

Eder Lara Trujillo

ANALISTA DE DATOS Y DESARROLLADOR FULLSTACK

ederlatru@gmail.com

aplicación CON PYTHON

USANDO EL FRAMEWORK DJANGO EN WINDOWS

Contenido

[1. PRE-REQUISITOS 3](#_Toc11071897)

[2. INSTALACION DE APLICACIONES NECESARIAS: 3](#_Toc11071898)

[a) INSTALACION DE PYTHON EN WINDOWS 4](#_Toc11071899)

[b) ACTUALIZACION DE PIP EN PYTHON DESDE LA TERMINAL 6](#_Toc11071900)

[c) INSTALACION DE ENTORNO VIRTUAL DE DESARROLLO 7](#_Toc11071901)

[d) INSTALACIÓN DE DJANGO: 10](#_Toc11071902)

[e) COMANDOS Y RECURSOS DE DJANGO 11](#_Toc11071903)

[3. CREACION DEL PROYECTO EN DJANGO: 13](#_Toc11071904)

[4. CARGAR EL PROYECTO AL EDITOR DE TEXTO 15](#_Toc11071905)

[5. CONEXIÓN A LA BASE DE DATOS MONGODB 16](#_Toc11071906)

[a) INSTALACION DE PYMONGO 16](#_Toc11071907)

[b) INSTALACION DE DNS MONGOPY 17](#_Toc11071908)

[c) CONEXION AL CLUSTER DE MONGODB 17](#_Toc11071909)

[d) INSTALACION DEL FRAMEWORK ‘DJONGO’ 18](#_Toc11071910)

[6. CREACION DE SUPERUSER E INTERFAZ ADMINISTRATIVA 19](#_Toc11071911)

[a) CONTEXTO: 19](#_Toc11071912)

[b) EL ARCHIVO SETTINGS.PY 19](#_Toc11071913)

[c) AGREGANDO LA CONEXIÓN A LA BASE DE DATOS: 20](#_Toc11071914)

[d) REALIZAR LA MIGRACION DESDE LA CONSOLA: 21](#_Toc11071915)

[e) CREAR EL SUPER USUARIO DESDE LA CONSOLA: 22](#_Toc11071916)

[f) PROBANDO LAS CREDENCIALES EN EL NAVEGADOR 22](#_Toc11071917)

[7. LAS REGLAS DEL NEGOCIO PARA EL MODELO DE DATOS. 25](#_Toc11071918)

[8. MODELO DE BASE DE DATOS CURRICULUM VITAE -GENERAL- 26](#_Toc11071919)

[9. INICIANDO UN MODULO: 27](#_Toc11071920)

[a) CREACION DE UN MODULO A TRAVES DE LA CONSOLA 27](#_Toc11071921)

[b) MODELS.PY: CREACION DEL PRIMER MODELO 28](#_Toc11071922)

[c) REGISTRAR EL MODELO EN EL ARCHIVO admin.py 30](#_Toc11071923)

[d) INCLUYENDO LIBRERIAS NECESARIAS PARA CONTINUAR CON EL PROYECTO: 33](#_Toc11071924)

[10. CREANDO LA VISTA DE NUESTRA APLICACIÓN. 35](#_Toc11071925)

[a) CARPETAS Y ARCHIVOS DE LA CARPETA CORE: 36](#_Toc11071926)

[b) El archivo view.py 37](#_Toc11071927)

[c) El archivo urls.py 37](#_Toc11071928)

[d) Archivos /static/ y /media/ 37](#_Toc11071929)

[11. IMPLEMENTACION DE FORMULARIOS. 41](#_Toc11071930)

[a) Creación del archivo forms.py: 41](#_Toc11071931)

[b) Creación del archivo que tiene la tupla de las opciones, pqrsf.py: 41](#_Toc11071932)

[c) Creación del modelo 42](#_Toc11071933)

[d) Creación del archivo view.py: 42](#_Toc11071934)

[e) Fusión del formulario con el HTML: 43](#_Toc11071935)

[12. VISTAS BASADAS EN CLASES POO 44](#_Toc11071936)

[a) App Usuarios 44](#_Toc11071937)

PRIMEROS PASOS:

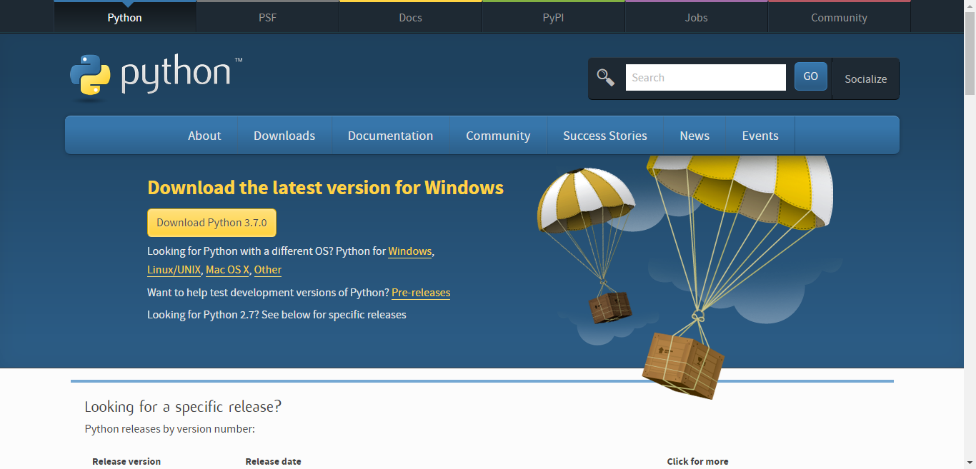
# PRE-REQUISITOS

Para poder desarrollar la actividad propuesta en este tutorial, debemos contar con los siguientes requisitos mínimos:

* Procesador a 1.4 GHz
* Memoria RAM de 2 GB a 800 MHz
* Espacio en Disco Duro 10 GB
* Conocimientos en programación
* Conocimientos de uso de Terminal
* Conocimientos Front-End con frameworks (foundation, bootstrap, materialize, etc.).
* Editor de Código: Visual Studio Code, Atom, Brackets, Sublime Text, etc.

# INSTALACION DE APLICACIONES NECESARIAS:

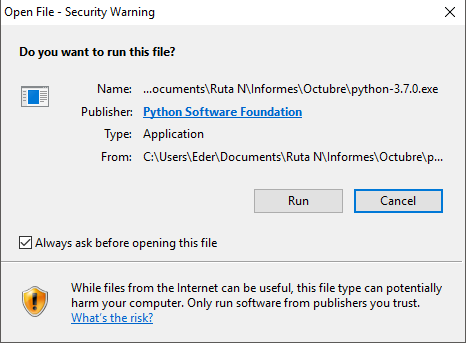
Instalar PYTHON como lenguaje de programación para el desarrollo de la aplicación, para ello ingresamos a la página oficial <https://www.python.org/downloads/> :



Las versiones superiores a la 2.4 de Python, ya vienen con el complemento o plugin necesario para la instalación del framework DJANGO que es el que utilizaremos en esta ocasión. El complemento usado es el PIP, *el PIP es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python*[[1]](#footnote-1)para descargar PIP debemos ingresar a la url: <https://pip.pypa.io/en/stable/> desde esta url podemos observar los comandos para actualizar a la última versión del PIP y desde la consola lanzar los comandos para descargar e instalar los paquetes necesarios para el despliegue de nuestra aplicación, tales como Bootstrap o Foundation.

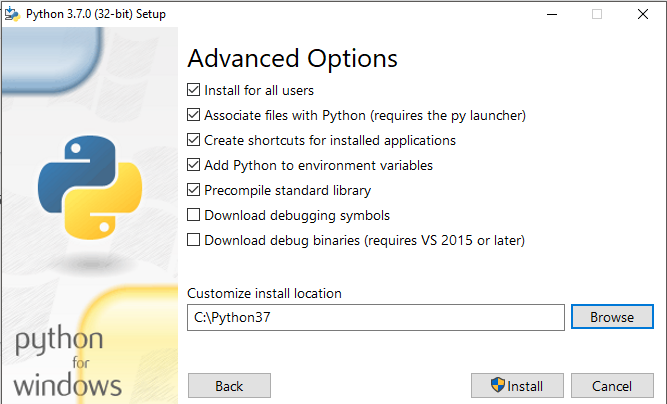
## INSTALACION DE PYTHON EN WINDOWS

Para instalar en Windows, descargaremos la última versión de PYTHON en este caso la 3.7:

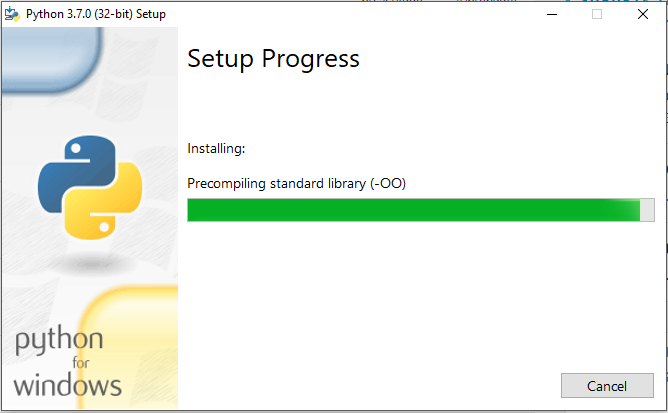




En este punto es bien importante, agregar a Python al PATH, ya que si no lo tenemos agregado debemos hacerlo manualmente. Para ello solo pulsamos y seleccionamos en la opción que dice ***Add Python 3.7 to PATH.*** Y luego si procedemos a instalar, aunque prefiero hacer de forma personalizada, ya que como se ve la carpeta de instalación quedará en la ruta: ***C:\Users\Eder\AppData\Local\Programs\Python\Python37-32***, y es preferible para el manejo de paquetes y complementos que lo ubiquemos en la raíz del disco duro es decir en ***C:\***, así que moveremos la ruta por defecto a la raíz ***C:\Python37***



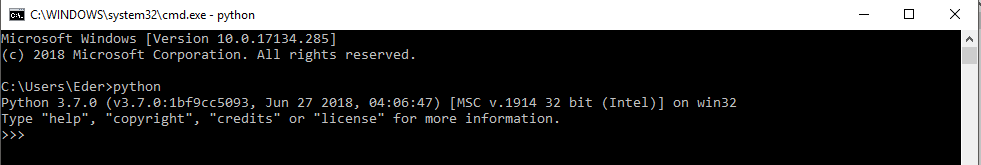
La carpeta Pyton37, la he creado directamente para poder tener la instalación de Python en una carpeta independiente. De esta forma también la hemos instalado para todos los usuarios, si dado el caso tu pc cuenta con diferentes cuentas de usuario.



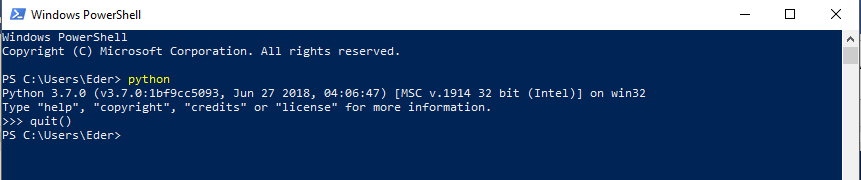
## ACTUALIZACION DE PIP EN PYTHON DESDE LA TERMINAL

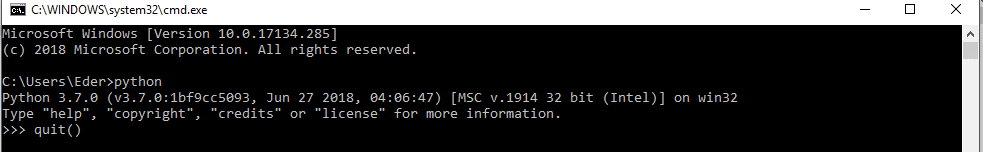
Al instalar Python en Windows, y al incluir el PATH en la instalación, ya podemos acceder al editor desde una consola o desde el CMD (Símbolo del sistema) de Windows, solo basta con escribir ***Python*** y dar enter y se despliega el prompt de Python tal y como lo mostramos a continuación:



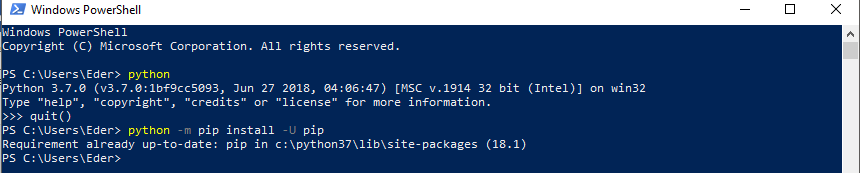


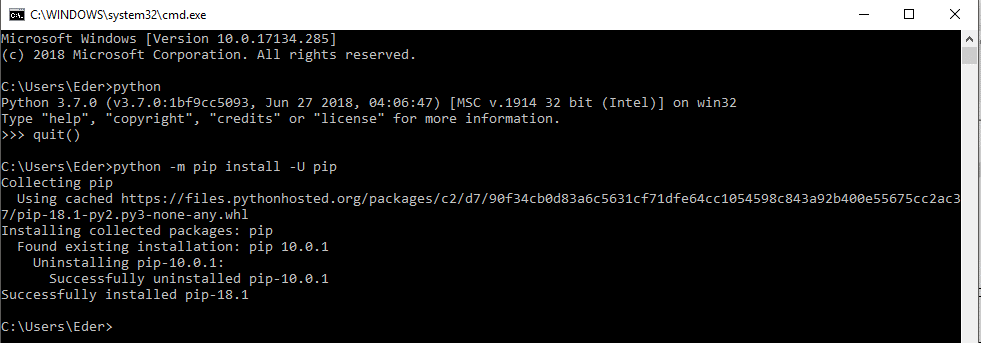
Con esto solo comprobamos que Python ha quedado correctamente instalado en Windows y podemos acceder a el prompt desde la consola o terminal. A continuación, saldremos de la ejecución de Python para poder actualizar el PIP. Para ello escribimos en el prompt de Python el comando quit():





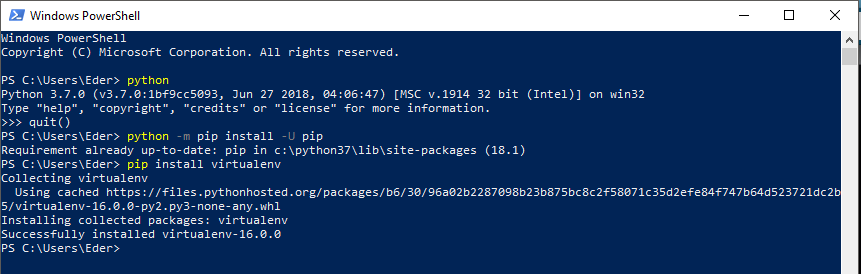
Ahora que hemos salido y estamos en la raíz del usuario, podemos ejecutar comandos, en este caso usaremos uno para actualizar el PIP de Python, solo basta con escribir: ***python -m pip install -U pip:***



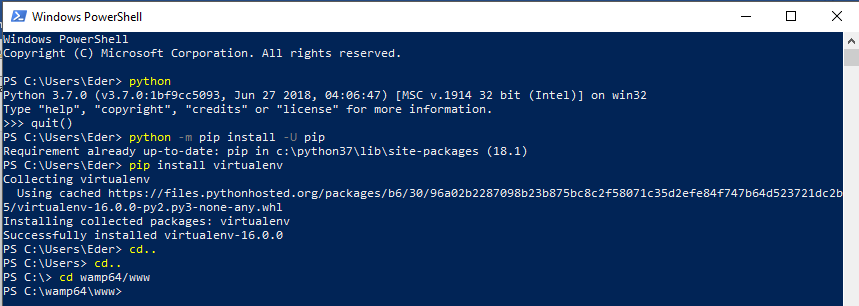


## INSTALACION DE ENTORNO VIRTUAL DE DESARROLLO

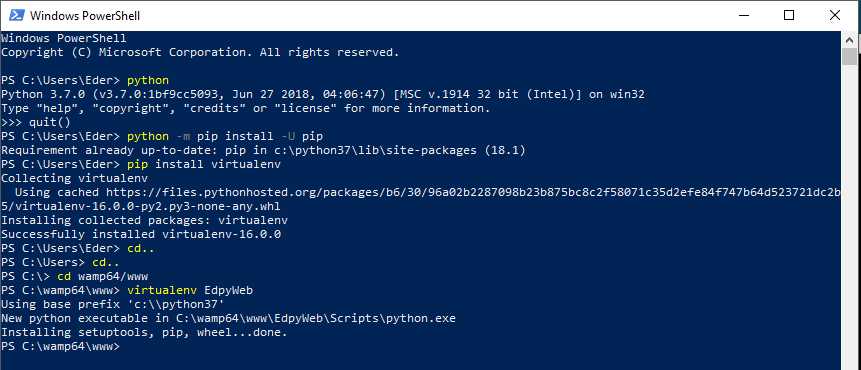
El entorno de desarrollo me permite montar servicios web para el despliegue de la aplicación, para instalarlo solo escribimos ***pip install virtualenv*** y damos enter:



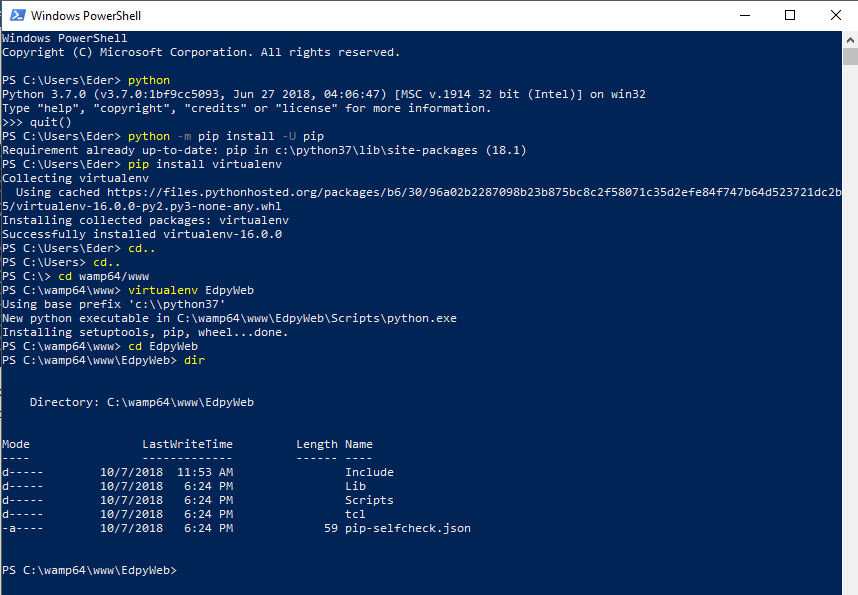
Luego de instalar el entorno virtual podemos hacer el montaje y creación del proyecto a desarrollar. Por lo que vamos a cambiar de directorio para hacer el montaje del entorno virtual para el proyecto en cuestión. En mi caso todos mis proyectos web los tengo en el directorio C:\wamp64\www, así que lo pondremos allí:



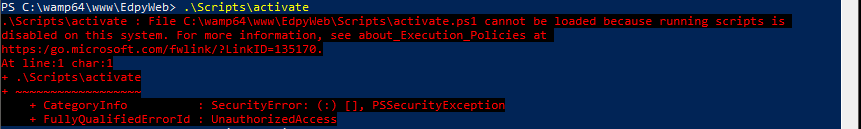
Desde esta ruta crearemos una carpeta con el nombre del proyecto, para este efecto, mi proyecto se llama EdpyWeb:



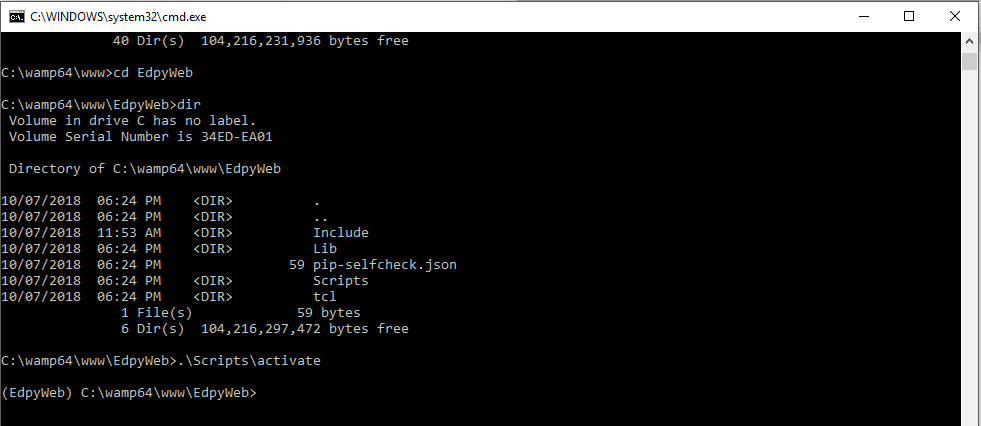
Con el entorno virtual creado, podemos ingresar con el comando ***cd (change directory)*** y observar los componentes descargados con el comando ***dir (directory)*** de esta forma:



Ahora debemos activar nuestra carpeta del entorno virtual, para ello usamos el comando: ***.\Scripts\activate*** tal y como lo muestra la figura:



Si lo corremos desde la terminal PowerShell de Windows, es probable que no se puedan ejecutar scripts de este tipo por políticas de seguridad, pero desde el cmd de Windows si, por lo que lo ejecutaremos desde ahí:

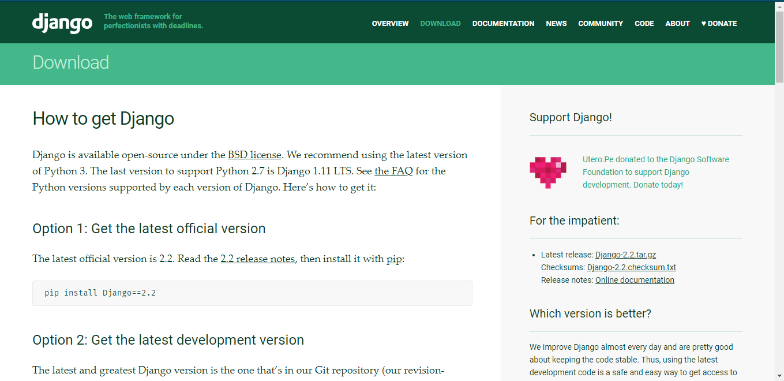


Como podemos observar en la última línea nos ha quedado: (EdpyWeb) C:\wamp64\www\EdpyWeb>

Esto quiere decir que nuestro entorno virtual ha quedado completo y activado.

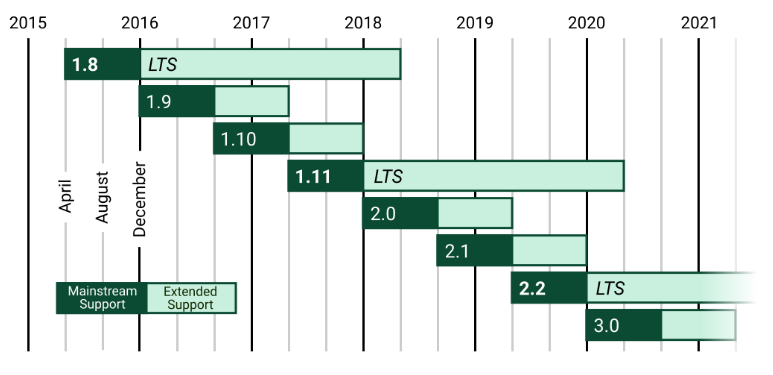
## INSTALACIÓN DE DJANGO:

Para instalar django, vamos a la url oficial de django para verificar la versión de este y poder escoger la mejor versión <https://www.djangoproject.com/download>/ :

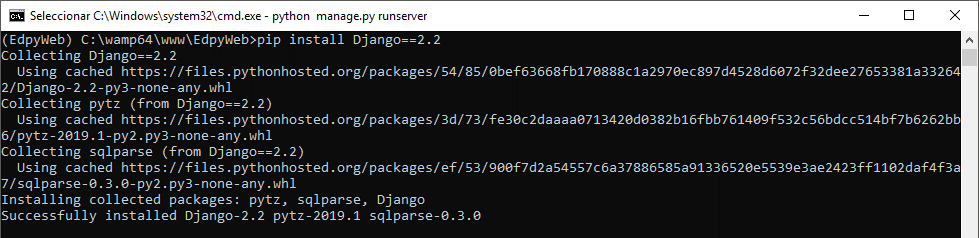


Y como podemos ver, en la pagina esta el comando para descargar e instalar django en la versión mas reciente, para mi caso la versión 2.1.2, pero si nos desplazamos un poco hacia abajo, podemos observar las versiones LTS, que son las de soporte a largo plazo (long term support), y que ofrecen mayores prestaciones a la hora de instalar e implementar django:

Ilustración 1: Tabla de soporte de versiones de Django:



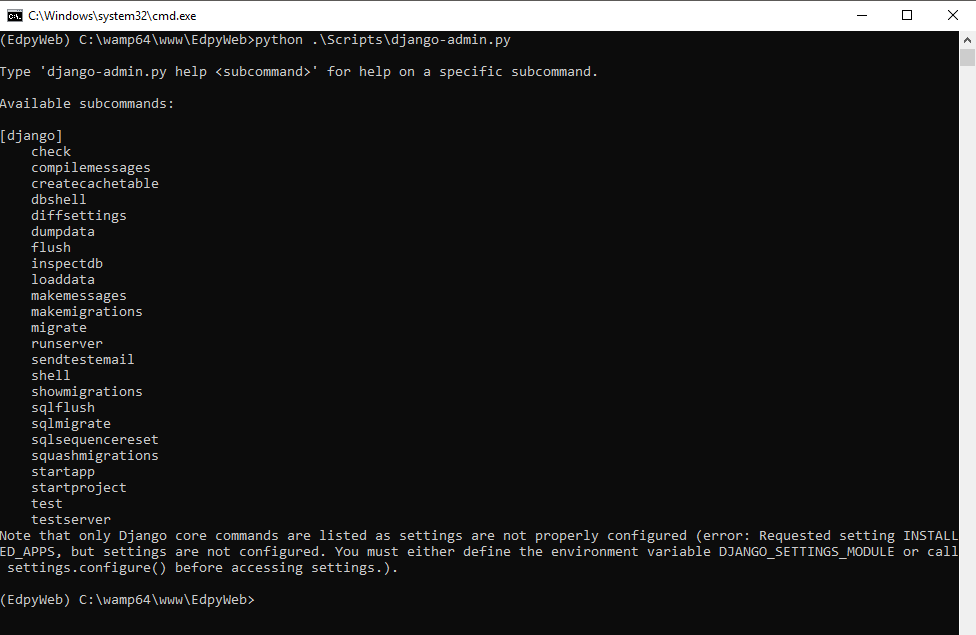
Si hacemos una evaluación de versiones, vemos que la versión 2.2 ofrece un tiempo extendido de soporte mucho mas amplio que la versión 2.1.2 que es la que nos recomienda por defecto django, por lo que tomare esta opción y la procederemos a instalar **pip install Django==2.2** :



## COMANDOS Y RECURSOS DE DJANGO

Ya hemos instalado el framework DJANGO, ya vimos uno de los comandos propios de este, el runserver, manage.py tiene otros comandos para diferentes situaciones, a continuación, te enseño una lista con los comandos de manage.py, estos se pueden acceder a través del comando:

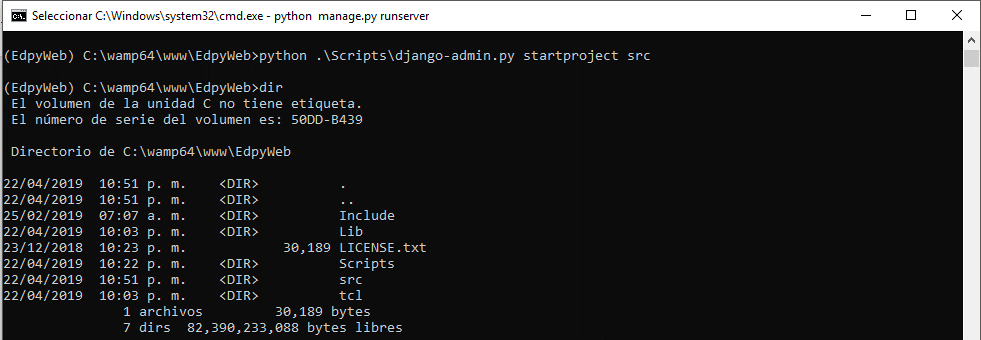
**python .\Scripts\django-admin.py**



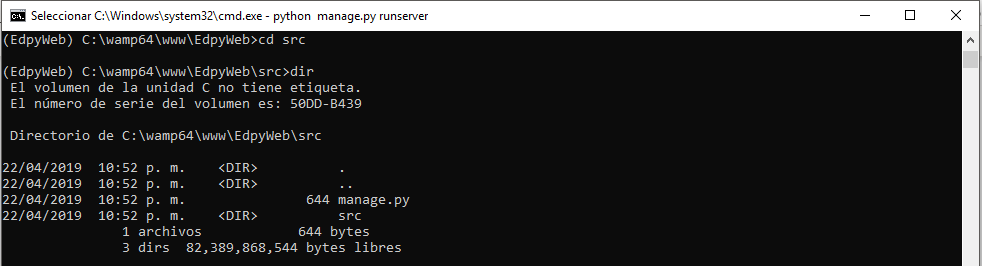
CONSTRUCCION DE LA APLICACION

# CREACION DEL PROYECTO EN DJANGO:

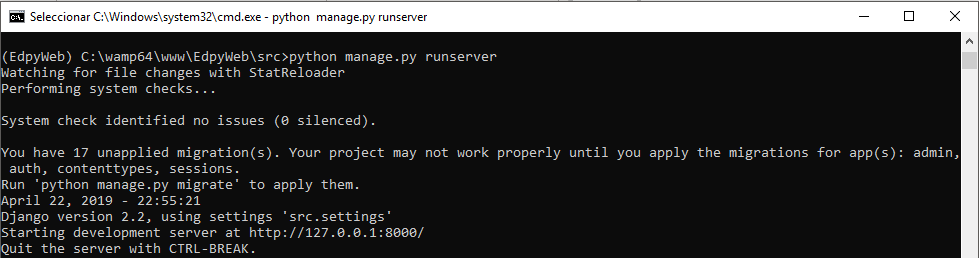
A partir de este momento podemos emplear comandos de Python para iniciar el proyecto. Pues bien continuamos con el siguiente código: **python .\Scripts\django-admin.py startproject src:**



Con esto hemos creado un proyecto en django en una carpeta llamada src. Debemos ingresar y administrar nuestro servicio, para ello vamos a ver que dentro de nuestra carpeta src ha quedado un archivo llamado manage.py, este es el encargado de la administración de django (encender, apagar, agregar usuarios, etc.):

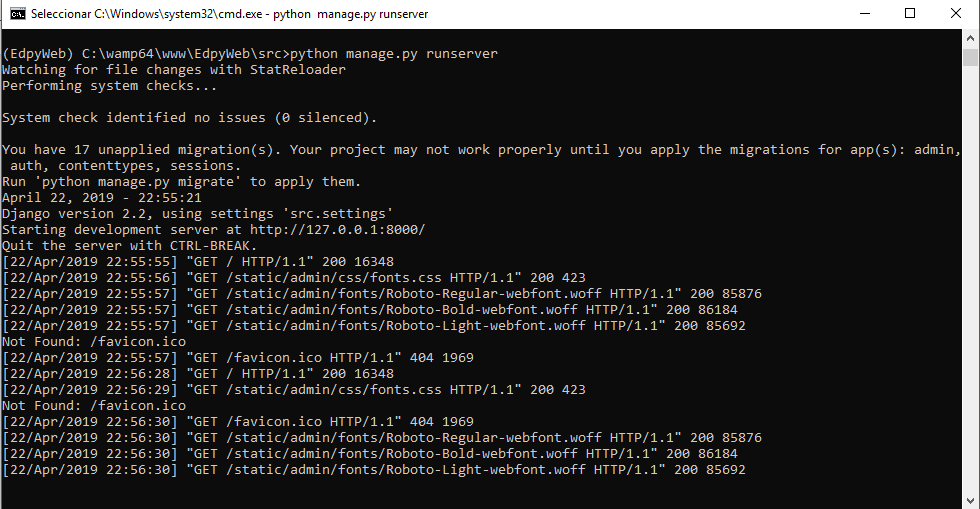


Manage.py es un paquete de administración de django, se encuentran diferentes funciones propias del servicio de django. Ahora encenderemos el servidor local para poder visualizarlo en el navegador, para ello usamos el comando **python manage.py runserver**, este nos arrojara la siguiente dirección: <http://127.0.0.1:8000/> como muestra en la imagen:



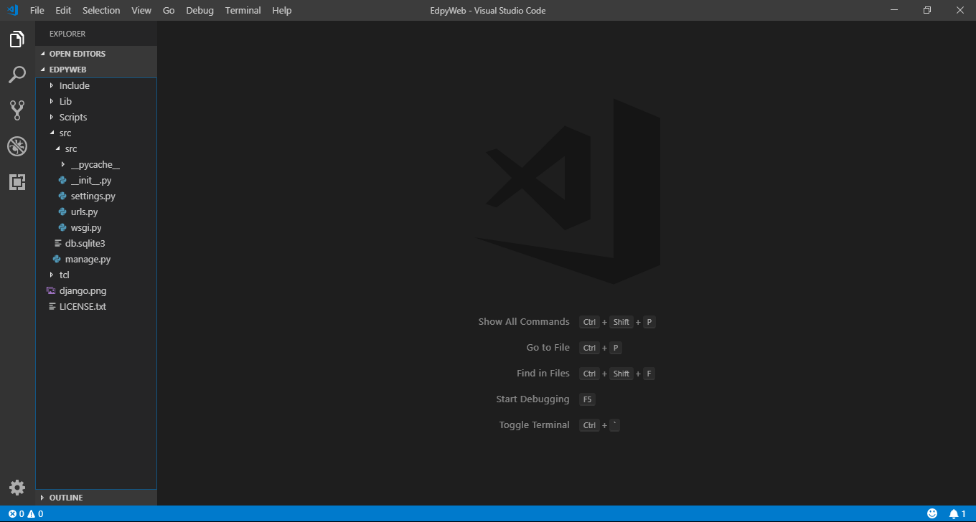
Y en nuestro navegador se verá así:

Mientras esté corriendo o instanciado el servicio, podemos observar los cambios que surjan desde la consola:

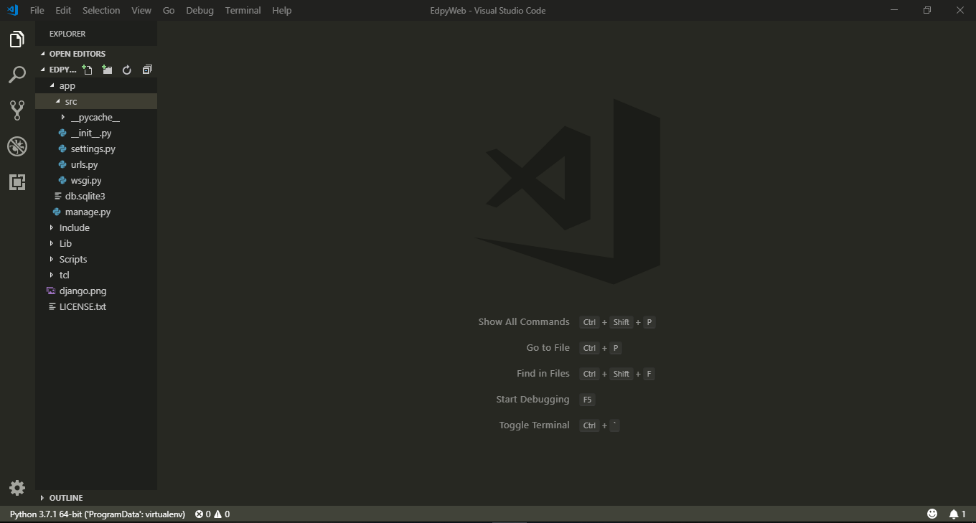


# CARGAR EL PROYECTO AL EDITOR DE TEXTO

Ya podemos agregar nuestro proyecto con en entorno virtual y la carpeta del contenido a un editor de texto, en este caso usare Visual Studio Code. Procedemos entonces a cargar el proyecto al editor:



Desde el entorno del editor podemos observar, que dentro de la carpeta src (abreviación de source), hay otra llamada igual, para poder diferenciar estas carpetas, llamare a la carpeta del primer nivel (src\src); app. Con esto sabremos que dentro de la aplicación (app), tenemos el contenido del entorno del proyecto src. Y nos ha quedado con la siguiente ruta: app/src:



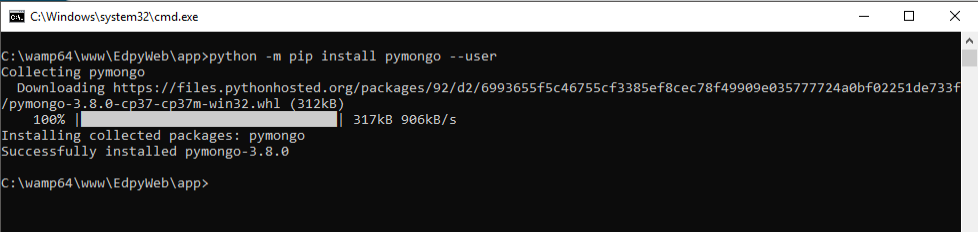
Recomendaciones:

* Guardar el workspace o el área de trabajo, hará más fácil el acceso en una próxima sesión de trabajo.
* Crear un glosario con los comandos más usados en Python para este proyecto.[[2]](#footnote-2)

# CONEXIÓN A LA BASE DE DATOS MONGODB

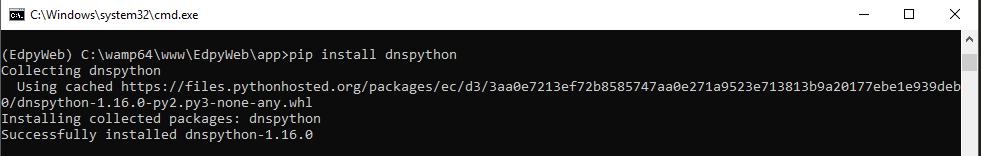
## INSTALACION DE PYMONGO

Para instalar mongopy es necesario corer la siguiente sentencia en el prompt del cmd: **python -m pip install pymongo –user,** asi como lo muestra la imagen:



## INSTALACION DE DNS MONGOPY

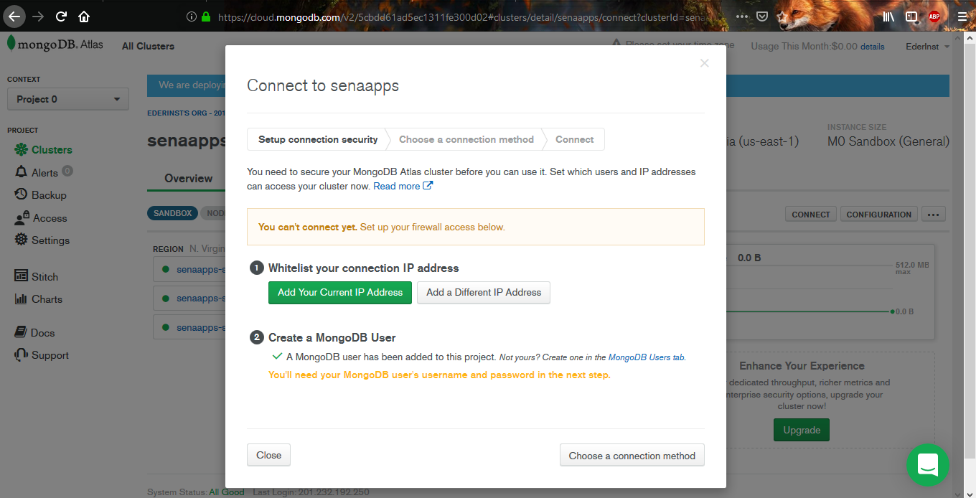
De igual forma, para poder conectarnos al cluster online de MongoDB Atlas, es necesario instalar el conector DNS con el comando: **python -m pip install dnspython –user,** desde el entorno virtual encendido:

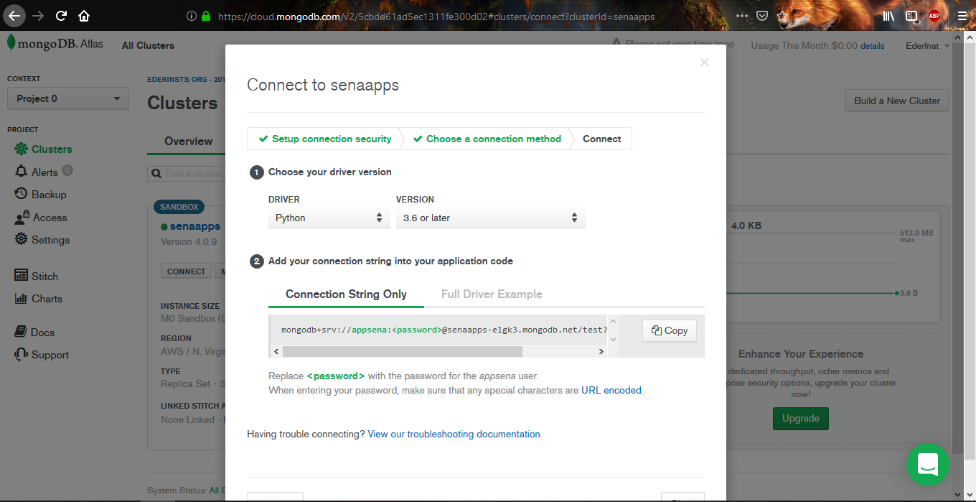


## CONEXION AL CLUSTER DE MONGODB

En este caso lo que debemos hacer es traer el srt de conexion desde mongoAtlas, para eso damos clic en connect, y seguimos los pasos del modal:

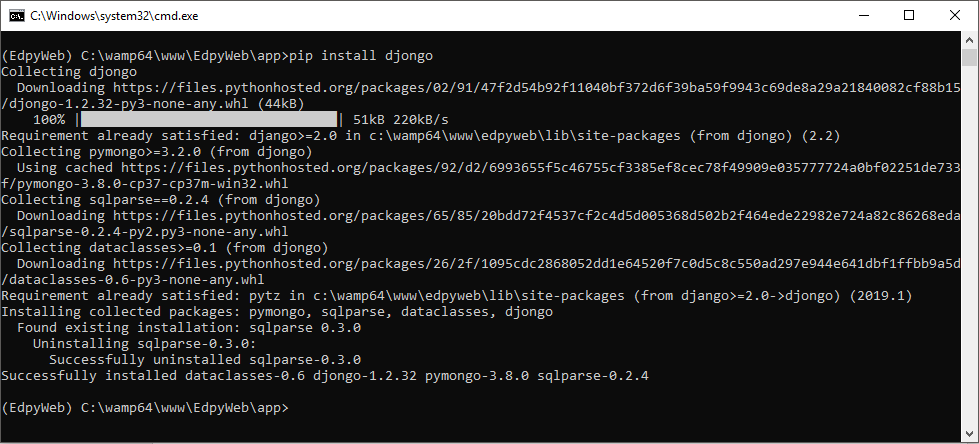






## INSTALACION DEL FRAMEWORK ‘DJONGO’

Para instalar este framework, nos dirigimos al entorno virtual y escribimos en el prompt: **pip install djongo** con esto, instalara la version mas reciente de las librerias pymongo, sqlparse y el propio djongo.



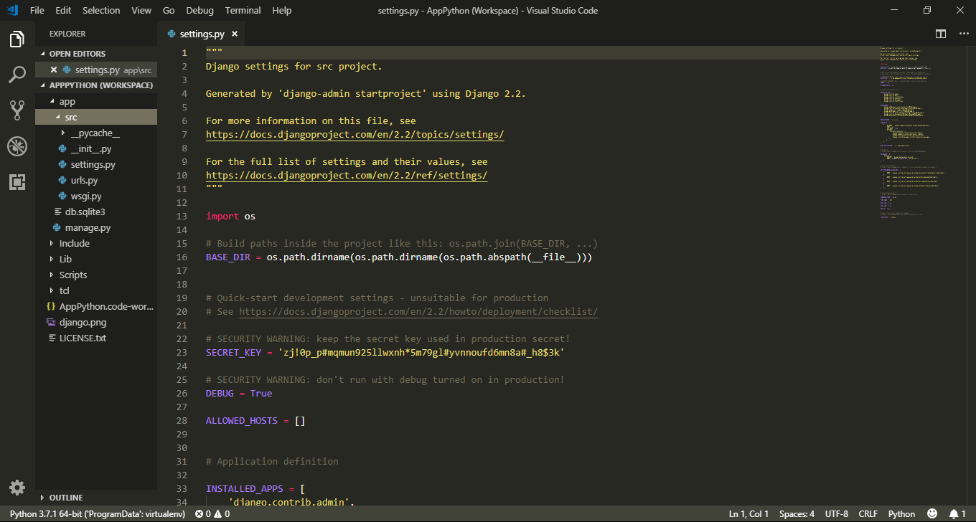
# CREACION DE SUPERUSER E INTERFAZ ADMINISTRATIVA

## CONTEXTO:

Un super usuario tiene la capacidad de controlar todo el proyecto, es por esto por lo que debemos plantear las credenciales para esta labor, teniendo en cuenta que en esta ocasión Python se convertirá en un servicio.

## EL ARCHIVO SETTINGS.PY

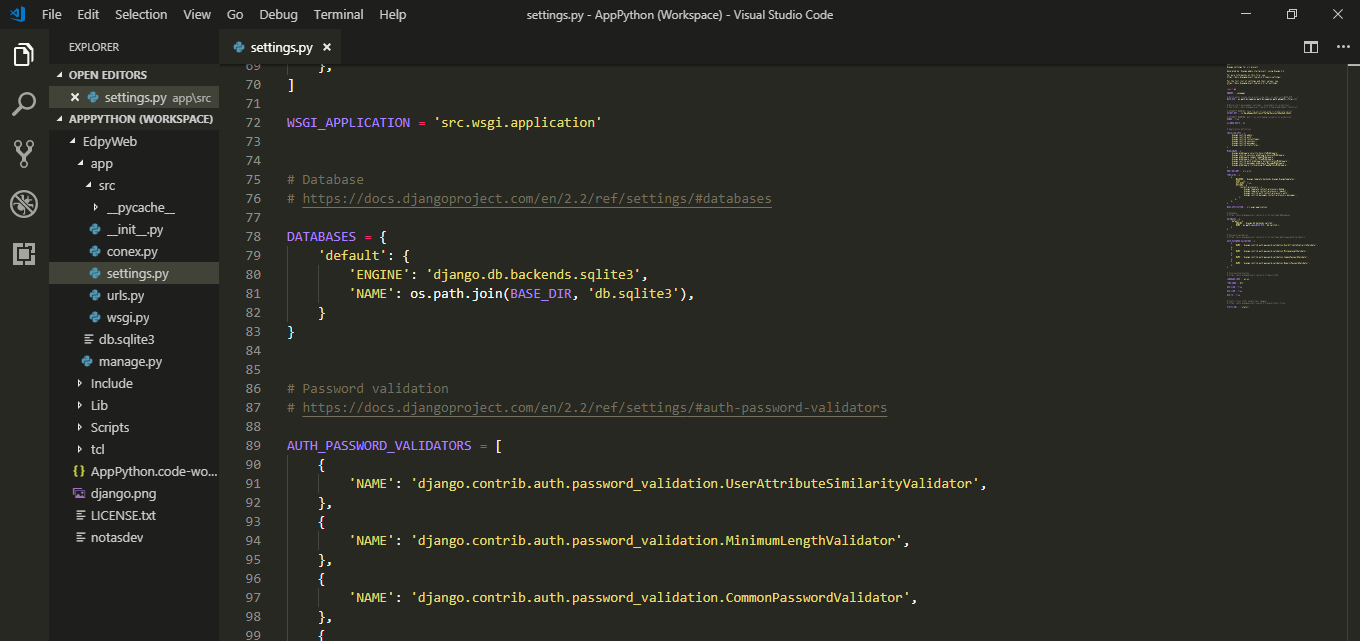
Dentro de la carpeta src, se encuentra el archivo settings.py, este archivo constituye la parametrización del entorno de django, es en este archivo donde configuraremos el acceso a la base de datos, el super usuario, las librerías necesarias, etc.



## AGREGANDO LA CONEXIÓN A LA BASE DE DATOS:

Django, provee un motor de base de datos llamado sqlite, y lo podemos encontrar anexo al proyecto. En el archivo settings.py se encuentra configurado este motor de base de datos por defecto, pero es necesario modificarlo para que podamos implementar la conexión que hemos creado anteriormente.

Por lo que primero buscaremos la sección DATABASES en el archivo settings.py:



Encontramos la siguiente línea de código:

# Database

# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

Las cambiaremos por las siguiente:

# Database

# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'djongo',

'NAME' : 'senaapps',

'HOST': 'mongodb+srv://appsena:2019sena@senaapps-e1gk3.mongodb.net/senaapps?retryWrites=true',

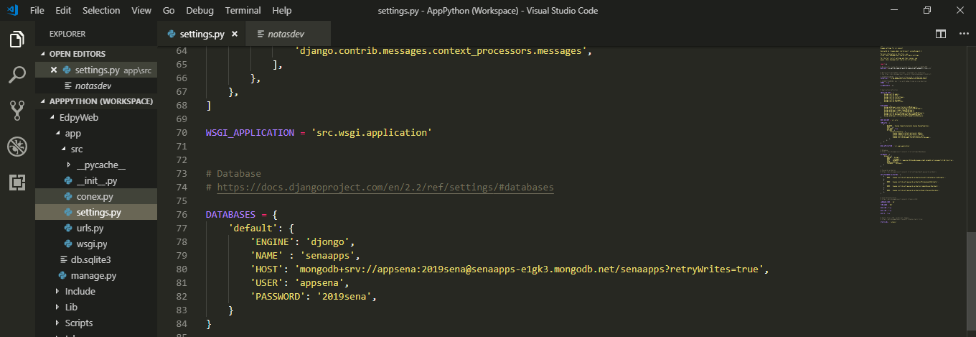
'USER': 'appsena',

'PASSWORD': '2019sena',

}

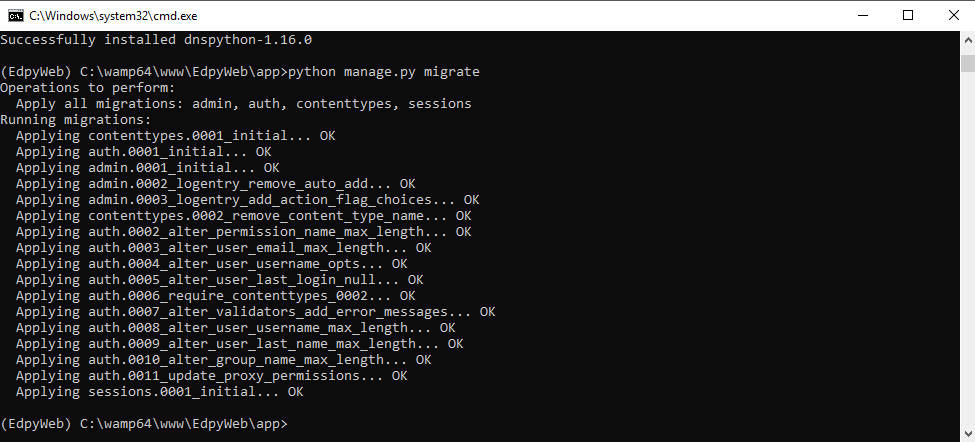
}

Y nos ha quedado de la siguiente manera:



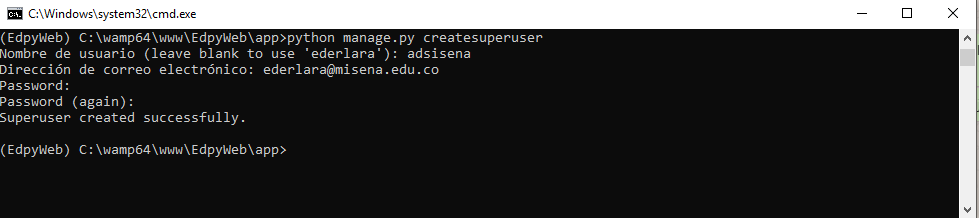
## REALIZAR LA MIGRACION DESDE LA CONSOLA:

En este paso es muy importante, ya que es necesario para la creación del super usuario, pero no es para nada complicado. Tan solo hay que instanciar la migración desde la base del proyecto y esperar que suceda, para eso usamos el comando **python manage.py migrate**



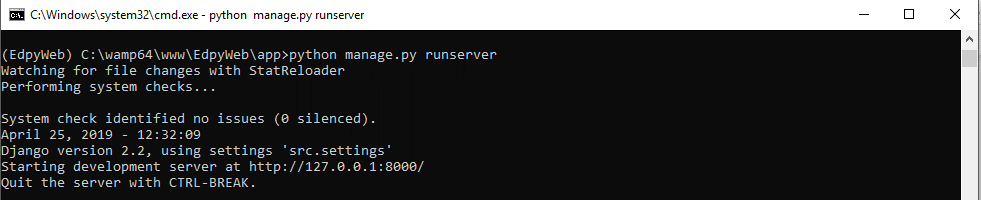
## CREAR EL SUPER USUARIO DESDE LA CONSOLA:

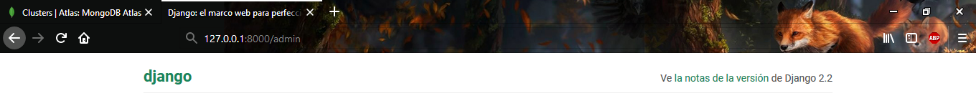
El super usuario será el encargado de administrar desde el panel que trae Django, por lo que también debe contener una contraseña. Desde la consola escribiremos el comando:   
**python manage.py createsuperuser *<<username>>*,**  nos solicitará el nombre del super usuario (si se deja en blanco nos pondrá uno por defecto en este caso lo llamare *adsisena*), el correo electrónico y el password (usaré el mismo de la base de datos -2019sena-), y de nuevo la confirmación del password:



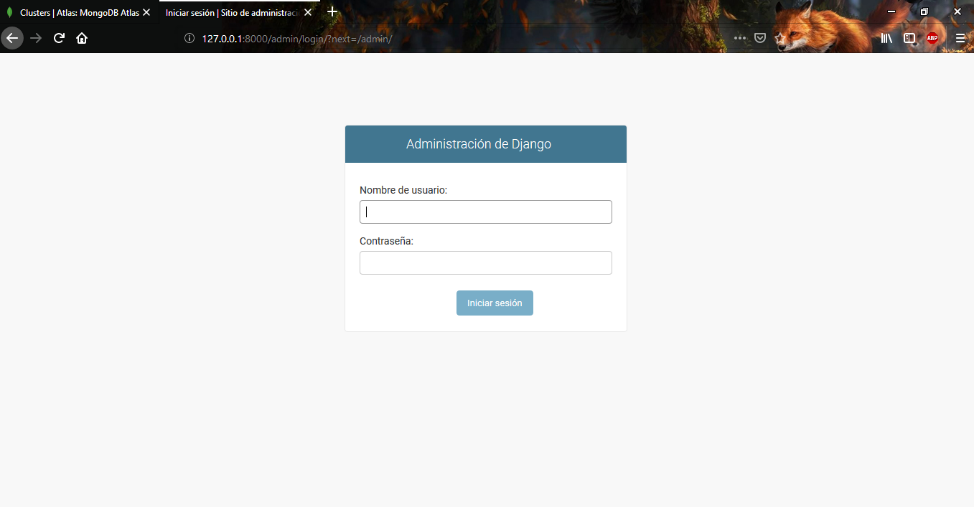
## PROBANDO LAS CREDENCIALES EN EL NAVEGADOR

Para comprobar que efectivamente ha sido creado el super usuario, debemos encender el servicio y dirigirnos al navegador ingresando a la dirección **127.0.0.1:8000/admin**:

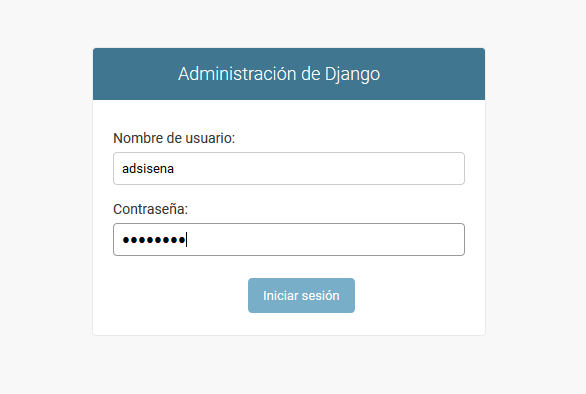




Nos aparece un formulario de inicio de sesión:



Digitamos nuestras credenciales y damos enter:



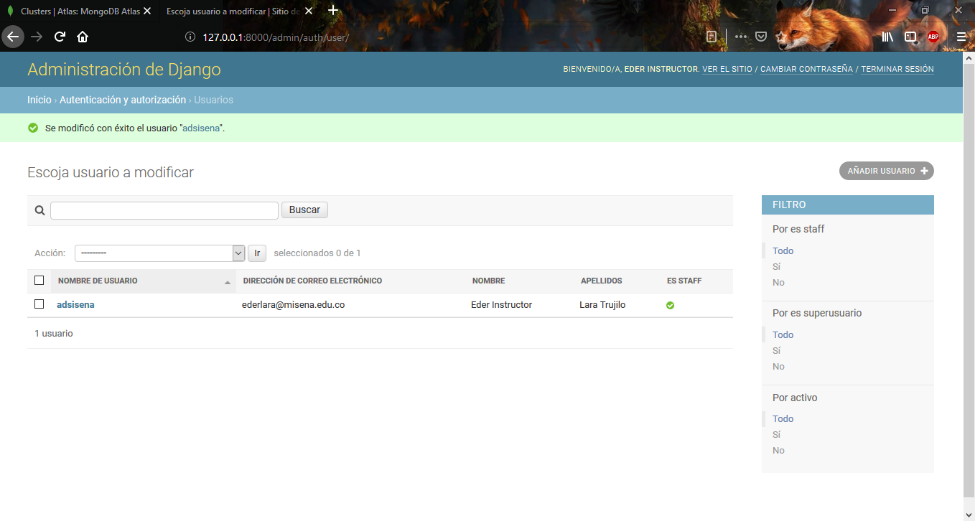
De inmediato, si las credenciales son las correctas nos redirige al módulo de administración de Django:



Ahora podemos dar clic en **usuarios** y modificar los datos que consideremos necesarios, en este caso yo adicionare los datos de nombre y apellidos. También puedo crear grupos de usuarios, agregar y denegar permisos, etc.:



Al final damos guardar y nos queda así:



Con esto queda nos ha quedado listo para que podamos agregar usuarios, tenemos una interfaz grafica con módulo de administración. Lo que sigue a continuación es la creación de los modelos para nuestro proyecto.

# LAS REGLAS DEL NEGOCIO PARA EL MODELO DE DATOS.

Para este ejercicio, haremos un Proyecto web estilo hoja de vida, tomando como ejemplo la version que existe en la [plataforma del SENA](https://agenciapublicadeempleo.sena.edu.co/Paginas/inicio.aspx), pero lo haremos enfocado al sector informatica e ingenieria. Es decir, el usuario se registrara en la aplicación y podrá interactuar con los modulos propios de la hoja de vida:

* Datos personales
* Estudios realizados
* Habilidades y Destresas
* Experiencia Laboral
* Muro para Postear logros y desarrollos.

Para entender que es lo que se requiere, dejo a continuación un ejemplo de modelo de clases que podria funcionar:[[3]](#footnote-3)

# MODELO DE BASE DE DATOS CURRICULUM VITAE -GENERAL-

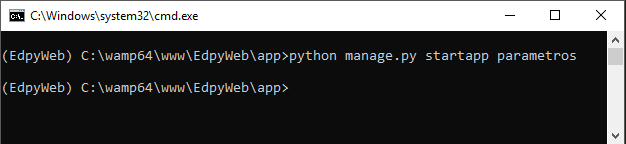
# INICIANDO UN MODULO:

Django integra cada “modulo” o “seccion” de nuestra aplicacion como si esta fuera una “**app”**, pero es solo semántica, ya que vamos a modelar nuestra aplicacion web tal como lo hacemos con otro framework (Laravel, MEAN, etc). solo que en este caso cada modulo tendra su propia carpeta con el modelo, la vista y la configuracion. Una de las ventajas de *Django* es que estos modulos pueden ser reutilizados en otros proyectos

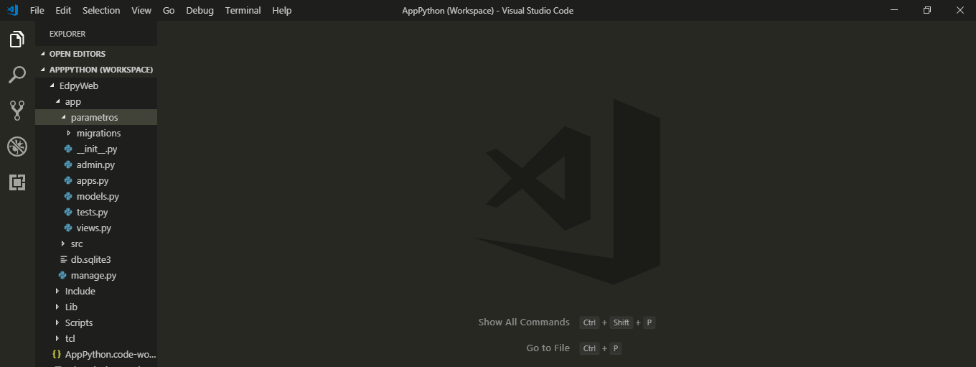
## CREACION DE UN MODULO A TRAVES DE LA CONSOLA

Nuestro primer modulo será el de parámetros del Sistema. Por qué parámetros del Sistema?, pues porque son las colecciones (*coleccion = table*), que heredan sus indices a otras coleciones. Y son desde estas colecciones que surge la base para la creación de todas las reglas del negocio. Estas colecciones serán: ***Etnias, TipoDocu, TipoCarg, TipoEdus, TipoLogr*** y una coleccion de documentos embebidos que se llamará localizacion y comprente las colecciones de ***pais, departamento y ciudad***.

Para empezar a construir nuestro primer modulo, escribiremos en la consola el comando:   
**python manage.py startapp <<nombreapp>>** , como en este caso nuestra app se llama parametros, pues la instruccion sera: **python manage.py startapp parametros:**



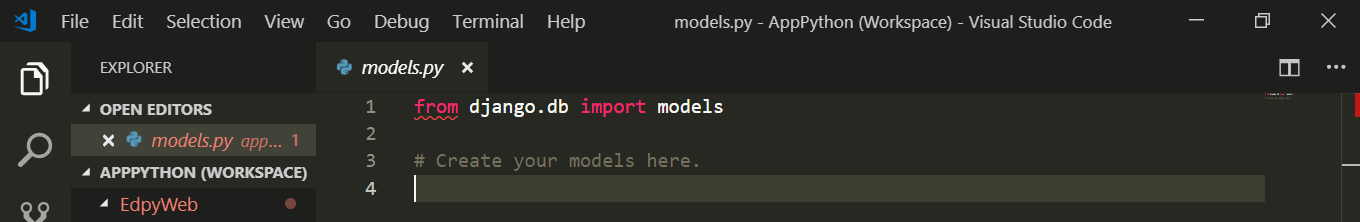
Y en el editor de codigo veremos que se nos ha creado una carpeta de nombre ***parametros*** que contiene la carpeta ***migrations*** y los archivos: *\_\_init.py\_\_, admin.py, apps.py, models.py, test.py* y *views.py*:



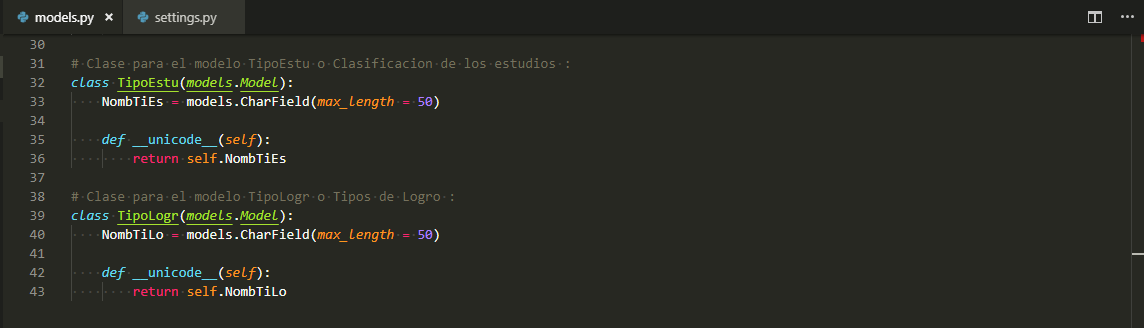
Cada uno de estos archivos tiene un fin que iremos comentando a medida que los vayamos utilizando. Ahora el primer archivo a modificar sera **models.py**, ya que la funcion de este es la de crear la estructura de los datos que vamos a persistir.

## MODELS.PY: CREACION DEL PRIMER MODELO

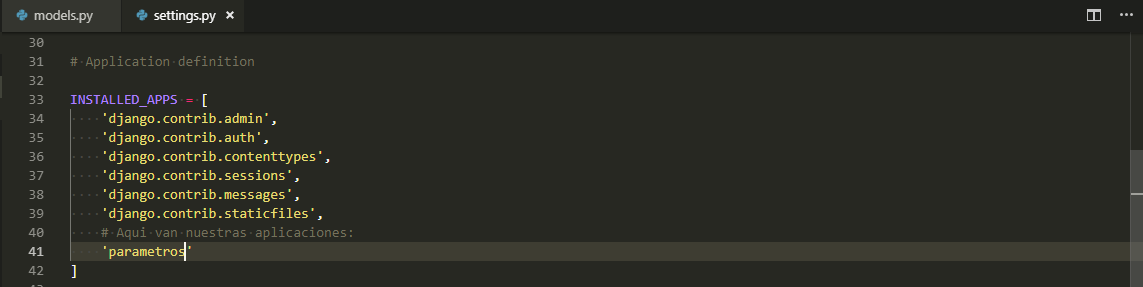
El archivo models.py, es donde definiremos la estructura de datos de las colecciones. Así que abrimos el archivo y veremos que se ha importado la librería models del paquete django.db:



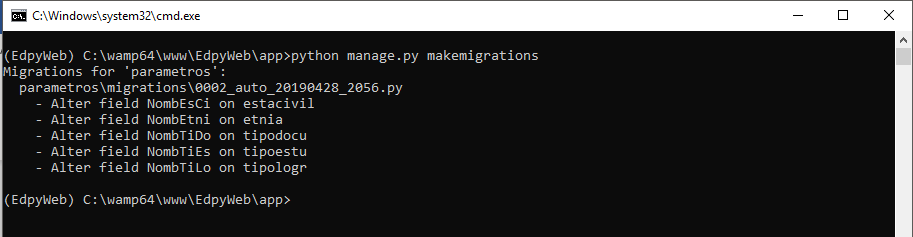
Creamos las clases con los modelos:



Luego de crear nuestros modelos de datos, debemos matricular en el archivo src/settings.py nuestra app:

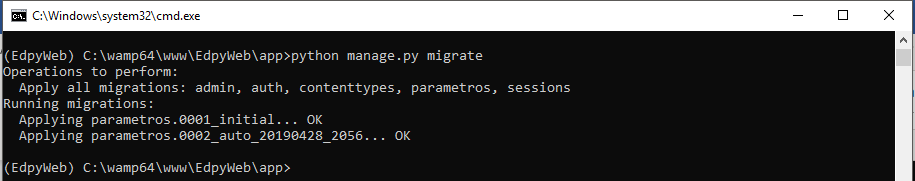


Ahora que hemos creado la estructura de las tablas de parámetros, debemos migrar a la base de datos, para ello usamos el comando : Python manage.py makemigrations:

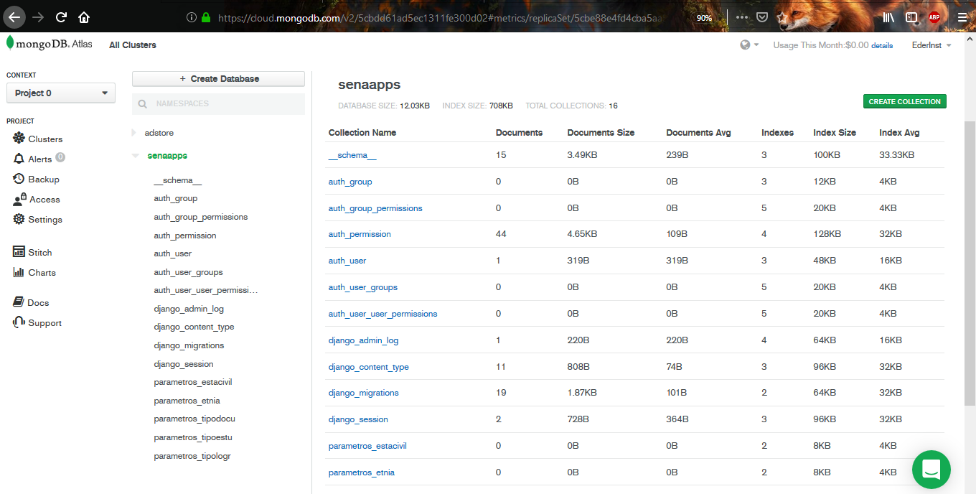


En este caso me aparece el mensaje de Alter field …, pero es porque he corrido la migración antes de hacer unos cambios, luego de actualizar los campos, la he corrido de nuevo obteniendo ese resultado.

Ya con esto hemos creado los parámetros de la migración, ahora corremos el comando:   
**python manage.py migrate:**

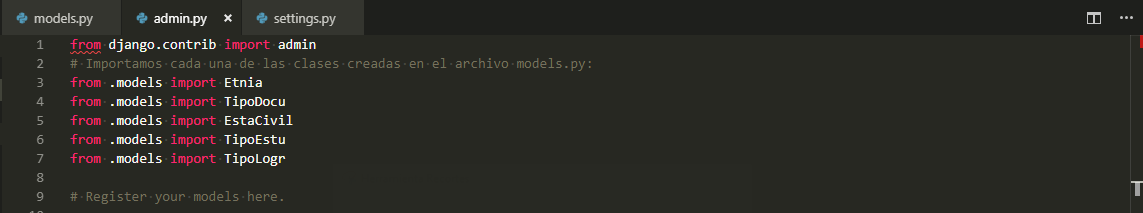


Si revisamos nuestra base de datos, observaremos que ya se han migrado las colecciones de la migración, con el super usuario. Y las nuevas colecciones del módulo ***parámetros***:

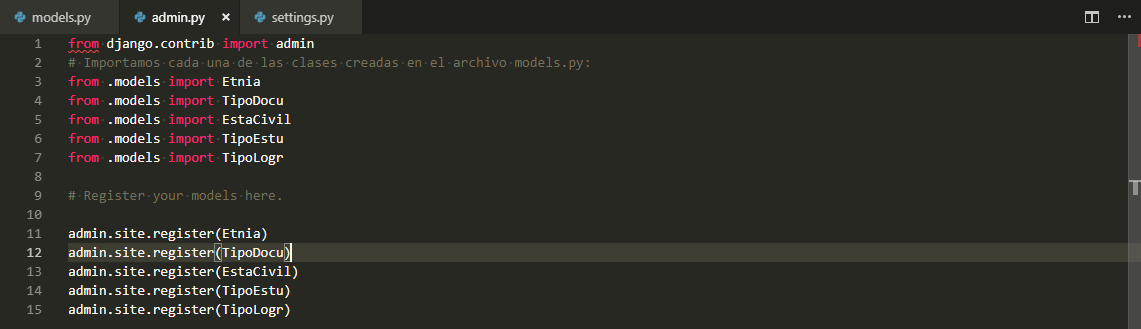


## REGISTRAR EL MODELO EN EL ARCHIVO admin.py

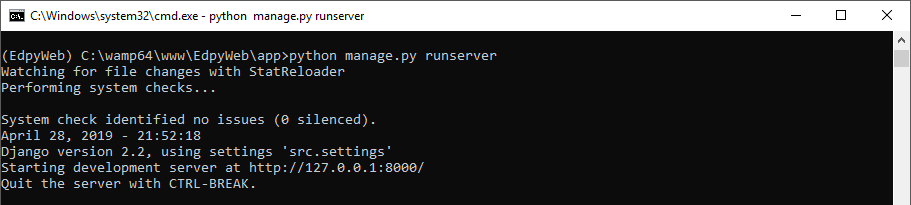
Para registrar los modelos creados anteriormente, debemos importar cada una de las clases creadas para cada modelo:



Y luego de importar las clases, las matriculamos con el comando ***admin.site.register(<<clase>>):***



Ejecutamos el servidor y revisamos en la página web:

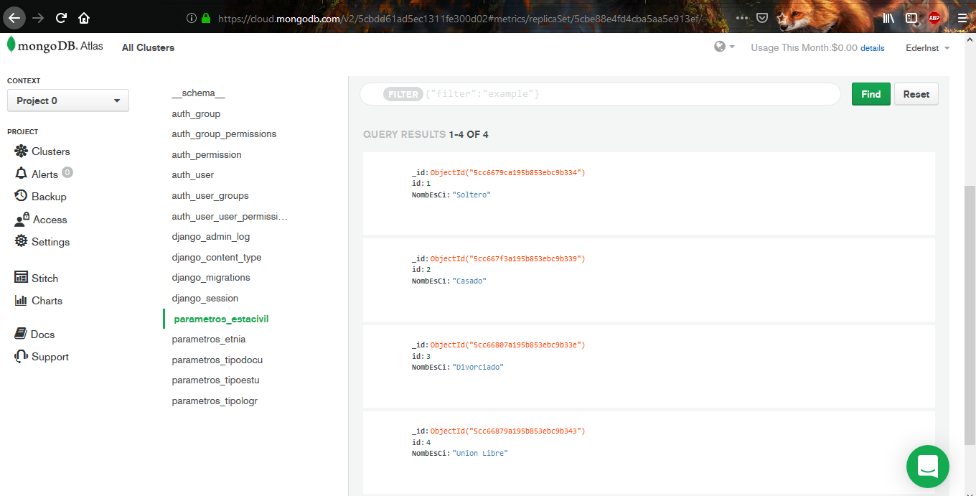




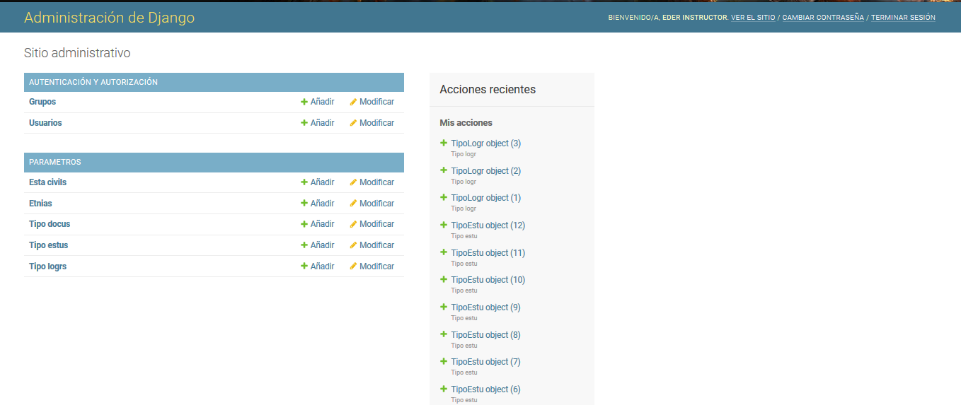
Ahora ya podemos agregar los parámetros necesarios, en este caso para el ejercicio agregare los estados civiles:



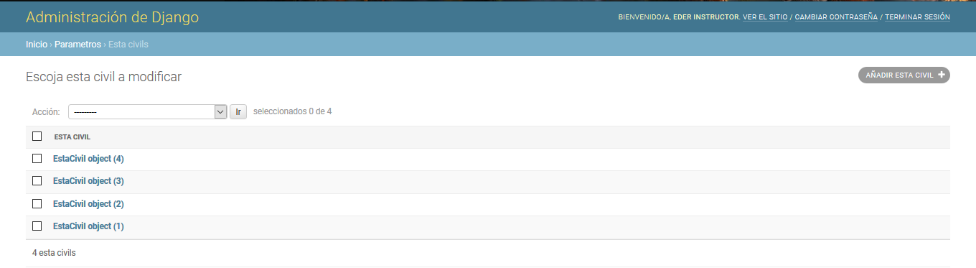
El framework, ya nos ha agregado el campo y los botones para agregar. Mientras voy agregando en la base de datos se han ido insertando los documentos:



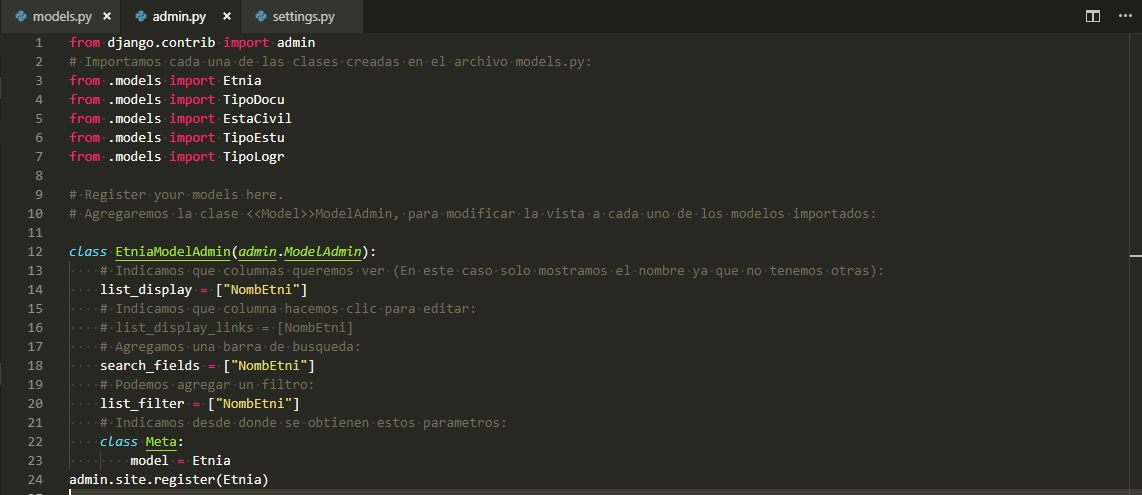
Haremos lo mismo para las otras colecciones. Luego de hacer las inserciones en la base de datos desde la vista que ofrece Django, veremos que aparece el log de las actividades recientes:



Y si miramos una cualquiera de los modelos creados, veremos que podemos acceder a los datos, pero solo al nombre del objeto que hemos creado:



Para mejorar un poco la vista de nuestro panel de administración, vamos a hacer unos cambios a continuación en el archivo admin.py para que podamos tener una vista mucho mas amena a un entorno de administración. Agregaremos la clase ***<<Modelo>>ModelAdmin*** a cada uno de los modelos:



## La APP CORE:

La app core, será nuestra base para poder cargar todos los estilos y vistas de nuestro fron-end, por ahora solo incluye la información básica de la página, el archivo index.html y nosotros.html.

La app core tendrá la siguiente estructura:

Carpeta static, para los archivos estáticos, como imágenes, etc.

Carpeta templates, para los archivos html del proyecto.

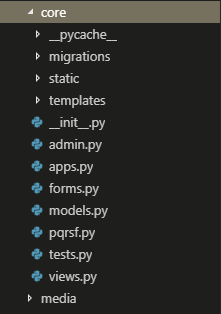
Archivo admin.py, para matricular los modelos de la aplicación.

Archivo forms.py, es muy similar a los modelos, solo que desde este archivo accederemos a formularios en la página principal.

Archivo models.py, para incluir los datos que persisten en esta app.

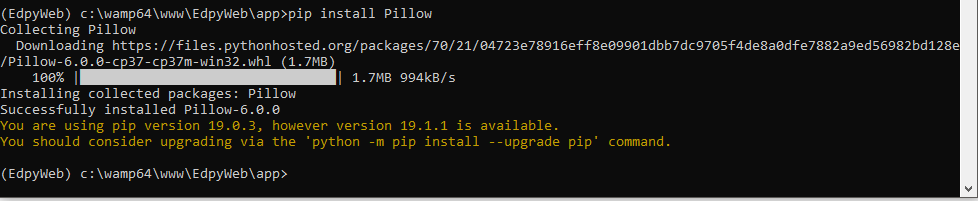
Archivo pqrsf.py, es un archivo tipo tupla, que nos permitirá realizar un selector dese el formulario.

Archivo views.py, para retornar o renderizar las vistas html.

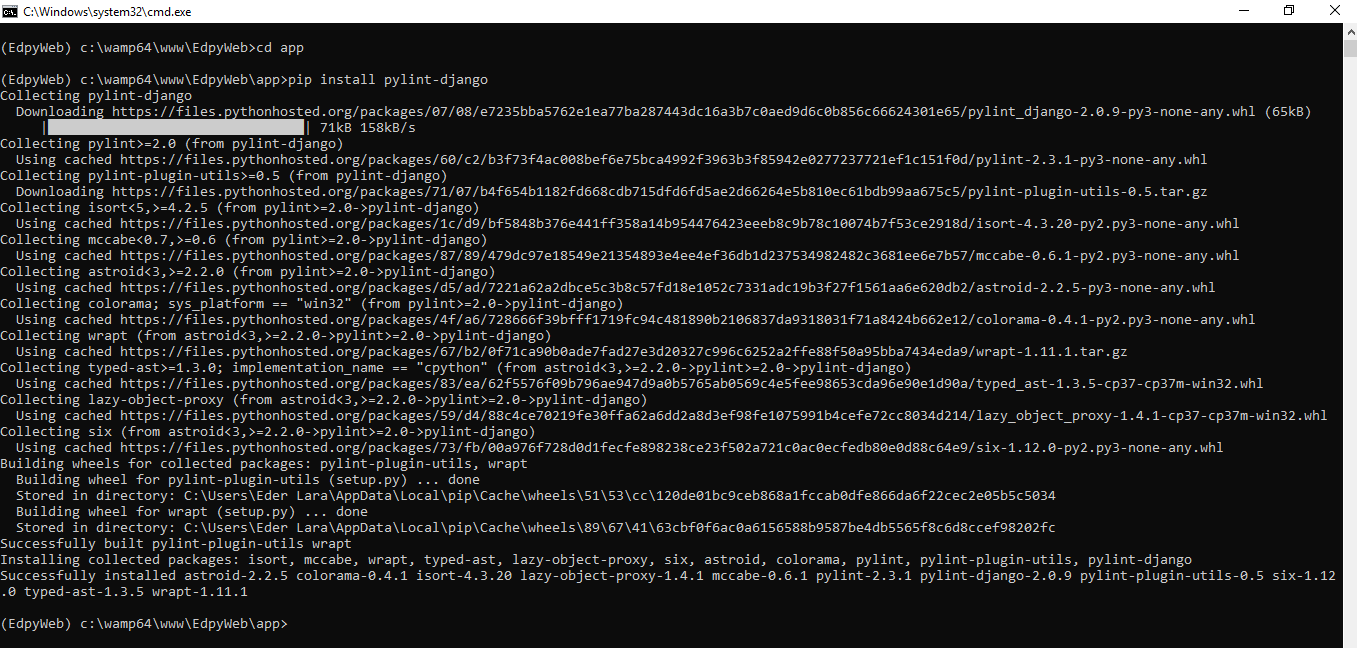


## INCLUYENDO LIBRERIAS NECESARIAS PARA CONTINUAR CON EL PROYECTO:

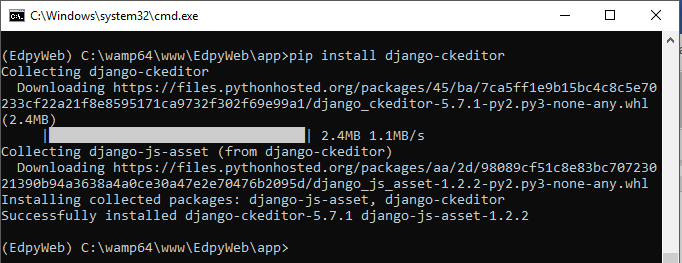
Antes de continuar, apagaremos el servidor e instalaremos una librería que nos otorgará dinamismo para el manejo de archivos, como las imágenes. Esto para que podamos contar con todo lo necesario para el manejo de nuestra aplicación. Una de estas librerías es ***Pillow***, para instalarla hacemos tal y como con la librería anterior (***mongopy***):



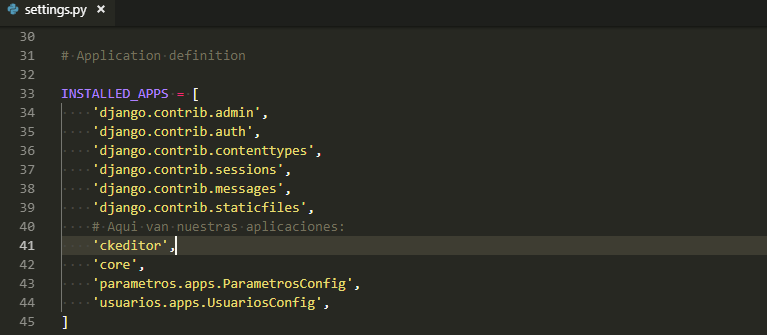
También deberemos instalar la librería ***pylint-django*** para la gestión de errores desde el editor ***visual code studio:***



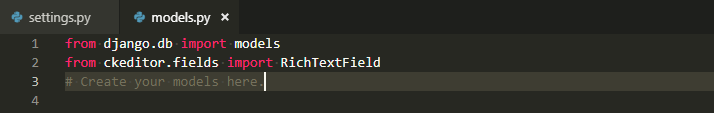
Otra de las dependencias que usaremos será ***django-ckeditor***, esta librería me permitirá anexar texto enriquecido HTML, este es muy importante en el ámbito de desarrollo web, ya que nos permite utilizar formato de texto en elementos textarea, e implementarlo directamente en la web tal como si fuese Office o Libre Office. Así que de una vez lo instalaremos:



Después de instalar la dependencia, debemos matricularla en el archivo settings.py de nuestra aplicación, como si fuese una app adicional. Nos debe quedar de la siguiente forma:

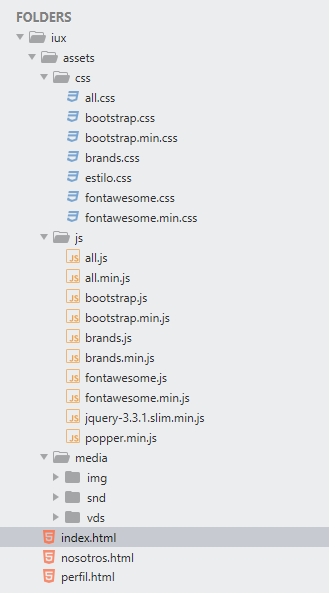


De esta forma ya podemos implementar un nuevo tipo de campo en el archivo models.py de cualquier app. Tal y como lo muestra la imagen a continuación:



Ejemplos de implementación de django-ckeditor: <https://github.com/django-ckeditor/django-ckeditor>

# CREANDO LA VISTA DE NUESTRA APLICACIÓN.

Para crear las vistas, lo primero que haremos será la interfaz de usuario en html, con eso tendremos mucho trabajo adelantado a la hora de montar las vistas. Para este ejercicio utilizare un framework html y css con gran aceptación dentro de la comunidad de diseñadores web y desarrolladores de aplicaciones; Bootstrap. Para ello debemos descargar Bootstrap, jquery, Font-Awesome, y algunas imágenes para nuestra aplicación (para ello recomiendo usar imágenes sin copyright).

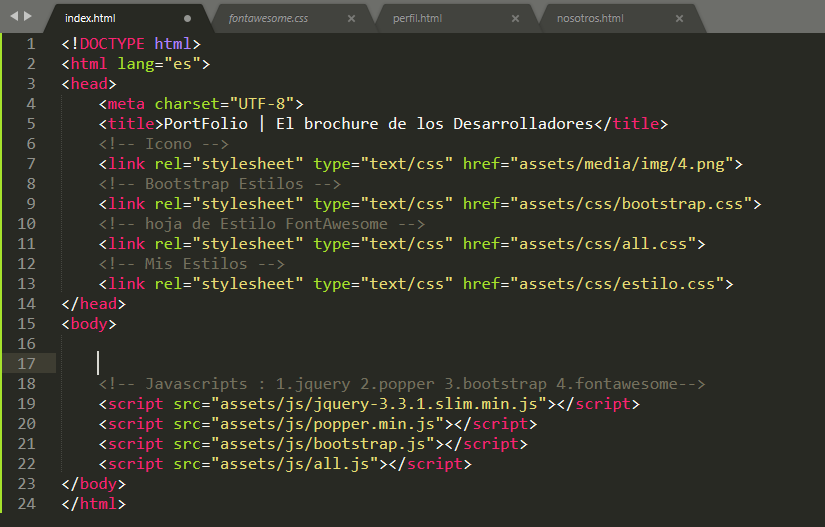
La estructura de la interfaz html es muy sencilla:

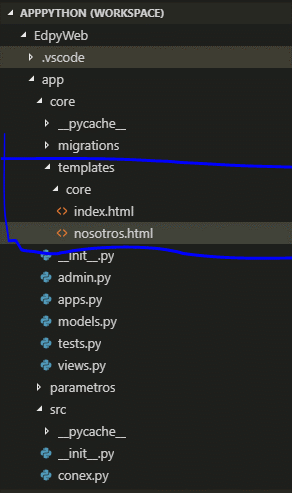
1. Interfaz uix:
   1. assets
      1. css
      2. js
      3. media
   2. index.html
   3. nosotros.html
   4. perfil.html

como se muestra en la imagen:

Ahora solo debemos empezar a maquetar nuestro proyecto curriculum con lo que deseamos que tenga la página.

Esta maquetación está sujeta a cambios ya que tampoco es una regla que tenga la estructura sugerida.

En los 3 archivos vamos a configurar la misma estructura html:

Ya tenemos una estructura básica de html, con esto podemos empezar a configurar nuestras vistas para complementar nuestra presentación.

Para el caso, hemos creado los módulos: parámetros, ***core*** y ***portafolio***. Dentro de cada app vamos a crear una carpeta llamada template, y dentro una carpeta con el nombre del módulo, y dentro los archivos html que sean necesarios:

En este caso muestro 2 archivos, pero aprenderemos a utilizar herencia con Python, para que construir la aplicación sea mucho más fácil y eficiente a la hora de hacer despliegues.

Desde esta configuración desplegaremos nuestra aplicación, así que manos a la obra:

## CARPETAS Y ARCHIVOS DE LA CARPETA CORE:



Migrations: esta carpeta tiene los archivos temporales de la migracion, al ejecutar los comandos python manage.py makemigratios y migrate, se guardará un registro de la actividad en esta carpeta.

Static: esta carpeta tendrá los archivos estaticos de la aplicacion, para este caso serán: archivos css, archivos javascript, archivos multimedia (imagenes, sonidos, videos).

Templates: esta carpeta contiene los archivos html de la aplicacion, contará con los siguientes archivos:

Base.html: este archivo contiene la estructura básica del html: html, head, links, scripts, y demás componentes necesarios.

Index.html: este archive tendra todas las vistas que veremos al iniciar en nuestra pagina web.

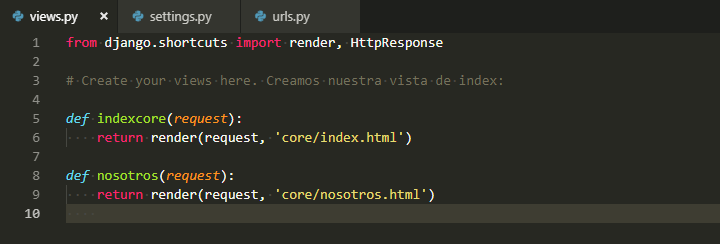
Nosotros.html: contiene los datos de las personas que colaboran en el Proyecto.

En todo caso estos archivos tienen estructura basica de html y podrá recibir parametros desde las funciones definidas por el framework.

Python usa la notacion por parametros ***{ parametro }*** en html, al igual que lo hacen otros frameworks como php. Y es con esta notacion que vamos a definer los contenidos de las páginas, pero primero configuraremos los arhivos **view.py** de la carpeta **core** y el archivo **url.py** de la carpeta **src**

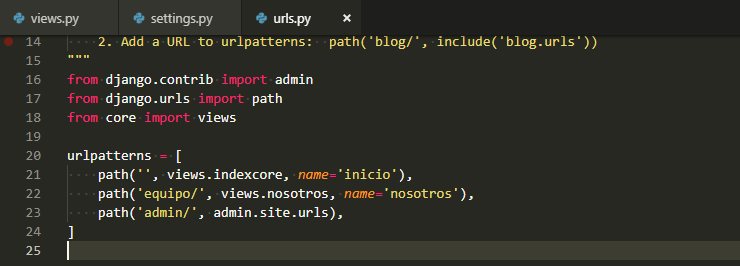
## El archivo view.py

Este archivo contiene la ruta donde debemos acceder para obtener las plantillas. Estas plantillas siempre deberán estar contenidas en la carpeta **templates**, ya que de lo contrario deberíamos indicar. Es por eso por lo que solo indicaremos lo que se supone debe existir dentro de la carpeta **templates**:



## El archivo urls.py

Para poder trabajar entre rutas y archivos, debemos configurar este archivo. Para ello debemos tener configuradas previamente las vistas de cada app en el archivo **views.py** y los mismos archivos en la carpeta **templates** de las mismas.

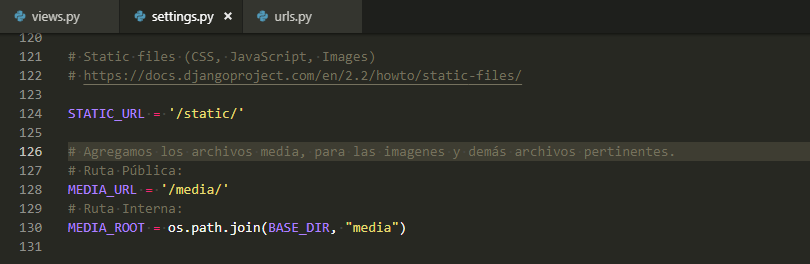


En el archivo urls.py la estructura es: **path(‘ruta que queremos ver en el navegador’, urlorigen, nombre).**  En esta ocasión lo dejaremos así. Ya veremos mas adelante que al añadir, nuevas apps, este va cambiando.

## Archivos /static/ y /media/

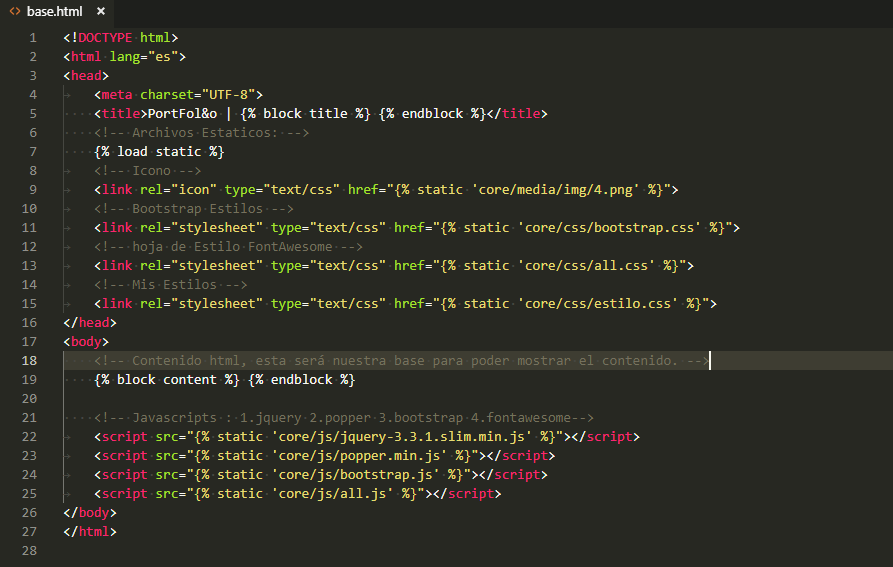
Python ya tiene configurado los archivos /static/, estos son los archivos que se utilizan como recursos para el funcionamiento de nuestras páginas web; tales como: estilos en archivos \*.css, funciones javascript, archivos multimedia entre otros. Como Python ya tiene configurado la carpeta /static/ solo haremos una modificación para que podamos tener acceso archivos estáticos de tipo multimedia.

Modificaremos en el archivo settings.py la última línea, que es donde se encuentra esta configuración; agregándole 2 variables globales para que se utilicen desde allí sin importar la profundidad del directorio donde se encuentren:



Ya estamos listos para trabajar con rutas desde nuestros templates html.

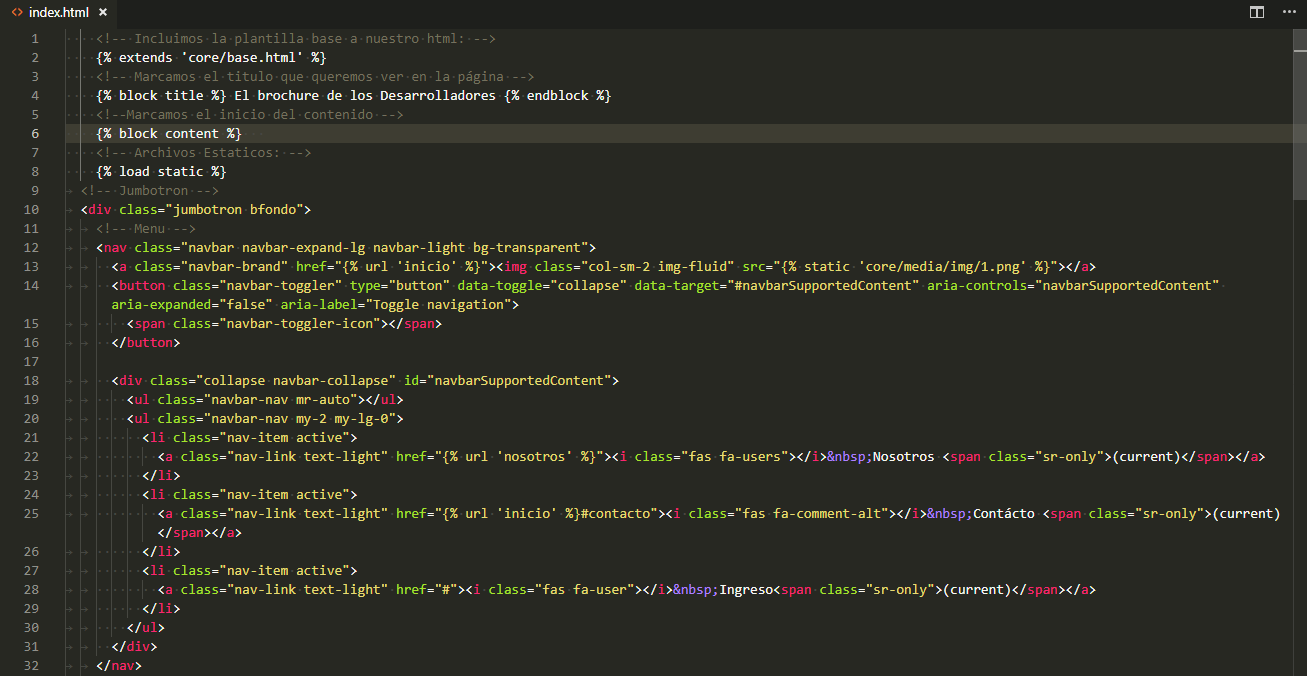
Y para empezar, tomaremos un archivo base desde el cual se desplieguen todas las demás páginas. Este archivo lo he llamado base.html:

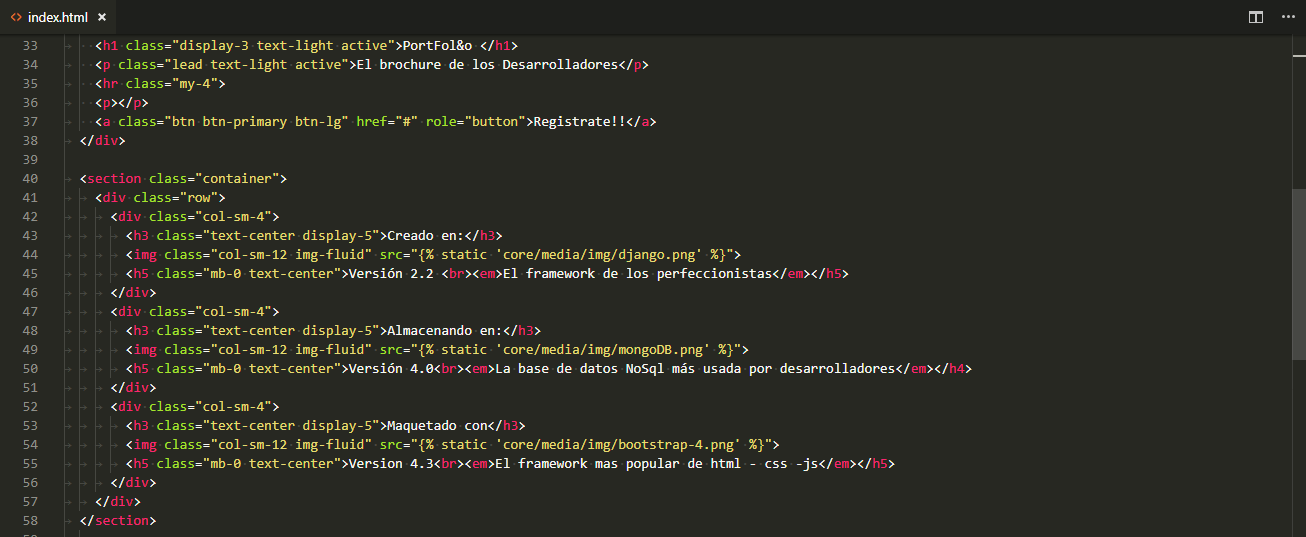


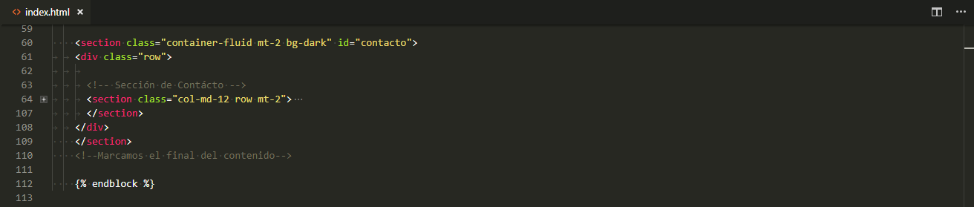
Como podemos ver, Python hace un paso de parámetros a través de {% blocks %}, es decir, utiliza bloques para indicar donde inicia y donde termina algún contenido HTML. Es así como podemos utilizar esta base para que se carguen de forma optima los recursos y demás contenido HTML. Puedes ver mas acerca de esto en la documentación oficial de django 2.2:

<https://docs.djangoproject.com/es/2.2/>

Ahora veremos como se ha de configurar los archivos índex y nosotros.html. Este par de archivos son los que contienen toda la información al respecto de la página.







Como podemos ver el archivo index.html inicia con

<!-- Incluimos la plantilla base a nuestro html: -->

{% extends 'core/base.html' %}

<!-- Marcamos el titulo que queremos ver en la página -->

{% block title %} El brochure de los Desarrolladores {% endblock %}

<!--Marcamos el inicio del contenido -->

{% block content %}

<!-- Archivos Estaticos: -->

{% load static %}

y finaliza con:

<!--Marcamos el final del contenido-->

{% endblock %}

De esta manera podemos crear bloques de contenido y pasarlos al archivo base.html para que los pinte, aunque lo que hacemos es incluir en nuestros archivos de maquetas, el archivo base.html.

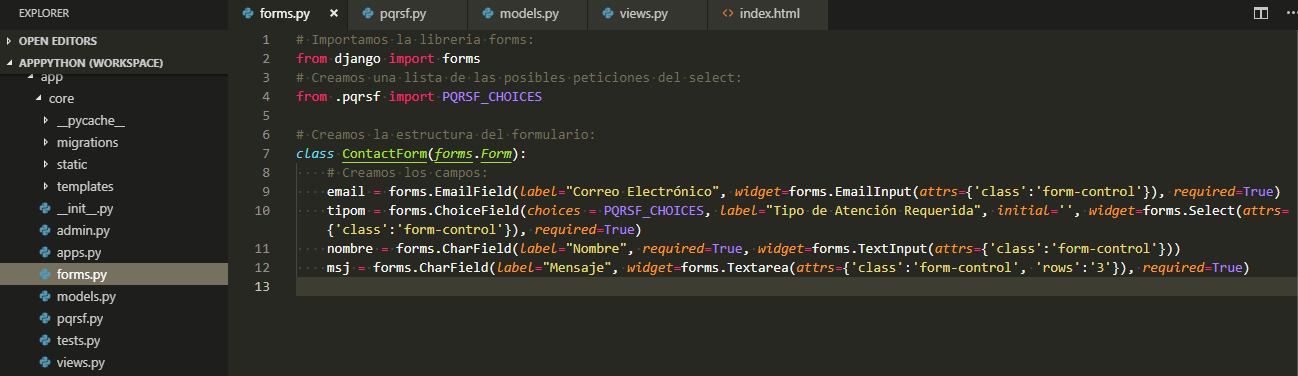
En resumen, creamos bloques con el nombre que queramos y debemos terminar el bloque, de la siguiente manera: **{% block <nombreblock> %} {% endblock %}**

# IMPLEMENTACION DE FORMULARIOS.

Django nos ofrece la posibilidad de crear un diseño de formulario muy similar a la forma de hacer modelos, así que nos permitirá crear HTML automáticamente. Adicionalmente los métodos para procesarlo y validarlo.

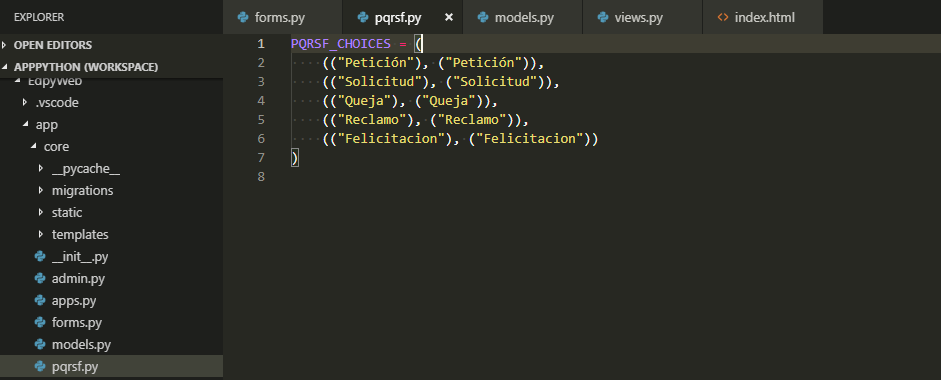
## Creación del archivo forms.py:

El primer paso es crear un archivo con el nombre ***forms.py***, este archivo tendrá una clase para indicar cuales serán los campos de entrada.[[4]](#footnote-4)



## Creación del archivo que tiene la tupla de las opciones, pqrsf.py:

Este archivo, en la raíz de la app core; contiene un pequeño listado en una lista llamada PQRSF\_CHOICES, esta lista tiene el contenido del elemento <select> del html:



En este archivo podríamos crear todas las listas que consideremos necesarias si no quisiéramos colocarlas en un modelo de datos. Puede ser útil en muchos casos, pero considerando las buenas practicas y las reglas de normalización, se debería ir a una colección. Sin embargo, por esta ocasión esta bien que la pongamos en una lista.

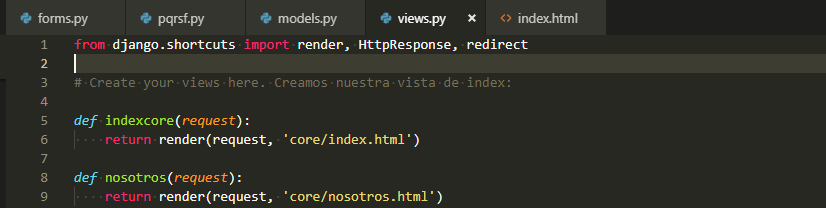
## Creación del modelo

En el archivo models.py, creamos la estructura de datos necesaria para almacenar la información del formulario:

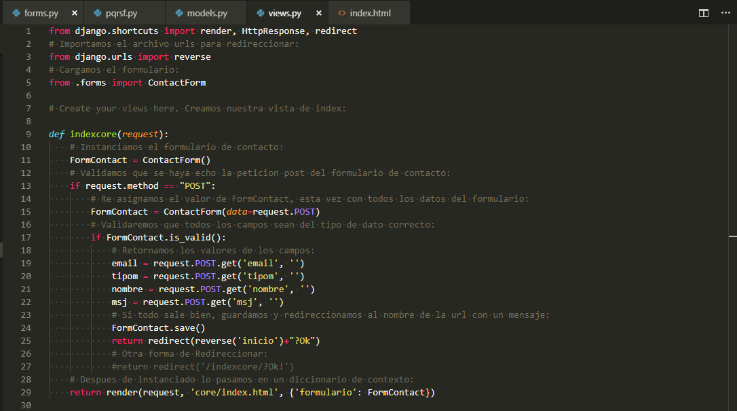


## Creación del archivo view.py:

El archivo view.py antes tenia una estructura simple y sin mas objetivo que renderizar HTML:



Pero ahora, tendrá un objeto adicional, que es el de capturar y gestionar la información pasada a través del formulario, en este caso el formulario de contacto:



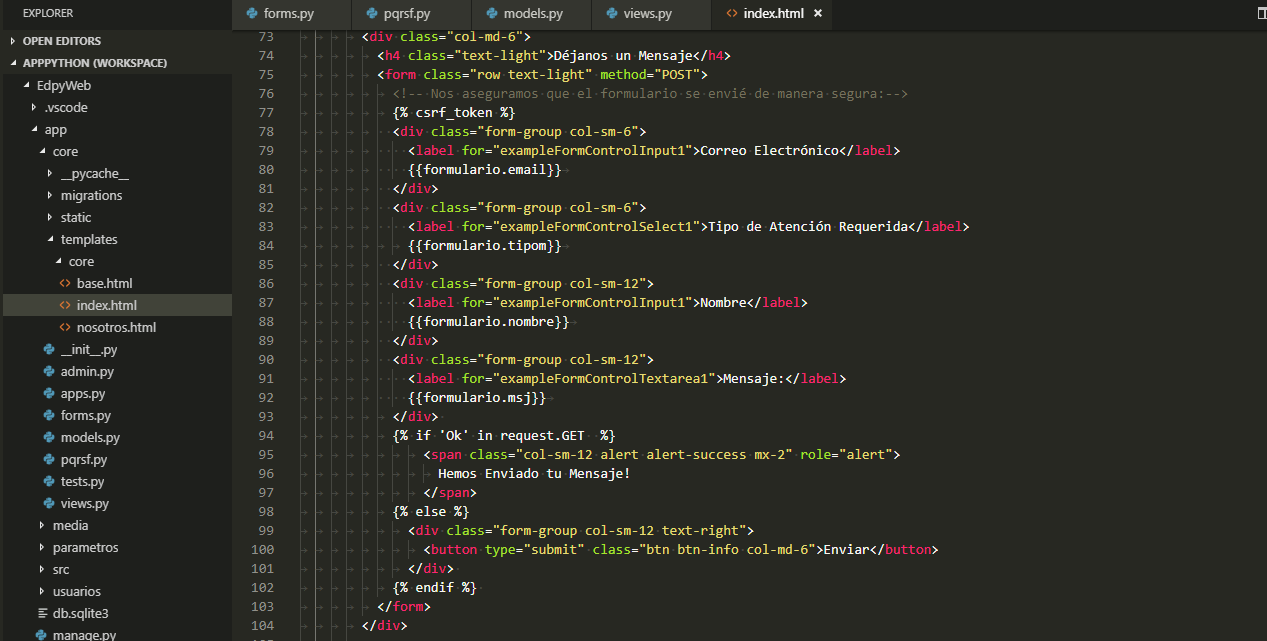
## Fusión del formulario con el HTML:

Desde el archivo forms.py, ya podemos embeber o anexar formato HTML, como por ejemplo etiquetas label e input a nuestros archivos del front-end, pero esto haría que quedara simple y sin estilos. Sin embargo, si observamos en el archivo forms.py, para cada campo se han agregado atributos propios de las etiquetas html, observemos un momento la siguiente línea:

tipom = forms.ChoiceField(*choices* = PQRSF\_CHOICES, *label*="Tipo de Atención Requerida", *initial*='', *widget*=forms.Select(*attrs*={'class':'form-control'}), *required*=True)

como podemos ver, le pasamos las opciones a mostrar <choices>, también una etiqueta <label>, y un widget, este ultimo se encarga de indicar que tipo de input es, en este caso un <select> y que atributos debe tener esta etiqueta para que pueda usarse de esta forma: <select class=”form-control” > </select>.

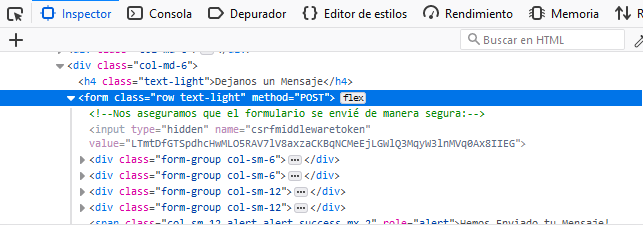
Dentro de nuestro archivo index.html en la sección de contacto tendremos lo siguiente:



En la línea 77 encontramos un content tag escrito de esta forma:

{% csfr\_token %}

Este content tag, se debe anexar a todos los formularios, y es utilizado para evitar las inyecciones de información de lugares externos a nuestra aplicación. El objetivo de este elemento es crear un token o código de verificación que se envía junto con los datos del formulario. Al observar el código fuente de la página, encontraremos el siguiente objeto:

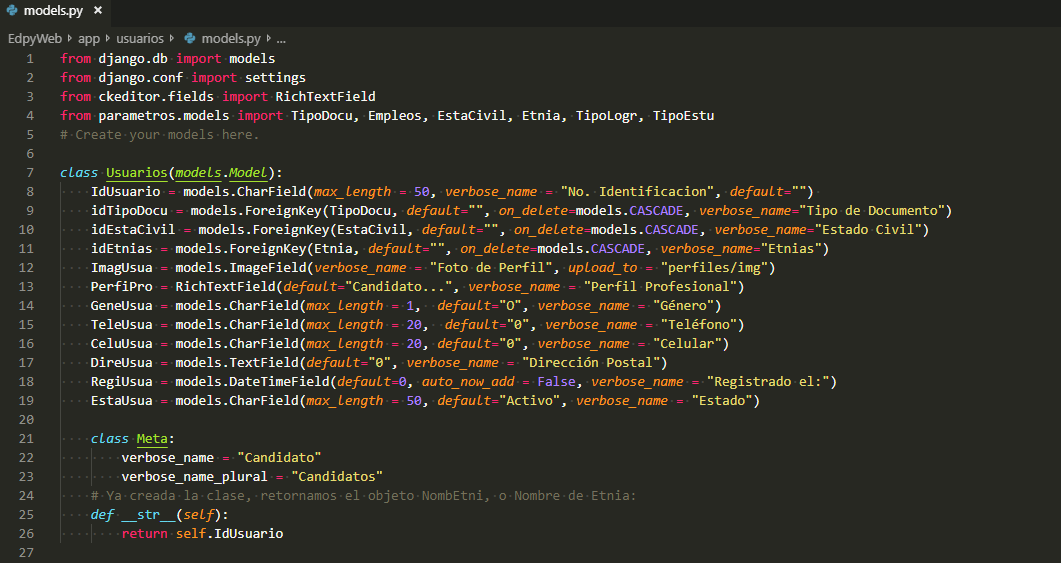


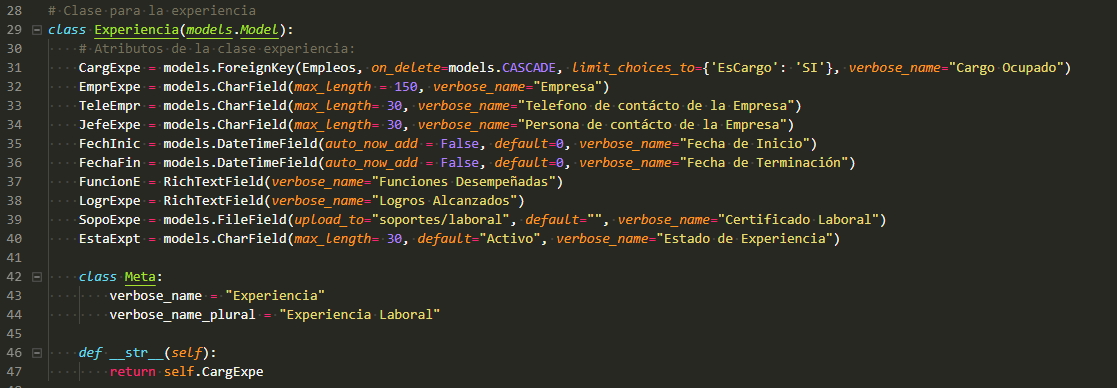
# VISTAS BASADAS EN CLASES POO

Las vistas hasta ahora las hemos hecho basadas en funciones, estas son practicas a la hora de tan solo renderizar, pero las vistas basadas en clases nos ofrecen todas las ventajas de la programación orientada a objetos; permiten crear herencias, abstracciones, paso de parámetros, extenderlas, ahorro de tiempo en el desarrollo, reutilización de métodos. Etc. Dicho esto nos dirigiremos al recurso puesto en el pie de esta página y utilizaremos algunos elementos que allí se proveen.[[5]](#footnote-5)

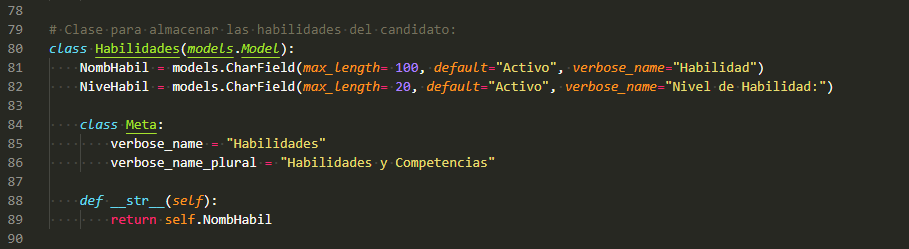
## App Usuarios y modelos de esta

Para el ejercicio, y enseñar como queda una vista basada en clases, he creado la app usuarios que tendrá los modelos: datosusua (la llamare usuarios), logros, estudios, experiencia laboral y habilidades o skills del usuario, para una mayor comprensión al momento de imprimir el documento.









En estas imágenes podemos encontrar algo bien interesante, los campos ***models.ForeignKey,*** estos campos como su nombre lo indica, son para incluir foráneas dentro de nuestro modelo.[[6]](#footnote-6)

En estos campos se pueden atribuir querys, como por ejemplo el atributo que definimos en el nombre del cargo en el modelo experiencia:

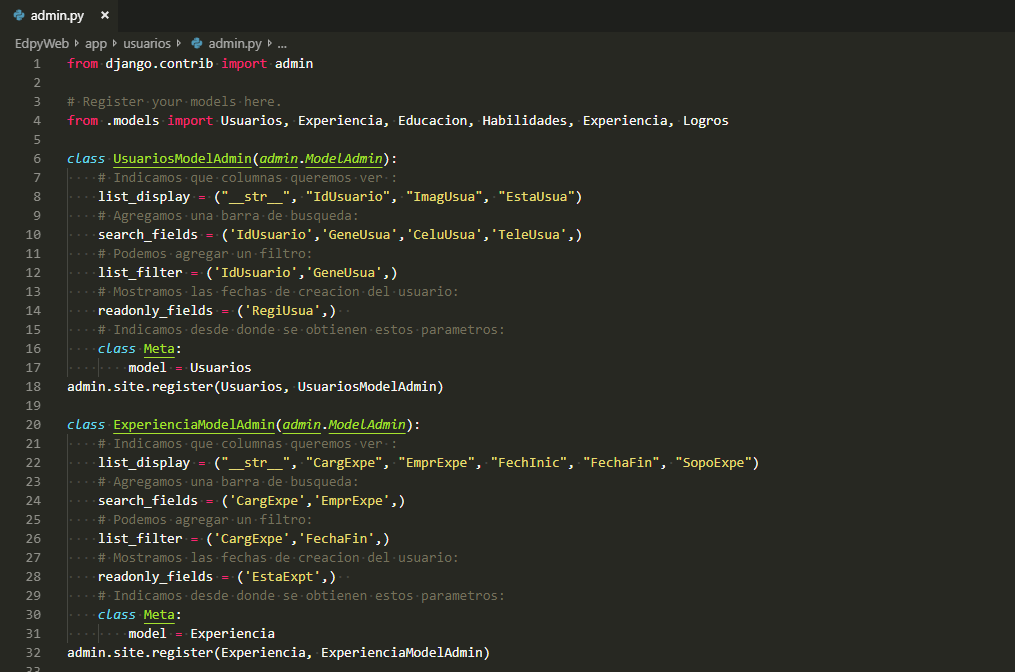
*limit\_choices\_to*={'EsCargo': 'SI'}

este campo es una llave foránea que ejecuta una condición similar a cualquier hecho en SQL:

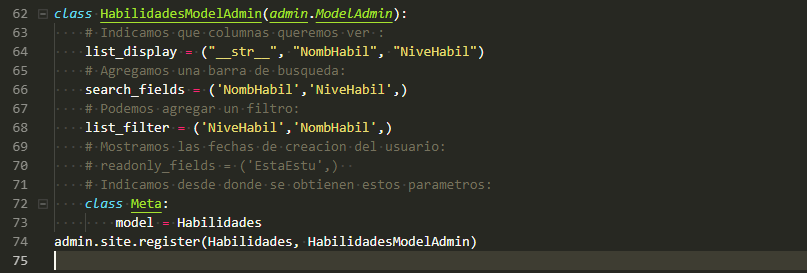
select \* from empleos where escargo = ‘SI’;

CargExpe = models.ForeignKey(Empleos, *on\_delete*=models.CASCADE, *limit\_choices\_to*={'EsCargo': 'SI'}, *verbose\_name*="Cargo Ocupado")

En el archivo admin.py:



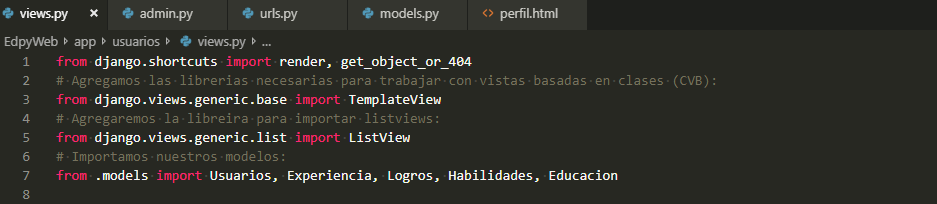




La idea es que todos los usuarios que se registren allí, y todos los datos que cada usuario registre, queden almacenados y podamos hacer clic y observar su contenido. Tomando las características de las vistas basadas en clases, podemos ir al total y al detalle de los datos que se carguen en la base de datos.

## Views.py de la app usuario:

El archivo views.py cambiara para este modelo, por la sencilla razón de que utilizaremos otra técnica para traer los objetos del modelo a la vista. Para poder implementar las vistas basadas en clases hay que agregar las librerías necesarias al archivo views.py de la app:



Persistencia en PDF:

https://docs.djangoproject.com/en/2.2/howto/outputting-pdf/

PUESTA EN MARCHA EN UN HEROKU

GLOSARIO

BIBLIOGRAFIA

<https://codeburst.io/create-a-django-api-in-under-20-minutes-2a082a60f6f3>

<https://www.udemy.com/probar-django-110-crear-un-blog/learn/v4/t/lecture/5682648?start=0>

<https://code.tutsplus.com/es/tutorials/create-a-database-cluster-in-the-cloud-with-mongodb-atlas--cms-31840>

https://www.youtube.com/watch?v=pWbMrx5rVBE

<https://www.youtube.com/watch?v=iKB_4HWKMCc>

<https://www.youtube.com/watch?v=3hdO_LXgZ_E&list=PLi26yBUIdF2-ppuQIOywLdoszLy5rnx83&index=1>

<https://www.codingforentrepreneurs.com/projects/try-django-2-2/>

<https://django-mongodb-engine.readthedocs.io/en/latest/tutorial.html>

<https://www.youtube.com/watch?v=DDt9VdTR_rg>

1. https://es.wikipedia.org/wiki/Pip\_(administrador\_de\_paquetes) [↑](#footnote-ref-1)
2. Al final de este documento encontrarás un glosario de términos, precisamente con estos comandos. [↑](#footnote-ref-2)
3. Este modelo Tambien lo puedes probar con PostgrSql, o MySQL o cualquier otro motor de base de datos relacional. [↑](#footnote-ref-3)
4. Podemos encontrar más información en : <https://docs.djangoproject.com/en/dev/ref/forms/fields/> [↑](#footnote-ref-4)
5. Para muchas plantillas basadas en clases, ejemplos, documentación y tutoriales vista: <https://ccbv.co.uk/> , es un recurso muy útil y está completamente actualizado. [↑](#footnote-ref-5)
6. Se pueden encontrar mucha más información en la documentación oficial de django: <https://docs.djangoproject.com/en/2.2/ref/models/fields/> [↑](#footnote-ref-6)