1. **INSTALACIÓN**:
   1. Instalar Python.
   2. Actualizar pip con **> python -m pip install --upgrade**
   3. Crear un entorno de trabajo dentro de un directorio con:
      * **python -m venv <nombre\_entorno>**
   4. Activar el entorno ingresando a la carpeta **Script** y hacer:
      * **activate**
   5. Una forma de ver que el entorno esta activado es que en el prompt aparece el nombre del entorno entre paréntesis.
   6. Instalamos **django** con:
      * **pip install django**
   7. Para verificar que se instaló correctamente el django, verificamos los paquetes instalados con:
      * **pip freeze --local**
2. Crear un proyecto nuevo:
   1. Primero activar el entorno.
   2. Fuera del entorno (carpeta) se debe crear el proyecto.
   3. Usar el comando siguiente para crear el nuevo proyecto:
      * **django-admin startproject <nombre\_proyecto>**
   4. Para validar que esta todo correcto, o cada vez que se hacen modificaciones se puede verificar dentro del directorio del proyecto (ejecutar el proyecto) con:
      * **python manage.py runserver**
   5. Aparecerá la dirección del navegador donde se puede verificar el proyecto.
3. Archivos principales:
   1. **Manage.py**: Archivo principal, ejecuta el código.
   2. **Db.sqlite3**: Base de datos por defecto.
   3. **\_\_init\_\_.py**: Archivo de referencia, para indicar que en la carpeta donde está el archivo existe código que el sistema debe leer.
   4. **Settings.py**: Archivo de configuración, por ejemplo, el idioma.
   5. **Urls.py**: Se encuentran las URLS del proyecto.
   6. **Wsgi.py**:
4. Organizar **settings.py**, ya que al crecer este archivo va a costar mucho mantener, entonces:
   1. Crear carpeta llamada **settings**.
   2. Dentro, crear 4 archivos.
   3. **base.py**: contiene la configuración que todos deben tener, todos heredan este archivo.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

* 1. **local.py**: contiene la configuración cuando se trabaja en local.

Texto

Descripción generada automáticamente

* 1. **prod.py**: contiene la configuración cuando se trabaja en producción, en un servidor alojado.
  2. **\_\_init\_\_.py:** archivo que indica a Django que hay código que considerar.
  3. Dentro de local.py importar base.py con **from .base import \***
  4. Para ejecutar la aplicación con la nueva distribución de settings se hace:

**python manage.py runserver --settings=<nombre\_proyecto>.settings.local**

* 1. Para no tener que escribir la línea de arriba cada vez que se ejecuta la aplicación, se debe editar **manage.py** y agregar el nuevo settings, por ejemplo:

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

* 1. Con esto ya se puede ejecutar normal.

1. Las **Apps** en Django es dividir el proyecto en pequeñas partes llamas **apps**.
2. Un **Apps** es para cubrir una necesidad, una Apps por cada necesidad.
3. Las Apps permite que cada una sea independiente y reutilizable.
4. Antes de crear las Apps hay que definir el prototipo y la base de datos.
5. Para crear una Apps:
   1. Activar el entorno de trabajo.
   2. Crear carpeta dentro del proyecto llamada, por convención, **applications**.
   3. Ingresar a la carpeta y crear el fichero **\_\_init\_\_.py**
   4. Dentro de la carpeta y por consola crear las Apps con:

**django-admin startapp <nombre\_aplicación>**

* 1. Dentro de **base.py** hay que incluir las aplicaciones creadas:

Texto

Descripción generada automáticamente

* 1. Dentro de cada Apps, ingresar a **apps.py**, y agregar la ruta correctamente, por ejemplo:

**Texto

Descripción generada automáticamente**

* 1. Indicar que cada apps tenga sus propias URL, para esto, en cada App crear archivo **urls.py**. Y agregar el código inicial:

**Texto

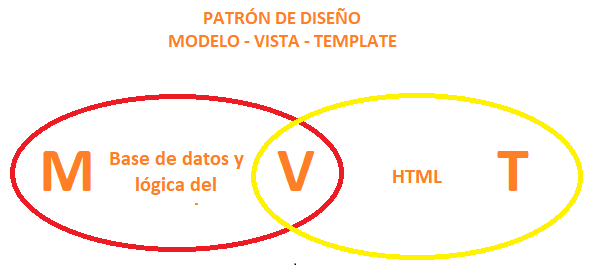
Descripción generada automáticamente**

* 1. Para que detecte las nuevas URL, en el **urls.py** principal, agregar los ficheros urls creados, importar **include**:

Texto

Descripción generada automáticamente

1. Django usa el patrón de diseño MVT, Modelo – Vista – Template. Donde Modelo es la base de datos, la Vista es la lógica del negocio y preparar todo para que el Template pinte en pantalla.
2. Algo sobre template:
   1. Después de configurar la app, en el fichero views.py, se llama a los template.



* 1. Django usan Vistas basadas en clases que permite resumir y reutilizar el código, y trabajar bajo un estándar.
  2. Para evitar que se escriba tantas líneas de código se usan las vistas genéricas.
  3. Por ejemplo, en views.py de la aplicación, importamos una vista genérica. para llamar a un fichero html:

Texto

Descripción generada automáticamente

* 1. Dentro del fichero urls.py que se creó en la App, le indicaremos la ruta del fichero html (el punto indica que está en la misma carpeta), **as\_view** indica que estamos usando una vista genérica:

Texto

Descripción generada automáticamente

* 1. Agregar el urls.py de la App a la urls.py principal:

Texto

Descripción generada automáticamente

* 1. Crear dentro de la App una carpeta llamada **templates**, y crear los ficheros html para que Django los encuentre.
  2. La vista genérica **TEMPLATEVIEW** siempre va a tener un **template\_name** siempre va a apuntar a un fichero html que se encuentra dentro de la carpeta **templates**.
  3. Las vistas **LISTVIEW**, **CREATEVIEW**, **UPDATEVIEW** y **DELETEVIEW** va a necesitar un **template\_name**, **model**, y **consulta a bd**.

1. Para usar una sola carpeta **templates** en todas las aplicaciones, se hace, por ejemplo:
   1. Crear una carpeta **templates** en la raíz.
   2. Instalar el paquete **unipath** por **pip**.
   3. Dentro de base.py modificar la línea y reemplazarla por:



Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

* 1. Donde ancestor(3), retrocede 3 directorios para llegar a la carpeta raíz.
  2. Dentro de base.py, en **TEMPLATE** modificar **‘DIRS’: [ ]**, y dejarla:



* 1. Para que el sqlite3 no arroje error, en local.py modificar DATABASES y dejar:

Texto

Descripción generada automáticamente

* 1. Eliminar las carpetas templates antiguas.
  2. Dentro de la carpeta templates, crear una carpeta por application para ordenar los templates.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* 1. Dentro de **views.py** cambiar el **template\_name** por:



1. Más vistas genéricas:
   1. **ListView**: se usa para listar algo, por ejemplo, elementos de una base de datos. Su estructura es:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

* + - Por ejemplo, si no tenemos una base de datos vamos a usar una **queryset**
    - Dentro de la aplicación, en views.py importamos **ListView**:

Texto

Descripción generada automáticamente

* + - La clase quedaría:

Texto

Descripción generada automáticamente

* + - Agregaremos esta lista en urls:

Texto

Descripción generada automáticamente

* + - Para acceder a la lista se crea una variable:

Texto

Descripción generada automáticamente

* + - El HTML queda:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. Models para vistas: Se usa ORM que transforma a código SQL.
   1. Para crear una tabla de la base de datos, en el archivo models.py de la APP colocar, por ejemplo:

Texto

Descripción generada automáticamente

* 1. Luego por consola hacer:
     + Verificar si hay cambios en la DB:



* + - Guardar cambios en la DB:



* + - En admin.py hay que registrar la clase haciendo, por ejemplo:

Texto

Descripción generada automáticamente

* + - Ir al admin <http://127.0.0.1:8000/admin> y crear un usuario con:



* + - Volver a revisar el admin.

1. Base de Datos en Django – Models.py:
   1. Tipos de campos de un modelo:
      * Para revisar todos los tipos de datos:

<https://docs.djangoproject.com/en/4.0/ref/models/fields/>

* + - Django internamente genera las ID de los campos, no hay que colocarlo.
    - Para que no sea obligatorio el campo, depende del tipo, pero por ejemplo en un campo de texto se usa **blank=True**.
    - Para que un campo no se repita se usa **unique=True**.
    - Para que un campo no pueda editarlo se usa **editable=False**.
    - Ejemplo de la creación de un modelo (tabla) en el archivo models.py:



Texto

Descripción generada automáticamente

* + - Para guardar cambios y que la tabla se cree en el motor de base de datos hacer:
      1. Verificar si hay cambios o errores:



* + - 1. Generar los cambios:



* + - Hay que registrar el modelo en admin.py:

Texto

Descripción generada automáticamente

Donde Departamento es en este caso el nombre del modelo.

* + - Para que Django reconozca los modelos la App debe estar registrada en base.py.
  1. Claves foráneas:
     + Para generar la clave se importa el modelo de las clases a relacionar, ejemplo:



* + - Usar **ForeingKey**, como en el ejemplo:



* + - Ejemplo de un modelo (tabla Empleado):



* 1. Instalación de PostgreSQL:
     + En Windows el ejecutable solo pide la clave para el usuario administrador.
     + Por consola:
       1. Crear base de datos: **CREATE DATABASE <nombre\_db>;**
       2. Listar base de datos: **\l** o **\l+**
       3. Usar una base de datos: **\c <nombre\_db>;**
       4. Crear usuario: **CREATE USER <nombre\_usuario>;**
       5. Cambiar contraseña de usuario: **ALTER USER <nombre\_usuario> WITH PASSWORD ‘<contraseña>’;**
       6. Asignarlo usuario a la base de datos, primero ingresar a la base de datos con **\c <nombre\_db>**, y luego **ALTER ROLE <nombre\_usuario> WITH PASSWORD ‘<contraseña>’;**
       7. Listar usuario con **\du**
  2. Conectar Django a PostgreSQL:
     + Instalar por pip **psycopg2**.
     + Dentro de settings > local.py configurar el PostgreSQL:

Texto

Descripción generada automáticamente

* + - Cuando entra a producción, se configura la conexión en settings > prod.py.
    - Antes de migrar a la nueva base de datos hay que limpiar la carpeta migrations de cada App.
    - Realizar la migración por DOS con **#python manage.py migrate**
    - El usuario creado antes para ingresar al admin de Django ya no va a servir porque cambiamos de DB. Creamos uno nuevo con **#python manage.py createsuperuser**.
    - Revisamos en el admin de Django si se crearon las tablas.

1. El administrador de Django:
   1. Class Meta, modelos y en el administrador de Django:
      * Class Meta: Son modificaciones que se pueden hacer en el administrador, va dentro del modelo (clase dentro de una clase). Ejemplo:

Texto

Descripción generada automáticamente

* 1. Personalizar tablas en el administrador de Django: