**Estructura del proyecto**

**Fecha: 01 / 05 / 2020**

**Objeto: Estructura del Proyecto**

**Framework: Flask**

**Autor: Manuel Semeco**

**Introducción**

**El objeto del presente documento es la de especificar la estructura del proyecto Flask y también la de describir cada conjunto ó elemento que le compone y una breve explicación de la interacción entre cada uno de ellos.**

**Indice:**

* [Directorio Minitienda](#Directorio_minitienda)
  + Directorios en minitienda:

[Develop\_env](#Minitienda_devenv)

[Project](#Minitienda_Project)

* + - [Directorio Project](#Directorio_Project)
      * Archivos:

[Dbase.py](#Project_dbase)

[Form.py](#Project_forms_py)

[Manage.py](#Manage_py)

* + - * [Directorio app](#Directorio_app)
        + [Directorios en app:](#directorios_en_app)

[Static](#app_static)

[Template](#app_templates)

* + - * + Archivos:

[App.py](#app_app_py)

[Config.py](#app_config_py)

[Login.py](#app_login_py)

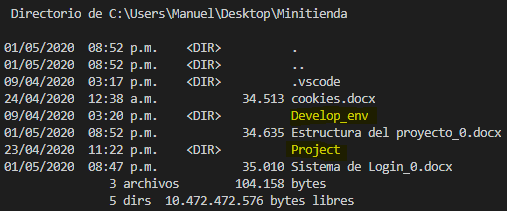
[Models.py](#app_models_py)

**Detallando estructura del proyecto:**

La estructura general del directorio de trabajo Minitienda es como sigue.

* **Directorio Minitienda**

**Directorios y archivos - Minitienda**



**Directorios en Minitienda:**

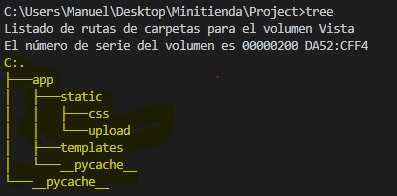
* **Develop\_env:** es la carpeta o directorio que contiene todas las dependencias del VirtualEnviroment generado a través del comando “ **python venv –m Develop\_env ’’** en la consola de comandos. Esto genera un entorno aislado en cuanto a dependencias del proyecto refiere. De manera que no existe conflicto entre el entorno **Develop\_env** y el entorno global (el entorno padre). Una vez creado, para activarlo solo hace falta ejecutar el **active** en la carpeta **script** desde la ventana de comandos, y asi, yá, poder instalar todas las dependencias del proyecto (librerías/modulos por ejemplo).
* **Project:** es la carpeta del proyecto, en él se encontrara todo sus componentes. Algunos como [**manage.py**](#Manage_py) funcionaran para meta-gestionar el proyecto y otros como models.py serán usados para abstraer el mecanismo de negociación con la capa de datos (motor de base de datos).

**Comentario**: Para saber más al respecto visitar los documentos **meta\_gestion\_0.docx** y **ORM\_0.docx.**

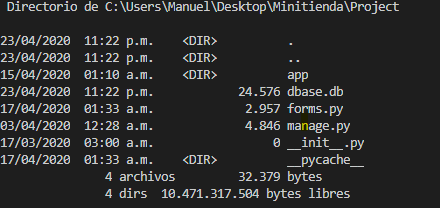
* **Directorio Project**

**Estructura-Project**

Dentro de **Minitienda** se encuentra **Project** y la **jerarquía** de este último es como sigue.



**Directorios y archivos - Project**



Como se puede apreciar se tienen **archivos .py .db** en el directorio project son:

* **dbase.db :** fichero de base de datos, el cual es soportado por **sqlite** y generado a partir del compaginación de la configuración de la app que es donde se define el motor, usuario, contraseña, y en algunos casos la ubicación del fichero que representa la base de datos.

**Dependencias:**

**config.py:** en donde hacemos setear el **SQL\_DATABASE\_URI**, el cual, su valor, básicamente especifica el motor de base de datos, contraseña, usuario del cliente y donde se va a generar el archivo para el caso de sqlite.

**wrapper SQLalchemy**: quien provee los métodos ORM de la app flask y en complemento con manage.py (quien es que el que ejecuta el método del wrapper en sú codigo).

**\_\_ Comentario:** no se está 100% seguro corregir al momento de hacer el cambio de base de datos a mysql.

\_\_ **Comentario:** redefinir este párrafo esta dependencia es a nivel de import, siendo una clase y no un fichero dispuesto directamente en el directorio del proyecto.

**Comentario:** Para saber más al respecto ver el documento **meta\_gestion\_0.docx.**

* **Form.py:** este documento viene a razón de la implementación Flask-WTF cuya función es la de abstraer el proceso de **elaborar** y **procesar** formularios mediante el uso de clases (ergo objetos) (su uso fue pensado para ser complementario en la elaboración de plantillas con el motor, antes(desde controladores enviar objeto,datos a plantilla) y posterior(controladores, procesamiento) en los controladores de flask) .

**Comentario:** Para más información ver el documento **Flask\_Form\_0.docx.** Su ubicación en directorio es a criterio del developer, puede ir donde sea pueda ser invocado sin conflicto.

\_\_ **Meta-Comentario**: Crear el documento respectivo.

**Dependencias:**

**-Con algún fichero .py NINGUNA**

**-Solo a nivel de importación de modulos.**

* **Manage.py:** es un script usado para ejecutar funciones entre las cuales se encuentran la funciones definidas por el desarrollador, todo mediante la interacción del usuario(generalmente el admin) haciendo uso de la consola de comandos. Este script Python hace uso de **Flask-Script** usando la clase **Manager,** esté, **es** un wrapper del objeto app que integra los métodos para la ejecución de dichas funciones de gestión desde el Shell ya sean las propias del wrapper y/o las definidas por el desarrollador. En este proyecto se implementó para realizar el modelo de base de datos implementando los objetos del **ORM,** Incluyendo tambien el wrapper SQLalchemy.

**Comentario:** Ver el documento **meta\_gestion\_0.docx**

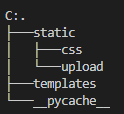
**Dependencias:**

**App.py :** depende de app.py por el objeto app que no es más que el wrapper fundamental de la app que contiene las configuraciones y los métodos de la aplicación, de manera que el Flask-Script con la clase wrapper Manager lo necesita para enlazar el app(conferir sus métodos y configuración) y asi implementar los comandos manager.

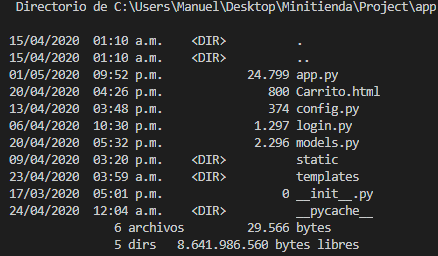
* **Directorio app**

El directorio dentro de **Project** se encuentra **app** y la **jerarquía** de directorio dentro de este último es como sigue.

**Estructura-app**



**Directorios y archivos - app**



**Directorios en app:**

Los directorios que alberga app son **static** y **templates.** La aplicación principal app.py hace una búsqueda por defecto en el directorio actual, de carpetas con esos nombres hasta donde se sabe solo hace uso de tampletes.

**Comentario:** para más información ver documentación **app\_0.docx** sección templates

\_\_ **Meta-comentario**:añadir el documento referido.

* **Static**: es la carpeta donde se destinan los archivos como imágenes documentos entre otros. En la aplicación actual se implementó para albergar las imágenes de los juegos.
* **Templates**: alberga los html(plantillas) que se van a renderizar.

**Definiendo archivos del directorio app:**

* **App.py:** Es el archivo que alberga los routers(como decoradores de una función controlador) y un controlador en forma de función esperando el llamado de esa url definida por el router, además en él se tiene la clase Flask que básicamente es un wrapper que le da la capacidad de instanciar a la aplicación como un objeto y poderla extender con otras con otros modulos(Flask-login,SQLAlchemy, entre otros ) y además le da la capacidad para funcionar como un servidor de aplicaciones(bajo el estándar wsgi) además de métodos y atributos (como .config).

**Dependencias:**

**Config.py:** Se usa para asignar el atributo config de las instancia de la aplicación “app”

**Form.py**: le da los objetos form para crear en los templates los formularios y también le permite procesar los datos de este último.

**Models**.**py**: les da los objetos **ORM** para interactuar con una base de datos. Ademas esta depende a su vez app para su correcto funcionamiento, ya que este último le provee una clase wrapper del **objeto app** para realizar distintas funciones en dicho **script(Models.py)** para concretar las clases de los **objetos ORM**.

* **Config.py:** Es un fichero que alberga la configuración del aplicativo, creado aparte con la intensión para hacer más modular la estructura del proyecto.

**Dependencia:**

**-**Sin dependencias con otros ficheros.py

-Solo se importa os para uso interno.

**Comentario:** para más información ver documentación **Config\_0.docx**

\_\_ **Meta-comentario**:añadir el documento referido.

* **Login.py:** Fichero para albergar funciones personalizadas hechas por el developer de la aplicación. Este mecanismo fue sustituido por la extensión de Flask-Login.

**Comentario:** para más información ver documentación **Sistema de login\_0.docx**

\_\_ **Meta-comentario**:añadir el documento referido.

* **Models.py:** Fichero que se usa para albergar las clases que representan las tablas (**ORM**) e intrínsecamente en sus definiciones el modelo de base de datos

**Comentario:** para más información ver documentación **Models\_0.docx**

\_\_ **Meta-comentario**:añadir el documento referido.