TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Archivos de Configuración.
- __init__.py
- settings.py
- Manejo de espacios de nombre.
- Modelo MVC y la creación de aplicaciones.
- Correlación del modelo.

ARCHIVOS DE CONFIGURACIÓN

__INIT__.PY

Python proporciona un sistema de empaquetado muy sencillo, que es simplemente una extensión del mecanismo del módulo a un directorio. Cualquier directorio con un archivo `__init__.py` se considera un paquete de Python. Los diferentes módulos del paquete se importan de manera similar a los módulos simples, pero con un comportamiento especial para el archivo `__init__.py`, que se usa para recopilar las definiciones de todo el paquete.

La siguiente imagen muestra la estructura de un módulo Python estándar.

```
package/
    __init__.py
    myprog1.py
    myprog12.py
    myprog13.py
    subpackage/
        __init__.py
        mysubprog1.py
        mysubprog2.py
        mysubprog2.py
```

Como se puede observar, en la estructura de un módulo estándar de Python en la imagen de arriba, la inclusión del archivo `__init__.py` en un directorio le indica al intérprete de Python que éste debe tratarse como un paquete de Python. Cuando se importa un módulo a un script, el archivo `__init__.py` de ese módulo se generará y ejecutará. Proporciona una manera fácil de agrupar carpetas grandes de muchas secuencias de comandos de Python, separadas en un solo módulo importable.

Un paquete se puede importar con `import foo.bar`, o con `foo import bar`. Para que un directorio forme un paquete, debe contener un archivo especial `__init__.py`, incluso si este archivo está vacío. Si elimina el archivo `__init__.py`, Python ya no buscará submódulos dentro de ese directorio, por lo que los intentos de importar el módulo fallarán. Dejar un archivo `__init__.py` vacío se considera normal, e incluso una buena práctica, siempre que los módulos y subpaquetes del paquete no necesiten compartir ningún código.

Una aplicación de Django es un paquete de Python, que está diseñado específicamente para su uso en un proyecto de Django. Una aplicación puede usar convenciones comunes de Django, como tener submódulos de modelos, pruebas, URL y vistas.

SETTINGS.PY

Generalmente, cuando creamos algún proyecto, el o los archivos de configuración siempre son los más importantes. En el caso particular de Django, al momento de generar un proyecto consigo se crea un archivo llamado settings.py, que es nuestro archivo de configuración.

El `settings.py` es, sin duda, uno de los archivos más importantes de todo el proyecto. Toda la configuración que usará Django en el proyecto estará dentro de este archivo, por lo que se debe tener mucho cuidado al momento de manipularlo. Por defecto, Django trae sólo un archivo `settings.py`.

Este es el archivo `settings.py` que Django crea por defecto, toda la configuración siempre está asociado al nombre de una variable, así que al momento de hacer las modificaciones siempre se toma en cuenta, y las variables no pueden ser modificadas porque Django simplemente ya no las tomará en cuenta. A continuación se explicarán las configuraciones más importantes que trae ese archivo.

BASE_DIR

Esta es la variable que tiene por contenido la ruta a una determinada zona de tu proyecto. Por defecto, lleva la ruta al archivo settings.py. Apunta a la jerarquía superior del proyecto, es decir, mi web del proyecto, y cualquier ruta que definamos en el proyecto es relativa a `BASE_DIR`.

```
# settings.py
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
```

SECRET_KEY

Todo proyecto en Django tiene esta variable que lleva una clave secreta, y su contenido siempre debe estar protegido y nunca escrito como texto plano dentro del archivo. Por defecto, Django la establece así, pero ya es trabajo del desarrollador protegerla.

```
# settings.py
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY   =   'django-insecure-b^k75h*ti0b(%wjap73)8ms^xgus(msw6)bkdi7hqa-
2kfpn1%'
```

DEBUG

Django proporciona un depurador integrado que facilita mucho la vida del desarrollador. `DEBUG` proporciona una variable booleana que sólo acepta `True` o `False`. True cuando tu proyecto está en modo de depuración, que se nota fácilmente cuando sale algún error y ves la pantalla amarilla de Django donde te muestra el detalle del error obtenido. Y `False` cuando el proyecto ya no se encuentra en modo de depuración, esto se suele utilizar cuando el proyecto ya se encuentra en producción.

```
# settings.py
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

ALLOWED_HOSTS

Hosts permitidos para el proyecto, mientras el `DEBUG` esté en True, esta variable puede permanecer vacía, y cuando `DEBUG` cambie a False, se debe agregar un host obligatorio que puede ser el dominio de tu sitio. Es una lista con las direcciones de todos los dominios que pueden ejecutar su proyecto Django.

```
# settings.py
ALLOWED_HOSTS =[]
```

INSTALLED_APPS

Variable donde agregamos las aplicaciones de terceros que vayamos utilizando, o creando.

```
# settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'boards.apps.BoardsConfig',
]
```

MIDDLEWARE

Es un marco de enlaces en el procesamiento de `request/reponse` de Django. Es un sistema de "complemento" ligero y de bajo nivel para alterar globalmente la entrada o salida de Django.

```
# settings.py
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

TEMPLATES

Esta carpeta de plantillas contiene todas las plantillas que creará en diferentes aplicaciones de Django.

```
# settings.py
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
```

```
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Como alternativa, se puede mantener una carpeta de plantilla para cada aplicación por separado. Si se hace de esa forma, no necesita actualizar el `DIRS` en `settings.py`. Asegúrese de que la aplicación esté agregada en `INSTALLED_APPS` en `settings.py`.

DATABASES

Variable de configuración de Base de Datos. Aquí podemos configurar cualquier motor de base de datos que Django utilice: MySQL, Postgres, Oracle, SQLite (por defecto), entre otras.

```
# settings.py
# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

VARIABLES DE URL

Son relativas a `BASE_DIR`. Estas variables se utilizan para almacenar archivos multimedia o estáticos. Cree carpetas estáticas y multimedia en el directorio principal. `MEDIA_URL` es la ruta relativa a `BASE_DIR`. Esta variable se utiliza para almacenar los archivos multimedia.

```
# settings.py
MEDIA_URL = '/ media /'
```

`STATIC_URL` es la ruta relativa a `BASE_DIR`. Esta variable se utiliza para almacenar archivos estáticos.

```
# settings.py
STATIC_URL = '/ static /'
```

VARIABLES ROOT

Son rutas absolutas. Estas variables se utilizan para recuperar archivos multimedia o estáticos.

`MEDIA_ROOT` es la ruta absoluta. Esta variable se utiliza para recuperar los archivos multimedia.

```
# settings.py
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

`STATIC_ROOT` es la ruta absoluta. Esta variable se utiliza para recuperar los archivos estáticos.

```
# settings.py
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

MANEJO DE ESPACIOS DE NOMBRE

Los espacios de nombres de URL le permiten invertir patrones de URL con nombre de manera única, incluso si diferentes aplicaciones usan los mismos nombres de URL. Es una buena práctica que las aplicaciones de terceros usen siempre direcciones URL con espacios de nombres. Del mismo modo, también le permite invertir las URL si se implementan varias instancias de una aplicación. Es decir, dado que varias instancias de una sola aplicación compartirán URL con nombre, los espacios de nombres proporcionan una forma de diferenciar estas URL con nombre.

Las aplicaciones de Django que hacen un uso adecuado del espacio de nombres de URL se pueden implementar más de una vez para un sitio en particular. Por ejemplo, `django.contrib.admin` tiene una clase `AdminSite` que le permite implementar más de una instancia del administrador, por lo que se puede implementar en una aplicación dos ubicaciones diferentes que brinden la misma funcionalidad.

Un espacio de nombres de URL se presenta en dos partes, las cuales son cadenas:

Espacios de nombres de una aplicación: describe el nombre de la aplicación que se está implementando. Cada instancia de una sola aplicación tendrá el mismo espacio de nombres de aplicación. Por ejemplo: la aplicación de administración de Django tiene el espacio de nombres de aplicación algo predecible de `'admin'`.

Espacio de nombres de una instancia: identifica una instancia específica de una aplicación. Los espacios de nombres de las instancias deben ser únicos en todo el proyecto. Sin embargo, un espacio de nombres de instancia puede ser el mismo que el espacio de nombres de la aplicación. Esto se utiliza para especificar una instancia predeterminada de una aplicación. Por ejemplo: la instancia de administración de Django predeterminada tiene un espacio de nombres de instancia de `'admin'`. Las direcciones URL con espacio de nombres se especifican mediante el operador `':'`.

Por ejemplo, se hace referencia a la página de índice principal de la aplicación de administración mediante `'admin:index'`. Esto indica un espacio de nombres de 'admin', y una URL con nombre de 'index'. Los espacios de nombres también se pueden anidar. La URL nombrada `'sports:polls:index'` buscaría un patrón llamado 'index' en el espacio de nombres 'polls', que a su vez está definido dentro del espacio de nombres de nivel superior 'sports'.

MODELO MVC Y LA CREACIÓN DE APLICACIONES

Django sigue el patrón `MVC` tan al pie de la letra que puede ser llamado un framework `MVC`. Someramente, la M, V y C se separan en Django de la siguiente manera:

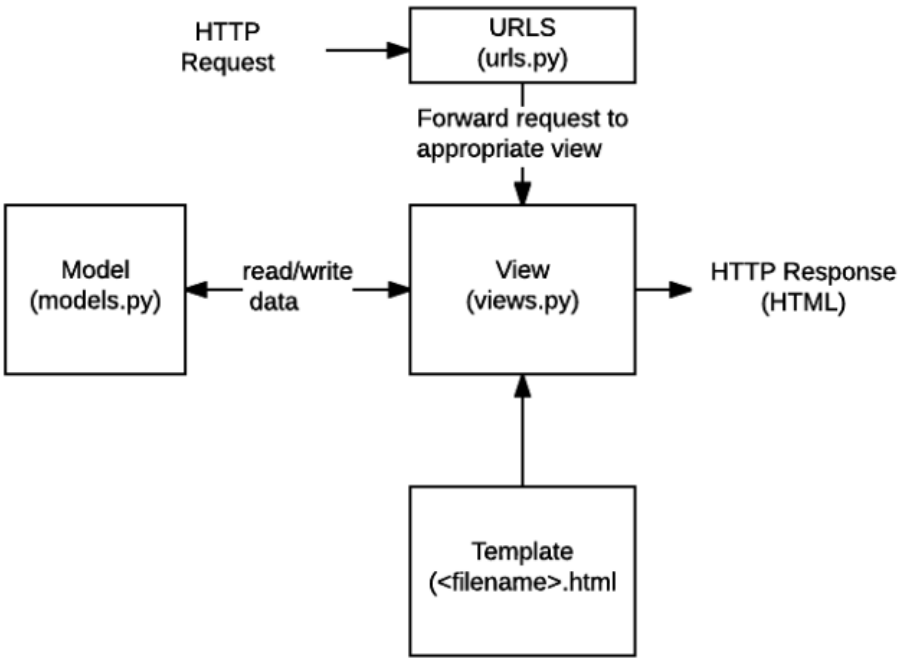
- M: la porción de acceso a la base de datos es manejada por la capa de la base de datos de Django, la cual describiremos en este capítulo.
- V: la porción que selecciona qué datos mostrar y cómo mostrarlos, es manejada por la vista y las plantillas.
- C: la porción que delega a la vista dependiendo de la entrada del usuario, es manejada por el framework mismo siguiendo tu `URLconf`, y llamando a la función apropiada de Python para la URL obtenida.

Debido a que la "C" es manejada por el mismo framework, y la parte más importante se produce en los modelos, las plantillas y las vistas, Django es conocido como un Framework MTV. En el patrón de diseño MTV:

- M: significa "Model" (Modelo), es la capa de acceso a la base de datos. Ésta contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- T: significa "Template" (Plantilla), es la capa de presentación. Ésta contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web u otro tipo de documento.
- V: significa "View" (Vista), es la capa de la lógica de negocios. Ésta contiene la lógica que accede al modelo, y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre el modelo y las plantillas.

CORRELACIÓN DEL MODELO

Las aplicaciones web de Django normalmente agrupan el código que gestiona cada uno de estos pasos en archivos separados:



Estos archivos pasan a ser los componentes esenciales de cualquier aplicación Django. A continuación se presenta una descripción de cada uno de estos componentes.

- **URLs:** aunque es posible procesar peticiones de cada URL individual vía una función individual, es mucho más sostenible escribir una función de visualización separada para cada recurso. Se usa un mapeador URL para redirigir las peticiones HTTP a la vista apropiada, basándose en la URL de la petición. El mapeador URL se usa para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL puede también emparejar patrones de cadenas o dígitos específicos que aparecen en una URL, pasarlos a la función de visualización como datos.
- **Vista (View):** es una función 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").
- **Modelos (Models):** son objetos de Python que definen la estructura de los datos de una aplicación, y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos.
- **Plantillas (Templates):** es un archivo de texto que define la estructura o diagrama de otro archivo (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real. Una vista puede crear dinámicamente una página usando una plantilla, rellenándola con datos de un modelo. Una plantilla se puede usar para definir la estructura de cualquier tipo de archivo, por lo general HTML, pero no necesariamente.