DJANGO

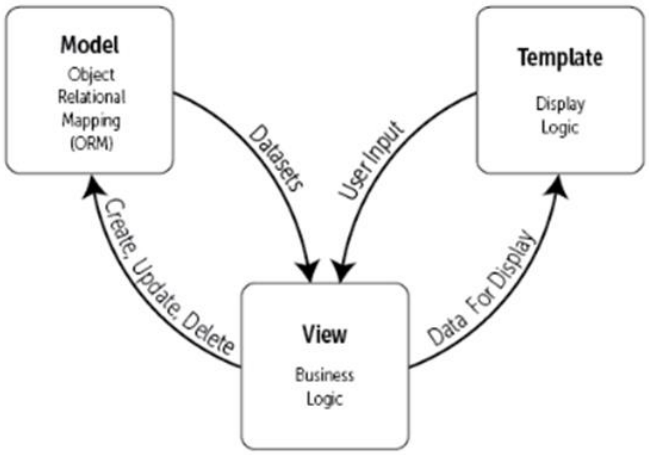
Documentación oficial

En ella encontrará mucha información y ejemplos de sintaxis.

https://www.djangoproject.com/

Patrón de diseño de Django

Hay tres componentes principales en la **arquitectura de Django**: elementos que ayudan a trabajar con la base de datos, un sistema de plantillas que funciona para personas que no programan y un marco que automatiza gran parte de la administración del sitio web. Esto se presta a un patrón de diseño **Modelo**, **Vista**, **Template**:



- **Modelo**: define la estructura de la base de datos.
- **Vista**: define la lógica que devuelve algo de una solicitud HTTP.
- **Template**: define la estructura de cómo se verá una página web.

¿Cómo instalar Django?

Dentro el símbolos de sistema (cmd) o en una terminal de visual studio code ejecutar uno de estos dos comandos, depende de las versiones de python que tenga es cual va a funcionar.

- **pip install django**
- **python –m pip install Django**

Pueden verificar si la instalación se ejecutó correctamente con el comando:
python -m django --version

Proyecto django

- **¿Qué es un proyecto django?**
  Es una colección de archivos y configuraciones necesarias para una aplicación web específica, incluida la configuración de la base de datos, opciones específicas de Django y configuraciones específicas de la aplicación.
  Por lo general, un proyecto Django consta de al menos un sitio web (aunque puede tener más de uno) y puede incluir varias aplicaciones que se encargan de diferentes aspectos o funcionalidades de ese sitio.

- **Crear un proyecto django**
  Dentro de símbolos de sistema (cmd) o en una terminal de visual studio code nos posicionamos en donde queremos que se cree el proyecto utilizando el comando **cd nombre o ruta a la que queremos acceder** en caso de no estar en la ubicación.
  Una vez ubicados ejecutaremos uno de los siguiente comandos:
  - **django-admin startproject nombreProyecto**
  - **python –m django startproject nombreProyecto**

  cambiando el texto de nombreProyecto por el que utilizaremos.

  Una vez ejecutada se creará la siguiente estructura:

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

  - El directorio raíz externo **mysite/** es un contenedor para su proyecto. Su nombre no le importa a Django; por lo que se puede cambiar el nombre.
  - **manage.py**: una utilidad de línea de comandos que te permite interactuar con el proyecto Django.
  - El **directorio interno mysite/** es el paquete Python real para el proyecto. Su nombre es el que se usa para importar cualquier cosa que contenga (p. ej mysite.urls.).
  - **__init__.py**: un archivo vacío que le dice a Python que este directorio debe considerarse un paquete de Python.
  - **settings.py**: Ajustes/configuración del proyecto Django. Como:
    INSTALLED_APPS es donde se configuran cada una de las aplicaciones que se crearon.
    DATABASE es la configuración de la base de datos que va a usar el proyecto.
    STATIC_URL indica donde están los archivos static son los CSS, JS.
  - **urls.py**: Las declaraciones de URL del proyecto Django; una “tabla de contenidos” del sitio.
  - **wsgi.py**: información para desplegar el proyecto en producción.

- **Servidor de desarrollo**
  Django cuenta con un servidor con el que se pueden realizar las pruebas durante el desarrollo del proyecto.
  Para utilizarlo dentro de la terminal ejecutar el siguiente comando:
    - **python manage.py runserver**

  El servidor de desarrollo recarga automáticamente el código Python para cada solicitud según sea necesario. No es necesario reiniciar el servidor para que los cambios de código surtan efecto. Sin embargo, algunas acciones como agregar archivos no activan un reinicio, por lo que tendrás que reiniciar el servidor en estos casos.

- **Crear una aplicación en el proyecto**
  Cada aplicación dentro de un proyecto Django se encarga de una función específica, como por ejemplo la gestión de usuarios, la autenticación, la administración del contenido, etc.
  Para crear una aplicación en la terminal ejecutar el siguiente comando:
    - **python manage.py startapp nombreAplicacion**

  Se creará la siguiente estructura en el proyecto

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

  *Dentro de la aplicación crearemos las carpetas **templates** (en esta carpeta van todos los html) y **static** (en esta carpeta irán los archivos estáticos como css js imagenes), además agregar un archivo llamado **urls.py** (se guardaran las urls de la aplicación)*

  Una vez creada la aplicación es necesario incluirla en el proyecto, para esto se necesita agregar una referencia dentro del archivo **setting.py** en la parte de **INSTALLED_APPS.** De la siguiente manera:

  INSTALLED_APPS = ["nombreAplicación",]

  RECUERDEN NO OLVIDAR LA COMA DEL FINAL

- **views.py**: en este se definen las vistas de la app.
- **urls.py**: en este establecemos los path indicando la URL donde vamos a enlazar las vistas creadas.

Plantillas: Templates

Django nos ofrece la posibilidad de utilizar plantillas HTML repletas de funcionalidades.
Para utilizar una plantilla debemos:

1) Crear una carpeta **templates** dentro de cada aplicación que nuestro proyecto tenga para que se mantenga un orden y sea más sencillo editarlo en caso de modificaciones o errores.
   Tenemos que hacerlo así porque Django funciona mezclando los directorios templates de las apps, de manera que al final él tiene un solo directorio templates y dentro otro para cada app.
2) Crear los archivos HTML que utilizaremos.

En el archivo views.py para devolver los templates HTML se utiliza el método render del módulo http de Django, que ya viene incluido por defecto.

Por ejemplo:

```
def home(request):
   return render(request, "core/home.html")
```

HERENCIA DE PLANTILLAS

1. Lo primero que se hace es crear una plantilla base, dentro de esta vamos a poner todo aquello que se repetira en todas las plantillas de nuestro sitio.

2. En el lugar que ocupará la parte que cambia en cada página vamos a escribir:
   {% block nombreBloque%} {% endblock %}
   Esto es un template tag, una etiqueta de template, que sirve para añadir lógica de programación dentro del HTML.
   El template tag ***block*** sirve para definir un bloque de contenido con un nombre.
3. En los template que queremos aplicar el contenido de base debemos poner al inicio:
   {% extends 'core/base.html' %}, esto indica que la plantilla es hija de base.
   El contenido propio de la plantilla hija, lo pondremos entre los tag **{% block nombreBloque%} {% endblock %}**

La estructura nos quedaría de la siguiente manera:

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

{% block content %}
   <h2>Mi Web</h2>
   <p>Bienvenidos.</p>
{% endblock %}
```

Template tag {% url %}

Este tag nos permite hacer referencia directamente a una view desde templates y es la forma correcta de escribir enlaces relativos dentro de la web.
Este tag se escribe: **{% url 'nombre_en_el_path %'}**
Dentro de urls.py añadir al path el parámetro name que será el que utilizaremos en el tag.
Esto hace que si en urls.py cambiamos una dirección la url se actualizará y seguirá funcionando.
Ejemplo de uso:

```
<!DOCTYPE html>
<head>
```

```
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <ul>
        <li><a href="{% url 'about' %}">Portada</a></li>
        <li><a href="{% url 'portafolio' %}">Acerca de</a></li>
    </ul>
    {% block content %}{% endblock %}
</body>
</html>
```

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

urlpatterns = [
    path('about-me/', views.about, name="about"),
    path('portfolio/', views.portfolio, name="portfolio"),
]
```

Archivos estáticos

Dentro de lo que son los archivos estáticos incluiremos los css, imágenes, javascript, etc.
Para utilizarlos debemos:

1. Crear una carpeta static dentro de nuestra aplicación asi como lo hicimos con templates.
2. Dentro de la carpeta crearemos subcarpetas donde pondremos cada uno de estos tipos de contenidos. Por ejemplo: css, imágenes, etc.
3. Por último se indicará en las plantilla que se carguen los ficheros estáticos usando el tag: **{% load static %}**
   Podemos realizarlo directamente en nuestro archivo base para que lo herede y no será necesario hacerlo en todas las plantillas.
   Para utilizar lo que tenemos en esa carpeta hacerlo de la siguiente manera:
   {% static 'adopcion400.png' %}

VISTAS

Dentro del archivo views.py se crearán todas las vistas correspondientes a nuestras aplicaciones.
Una vista hace referencia a la lógica que se ejecuta cuando se hace una petición a nuestra web. Estas son funciones de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP. Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de modelos, y delegan el formateo de la respuesta a las plantillas ("templates").
Una vista puede devolver un mensaje plano de la siguiente manera:

- **Módulo django.http llamado HttpResponse:**
  from django.shortcuts import render, HttpResponse
  Este método que nos permite contestar a una petición devolviendo un código
  return HttpResponse("<h1>Mi Web Personal</h1><h2>Portada</h2>")
  Esto nos permite devolver HTML plano.

El request recibe información del cliente o sitio. Para responder al cliente o sitio se envía una respuesta httpResponse para esto es necesario importar from django.http import HttpResponse.
Por ejemplo:

```
def about(request):
    return HttpResponse("About")
```

o un template:

- En el archivo views.py para devolver los templates HTML se utiliza el método render del módulo http de Django, que ya viene incluido por defecto.

Por ejemplo:

```
def saludo(request):
    return HttpResponse("HOLA")

def inicio(request):
    return render(request,'index.html')
```

En el archivo de vistas vamos a crear una clase por cada página que tengamos, recordar pasar como parámetro request.

URLS

En cada aplicación crearemos el archivo urls.py donde pondremos las url propias de la app, dentro del contenido de este archivo copiaremos lo que está en el urls del proyecto.

1. Dentro del archivo urls.py de la aplicación lo primero que haremos es importar todas las views:
   from . import views
   o
   from nombreApp import views
2. Crear una url por cada vista que tengamos, como por ejemplo:
   path('Inicio/',views.inicio)
   NO OLVIDAR POR UNA COMA AL FINAL DEL ULTIMO PATH CREADO
3. Dentro de urls del proyecto vamos a añadir en la importación del path include quedando de la siguiente manera: ***from django.urls import path, include***
4. Y para finalizar en urlpatterns agregar las url de cada aplicación usando el include como en siguiente ejemplo:

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

Para cerrar en esta parte de la configuración del proyecto se debe:

- *Configurar las plantillas: para esto creamos las carpetas template y static, en cada una de estas poner los archivos necesarios.*
- *Configurar las vistas: se crean vistas para cada parte de la app.*
- *Configurar la url: se creará la url de cada parte de la aplicación.*

Modelos

Un modelo es el esquema de base de datos, para cada tabla de la base de datos se creará una clase la cual contendrá los atributos de la tabla con su tipo.

- Los atributos se definirán con un nombre y un tipo de dato (usando la clase models). Cada atributo será un elemento de la tabla.
- Si se quiere definir un atributo que tenga relación con otra tabla usar: proyecto = models.ForeignKey(Proyecto, on_delete=models.CASCADE) con el on_delete le digo que si se elimina algo de la tabla con la que está relacionado esto se elimine en cascada

Cuando el tipo de datos es CharField debe tener definido un atributo max_length siempre. *Recordar que al crear una clase esta debe heredar de una clase padre llamada models.Model. Y es importante que el nombre de la clase siempre sea en singular ya que cuando se guarden los cambios en las base de datos se le agrega una s al final automáticamente.*

Ejemplo:

```
class Project(models.Model):
    title = models.CharField()
    description = models.TextField()
    image = models.ImageField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
```

Una vez creados los modelos tenemos que crear y guardar estos cambios en la base de datos. Para esto utilizaremos los comando:

1) **python manage.py makemigrations** que sirve para indicar a Django que hay cambios en algún modelo, y se creará un fichero de migraciones. (si al final añadimos el nombre de la aplicación de la que queremos hacer las migraciones solo tomara esas, sino migrara todo lo que este en install_app del setting.py)
2) **python manage.py migrate** que sirve para guardar los cambios detectados con el comando anterior.

BASES DE DATOS QUE ADMITE DJANGO

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

Por defecto la base de datos con la que inicia el proyecto es sqlite3, para configurar otras bases de datos se puede cambiar las instrucción alguna de las siguientes:

- `'django.db.backends.postgresql'`
- `'django.db.backends.mysql'`
- `'django.db.backends.sqlite3'`
- `'django.db.backends.oracle'`

PANEL DE ADMINISTRADOR

Lo primero que necesitamos para manejar el panel es crear un superusuario, para hacerlo debemos utilizar el siguiente comando:
python manage.py createsuperuser
Esto nos solicitará un nombre de usuario, correo y contraseña.
En caso de necesitar cambiar la contraseña utilizar el siguiente comando:
py manage.py changepassword nombreUsuario

Ahora para que podamos administrar las aplicaciones del proyecto debemos en cada archivo admin.py correspondiente a cada aplicacion importar los modelos que queremos incluir en el panel como en el ejemplo:

```
from .models import Categoria, Post

class CategoriaAdmin(admin.ModelAdmin):
    list_display=('nomProyecto','descripcion')
```

admin.site.register(Categoria, CategoriaAdmin) → de esta manera podremos personalizar la forma en que se ve cada modelo.

admin.site.register(Post) → si lo indicamos de esta manera seguirá la función str creada dentro de la clase en el modelo.

PARA PERSONALIZAR EL NOMBRE QUE SE MUESTRA EN EL PANEL, ASI EN LUGAR DE USAR EL VALOR DE LA BASE SE UTILIZA EL PARAMETRO verbose_name EN EL MODELO: ***f_tarea=models.DateField(verbose_name="Fecha Entrega Tarea")***

FORMULARIOS

Crear un archivo.py donde utilizaremos la herramienta de django para forms, importarla de la siguiente manera:

from django import forms

Dentro de este haremos una clase con los campos como en modelo, cambiando por form y los tipos de campos de los formularios, por ejemplo:

class FromNuevaTareas(forms.Form):

title = forms.CharField(label='Título de la Tarea', max_length=200)

descripcion = forms.Textarea(label='Descripcion de la tarea',required=False)

Despues dentro de la vista importamos este nuevo archivo

<form method="POST">{% csrf_token%}

{{form}}

<button>Guardar</button>

</form>

en las vistas se pueden usar métodos como:

request.GET['campo']

request.GET

De esta forma consultamos que envía el formulario

Tareas.objects.create(title=request.POST['title'], descripcion = request.POST['descripcion'],projecto_id=1)

return redirect('/Tasks2/')

Se utiliza el post para guardar

USO DE IMÁGENES DE BASE DE DATOS EN WEB

1) Crear una carpeta en el proyecto llamada ***media***
2) En settings.py configurar:
   a. **MEDIA_URL = '/media/'**
   b. **MEDIA_ROOT = os.path.join(BASE_DIR, 'media')**
   c. Sino función el punto b usar: **MEDIA_ROOT = BASE_DIR/'media'**
3) En urls.py añadir lo siguiente
   from django.conf import settings
   from django.conf.urls.static import static

   Al final de todo añadir
   Urlpatterns += static(setting.MEDIA_URL, document_root=setting.MEDIA_ROOT)

4) Ahora para mostrar en el template usar el tag de esta manera:
   <img src="{{servicio.imagen.url}}">

ENVIO DE CORREO ELECTRONICO

1) En su casilla de correo electrónico buscar el código, para esto ingresar en la configuración de seguridad y busca "contraseñas de aplicación". Le ponemos un nombre a esta clave y la copiamos el código en EMAIL_HOST_PASSWORD.

2) EMAIL_BACKEND="django.core.mail.backends.smtp.EmailBackend"
   EMAIL_HOST="smtp.gmail.com"
   EMAIL_USE_TLS = True
   EMAIL_PORT= 587
   EMAIL_HOST_USER="programacion.emilse@gmail.com"
   EMAIL_HOST_PASSWORD="ssss sss ssss " → código sacado del correo en paso 1

3) En form.py Crear el formulario e incrustarlo en la página web

4) En la views.py:

```
if request.method=="POST":
    miFormulario = FormularioContacto(request.POST) →formulario creado en form
    #llevan validacion implicita
    if miFormulario.is_valid():
        #si el formulario es válido almaceno la información cargada
        infoForm=miFormulario.cleaned_data

        #Los parámetros de send_mail(asunto, mensaje, remitente, destinatario
        que va como lista)

        #ENVIO CORREO A CASILLA DEL SITIO
        send_mail(infoForm['asunto'],infoForm['mensaje'],
                infoForm.get('email',''),['programacion@gmail.com'])

        #ENVIO CORREO RESPUESTA AUTOMATICA AL USUARIO
        send_mail('prueba','respuestas automatica', settings.EMAIL_HOST_USER,
        [infoForm.get('email','')])
        return render(request,"gracias.html")
else:
    #no se envio todavia crea un formulario vacio
    miFormulario=FormularioContacto()
```