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: video 17

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

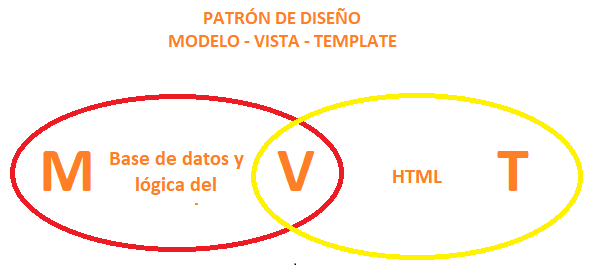
Descripción generada automáticamente**

* 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. Por ejemplo, en views.py de la aplicación, para llamar a un fichero html:

Texto

Descripción generada automáticamente

* 1. Dentro del fichero urls.py que se creo en la App, le indicaremos la ruta del fichero html:

Texto

Descripción generada automáticamente

* 1. Agregar el urls.py del 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 **template**.
  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. Eliminar las carpetas templates antiguas.
  2. Dentro de la carpeta templetes, 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