

Universidad Tecnológica de Tula-Tepeji

**Área de desarrollo de software
Multiplataforma**

“Reporte de Django”

Lic. Edel Meza Hernández

- **Juan Mateo Hernández De Luna**
- **Brian Emmanuel Flores Hernández**

5TIDSM-G1

Entorno y Configuración

¿Qué es Django?

Django es un framework web de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles. Desarrollado por programadores experimentados, Django se encarga de gran parte de las complicaciones del desarrollo web, por lo que puedes concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda.

Django es un marco web de código abierto de alto nivel escrito en Python que facilita el desarrollo de aplicaciones web rápidas y limpias.

¿Y cuál es su entorno de desarrollo?

Las principales herramientas que proporciona Django son un conjunto de scripts de Python para construir y trabajar con proyectos de Django, así como un servidor web de desarrollo simple que puede usar para realizar pruebas localmente (es decir, en su computadora, no en un servidor). Web externa) Aplicaciones web Django usando el navegador web de su computadora.

Pero, antes de comenzar, necesitas tener instalado Python en tu sistema. Luego, puedes instalar Django utilizando pip, el administrador de paquetes de Python.

Dentro del bash, agrega este comando:

1. Instalación de Python y pip (administrador de paquetes de Python).
2. Creación de un entorno virtual para aislar las dependencias del proyecto.
3. Instalación de Django usando pip.
4. Creación de un nuevo proyecto Django usando el comando `django-admin startproject proyectofinalcine``.

La configuración principal de Django se encuentra en el archivo `settings.py` dentro de cada proyecto, donde se especifican aspectos como la base de datos a utilizar, las aplicaciones instaladas, la configuración de autenticación, etc.

pip install django

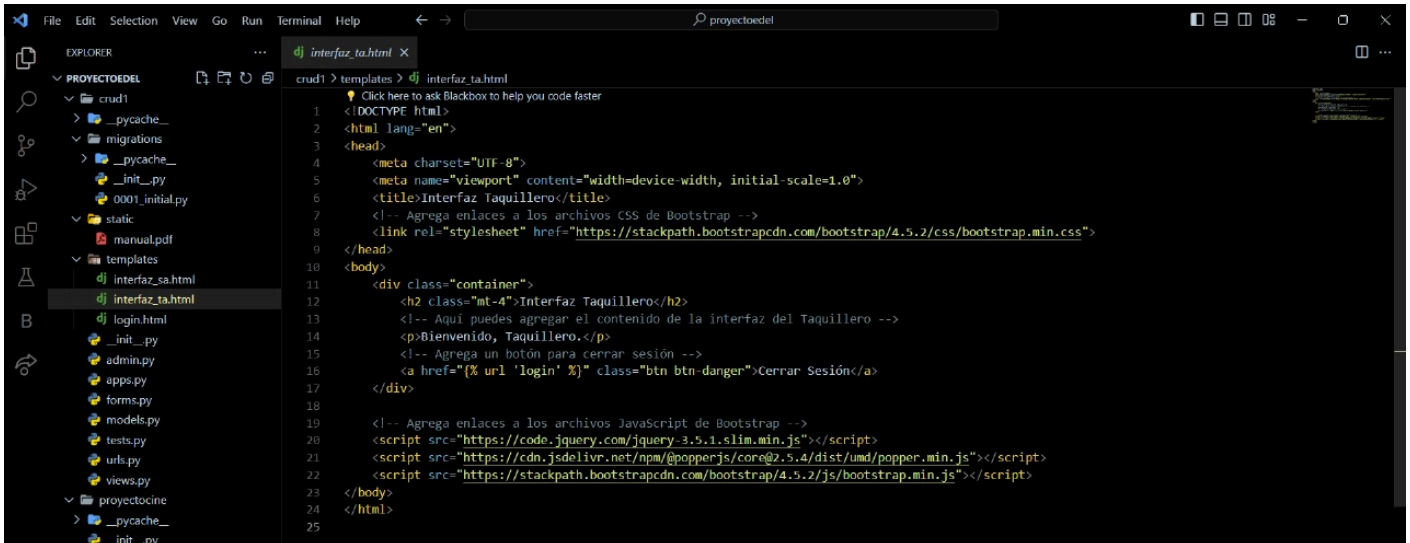
```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.3030]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\matej>pip install django
Collecting django
  Downloading Django-5.0.2-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref<4,>=3.7.0 (from django)
  Downloading asgiref-3.7.2-py3-none-any.whl.metadata (9.2 kB)
Collecting sqlparse>=0.3.1 (from django)
  Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
----- 41.2/41.2 kB 247.5 kB/s eta 0:00:00
Collecting tzdata (from django)
  Downloading tzdata-2023.4-py2.py3-none-any.whl.metadata (1.4 kB)
Download Django-5.0.2-py3-none-any.whl (8.2 MB)
----- 8.2/8.2 MB 7.7 MB/s eta 0:00:00
Download asgiref-3.7.2-py3-none-any.whl (24 kB)
Download tzdata-2023.4-py2.py3-none-any.whl (346 kB)
----- 346.6/346.6 kB 3.1 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.7.2 django-5.0.2 sqlparse-0.4.4 tzdata-2023.4

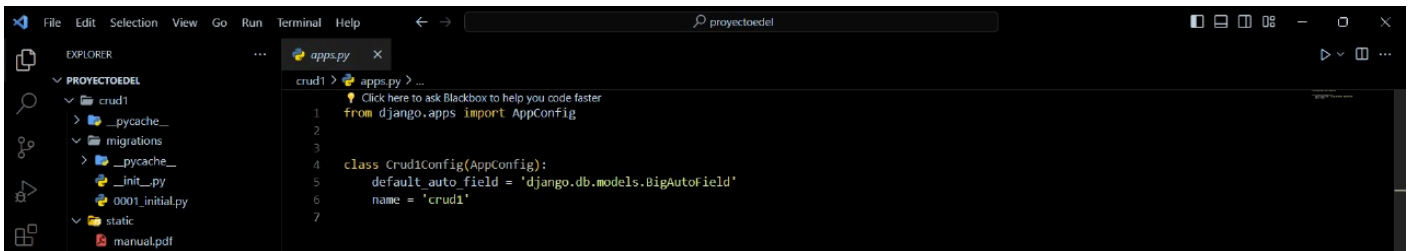
[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\matej>
```

Configuración Manual



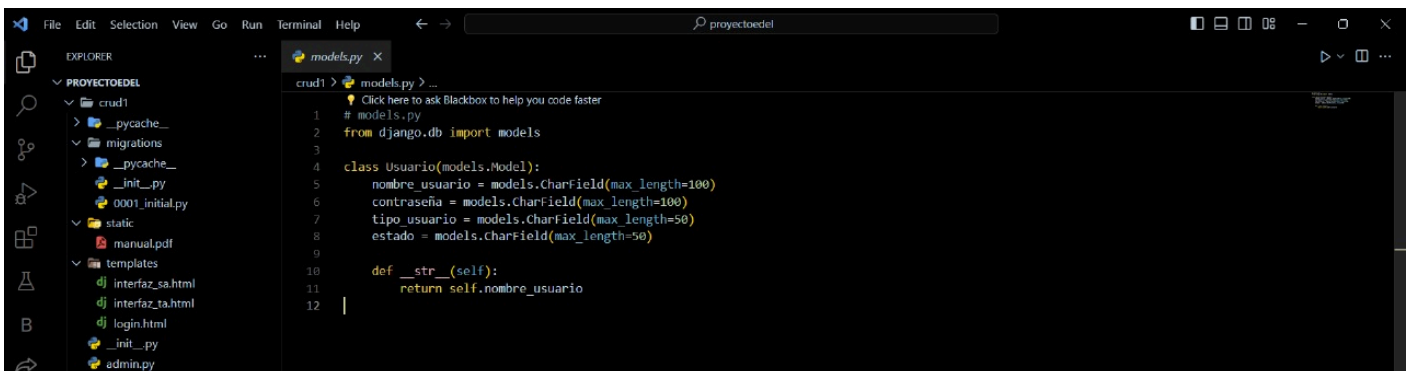
```
crud1 > templates > dj interfaz_ta.html
1  Click here to ask Blackbox to help you code faster
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5  <meta charset="UTF-8">
6  <meta name="viewport" content="width=device-width, initial-scale=1.0">
7  <title>Interfaz Taquillero</title>
8  <!-- Agrega enlaces a los archivos CSS de Bootstrap -->
9  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
10 </head>
11 <body>
12 <div class="container">
13 <h2 class="mt-4">Interfaz Taquillero</h2>
14 <!-- Aquí puedes agregar el contenido de la interfaz del Taquillero -->
15 <p>Bienvenido, Taquillero.</p>
16 <!-- Agrega un botón para cerrar sesión -->
17 <a href="{% url 'login' %}" class="btn btn-danger">Cerrar Sesión</a>
18 </div>
19 <!-- Agrega enlaces a los archivos JavaScript de Bootstrap -->
20 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
21 <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
22 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
23 </body>
24 </html>
25
```



```
crud1 > apps.py > ...
1  Click here to ask Blackbox to help you code faster
2  from django.apps import AppConfig
3
4  class Crud1Config(AppConfig):
5  default_auto_field = 'django.db.models.BigAutoField'
6  name = 'crud1'
7
```



```
crud1 > forms.py > UsuarioForm > Meta
1  Click here to ask Blackbox to help you code faster
2  # forms.py
3  from django import forms
4  from .models import Usuario
5
6  class UsuarioForm(forms.ModelForm):
7  class Meta:
8  model = Usuario
9  fields = ['nombre_usuario', 'contraseña', 'tipo_usuario', 'estado']
```



```
crud1 > models.py > ...
1  Click here to ask Blackbox to help you code faster
2  # models.py
3  from django.db import models
4
5  class Usuario(models.Model):
6  nombre_usuario = models.CharField(max_length=100)
7  contraseña = models.CharField(max_length=100)
8  tipo_usuario = models.CharField(max_length=50)
9  estado = models.CharField(max_length=50)
10
11 def __str__(self):
12 return self.nombre_usuario
```

The screenshot shows the VS Code editor with the 'views.py' file open. The file contains a Django view function named 'login' that handles user authentication. The code includes imports for 'render', 'redirect', 'UsuarioForm', and 'Usuario'. The 'login' function checks if the request method is 'POST', retrieves the username and password, and filters the 'Usuario' objects to find a matching user. It then checks if the password is correct and if the user is a 'taquillero' (ticket collector). If the user is a 'taquillero', it redirects to 'interfaz_ta.html'. Otherwise, it redirects to 'interfaz_sa.html'. If the password is incorrect, it displays an error message. The code also includes a comment about functionality before editing user data and a function 'interfaz_sa' that is partially visible.

```
1 # views.py
2 from django.shortcuts import render, redirect
3 from .forms import UsuarioForm
4 from .models import Usuario
5
6 def login(request):
7     if request.method == 'POST':
8         username = request.POST.get('user')
9         password = request.POST.get('password')
10
11         # Buscar un usuario con el nombre de usuario proporcionado en tu modelo Usuario
12         user = Usuario.objects.filter(nombre_usuario=username).first()
13
14         # Verificar si se encontró un usuario y si la contraseña es correcta
15         if user and user.contrasena == password:
16             # Si las credenciales son correctas, verificar el tipo de usuario
17             if user.tipo_usuario == 'taquillero':
18                 # Si el usuario es taquillero, redirigir a la interfaz ta
19                 request.session['user_id'] = user.id # Almacenar el ID del usuario en la sesión
20                 return redirect('interfaz_ta')
21             else:
22                 # Si el usuario es otro tipo, redirigir a la interfaz sa
23                 request.session['user_id'] = user.id # Almacenar el ID del usuario en la sesión
24                 return redirect('interfaz_sa')
25         else:
26             # Si las credenciales son incorrectas, mostrar un mensaje de error
27             error_message = "Nombre de usuario o contraseña incorrectos."
28             return render(request, 'login.html', {'error_message': error_message})
29     else:
30         # Si la solicitud no es POST, mostrar el formulario de inicio de sesión
31         return render(request, 'login.html')
32
33
34
35 #FUNCIONALIDAD ANTES DE EDITAR LOS DATOS DEL USUARIO
36 #def interfaz_sa(request):
37 #     if request.method == 'POST':
38 #         form = UsuarioForm(request.POST)
39 #         if form.is_valid():
40 #             # Crear el usuario
41 #             user = Usuario.objects.create(nombre_usuario=form.cleaned_data['nombre_usuario'],
42 #                                           contrasena=form.cleaned_data['contrasena'],
43 #                                           tipo_usuario=form.cleaned_data['tipo_usuario'])
44 #             return redirect('interfaz_sa')
```

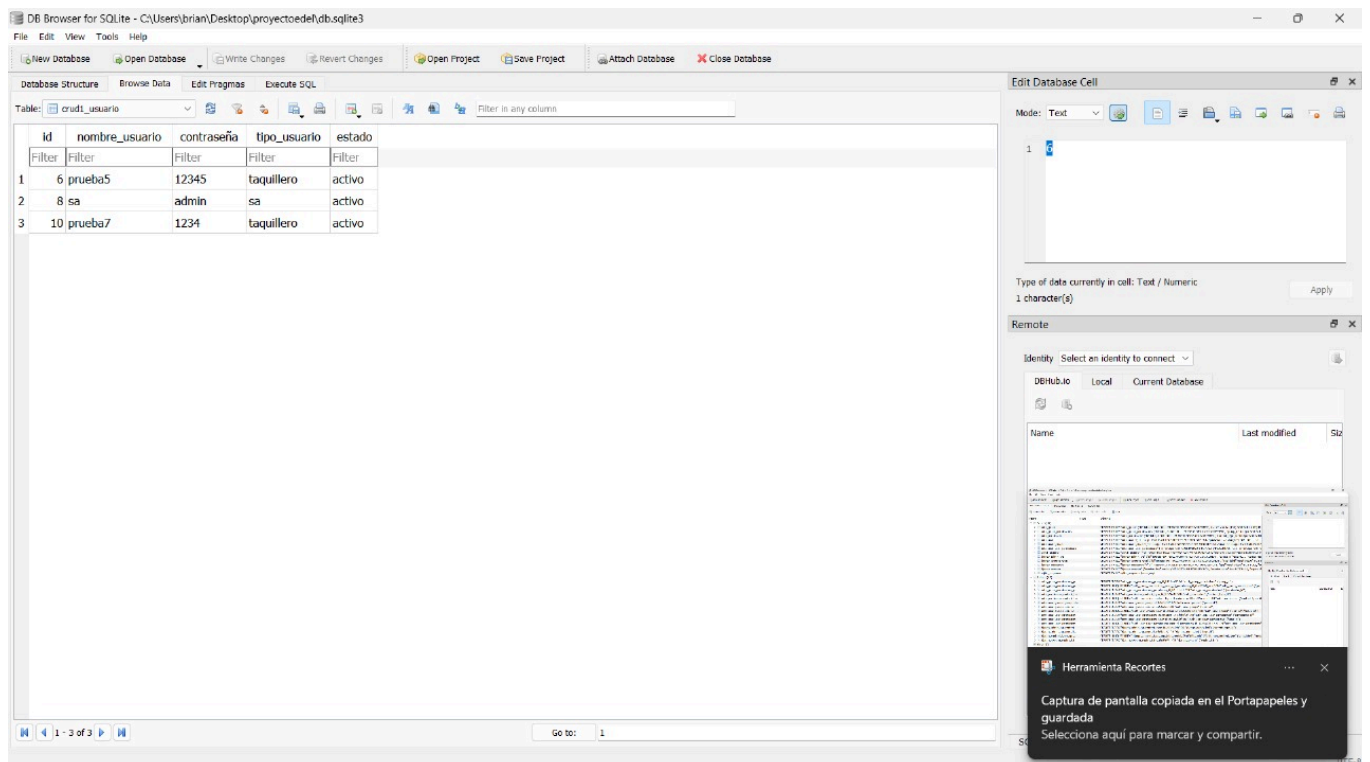
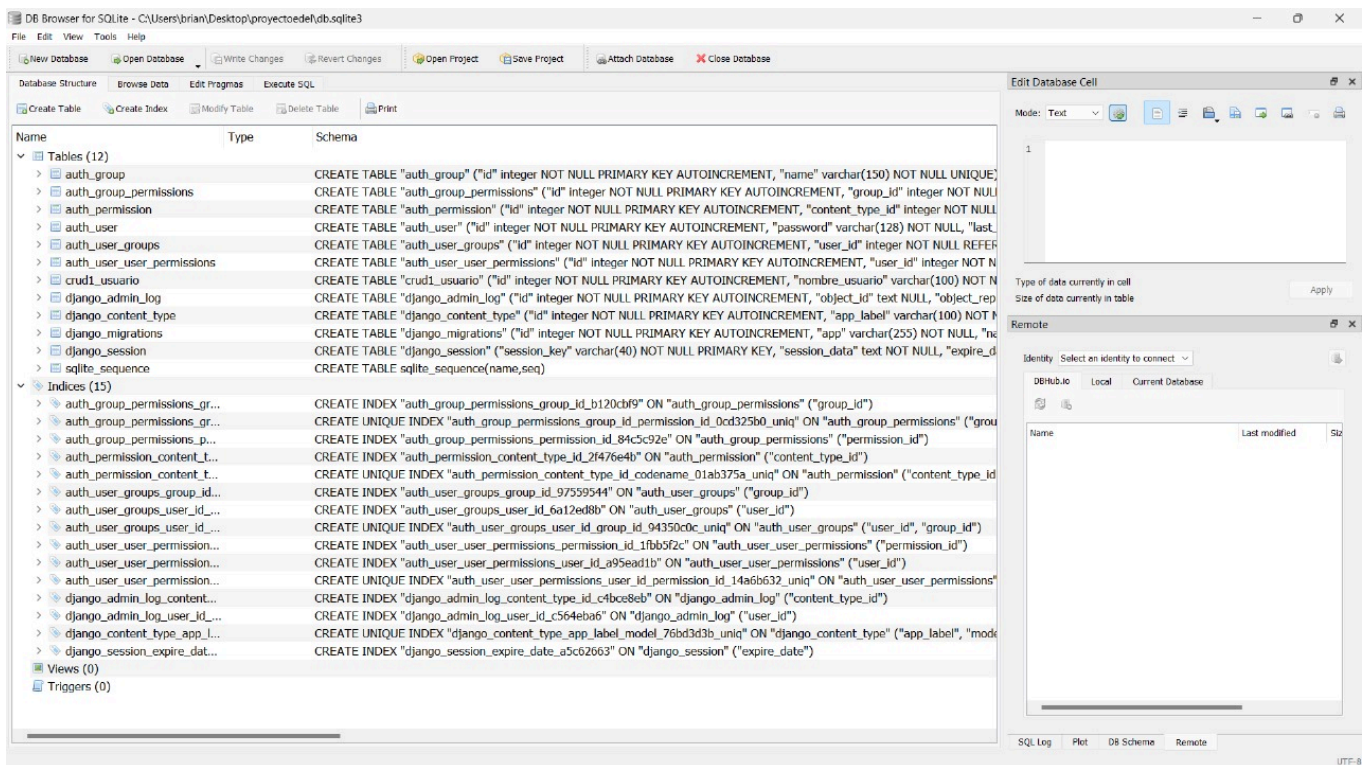
This screenshot is identical to the one above, showing the 'views.py' file in the 'proyectoodel' project. It displays the 'login' function and the beginning of the 'interfaz_sa' function, which is commented out.

The terminal window shows the output of the Django development server. It displays several GET requests for the root URL and the static files directory. The server is running on port 2534. The user has entered the command 'PS C:\Users\brian\Desktop\proyectoodel> .\venv\Scripts\activate' to activate the virtual environment.

```
Quit the server with CTRL-BREAK.
[13/Feb/2024 18:49:32] "GET / HTTP/1.1" 200 2534
[13/Feb/2024 18:49:35] "GET /static/manual.pdf HTTP/1.1" 404 1789
[13/Feb/2024 18:50:13] "GET / HTTP/1.1" 200 2534
[13/Feb/2024 18:50:14] "GET /static/manual.pdf HTTP/1.1" 200 48682
[13/Feb/2024 18:50:48] "GET /static/manual.pdf HTTP/1.1" 200 48682
(vnv) PS C:\Users\brian\Desktop\proyectoodel>
* History restored
PS C:\Users\brian\Desktop\proyectoodel> .\venv\Scripts\activate
(vnv) PS C:\Users\brian\Desktop\proyectoodel>
```

¿Qué base de datos?

Con base en nuestra investigación, les recomendamos que elija la misma base de datos para producción y desarrollo (aunque Django abstrae muchas de las diferencias entre bases de datos utilizando un Mapeador relacional de objetos (ORM), todavía existen problemas potenciales que es mejor evitar) En cuanto a las BD, podemos destacar a: *PostgreSQL, MySQL, Oracle y SQLite* Para nuestro trabajo utilizamos SQLite, así como se muestra de la siguiente manera.



Django se puede implementar en una variedad de servicios de alojamiento web, incluidos servidores compartidos, servidores virtuales privados (VPS) y plataformas de computación en la nube como AWS, Google Cloud Platform o Heroku. Estos servicios suelen requerir la configuración de un servidor web (como Apache o Nginx) y una base de datos (como PostgreSQL, MySQL y también SQLite) para alojar una aplicación Django. A opinión personal también podría utilizarse InfinityFree que es un servicio de alojamiento gratuito.

Envío de Correos:

Django proporciona una interfaz simple para enviar correos electrónicos utilizando el módulo ``django.core.mail``. Para enviar correos, se debe configurar la sección ``EMAIL`` en el archivo ``settings.py``, que incluye detalles como el servidor SMTP, el puerto, las credenciales, etc. Luego, se pueden enviar correos electrónicos utilizando funciones proporcionadas por Django, como ``send_mail()``.

Para enviar un correo electrónico con Django, debemos crear un objeto de la clase `EmailMessage`. Este objeto representa el correo electrónico que queremos enviar.

Como se muestra de la siguiente manera:

```
from django.core.mail import EmailMessage
email = EmailMessage(
    subject='Asunto del correo electrónico',
    body='Cuerpo del correo electrónico',
    from_email='tu_correo@gmail.com',
    to_email='destinatario@example.com',
)
```

Los argumentos de `EmailMessage` son los siguientes:

- `subject`: El título del correo electrónico.
- `body`: El cuerpo del correo electrónico.
- `from_email`: La dirección de correo electrónico del remitente.
- `to_email`: La dirección de correo electrónico del destinatario.

Creación de PDF:

Para crear PDF en Django, se pueden utilizar bibliotecas como ReportLab o WeasyPrint. Estas bibliotecas permiten generar archivos PDF dinámicamente utilizando datos de la aplicación Django y plantillas HTML.

```
<div class="text-center mt-3">
  <a href="{% static 'manual.pdf' %}" download="manual.pdf" class="btn btn-secondary">Descargar Manual</a>
</div>
```

```
from django.conf import settings Esto en views.py
from django.http import HttpResponse, Http404
import os

def download_pdf(request, filename):
    # Obtén la ruta completa del archivo PDF
    pdf_path = os.path.join(settings.MEDIA_ROOT, 'pdfs', filename)

    # Verifica si el archivo existe
    if os.path.exists(pdf_path):
        # Abre el archivo y lo sirve como una respuesta HTTP
        with open(pdf_path, 'rb') as pdf_file:
            response = HttpResponse(pdf_file.read(),
            content_type='application/pdf')
            response['Content-Disposition'] = f'attachment;
            filename="{filename}"'
            return response
    else:
        # Si el archivo no existe, devuelve un error 404
        raise Http404("El archivo PDF no se encontró")
```

Y en "urls.py" agregamos esto

```
from django.urls import path
from .views import download_pdf
urlpatterns = [
    path('download-pdf/<str:filename>/', download_pdf,
    name='download_pdf'),
    # Otras URLs de tu aplicación...
```

Descargar Archivos del Servicio:

En Django, la descarga de archivos se puede implementar mediante vistas que sirven archivos estáticos almacenados en el sistema de archivos del servidor o archivos generados dinámicamente. Se pueden utilizar las respuestas de archivo proporcionadas por Django para enviar archivos al navegador del usuario.

Primero que nada, abrimos el archivo `views.py` y agregamos la función `index()` que carga la página HTML y la función `descargar_archivo()` que realiza el proceso de descargar del archivo:

```
from django.shortcuts import render
from django.http.response import HttpResponseRedirect
import mimetypes
import os

# Create your views here.
def index(request):
    return render(request, 'index.html')

def descargar_archivo(request):
    BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    filename = 'mi_archivo.txt'
    filepath = BASE_DIR + '/descargar/archivos/' + filename
    path = open(filepath, 'r')
    mime_type, _ = mimetypes.guess_type(filepath)
    response = HttpResponseRedirect(path, content_type = mime_type)
    response['Content-Disposition'] = f'attachment; filename={filename}'
    return response
```

Paso seguido abrimos el archivo `urls.py` y agregamos 2 rutas, una para la página HTML y otra para el enlace de descarga del archivo:

```
"""djangodownload URL Configuration
```


The `urlpatterns` list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/4.0/topics/http/urls/>

Examples:

Function views

1. Add an import: `from my_app import views`
2. Add a URL to `urlpatterns`: `path('', views.home, name='home')`

Class-based views

1. Add an import: `from other_app.views import Home`
2. Add a URL to `urlpatterns`: `path('', Home.as_view(), name='home')`

Including another `URLconf`

1. Import the `include()` function: `from django.urls import include, path`

2. Add a URL to `urlpatterns`: `path('blog/', include('blog.urls'))`

"""

```
from django.contrib import admin
```

```
from django.urls import path
```

```
from descargar import views
```

```
urlpatterns = [
```

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

```
    path('', views.index, name='home'),
```

```
    path('descargar/', views.descargar_archivo, name = "descargar"),
```

```
]
```

Ahora, dentro del directorio `descargar` creamos el directorio `templates` y dentro de el creamos un archivo con el nombre `index.html`, abrimos el archivo `index.html` y agregamos lo siguiente:

```
<!-- Enlace de Descarga -->
```

```
<a href="{% url 'descargar' %}">Descargar</a>
```

Encriptación y Des encriptación:

Django proporciona funciones integradas para encriptar y desencriptar datos de manera segura. Por ejemplo, el módulo `django.crypt` proporciona funciones como `encrypt()` y `decrypt()` para cifrar y descifrar datos.

En Django, la encriptación y desencriptación de datos generalmente se realiza utilizando la biblioteca `django.core.signing`. Esta biblioteca proporciona métodos simples para firmar y verificar datos utilizando claves secretas.

```
from django.core.signing import Signer

SECRET_KEY = 'tu_clave_secreta_aqui'

signer = Signer(SECRET_KEY)

//Datos que vamos a encriptar

data_to_encrypt = 'Hello, World!'

//Encriptar los datos

encrypted_data = signer.sign(data_to_encrypt)

print("Datos encriptados:", encrypted_data)

# Desencriptar los datos

decrypted_data = signer.unsign(encrypted_data)

print("Datos desencriptados:", decrypted_data)
```

Creación de Menús y Submenús:

La creación de menús y submenús en Django generalmente se realiza utilizando plantillas HTML y CSS para la interfaz de usuario, junto con datos dinámicos proporcionados por la aplicación Django. Se pueden crear modelos de base de datos para representar elementos de menú y luego renderizar estos modelos en plantillas HTML para generar la interfaz de usuario.

```
print("""
1) Herramientas                                3) fye

2) Noticias                                    4) Mas...
""")
eligio = input("-Selecciona algo : ")

if eligio=="1":
    print("""
Listamos las herramientas para...
1) Opcion 1                                2) Opcion 2""")

    eligio = input("-Selecciona algo : ")
    if eligio == "1":
        print("Ha seleccionado opcion 1")
    elif eligio == "2":
        print("Ha seleccionado opcion 2")
    else:
        print("Opcion no valida")

elif eligio == "2":
    print("Que noticias?")
elif eligio == "3":
    print("¿Que es fye?")
elif eligio == "4":
    print("otra opción")
else:
    print("Opcion no valida")
```

Almacenar Imágenes:

Para almacenar imágenes en Django, se pueden utilizar campos de tipo `ImageField` proporcionados por el ORM de Django. Estos campos permiten subir imágenes y almacenarlas en el sistema de archivos del servidor o en un servicio de almacenamiento en la nube como Amazon S3.

Pero también para almacenar imágenes en Python utilizando Django, puedes aprovechar el modelo de almacenamiento de archivos proporcionado por Django

Primero debes crear y configurar el Modelo para almacenar las imágenes, se puede hacer de la siguiente forma:

1.-Configurar tu settings.py

```
MEDIA_URL = '/media/' MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

2.- Configurar tu archivo urls.py

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

urlpatterns = [
    # tus urls
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

3.-Definiendo tu models.py

```
from django.db import models

class Documento(models.Model):
    descripcion = models.CharField(max_length=255, blank=True)
    documento = models.FileField(upload_to='documentos/')
    subido_a = models.DateTimeField(auto_now_add=True)
```

4.-Definiendo tu forms.py

```
from django import forms
import *.models

class DocumentoForm(forms.ModelForm):
    class Meta:
        model = Documento
        fields = ('descripcion', 'documento', )
```

5.-Define tu views.py

```
def mi_metodo(request):
    if request.method == 'POST':
        form = DocumentoForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('index')#redirigue a donde deseas
    else:
        form = DocumentoForm()
        return render(request, 'mi_template.html', {
            'form': form
        })
```

6.-Tu template.html

```
{% extends 'base.html' %}

{% block content %}
<form method="post" enctype="multipart/form-data">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Subir</button>
</form>

<p><a href="{% url 'index' %}">Regresar</a></p>
{% endblock %}
```

Debug:

Django nos ofrece un potente conjunto de herramientas de manejo de errores, depuración y registro de eventos que pueden resultar útiles durante el desarrollo y la depuración de aplicaciones web.

En cuanto a los **mensajes de depuración**, Django proporciona un sistema de mensajes de depuración que le permite imprimir mensajes en la consola durante el desarrollo. Puede utilizar la función `print()` o el módulo de registro para enviar mensajes de depuración.

El primer paso para depurar, es establecer un punto de ruptura en la línea deseada. En este caso lo estableceremos en la vista principal (`IndexView`), en el fichero `polls/views.py`. Es necesario importar el módulo `pdb`.

```
import pdb

class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_poll_list'

    def get_queryset(self):
        """Return the last five published polls (not
including those set to be
published in the future)
        """
        pdb.set_trace() ## Punto de ruptura
        return Poll.objects.filter(
            pub_date__lte=timezone.now()
        ).order_by('-pub_date')[:5]
```

Herramientas de depuración de terceros: además de las herramientas integradas en Django, existen muchas herramientas de terceros que pueden ayudar con la depuración,

como la barra de herramientas de depuración de Django, que proporciona información detallada sobre el rendimiento de la vista, consultas de bases de datos, plantillas, etc. en.

Pase de parámetros:

En el desarrollo web con Django, los parámetros son elementos fundamentales que permiten a las vistas procesar datos específicos proporcionados por el usuario. Los parámetros pueden ser parte de la URL, enviados a través de solicitudes GET o POST, y pueden ser utilizados para personalizar la funcionalidad de las vistas en consecuencia.

Entro de lo que es el archivo de Template debemos agregar esto:

```
<!DOCTYPE html>
<html>
<head>
    <title>Django Forms #1</title>
</head>
<body>
<div class="container">
    {% block mycontent %}
    {% endblock %}
</div>
</body>
</html>
```

El archivo index-view.html es nuestra primera vista y sera un formulario con 3 datos, nombre, email y password.

```
{% extends 'djforms1/template.html' %}
{% block mycontent %}

<h1>Formulario</h1>
<p>Hola esta es una vista</p>
<form action="result" method="get">
    <label for="your_name">Tu nombre: </label>
    <input id="name" type="text" name="name" placeholder="Tu
nombre">
    <br>
    <label for="your_name">Tu email: </label>
    <input id="name" type="text" name="email" placeholder="Tu
email">
    <br>
    <label for="your_name">Tu password: </label>
    <input id="name" type="password" name="password"
placeholder="Tu password">
```

```
<input type="submit" value="Enviar Datos">
</form>
{% endblock %}
```

Lo que hace esa vista es leer los datos del usuario y despues llamar al archivo “result” que veremos mas adelante como una función o ruta en python.

El archivo mostrar-view.html se ejecuta en el codigo de la funcion o metodo “result” y lo que se hace es obtener la informacion y despues mostrarla.

```
{% extends 'djforms1/template.html' %}

{% block mycontent %}

<h1>Mostrar Datos</h1>

<p>Tu nombre: {{name}}</p>
<p>Tu correo: {{email}}</p>
<p>Tu password: {{password}}</p>

{% endblock %}
```

El archivo startproject.py contiene las funciones para llamar a las vistas, y en la funcion result esta el codigo para Recibir los valores via GET.

```
from django.http import HttpResponse
from django.shortcuts import render

# Funcion para el metodo principal que muestra el archivo
index-view.html que asu vez carga el template template.html
def index(request):
    return render(request, "djforms1/index-view.html", {})

# Funcion para el metodo result que recibe los datos via GET y
despues muestra en el archivo mostrar-view.html que asu vez
carga el template template.html
def result(request):
    name = request.GET["name"] # obtener valor de la clave
'nombre' via GET
    email = request.GET["email"] # obtener valor de la clave
'email' via GET
    password = request.GET["password"] # obtener valor de la
clave 'password' via GET
    # Una vez obtenido los datos los pasamos en el tercer
parametro de la funcion render asociando un nombre y la variable
que contiene el valor
```

```
        return render(request, "djforms1/mostrar-view.html",
{'name':name, 'email': email, 'password':password} )
```

Por ultimo pero no menos importante es necesario agregar nuestras URLs o Rutas a el archivo urls.py.

Archivo urls.py

Este archivo contiene todas nuestras rutas y la función o metodo que deben ejecutar.

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

from . import startproject

urlpatterns = [
    path("djforms1/", startproject.index, name="index"),
    path("djforms1/result", startproject.result, name="result")
]
```

1. Parámetros de URL

Los parámetros de URL son parte de la propia dirección web y se utilizan para pasar datos específicos a una vista. Por ejemplo, en una aplicación de blog, la URL `blog/post/1/` puede contener el identificador del post, que se pasa como un parámetro en la URL (`1` en este caso). La vista correspondiente puede recuperar este parámetro de la URL para mostrar el post específico.

2. Parámetros de Consulta GET

Los parámetros de consulta GET se envían a través de la URL como pares de clave-valor después del signo de interrogación (`?`). Por ejemplo, en una búsqueda en un sitio web de comercio electrónico, la URL puede ser algo como `search?query=producto`, donde `query` es el parámetro y `producto` es el valor. Estos parámetros pueden ser utilizados por la vista para realizar la búsqueda correspondiente y devolver los resultados.

3. Parámetros de Formulario POST

Los parámetros de formulario se envían a través de una solicitud POST y son útiles cuando se necesitan enviar datos sensibles o una gran cantidad de información. Por ejemplo, al completar un formulario de registro en un sitio web, los datos como el nombre, correo

electrónico y contraseña se envían al servidor a través de una solicitud POST. La vista puede procesar estos datos y crear una nueva cuenta de usuario en la base de datos.

4. Parámetros de URL Nombrados

Django permite nombrar las URLs, lo que hace que el código sea más legible y mantenible. Esto se hace asignando un nombre a la URL en `urls.py` y luego referenciando este nombre en las plantillas y vistas. Por ejemplo, en lugar de escribir la URL directamente en una plantilla HTML, se puede usar el nombre asignado a la URL, lo que facilita el mantenimiento del código y la actualización de las rutas de URL en el futuro.

Https:

Puede obtenerlo a través de un proveedor de certificados como Let's Encrypt (que ofrece certificados gratuitos) o comprándolo a un proveedor de certificados.

Configure su servidor web: HTTPS generalmente se maneja a nivel de servidor web, como Nginx o Apache. Configure su servidor web para servir su aplicación Django a través de HTTPS.

Para lograr una buena conexión, debemos seguir los siguientes pasos.

```
server {  
    listen 443 ssl;  
    server_name tu_dominio.com;  
  
    ssl_certificate /ruta/al/certificado.crt;  
    ssl_certificate_key /ruta/a/llave_privada.key;  
  
    location / {  
        proxy_pass http://127.0.0.1:8000;    # Cambia el puerto si tu  
aplicación Django se ejecuta en otro puerto  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        proxy_set_header X-Forwarded-Proto $scheme;  
    }  
}
```

Posteriormente, configure Django para que funcione a través de un proxy inverso. Si su aplicación Django se ejecuta a través de un proxy inverso (como Nginx en este caso), asegúrese de configurar Django correctamente para que reconozca lo que hay detrás del proxy.

```
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
```

Habilite HTTPS en Django. Si bien el procesamiento HTTPS generalmente se realiza en el servidor web, puedes configurar Django para generar URL a través de HTTPS.

```
SECURE_SSL_REDIRECT = True
```

Verifique su configuración: reinicie su servidor web y la aplicación Django, y luego acceda a la aplicación a través de HTTPS (https://your_domain.com). Asegúrese de que los enlaces generados dinámicamente en su aplicación también utilicen HTTPS.

Responsivo:

En el contexto de Django, el desarrollo de un diseño web responsivo implica principalmente el uso de HTML y CSS, junto con algunas técnicas adicionales para integrar la funcionalidad de Django. A continuación, se presentan algunas estrategias específicas para lograr un diseño web responsivo en una aplicación Django:

1. Uso de Plantillas HTML Responsive:

Utiliza plantillas HTML que sean responsivas, es decir, que estén diseñadas para adaptarse a diferentes tamaños de pantalla. Puedes utilizar frameworks CSS como Bootstrap o Foundation, que proporcionan componentes y estilos predefinidos para crear fácilmente diseños responsivos.

2. Media Queries:

Utiliza media queries en tu archivo CSS para aplicar estilos específicos según el tamaño de la pantalla del dispositivo. Esto te permitirá definir estilos personalizados para diferentes resoluciones de pantalla, como anchos mínimos y máximos, alturas mínimas y máximas, y orientaciones (horizontal o vertical).

3. Uso de Plantillas Base:

Crea una plantilla base en Django que contenga la estructura HTML básica de tu sitio web, incluyendo la declaración de la etiqueta meta viewport para asegurar un comportamiento adecuado en dispositivos móviles. Luego, extiende esta plantilla base en tus otras plantillas para mantener la consistencia en todo el sitio.

4. Anidamiento de Plantillas:

Django permite el anidamiento de plantillas, lo que te permite dividir tu código HTML en partes más pequeñas y reutilizables. Utiliza este enfoque para organizar tu código y hacerlo más mantenible.

5. Integración con Bootstrap o Otros Frameworks CSS:

Integra un framework CSS como Bootstrap en tu proyecto Django para facilitar el desarrollo de un diseño responsivo. Puedes instalar Bootstrap usando pip o simplemente incluyendo los archivos CSS y JavaScript en tu proyecto.

6. Pruebas en Dispositivos Reales y Emuladores:

Realiza pruebas exhaustivas en dispositivos reales y emuladores para asegurarte de que tu diseño responsivo funcione correctamente en una variedad de dispositivos y tamaños de pantalla. Esto te permitirá identificar y corregir cualquier problema de diseño o visualización.

7. Optimización de Imágenes:

Optimiza tus imágenes para la web utilizando herramientas como Pillow en Django o herramientas externas como ImageMagick. Esto te permitirá reducir el tamaño de los archivos de imagen y mejorar el rendimiento de tu sitio web en dispositivos móviles.

Al seguir estas estrategias, podrás desarrollar una aplicación Django con un diseño web responsivo que proporcione una experiencia de usuario óptima en una amplia gama de dispositivos y tamaños de pantalla.

JQuery - Ajax:

Primeramente, debemos tener instalado jQuery y AJAX en nuestro proyecto o utilizar CDN para cargar jQuery

Necesitas una vista en Django que procese la solicitud AJAX. Por ejemplo, en tu archivo `views.py`, y ahí poner algo como esto:

```
from django.http import JsonResponse

def ajax_example(request):
    # Procesa la solicitud AJAX
    data = {'message': '¡Hola desde Django!'}
    return JsonResponse(data)
```

Asegúrate de tener una URL definida en tu archivo `urls.py` que apunte a la vista que acabas de crear

```
from django.urls import path
from .views import ajax_example

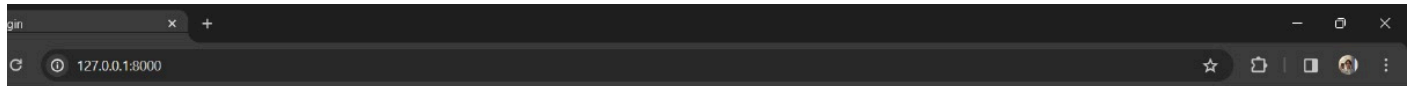
urlpatterns = [
    path('ajax/', ajax_example, name='ajax_example'),
    # Otras URLs de tu aplicación...
]
```

Ahora, puedes usar jQuery para hacer una solicitud AJAX a tu vista de Django. Por ejemplo, en un archivo JavaScript dentro de tu plantilla HTML

```
$(document).ready(function() {
    $.ajax({
        url: '/ajax/', // La URL de la vista en Django
        type: 'GET', // Método de solicitud
        success: function(data) {
            // Maneja la respuesta exitosa
            console.log(data.message); // Imprime el mensaje recibido desde
Django en la consola del navegador
        },
        error: function(xhr, status, error) {
            // Maneja errores de la solicitud
            console.error('Error:', error);
        }
    });
});
```

```
}  
});  
});
```

Visualización Previa



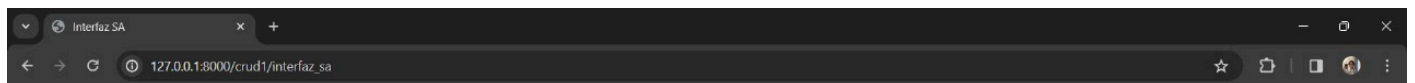
Login

Usuario:

Contraseña:

[Entrar](#)

[Descargar Manual](#)



Agregar Usuarios

[Agregar Usuario](#) [Tipo de Usuario](#) [Estado del Usuario](#)

Buscar usuarios... [Buscar](#)

Nombre de Usuario	Tipo de Usuario	Estado	Acciones
prueba5	taquillero	activo	Editar Eliminar
sa	sa	activo	Editar Eliminar
prueba7	taquillero	activo	Editar Eliminar

