Índice de contenido

Presentación:

Idea Principal

Durante la realización de test de seguridad a aplicaciones web o servidores en general, por norma un auditor tiene sus herramientas establecidas, muchas de las cuales, si se maneja en entornos *GNU/Linux*, podrán ser por ejemplo comandos del sistema o scripts lanzados desde la shell entre otras posibilidades. Con este proyecto pretendo automatizar esta tarea de lanzar comandos y ordenar un poco el caos que genera almacenar un fichero de texto con la salida del comando en cualquier sitio. El proyecto se llama SecTesting y el objetivo principal es organizar los comandos lanzados a su respectiva auditoría o **test,** para poder después redactar un informe teniendo toda la información organizada para consultarla.

Introducción

En el presente documento se redactan los detalles y pormenores de la aplicación SecTesting, la función de dicho software es almacenar clientes que soliciten una auditoría, almacenar tests relacionados con su respectivo cliente, comandos predefinidos que se puedan necesitar en una auditoría y poder lanzar comandos por consola desde un panel web y almacenar su salida. En dicho panel web el empleado podrá listar y ver los detalles de los clientes, listar los comandos lanzados, ver su salida y lanzar un comando asociándolo a un test de un cliente. Finalmente el empleado podrá redactar un informe de auditoría y almacenarlo en la aplicación para luego poder enviarlo a imprimir desde el navegador.

Hardware y software a utilizar

Hardware

Sectesting es una aplicación que de base no requiere hardware de grandes características, no obstante, si se desea utilizar comandos más pesados, se recomienda un pc de al menos **2Gb de memoria RAM** y un procesador de **dos núcleos**, también es un requisito indispensable tener una **tarjeta de red**, si la tarjeta es de red inalámbrica debemos asociar nuestro sistema antes de lanzar la aplicación. Recordamos que esta aplicación es un panel web para lanzar comandos al sistema, por tanto depende de las características de tal su rendimiento. Debido a la funcionalidad de los Threads(hilos), la aplicación podrá ejecutar procesos en segundo plano, lo que podría mermar el rendimiento de la máquina servidor.

Por otra parte también es compatible con RaspberryPI, concretamente se ha probado en la distribución raspbian (debian para raspberry), sólo debemos tener instaladas sus dependencias (descritas anteriormente), su funcionamiento es más lento pero proporciona una mayor versatilidad a la hora de realizar pruebas in situ en una empresa.

Software

Sectesting es una aplicación que por ahora funciona en sistemas **Debian/Ubuntu**, para su correcta ejecución necesitamos instalar una serie de dependencias en el sistema donde vayamos a correr la aplicación, a continuación dejo una lista de dependencias necesarias:

1. apt-get install python2.7
2. apt-get install python-django
3. apt-get install python-pip
4. pip install django-extensions
5. python-threading
6. apt-get install python-pygraphviz
7. python-subproccess
8. apt-get install nmap
9. apt-get install whois
10. ping

Una vez tengamos todas las dependencias instaladas podemos ejecutar el servidor a través de la consola. Los detalles de la administración del servidor a través de la consola figuran en el manual del administrador en páginas posteriores.

Diseño

Tecnología usada

Django: En la elaboración de este proyecto he usado principalmente el abanico de herramientas que nos ofrece Django. Gracias a las herramientas de Django podemos programar en Python el sitio web que necesitemos. Su principal característica es la compartimentación del código, es decir, Django separa el código fuente en distintas clases dependiendo de la función de cada clase, de tal forma tenemos una clase para los datos, otra para las vistas, otra para las url ... Posteriormente veremos un esquema más detallado acerca de las clases. Otra principal ventaja es que incorpora su propio servidor, así que sólo necesitamos tener las dependencias instaladas en nuestro sistema y ejecutar el servidor para poder usarlo.

Python: La base principal de Django y de este proyecto es desarrollar una aplicación web con Python. El motivo de esta determinación son las múltiples ventajas a la hora de programar en este lenguaje de programación. Como primer aspecto me gustaría destacar la limpieza e intendancia en el código que el propio intérprete te obliga a tener, dejando por sistema el código más visible que en otros lenguajes. Esto sucede debido a que el intérprete de Python interpreta los espacios para ubicar el código, es decir, no necesitamos llaves { }, simplemente tabulando bien el código dejamos dicho qué parte del código pertenece a una clase, o a un bucle que esté dentro de un método. Posteriormente se podrá observar con más detalle estas referencias en el apartado de Codificación.

Aparte de las funcionalidades de Django he realizado las operaciones relacionadas con interacciones del sistema en un motor desarrollado en Python puro. Este motor cumple la función de lanzar el comando al sistema y de escribir el estado según se encuentre el comando lanzado.

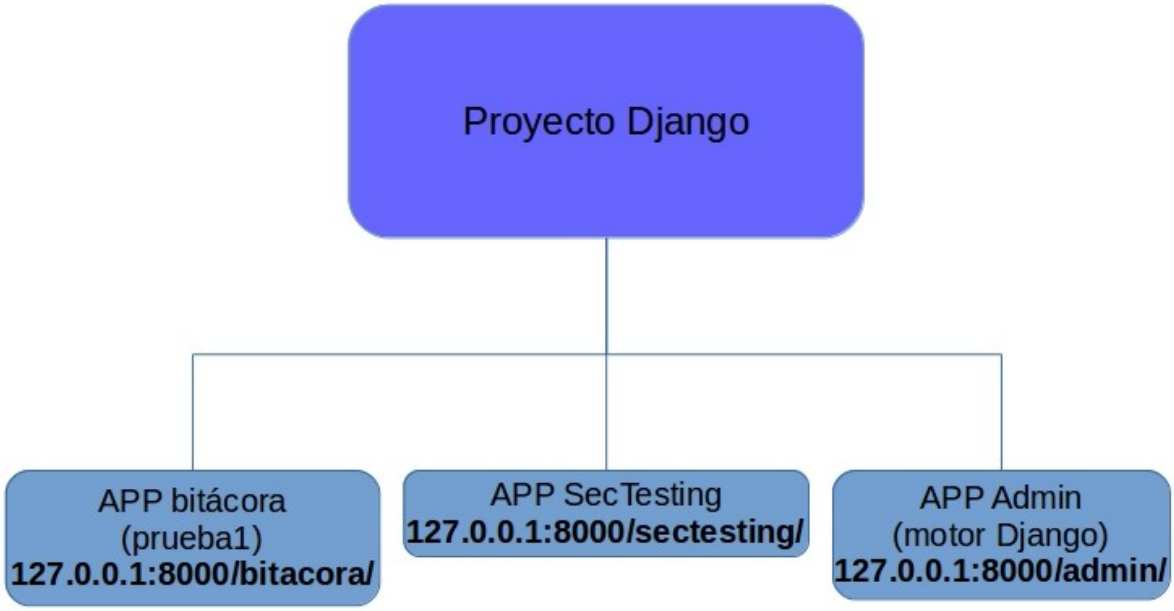
CSS 3.0: Para el diseño gráfico y maquetación de la aplicación he usado principalmente hojas de estilo CSS. El motivo principal de elegir CSS es tener todos los estilos compartimentados en una misma hoja de estilo, siendo muy útil a la hora de programar plantillas y dejar toda la web con el mismo estilo. También será una ventaja cuando se pretenda modificar la aplicación para versiones posteriores. Por otra parte no se depende de plugin como flash o java, en este caso la aplicación **SecTesting sólo requerirá un navegador de internet por defecto**, sin ningún plugin adicional.

SQLite3.0: Django ofrece la posibilidad de interactuar con varios motores de bases de datos como por ejemplo MySQL o MariaDB, el objetivo principal de esta aplicación es que pueda ser **portable**, es decir, que pueda guardarse tal y como está y ejecutarse en otro ordenador. Por tanto, veo más conveniente utilizar un motor de base de datos que pueda incluirse en la propia aplicación como SQLite. A posteriori, si se desea, django ofrece la posibilidad de cifrar la base de datos SQLite3.0 añadiendo unas librerías destinadas a tal propósito.

Implementación en Django:

A la hora de implementar un proyecto en Django debemos tener en cuenta que en un mismo proyecto pueden haber más de una app, en este caso, el proyecto consta de dos apps desarrolladas además de la app de administración, la cual sólo tenemos que configurar correctamente con su clase admin.py (describiremos la función de cada clase más adelante).

Esquema de las apps del proyecto:



Como se ha indicado anteriormente, Django posee una estructura de clases destinadas para una función en particular cada una, como clases principales de una aplicación Django podemos enumerar las siguientes:

views.py: En esta clase programamos las vistas, es decir codificamos los datos que se le entregan a la plantilla para que ésta los muestre por pantalla. La vista se encarga de gestionar la parte interna de programación de python y devolver los resultados a modo de diccionario de datos.

urls.py: esta clase almacena las urls de la aplicación, cada url estará vinculada a una vista, en ese momento la vista cargará el template que tiene asociado para mostrar la página.

model.py: Es la clase destinada al modelo de datos, es decir, a indicar qué información guarda la aplicación y cómo la guarda. Esta clase es uno de los mejores aportes de Django, gracias a ella podemos dejar reflejado un modelo de datos único independientemente del motor de base de datos al que lo podamos implementar. Este modelo de datos es una capa de abstracción para la base de datos que transforma las entidades en objetos pudiendo relacionarlos y establecer herencias entre clases dependiendo de la relación que le indiquemos al campo del modelo. Esto es una gran ventaja a la hora de usarlo en plantillas para acceder directamente a su clave foránea en busca de información sin necesidad de realizar ninguna consulta, sólo consultando la información que nos indica el objeto, el cual se encarga de escribir la información necesaria en la base de datos.

forms.py: Esta clase está destinada a almacenar formularios de entrada de datos. Con esto podemos dejar definidos distintas clases de formularios para una misma entidad que luego serán llamados desde su respectiva plantilla. Dentro de los formularios también se puede programar otra capa de validación, aunque por defecto la validación del formulario depende directamente de la que haya establecida en el modelo de datos.

admin.py: En esta clase definimos las entidades de nuestro modelo de datos que vamos a poder modificar en nuestro Django Admin, es decir, esta es la hoja de programación y configuración para nuestro panel administrador.

Django también posee una serie de directorios donde depositar un tipo de recurso u otro, a continuación dejamos descritos los más importantes:

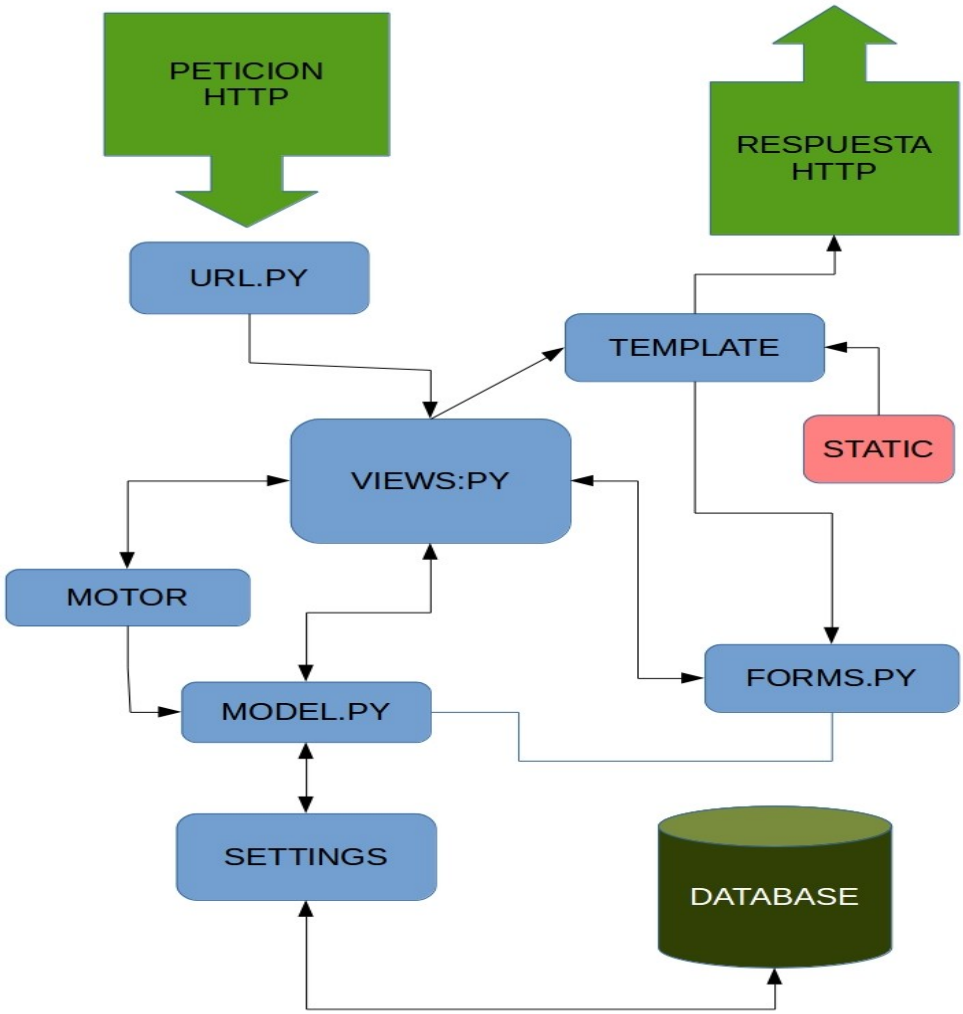
/templates/: En él se almacenan las plantillas de la aplicación.

/static/: En este directorio se almacenan contenido estático como Javascript, CSS, fuentes utilizadas, etcétera.

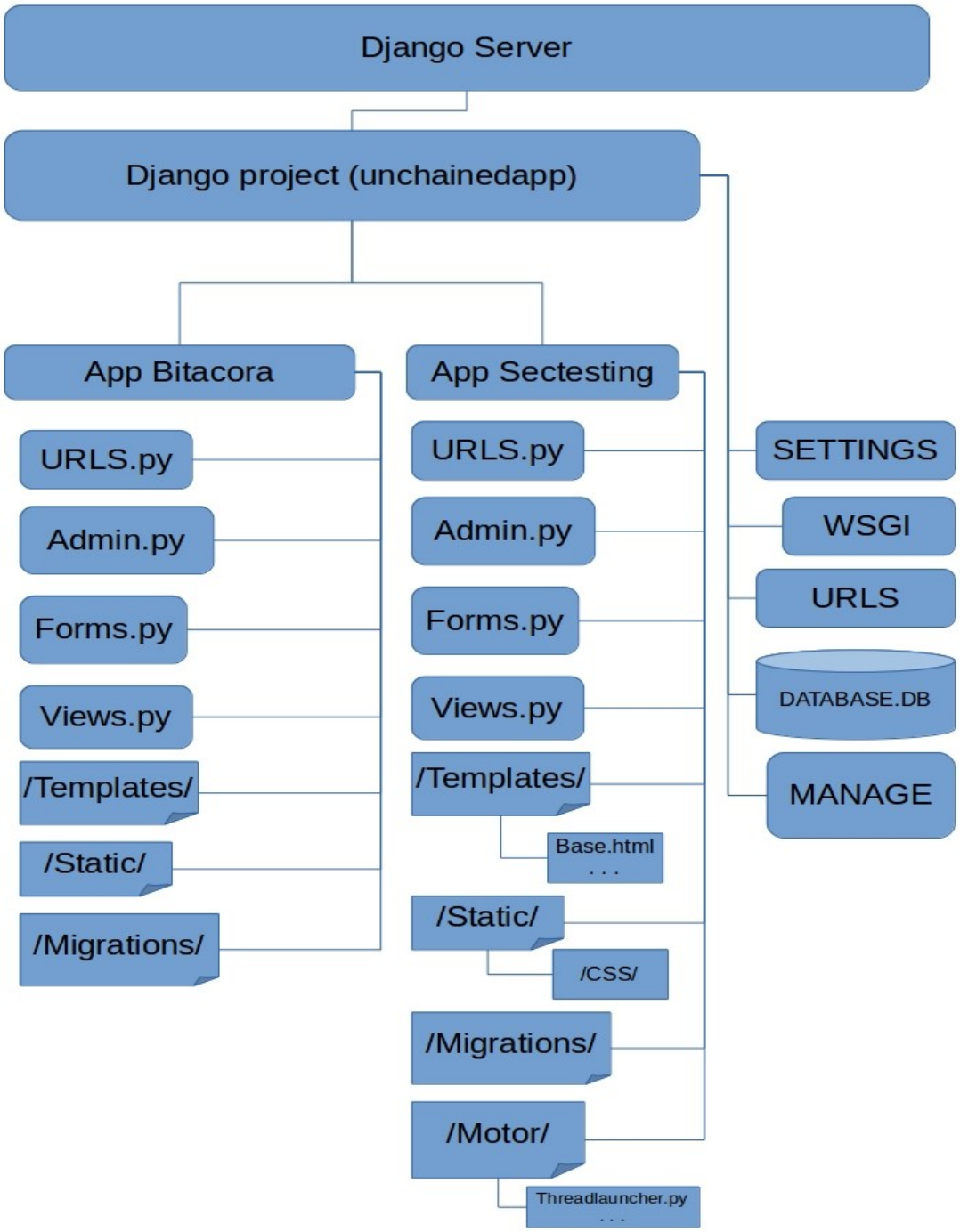
/migrations/: Dentro de este directorio se almacena la información relacionada con las migraciones de nuestra aplicación. Una migración es una operación necesaria para implementar el modelo actual en una base de datos determinada. En este proceso se escribe lo que está descrito en el modelo de datos a la base de datos, la aplicación al ser escalable permite la modificación del modelo de datos con la única tarea obligatoria de realizar la migración a la base de datos. Posteriormente veremos este aspecto en los manuales de administración.

A continuación vamos a mostrar los esquemas de la aplicación. Para lograr entenderlos hay que explicar que la aplicación consta de varias partes, por ello adjunto un esquema del modelo de datos completo de la aplicación (incluyendo los datos que recoge el motor de Django) y más abajo adjunto un esquema exclusivamente del modelo de datos de la app SecTesting. En cuanto al esquema funcional, reza al esquema general de una aplicación en Django con una modificación, el directorio motor, donde residen las clases relacionadas con el sistema y el motor de comandos que debe comunicar directamente con la vista y el modelo de datos.

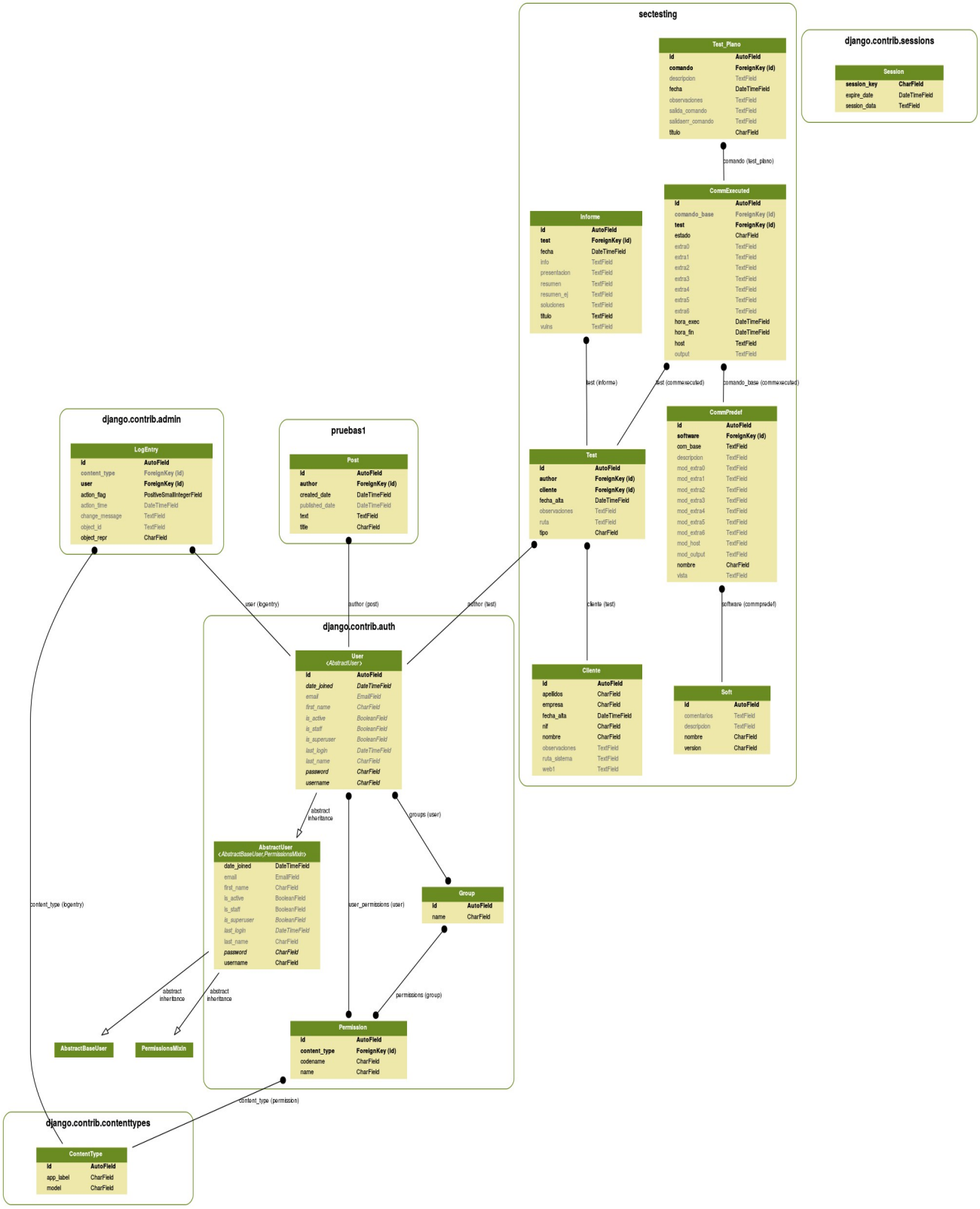
Esquema funcional de la aplicación:



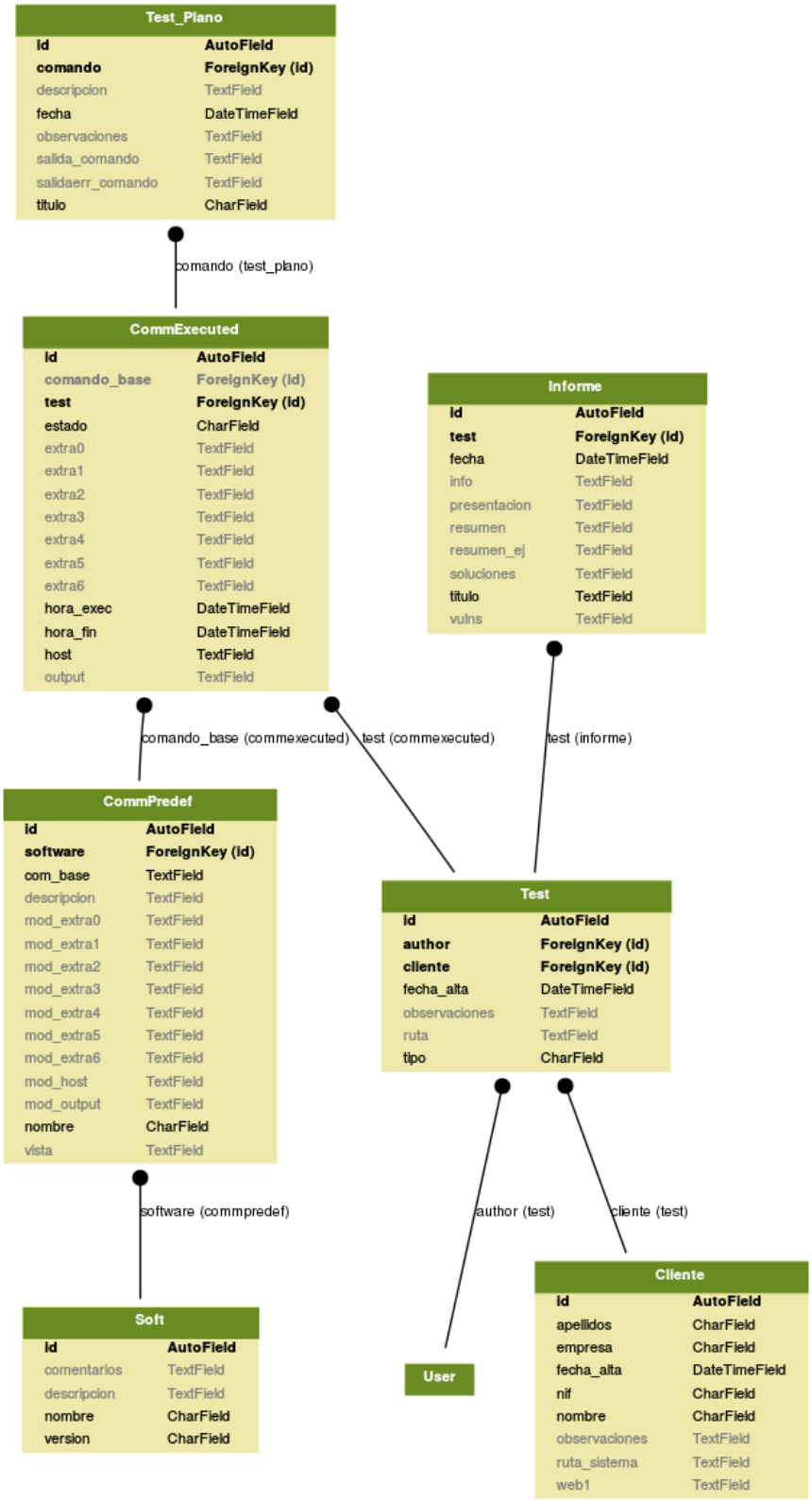
Esquema de la estructura de directorios del proyecto:



Esquema completo del modelo de datos:



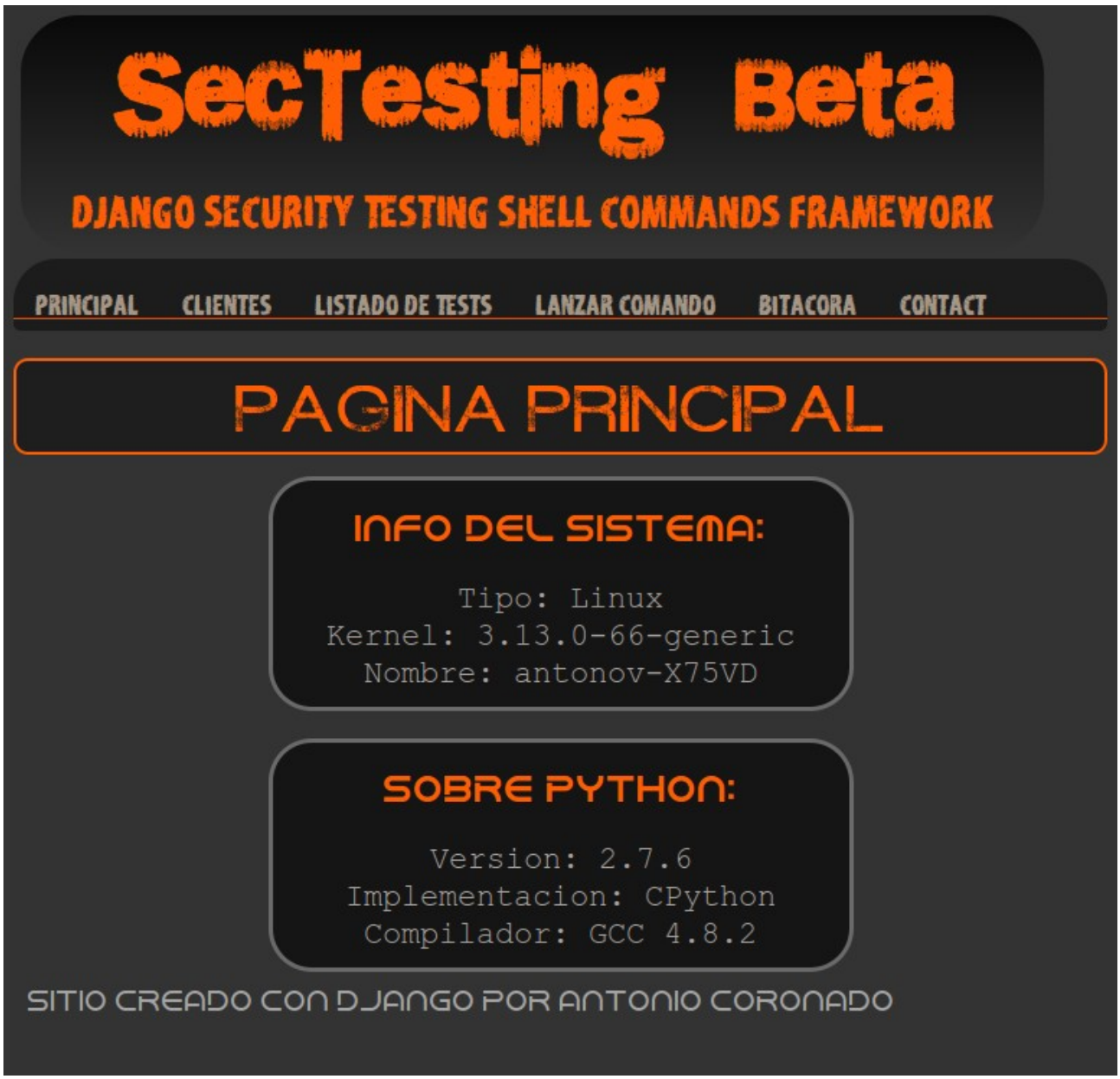
Modelo de datos de la app Sectesting:



Diseño de Plantillas

Las plantillas de nuestra aplicación tienen una "plantilla base" de la cual extienden el resto de plantillas que sólo tienen la referencia a la base y el bloque de contenido. Se ha desarrollado una jeraruía de plantillas para las apps SecTesting y Bitácora, ambas están maquetadas con CSS3.0

Captura de la plantilla de la página principal:



Codificación.

Entorno - Ninja IDE

Es un entorno de programación especialmente orientado a programar en Python, si usamos una distribución GNU/Linux lo tendremos por norma general en los repositorios. Este entorno ofrece una serie de ventajas a la hora de programar con Python, por ejemplo, interpreta a la perfección los espacios y le da su correcta dimensión para que no haya "descuadres" a la hora de intendar el código a base de espacios (ya sabemos que algunos IDE reducen el tamaño del espacio con respecto a un carácter normal, esto nos puede levar a confusión a la hora de programar en Python). Otra ventaja es el fondo oscuro sobre letras claras, lo cual lo hace cómodo a la vista.

Ninja IDE cuenta con una serie de plugins para facilitarnos la tarea de programar en python, en este caso sólo he instalado el plugin para Django de su web oficial, que permite reconocer las sintaxis de las clases de la aplicación.

Para su instalación en sistemas Debian/Ubuntu teclearemos en el terminal:

```
sudo apt-get install ninja-ide
```

De no tenerlo deberíamos instalar sus repositorios, a continuación dejo el enlace al tutorial completo de instalación:

http://conectabell.com/python-ninja-ide/

Lenguajes y herramientas.

Para realizar este proyecto se ha usado fundamentalmente el lenguaje de programación Python. A parte de los motivos indicados anteriormente, se ha de decir que el intérprete de Python da bastante rendimiento pudiendo procesar rápidamente los comandos, se puede hacer la prueba usando el comanfo "nmap -F" que ya viene predefinido por defecto. Se programa principalmente con las librerías por defecto de python y django usando además librerías adicionales propias del leguaje python como subproccess o "os"

Funcionalidades básicas:

Vistas:

Como indicamos anteriormente, las vistas son las encargadas de darle a la plantilla los datos que debe mostrar, son parte fundamental del motor principal de la aplicación. Dentro de las vistas programadas en este proyecto caben destacar las siguientes:

Vista clientes_list:

En esta parte podemos ver como en la vista prepara los datos para enviar a la plantilla, en este caso simplemente consulta todos los objetos de cliente y los devuelve en forma de diccionario de datos. El diccionario de datos es un tipo de variable que permite almacenar datos, es básicamente como lo que conocemos en java como un array multidimensional. Cabe destacar del código que en el retorno devolvemos la renderización de todo el contexto que engloba la petición (request), plantilla y los datos para la plantilla. A continuación dejo el trozo de código:

```
def clientes_list(request):

clientes = Cliente.objects.all()
return render(request, 'sectesting/cliente_list.html',
                          {'clientes': clientes})
```

Vista escribir_informe:

Esta vista recoge la petición POST y valida el formulario, en este caso valida con las reglas establecidas en el modelo, si hubiese reglas adicionales se escribirían en el fichero forms.py. Cuando ha terminado nos devuelve a la vista de "lista de informes". Si no está validado correctamente el formulario nos devolverá de nuevo al formulario.

```
def escribir_informe(request):
     if request.method == "POST":
     form = InformeForm(request.POST)
          if form.is_valid():
          inf = form.save(commit=False)
          inf.save()
          return informe_list(request)
     else:
          form = InformeForm()
     return render(request, 'sectesting/escribir_informe.html',
                                                  {'form': form})
```

Vista Nmap -A:

En la vista de nmap, debido a la cantidad, de tiempo que puede necesitar (dependiendo de la máquina) decidí implementarlo a base de threads, de esta forma, la aplicación ejecuta el comando en segundo plano, y no deja en espera al navegador como pasa con ping o whois, la aplicación web podrá continuar sin problemas. Al no tener implementadas las colas dinámicas de hilos, no es posible ejecutar a la vez en segundo plano dos nmap de rango completo, hay que esperar a que el NMAP en ejecución cambie su estado de init a ended. A continuación dejo el código:

```
def nmap_all_tcp(request):
      if request.method == "POST":
            form = NmapAllTCPForm(request.POST)
            if form.is_valid():
                  com = form.save(commit=False)
                  host = com.host
                  base = 5
                  com.save()
                  compredef = CommPredef.objects.get(pk=base)
                  pkh = com.pk
                  comlaunched = CommExecuted.objects.get(pk=pkh)
                  comlaunched.comando_base = compredef
                  comlaunched.save()
                  thread1 = CommandThreadBH(3, "Nmap All TCP", base,
                                                      host, pkh)
                  thread1.start()
                  return test_list(request)
            else:
            form = NmapAllTCPForm()
return render(request, 'sectesting/nmap_fullrange.html', {'form': form})
```

Codificación de plantillas

Plantilla consulta foreign key

En este trozo de código accedo desde la variable retornada por la vista a una clave ajena, pudiendo consultar sus campos. En este ejemplo la variable cmd almacena el comando ejecutado, consulta la pk almacenada en comando_base y extrae el comando base.

```
{{ cmd.comando_base.com_base }}
```

Plantilla URL

Este código python de plantilla apunta a una vista y le pasa un argumento, pk.

```
{% url 'sectesting.views.command_detail' pk=cmd.pk %}
```

Plantilla Base

Esta plantilla es la que se usará de base para montar el resto de plantillas, esto se lo indicamos con los indicadores de bloques de django, a continuación pongo un trozo de código para visualizarlo.

```
<div>
     {% block content %}
     {% endblock %}
</div>
```

Plantillas Hija

Las plantillas hija son las que descienden de la plantilla base, en estas páginas llamamos a la plantilla base de la siguiente forma:

```
{% extends 'sectesting/base.html' %}

{% block content %}
     <div class=titulazo3> Listado de Comandos </div>
{% endblock content %}
```

Motor:

Separada de la vista, el motor es una parte fundamental de la aplicación, de él se obtienen los métodos que lanzan el comando al sistema o que obtiene información del sistema.

Threadlaucher.py

Esta es la clase que lanza los hilos, la creamos con el método constructor y le pasamos un ID, un nombre, un ID de comando base, un host y el comando ejecutado del modelo de datos para escribir el estado posteriormente. Una vez la hayamos creado, sólo tenemos que llamar al método run, el cual lanzará el comando y se encargará de escribir las salidas a sus correspondientes campos en la entidad Test_Plano

```
class CommandThreadBH (threading.Thread):

      def __init__(self, threadID, name, com, hos, p):
            threading.Thread.__init__(self)
            self.threadID = threadID
            self.name = name
            self.comando = CommPredef.objects.get(pk=com)
            self.host = hos
            self.pkh = CommExecuted.objects.get(pk=p)


      def run(self):
            cmbase = self.comando.com_base
            host = self.host
            pkh = self.pkh
            pkh.hora_exec = unicode(datetime.now())
            pkh.estado = "init"
            pkh.save()
            comando = cmbase + " " + host
            proc = subprocess.Popen(comando, stdout=subprocess.PIPE, shell=True)
            (out, err) = proc.communicate()
            pkh.hora_fin = unicode(datetime.now())
            pkh.estado = "ended"
            pkh.save()
            nuevoplano = Test_Plano.objects.create(titulo=self.name,
            salida_comando=out,
            salidaerr_comando=err,
            comando=pkh)
            nuevoplano.save()
```

Información del sistema.

Para obtener información del sistema he utilizado las librerías **os** y **platform**, ambas libreriías están incluidas en python por defecto, con os saco el uname del sistema, con platform saco información sobre la versión de Python instalada, compilador, etc. Antes de devolver el resultado, empaqueto todos los resultados que vaya a usar en un diccionario de datos, así en la vista trabajo más organizadamente con el.

```
def getOSbasic():
    sysuname = os.uname()
    tsis = sysuname[0]
    ker = sysuname[2]
    mnom = sysuname[1]
    dev = {'tipo': tsis, 'kernel': ker, 'nombre': mnom}
    return dev
```

Modelo de datos:

El modelo de datos es una parte fundamental de la aplicación, en el definimos todos las entidades de la aplicación, a la hora de salvar los datos desde el código usaremos ordenes como **`form.save()`**.

Podemos comprobar en el código que añadimos dos métodos debajo de la definición de los campos, el primer método es simplemente para guardar, el siguiente método usa **`__str__`** como método para devolver un string que será empleado en Django Admin para mostrar más información acerca de la entidad a la hora de seleccionarla, lo comprobaremos más adelante en el manual del administrador.

```
class Soft(models.Model):
      nombre = models.CharField(max_length=100)
      version = models.CharField(max_length=100)
      descripcion = models.TextField(null=True, blank=True)
      comentarios = models.TextField(null=True, blank=True)

      def publish(self):
            #self.published_date = timezone.now()
            self.save()

      def __str__(self):
            devuelve = self.nombre + " (" + self.version + ")"
            return devuelve
```

Existen limitadores para estos campos, blank por ejemplo indica si en el formulario puede introducir nada, null, indica si el campo de la db es nulleable o no.

Conexión con base de datos

Como hemos indicado antes, el modelo es una capa de abstracción para la base de datos, ahora podemos configurar el motor de base de datos que queramos en django, ya sea mysql, oracle... yo en este caso me he decantado por SQLite3, ya que es la solución portable más a mano que dispongo, y su límite de almacenamiento son 2Tb.

Por otra parte obtengo una buena velocidad a la hora de escribir datos y no necesito conexiones a localhost, voy directamente al fichero de base de datos.

Para configurar un motor de base de datos debemos irnos al fichero settings.py, en el directorio del proyecto django.

A continuación adjunto las líneas de código del archivo setting.py relacionados con la base de datos:

```
DATABASES = {
     'default': {
     'ENGINE': 'django.db.backends.sqlite3',
     'NAME': os.path.join(BASE_DIR, 'database.db'),
               }
          }
```

Como podemos ver, tan sólo necesitamos instalar el módulo django de la base de datos que queramos e indicar en esta parte el motor que usará este proyecto de django.

Implementaciones de seguridad:

Sesiones

En este proyecto se utilizan sesiones para controlar el acceso de un usuario, gracias al administrador de Django podemos gestionar usuarios y permisos, denegando incluso el acceso al panel de administración a determinados usuarios. En este caso he creado dos usuarios y un grupo. El primer usuario es el administrador de la aplicación, con este usuario podemos gestionar las entidades del modelo de datos, usuarios y grupos entre otras funcionalidades que se le pueden añadir a posteriori.

Los usuarios implementados en la aplicación son los siguientes:

Administrador: antonov **pass:** antonov

Usuario: sectesting **pass:** sectesting

El **grupo** implementado se llama **sectesting**, este grupo **no posee privilegios para acceder al panel de administración**, sólo se le permite acceder a la aplicación sectesting y al blog (bitácora). En esta situación el administrador es el encargado de la mayoría de las tareas, dejando en manos del usuario la ejecución de comandos, la visualización de datos y el listado y escritura de informe. Se considera que este tipo de aplicación es bueno que un administrador la vaya gestionando, el uso de comandos de sistema, poder ejecutarlos en segundo plano puede llevar a un uso indebido por despiste.

Decoradores

Para la implementación del **requerimiento de sesión autenticada** he usado un decorador en la vista, el cual comprueba si el usuario está logueado y lo envía al login en caso de no estarlo. A continuación dejo un decorador, su sintaxis es bastante simple:

```
@login_required(login_url='django.contrib.auth.views.login')
```

Cifrado de contraseñas

Las contraseñas de los usuarios y administradores están cifradas con el algoritmo **sha_256** con **20000 iteraciones**, algo que resulta imposible de sacar con equipos convencionales.

Manuales

Guía de Instalación:

La guía de instalación es relativamente corta, ya que la aplicación viene preparada para ser migrada de un sistema linux a otro. En primer lugar debemos instalar todas las dependencias necesarias, en este caso son pocas, listo a continuación los comandos necesarios para instalar las dependencias en Debian/Ubuntu:

```
# apt-get install whois
# apt-get install nmap
# apt-get install python2.7
# apt-get install python-pip
# pip install django
```

Para su instalación descomprimimos el archivo ProyectoAntonioCoronado.rar (que está en el disco adjunto) en el directorio que queramos y listo. Usaremos los siguientes comandos:

```
unzip  ProyectoAntonioCoronado.rar
```

Luego en la guía de administración explicaré cómo arrancar el servidor y como realizar migraciones a la base de datos cuando actualicemos el modelo de datos.

Guía del Administrador:

Ejecución del servidor:

A la hora de ejecutar el servidor debemos abrir una shell, preferiblemente en bash (la que usa Ubuntu y Debian por defecto) ingresa en el directorio que hemos descomprimido en el proceso de instalación y ejecutar el siguiente comando:

```
python manage.py runserver 0.0.0.0:8000
```

Podemos indicar 127.0.0.1:8000 para que sólo sea accesible para localhost, en este caso sólo podemos acceder desde la máquina donde se esté ejecutando. He puesto el puerto 8000 en el ejemplo, pero igualmente lo puedes cambiar al puerto que desees.

Los errores de inicio suelen ser indicativos de que falta alguna dependencia, hay que revisar que esté todo bien instalado.

Migración:

Migrar es una tarea que se realiza a la hora de haber realizado una modificación en el modelo de datos que tenga que representarse en la base de datos física, para ello tenemos dos comandos fundamentales, uno genera la lista de lo que hay que migrar y el otro ejecuta las operaciones de migración. Hay que tener en cuenta que es un proceso delicado, ya que puede hacerse un lío con campos nulleables que pasen a no serlo, por ejemplo. Si algo ocurriera y no pudiésemos realizar la migración o perdiéramos la base de datos, tan sólo eliminamos el contenido del directorio /migrate/ y ejecutamos la migración. De esta forma eliminamos el histórico de migraciones y operaciones pendientes ya que aquí es donde se genera el problema básicamente.

Los comandos necesarios para realizar la migración son los siguientes:

```
python manage.py makemigrations sectesting
```

```
python manage.py migrate sectesting
```

Los comandos más o menos se entienden a simple vista, al final ponemos el nombre de la app donde hayamos realizado los cambios.

Panel de Administración

URL de Acceso: http://ipdelserver:8000/admin/

Esto nos llevará al panel de administración de la aplicación, aquí podemos seleccionar los modelos que hayamos indicado en **admin.py**, si deseamos poder acceder a un modelo desde este panel, necesitamos especificarlo en este fichero.

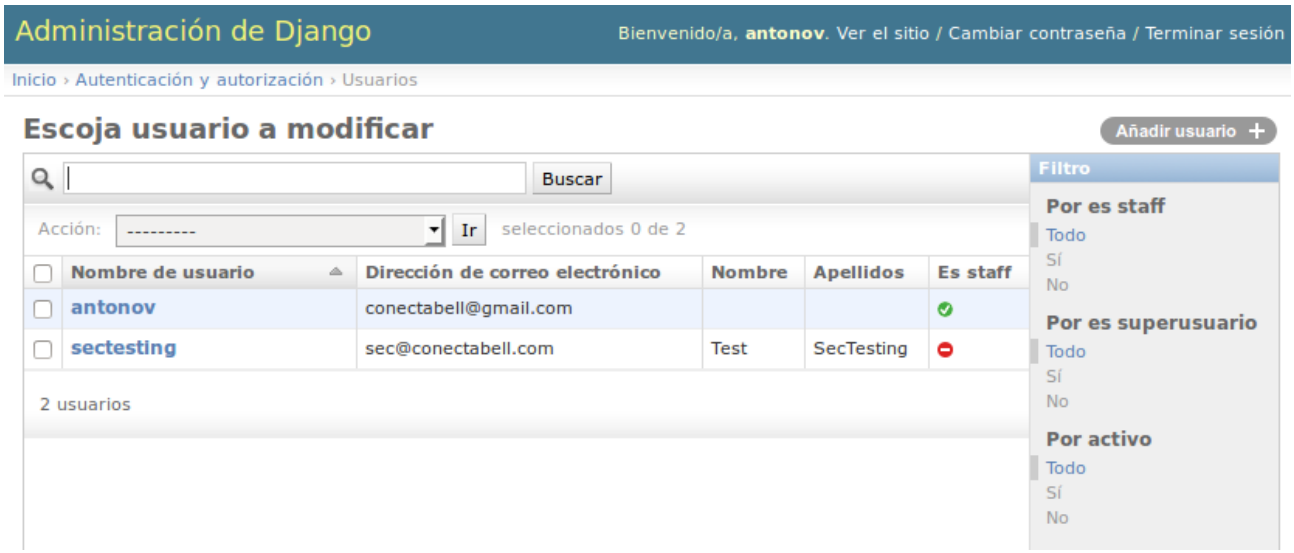
En el panel de administración podemos leer el contenido de la base de datos en base a nuestros modelos, también podemos añadir, editar y eliminar estos elementos, y borrarlos en cascada eliminando también sus referencias.

Cuando accedemos nos encontramos con este menú:

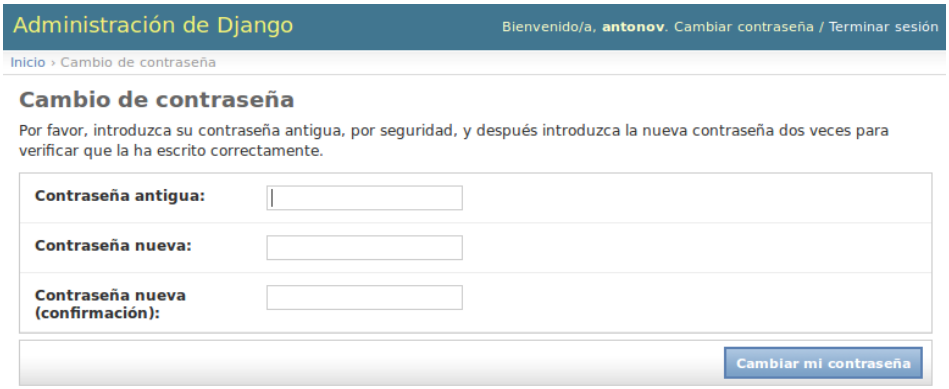
Es bastante intuitivo, podemos básicamente añadir y modificar los elementos incluyendo los usuarios grupos y privilegios.

Edición de usuarios grupos y privilegios

Para acceder al listado de usuarios simplemente haremos click en el vínculo usuarios del menú principal del administrador, esto nos llevará a un listado donde podemos pinchar para modificar el usuario que deseemos. En este listado podemos ver además las características generales del usuario, como por ejemplo si es **staff (permiso de acceso al sitio de administración)** o su correo electrónico y también podemos organizar los usuarios (si tuviéramos un gran volumen) por permisos generales.

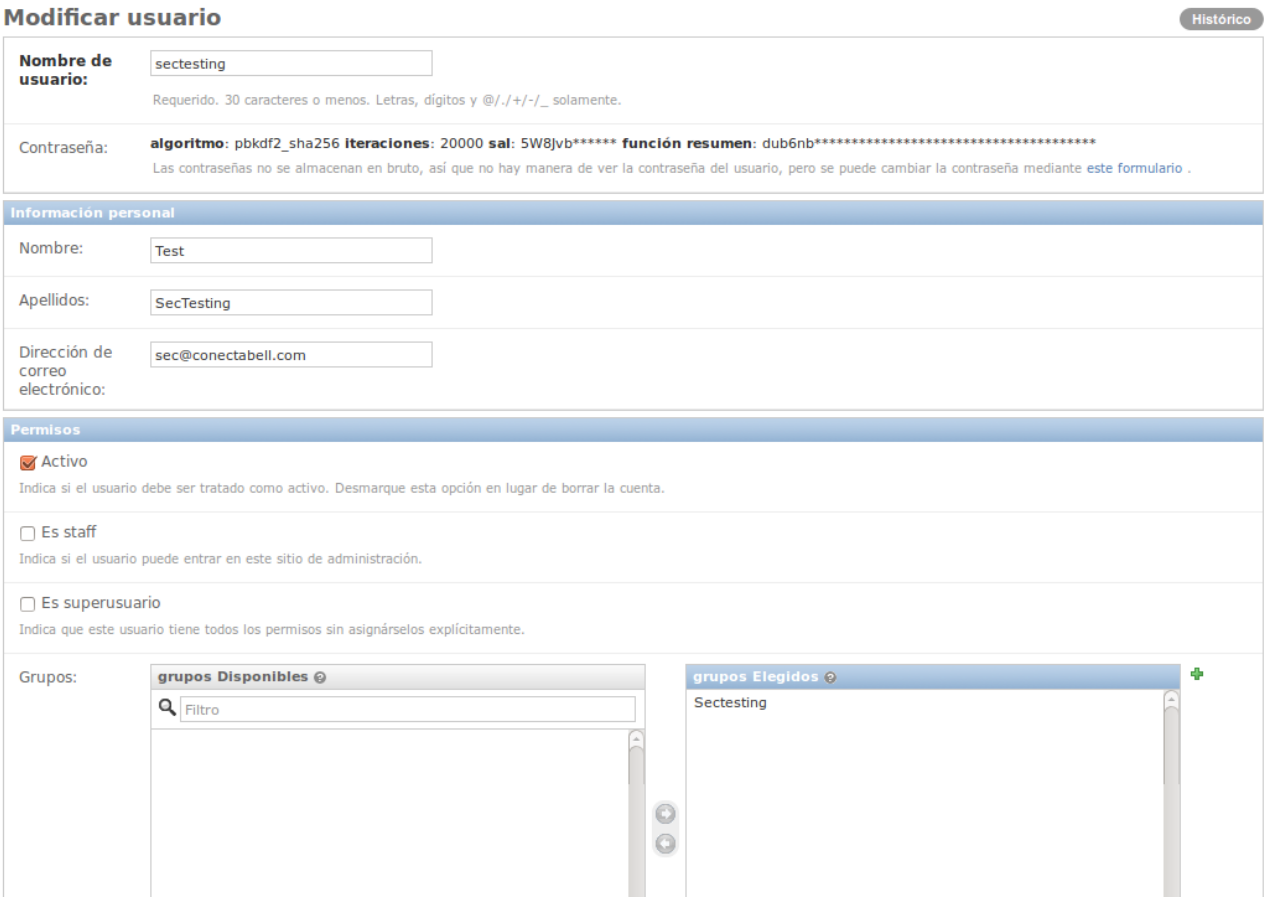


Una vez hayamos pinchado en el vínculo se nos abrirá una página donde podremos editar la información del usuario, para modificar la password debemos acceder a un formulario, el vínculo está al final del campo contraseña. Una vez ahí escribiremos la nueva contraseña.



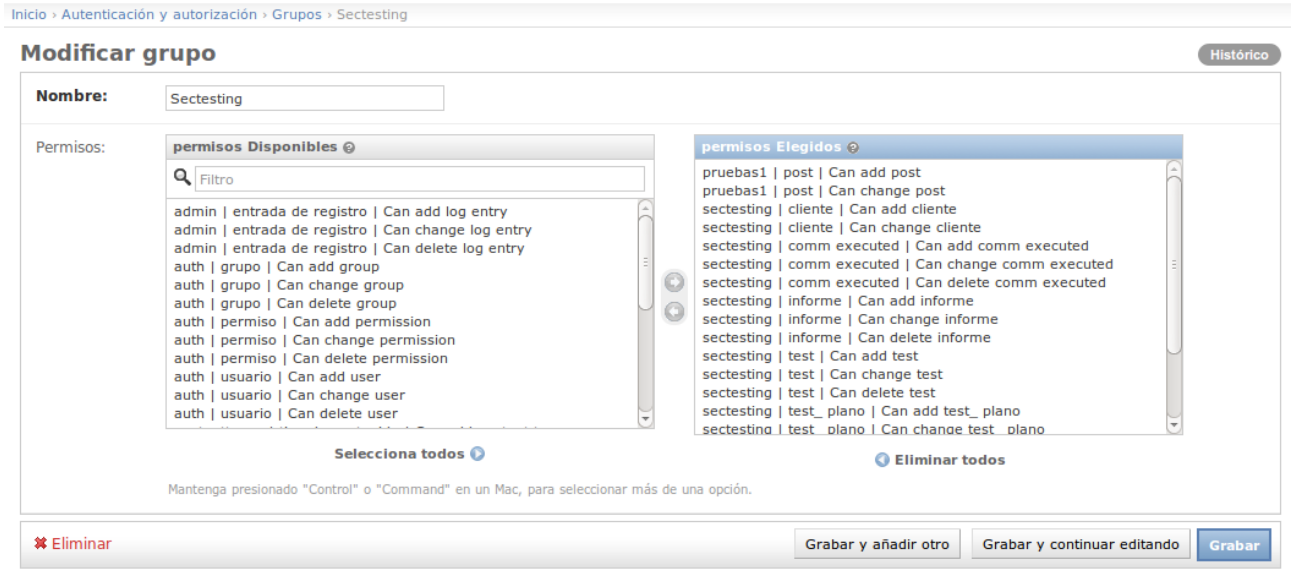
Sobre los permisos cabe destacar que podemos administrarlos de dos formas, la primera es añadiendo todos los permisos a mano a cada usuario, la segunda es asignarle permisos mediante grupos. En el panel de administración también podemos indicar si un usuario puede tener acceso al panel de administración (STAFF) y si es supersusuario, para modificar esta característica simplemente haz click en una casilla de verificación y dale a "grabar" al final del documento.En este caso he creado un grupo llamado sectesting el cual tiene privilegios para modificar los modelos de la app sectesting y bitácora(prueba1), no es STAFF, por tanto no tiene privilegios para entrar al panel administrativo.

La contraseña viene cifrada y por seguridad el panel no muestra todo el hash ni todo el salt y creo que **20000 iteraciones de sha256** son suficientes para disuadir a algunos malechores que pretendan descifrar la password.



Por otra parte si nos vamos al final de la página podemos ver el botón para **eliminar el usuario**.

Para editar un grupo nos vamos al menú principal y hacemos click en el vínculo correspondiente, esto nos llevará una lista donde si pinchamos en el único grupo que hay (sectesting) podemos ver un campo para escribir el nombre y un listado de privilegios que podemos añadir a este grupo. Más abajo tenemos la opción de eliminar el grupo. También tiene atajos para seleccionar todos los privilegios.



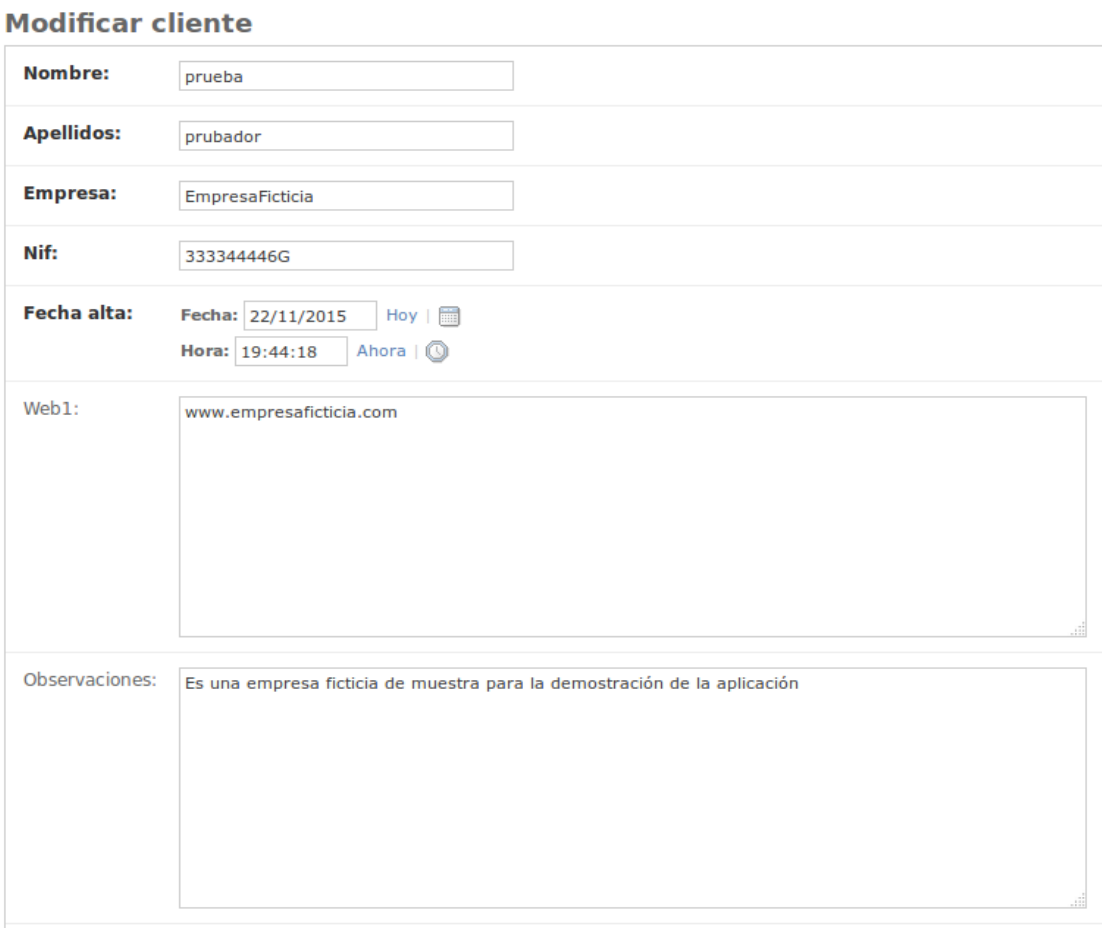
Administración general de entidades

Todas las entidades que se pueden administrar en este panel se pueden añadir, eliminar o editar, dependiendo de si está asociada o no con otra entidad, la eliminación podrá eliminar los campos de otras entidades que estén relacionadas con la que estamos eliminando. Para editar la entidad que queramos tan sólo tenemos que pinchar sobre ella para acceder al formulario relleno con sus datos. Estas acciones son similares exceptuando los cambios relacionados con el modelo de datos.

Administración de Clientes

Podemos añadir, editar y eliminar los campos de la entidad Cliente de la misma forma que modificamos los usuarios, tan sólo tenemos que hacer click en el vínculo Clientes del menú principal y podemos acceder a la edición de cada uno. Hemos de tener en cuenta que los campos nombre, apellidos, empresa y NIF son obligatorios, si no, no nos permitirá dar de alta el cliente.

Abajo del todo tenemos las opciones de eliminar el cliente o de guardar y dar de alta otro cliente nuevo. Debemos de tener en cuenta que debemos de cumplir los requerimiento que tengamos definidos en el modelo de datos, el formulario te avisará cuando los campos no estén correctamente rellenos.



Administración de Tests y comandos

Actualmente se pueden modificar los comandos que están implementados en la base de datos, si necesitásemos añadir un comando nuevo tendremos que programar también una vista para ello.
En la aplicación se almacenan los comandos y su correspondiente salida en tablas distintas, es decir, tenemos los comandos ejecutados en una tabla y la salida del comando en otra tabla relacionada a ésta. **La tabla comandos ejecutados y la tabla de test plano no hay que tocarlas**, aunque están en el panel de administración por si hiciera falta eliminar algún campo mientras se están haciendo pruebas (recordamos que es una versión beta).

Si queremos modificar el test tan sólo tenemos que hacer click en su correspondiente modelo y se nos abrirá la lista de test para seleccionar el que queramos modificar. Una vez lo hayamos seleccionado se nos abrirá un formulario con los campos del test relleno.

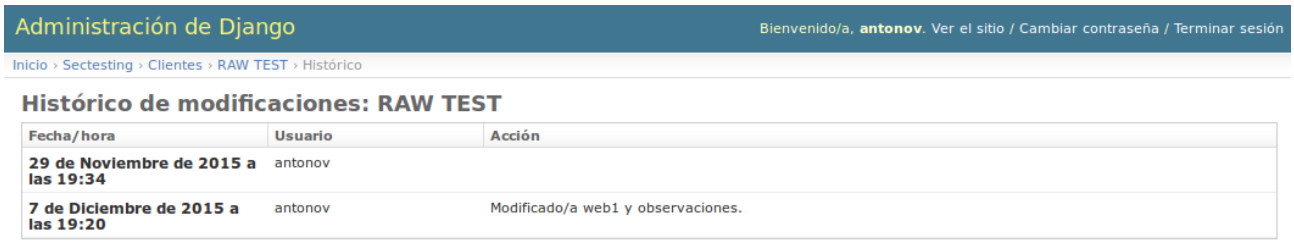
Debemos seleccionar cliente y tipo, en las observaciones irá anotada la información que nos remita el cliente acerca de su website o servidor, es tarea fundamental del administrador que al auditor le llegue la mayor cantidad de información posible.

Test Plano

Al acceder podemos ver una lista donde se implemente lo comentado en la parte de código, el vínculo que vemos es el **__str__** que definimos en el modelo de datos, así vemos algo de más información del contenido de los tests. Si seleccionamos un test plano nos llevará a la pantalla de edición de test plano, también podemos añadir un test plano nuevo añadiendo nosotros mismos la información.

Histórico

El panel de administración también guarda la referencia de cuando se modificó o eliminó alguna cosa, así el administrador podrá visualizar quién ha eliminado o editado el elemento que sea. Podemos hacer click arriba a la derecha del elemento que estemos editando para verlo. A continuación adjunto una captura del histórico del test "RAW_TEST" usado generalmente para comandos sueltos.



Guía del Usuario

Acceso a la aplicación

URL de acceso: http://127.0.0.1/sectesting/

La URL de arriba nos llevará directamente a la ventana principal de sectesting, ahí podemos ver algo de información del sistema y la versión de python que tiene el sistema. Si intentáis acceder a cualquier otra zona de la página requerirá una validación de usuario y contraseña, se debe estar registrado para poder acceder y el registro sólo lo puede llevar a cabo un administrador.

El **usuario por defecto** es sectesting, y su **password** sectesting también.

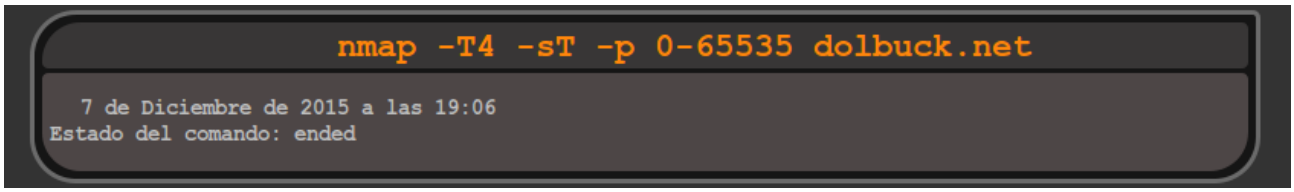
Listado de Test

En este apartado podemos acceder a la información de cada test, al hacer click sobre el se nos abrirá un listado para seleccionar el test que queramos. Al seleccionar el test podemos ver su información y también los comandos que se han ejecutado para ese test, si hacemos click en un comando iremos a sus detalles, donde estará la salida completa o la salida de error si la tuviera. El test llamado **RAW_TEST** es un test destinado a almacenar comandos de hosts aleatorios, es decir, que no vayan asociados a ningún test que se este realizando a un cliente.

Lanzar comando

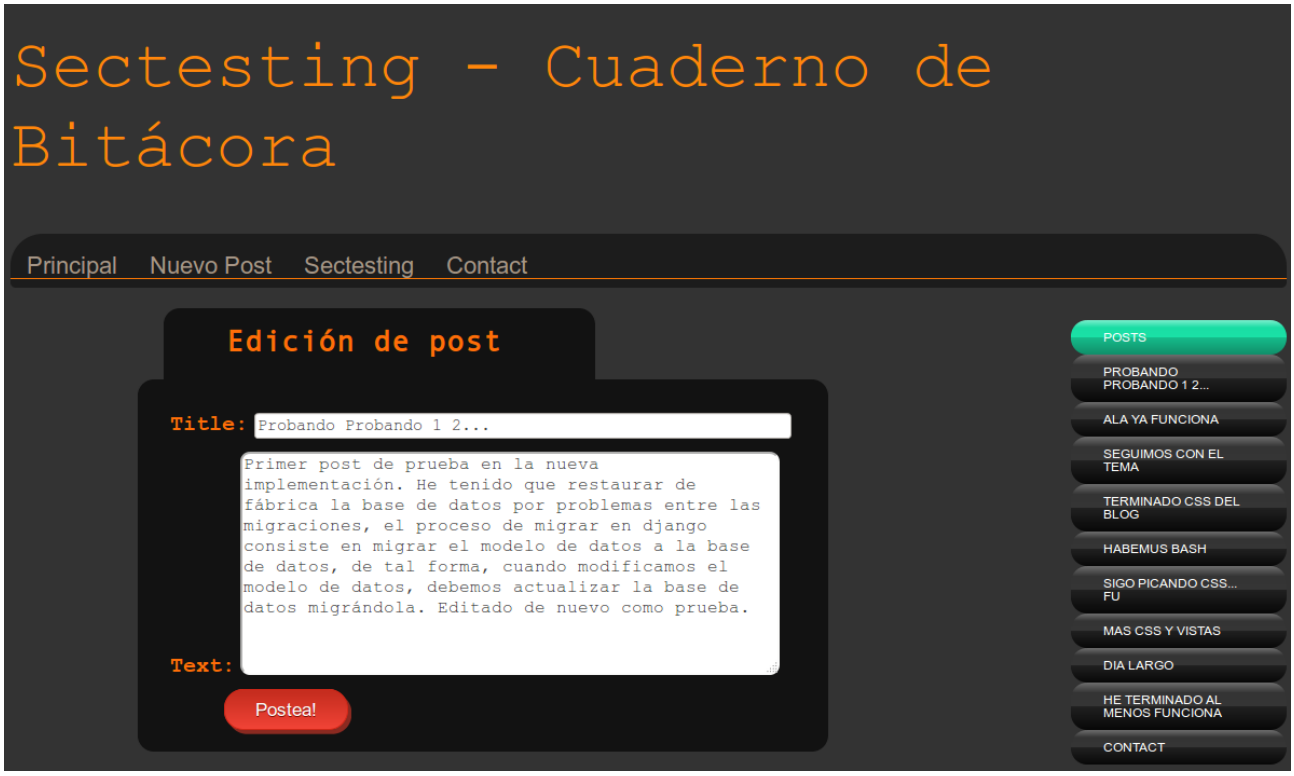
En este apartado dispondremos de una lista generada de los comandos dados de alta en el modelo, para seleccionar el comando haz click encima de el, esto te llevará al menú para especificar el test al que debemos relacionar el comando. Si no se tiene claro a qué test o el comando no va destinado a un test en concreto, seleccionaremos el test RAW_TEST destinado para estos menesteres.

Una vez se haya lanzado el comando nos llevará de nuevo a la página de test donde podemos en el listado de test el estado del comando lanzado, **no podemos acceder a los detalles con su salida hasta que el comando no haya concluido y cambie su estado a ended (terminado),** si continúa con el estado init (iniciado) no creará el objeto de test_plano y nos dará un **error 404 de django** por no encontrar el objeto. Podemos refrescar la página o navegar a donde queramos de la APP, sólo debemos tener en cuenta que si hemos lanzado un comando de la lista, hasta que no haya terminado no podremos lanzar ese mismo comando, pero sí de otra clase, por ejemplo, podemos dejar tirando un NMAP tcp completo, e ir haciendo ping y whois a nuestro antojo, o un nmap fast.



Bitácora

Es un apartado que se emplea para anotaciones, es una especie de blog interno de la aplicación, para que los auditores puedan dejarse mensajes ahí. Al pinchar en el enlace comprobaremos que se nos cambia el tipo de menú y la cabecera, no hay de que preocuparse, eso es debido a que es otra app aparte, que forma parte del mismo proyecto django. Podemos escribir un nuevo post haciendo click en el vínculo "Nuevo Post" también podemos editar un post existente accediendo a sus detalles y haciendo click en el botón editar.



Tenemos además a la derecha un menú con todos los post escritos para acceder a sus detalles en cualquier momento. Para volver a la app sectesting tan sólo tenemos que pinchar en el enlace y nos llevará directamente a la página principal.

Informes

Uno de los objetivos es poder redactar un informe gracias a la información que hayamos podido sacar de los tests, en el apartado informes podemos ver una lista de los informes redactados hasta la fecha. Si necesitamos añadir un informe nuevo, lo haremos haciendo click en el botón rojo de la parte de arriba. Una vez escrito no se puede modificar, si se necesitara realizar una modificación habría que pedírselo al administrador de la aplicación. Al final de la página de alta de informe debemos seleccionar a qué test va relacionado este informe.

Una vez terminado haremos click en el botón que hay al final de la página para guardarlo. Desde ese momento el informe pasa a ser responsabilidad del administrador.

Clientes

En este apartado podemos ver una lista completa de los clientes que tiene almacenados la aplicación, el usuario no no puede modificar la ficha de un cliente, se supone que la tarea del usuario es simplemente lanzar los test, visualizar el resultado y redactar el informe. El usuario sectesting, en cambio, sí podrá ver los detalles el cliente incluyendo las observaciones que el usuario administrador haya puesto para él.

Pruebas de ejecución

En este caso las pruebas de ejecución las he realizado en mi propio ordenador portátil, sus características son:

Procesador: i5

Memoria ram: 12gb

Disco Duro 1TB sata3

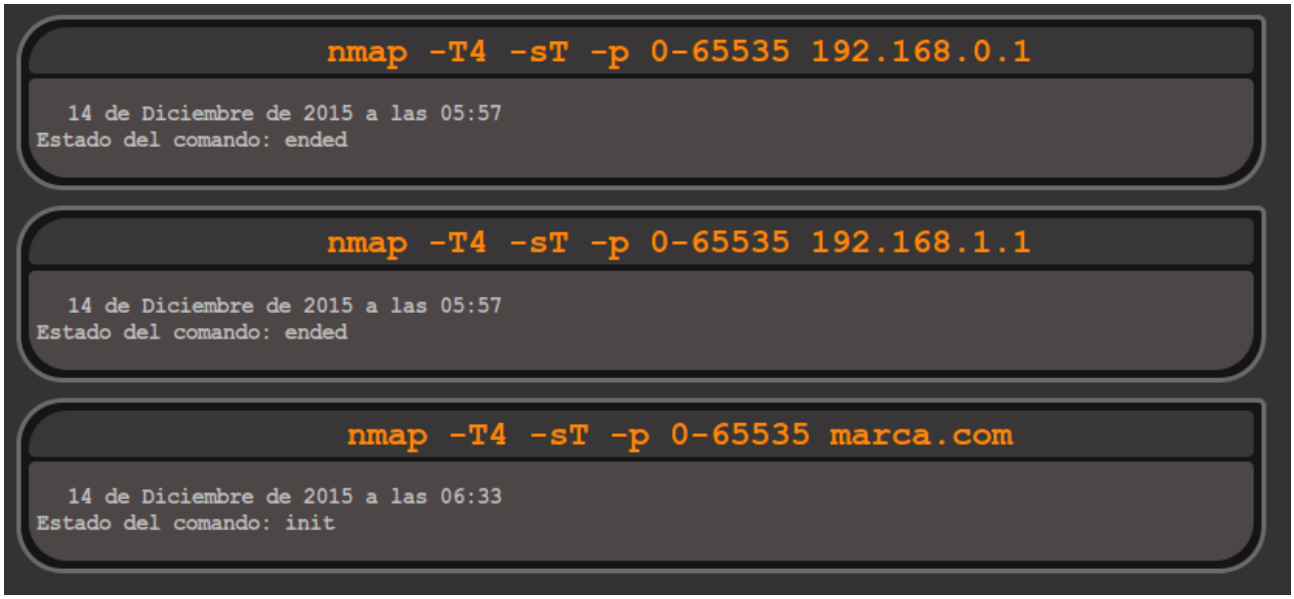
Las pruebas serán realizadas en localhost para evitar problemas de red, aunque se ha probado en varios VPS y raspberry pi y funciona sin problemas. Esta aplicación pasó finalmente su prueba de fuego incluyéndose en el CTF de SecAdmin 2015, la conferencia de seguridad informática de Sevilla.

Comandos en red local

Primero realizamos la prueba de un **escaneo TCP SYN a rango completo** a un **host de la red local**. Así mas adelante podemos ver las diferencias en tiempo de ejecución de un host en red local a uno por internet con el mismo tipo de escaneo. Comprobamos que **tarda 126 segundos en ejecutarse** y guarda correctamente su salida en la base de datos. Mientras se estaba ejecutando mostraba correctamente su flag "init" y una vez ha concluido ha pasado a "ended" pudiendo hacer click para visualizarlo a continuación.

Comandos hacia internet

Ahora nos disponemos a escanear un host de internet de la misma forma que hemos escaneao un host de nuestra intranet. Una vez lo ejecutamos comprobamos que cambia su flag a init, esperamos y le damos a f5 de vez en cuando para actualizar la página.



A la hora de realizar un escaneo de puertos TCP a través de internet comprobamos que el tiempo aumenta considerablemente, pasando a ser de **234.31 segundos**, aun así la respuesta de los comandos es bastante rápida debido al flag -T4 que le hemos añadido al comando. Además mientras se ejecutaba el comando hemos podido ejecutar con éxito un whois a otra página web.

Comando Nmap Rápido hacia Internet

Este comando está pensado para tener información en poco tiempo de un host en internet, ahora vamos a comprobar si realmente lo conseguimos a través de la aplicación. Primero lanzamos una prueba con la shell y vemos cuanto tarda:

```
Starting Nmap 6.40 ( http://nmap.org ) at 2015-12-14 07:52 CET
Nmap scan report for conectabell.com (217.160.132.34)
Host is up (0.083s latency).
rDNS record for 217.160.132.34: s18246108.onlinehome-server.info
Not shown: 99 filtered ports
PORT   STATE SERVICE
80/tcp open  http
Nmap done: 1 IP address (1 host up) scanned in 5.08 seconds
```

Lanzamos nmap a conectabell.com, lo asociamos a raw test. Al finalizar podemos comprobar que el esceaneo ha tardado 5.16 segundos tarda prácticamente lo mismo que desde la shell. Adjunto captura del resultado.

Conclusión

Haciendo inventario de lo que lleva el proyecto y pensando en futuras modificaciones llego a la conclusión de:

- Este es un proyecto de largo plazo, las ventajas de programar en esta plataforma permiten una fácil escalabilidad y un gran rendimiento.

- Es un proyecto funcional y usable en el mundo real, teniendo multitud de aplicaciones para administradores de sistemas y pentesters.

- Según las pruebas de ejecución realizadas es una aplicación rápida y eficiente para lanzar comandos ya que tarda lo mism que ejecutándolo desde shell.

- Gracias a la programación en Python podemos implementar librerías con nuevas funcionalidades de tal forma que en esta ocasión hemos implementado hilos, pero se podrían implementar más adelante colas automáticas para los hilos por ejemplo.

- He aprendido y profundizado en el diseño, codificación y maquetación de un sitio web en Django, gracias a esto puedo implementar un sitio web seguro con panel de administración en pocas semanas programando un proyecto desde 0, a la hora de realizar modificaciones en plantilla o en el motor el tiempo de codificación es menor debido a la simpleza del lenguaje de programación.

- He logrado de poner en práctica muchos de los conocimientos adquiridos en el grado superior como por ejemplo poder interactuar con controladores en bases de datos.

Como conclusión final, pienso que es un proyecto que exige el entendimiento de lenguajes de programación, sistemas y bases de datos, Dependiendo del soporte y actualización que se le dé de aquí en adelante podría conmvertirlo en una herramienta usable y necesaria para algún ámbito que no se quiera webmin.

Bibliografía o fuentes documentales de Internet

Djangogirls http://djangogirls.org

Django Oficial https://www.djangoproject.com/

Github https://github.com/

Gitlab https://gitlab.com

Blog de ElBinario http://elbinario.net

Doc oficial de Python https://www.python.org/doc/

Documento resumen

Sectesting es una aplicación especialmente destinada a lanzar comandos predefinidos desde un panel web, está pensado para el uso de administradores de red o auditores de sistemas, dichos comandos generan una salida que se guardará en la base de datos del proyecto, posteriormente se podrá consultar los comandos lanzados a un cliente en concreto con el objetivo de realizar el informe de auditoría con las vulnerabilidades encontradas.

El proyecto se ha realizado fundamentalmente con lenguaje de programación Python en un framework especialmente destinado a la implementación de aplicaciones web (Django). Consta de varias partes básicas, las más importantes son los modelos, las vistas y las plantillas: los modelos van destinados al almacenamiento de datos, dando forma de clases a las entidades guardadas en la base de datos, las vista son el motor de las plantillas, cada plantilla tiene asociada una vista que se encarga de procesar la información que le va a devolver a la plantilla.

Esta aplicación consta de tres partes, un blog, la parte de administración y la aplicación para lanzar comandos. La aplicación para lanzar comandos es muy simple, tan sólo tenemos que seleccionar el comando y lanzarlo contra el host que le indiquemos, por ahora los comandos implementados son **ping**, **whois** y **nmap** pudiendo ampliarlo rápidamente con pocas lineas de código. A la hora de lanzar un ataque debemos asociarlo a un test que se esté realizando a un cliente. Después el auditor podrá redactar un documento con las vulnerabilidades encontradas en base a los test realizados.
Para hacer un poco más segura la aplicación se han implementado sesiones y control de usuario y contraseña. Los usuarios podrán ser administradores o no tener acceso al panel dependiendo del tipo de privilegio que tengan.

URL de administración: http://127.0.0.1/admin/

URL de acceso: http://127.0.0.1/sectesting/

Credenciales
admin: antonov antonov
user: sectesting sectesting