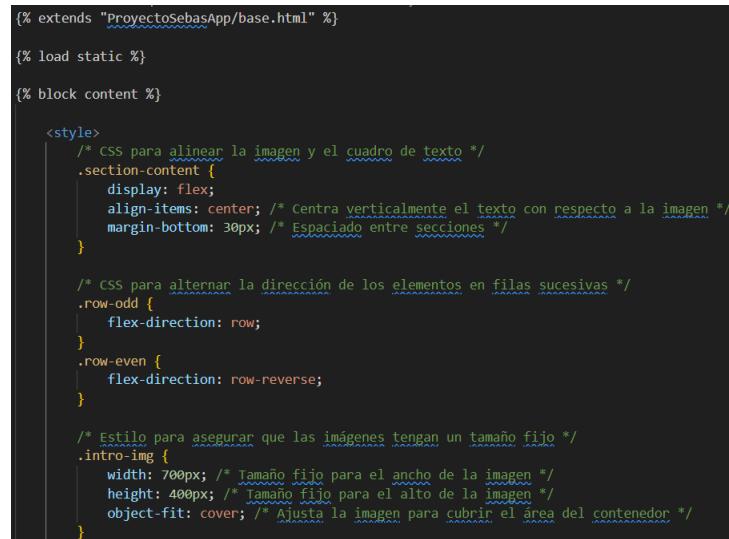


```

urls.py •
ProyectoSebas > ProyectoSebas > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path
19 from django.urls import include
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('servicios/', include('servicios.urls')),
24     path('', include('ProyectoSebasApp.urls')),
25 ]

```

17-. Una vez que hayamos creado y cargado los servicios, copiaremos el estilo CSS de la plantilla de “inicio” (la parte de heading) y lo aplicaremos a la plantilla de servicios con algunos ajustes.



```

{% extends "ProyectoSebasApp/base.html" %}
{% load static %}
{% block content %}

<style>
    /* CSS para alinear la imagen y el cuadro de texto */
    .section-content {
        display: flex;
        align-items: center; /* Centra verticalmente el texto con respecto a la imagen */
        margin-bottom: 30px; /* Espaciado entre secciones */
    }

    /* CSS para alternar la dirección de los elementos en filas sucesivas */
    .row-odd {
        flex-direction: row;
    }
    .row-even {
        flex-direction: row-reverse;
    }

    /* Estilo para asegurar que las imágenes tengan un tamaño fijo */
    .intro-img {
        width: 700px; /* Tamaño fijo para el ancho de la imagen */
        height: 400px; /* Tamaño fijo para el alto de la imagen */
        object-fit: cover; /* Ajusta la imagen para cubrir el área del contenedor */
    }

```

```

/* Estilo para el texto de fondo */
.intro-text {
    background-color: #rgba(24, 64, 8, 0.8);
    color: white;
    padding: 20px;
    border-radius: 8px;
    width: 100%; /* Asegura que el texto tenga el mismo ancho en la columna */
    box-sizing: border-box; /* Incluye el padding en el ancho total */
}

.intro-text h2 {
    margin: 0; /* Elimina márgenes para una alineación más precisa */
}

</style>

{% for servicio in servicios %}

<section class="page-section clearfix">
    <div class="container">
        <div class="section-content {% if forloop.counter|divisibleby:2 %}row-even{% else %}row-odd{% endif %}">
            <div class="col-md-6">
                
            </div>
            <div class="col-md-6">
                <div class="intro-text text-center p-5 rounded">
                    <h2 class="section-heading mb-4">
                        <span class="section-heading-lower">{{ servicio.titulo }}

```

Parte 5: Creación de la aplicación “blog”.

1-. Creamos una carpeta principal para el proyecto.

```
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py startapp blog
```

2-. Registraremos la app en el archivo “settings”.

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ProyectoSebasApp',
    'servicios',
    'blog',
]

```

3-. Buscaremos en la app del blog el archivo “models.py” y empezaremos a construir los modelos correspondientes (es decir, las categorías y el post).

```

models.py
ProyectoSebas > blog > models.py > Post > Meta
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  # Create your models here.
5
6  class Categoria(models.Model):
7      Nombre=models.CharField(max_length=50)
8      Created=models.DateTimeField(auto_now_add=True)
9      Updated=models.DateTimeField(auto_now_add=True)
10
11     class Meta:
12         verbose_name='categoria'
13         verbose_name_plural='categorias'
14
15     def __str__(self):
16         return self.Nombre

Post.py
class Post(models.Model):
    Titulo=models.CharField(max_length=50)
    Contenido=models.CharField(max_length=255)
    Imagen=models.ImageField(upload_to='blog', null=True, blank=True)
    Created=models.DateTimeField(auto_now_add=True)
    Updated=models.DateTimeField(auto_now_add=True)
    Autor=models.ForeignKey(User, on_delete=models.CASCADE)
    Categorias=models.ManyToManyField(Categoria)

    class Meta:
        verbose_name='post'
        verbose_name_plural='posts'

    def __str__(self):
        return self.Titulo

```

Nota: Este código define dos modelos de Django:

- **Categoría:** Representa una categoría de blog con un nombre y fechas de creación y actualización.
- **Post:** Representa un post de blog con un título, contenido, imagen opcional, fechas de creación y actualización, un autor (relacionado con el modelo User de Django) y múltiples categorías (relacionadas con el modelo Categoria).

Nota: En este caso, tanto el administrador como los usuarios podrán agregar diversos blogs en la página web. Si un usuario con permisos para crear posts es eliminado, también se eliminarán los posts que haya subido. Esto se debe a la forma en que están establecidas las relaciones entre las tablas.

Nota: Cabe mencionar que un post puede pertenecer a dos o más categorías, y una categoría puede incluir varios posts. Esto establece una relación de muchos a muchos entre posts y categorías.

4-. Ahora realizaremos las migraciones correspondientes.

5-. Podremos visualizar en DB Browser los modelos que vayamos creando.



6-. Registraremos esta aplicación en el panel de administración mediante el archivo “admin.py” de nuestra aplicación.

The diagram shows a file structure on the left and its corresponding Django Admin interface on the right. On the left, there is a code editor window titled 'admin.py' containing Python code for registering models. An arrow points from this to the right side, where a screenshot of the 'Administración de Django' interface is shown. The interface has a sidebar with 'SITIO ADMINISTRATIVO', 'AUTENTIFICACIÓN Y AUTORIZACIÓN', 'BLOG', and 'SERVICIOS'. Under 'BLOG', there are sections for 'Categorías' and 'Posts', each with 'Añadir' and 'Modificar' buttons.

```

    admin.py
    1  from django.contrib import admin
    2  from .models import Categoria
    3  from .models import Post
    4
    5  # Register your models here.
    6
    7  class CategoriaAdmin(admin.ModelAdmin):
    8      readonly_fields=('created','Updated')
    9
   10 class PostAdmin(admin.ModelAdmin):
   11     readonly_fields=('created','Updated')
   12
   13 admin.site.register(Categoria, CategoriaAdmin)
   14 admin.site.register(Post, PostAdmin)

```

7-. Crearemos en el panel de administración las categorías y los posts.

This diagram shows two screenshots of the Django Admin interface. The top screenshot is titled 'Añadir post' and includes fields for 'Título', 'Contenido', 'Imagen' (with a 'Elegir archivo' button), 'Autor' (with a dropdown and a '+' icon), and 'Categorías' (a dropdown menu listing categories like Electrónica, Ropa y Moda, Hogar y Cocina, etc.). The bottom screenshot is titled 'Añadir categoría' and includes fields for 'Nombre', 'Created', 'Updated', and three buttons at the bottom: 'GUARDAR', 'Guardar y añadir otro', and 'Guardar y continuar editando'. Both screenshots have a large black arrow pointing from the top to the bottom.

8-. Configuraremos las vistas para que esa información sea visible desde la web. Por lo que importaremos desde el archivo “views.py” de la aplicación “ProyectoSebasApp”, la función “blog” que hemos creado al archivo “views.py” de la aplicación del blog.

The diagram shows a code editor window with two versions of a 'views.py' file. A large black arrow points from the left version to the right version. The left version contains several view functions: 'inicio', 'tienda', 'blog', and 'contacto'. The right version has the same structure but with the 'blog' function renamed to 'blog'. Both versions also include imports for 'django.shortcuts' and 'render'.

```

    views.py
    1  from django.shortcuts import render
    2  from django.shortcuts import HttpResponseRedirect
    3
    4  # Create your views here.
    5
    6  def inicio(request):
    7      return render(request, "ProyectoSebasApp/inicio.html")
    8
    9  def tienda(request):
   10     return render(request, "ProyectoSebasApp/tienda.html")
   11
   12 def blog(request):
   13     return render(request, "ProyectoSebasApp/blog.html")
   14
   15 def contacto(request):
   16     return render(request, "ProyectoSebasApp/contacto.html")

```

```

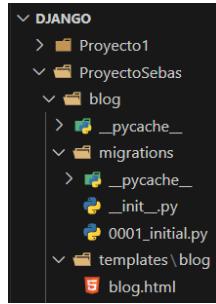
    views.py
    1  from django.shortcuts import render
    2
    3  # Create your views here.
    4
    5  def blog(request):
    6      return render(request, "blog/blog.html")

```

Nota: Cambiaremos el nombre de la URL en la línea 6 al nombre de la carpeta “blog”.

9-. Ubicaremos y acomodaremos cada archivo y cada plantilla perteneciente a la sección servicios a la aplicación del mismo nombre.

Crearemos una carpeta llamada “templates” en la aplicación blog. Dentro de esta carpeta, crearemos una subcarpeta llamada “blog” para trasladar y modificar el template con el mismo nombre que habíamos creado anteriormente.



Archivo “blog.html” modificado:

10.- Crearemos un archivo “urls.py” en la carpeta “blog”, donde transferiremos las líneas de código relacionadas con el tema del blog desde el archivo “urls.py” de la aplicación “ProyectoSebasApp”.

```

urls.py
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.blog, name="Blog"),
6  ]

```

Nota: Eliminaremos ese path (una vez cargado) en el archivo “urls.py” de la aplicación “ProyectoSebasApp”.

11.- Registraremos la URL de nuestra aplicación en el archivo “urls.py” de “ProyectoSebas”.

```

urls.py
17  from django.contrib import admin
18  from django.urls import path
19  from django.urls import include
20
21  urlpatterns = [
22      path('admin/', admin.site.urls),
23      path('servicios/' , include('servicios.urls')),
24      path('blog/' , include('blog.urls')),
25      path('' , include('ProyectoSebasApp.urls')),
26  ]

```

12.- Modificaremos la vista de blog de manera que podamos visualizar todos los posts del panel de administración.

```

views.py
1  from django.shortcuts import render
2  from blog.models import Post
3
4  # Create your views here.
5
6  def blog(request):
7      posts=Post.objects.all()
8      return render(request, "blog/blog.html", {"posts": posts})

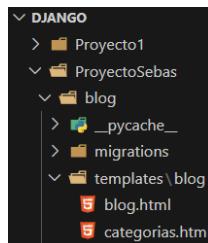
```

13.- Copiaremos el estilo CSS de la plantilla de “inicio” (la parte de heading) y lo aplicaremos a la plantilla de servicios con algunos ajustes.

Nota: Para que el nombre del autor aparezca en el cuadro de texto de la página web, crearemos una nueva etiqueta <div>.

Nota: Las categorías se especificarán al final del endfor y al comienzo del endblock.

14.- Crearemos una vista que permita ver los posts de las diferentes categorías (manualmente) ya filtrados. Para ello, copiaremos, pegaremos y modificaremos el documento “blog.html” en la carpeta “blog”, renombrándolo como “categorias.html”.



Archivo “categorias.html”:

```

    categorias.html
    ProyectoSebas > blog > templates > blog > categorias.html > section
    1  {% extends "ProyectoSebasApp/base.html" %}
    2
    3  {% load static %}
    4
    5  {% block content %}
    6
    7  <style>
    8      .intro-img {
    9          width: 100%;
   10         height: 400px;
   11         object-fit: cover;
   12     }
   13     .intro-text {
   14         background-color: #rgba(24, 64, 8, 0.8);
   15         color: white;
   16         padding: 20px;
   17         height: 400px;
   18         display: flex;
   19         flex-direction: column;
   20         justify-content: center;
   21     }
   22     .section-heading-lower {
   23         font-size: 2em;
   24     }
   25     .section-heading-upper {
   26         font-size: 1em;
   27         margin-top: 13px;
   28         text-transform: none;
   29     }

```

```

    .author-text {
        text-align: left;
        font-size: 0.8em;
    }
    .categories {
        width: 100%;
        text-align: center;
        color: white;
        margin-top: 20px;
    }
</style>

{% for post in posts %}
    <section class="page-section clearfix">
        <div class="container">
            <div class="row align-items-center">
                <div class="col-md-6">
                    
                </div>
                <div class="col-md-6">
                    <div class="intro-text text-center p-5 rounded">
                        <h2 class="section-heading mb-4">
                            <span class="section-heading-lower">{{ post.Titulo }}{{ post.Contenido }}

```

Luego, empezaremos a crear su respectiva vista en el archivo “views.py”.

Nota: El propósito es registrar la URL en la barra de navegación, basándonos en el ID de cada categoría (que hicimos en el panel) generado por la base de datos SQLite.

```

    views.py
    ProyectoSebas > blog > views.py > categoria
    1  from django.shortcuts import render
    2  from blog.models import Post
    3  from blog.models import Categoria
    4
    5  # Create your views here.
    6
    7  def blog(request):
    8      posts=Post.objects.all()
    9      return render(request, "blog/blog.html", {"posts": posts})
    10
    11 def categoria(request, categoria_id):
    12     categoria=Categoria.objects.get(id=categoria_id)
    13     return render(request, "blog/categorias.html", {"categorias": categoria, "posts": posts})

```

Nota: Con ese código podemos ver la categoría correspondiente, pero no los posts pertenecientes a cada una. Para lograrlo, agregamos una nueva línea con el método filter para filtrar los posts según el ID de la categoría.

```

    posts=Post.objects.filter(Categorias=categoria)
    return render(request, "blog/categorias.html", {"categorias": categoria, "posts": posts})

```

Después registraremos su url en la aplicación “blog”.

```

urls.py

ProyectoSebas > blog > urls.py > ...
1  from django.urls import path
2  from . import views
3  from django.conf import settings
4  from django.conf.urls.static import static
5
6  urlpatterns = [
7      path('', views.blog, name="Blog"),
8      path('categoria/<int:categoria_id>', views.categoria, name="Categoria"),
9
10 ]

```

15-. Una vez que hayamos probado la funcionalidad manualmente, procederemos a crear los enlaces para cada categoría, lo que facilitará la filtración de los posts correspondientes. Esto se logrará modificando la etiqueta <div> que agregamos anteriormente en la plantilla de blog.

```

<div class="categories">
    Categorías:
    {% for post in posts %}
        {% for categoria in post.Categorias.all %}
            <a href="{% url 'Categoria' categoria.id %}" class="linksCategorias"{{ categoria.Nombre }}></a>&ampnbsp&ampnbsp&ampnbsp
        {% endfor %}
    {% endfor %}
</div>
</section>
{% endblock %}

```

Para personalizar el color de los enlaces según nuestras preferencias, aplicaremos el estilo deseado en el archivo “gestión.css”.

```

gestion.css

ProyectoSebas > ProyectoSebasApp > static > Pro
281 .linksCategorias{
282     color: white !important;
283     text-decoration: underline;
284 }
285
286 .linksCategorias:hover{
287     text-decoration: none;
288 }

```

Nota: Despues de guardar, nos dirigiremos a la página web y presionaremos las teclas Ctrl + F5 para que los links carguen correctamente.

Nota: Para volver a “marcar notoriamente” cada sección de la página web tendremos que irnos a la plantilla “base.html” e introducir en cada “request.path” una diagonal al final del nombre de cada sección.

```

<li class="nav-item {% if request.path == '/' %} active {% endif %} px-lg-4">
    <a class="nav-link text-uppercase text-expanded" href="{% url 'inicio' %}>Inicio</a>
</li>
<li class="nav-item {% if request.path == '/servicios/' %} active {% endif %} px-lg-4">
    <a class="nav-link text-uppercase text-expanded" href="{% url 'Servicios' %}>Servicios</a>
</li>
<li class="nav-item {% if request.path == '/tienda/' %} active {% endif %} px-lg-4">
    <a class="nav-link text-uppercase text-expanded" href="{% url 'Tienda' %}>Tienda</a>
</li>
<li class="nav-item {% if request.path == '/contacto/' %} active {% endif %} px-lg-4">
    <a class="nav-link text-uppercase text-expanded" href="{% url 'Contacto' %}>Contacto</a>
</li>
<li class="nav-item {% if request.path[slice:-1] == '/blog/' %} active {% endif %} px-lg-4">
    <a class="nav-link text-uppercase text-expanded" href="{% url 'Blog' %}>Blog</a>
</li>

```

Nota: En el “request.path” de la sección blog, realizamos ajustes adicionales debido a que todas las URLs de categorías (un total de 9) están vinculadas a esta sección.

16-. Por último, modificaremos las vistas de la aplicación “blog” para que de esta forma nos salga una sola vez el nombre de cada categoría al momento de seleccionar la parte del blog en la página web.

```

def blog(request):
    posts = Post.objects.all()
    categorias = Categoria.objects.all().distinct()
    return render(request, "blog/blog.html", {"posts": posts, "categorias": categorias})

def categoria(request, categoria_id):
    categoria=Categoria.objects.get(id=categoria_id)
    posts=Post.objects.filter(categorias=categoria)
    return render(request, "blog/categorias.html", {"categoria": categoria, "posts": posts})

```

Parte 6: Creación de la aplicación “contacto”.

- 1-. Creamos una carpeta principal para el proyecto.

```
● PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py startapp contacto
```

- 2-. Registraremos la app en el archivo “settings”.

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ProyectoSebasApp',
    'servicios',
    'blog',
    'contacto',
]

```

- 3-. Importaremos desde el archivo “views.py” de la aplicación “ProyectoSebasApp”, la vista “contacto” que hemos creado, al archivo “views.py” de la aplicación contacto.



```

views.py  x | 
ProyectoSebas > ProyectoSebasApp > views.py > ...
1  from django.shortcuts import render
2  from django.shortcuts import HttpResponseRedirect
3
4  # Create your views here.
5
6  def inicio(request):
7      return render(request, "ProyectoSebasApp/inicio.html")
8
9  def tienda(request):
10     return render(request, "ProyectoSebasApp/tienda.html")
11
12 def contacto(request):
13     return render(request, "ProyectoSebasApp/contacto.html")

```



```

views.py  x | 
ProyectoSebas > contacto > views.py > ...
1  from django.shortcuts import render
2
3  # Create your views here.
4
5  def contacto(request):
6      return render(request, "contacto/contacto.html")

```

- 4-. Luego, crearemos un archivo “urls.py” en la carpeta “contacto”, donde transferiremos las líneas de código relacionadas con el tema del blog desde el archivo “urls.py” de la aplicación “ProyectoSebasApp”.

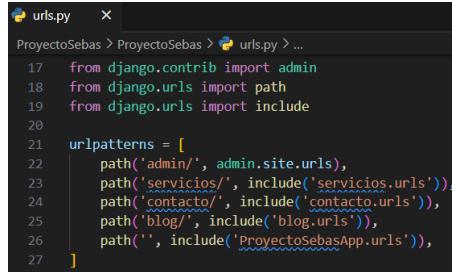
```

urls.py  x | 
ProyectoSebas > contacto > urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = []
5
6  path('', views.contacto, name="Contacto"),
7
8 ]

```

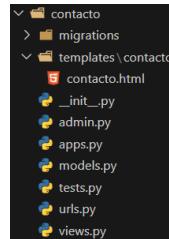
Nota: Eliminaremos ese path (una vez cargado) en el archivo “urls.py” de la aplicación “ProyectoSebasApp”.

5-. Una cosa que se me ha olvidado mencionar en las últimas secciones es la necesidad de incluir en el archivo “urls.py” del proyecto, los “path” de cada una de las aplicaciones.

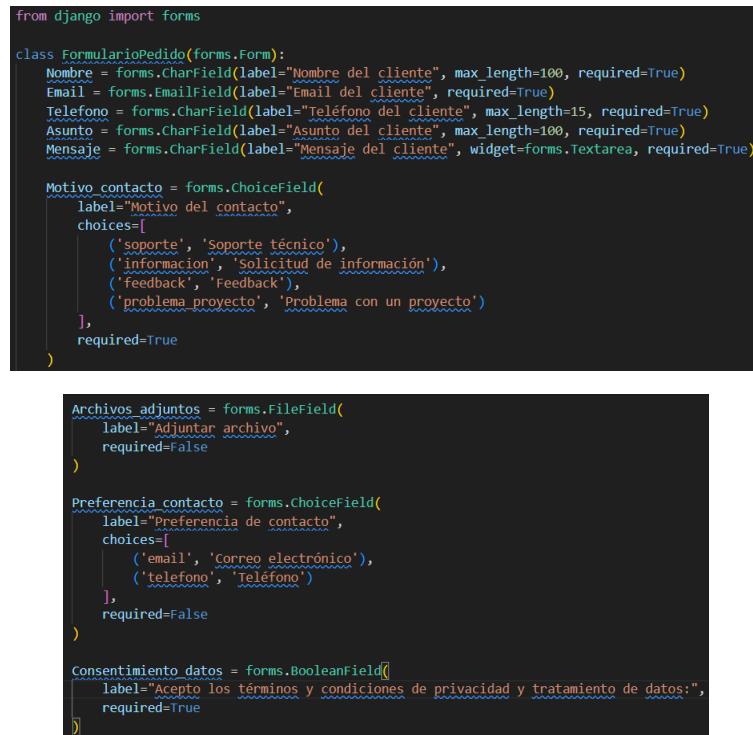


```
urls.py  x
ProyectoSebas > ProyectoSebas > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path
19 from django.urls import include
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('servicios/', include('servicios.urls')),
24     path('contacto/', include('contacto.urls')),
25     path('blog/', include('blog.urls')),
26     path('', include('ProyectoSebasApp.urls')),
27 ]
```

6-. Crearemos una carpeta llamada “templates” en la aplicación de contacto. Dentro de esta carpeta, crearemos una subcarpeta llamada “contacto” para trasladar el template con el mismo nombre que habíamos creado anteriormente.



7-. Dentro de la aplicación crearemos un archivo llamado “forms.py” para construir nuestro formulario con sus respectivos campos.



```
from django import forms

class FormularioPedido(forms.Form):
    Nombre = forms.CharField(label="Nombre del cliente", max_length=100, required=True)
    Email = forms.EmailField(label="Email del cliente", required=True)
    Telefono = forms.CharField(label="Teléfono del cliente", max_length=15, required=True)
    Asunto = forms.CharField(label="Asunto del cliente", max_length=100, required=True)
    Mensaje = forms.CharField(label="Mensaje del cliente", widget=forms.Textarea, required=True)

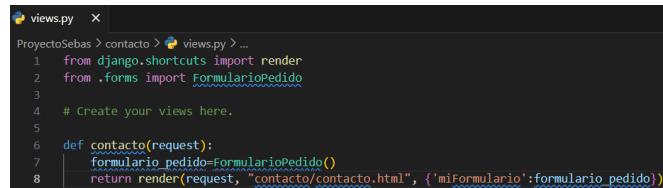
    Motivo_contacto = forms.ChoiceField(
        label="Motivo del contacto",
        choices=[
            ('soporte', 'Soporte técnico'),
            ('información', 'Solicitud de información'),
            ('feedback', 'Feedback'),
            ('problema proyecto', 'Problema con un proyecto')
        ],
        required=True
    )

    Archivos_adjuntos = forms.FileField(
        label="Adjuntar archivo",
        required=False
    )

    Preferencia_contacto = forms.ChoiceField(
        label="Preferencia de contacto",
        choices=[
            ('email', 'Correo electrónico'),
            ('telefono', 'Teléfono')
        ],
        required=False
    )

    Consentimiento_datos = forms.BooleanField(
        label="Acepto los términos y condiciones de privacidad y tratamiento de datos",
        required=True
    )
```

8-. Importaremos y registraremos el formulario en el archivo “vistas.py”.



```
views.py
1  from django.shortcuts import render
2  from .forms import FormularioPedido
3
4  # Create your views here.
5
6  def contacto(request):
7      formulario_pedido=FormularioPedido()
8      return render(request, "contacto/contacto.html", {'miFormulario':formulario_pedido})
```

9-. Nos dirigiremos al archivo HTML que registramos en pasos anteriores y colocaremos el nombre de la clave que pusimos en el diccionario de la instrucción return, dentro de la etiqueta “block content”.

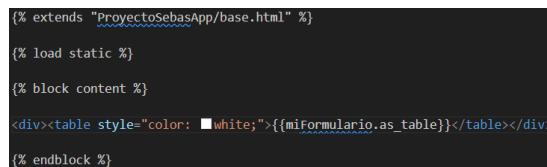


```
contacto.html
1  {% extends "ProyectoSebasApp/base.html" %}
2
3  {% load static %}
4
5  {% block content %}
6
7  

<p>{{miFormulario}}</p></div>
8
9  {% endblock %}


```

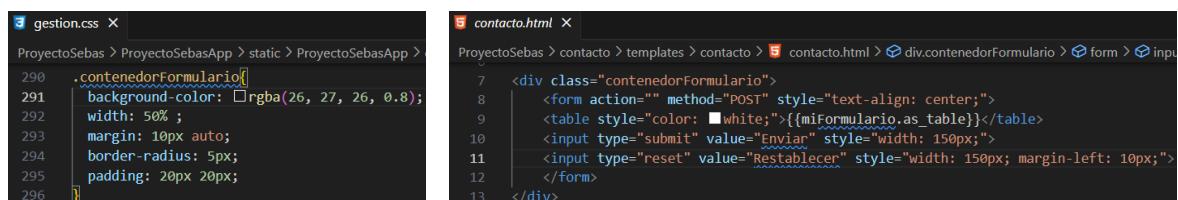
Nota: Le aplicaremos un formato de estructura y le pondremos un color a las palabras.



```
{% extends "ProyectoSebasApp/base.html" %}
{% load static %}
{% block content %}
<div><table style="color: black; border-collapse: collapse; width: 100%; border: none;">{{miFormulario.as_table}}</table></div>
{% endblock %}
```

10-. El siguiente paso es aplicar estilos CSS al formulario mediante el archivo “gestión.css.”

- Poner de un color determinado el fondo del formulario.
- Poner el formulario en el centro.
- Agregar una pequeña separación tanto arriba como abajo del formulario.
- Agregar bordes redondeados.
- Agregar un botón de enviar.
- Agregar un botón de restablecer.



```
gestion.css
1  .contenedorFormulario{
2      background-color: #f0f0f0;
3      width: 500px;
4      margin: 10px auto;
5      border-radius: 5px;
6      padding: 20px 20px;
7  }

contacto.html
1  <div class="contenedorFormulario">
2      <form action="" method="POST" style="text-align: center;">
3          <table style="color: black; border-collapse: collapse; width: 100%; border: none;">{{miFormulario.as_table}}</table>
4          <input type="submit" value="Enviar" style="width: 150px; height: 30px; border: 1px solid black; border-radius: 5px; font-size: 14px; margin-bottom: 10px;">
5          <input type="reset" value="Restablecer" style="width: 150px; height: 30px; border: 1px solid black; border-radius: 5px; font-size: 14px; margin-left: 10px;">
6      </form>
7  </div>
```

Resultado:

The screenshot shows a contact form with the following fields filled:

- Nombre del cliente: [redacted]
- Email del cliente: [redacted]
- Teléfono del cliente: [redacted]
- Asunto del cliente: [redacted]
- Mensaje del cliente: [large text area]
- Motivo del contacto: Soporte técnico
- Adjuntar archivo: Elejir archivo (No se ha seleccionado ningún archivo)
- Preferencia de contacto: Correo electrónico
- Acepto los términos y condiciones de privacidad y tratamiento de datos: [checkbox checked]

Buttons at the bottom: Enviar (Send) and Restablecer (Reset).

11.- Lo siguiente que haremos es agregar la validación correspondiente a la información de cada uno de los campos del formulario. De esta manera, aseguramos que al enviar la información no se pueda introducir contenido malicioso que podría ser utilizado para robar datos.

```
<form action="" method="POST" style="text-align: center;">
    {% csrf_token %}
    <table style="color: #white;">{{ miFormulario.as_table }}</table>
```

Nota: Asegúrate de que en tu formulario HTML se incluya enctype="multipart/form-data" para que los archivos (del campo “Archivos_adjuntos”) se envíen correctamente al servidor.

```
<form action="" method="POST" enctype="multipart/form-data" style="text-align: center;">
    {% csrf_token %}
```

12.- Para enviar correos electrónicos con Django, necesitamos configurar un servidor de correo y establecer los parámetros correspondientes en el archivo “settings.py”.

```
settings.py X
ProyectoSebas > ProyectoSebas > settings.py > ...
133 EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
134 EMAIL_HOST = "smtp.gmail.com"
135 EMAIL_PORT = 587
136 EMAIL_USE_TLS = True
137 EMAIL_HOST_USER = "sebastiansolvil@gmail.com"
138 EMAIL_HOST_PASSWORD = "koju Ibeo rysv fihp"
```

Nota: Para poder enviar correos electrónicos mediante Gmail, es necesario configurar nuestra cuenta para permitir el acceso de aplicaciones de terceros.

Nota: La cuarta línea indica el protocolo de seguridad que utiliza Google para enviar correos.

13.- Para ver la información que el usuario ha introducido en nuestro formulario, necesitamos utilizar el método POST en la vista de la aplicación.

Tambien configuraremos la vista para que la página web pueda enviar la información dada por el usuario, al correo solicitado.

```

views.py
1 from django.shortcuts import render, redirect
2 from django.core.mail import EmailMessage
3 from .forms import FormularioPedido
4
5 def contacto(request):
6     if request.method == "POST":
7         formulario_pedido = FormularioPedido(request.POST, request.FILES)
8         if formulario_pedido.is_valid():
9
10             campo1 = formulario_pedido.cleaned_data.get("Nombre")
11             campo2 = formulario_pedido.cleaned_data.get("Email")
12             campo3 = formulario_pedido.cleaned_data.get("Telefono")
13             campo4 = formulario_pedido.cleaned_data.get("Asunto")
14             campo5 = formulario_pedido.cleaned_data.get("Mensaje")
15             campo6 = formulario_pedido.cleaned_data.get("Motivo_contacto")
16             campo7 = formulario_pedido.cleaned_data.get("Archivos_adjuntos")
17             campo8 = formulario_pedido.cleaned_data.get("Preferencia_contactar")
18             campo9 = formulario_pedido.cleaned_data.get("Consentimiento_datos")
19
20             mensaje = f"""
21
22             Nombre: {campo1}
23             Email: {campo2}
24             Teléfono: {campo3}
25             Asunto: {campo4}
26             Mensaje: {campo5}
27             Motivo de Contacto: {campo6}
28             Preferencia de Contacto: {campo8}
29             Consentimiento de Datos: {'Si' if campo9 else 'No'}
30
31 """
32
33             email = EmailMessage(
34                 "Nuevo pedido recibido",
35                 mensaje,
36                 "",
37                 ["mac.sebas.acatlan@gmail.com"],
38                 reply_to=[campo2]
39             )

```

Implementaremos una condición if y un bucle for para gestionar los archivos adjuntos.

```

if request.FILES.get('Archivos_adjuntos'):
    archivos_adjuntos = request.FILES.getlist('Archivos_adjuntos')
    for archivo in archivos_adjuntos:
        email.attach(archivo.name, archivo.read(), archivo.content_type)

```

Ahora podemos utilizar una redirección para que, después de haber realizado todas las operaciones necesarias en los campos de la vista, se pase un parámetro en la recarga de la página web. Es decir, podemos configurar la redirección para que apunte a la URL de “contacto” y pase un parámetro adicional.

```

try:
    email.send()
    return redirect("/contacto/?valido")
except Exception as e:
    print(f"Error al enviar el correo: {e}")
    return redirect("/contacto/?novalido")
else:
    print(formulario_pedido.errors)
    return redirect("/contacto/?novalido")
else:
    formulario_pedido = FormularioPedido()

return render(request, "contacto/contacto.html", {'miFormulario': formulario_pedido})

```

En “contacto.html”, debemos configurar la página para que, cuando se cargue mediante una solicitud GET, se verifique si se ha recibido el parámetro “valido”. Si es así, significa que la página se está recargando después de enviar la información del formulario, y por lo tanto, debe mostrar un mensaje de retroalimentación informando al usuario que la información se ha enviado correctamente.

```

<p style="color: white;">Informacion enviada correctamente</p>
{% endif %}

```

Resultado:

Nombre del cliente: Sebastián Solis
 Email del cliente: mac.sebas.acatlan@gmail.com
 Teléfono del cliente: 5553001841
 Asunto del cliente: Duda sobre el feedback

Mensaje del cliente:

```
Recientemente completé una solicitud de soporte técnico y recibí un correo electrónico solicitando feedback sobre mi experiencia. Sin embargo, tengo algunas dudas sobre cómo se utilizará la información que proporciono. ¿Podrían aclarar cómo se manejarán mis comentarios y si permanecerán anónimos? Además, ¿hay algún beneficio en particular por proporcionar este feedback?
```

Motivo del contacto: Solicitud de información
 Adjuntar archivo: Elegir archivo feedback.jpg
 Preferencia de contacto: Teléfono
 Acepto los términos y condiciones de privacidad y tratamiento de datos: Sí

Enviar Restablecer

127.0.0.1:9000/contacto/valido
 Cursos en línea - en... Replit - Repls Test: How Sociable... Quizzes about scienc... Información enviada correctamente Nombre del cliente: Email del cliente:

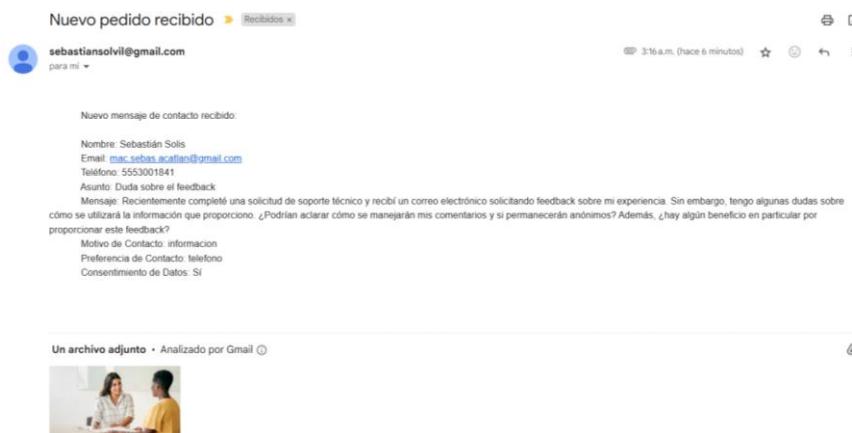
Nota: Para observar cómo funciona el método POST de manera sencilla, podemos ir a la plantilla de la aplicación y añadir la siguiente instrucción antes del “endblock”.



Cuando introduzcamos toda la información en los campos del formulario, el diccionario se llenará con un token y con la información proporcionada en cada uno de los campos.

14-. Ingresaremos toda la información en el formulario y le daremos al botón de enviar.

Al cliente le llegará el siguiente correo:



Parte 7: Creación de la aplicación “tienda”.

1-. Creamos una carpeta principal para el proyecto.

```
PS C:\Users\52556\Desktop\DJango\ProyectoSebas> python manage.py startapp tienda
```

2-. Registraremos la app en el archivo “settings”.

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ProyectoSebasApp',
    'servicios',
    'blog',
    'contacto',
    'tienda',
]

```

3-. Importaremos desde el archivo “views.py” de la aplicación “ProyectoSebasApp”, la vista “tienda” que hemos creado, al archivo “views.py” de la aplicación tienda.

```

views.py  x
ProyectoSebas > ProyectoSebasApp > views.py > inicio
1  from django.shortcuts import render
2  from django.shortcuts import HttpResponseRedirect
3
4  # Create your views here.
5
6  def inicio(request):
7      return render(request, "ProyectoSebasApp/inicio.html")
8
9  def tienda(request):
10     return render(request, "ProyectoSebasApp/tienda.html")

```



```

views.py  x
ProyectoSebas > tienda > views.py > ...
1  from django.shortcuts import render
2
3  # Create your views here.
4
5  def tienda(request):
6      return render(request, "tienda/tienda.html")

```

4-. Luego, crearemos un archivo “urls.py” en la carpeta “tienda”, donde transferiremos las líneas de código relacionadas con el tema de la tienda desde el archivo “urls.py” de la aplicación “ProyectoSebasApp”.

```

urls.py  x
ProyectoSebas > tienda > urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.tienda, name="Tienda"),
6
7 ]

```

Nota: Eliminaremos ese path (una vez cargado) en el archivo “urls.py” de la aplicación “ProyectoSebasApp”.

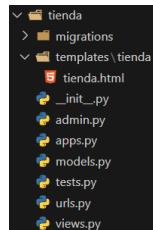
5-. Hay que incluir en el archivo “urls.py” del proyecto, los “path” de cada una de las aplicaciones.

```

urls.py  x
ProyectoSebas > ProyectoSebas > urls.py > ...
17  from django.contrib import admin
18  from django.urls import path
19  from django.urls import include
20
21  urlpatterns = [
22      path('admin/', admin.site.urls),
23      path('servicios/', include('servicios.urls')),
24      path('tienda/', include('tienda.urls')),
25      path('contacto/', include('contacto.urls')),
26      path('blog/', include('blog.urls')),
27      path('', include('ProyectoSebasApp.urls')),
28 ]

```

6-. Crearemos una carpeta llamada “templates” en la aplicación de tienda. Dentro de esta carpeta, crearemos una subcarpeta llamada “tienda” para trasladar el template con el mismo nombre que habíamos creado anteriormente.



7-. Buscaremos en la app de servicios el archivo “models.py” y empezaremos a construir los modelos correspondientes.

```

models.py X
ProyectoSebas > tienda > models.py > Producto > __str__
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  # Create your models here.
5
6  class Categoria(models.Model):
7      Nombre=models.CharField(max_length=80)
8      Created=models.DateTimeField(auto_now_add=True)
9      Updated=models.DateTimeField(auto_now_add=True)
10
11     class Meta:
12         verbose_name='categoria'
13         verbose_name_plural='categorias'
14
15     def __str__(self):
16         return self.Nombre

```

```

class Producto(models.Model):
    Nombre=models.CharField(max_length=90)
    Contenido=models.CharField(max_length=255)
    Imagen=models.ImageField(upload_to='tienda', null=True, blank=True)
    Precio=models.FloatField()
    Disponibilidad=models.BooleanField(default=True)
    Categorias=models.ForeignKey(Categoria, on_delete=models.CASCADE)
    Created=models.DateTimeField(auto_now_add=True)
    Updated=models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name='producto'
        verbose_name_plural='productos'

    def __str__(self):
        return self.Nombre

```

Nota: Para poder manipular imágenes necesitamos de un comando especial.

```

● PS C:\Users\52556\Desktop\ Django\ProyectoSebas> pip install Pillow
Requirement already satisfied: Pillow in c:\users\52556\appdata\local\programs\python\python312\lib\site-packages (10.3.0)

[notice] A new release of pip is available: 24.0 -> 24.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip

```

8-. Ahora realizaremos las migraciones correspondientes.

```

PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py makemigrations
● Migrations for 'servicios':
  services\migrations\0001_initial.py
    - Create model Servicio
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py migrate
● Operations to perform:
  Apply all migrations: admin, auth, contenttypes, servicios, sessions
  Running migrations:
    Applying servicios.0001_initial... OK
○ PS C:\Users\52556\Desktop\ Django\ProyectoSebas>

```

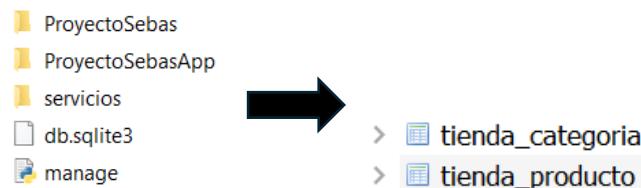
Nota: Si por alguna razón introducimos nuevos campos a un modelo, podremos encontrarnos con esta advertencia al momento de volver a realizar las migraciones (en este caso pondremos el número 1 para que Django nos dé un valor por defecto por cada registro, para después pulsar la tecla enter).

```

It is impossible to add the field 'Created' with 'auto_now_add=True' to producto without providing a default. This is because the database needs something to populate existing rows.
1) Provide a one-off default now which will be set on all existing rows
2) Quit and manually define a default value in models.py.
Select an option: 1

```

9-. Podremos visualizar en DB Browser los modelos que vayamos creando.



10-. Registraremos los modelos en el panel de administración mediante el archivo “admin.py” de nuestra aplicación.

```

admin.py x
ProyectoSebas > tienda > admin.py > ...
1  from django.contrib import admin
2  from .models import Categoria
3  from .models import Producto
4
5  # Register your models here.
6
7  class CategoriaAdmin(admin.ModelAdmin):
8      readonly_fields=('Created','Updated')
9
10 class ProductoAdmin(admin.ModelAdmin):
11     readonly_fields=('Created','Updated')
12
13 admin.site.register(Categoria, CategoriaAdmin)
14 admin.site.register(Producto, ProductoAdmin)

```

11-. Crearemos en el panel de administración las categorías y los productos y modificar el archivo “views” de la aplicación para que nos permita visualizar los productos en su respectiva sección en el HTML.

```

views.py x
ProyectoSebas > tienda > views.py > ...
1  from django.shortcuts import render
2  from .models import Producto
3
4  # Create your views here.
5
6  def tienda(request):
7      productos=Producto.objects.all()
8      return render(request, "tienda/tienda.html", {"productos":productos})

```

12-. Empezaremos a diseñar nuestra plantilla HTML de manera diferente para crear un entorno parecido a la de tiendas como Amazon, Ebay, etc.

- Se ajustan los márgenes, la altura, el ancho y la visualización de las imágenes de las tarjetas de productos, y se centra el texto dentro de las tarjetas.
- Las tarjetas se organizan en un contenedor con filas, asegurando que cada fila contenga hasta 4 tarjetas para mantener una disposición ordenada.
- Utiliza un bucle para recorrer y mostrar cada producto en una columna responsive, ajustándose a diferentes tamaños de pantalla.
- Cada producto se presenta en una tarjeta que incluye una imagen, el nombre, la descripción y el precio del producto.
- El precio de cada producto se muestra con dos decimales y un espacio antes del símbolo del dólar, asegurando claridad en la presentación.

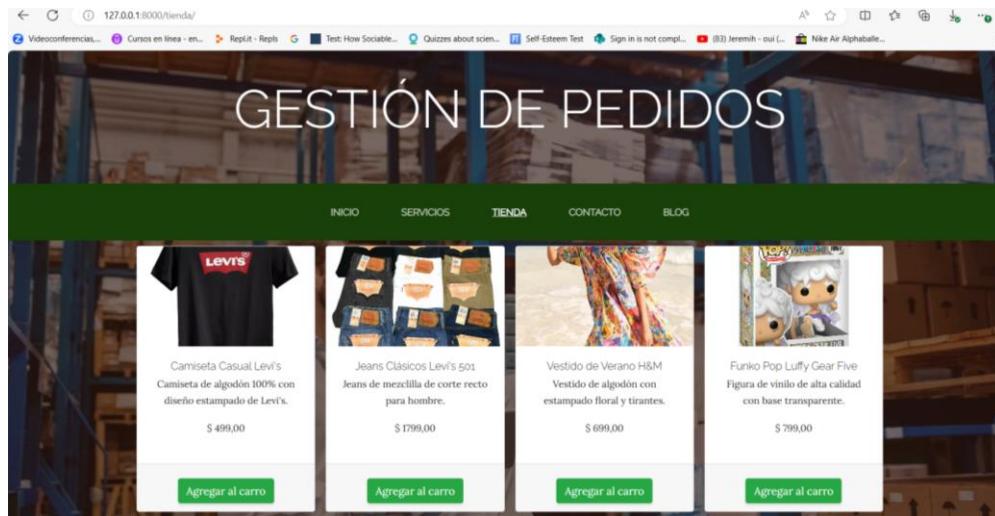
- Cada vez que se añade un cuarto producto a la fila, se cierra la fila actual y se inicia una nueva para asegurar que las tarjetas se distribuyan correctamente.
- En el pie de cada tarjeta, se incluye un botón con el texto "Aregar al carro" para permitir a los usuarios agregar el producto al carrito de compras.

```
% extends "ProyectoSebasApp/base.html" %}
% load static %

% block content %
<style>
    .card {
        margin-bottom: 20px;
        height: 400px;
    }
    .card img {
        width: 100%;
        height: 150px;
        object-fit: cover;
    }
    .card-body {
        text-align: center;
    }
    .card-title {
        font-size: 1em;
        margin-bottom: 5px;
    }
    .card-text {
        font-size: 0.9em;
    }
    .card-footer {
        text-align: center;
    }
    .btn-success {
        margin-top: 10px;
    }
</style>

<div class="container">
    <div class="row">
        <% for producto in productos %>
            <% if forloop.counter0|divisibleby:4 and not forloop.first %>
                </div><div class="row">
            <% endif %>
            <div class="col-md-4 col-lg-3">
                <div class="card">
                    
                    <div class="card-body">
                        <h5 class="card-title">{{ producto.Nombre }}
                        <p class="card-text">{{ producto.Contenido }}
                        <p class="card-text">S$ {{ producto.Precio|floatformat:2 }}
                    </div>
                    <div class="card-footer text-center">
                        <a href="#" class="btn btn-success">Aregar al carro</a>
                    </div>
                </div>
            <% endfor %>
        </div>
    <% endblock %>
</div>
```

Resultado:



Parte 8: Creación de la aplicación “carro”.

- 1-. Creamos una carpeta principal para el proyecto.

```
● PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py startapp carro
```

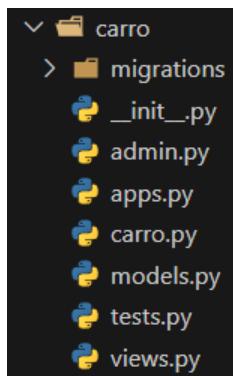
- 2-. Registraremos la app en el archivo “settings”.

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ProyectoSebasApp',
    'servicios',
    'blog',
    'contacto',
    'tienda',
    'carro',
]

```

3-. Crearemos un archivo llamado “carro.py” dentro de la carpeta con el mismo nombre.



4-. En ese mismo archivo crearemos las diversas funcionalidades que tendra el carro de la compra:

- Manejo de sesion
- Restar productos
- Agregar productos
- Vaciar el carro
- Eliminar productos

```

class Carro:
    def __init__(self, request):
        self.request = request
        self.session = request.session
        carro = self.session.get("carro")
        if not carro:
            self.carro = self.session["carro"] = {}
        else:
            self.carro = carro

    def agregar(self, producto):
        if str(producto.id) not in self.carro.keys():
            self.carro[str(producto.id)] = {
                "Producto_id": producto.id,
                "Nombre": producto.Nombre,
                "Precio": float(producto.Precio),
                "Cantidad": 1,
                "Imagen": producto.Imagen.url,
            }
        else:
            for key, value in self.carro.items():
                if key == str(producto.id):
                    value["Cantidad"] += 1
            break
        self.guardar_carro()

    def guardar_carro(self):
        self.session["carro"] = self.carro
        self.session.modified = True

    def eliminar(self, producto):
        producto.id = str(producto.id)
        if producto.id in self.carro:
            del self.carro[producto.id]
            self.guardar_carro()

    def restar(self, producto):
        for key, value in self.carro.items():
            if key == str(producto.id):
                value["Cantidad"] -= 1
                if value["Cantidad"] < 1:
                    self.eliminar(producto)
                break
        self.guardar_carro()

    def limpiar(self):
        self.session["carro"] = {}
        self.session.modified = True

```

5-. Crearemos las vistas importando la clase “Carro” con todos sus métodos, así como los productos. Además, implementaremos redirecciones, ya que cada vez que agreguemos,

eliminemos o vacíos un producto del carrito, será necesario redirigirnos nuevamente a la página de la tienda para reflejar los cambios realizados en el carrito.

```
from django.shortcuts import render
from .carro import Carro
from tienda.models import Producto
from django.shortcuts import redirect

# Create your views here.

def inicializar_carro(request):
    if "carro" not in request.session:
        request.session["carro"] = {}

def agregar_producto(request, producto_id):
    inicializar_carro(request)
    carro = Carro(request)
    producto = Producto.objects.get(id=producto_id)
    carro.agregar(producto=producto)
    return redirect("Tienda")

def eliminar_producto(request, producto_id):
    inicializar_carro(request)
    carro = Carro(request)
    producto = Producto.objects.get(id=producto_id)
    carro.eliminar(producto=producto)
    return redirect("Tienda")

def restar_producto(request, producto_id):
    inicializar_carro(request)
    carro = Carro(request)
    producto = Producto.objects.get(id=producto_id)
    carro.restar(producto=producto)
    return redirect("Tienda")

def limpiar_producto(request):
    inicializar_carro(request)
    carro = Carro(request)
    carro.limpiar()
    return redirect("Tienda")
```

6- A continuación procederemos a crear una variable global a través de un procesador de contexto que almacenará todos los productos agregados al carrito, permitiendo que esta información sea accesible desde cualquier parte de nuestro proyecto.

```
context_processors.py
ProyectoSebas > carro > context_processors.py > importe_total
1 def importe_total(request):
2     total = 0
3     if request.user.is_authenticated and "carro" in request.session:
4         for key, value in request.session["carro"].items():
5             total = total + float(value["Precio"]) * value["Cantidad"]
6     return {"importe_total": total}
```

A continuación, registraremos en la sección de “templates” del archivo “settings.py” (ubicado en la carpeta “ProyectoSebas”) tres enlaces para poder trabajar con DIRS y los procesadores de contexto.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'carro.context_processors.importe_total',
            ],
        },
    },
]
```

7-. Agregar funcionalidades al carrito:

- Daremos funcionalidad al carrito para que pueda añadir, eliminar y restar productos, así como vaciar el carrito por completo.
- Para esto, copiaremos, pegaremos y modificaremos el archivo “urls.py” de la aplicación “tienda” en la aplicación “carro”.

```

from django.urls import path
from . import views

app_name = "carro"

urlpatterns = [
    path("agregar/<int:producto_id>", views.agregar_producto, name="agregar"),
    path("eliminar/<int:producto_id>", views.eliminar_producto, name="eliminar"),
    path("restar/<int:producto_id>", views.restar_producto, name="restar"),
    path("limpiar/", views.limpiar_producto, name="limpiar"),
]

```

- Registraremos nuestras nuevas URLs en el archivo “urls.py” de la aplicación “ProyectoSebas”.

```

urls.py  ×

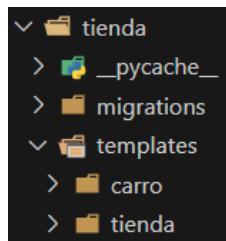
ProyectoSebas > ProyectoSebas > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path
19 from django.urls import include
20
21 urlpatterns = [
22
23     path('admin/', admin.site.urls),
24     path('servicios/', include('servicios.urls')),
25     path('tienda/', include('tienda.urls')),
26     path('contacto/', include('contacto.urls')),
27     path('blog/', include('blog.urls')),
28     path('', include('ProyectoSebasApp.urls')),
29     path('carro/', include('carro.urls')),
30
31 ]

```

- 8-. El siguiente paso será crear el widget del carrito. Para ello, debemos de colocar (en donde queramos) el widget del carrito en la plantilla de la tienda (yo lo pondre en la plantilla “base.html”).

Crear la estructura del widget:

- Crearemos una subcarpeta llamada “carro” dentro de la carpeta “templates” de la aplicación “tienda”.



- Dentro de esta subcarpeta, crearemos un archivo llamado “widget.html”, en el cual añadiremos una tabla con tres campos.
- Crearemos todo el código HTML.

```


| Carro de compra                                                                                                                                                                                                                                                                                                                                                  |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Producto                                                                                                                                                                                                                                                                                                                                                         | Cantidad | Suma |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| {% if request.session.carro.items %}                                                                                                                                                                                                                                                                                                                             |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| {% for key, value in request.session.carro.items %}                                                                                                                                                                                                                                                                                                              |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| <td>{% value.Nombre %}&lt;/td&gt; <td>{% value.Cantidad %}&lt;/td&gt; <td> <a class="btn btn-sm btn-custom-blue" href="{% url 'carro:agregar' value.Producto_id %}">&gt;+&lt;/a&gt; <a class="btn btn-sm btn-custom-white" href="{% url 'carro:restar' value.Producto_id %}">&gt;-&lt;/a&gt;&lt;br/&gt; \$&amp;nbsp;{{ value.Precio }}</a> </a></td> </td> </td> |          |      | {% value.Nombre %}</td> <td>{% value.Cantidad %}&lt;/td&gt; <td> <a class="btn btn-sm btn-custom-blue" href="{% url 'carro:agregar' value.Producto_id %}">&gt;+&lt;/a&gt; <a class="btn btn-sm btn-custom-white" href="{% url 'carro:restar' value.Producto_id %}">&gt;-&lt;/a&gt;&lt;br/&gt; \$&amp;nbsp;{{ value.Precio }}</a> </a></td> </td> | {% value.Cantidad %}</td> <td> <a class="btn btn-sm btn-custom-blue" href="{% url 'carro:agregar' value.Producto_id %}">&gt;+&lt;/a&gt; <a class="btn btn-sm btn-custom-white" href="{% url 'carro:restar' value.Producto_id %}">&gt;-&lt;/a&gt;&lt;br/&gt; \$&amp;nbsp;{{ value.Precio }}</a> </a></td> | <a class="btn btn-sm btn-custom-blue" href="{% url 'carro:agregar' value.Producto_id %}">&gt;+&lt;/a&gt; <a class="btn btn-sm btn-custom-white" href="{% url 'carro:restar' value.Producto_id %}">&gt;-&lt;/a&gt;&lt;br/&gt; \$&amp;nbsp;{{ value.Precio }}</a> </a> |
| {% endfor %}                                                                                                                                                                                                                                                                                                                                                     |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| {% else %}                                                                                                                                                                                                                                                                                                                                                       |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| <td colspan="3" style="text-align: center;">Sin productos</td>                                                                                                                                                                                                                                                                                                   |          |      | Sin productos                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| {% endif %}                                                                                                                                                                                                                                                                                                                                                      |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| Total: \$&nbsp;{{ importe_total }}                                                                                                                                                                                                                                                                                                                               |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| {% if request.session.carro.items %}                                                                                                                                                                                                                                                                                                                             |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| <a class="btn btn-success btn-custom-blue" href="{% url 'procesar_pedido' %}">Comprar</a>                                                                                                                                                                                                                                                                        |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |
| {% endif %}                                                                                                                                                                                                                                                                                                                                                      |          |      |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                      |


```

Parte 9: Creación de la aplicación “autenticación”.

- 1-. Creamos una carpeta principal para el proyecto.

```
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py startapp autenticacion
```

- 2-. Crearemos la vista de la aplicación en su archivo correspondiente (luego la modificaremos).

```

views.py  ×
ProyectoSebas > autenticacion > views.py > autenticacion
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def autenticacion(request):
6     pass

```

- 3-. A continuación, crearemos el archivo “urls.py” en la aplicación y registraremos su ruta.

```
urls.py  x
ProyectoSebas > autenticacion > urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.autenticacion, name="Autenticacion"),
6  ]
```

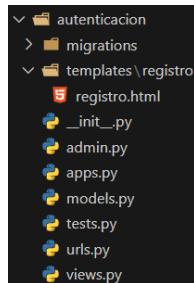
4-. Haremos lo mismo con el archivo “urls.py” de la carpeta general del proyecto.

```
urls.py  x
ProyectoSebas > ProyectoSebas > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path
19 from django.urls import include
20
21 urlpatterns = [
22
23     path('admin/', admin.site.urls),
24     path('servicios/', include('servicios.urls')),
25     path('tienda/', include('tienda.urls')),
26     path('contacto/', include('contacto.urls')),
27     path('blog/', include('blog.urls')),
28     path('', include('ProyectoSebasApp.urls')),
29     path('carro/', include('carro.urls')),
30     path('autenticacion/', include('autenticacion.urls')),
31
32 ]
```

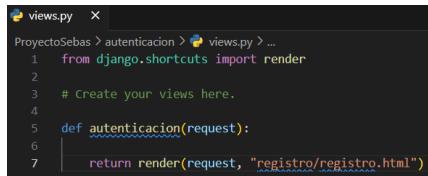
5-. Registraremos la app en el archivo “settings”.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ProyectoSebasApp',
    'servicios',
    'blog',
    'contacto',
    'tienda',
    'carro',
    'autenticacion',
]
```

6-. Crearemos una carpeta llamada “templates” en la aplicación. Dentro de esta carpeta, crearemos una subcarpeta llamada “registro” en donde crearemos el archivo llamado “registro.html”.



7-. Modificaremos la vista de la aplicación.

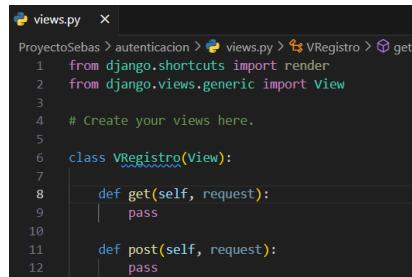


```
views.py x
ProyectoSebas > autenticacion > views.py > ...
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def autenticacion(request):
6     return render(request, "registro/registro.html")
```

8-. Arrancaremos el servidor e ingresaremos la URL para ver si funciona.

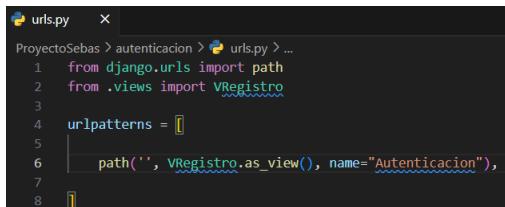
9-. Ahora crearemos en el HTML un formulario para registrar a los usuarios.

- Modificaremos la vista de la aplicación por segunda vez, creando una clase que gestionará el formulario con dos métodos (GET y POST).



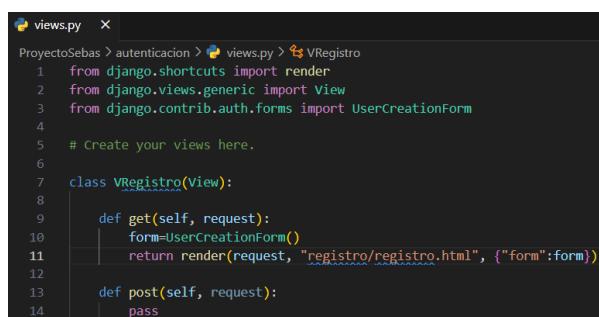
```
views.py x
ProyectoSebas > autenticacion > views.py > VRegistro > get
1 from django.shortcuts import render
2 from django.views.generic import View
3
4 # Create your views here.
5
6 class VRegistro(View):
7     def get(self, request):
8         pass
9
10    def post(self, request):
11        pass
```

- Al hacer este cambio, modificaremos el archivo “urls.py” de la aplicación.



```
urls.py x
ProyectoSebas > autenticacion > urls.py > ...
1 from django.urls import path
2 from .views import VRegistro
3
4 urlpatterns = [
5     path('', VRegistro.as_view(), name="Autenticacion"),
6 ]
7
8 ]
```

- Y al hacer este cambio, modificaremos la función GET debido a que es la encargada de mostrarnos nuestro formulario de la vista.



```
views.py x
ProyectoSebas > autenticacion > views.py > VRegistro
1 from django.shortcuts import render
2 from django.views.generic import View
3 from django.contrib.auth.forms import UserCreationForm
4
5 # Create your views here.
6
7 class VRegistro(View):
8     def get(self, request):
9         form=UserCreationForm()
10        return render(request, "registro/registro.html", {"form":form})
11
12    def post(self, request):
13        pass
```

Nota: “UserCreationForm” es una clase en Django que proporciona un formulario de registro de usuario listo para usar. Es parte del módulo “django.contrib.auth.forms” y hereda de “forms.ModelForm”. Su propósito principal es simplificar el proceso de creación de un nuevo usuario con validación y manejo de formularios incluidos.

- Llenaremos nuestro archivo HTML con el formulario que estamos enviando desde el método GET en la vista.

```

registro.html x
ProyectoSebas > autenticacion > templates > registro > registro.html > form
2
3   {% load static %}
4
5   {% block content %}
6
7     <form method="post" action="" style="color: black; background-color: white;">
8       |   {% csrf_token %}
9       |   {{form}}
10      |   </form>
11
12  {% endblock %}

```

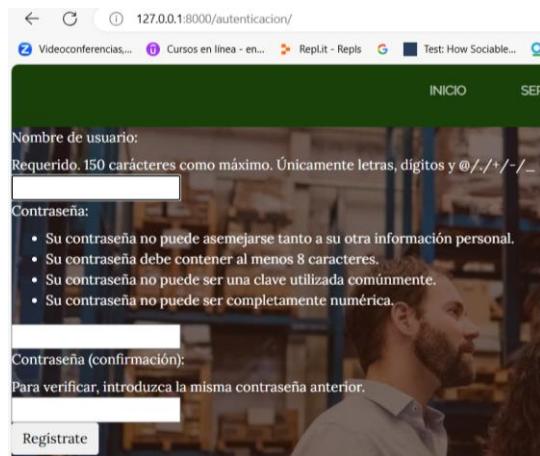
- Crearemos un botón de enviar en el archivo HTML.

```

{% csrf_token %}
{{form}}
<button type="submit" class="btn btn-sucess">Regístrate</button>

```

- Visualizaremos los resultados:



10-. Ahora le daremos un formato al formulario.

- Ejecutaremos el siguiente comando para instalar una versión del paquete “crispy”.

```
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> pip install django-crispy-forms==1.14.0
```

- Para utilizarlo, deberemos cargar ese paquete en el archivo HTML de la aplicación.

```

{% extends "ProyectoSebasApp/base.html" %}

{% load static %}

{% load crispy_forms_tags %}

{% block content %}

<form method="post" action="" style="color: black; background-color: white;">
|   {% csrf_token %}
|   {{form|crispy}}
|   <button type="submit" class="btn btn-sucess">Regístrate</button>
</form>

{% endblock %}

```

- Vamos a “settings.py” para registrar el paquete. Al final de este archivo, especificaremos que debe cargar el paquete en Bootstrap.

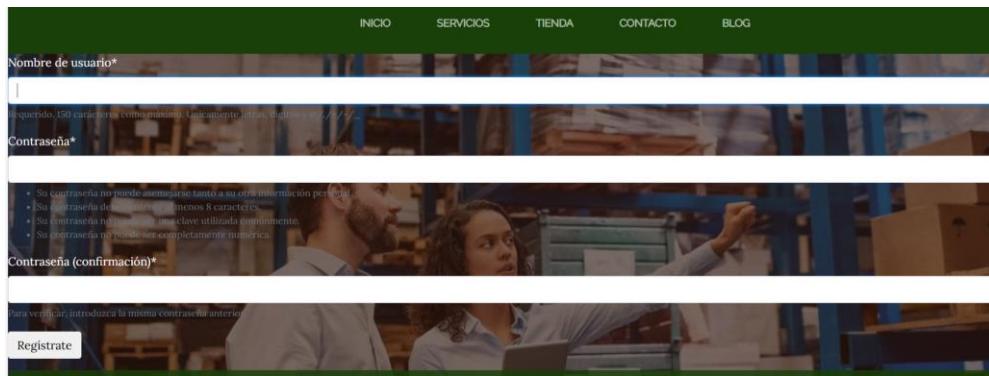
```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ProyectoSebasApp',
    'servicios',
    'blog',
    'contacto',
    'tienda',
    'carro',
    'autenticacion',
    'crispy_forms',
]

CRISPY_TEMPLATE_PACK = 'bootstrap5'

```

- Visualizaremos los resultados:



- 11-. Modificaremos el formato actual para añadirle algunos toques personales.

```

<div style ="width: 75%; margin: auto;">
{{form|crispy}}
<button type="submit" class="btn btn-sucess">Regístrate</button>
</div>

```

Nota: Para cambiar el color del texto en el formulario, abre el archivo “bootstrap.min.css” en Visual Studio Code, utiliza las funciones de búsqueda y reemplazo para localizar la clase “.text-muted”, y modifica el código de color dentro de esa clase para ajustar el color a tu preferencia.

The screenshot shows two instances of the Visual Studio Code search and replace interface. On the left, the search term ".text-muted" is highlighted in the search bar, and the replacement value "#343a40" is shown in the replace field. On the right, the result of the search is displayed, showing the original code with ".text-muted{color:#f8f9fa!important}" and the modified code with ".text-muted{color:#343a40!important}". A large black arrow points from the left interface to the right one, indicating the process of applying the changes.

- Visualizaremos los resultados:

Nombre de usuario*

Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @./+/-/_

Contraseña*

- Su contraseña no puede asemejarse tanto a su otra información personal.
- Su contraseña debe contener al menos 8 caracteres.
- Su contraseña no puede ser una clave utilizada comúnmente.
- Su contraseña no puede ser completamente numérica.

Contraseña (confirmación)*

Para verificar, introduzca la misma contraseña anterior.

Registre

12.- Después, el formulario deberá crear una cuenta de usuario. Es decir, el usuario se registrará en el formulario, su información se guardará en la base de datos y quedará autenticado automáticamente.

- Modificaremos el método POST de la vista y luego crearemos una variable que almacenará la información del formulario en la tabla SQLite llamada “auth_user”, logrando así un inicio de sesión automático y por último redireccionando la URL a la sección de “inicio”.

```
from django.shortcuts import render
from django.views.generic import View
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import login
from django.shortcuts import redirect

# Create your views here.

class VRegistro(View):

    def get(self, request):
        form = UserCreationForm()
        return render(request, "registro/registro.html", {"form": form})

    def post(self, request):
        form = UserCreationForm(request.POST)
        if form.is_valid():
            usuario = form.save()
            login(request, usuario)
            return redirect('Inicio')
        return render(request, "registro/registro.html", {"form": form})
```

- Visualizaremos los resultados al querer ingresar un usuario en nuestro formulario:

Nombre de usuario*

sebastian.solis.mac@gmail.com

Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @./+/-/_

Contraseña*

- Su contraseña no puede asemejarse tanto a su otra información personal.
- Su contraseña debe contener al menos 8 caracteres.
- Su contraseña no puede ser una clave utilizada comúnmente.
- Su contraseña no puede ser completamente numérica.

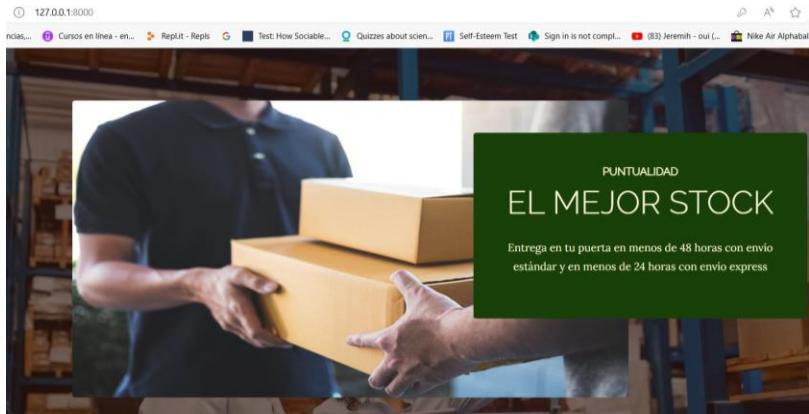
Contraseña (confirmación)*

Para verificar, introduzca la misma contraseña anterior.

Registre



Nos redirigió a “Inicio”



Podremos ver en DB Browser que nuestro usuario se registró con éxito.

DB Browser for SQLite - C:\Users\52556\Desktop\Django\ProyectoSebas\db.sqlite3				
File Edit View Tools Help New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database				
Database Structure Browse Data Edit Pragmas Execute SQL				
Table: auth_user				
id	password	last_login	is_superuser	username
1	pbkdf2_sha256\$720000\$2SEG7DAJ1c...	2024-07-19 02:00:10.978702	1	Sebas
2	pbkdf2_sha256\$720000\$WBySC66It...	2024-07-26 05:21:37.898579	0	sebastian.solis.mac@gmail.com

Nota: Es importante tener en cuenta que, al intentar acceder al panel de administración, la página web mostrará un error si el usuario con el que iniciamos sesión no tiene privilegios de superusuario. Para acceder correctamente, necesitamos iniciar sesión con la cuenta de superusuario.

13.- Agregaremos enlaces o textos en la página para indicar que el usuario que está navegando por el sitio está logueado, así como un enlace típico que permita cerrar sesión en caso de que el usuario desee hacerlo (todo esto lo introduciremos al template “base.html”).

```
<body>
    <div style="display: flex; justify-content: space-between; align-items: center; margin: 20px;">
        <div style="text-align: left; margin-left: 100px; margin-top: 210px; color: black;">
            {% if user.is_authenticated %}
                {{user.username}} &nbsp; &nbsp; <a href="{% url 'cerrar_sesion' %}" style="color: black;">Cerrar sesión</a>
            {% else %}
                <a href="#" style="color: black;">Login</a> &nbsp; &nbsp; <a href="{% url 'Autenticacion' %}" style="color: black;">Registrate</a>
            {% endif %}
        </div>
        <h1 class="site-heading text-center text-white" style="flex-grow: 1; margin: 0;">
            | <span class="site-heading-lower">gestión de Pedidos</span>
        </h1>
        <div style="flex-shrink: 1;">
            | | {% include "Carro/widget.html" %}
        </div>
    </div>

```

- Para crear el botón de “cerrar sesión” tenemos que añadir una nueva vista.

```
from django.contrib.auth import logout
def cerrar_sesion(request):
    logout(request)
    return redirect('Inicio')
```

- Registraremos la URL en su respectivo archivo.

```

from django.urls import path
from .views import VRegistro
from .views import cerrar_sesion

urlpatterns = [
    path('', VRegistro.as_view(), name="Autenticacion"),
    path('cerrar_sesion', cerrar_sesion, name="cerrar_sesion"),
]

```

- Visualizaremos los resultados:

Nota: Hasta este punto, el “Login” no funciona.

The figure consists of three vertically stacked screenshots of a web application. The top screenshot shows the homepage with a banner image of a warehouse, a title 'GESTIÓN DE PEDIDOS', and a navigation bar with links for 'INICIO', 'SERVICIOS', 'TIENDA', 'CONTACTO', and 'BLOG'. To the right is a 'Carro de compra' (Shopping Cart) table with one row: 'Sin productos' (No products). The middle screenshot shows the same homepage but with a user input field for 'Nombre de usuario*' (Username*). The bottom screenshot shows the same homepage with the user 'MrBeast' logged in, as indicated by the name in the top right corner and the 'Cerrar sesión' (Logout) link.

- Crearemos la vista para el “Login”.

```

from django.contrib.auth.forms import AuthenticationForm

def logear(request):
    form=AuthenticationForm()
    return render(request, "login/login.html", {"form": form})

```

- Crearemos una subcarpeta llamada “login” en donde copiaremos, pegaremos y modificaremos el código de la plantilla “registro” y lo renombraremos como “login.html”.

```
{% extends "projectoSebasApp/base.html" %}

{% load static %}

{% load crispy_forms_tags %}

{% block content %}

<form method="post" action="" style="color: black; background-color: white;">
    {% csrf_token %}
    <div style="width: 75%; margin: auto;">
        {{ form|crispy}}
        <button type="submit" class="btn btn-sucess">Login</button>
    </div>
</form>

{% endblock %}
```

- Registraremos la URL en su respectivo archivo.

```
from .views import logear

urlpatterns = [
    path('', VRegistro.as_view(), name="Autenticacion"),
    path('cerrar_sesion', cerrar_sesion, name="cerrar_sesion"),
    path('logear', logear, name="logear"),
]
```

- Visualizaremos los resultados:



14.- El siguiente paso será validar el formulario.

```
from django.contrib.auth import authenticate

def logear(request):
    if request.method=="POST":
        form=AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            nombre_usuario=form.cleaned_data.get("username")
            contra=form.cleaned_data.get("password")
            usuario=authenticate(username=nombre_usuario, password=contra)
            if usuario is not None:
                login(request, usuario)
                return redirect('Inicio')
            else:
                return render(request, "login/login.html", {"form": form})
        else:
            return render(request, "login/login.html", {"form": form})
    form=AuthenticationForm()
    return render(request, "login/login.html", {"form": form})
```

Por último, enlazaremos el texto de “Login” para que nos redirija al formulario correspondiente.

```
<a href="{% url 'logear' %}" style="color: black; text-decoration: none; font-weight: bold;">Login</a>
```

Nota opcional: Si queremos personalizar nuestros mensajes de advertencia, podemos hacerlo mediante una importación en el archivo “settings.py”. Esto permitirá al usuario ver en qué falló al intentar iniciar sesión en el formulario y también nos permitirá utilizarlos en nuestras demás plantillas.

```
from pathlib import Path  
from django.contrib.messages import constants as mensajes_de_error
```

```
MESSAGE_TAGS={  
    mensajes_de_error.DEBUG: 'debug',  
    mensajes_de_error.INFO: 'info',  
    mensajes_de_error.SUCCESS: 'success',  
    mensajes_de_error.WARNING: 'warning',  
    mensajes_de_error.ERROR: 'danger',  
}
```

Después, abriremos el archivo “login.html” y construiremos un conjunto de condicionales para mostrar nuestros mensajes.

```
<div style="width: 75%; margin: auto;">  
    {% if messages %}  
        {% for mensaje in messages %}  
            {{mensaje}}  
        {% endfor %}  
        {% endif %}  
    {{form|crispy}}  
</div>
```

Nota opcional: Para que el carrito se muestre únicamente cuando estemos autenticados, debemos modificar la plantilla “base.html”.

```
<div style="display: flex; justify-content: space-between; align-items: center; margin: 10px;">  
    <div style="text-align: left; margin-left: -12px; margin-top: 210px; color: black; flex-shrink: 1;">  
        {% if user.is_authenticated %}  
            {{user.username}} &ampnbsp <a href="{% url 'cerrar_sesion' %}" style="color: black; text-decoration: none; font-weight: bold;">Cerrar sesión</a>  
        {% else %}  
            <a href="{% url 'logear' %}" style="color: black; text-decoration: none; font-weight: bold;">Login</a> &ampnbsp <a href="{% url 'Autenticacion' %}" style="color: black; text-decoration: none; font-weight: bold;">Registrate</a>  
        {% endif %}  
    </div>  
    <h1 class="site-heading text-center text-white" style="margin: 0; flex-grow: 1; text-align: center;">  
        <span class="site-heading-lower">Gestión de Pedidos</span>  
    </h1>  
    <div style="flex-shrink: 1; text-align: right;">  
        {% if request.user.is_authenticated %}  
            {% include "carro/widget.html" %}  
        {% endif %}  
    </div>  
</div>
```

- Visualizaremos los resultados:





Parte 10: Creación de la aplicación “pedidos”.

1-. Creamos una carpeta principal para el proyecto.

```
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py startapp pedidos
```

2-. Registraremos la app en el archivo “settings”.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ProyectoSebasApp',
    'servicios',
    'blog',
    'contacto',
    'tienda',
    'carro',
    'autenticacion',
    'crispy_forms',
    'pedidos',
]
```

3-. Buscaremos en la app el archivo “models.py” y empezaremos a construir los modelos correspondientes (por ende, también realizaremos las migraciones correspondientes).

```

from django.db import models
from django.contrib.auth import get_user_model
from tienda.models import Producto
from django.db.models import F, Sum, FloatField

User = get_user_model()

class Pedido(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return str(self.id)

    @property
    def total(self):
        return self.lineapedido_set.aggregate(
            total=Sum(F("producto_Precio") * F("cantidad"), output_field=FloatField())
        )["total"]

    class Meta:
        db_table = 'pedidos'
        verbose_name = 'pedido'
        verbose_name_plural = 'pedidos'
        ordering = ['id']

class LineaPedido(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    producto = models.ForeignKey(Producto, on_delete=models.CASCADE)
    pedido = models.ForeignKey(Pedido, on_delete=models.CASCADE)
    cantidad = models.IntegerField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self.cantidad} unidades de {self.producto.Nombre}'

    class Meta:
        db_table = 'lineapedidos'
        verbose_name = 'Linea Pedido'
        verbose_name_plural = 'Lineas Pedidos'
        ordering = ['id']

```

```

pedidos\migrations\0001_initial.py
- Create model Pedido
- Create model LineaPedido
PS C:\Users\52556\Desktop\ Django\ProyectoSebas> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, blog, contenttypes, pedidos, servicios, sessions, tienda
Running migrations:
  Applying pedidos.0001_initial... OK

```

Nota: El código define dos modelos en Django para gestionar pedidos y líneas de pedidos en una tienda en línea.

Modelo Pedido: Representa un pedido realizado por un usuario.

- **user:** Una relación ForeignKey con el modelo de usuario.
- **created_at:** Marca la fecha y hora en que se creó el pedido.
- **__str__:** Retorna el ID del pedido como su representación en cadena.

- **total**: Una propiedad que calcula el total del pedido sumando el precio de cada producto multiplicado por la cantidad correspondiente.

Modelo LineaPedido: Representa una línea de un pedido, que incluye un producto y su cantidad.

- **user**: Una relación ForeignKey con el modelo de usuario.

- **producto_id**: Una relación ForeignKey con el modelo de producto.

- **pedido_id**: Una relación ForeignKey con el modelo de pedido.

- **cantidad**: Cantidad de producto en la línea del pedido.

- **created_at**: Marca la fecha y hora en que se creó la línea del pedido.

- **__str__**: Retorna una cadena que describe la cantidad de unidades y el nombre del producto.

Meta:

- **db_table**: Nombre de la tabla en la base de datos.

- **verbose_name** y **verbose_name_plural**: Nombres descriptivos del modelo.

- **ordering**: En el primer modelo ordena los pedidos por su ID y en el otro ordena las líneas de pedido por su ID.

4-. El siguiente paso será registrar las tablas “lineapedidos” y “pedidos” en el panel de administración.

```
from django.contrib import admin
from .models import Pedido
from .models import LineaPedido

# Register your models here.

admin.site.register([Pedido, LineaPedido])
```

5-. Agregaremos el botón de “comprar” en el archivo “base.html” para que aparezca cuando ya tengamos productos en nuestro carrito.

```
{% if request.session.carro.items %}
<tr>
  <td colspan="3" class="text-center">
    <a href="{% url 'procesar_pedido' %}" class="btn btn-success btn-custom-blue">
      Comprar
    </a>
  </td>
</tr>
{% endif %}
```

6-. A continuación, crearemos el archivo “urls.py” en la aplicación y registraremos su ruta.

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.procesar_pedido, name="procesar_pedido"),
```

7-. Haremos lo mismo con el archivo “urls.py” de la carpeta general del proyecto.

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('servicios/', include('servicios.urls')),
    path('tienda/', include('tienda.urls')),
    path('contacto/', include('contacto.urls')),
    path('blog/', include('blog.urls')),
    path('', include('ProyectoSebasApp.urls')),
    path('carro/', include('carro.urls')),
    path('autenticacion/', include('autenticacion.urls')),
    path('pedidos/', include('pedidos.urls')),
]
```

8-. Lo siguiente será crear un código (en la pestaña “views.py”) para procesar un pedido, creando registros de pedido y líneas de pedido en la base de datos, enviando un correo electrónico de confirmación al usuario y mostrando un mensaje de éxito antes de redirigir al usuario a la tienda.

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from pedidos.models import Pedido, LineaPedido
from carro.carro import Carro
from django.contrib import messages
from django.template.loader import render_to_string
from django.utils.html import strip_tags
from django.core.mail import send_mail
from django.conf import settings

@login_required(login_url="/autenticacion/logear")
def procesar_pedido(request):
    pedido = Pedido.objects.create(user=request.user)
    carro = Carro(request)
    lineas_pedido = []

    for key, value in carro.carro.items():
        lineas_pedido.append(LineaPedido(
            producto_id=key,
            cantidad=value["cantidad"],
            user=request.user,
            pedido=pedido
        ))
    LineaPedido.objects.bulk_create(lineas_pedido)

    enviar_mail(
        pedido=pedido,
        lineas_pedido=lineas_pedido,
        nombre_usuario=request.user.username,
        email_usuario=request.user.email
    )

    messages.success(request, "El pedido se ha creado correctamente")

    return redirect("../tienda")

def enviar_mail(**kwargs):
    asunto = "Gracias por el pedido"
    mensaje = render_to_string("emails/pedido.html", {
        "pedido": kwargs.get("pedido"),
        "lineas_pedido": kwargs.get("lineas_pedido"),
        "nombre_usuario": kwargs.get("nombre_usuario"),
        "email_usuario": kwargs.get("email_usuario")
    })

    mensaje_texto = strip_tags(mensaje)
    from_email = settings.EMAIL_HOST_USER
    to=kwargs.get("email_usuario")

    send_mail(asunto, mensaje_texto, from_email, [to], html_message=mensaje)
```

- **@login_required(login_url="/autenticacion/logear")**: Decorador que asegura que solo los usuarios autenticados puedan acceder a esta vista. Si no están autenticados, se redirige a la URL de inicio de sesión especificada.

- **def procesar_pedido(request)**: Define la vista procesar_pedido que maneja la creación y procesamiento de un pedido cuando un usuario autenticado lo solicita.

- **for key, value in carro.carro.items()**: Recorre los elementos en el carrito, donde key es el identificador del producto y value contiene los detalles del producto.

- **lineas_pedido.append(LineaPedido(...))**: Crea una instancia de LineaPedido por cada producto en el carrito y la añade a la lista lineas_pedido.

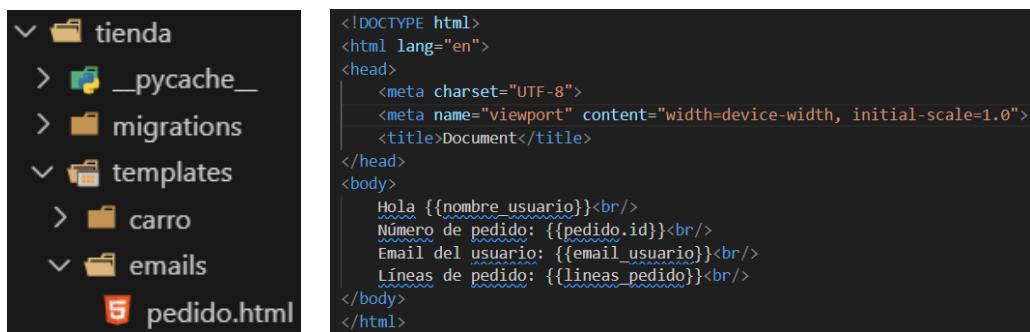
- **LineaPedido.objects.bulk_create(lineas_pedido)**: Guarda todas las instancias de LineaPedido en la base de datos de una sola vez para optimizar la operación.

- **enviar_mail(...)**: Llama a la función enviar_mail para enviar un correo electrónico de confirmación del pedido, pasando el pedido, las líneas de pedido, el nombre de usuario y el correo electrónico del usuario como argumentos.

- **messages.success(request, "El pedido se ha creado correctamente")**: Añade un mensaje de éxito que se mostrará al usuario.

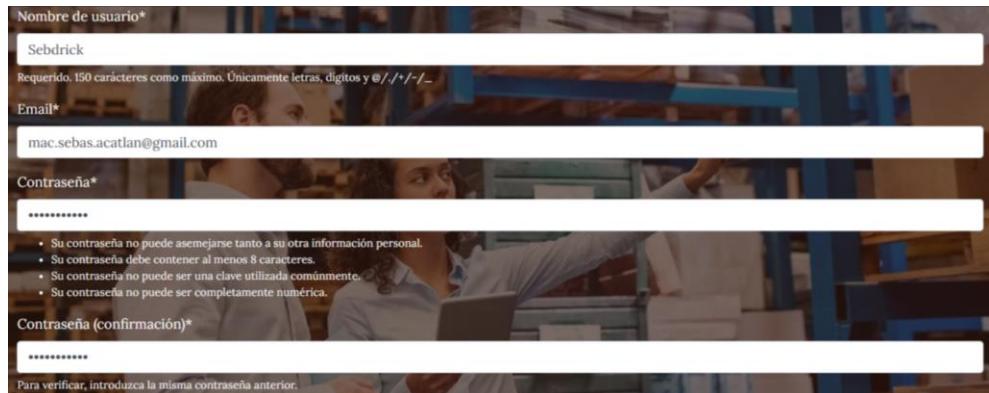
- **return redirect("../tienda")**: Redirige al usuario a la tienda.

9-. Crearemos el template llamado “emails/pedido.html” para vincularlo con el archivo “views.py” en la aplicación de la tienda.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    Hola {{nombre_usuario}}<br/>
    Número de pedido: {{pedido.id}}<br/>
    Email del usuario: {{email_usuario}}<br/>
    Líneas de pedido: {{lineas_pedido}}<br/>
</body>
</html>
```

- Visualizaremos los resultados:



Nombre de usuario*

Sebdrick

Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/-/_

Email*

mac.sebas.acatlan@gmail.com

Contraseña*

• Su contraseña no puede asemejarse tanto a su otra información personal.
• Su contraseña debe contener al menos 8 caracteres.
• Su contraseña no puede ser una clave utilizada comúnmente.
• Su contraseña no puede ser completamente numérica.

Contraseña (confirmación)*

Para verificar, introduzca la misma contraseña anterior.



Carro de compra		
Producto	Cantidad	Suma
Camiseta Casual Levi's	1	\$ 499,0
Videojuego Resident Evil Village	1	\$ 1500,0
		Total: \$ 1999,0

Comprar

Gracias por el pedido ➤ Recibidos x



sebastiansolvil@gmail.com

para mí ▾

Hola Sebdrick

Número de pedido: 12

Email del usuario: mac_sebas_acatlan@gmail.com

Líneas de pedido: [<LineaPedido: 1 unidades de Camiseta Casual Levi's>, <LineaPedido: 1 unidades de Videojuego Resident Evil Village>]

Nota: Para obtener este resultado, tuve que modificar el archivo “views.py” de la aplicación de autenticación añadiendo lo siguiente.

```
from django.shortcuts import render, redirect
from django.views.generic import View
from django.contrib.auth import login, logout, authenticate
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth.forms import UserCreationForm

class CustomUserCreationForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ("username", "email", "password1", "password2")

    def save(self, commit=True):
        user = super(CustomUserCreationForm, self).save(commit=False)
        user.email = self.cleaned_data["email"]
        if commit:
            user.save()
        return user

def cerrar_sesion(request):
    logout(request)
    return redirect('Inicio')
```