

crear un ejemplo de lo mas basico para poder en django mostrar en una plantilla un documento pdf y con la opcion de descargarlo, este proyecto debera ser explicado paso a paso y a detalle para un novato ,

ChatGPT

¡Por supuesto! Te guiaré paso a paso para crear una aplicación básica en Django que permita mostrar un documento PDF en una plantilla y ofrecer la opción de descargarlo. Aquí está el proceso completo:

1. Crear un Proyecto Django

Primero, asegúrate de tener Django instalado. Si no lo tienes, puedes instalarlo usando pip:

```
bash
Copiar código
pip install django
```

Luego, crea un nuevo proyecto Django:

```
bash
Copiar código
django-admin startproject myproject
```

Cambia al directorio del proyecto:

```
bash
Copiar código
cd myproject
```

2. Crear una Aplicación Django

Dentro de tu proyecto, crea una nueva aplicación:

```
bash
Copiar código
python manage.py startapp myapp
```

3. Configurar el Proyecto y la Aplicación

Añade myapp a INSTALLED_APPS en myproject/settings.py:

```
python
Copiar código
INSTALLED_APPS = [
    # ...
    'myapp',
]
```

Configura los archivos estáticos y de medios en `myproject/settings.py`:

```
python
Copiar código
import os

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

Añade estas líneas al final de `myproject/urls.py` para manejar los archivos de medios durante el desarrollo:

```
python
Copiar código
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # ...
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

4. Crear un Modelo para los Documentos PDF

En `myapp/models.py`, define un modelo para los documentos PDF:

```
python
Copiar código
from django.db import models

class Document(models.Model):
    title = models.CharField(max_length=100)
    pdf = models.FileField(upload_to='documents/')

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

Luego, migra los cambios:

```
bash
Copiar código
python manage.py makemigrations
python manage.py migrate
```

5. Crear un Formulario para Subir PDFs

En `myapp/forms.py`, crea un formulario para subir archivos PDF:

```
python
Copiar código
from django import forms
from .models import Document
```

```
class DocumentForm(forms.ModelForm):  
    class Meta:  
        model = Document  
        fields = ('title', 'pdf')
```

6. Crear Vistas para Subir y Mostrar PDFs

En `myapp/views.py`, añade las vistas para manejar la subida y la visualización de los documentos PDF:

```
python  
Copiar código  
from django.shortcuts import render, redirect  
from .forms import DocumentForm  
from .models import Document  
  
def upload_document(request):  
    if request.method == 'POST':  
        form = DocumentForm(request.POST, request.FILES)  
        if form.is_valid():  
            form.save()  
            return redirect('document_list')  
    else:  
        form = DocumentForm()  
    return render(request, 'myapp/upload_document.html', {'form': form})  
  
def document_list(request):  
    documents = Document.objects.all()  
    return render(request, 'myapp/document_list.html', {'documents':  
documents})
```

7. Crear Plantillas para las Vistas

Crea un directorio llamado `templates` dentro de `myapp`, y dentro de ese directorio, crea un subdirectorio llamado `myapp`. Luego, crea los archivos `upload_document.html` y `document_list.html`.

upload_document.html

```
html  
Copiar código  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Upload Document</title>  
</head>  
<body>  
    <h1>Upload Document</h1>  
    <form method="post" enctype="multipart/form-data">  
        {% csrf_token %}  
        {{ form.as_p }}  
        <button type="submit">Upload</button>
```

```
</form>
<a href="{% url 'document_list' %}">Back to document list</a>
</body>
</html>

document_list.html
html
Copiar código
<!DOCTYPE html>
<html>
<head>
  <title>Document List</title>
</head>
<body>
  <h1>Document List</h1>
  <ul>
    {% for document in documents %}
      <li>
        {{ document.title }} - <a href="{{ document.pdf.url }}">View PDF</a> - <a href="{{ document.pdf.url }}">download</a>Download PDF</a>
      </li>
    {% endfor %}
  </ul>
  <a href="{% url 'upload_document' %}">Upload new document</a>
</body>
</html>
```

8. Configurar las URLs

En myapp/urls.py, configura las URLs para las vistas:

```
python
Copiar código
from django.urls import path
from .views import upload_document, document_list

urlpatterns = [
    path('upload/', upload_document, name='upload_document'),
    path('', document_list, name='document_list'),
]
```

Luego, incluye estas URLs en myproject/urls.py:

```
python
Copiar código
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]
```

9. Ejecutar el Servidor

Finalmente, ejecuta el servidor de desarrollo para probar la aplicación:

```
bash
Copiar código
python manage.py runserver
```

Ahora, puedes visitar `http://127.0.0.1:8000/upload/` para subir un documento PDF y `http://127.0.0.1:8000/` para ver la lista de documentos y descargarlos.

¡Y eso es todo! Has creado una aplicación Django básica para subir y mostrar documentos PDF con la opción de descargarlos.

40

```
from .views import upload_document, document_list
File "C:\jemplo-django-pdf\myproject\myapp\views.py", line 2, in <module>
from .forms import DocumentForm
ModuleNotFoundError: No module named 'myapp.forms'
```

ChatGPT

Parece que el error indica que no puede encontrar el módulo `forms`. Vamos a asegurarnos de que has creado el archivo `forms.py` dentro de la aplicación `myapp` y que su contenido está correctamente definido. Aquí tienes una revisión de los pasos necesarios para asegurar que todo está en orden:

1. Crear el Archivo `forms.py`

Dentro del directorio `myapp`, crea un archivo llamado `forms.py` si no lo has hecho ya. El archivo `forms.py` debe contener el siguiente código:

```
python
Copiar código
# myapp/forms.py
from django import forms
from .models import Document

class DocumentForm(forms.ModelForm):
    class Meta:
        model = Document
        fields = ('title', 'pdf')
```

2. Verificar la Estructura del Proyecto

Asegúrate de que la estructura del proyecto se vea algo así:

```
markdown
Copiar código
myproject/
  myproject/
    __init__.py
    settings.py
    urls.py
    wsgi.py
  myapp/
    __init__.py
    admin.py
    apps.py
    forms.py
    models.py
    views.py
    templates/
      myapp/
        upload_document.html
        document_list.html
  manage.py
```

3. Revisar las Importaciones

Asegúrate de que las importaciones en `myapp/views.py` estén correctas. Deberían verse así:

```
python
Copiar código
# myapp/views.py
from django.shortcuts import render, redirect
from .forms import DocumentForm
from .models import Document

def upload_document(request):
    if request.method == 'POST':
        form = DocumentForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('document_list')
    else:
        form = DocumentForm()
    return render(request, 'myapp/upload_document.html', {'form': form})

def document_list(request):
    documents = Document.objects.all()
    return render(request, 'myapp/document_list.html', {'documents':
documents})
```

4. Ejecutar el Servidor

Si todo está en su lugar, intenta ejecutar el servidor de nuevo:

```
bash
```

```
Copiar código  
python manage.py runserver
```

Verificar y Corregir el Error

Si el error persiste, asegúrate de que el archivo `forms.py` está en el directorio correcto (`myapp/`) y que no contiene errores tipográficos. Puedes comprobarlo listando los archivos en el directorio `myapp`:

```
bash  
Copiar código  
ls myapp
```

Deberías ver `forms.py` en la lista. Si todo está en su lugar y sigues teniendo problemas, intenta reiniciar el servidor de desarrollo y asegúrate de que no haya cachés antiguas:

```
bash  
Copiar código  
python manage.py runserver --noreload
```

Con estas verificaciones y correcciones, deberías poder resolver el problema del módulo `forms` no encontrado.

4o

si lo sube si lo descarga pero no lo muestra

ChatGPT

Para mostrar el PDF en una plantilla, necesitamos usar un visor de PDF en HTML. Aquí hay una forma de hacerlo usando el elemento `<embed>` o `<iframe>`. Vamos a modificar el archivo `document_list.html` para que incluya la capacidad de mostrar el PDF dentro de la página.

Modificar `document_list.html` para Mostrar PDFs

Edita `document_list.html` para que se vea así:

```
html  
Copiar código  
<!DOCTYPE html>  
<html>  
<head>  
  <title>Document List</title>  
</head>  
<body>  
  <h1>Document List</h1>  
  <ul>
```

```
{% for document in documents %}
  <li>
    {{ document.title }} -
    <a href="{{ document.pdf.url }}" target="_blank">View PDF
in New Tab</a> -
    <a href="{{ document.pdf.url }}" download>Download
PDF</a>
    <br>
    <!-- Embed PDF directly in the page -->
    <embed src="{{ document.pdf.url }}"
type="application/pdf" width="600" height="400">
  </li>
{% endfor %}
</ul>
<a href="{% url 'upload_document' %}">Upload new document</a>
</body>
</html>
```

Explicación de los Cambios

1. **Link para Ver en Nueva Pestaña:** Se ha añadido un enlace para ver el PDF en una nueva pestaña del navegador (`target="_blank"`).
2. **Elemento `<embed>`:** Se ha añadido un elemento `<embed>` para mostrar el PDF directamente en la página. El atributo `src` se establece en la URL del PDF, y los atributos `width` y `height` especifican el tamaño del visor de PDF.

Si prefieres usar un `<iframe>`, puedes hacerlo así:

```
html
Copiar código
<iframe src="{{ document.pdf.url }}" width="600" height="400"
style="border: none;"></iframe>
```

Verificar los Resultados

1. **Sube un Documento PDF:** Visita `http://127.0.0.1:8000/upload/` y sube un archivo PDF.
2. **Ver la Lista de Documentos:** Visita `http://127.0.0.1:8000/` para ver la lista de documentos. Deberías ver el PDF incrustado en la página y tener la opción de descargarlo o abrirlo en una nueva pestaña.

Con estos cambios, deberías poder ver y descargar los PDFs directamente desde la plantilla en Django.

4o

en firefox no se ve en chrome si manda la liga ver documento pero en lista documento abajo de las ligas ver y descargar documento sale un recuadro como si quisiera mostrar el documento

ChatGPT

Para mejorar la compatibilidad entre diferentes navegadores, especialmente Firefox y Chrome, y garantizar que el PDF se muestre correctamente, vamos a revisar el uso de elementos HTML y asegurarnos de que la configuración es adecuada para todos los navegadores. Aquí hay un enfoque actualizado utilizando tanto `<embed>` como `<iframe>`.

1. Modificar `document_list.html` para Mejor Compatibilidad

Edita `document_list.html` para probar diferentes métodos de visualización de PDFs:

```
html
Copiar código
<!DOCTYPE html>
<html>
<head>
    <title>Document List</title>
</head>
<body>
    <h1>Document List</h1>
    <ul>
        {% for document in documents %}
            <li>
                {{ document.title }} -
                <a href="{{ document.pdf.url }}" target="_blank">View PDF
in New Tab</a> -
                <a href="{{ document.pdf.url }}" download>Download
PDF</a>
                <br>
                <!-- Attempt to embed the PDF using different methods -->
                <embed src="{{ document.pdf.url }}"
type="application/pdf" width="600" height="400" />
                <iframe src="{{ document.pdf.url }}" width="600"
height="400" style="border: none;"></iframe>
                <object data="{{ document.pdf.url }}"
type="application/pdf" width="600" height="400">
                    <a href="{{ document.pdf.url }}">Download PDF</a>
                </object>
            </li>
        {% endfor %}
    </ul>
    <a href="{% url 'upload_document' %}">Upload new document</a>
</body>
</html>
```

Explicación de los Métodos

1. **Elemento `<embed>`:** Es útil para navegadores que soportan la incrustación de PDFs.
2. **Elemento `<iframe>`:** A veces, los navegadores manejan mejor los PDFs con este elemento.
3. **Elemento `<object>`:** Proporciona una tercera forma de incrustar PDFs y es especialmente útil si los otros dos métodos fallan.

2. Verificar en Diferentes Navegadores

1. **Sube un Documento PDF:** Visita `http://127.0.0.1:8000/upload/` y sube un archivo PDF.
2. **Ver la Lista de Documentos:** Visita `http://127.0.0.1:8000/` para ver la lista de documentos y probar la visualización en diferentes navegadores.

Nota sobre Estilos y Tamaño

Ajusta el tamaño (`width` y `height`) de los elementos según sea necesario para que se ajusten a tu diseño. Los estilos en línea (`style`) pueden ser útiles para personalizar la apariencia de los visores de PDF.

Verificación y Solución de Problemas

Si alguno de los métodos no funciona en un navegador específico, asegúrate de revisar las configuraciones de seguridad del navegador y si el visor de PDF integrado está habilitado. Los navegadores pueden tener diferentes configuraciones que afectan cómo se manejan los archivos PDF.

Este enfoque debería mejorar la compatibilidad y garantizar que los PDFs se muestren correctamente en la mayoría de los navegadores.

4o

no se ve el pdf pero si funciona el ver pdf y el descarga pero en el recuadro no se ve y manda este mensaje Se bloqueó 127.0.0.1 127.0.0.1 rechazó la conexión.
ERR_BLOCKED_BY_RESPONSE

ChatGPT

El error `ERR_BLOCKED_BY_RESPONSE` generalmente ocurre cuando el navegador no puede cargar el contenido debido a restricciones de seguridad o configuración del servidor. Aquí hay algunas cosas que puedes hacer para solucionar este problema:

1. Verificar la Configuración del Servidor

Asegúrate de que Django esté configurado para servir archivos estáticos y de medios correctamente. Ya configuraste `MEDIA_URL` y `MEDIA_ROOT` en `settings.py`, pero vamos a revisarlo nuevamente para asegurarnos de que todo esté en orden:

```
python
Copiar código
import os
```

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

En `urls.py` del proyecto principal (`myproject/urls.py`), asegúrate de que los archivos de medios se estén sirviendo correctamente:

```
python  
Copiar código  
from django.conf import settings  
from django.conf.urls.static import static  
from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('', include('myapp.urls')),  
]  
  
if settings.DEBUG:  
    urlpatterns += static(settings.MEDIA_URL,  
        document_root=settings.MEDIA_ROOT)
```

2. Configuración de CORS

El problema también puede estar relacionado con las políticas de CORS (Cross-Origin Resource Sharing). Aunque no debería ser un problema en `localhost`, podrías intentar configurar Django para permitir estos recursos explícitamente.

Instala el paquete `django-cors-headers`:

```
bash  
Copiar código  
pip install django-cors-headers
```

Agrega `corsheaders` a tu `INSTALLED_APPS` en `settings.py`:

```
python  
Copiar código  
INSTALLED_APPS = [  
    ...  
    'corsheaders',  
]
```

Añade el middleware de `corsheaders` al principio de la lista de `MIDDLEWARE`:

```
python  
Copiar código  
MIDDLEWARE = [  
    'corsheaders.middleware.CorsMiddleware',  
    ...  
]
```

Configura las políticas de CORS en `settings.py`:

```
python
Copiar código
CORS_ALLOW_ALL_ORIGINS = True
```

3. Asegúrate de que los PDFs se Sirven Correctamente

Asegúrate de que los archivos PDF realmente se están sirviendo desde el servidor y que sus URLs son accesibles. Puedes probar accediendo directamente a la URL del PDF en tu navegador.

4. Uso de `X-Frame-Options` en las Respuestas HTTP

Asegúrate de que tu servidor no esté bloqueando el contenido debido a restricciones de cabeceras `X-Frame-Options`. En Django, puedes configurar esto en `settings.py`:

```
python
Copiar código
X_FRAME_OPTIONS = 'ALLOWALL'
```

5. Prueba con un Servidor HTTP en Lugar de `runserver`

El servidor de desarrollo de Django (`runserver`) puede tener algunas limitaciones. Prueba usar un servidor HTTP real como `gunicorn` para servir tu aplicación:

Instala `gunicorn`:

```
bash
Copiar código
pip install gunicorn
```

Ejecuta tu aplicación con `gunicorn`:

```
bash
Copiar código
gunicorn myproject.wsgi
```

6. Modificar `document_list.html`

Asegúrate de que tu HTML esté configurado correctamente para incrustar PDFs. Aquí hay un ejemplo simplificado usando solo un método para mostrar PDFs:

```
html
Copiar código
<!DOCTYPE html>
<html>
<head>
```

```
<title>Document List</title>
</head>
<body>
  <h1>Document List</h1>
  <ul>
    {% for document in documents %}
      <li>
        {{ document.title }} -
        <a href="{{ document.pdf.url }}" target="_blank">View PDF
in New Tab</a> -
        <a href="{{ document.pdf.url }}" download>Download
PDF</a>
        <br>
        <!-- Use only one method for embedding PDF -->
        <iframe src="{{ document.pdf.url }}" width="600"
height="400" style="border: none;"></iframe>
      </li>
    {% endfor %}
  </ul>
  <a href="{% url 'upload_document' %}">Upload new document</a>
</body>
</html>
```

Después de hacer estos cambios, reinicia tu servidor Django y prueba nuevamente. Con estas configuraciones, deberías poder ver y descargar PDFs sin problemas en diferentes navegadores.

40

```
from gunicorn import util File
"C:\Users\85124\AppData\Local\Programs\Python\Python39\lib\site-
packages\gunicorn\util.py", line 8, in <module> import fcntl ModuleNotFoundError: No
module named 'fcntl'
```

ChatGPT

El módulo `fcntl` es específico de sistemas Unix y no está disponible en Windows, por lo que `gunicorn` no puede ser usado directamente en Windows. Sin embargo, hay otras formas de ejecutar tu aplicación Django en Windows para probar la funcionalidad de servir archivos PDF correctamente.

Opciones para Ejecutar el Proyecto en Windows

1. **Usar waitress para Servir la Aplicación:** `waitress` es un servidor WSGI que funciona en Windows. Puedes instalarlo y usarlo de la siguiente manera:

Instala waitress:

```
bash
```

```
Copiar código
pip install waitress
```

Ejecuta tu aplicación con waitress:

```
bash
Copiar código
waitress-serve --port=8000 myproject.wsgi:application
```

2. **Seguir usando el servidor de desarrollo de Django (`runserver`):** Aunque tiene limitaciones, para pruebas locales y desarrollo debería ser suficiente. Asegúrate de que tu configuración de medios y archivos estática está correcta.

Revisión y Corrección del Código

Revisemos nuevamente el flujo completo para asegurar que no haya problemas en la configuración.

settings.py

Asegúrate de que tu configuración de medios está correcta:

```
python
Copiar código
import os

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

INSTALLED_APPS = [
    ...
    'corsheaders',
]

MIDDLEWARE = [
    ...
    'corsheaders.middleware.CorsMiddleware',
]
```

```
CORS_ALLOW_ALL_ORIGINS = True
```

```
X_FRAME_OPTIONS = 'ALLOWALL'
```

urls.py

Configura `urls.py` para servir archivos de medios durante el desarrollo:

```
python
Copiar código
from django.conf import settings
```

```
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]

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

document_list.html

Asegúrate de que tu plantilla `document_list.html` está configurada para intentar mostrar el PDF:

```
html
Copiar código
<!DOCTYPE html>
<html>
<head>
    <title>Document List</title>
</head>
<body>
    <h1>Document List</h1>
    <ul>
        {% for document in documents %}
            <li>
                {{ document.title }} -
                <a href="{{ document.pdf.url }}" target="_blank">View PDF
in New Tab</a> -
                <a href="{{ document.pdf.url }}" download>Download
PDF</a>
                <br>
                <!-- Use iframe to embed PDF -->
                <iframe src="{{ document.pdf.url }}" width="600"
height="400" style="border: none;"></iframe>
            </li>
        {% endfor %}
    </ul>
    <a href="{% url 'upload_document' %}">Upload new document</a>
</body>
</html>
```

Ejecutar el Proyecto

1. **Usar waitress:** Ejecuta el siguiente comando para servir tu aplicación:

```
bash
Copiar código
waitress-serve --port=8000 myproject.wsgi:application
```

2. **Usar runserver:** Ejecuta el siguiente comando para iniciar el servidor de desarrollo:

```
bash
Copiar código
python manage.py runserver
```

Pruebas

1. **Subir un Documento PDF:** Visita <http://127.0.0.1:8000/upload/> y sube un archivo PDF.
2. **Ver la Lista de Documentos:** Visita <http://127.0.0.1:8000/> para ver la lista de documentos y verificar que el PDF se muestra correctamente.

Si aún tienes problemas para mostrar el PDF, verifica las herramientas de desarrollo del navegador (F12) para ver mensajes de error específicos que puedan dar más pistas sobre el problema.

4o

ya se vio , por favor un resumen de lo que se izo ya que fueron muchos pasos y lo quiero aplicar a otra aplicacion ya formal

ChatGPT

Claro, aquí tienes un resumen de los pasos que tomamos para crear una aplicación Django que permite subir, ver y descargar documentos PDF, asegurando la compatibilidad entre diferentes navegadores.

Resumen de Pasos

1. Crear el Proyecto y la Aplicación Django

1. **Instalar Django y crear el proyecto:**

```
bash
Copiar código
pip install django
django-admin startproject myproject
cd myproject
```

2. **Crear una nueva aplicación:**

```
bash
Copiar código
python manage.py startapp myapp
```


2. Configurar el Proyecto

1. Añadir la aplicación a `INSTALLED_APPS` en `myproject/settings.py`:

```
python
Copiar código
INSTALLED_APPS = [
    # ...
    'myapp',
]
```

2. Configurar archivos de medios:

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

3. Añadir configuración CORS y `x-Frame-Options` en `settings.py`:

```
python
Copiar código
INSTALLED_APPS = [
    # ...
    'corsheaders',
]

MIDDLEWARE = [
    # ...
    'corsheaders.middleware.CorsMiddleware',
]

CORS_ALLOW_ALL_ORIGINS = True
X_FRAME_OPTIONS = 'ALLOWALL'
```

4. Configurar URLs para servir archivos de medios en `myproject/urls.py`:

```
python
Copiar código
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]

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

3. Crear el Modelo y el Formulario

1. Definir el modelo `Document` en `myapp/models.py`:

```
python
Copiar código
from django.db import models

class Document(models.Model):
    title = models.CharField(max_length=100)
    pdf = models.FileField(upload_to='documents/')

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

2. Crear el formulario `DocumentForm` en `myapp/forms.py`:

```
python
Copiar código
from django import forms
from .models import Document

class DocumentForm(forms.ModelForm):
    class Meta:
        model = Document
        fields = ('title', 'pdf')
```

4. Crear las Vistas

1. Definir las vistas `upload_document` y `document_list` en `myapp/views.py`:

```
python
Copiar código
from django.shortcuts import render, redirect
from .forms import DocumentForm
from .models import Document

def upload_document(request):
    if request.method == 'POST':
        form = DocumentForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('document_list')
    else:
        form = DocumentForm()
    return render(request, 'myapp/upload_document.html', {'form': form})

def document_list(request):
    documents = Document.objects.all()
    return render(request, 'myapp/document_list.html', {'documents': documents})
```

5. Crear las Plantillas

1. Crear la plantilla `upload_document.html`:

```
html
Copiar código
<!DOCTYPE html>
<html>
<head>
    <title>Upload Document</title>
</head>
<body>
    <h1>Upload Document</h1>
    <form method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Upload</button>
    </form>
    <a href="{% url 'document_list' %}">Back to document list</a>
</body>
</html>
```

2. Crear la plantilla `document_list.html`:

```
html
Copiar código
<!DOCTYPE html>
<html>
<head>
    <title>Document List</title>
</head>
<body>
    <h1>Document List</h1>
    <ul>
        {% for document in documents %}
            <li>
                {{ document.title }} -
                <a href="{{ document.pdf.url }}"
target="_blank">View PDF in New Tab</a> -
                <a href="{{ document.pdf.url }}" download>Download
PDF</a>
                <br>
                <iframe src="{{ document.pdf.url }}" width="600"
height="400" style="border: none;"></iframe>
            </li>
        {% endfor %}
    </ul>
    <a href="{% url 'upload_document' %}">Upload new document</a>
</body>
</html>
```

6. Configurar las URLs

1. Configurar las URLs de la aplicación en `myapp/urls.py`:

```
python
Copiar código
from django.urls import path
from .views import upload_document, document_list

urlpatterns = [
    path('upload/', upload_document, name='upload_document'),
    path('', document_list, name='document_list'),
]
```

2. Incluir las URLs de la aplicación en `myproject/urls.py`:

```
python
Copiar código
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]
```

7. Ejecutar el Servidor

1. Ejecutar el servidor de desarrollo de Django:

```
bash
Copiar código
python manage.py runserver
```

2. Alternativa: Usar `waitress` para servir la aplicación en Windows:

```
bash
Copiar código
pip install waitress
waitress-serve --port=8000 myproject.wsgi:application
```

Pruebas

1. **Subir un Documento PDF:** Visita `http://127.0.0.1:8000/upload/` y sube un archivo PDF.
2. **Ver la Lista de Documentos:** Visita `http://127.0.0.1:8000/` para ver la lista de documentos, ver y descargar los PDFs.

Este resumen debería ayudarte a replicar el proceso en cualquier otra aplicación Django.

4o

y cuales so las aplicacione secundarias que debo instalar

ChatGPT

Para asegurarte de que tienes todas las dependencias necesarias para tu proyecto Django, incluyendo las aplicaciones secundarias que mencionamos, aquí hay una lista de paquetes que debes instalar:

1. **Django:** El marco web principal.
2. **django-cors-headers:** Para manejar las políticas de CORS.
3. **waitress:** Un servidor WSGI para ejecutar Django en Windows.

Instalación de Dependencias

Puedes instalar todas estas dependencias usando `pip`. Aquí hay un resumen de los comandos que necesitas ejecutar:

1. Instalar Django:

```
bash
Copiar código
pip install django
```

2. Instalar django-cors-headers:

```
bash
Copiar código
pip install django-cors-headers
```

3. Instalar waitress:

```
bash
Copiar código
pip install waitress
```

Archivo `requirements.txt`

Para gestionar estas dependencias de manera más fácil, especialmente si estás trabajando en un equipo o en diferentes entornos, es útil crear un archivo `requirements.txt`. Aquí hay un ejemplo de cómo debería verse:

```
plaintext
Copiar código
django
django-cors-headers
waitress
```

Luego, puedes instalar todas las dependencias listadas en el archivo `requirements.txt` con un solo comando:

```
bash
Copiar código
pip install -r requirements.txt
```

Resumen de Configuraciones y Pasos Adicionales

1. Configurar `INSTALLED_APPS` y `MIDDLEWARE` en `settings.py`:

```
python
Copiar código
INSTALLED_APPS = [
    ...
    'corsheaders',
    'myapp',
]

MIDDLEWARE = [
    ...
    'corsheaders.middleware.CorsMiddleware',
    ...
]

CORS_ALLOW_ALL_ORIGINS = True

X_FRAME_OPTIONS = 'ALLOWALL'
```

2. Configurar URLs para servir archivos de medios:

```
python
Copiar código
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]

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

Comandos Esenciales para Ejecutar el Proyecto

1. Realizar migraciones y aplicar cambios a la base de datos:

```
bash
Copiar código
python manage.py makemigrations
python manage.py migrate
```

2. Crear un superusuario para el panel de administración:

```
bash
Copiar código
python manage.py createsuperuser
```

3. Ejecutar el servidor de desarrollo:

```
bash
Copiar código
python manage.py runserver
```

4. Ejecutar el servidor con waitress (en Windows):

```
bash
Copiar código
waitress-serve --port=8000 myproject.wsgi:application
```

Siguiendo estos pasos y configuraciones, deberías tener un entorno Django completamente funcional que pueda manejar la subida, visualización y descarga de archivos PDF, compatible con diferentes navegadores.

40