	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros



**IES LOS SAUCES
BENAVENTE (ZAMORA)**

**TITULO GRADO SUPERIOR DE
DESARROLLO DE APLICACIONES WEB**



DEPARTAMENTO: INFORMÁTICA

TITULO PROYECTO: Framework de desarrollo web con Django

AUTOR: Israel García Cabañeros

CURSO: 2019/2020


PERIODO: JUNIO

TIPO DE PROYECTO: Proyecto de investigación y desarrollo

CÓDIGO DE PROYECTO: 1070


TUTORÍA COLECTIVA: Amor Rodríguez Navarro

TUTOR COLABORADOR: María Criado Domínguez

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Django

Framework para desarrollo web

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

El presente proyecto de investigación y desarrollo cuyo título es “Framework de desarrollo web con Django” del módulo PROYECTO ha sido realizado por Israel García Cabañeros del alumno del IES LOS SAUCES del ciclo Ciclo Formativo Grado Superior Desarrollo de aplicaciones web con el fin de la obtención del Título de Técnico Superior en Desarrollo de Aplicaciones Web.

La tutoría individual de dicho proyecto ha sido llevada a cabo por D./D^a. *Amor Rodríguez Navarro* profesor de segundo curso de CFGS Desarrollo de Aplicaciones web.

En Benavente, __ de mayo de 2020


Autor	Vº Bº	Vº Bº	Vº Bº
	Tutoría colectiva	Tutoría individual	Colaboración

Fdo. Israel García
Cabañeros

Fdo. Amor Rodríguez
Navarro


Fdo.: Amor
Rodríguez Navarro

Fdo.: María Criado
Domínguez

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

ÍNDICE

1	Resumen del proyecto.....	5
2	Conceptualización del problema.....	5
3	Objetivos.....	5
4	Estudio teórico del Framework.....	5
5	Django vs. Flask	8
6	Implementación	9
6.1	Modelo de red, equipos y sistemas operativos	9
6.2	IDE y herramientas de desarrollo utilizados	10
6.3	Lenguajes de programación y lenguajes de marcas utilizados.....	10
6.4	SGBD o sistema de almacenamiento utilizado	11
6.5	Repositorio de software.....	11
6.6	Framework, ... (otras herramientas o tecnologías utilizadas).....	14
6.7	Análisis y diseño del supuesto práctico.....	21
6.8	Modelo de clases	22
6.9	Modelo de casos de uso.....	23
6.10	Árbol de navegación	24
6.11	Bases de datos	24
6.12	Presupuesto y financiación	27
6.13	Paso a producción.....	27
7	Conclusiones y líneas futuras	34

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

1 Resumen del proyecto

El proyecto se va a centrar en el marco de trabajo o Framework llamado Django. Se mostrará la facilidad de usar un Framework y, en concreto, como puede hacerse un mantenimiento, en este caso, de productos, tanto como si es el desarrollador el que lo hace, como el cliente, y esto se realizará mediante el administrador de Django.

Además, se mostrarán las ventajas del uso de un gestor de paquetes como es Conda, o en este caso, Miniconda.

El resultado final será una tienda sencilla, con un CRUD básico, con la que espero, queden claros los objetivos de usar un Framework, en vez de desarrollar todo desde cero, con una comunidad muy amplia, lo que ayuda en la resolución de problemas a la hora de desarrollar.

2 Conceptualización del problema.

Se analizarán varios campos o problemáticas:

En primer lugar el lenguaje de programación utilizado. Se usó el lenguaje Python por la razón de que quería aprender un nuevo lenguaje y esta fue la elección debido a que es un lenguaje que ha crecido mucho en estos últimos años y tiene una amplia lista de librerías.

El segundo es el uso de un Framework frente al uso de un lenguaje puro en el desarrollo de una página web. Esto es porque durante el tiempo que se está en el curso de Diseño de Aplicaciones Web no se enseña ningún Framework y considero que se sale sin saber exactamente lo que es un Framework y cuando cuesta aprender a usarlo.

Y la última problemática fue durante el tiempo que llevo trabajando, se trata de el trabajo con diferentes versiones de un Framework, librería o lenguaje de programación y lo costoso que es a veces cambiar de una versión a otra, o simplemente la instalación de versiones antiguas.


3 Objetivos

- Uso de un Framework.
- Necesidad de aprender el lenguaje en el que se basa dicho Framework.
- Uso de un gestor de paquetes.

4 Estudio teórico del Framework

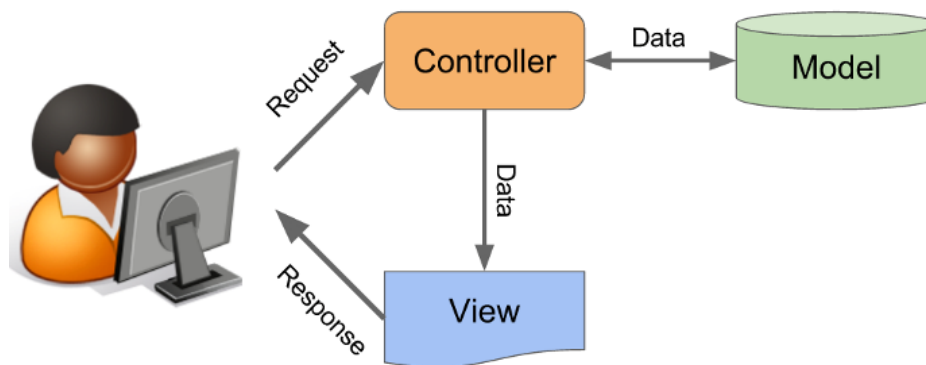
Django. Lo primero que haremos será comparar el patrón arquitectura MVC (Modelo – Vista – Controlador), con el patrón de arquitectura propio de Django llamado MVT (Model – View – Template).

El patrón de arquitectura MVC tiene 3 capas:

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	--


- **Modelo (Model):** El modelo se encarga de la lógica de la aplicación, es decir, que se encarga de recibir solicitudes para recuperar o modificar información de las bases de datos que también se encuentran aquí.
- **Vista (View):** La vista será lo que verá el usuario en la aplicación de escritorio, página web, etc. Y, a través de la cual, el usuario enviará las peticiones. La vista será la parte generada con HTML en el caso de las páginas web.
- **Controlador (Controller):** El controlador se encargará de procesar la información que recibe, tanto del modelo, como de la vista.

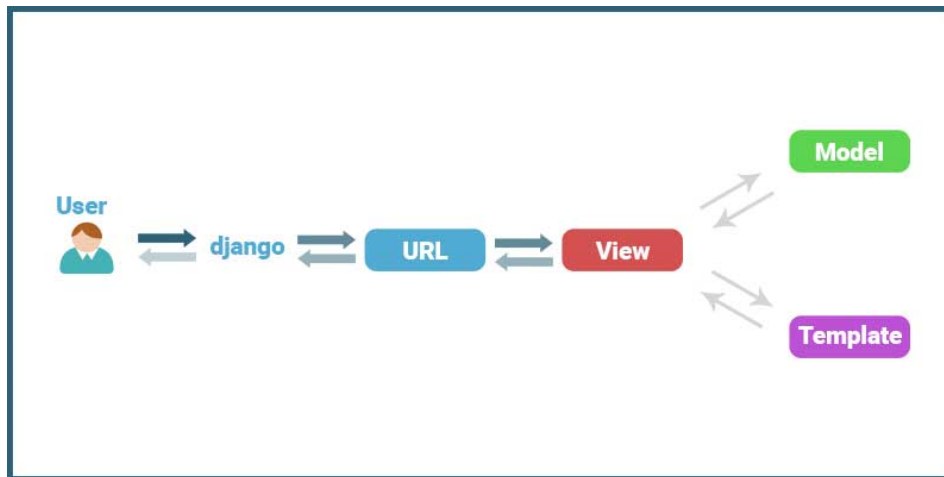
De una forma simple, el flujo de datos del MVC a un alto nivel se puede explicar como que el usuario a través de la vista hace una petición, llega al controlador, que al necesitar información se la pide al modelo que contiene toda la información, se la devuelve al controlador y este la procesa para devolvérsela finalmente a la vista para que el usuario pueda verla.



El patrón de arquitectura MVT tiene 3 capas:

- **Modelo (Model):** El modelo, en este caso también se encarga de la lógica de la aplicación. Los modelos están basados en clases, y como ya se ha explicado anteriormente, estas son mapeadas por Django. Los modelos se encuentran en los archivos *models.py*.
- **Vista (View):** La vista, actúa como el controlador del MVC, usando la información de los modelos, y devolviendo la información a los templates. Las vistas se encuentran en los archivos *views.py*.
- **Plantilla (Template):** Las plantillas, son archivos HTML en los que se muestra la información procesada por las vistas.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	--




Django, está formado por la propia estructura inicial del proyecto, y a este proyecto pueden añadirse multitud de aplicaciones, ya sea creadas por el propio desarrollador, como las que se creen por otros desarrolladores, en un repositorio, etc.

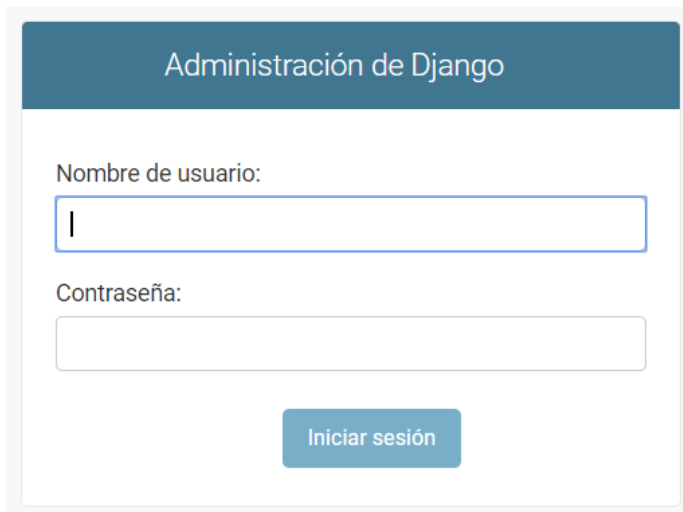


Administrador Django. Una de las ventajas de este Framework, es que dispone de un administrador con el cual podemos acceder a los modelos mapeados en Django, y desde este, realizar un CRUD básico de nuestros modelos.

Además, este administrador ya viene con una gestión de usuarios y grupos por defecto.

Para acceder al administrador tendremos que arrancar nuestro servidor local, o si ya tenemos la página en un hosting, deberemos añadir a la ruta `/admin`. Al acceder a este path veremos una pantalla de login con la que podremos acceder al administrador mediante las credenciales de un usuario staff.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros



Una vez dentro, podremos diferenciar un menú con las opciones:

- **VER EL SITIO.** El cual nos llevará a nuestra página web sin cerrar la sesión actual.
- **CAMBIAR CONTRASEÑA.** El cual nos llevará a un formulario en el que podremos cambiar nuestra contraseña.
- **TERMINAR SESIÓN.** El cual nos llevará a la página principal de nuestra página web, cerrando la sesión actual.

También podemos ver un apartado de autenticación y autorización, en el cual podremos administrar nuestros usuarios y grupos.

Además del apartado anterior veremos el resto de modelos mapeados y registrados los cuales han sido hecho por el desarrollador de la página web.


Y por último veremos un listado de todas las acciones realizadas por el usuario logueado actualmente (añadir un nuevo vino, eliminar un usuario, modificar un dato, etc.).

5 Django vs. Flask

Django es un Framework web Python de alto nivel (esto quiere decir que es de fácil comprensión para un humano) que fomenta el desarrollo rápido y limpio.

Una herramienta importante es el mapeo objeto-relacional (ORM). Esta se trata de una técnica de programación que consiste en convertir los datos de un lenguaje de programación orientado a objetos en una base de datos relacional para lograr una persistencia de datos, la cual nos permite hacer queries.

Django también tiene enrutamiento de rutas, el cual nos ayuda a determinar que lógica seguir dependiendo de la ruta de una petición web.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Otra característica son las plantillas HTML, que nos permiten definir alguna lógica de presentación e insertar información dinámica dentro de nuestro HTML.

Además, Django nos permite el manejo de formularios y las pruebas unitarias.

Por último, hay que saber que Django no es un lenguaje de programación, Django es una herramienta escrita en Python para el desarrollo de aplicaciones web.

Flask es un Framework web Python que permite un levantamiento y ejecución rápido. Esto es porque destaca en la instalación de extensiones para el tipo de proyecto que se va a desarrollar, es decir, que el prototipo se puede hacer rápidamente.

Permite pruebas unitarias y cookies de seguridad.

Al ser un diseño más simple que Django, este puede ser un determinante a la hora de saber cual usar, ya que este diseño lo hace significativamente más rápido.

Una desventaja frente a Django, es que tiene un sistema de autenticación muy simple y menos seguro. Django tiene un sistema de autenticación mucha más potente.

El mapeo objeto-relacional u ORM es externo, a diferencia de Django.

6 Implementación

6.1 Modelo de red, equipos y sistemas operativos


Modelo de red inalámbrica por router. Solamente se necesitará una red inalámbrica para la descarga e instalación de ciertas herramientas, además, de para las pruebas en producción cuando se suba a internet a través de PythonAnywhere, que se explicará más adelante, en el punto 4.14.

Otro motivo por el que se necesitaría red, independientemente del modelo de red, sería para el uso, por ejemplo de links en el HTML, en vez de tenerlos descargados e implementados en un directorio de nuestro proyecto.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
<link href="https://fonts.googleapis.com/css?family=Cinzel:400,700|Montserrat:400,700|Roboto&display=swap"
rel="stylesheet">
```

Ordenador portátil. Dispositivo que contendrá las herramientas utilizadas para el desarrollo de la web.

Dispositivo móvil. Dispositivo que será utilizado únicamente para probar el diseño responsive o adaptable de la web en otros dispositivos diferentes a un ordenador.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Windows 10. Sistema operativo que tendrá el ordenador con el que desarrollaremos la web.

6.2 IDE y herramientas de desarrollo utilizados

Visual Studio Code. El motivo por el que se ha usado este editor de código de código abierto es desarrollado por Microsoft, es la gran cantidad de extensiones que tiene, desarrolladas por la amplia comunidad de usuarios que soportan dicho editor.

Además, tiene incorporado un depurador de código y tiene control de versiones en Git.

Entre la cantidad de extensiones que tiene este editor vamos a resaltar las que usaremos para este proyecto:

- **Python.** Esta extensión de Microsoft incluye, además de funciones de Python, debugging, formateo de código, test unitarios, etc.
- **Django.** Esta extensión la usaremos para añadir funciones de Django que no aparecen por defecto con la extensión de Python en nuestro editor de código.
- **Django-intellisense.** Esta extensión la usaremos para el soporte de autocompletado de métodos de Django.

Y por último, otra de las razones por las que se eligió este editor, es la facilidad con la que podemos cambiar de entorno virtual.

Solo tenemos que pinchar en nuestro editor en la parte de abajo a la izquierda la zona que aparece a continuación remarcada.




Después de esto en la parte superior del editor nos mostrará un listado con todos nuestros entornos virtuales de los cuales podremos ya elegir el que necesitamos en el momento.

6.3 Lenguajes de programación y lenguajes de marcas utilizados.

Python. Este lenguaje orientado a objetos será utilizado para el desarrollo de funciones usadas en la página web. Se ha usado este lenguaje como lenguaje principal del proyecto debido a que es un lenguaje que ha crecido mucho estos últimos años y tiene una sintaxis fácil de comprender con la característica que no usa llaves para señalar los bloques de código, si no que usa la propia tabulación del código para saber dónde empieza y acaba un bloque.

HTML5. Este lenguaje se ha usado como lenguaje de marcas para construir lo que viene a ser el esqueleto o estructura de la página web.

CSS3. Este lenguaje se ha usado para los estilos de la página web.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

JavaScript. Este lenguaje se ha usado, tanto para estilos de la página web, como para funcionalidad y validaciones del lado del cliente.

JQuery. JQuery no se trata tanto de un lenguaje, si no de una librería basada en JavaScript, con la que se pueden hacer cosas como manipulación de documentos HTML, eventos de teclado, animaciones, etc.

Ajax. Al igual que JQuery, no se trata de un lenguaje en sí, si no que se trata de una herramienta para desarrollar aplicaciones dinámicas e interactivas de forma asíncrona que se ejecutan del lado del cliente mientras se mantiene esa comunicación asíncrona con el servidor en segundo plano.

6.4 SGBD o sistema de almacenamiento utilizado

El Sistema Gestor de Bases de Datos que se usará será SQLite. Este se trata de un gestor relacional ligero y abierto.

La principal diferencia con otros gestores es que no requiere de un servidor como soporte para funcionar, además, no es necesario configurar el host y los puertos. Esto último podemos observarlo en el archivo settings.py de nuestro proyecto Django.

Añadir también, que Django soporta gestores como MySQL, PostgreSQL y Oracle, además del ya mencionado SQLite, que es el que vamos a utilizar.

6.5 Repositorio de software

En cuando al repositorio usaremos GitHub, este repositorio se trata de una plataforma que podemos encontrar, tanto en versión web, como su versión de escritorio. GitHub, se basa en el control de versiones de Git, el cual descargaremos e instalaremos en nuestro ordenador desde la página oficial de Git, <https://git-scm.com/>.

Actualmente, pertenece a Microsoft, además de a GitHub Inc., y se trata de una plataforma de código abierto.


A continuación, se mostrará cómo se hizo creó el repositorio y los comandos de git que se utilizaron en el proceso:

Creación del repositorio en la plataforma GitHub.

Entraremos en la página oficial de GitHub, <https://github.com/>, si no hemos iniciado sesión o no estamos registrados, tendremos que hacerlo.


Una vez dentro de GitHub, con nuestra sesión iniciada, iremos al dropdown de arriba a la derecha de la pantalla, e iremos a *Your repositories*.

Aquí encontraremos una lista con los repositorios que tenemos ya creados, si es que tenemos alguno, además, de un botón verde en el que pone *New*. Al pinchar en ese botón nos aparecerá la siguiente pantalla.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Repository template
Start your repository with a template repository's contents.

Owner **Repository name ***

 israxx97 /

Great repository names are short and memorable. Need inspiration? How about **fuzzy-waffle**?

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.


[?](#)

En dicha pantalla, escribiremos el nombre de nuestro nuevo repositorio, una descripción opcional, seleccionaremos uno de los **RadioButton** (público o privado), en este caso seleccionaremos *Public*. A continuación, podremos seleccionar mediante un **CheckBox**, si queremos o no generar un **README.md**, en mi caso, siempre lo genero para poder comentar cosas que considere importantes respecto al proyecto en el que esté trabajando (novedades de la nueva versión, etc.). Y, por último, tenemos dos **drop down**. El primero, que para este proyecto será de tipo *Python*, será para añadir el tipo de **.gitignore**, este archivo se usa para decirle al repositorio, que archivos, queremos ignorar para que no se suban a GitHub, esto es porque ciertos archivos contienen ciertos datos que no nos interesa que se suban por temas de seguridad. Por otro lado, tenemos el segundo **drop down**, que se usa para añadir un archivo más con un tipo de licencia, en mi caso, suelo añadir el tipo de licencia *GNU General Public License v3.0*, que es un tipo de licencia para el uso libre del Software, por lo que todo el mundo puede usar, compartir y modificar dicho Software.

Finalmente, después de haber seguido los pasos indicados anteriormente, pulsamos en el botón verde *Create repository*. Ahora, si vamos *Your repositories*, encontraremos nuestro nuevo repositorio, e iremos a él.

Instalación del repositorio Git en Windows.

Entraremos en la página oficial de Git, <https://git-scm.com/>, al entrar en esta página, encontraremos la imagen de un monitor, con un botón, desde el que descargaremos la última versión para Windows, cuando yo lo hice, era la versión 2.26.0.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Después de descargar el ejecutable, lo ejecutaremos y seguiremos los pasos que nos da el instalador de Git con nuestras preferencias.

Finalmente, después de haber cerrado el instalador de Git, deberemos añadir dos nuevas líneas en el *Path* de nuestras variables de entorno del sistema, siendo *GitPath*, la ruta donde decidimos instalar Git durante el proceso de instalación:

- *GitPath\Git\bin*
- *GitPath\Git\cmd*

Para probar qué si ha instalado correctamente, desde cualquier directorio con el cmd, ejecutamos el siguiente comando:

- `git --version`

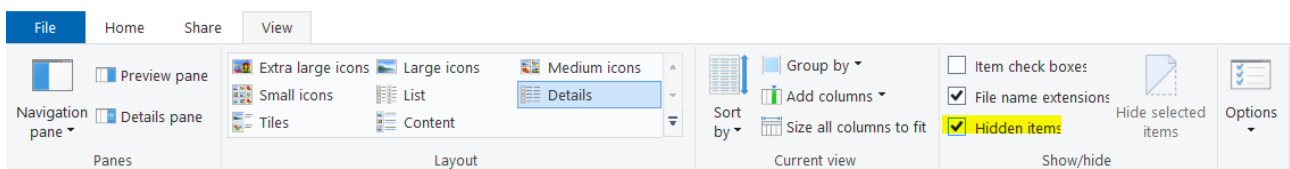
Iniciar en Git el repositorio creado en GitHub.

Usaremos el siguiente comando para copiar localmente nuestro repositorio de GitHub en nuestro ordenador:


- `git clone urlrepositorio`


Obtendremos *urlrepositorio* de la siguiente forma. Primero, iremos al repositorio de GitHub que creamos anteriormente. Cuando nos hayamos situado en dicho repositorio, es la pestaña *Code*, encontramos un drop down verde *Clone or download*, ahí copiamos la url del repositorio y la pegamos en el cmd con click derecho, ejecutando así, el comando que se mencionó, anteriormente.

Al hacer esto, ya tenemos, al menos, un directorio oculto llamado *.git*, que podemos mostrar marcando el CheckBox *Hidden items* en la pestaña *View* en el Explorador de Windows como se muestra a continuación:








Además de dicho directorio, encontraremos los archivos que elegimos durante la creación del repositorio en GitHub. En mi caso, encontraremos los siguientes archivos adicionales:

 .gitignore	3/27/2020 10:52 A...	Text Document	2 KB
 LICENSE	3/27/2020 10:52 A...	File	35 KB
 README.md	3/27/2020 10:52 A...	MD File	1 KB

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Lo que haremos ahora, será mover el proyecto creado con Django (esto se ve en el punto 4.6 en el apartado *Django*) desde la raíz, al directorio que se ha creado al hacer el git clone. Nos quedará de la siguiente forma:

 .git	3/27/2020 10:55 A...	File folder	
 Django_Israel_Garcia_Cabaneros	3/27/2020 11:05 A...	File folder	
 .gitignore	3/27/2020 10:52 A...	Text Document	2 KB
 LICENSE	3/27/2020 10:52 A...	File	35 KB
 README.md	3/27/2020 10:52 A...	MD File	1 KB

Registraremos nuestros cambios con el siguiente comando:

- `git add .`

El punto (.) será para que recoja todo desde el directorio actual (que será el mostrado en la captura anterior.

Lo siguiente será hacer un commit, que no hará que nuestros cambios se suban al repositorio de GitHub, se hará de la siguiente forma:

- `git commit -m "Mensaje descriptivo del commit"`

Por último, para subir nuestros cambios al repositorio remoto de GitHub, ejecutaremos el siguiente comando:

- `git push origin master`

Ya tenemos nuestro proyecto en GitHub.


Añadir que, existen más comandos de Git para trabajar con repositorios, pero en nuestro caso no necesitaremos hacer más que lo básico, que es lo mostrado en los pasos anteriores. A partir de ahora podemos hacer los commit que queramos y subirlos todos a la vez con push.

URL del repositorio en GitHub: https://github.com/israxe97/Django_Proyecto_Final

6.6 Framework, ... (otras herramientas o tecnologías utilizadas)

Miniconda. Se trata de la versión ligera de Conda, y ambos, son gestores de paquetes. Estos son colecciones de herramientas utilizados para facilitar la instalación, actualización, configuración y eliminación de paquetes de Software.

El gestor que nosotros usaremos es compatible con muchos lenguajes, entre ellos, Python, Java, C/C++, Ruby, etc.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	--

Este gestor funciona de una forma particular, esta forma son entornos virtuales. Y es que puedo crear todos los entornos virtuales que yo quiera, y cada uno de ellos con diferentes características y con los paquetes que Miniconda tenga a mi disposición.

La ventaja de todo esto son las versiones de un lenguaje o paquete, si estoy trabajando en dos proyectos distintos y cada uno de ellos en una versión diferente, puedo tener también dos entornos, cada uno con la versión que necesito en cada proyecto, y así no tengo problemas de compatibilidad ni estar configurando archivos, etc., cada vez que cambie de proyecto. Todos los entornos virtuales son independientes unos de otros.

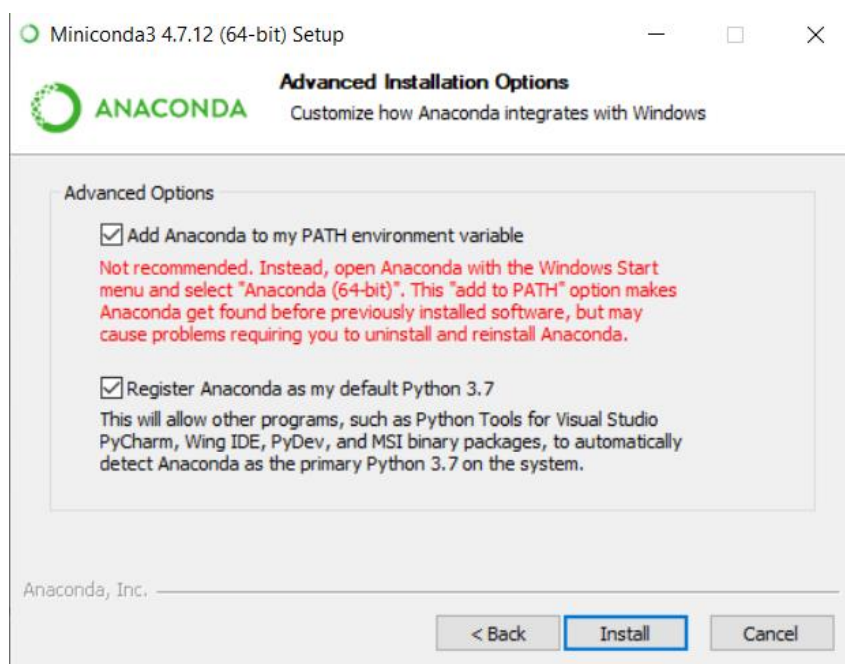
Instalación de Miniconda.


Antes de comenzar con la instalación de Miniconda es imprescindible no tener instalado Python anteriormente. Si ya lo teníamos, deberemos desinstalar todo lo referente a Python, ya que podría haber problemas.

Iremos a la página oficial de Conda, a la sección de Miniconda, <https://docs.conda.io/en/latest/miniconda.html#windows-installers>, descargaremos la última versión, si tu ordenador soporta 64-bit, descargaremos esta.

A continuación, ejecutaremos el instalador, durante este proceso de instalación daremos a *Next* y *I Agree*, hasta el momento en que lleguemos a la pantalla de *Destination Folder*, en la que elegiremos el directorio en el que queramos guardar lo referente a Miniconda. Yo he elegido el directorio escrito por defecto.

La última pantalla de instalación es la siguiente:



	<p>CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	---

En esta pantalla viene la primera opción sin marcar, deberemos marcarla para que nos añada automáticamente a la variable de entorno *Path*, una nueva línea referente a Miniconda, para que podamos ejecutar sus comandos en el cmd desde cualquier directorio.

Si durante la instalación, seleccionamos la opción de instalación de Miniconda solamente para el usuario actual, nos modificará la variable de entorno *Path* en el apartado de las variables de entorno del propio usuario desde el que hicimos dicha instalación, si seleccionamos la segunda opción, nos modificará el *Path* de las variables de entorno del sistema.

Si no hicimos check en la parte de la instalación de la captura anterior, deberemos ir a la variable de entorno *Path*, y añadir las siguientes líneas:

```
C:\Users\igarcia\AppData\Local\Continuum\miniconda3
C:\Users\igarcia\AppData\Local\Continuum\miniconda3\Library\mingw-w64\bin
C:\Users\igarcia\AppData\Local\Continuum\miniconda3\Library\usr\bin
C:\Users\igarcia\AppData\Local\Continuum\miniconda3\Library\bin
C:\Users\igarcia\AppData\Local\Continuum\miniconda3\Scripts
```

Lógicamente, la ruta hasta estos directorios no será la misma en todos los ordenadores, pero sí, muy similar.

Si hemos hecho correctamente la instalación de Miniconda, tendremos disponibles, tanto los comandos de Python, como los de Miniconda. Para comprobarlo ejecutaremos desde el cmd los siguientes comandos:

- python --version
- conda --version

Si en ambos casos el cmd nos responde con sus respectivas versiones, significa que ha funcionado.


Probando Miniconda.

Antes de comenzar, veremos que se usa en muchas ocasiones el comando pip. Se trata de un sistema de gestión de paquetes que se utiliza para la instalación y administración de paquetes que están escritos en lenguaje Python. Este sistema se instalará automáticamente cuando creamos nuestro entorno virtual.

Ahora que tenemos todo listo, seguiremos con su uso:

Lo primero de todo será crear un nuevo entorno virtual mediante el comando:

- conda create -n *EnvironmentName* python=*versión*

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	--

Para ver el listado de entornos virtuales usaremos el comando:

- `conda env list`

Para entrar en el entorno virtual ejecutaremos el siguiente comando:

- `conda activate EnvironmentName`

Para salir de nuestro entorno virtual ejecutaremos el siguiente comando:

- `conda deactivate`

Ahora instalaremos los paquetes necesarios, esto podemos hacerlo de dos formas:

1. De uno en uno mediante el comando:

- `pip install nombrepaquete==versión`

2. Mediante un archivo .txt con el siguiente comando:

- `pip install -r nombrearchivo.txt`


Dicho archivo tendrá el siguiente formato:

```
requirements.txt - Notepad
File Edit Format View Help
django==3.0.4
pillow==7.0.0
django-ckeditor==5.9.0
```

Solo usaremos esos 3 paquetes:

- **django:** Este paquete es para instalar el Framework Django en nuestro entorno virtual.
- **pillow:** Este paquete es necesario si queremos introducir modelos con campos ImageField, ya que si no está instalado, a la hora de mapear los modelos, nos dará error.
- **django-ckeditor:** Este paquete se utiliza para añadir en nuestros modelos campos de tipo CKEditorField, que es como un tipo TextField, pero con la opción de añadir párrafos, listas, negrita, cursiva, etc. Finalmente se ha usado TextField, por lo que se puede prescindir de este paquete, pero está bien tenerlo en cuenta.

Para ver los paquetes instalados en nuestro entorno ejecutaremos el siguiente comando:

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

- `pip list`

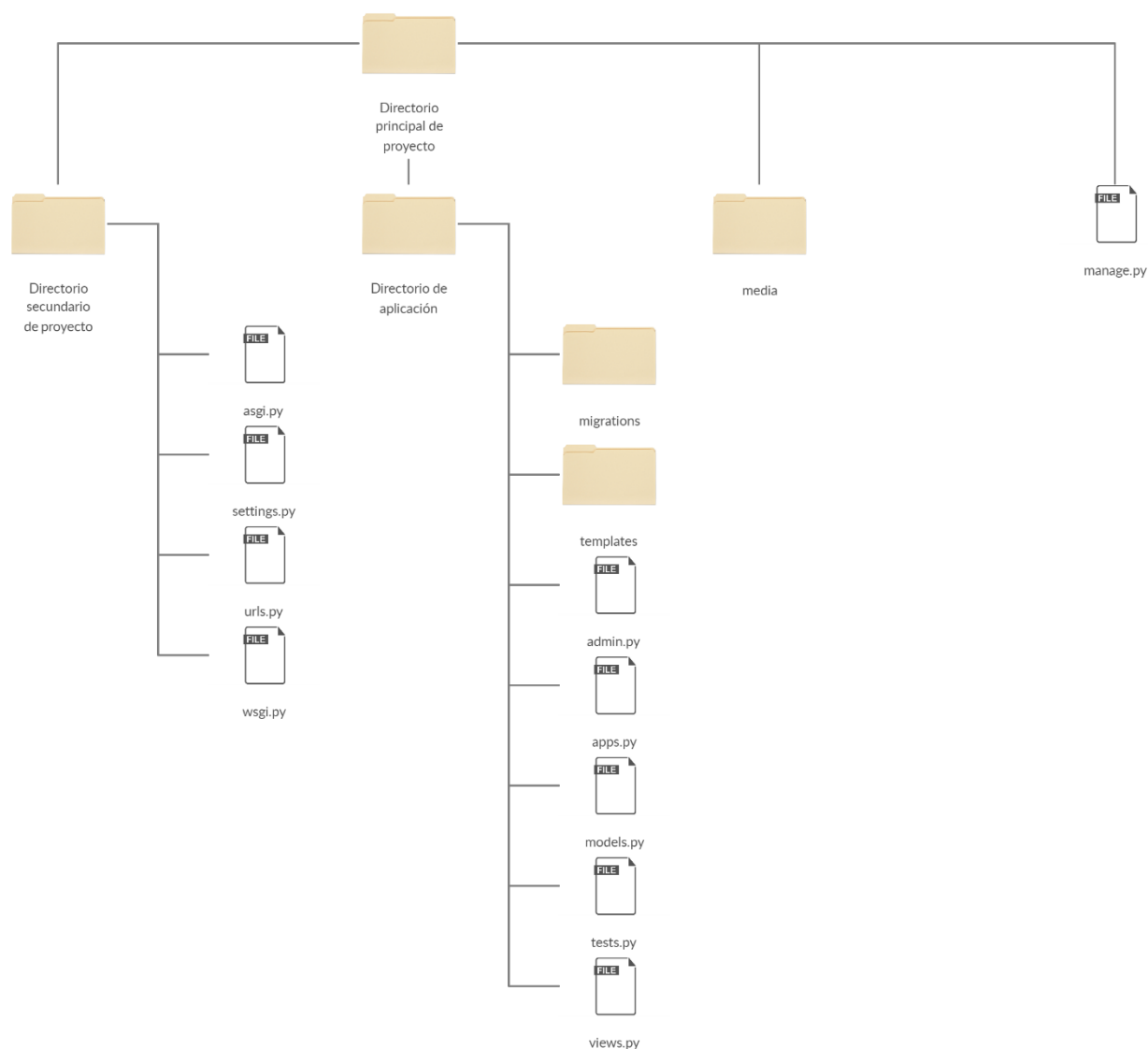
Ya solo nos queda crear nuestro proyecto con el comando:


- `django-admin startproject ProjectName`

Para crear una nueva App se ejecutará el siguiente comando:

- `python manage.py startapp appname`

Nuestro proyecto tendrá una estructura de directorios parecida a la de la siguiente imagen:



	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	--


Será parecida debido a que nosotros tendremos más de una App creada, y por lo tanto, tantos *Directorio de aplicación* como Apps hayamos creado, pero todos con la misma estructura.

A continuación, se explicará la utilidad de cada uno de nuestros directorios y archivos:

- **__init.py__**: Veremos estos archivos por todo nuestro proyecto, son archivos vacíos creados por Django por defecto para que el Framework reconozca al directorio donde se encuentra dicho archivo como un paquete de Python y permita usar los objetos en otras partes del proyecto.
- **manage.py**: Este archivo es el main del proyecto, en él podemos ver declarada una función en Python llamada `main()`, que se trata de un script que se encarga de ejecutar todas las herramientas de Django. Además de la siguiente llamada a dicho script Python:


```
if __name__ == '__main__':
    main()
```

- **project/settings.py**: Como su propio nombre indica, es el archivo de configuración de todo nuestro proyecto. En registraremos todas las Apps que creemos/importemos, la configuración de la base de datos, etc. A continuación, se explican las configuraciones que hemos hecho además de las que ya venían por defecto (todo esto aparece en la página oficial de la documentación de Django en la siguiente dirección url <https://docs.djangoproject.com/en/3.0/ref/settings/>):
 - **DEBUG**: Este parámetro de configuración se trata de un booleano que pondremos en True si estamos en desarrollo, y en False si nos encontramos en Producción como se explica en el punto 4.13 Paso a producción.
 - **INSTALLED_APPS**: A este parámetro de configuración, que es una variable de tipo list (todas las estructuras de datos de Python las encontraremos en la documentación de Python en su página oficial en la siguiente dirección url <https://docs.python.org/3/tutorial/datastructures.html>, le pasamos las Apps que usaremos en nuestro proyecto.
 - **TEMPLATES**: A este parámetro de configuración, que es una variable que se trata de una anidación de estructuras de datos de Python de tipo list y dict. Y en este caso solo tocaremos el list `context_processors`, para declarar ciertas funciones que nos devolverán estructuras de datos de tipo dict, con información recuperada de nuestra base de datos.
 - **ADMINS**: A este parámetro de configuración, que es una variable de tipo list con varios tuple, le pasaremos, por cada tuple, dos datos. El primer dato un nombre de un administrador (no tiene que existir como superuser) y un correo electrónico. A estos administradores se les enviará correos con las excepciones lanzadas por el proyecto cuando tenemos otros parámetros de configuración,

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	--

configurados de cierta forma (leer documentación Django). En este caso no lo usaré pero este parámetro ya viene por defecto.


- **DATABASES:** Este parámetro se trata de un tipo de dato dict dentro de otro dict. En este punto configuraremos nuestra base de datos, y como ya se explicó en el punto 4.4 SGBD o sistema de almacenamiento utilizado, en Django, además de SQLite, pueden utilizarse MySQL, PostgreSQL y Oracle. En nuestro caso se usará SQLite porque a diferencia del resto, solo hay que configurar los parámetros *ENGINE* y *NAME*. Las configuraciones pueden verse en la documentación de Django.
- **LANGUAGE_CODE:** Este parámetro de configuración es un String con el que asignaremos un idioma para Django. Por defecto viene con *en-EN*, nosotros lo pondremos en *es-ES*.
- **TIME_ZONE:** Este parámetro de configuración es un String con el que asignaremos la zona horaria. Nosotros lo pondremos en *Europe/Madrid*.
- **MEDIA_URL:** Este parámetro es un string que maneja los archivos media servidos por el parámetro *MEDIA_ROOT*. En nuestro caso le pasaremos *media*.
- **MEDIA_ROOT:** Parámetro con el que configuraremos la ruta absoluta de la raíz del proyecto con el string */media/*. Este parámetro lo usaremos para guardar los archivos media que subamos en nuestro proyecto mediante el Administrador Django.
- **LOGIN_REDIRECT_URL:** Este es un parámetro al que le pasaremos un string, al cual nos redirigirá cuando hagamos login en nuestra página. En nuestro caso le pasaremos *home*.
- **LOGOUT_REDIRECT_URL:** Este es un parámetro al que le pasaremos un string, al cual nos redirigirá cuando hagamos logout en nuestra página. En nuestro caso le pasaremos *home*.
- **project/asgi.py:** Este archivo es parecido a *wsgi.py*, pero para comunicarse con servidores web asíncronos y aplicaciones. En mi caso no lo uso en ningún punto del proyecto.
- **project/urls.py:** En este archivo encontraremos una variable de tipo list, en la que se le pasarán funciones importadas de *django.urls*, a las que pasaremos como parámetros el path de nuestras vistas, las funciones declaradas con nuestros archivos *views.py*, y un name entre los que destacaremos:
 - **home:** Debido a este name, le hemos pasado ese string a nuestros parámetros de configuración *LOGIN_REDIRECT_URL* y *LOGOUT_REDIRECT_URL*.
 - **accounts:** Estos dos path() son algo diferentes, esto es porque una de las ventajas de Django, es que cosas como el Login/Logout ya vienen creados. Entonces importaremos la función *include()* de *django.urls* y le pasaremos como parámetros, los string que podemos ver en el archivo.
 - **admin:** Este path ya viene por defecto cuando creamos el proyecto, esto es porque este es el path con el que accederemos a nuestro Administrador Django.

	<p style="text-align: right;">CÓDIGO DEL PROYECTO: 1070</p> <p>TÍTULO: Framework de desarrollo web con Django</p> <p>AUTOR: Israel García Cabañeros</p>
---	--

- **project/wsgi.py:** Este archivo solo lo utilizaremos cuando hagamos el pase a producción como se explica en el punto 4.13 Paso a producción. Se usa para comunicar al proyecto Django a comunicarse con el servidor web.
- **app/migrations:** En este directorio se genera un archivo Python con la migración de los modelos pertenecientes a la app.
- **app/templates:** En este directorio están todos los archivos HTML. Entre ellos tenemos:
 - **base.html:** Este archivo contiene el head, body y footer común para todos los archivos .html que extienden de este.
 - **.html:** Estos son el resto de archivos HTML que encontramos en el directorio templates. Para extender a un HTML de otro se usa uno de los llamados template tags (podemos encontrar la documentación de los template tags en la dirección url <https://docs.djangoproject.com/en/3.0/ref/templates/builtins/>), para este caso se usará `{% extends 'app/base.html' %}`.
- **app/urls.py:** Este archivo lo crearemos si queremos hacer una variable de tipo list con todos los pattern y heredarlos desde project/urls.py, podemos ver un ejemplo con *registration/urls.py*. Esto se hace a preferencia del desarrollador, en este caso se ha hecho de las dos formas para mostrar que se puede hacer todo en un archivo o hacerlo por apps que es más ordenado y esto ayuda en la escalabilidad tanto del proyecto como de la propia app.
- **app/models.py:** En este archivo se declararán todos los modelos de la app que posteriormente mapearemos y generaremos las bases de datos migrándolos. En este archivo encontraremos clases, cada una de ellas como un modelo.
- **app/tests.py:** En este archivo haremos pruebas unitarias para testear la app.
- **app/apps.py:** Esta es una configuración general para todas las apps, así que no la tocaremos.
- **app/processors.py:** Este archive ya se ha explicado anteriormente, pero es donde haremos las consultas a las bases de datos y que declararemos en el archivo settings.py, en el parámetro de configuración *TEMPLATES* y lo podremos usar en todo el proyecto.
- **app/views.py:** En este archivo definiremos funciones en Python que nos devolverán una ruta del archivo HTML requerido en cada función. Dichas funciones las usaremos en el archivo *project/urls.py*.
- **app/admin.py:** En este archivo registraremos los modelos hechos en *models.py* para que nos aparezcan en el administrador Django. Crearemos un *ModelAdmin* heredado del paquete *admin*, y en este caso tendrán la propiedad *readonly_fields* que funcionará como un variable de tipo *tuple* para que al crear un nuevo objeto en el administrador Django, en el formulario salga como campo no rellenable.
- **project/media:** En este directorio guardaremos los archivos media (.png, jpeg, etc.) que subamos mediante formularios a nuestro proyecto.
- **project/db.sqlite3:** Por último, tenemos este archivo .sqlite3, el cual se genera cuando hacemos las migraciones de los modelos.

6.7 Análisis y diseño del supuesto práctico

A continuación se mostrará el catálogo de requisitos del proyecto:

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

1. Tendrá persistencia de datos con el Sistema Gestor de Bases de Datos, SQLite.
2. Tendrá un administrador Django.
3. Habrá tres tipos de usuarios, superusuarios, vendedores y clientes.
4. Un usuario no autenticado podrá loguearse y registrarse.
5. Un usuario de tipo superusuario o vendedor podrá acceder al administrador Django.
6. Un usuario de tipo superusuario podrá administrar todo lo referente a usuarios y los modelos.
7. Un usuario de tipo vendedor podrá administrar todo lo referente a los modelos.
8. Los superusuarios tendrán acceso total al administrador Django, incluyendo la autenticación y autorización de grupos y usuarios.

Debido a un tema de tiempo no ha podido realizarse toda la funcionalidad pensada para la página web, los siguientes puntos son un planteamiento de lo que debería ser la página completa.

9. Un usuario autenticado podrá añadir productos al carrito.
10. Un usuario no autenticado podrá añadir productos al carrito.
11. Un usuario no autenticado deberá autenticarse antes de completar la compra.
12. Un usuario autenticado tendrá acceso a su perfil.
13. Un usuario autenticado podrá modificar su contraseña.


6.8 Modelo de clases

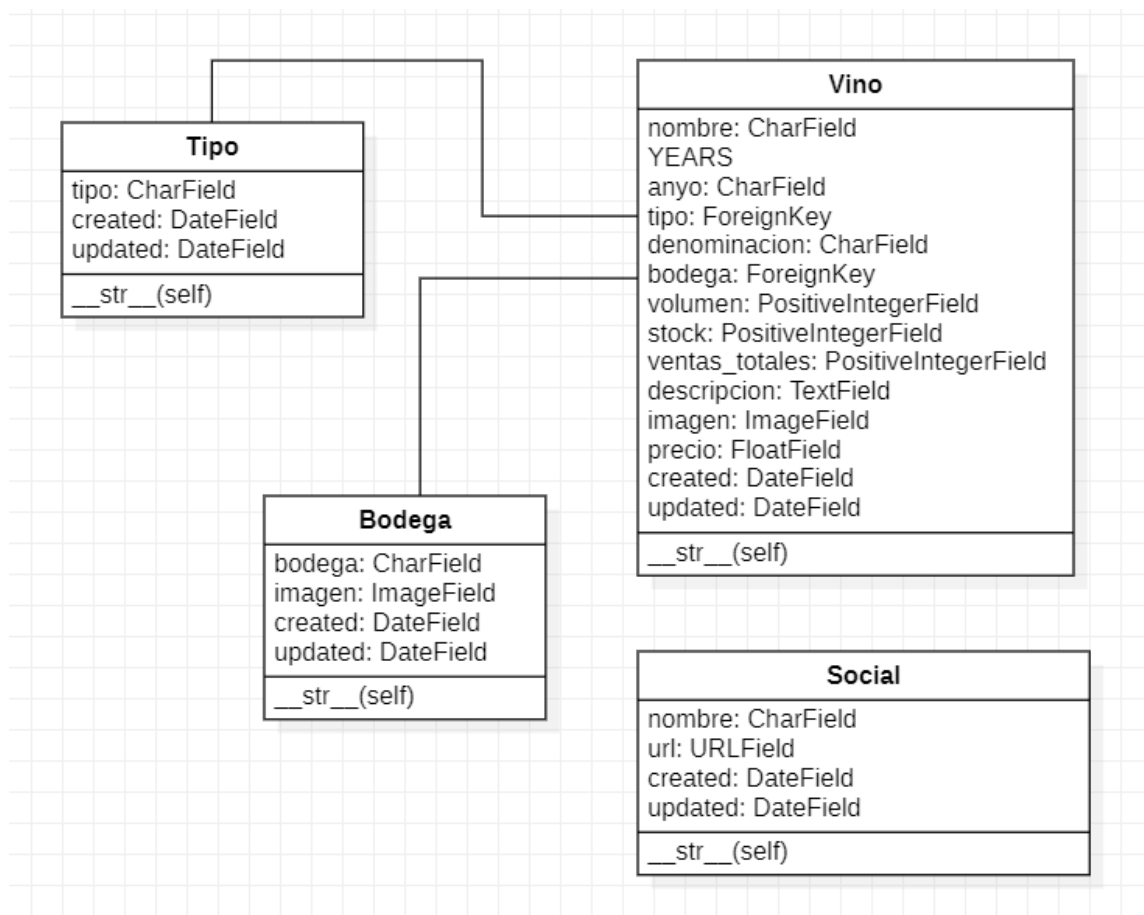
A continuación se mostrará nuestro diagrama de clases, esto lo podemos encontrar en la App core/models.py (Tipo, Bodega y Vino), y en la App social/models.py.

Lo que hará Django al ejecutar los siguientes comandos:

- `python manage.py makemigrations`
- `python manage.py migrate`

Será mapear los archivos mencionados anteriormente para crear automáticamente en la base de datos las tablas en las que se almacenará nuestra información.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros



Este diagrama está creado con la aplicación de escritorio StarUML (se dejará el .mdj con el resto de documentos).


Una cosa que no vemos son los modelos para la gestión de los usuarios, esto es porque Django ya crea automáticamente todas estas tablas en nuestra base de datos.

Por último, una cosa que podemos observar en la imagen anterior es que existe un modelo llamado Social. Esto se debe a que si nos ponemos en la situación de que el cliente que compra la aplicación quiere añadir, modificar, eliminar sus redes sociales referentes a la web, no tiene porque llamarnos y modificar el código cada vez que haga esto. Mediante el administrador de Django el mismo cliente puede administrar sus redes sociales ya que tendrá permisos para acceder a este apartado.

6.9 Modelo de casos de uso

En esta aplicación tendremos dos modelos de casos de uso. En primer lugar tendremos el modelo de nuestro administrador Django:

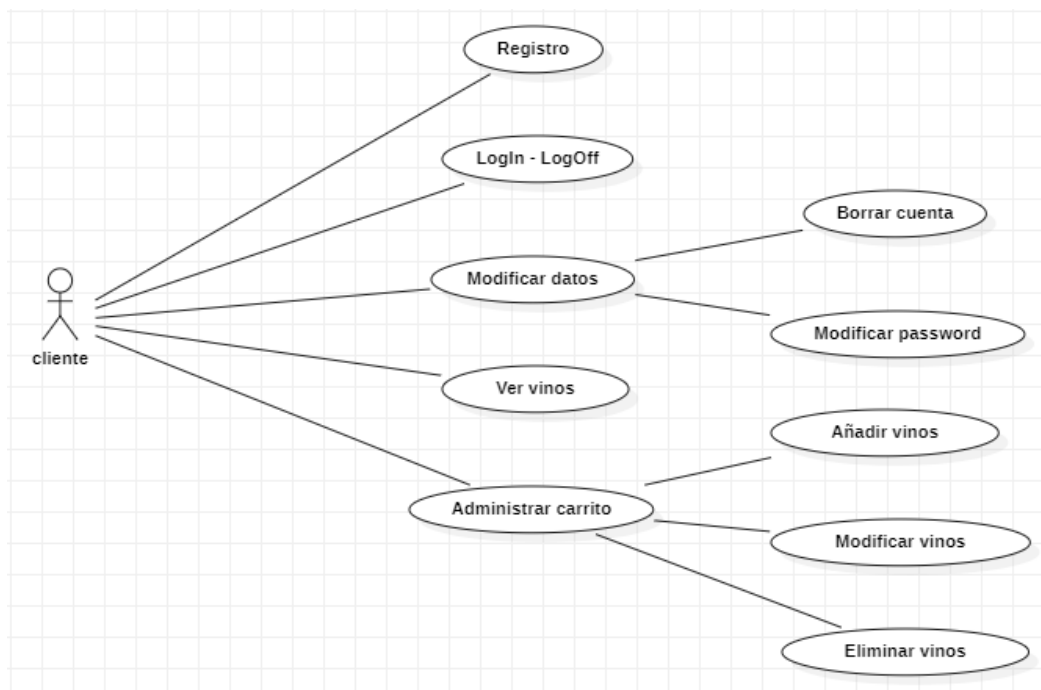
Este diagrama está creado con la aplicación de escritorio StarUML (se dejará el .mdj con el resto de documentos, ya que es demasiado grande para pegarlo en este documento).

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

En este caso entremos dos tipos de actor, el primer actor es el superusuario, el cual tiene control absoluto del administrador Django, incluso puede crear nuevos superusuarios. Con este actor, también podemos gestionar grupos con ciertas características para cada tipo de usuarios, como por ejemplo vendedores que veremos a continuación.

El otro actor es el vendedor, este tipo de actor tiene todos los privilegios relacionados con la gestión de la tienda, es decir, los mismos privilegios que el superusuario pero con la diferencia de que el vendedor no puede acceder a la parte del administrador Django de Autenticación y autorización.

A continuación se muestra el otro modelo, referente a la propia página web:




Este diagrama está creado con la aplicación de escritorio StarUML (se dejará el .mdj con el resto de documentos).

Este actor llamado cliente no tiene permisos para acceder al administrador Django y se crearán con esta característica desde el formulario de registro de la página web.

6.10 Árbol de navegación

6.11 Bases de datos

La estructura de las tablas de la base de datos SQLite, se generarán con el mapeado de los modelos de Django. Lo primero que debemos saber es que podremos acceder a esta base de datos mediante el siguiente comand:

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

- `python manage.py dbshell`

Al entrar en la consola de SQLite, si no conocemos los comandos, ejecutaremos:

- `.help`

Este comando nos mostrará todos los posibles comandos de SQLite.

El siguiente comando se usará para mostrar la configuración de como se nos mostrará la información de las tablas cuando hagamos una consulta SQL:

- `.show`

```
sqlite> .show
echo: off
eqp: off
explain: auto
headers: off
mode: list
nullvalue: ""
output: stdout
colseparator: "|"
rowseparator: "\n"
stats: off
width:
filename: C:\Users\igarcia\Desktop\Django_Project\Django_Proyecto_Final\Django_Israel_Garcia_Cabaneros\db.sqlite3
```

Nos interesan dos puntos de dicha configuración, uno es `headers`, el cual cambiaremos a *on*. Y el otro es `mode`, el cual cambiaremos a *column*.

Con la configuración anterior las consultas se mostrarán así:


```
sqlite> select * from core_bodega;
1|Ramón Bilbao|bodegas/ramon-bilbao-518_g_B3QJi0d.jpg|2020-04-30|2020-04-30
2|Soto Manrique|bodegas/soto-manrique-64512_d_9vVZWC8.png|2020-04-30|2020-04-30
3|Bodegas Martín Codax|bodegas/martin-codax-1462738_p.jpg|2020-04-30|2020-04-30
```

Modificaremos la configuración con los siguientes comandos:

- `.headers on`
- `.mode column`

La configuración *headers* será para que la consulta nos muestre los nombres de las columnas de la tabla.

La configuración *mode* será para que nos muestre la consulta por columnas correctamente tabuladas.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

A continuación, se muestra la misma consulta anterior pero con la nueva configuración:

```
sqlite> select * from core_bodega;
```

id	bodega	imagen	created	updated
1	Ramón Bilbao	bodegas/ramon-bilbao-518_g_B3QJi0d.jpg	2020-04-30	2020-04-30
2	Soto Manriqu	bodegas/soto-manrique-64512_d_9vVZWC8.	2020-04-30	2020-04-30
3	Bodegas Mart	bodegas/martin-codax-1462738_p.jpg	2020-04-30	2020-04-30

También necesitamos saber que tablas tenemos en nuestra base de datos, esto lo haremos con el comando:

- `.tables`

```
sqlite> .tables
auth_group          core_tipo
auth_group_permissions core_vino
auth_permission     django_admin_log
auth_user           django_content_type
auth_user_groups    django_migrations
auth_user_user_permissions django_session
core_bodega         social_social
```

Como podemos observar, tenemos las tablas con los modelos mapeados que hicimos en Django, con el prefijo de la App en la que se encuentran, y otras tablas que genera Django como por ejemplo de los usuarios y grupos de los que ya hemos hablado anteriormente.

También, necesitaremos saber en ocasiones la estructura de una tabla, esto lo haremos ejecutando una sentencia SQLite (todas las sentencias de SQLite tendrán que terminar en punto y coma (;)):


- `pragma table_info("tablename");`

```
sqlite> pragma table_info("core_bodega");
```

cid	name	type	notnull	dflt_value	pk
0	id	integer	1		1
1	bodega	varchar(70)	1		0
2	imagen	varchar(10)	0		0
3	created	date	1		0
4	updated	date	1		0

Por último, para salir de la consola SQLite, podremos hacerlo de dos formas, ejecutando el comando:

- `.exit`

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Y presionando Ctrl + C, pero esta segunda no es recomendable ya que es como forzar el cerrado.

6.12 Presupuesto y financiación

En el desarrollo de la página web se ha tardado unos 7 días (9 horas al día), aún que le falta funcionalidad y aún le quedaría más tiempo de trabajo. Poniendo como referencia que el precio mínimo en la página web de freelance <https://www.malt.es/> es de 90€/día, sale un precio de 630€.

Por otro lado, el costo de la aplicación web serían tanto el hosting, como el dominio. Pero en este caso hemos elegido un hosting gratuito, además, de un subdominio también gratuito.

En cuanto a la formación para el proyecto no viene incluida en gastos.

6.13 Paso a producción


PythonAnywhere. Se trata de una web de hosting especializada en proyectos desarrollados en Python, y que además, nos da la posibilidad de subir estos proyectos de forma completamente gratuita. Aún que para tener más prestaciones como más de un dominio entre otras, hay distintos planes mensuales, más acordes a lo que necesite cada usuario.

Creación de la cuenta en PythonAnywhere.

Lo primero que deberemos hacer, es acceder a la página de PythonAnywhere, <https://www.pythonanywhere.com/>. Al entrar, veremos un botón verde en el que pone *Start running Python in less than a minute!*. Una vez hayamos pulsado en el botón, nos llevará a los planes que se comentaron antes. Como vemos, no crearemos dominios propios al usar esta web, si no que usaremos subdominios de *.pythonanywhere.com.

Pulsaremos en *Create beginner account*, este botón nos habrá llevado a un formulario de registro, siendo el campo *Username*, el campo que estará en la primera parte de nuestro dominio. Nos registraremos e iremos a el correo que hayamos puesto en el formulario para validar la cuenta.

Una vez dentro nos encontraremos una pantalla como la que se muestra a continuación, la cual nos muestra las características que nos ofrece el plan gratuito.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Send feedback Forums Help Blog Account Log out

pythonanywhere

Dashboard Consoles Files Web Tasks Databases

Dashboard

Welcome, [webpersonaldaw](#)

CPU Usage: 0% used – 0.00s of 100s. Resets in 23 hours, 51 minutes [More Info](#)

File storage: 0% full – 48.0 KB of your 512.0 MB quota

Upgrade Account

Recent Consoles

+ 5 -

You have no recent consoles.

New console:

\$ Bash

>>> Python

More...

Recent Files

+ 5 -

You have no recently edited files.

+ Open another file

Browse files

Recent Notebooks

+ 5 -

Your account does not support Jupyter Notebooks. [Upgrade your account](#) to get access!

All Web apps

You don't have any web apps.

Open Web tab

Desplegando nuestro proyecto.

Al igual que hicimos anteriormente con Miniconda, deberemos crear un nuevo entorno virtual, pero en vez de desarrollo, de producción, y esto lo haremos con los siguientes pasos.

En nuestra página principal de PythonAnywhere, en el botón de *Consoles* que encontraremos arriba a la derecha, accederemos a una pantalla como la que se mostrará a continuación, en la que deberemos pulsar en el botón *bash*. Con el que nos aparecerá una consola de comandos.

Send feedback Forums Help Blog Account Log out

pythonanywhere

Dashboard **Consoles** Files Web Tasks Databases

CPU Usage: 0% used – 0.00s of 100s. Resets in 23 hours, 26 minutes [More Info](#)

Start a new console:

Python: [3.8 / 3.7 / 3.6 / 3.5 / 2.7](#) IPython: [3.8 / 3.7 / 3.6 / 3.5 / 2.7](#) PyPy: [2.7](#)
 Other: [Bash](#) | [MySQL](#)
 Custom: [+](#)

Your consoles:


You have no consoles. [Click a link above to start one.](#)

Consoles shared with you

No-one has shared any consoles with you :-)

Running processes

Fetch process list

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

En dicha consola, ejecutaremos el siguiente comando para crear un nuevo entorno virtual (En nuestro caso como el entorno de desarrollo fue con la versión 3.7.6 de Python, pero en PythonAnywhere la última versión disponible es la 3.7, así que lo haremos con esta).

- `mkvirtualenv --python=python3.7 environmentname`

Al momento de crear el entorno virtual, la consola nos pondrá directamente dentro de este entorno virtual, pero necesitaremos un comando para el resto de veces que queramos entrar en él, lo haremos de la siguiente forma.

- `workon environmentname`

Para salir del entorno virtual, ejecutaremos el siguiente comando.







- `deactivate`

Para ver la lista de paquetes y sus versiones correspondientes que tenemos instalados en el entorno, ejecutaremos el siguiente comando.


- `pip list`

Con el comando pip, que es un paquete que se instala automáticamente al crear el entorno, tanto en desarrollo, como en producción, podemos instalar los paquetes que necesitaremos (django, Pillow, django-ckeditor) para que la aplicación web funcione correctamente, pero en este caso lo haremos de una forma más óptima.

Iremos a la raíz del proyecto, donde tenemos el .git, y crearemos un nuevo .txt que se llamará requirements.txt, el cual rellenaremos con los paquetes y sus respectivas versiones que queremos instalar.

 .git	4/1/2020 11:49 AM	File folder	
 webpersonaldaw	4/1/2020 11:48 AM	File folder	
 .gitignore	4/1/2020 11:47 AM	Text Document	2 KB
 LICENSE	4/1/2020 11:47 AM	File	35 KB
 README.md	4/1/2020 11:47 AM	MD File	1 KB
 requirements.txt	4/1/2020 11:32 AM	Text Document	1 KB

Dejaremos el archivo de la siguiente forma:

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

```

1 pillow==7.0.0
2 django==3.0.4
3 django-ckeditor==5.9.0

```

A continuación, iremos desde la consola de comandos o cmd al directorio mostrado anteriormente, y desde ahí, ejecutaremos los siguientes comandos:

- `git add requirements.txt`
- `git commit -m "mensaje del commit"`
- `git push origin master`

Hecho esto, ya tenemos nuestro archivo `requirements.txt` en nuestro repositorio en GitHub.

Ahora nos dirigiremos a nuestra consola bash de pythonanywhere y ejecutaremos el siguiente comando:

- `git clone url_github_repositorio`

Después de hacer el clone en nuestro pythonanywhere podemos ejecutar cualquiera de los dos comandos siguientes:

- `ls ->` Que nos mostrará el archivo `README.txt` (Si lo creamos cuando hicimos el repositorio o posteriormente), y el directorio raíz de nuestro proyecto.
- `ls -la ->` Además de mostrarnos lo mencionado anteriormente, nos mostrará los archivos y directorios ocultos, es decir, los que comienzan con un punto (`.`), como por ejemplo `.cache` o `.pythonstartup.py`.

Iremos al directorio raíz de nuestro proyecto con:

- `cd directorio`


Y desde aquí ejecutaremos el siguiente comando para la instalación de los paquetes con sus respectivas versiones del archivo `requirements.txt`:

- `pip install -r requirements.txt`

La instalación tardará unos minutos.

Ahora que tenemos nuestro proyecto y los paquetes necesarios, ya podemos introducir comandos que el archivo `manage.py` sea capaz de ejecutar, uno de ellos es el que se mostrará a continuación, el cual sirve para que nos muestre posibles errores o warnings que hay que solucionar:

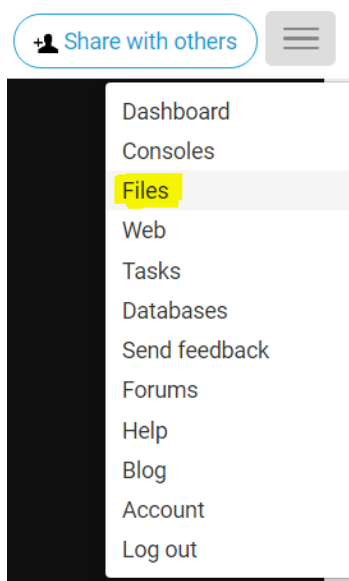
- `python manage.py check --deploy`

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Nos saldrán varios errores, de los cuales hay dos que son más importantes que el resto:

1. **(security.w018) You should not have DEBUG set to True in deployment.** Esto porque en producción no puede dejarse el modo DEBUG, ya que si un usuario fuerza un error, puede ver configuración de nuestro proyecto que no queremos que vea, es decir, esto sería una vulnerabilidad.
2. **(security.w020) ALLOWED_HOSTS must not be empty in deployment.** Esta variable es la que muestra desde que servidores o dominios/subdominios desde los cuales podemos acceder a nuestra página. En nuestro caso será el nombre del subdominio que pusimos anteriormente.

Para modificar el archivo correspondiente, *settings.py*, iremos al menú de arriba a la derecha que encontraremos desde nuestra consola bash pythonanywhere, y pincharemos en *Files*.




Navegaremos entre los directorios de nuestro proyecto hasta que encontremos el archivo *settings.py*, el cual podremos editar como si estuviéramos en nuestro IDE desde local. Modificaremos los dos campos siguientes y los dejaremos como se muestra en la imagen:

```
DEBUG = False
```

```
ALLOWED_HOSTS = ['subdominio.pythonanywhere.com', 'localhost', '127.0.0.1']
```

Los parámetros *localhost* y *127.0.0.1*, los añadiremos si se da el caso de queremos trabajar con el `DEBUG = False` en local.

Si volvemos a ejecutar el check de los errores ya no aparecerán los errores o warnings de mayor importancia que se comentaron anteriormente.

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Ahora que tenemos la configuración mínima recomendada para realizar un despliegue, crearemos una App que se encargue de mantener el servicio de Django en marcha. Para crear una App en pythonanywhere, desplegaremos el menú mostrado anteriormente, cuando fuimos a *Files*, pero ahora pincharemos en *Web*. O en el caso de que aún no hayamos salido de *Files*, encontraremos un menú arriba dónde también encontraremos la pestaña *Web*.

Dashboard Consoles **Files** **Web** Tasks Databases

Encontraremos un botón azul en el que pone *Add a new web app*, en el que pincharemos.

En la ventana central, realizaremos los siguientes pasos:


1. *Next*.
2. *Manual configurations (including virtualenvs)*.
3. *Python 3.7* (que es la versión que instalamos en ambos entornos virtuales).
4. *Next*.

Y después de seguir estos pasos ya tenemos nuestra App creada, pudiendo así acceder ya a nuestro subdominio en el que nos va a aparecer una pantalla por defecto de pythonanywhere.

Si bajamos hasta abajo en la pantalla a la que nos ha llevado pythonanywhere después de crear la App, veremos un apartado *Code*, el cual tiene un enlace al *wsgi.py*, que es un archivo que crea Django por defecto, el cual, si pinchamos en él, tiene el código HTML con el *Hello, World!* que nos muestra actualmente pythonanywhere.


Code:

What your site is running.

Source code:	Enter the path to your web app source code	
Working directory:	/home/webpersonaldaw/	Go to directory
WSGI configuration file:	/var/www/webpersonaldaw_pythonanywhere_com_wsgi.py	
Python version:	3.7 	

Pinchamos en *Enter the path to your web app source code*, donde nos aparecerá un input, en el que pegaremos la ruta al directorio donde se encuentra el archivo *manage.py*. Esta ruta podemos conseguirla yendo a dicho directorio desde la consola bash pythonanywhere, y ejecutando el siguiente comando:

- `pwd`

	CÓDIGO DEL PROYECTO: 1070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

Debajo del apartado *Code* encontraremos otro apartado llamado *Virtualenv*, en el cual pegaremos la ruta obtenida ejecutando el comando mostrado a continuación, con el entorno virtual activado, y eliminando la parte de la ruta */bin/Python*:

- which Python

Ahora ya podemos editar el archivo *wsgi.py*, al que podemos acceder desde el enlace de la captura anterior. Este archivo lo vaciaremos (eliminaremos su contenido por completo), y lo completaremos con el siguiente código Python:

```
import os
import sys

# mysite lo sustituiremos por /dos_ultimas/partes,
# que obtendremos de ejecutar pwd en la consola de pythonanywhere
# desde el directorio donde tenemos manage.py.
path = os.path.expanduser('~mysite')

if path not in sys.path:
    sys.path.append(path)

# mysite.setting deberá cambiarse al nombre de nuestro proyecto
# seguido de .settings.
os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'


from django.core.wsgi import get_wsgi_application
from django.contrib.staticfiles.handlers import StaticFilesHandler

application = StaticFilesHandler(get_wsgi_application())
```

URL de la página web en producción: <http://igcproyectodjangodaw.pythonanywhere.com/>

Usuario	Contraseña	Tipo
admin	paso1234	Administrador
vendedor1	paso1234	Vendedor (Grupo Vendedores)

Los clientes no staff (sin acceso a administrador Django) se crearán a partir de un registro ordinario (formulario de registro de la propia página web).

	CÓDIGO DEL PROYECTO: I070
	TÍTULO: Framework de desarrollo web con Django
	AUTOR: Israel García Cabañeros

7 Conclusiones y líneas futuras

Las conclusiones que se han obtenido de este proyecto son:

- La importancia de aprender el lenguaje de programación en el que está basado el Framework que se va a utilizar. Ya sea desde lo más básico, como la declaración y uso de variables, el uso de estructuras de flujo, estructuras de datos, hasta la creación de clases, modelos, etc.
- Las ventajas que conlleva el uso de un Framework. En este caso el mapeo objeto-relacional, la estructura básica del proyecto, la cantidad de cosas que ya están hechas y listas para usar, como el control de autenticación, la configuración del proyecto, etc.
- Y por otro lado, la utilidad de un gestor de paquetes como es Anaconda/Miniconda, ya que permite adaptarse a las versiones de un software de una forma sencilla.