Manual de Despliegue

Índice:

1. Introducción

En este manual se detallará como **desplegar la aplicación** PGDI para gestionar el alta y mantenimiento de los alumnos matriculados en cursos del departamento de informática de un centro de enseñanza superior.

Funcionalidades

La aplicación permitirá a los **alumnos**:

- Registrarse en el curso matriculado
- Elegir su nombre de usuario (si el tutor del curso así lo desea)
- Cambiar su contraseña, mediante autorización/validación previa.
- Consultar su cuota de espacio en disco.
- Cambiar determinados datos de su perfil.

La aplicación permitirá a los **profesores**:

- Confirmar/validar el alta de los alumnos registrados.
- Confirmar/dar de baja a los alumnos de la aplicación.
- Consultar/modificar determinados datos del alumno.

Requisitos previos

Para el correcto despliegue de esta aplicación será necesario cumplir con los siguientes **requisitos de software**:

Requisito necesario:	***En este manual:***
Sistema Operativo Linux	Ubuntu (16.04)
Servidor Web (+ Django)	Apache (+ Django)
Backend Base de datos relacional	Postgres
Backend LDAP	OpenLDAP

No se necesitan **requisitos de hardware** especiales.

Tanto si se opta por desplegar las diferentes tecnologías necesarias en diferentes hosts (algo habitual y recomendado) como si no, se debe cumplir con los **requisitos de seguridad** necesarios para garantizar una comunicación segura entre los distintos equipos que interactúen con la aplicación.

Para cumplir con estos requisitos se recomienda, por ejemplo, el uso de **certificados** y **protocolos seguros** como los usados en este manual.

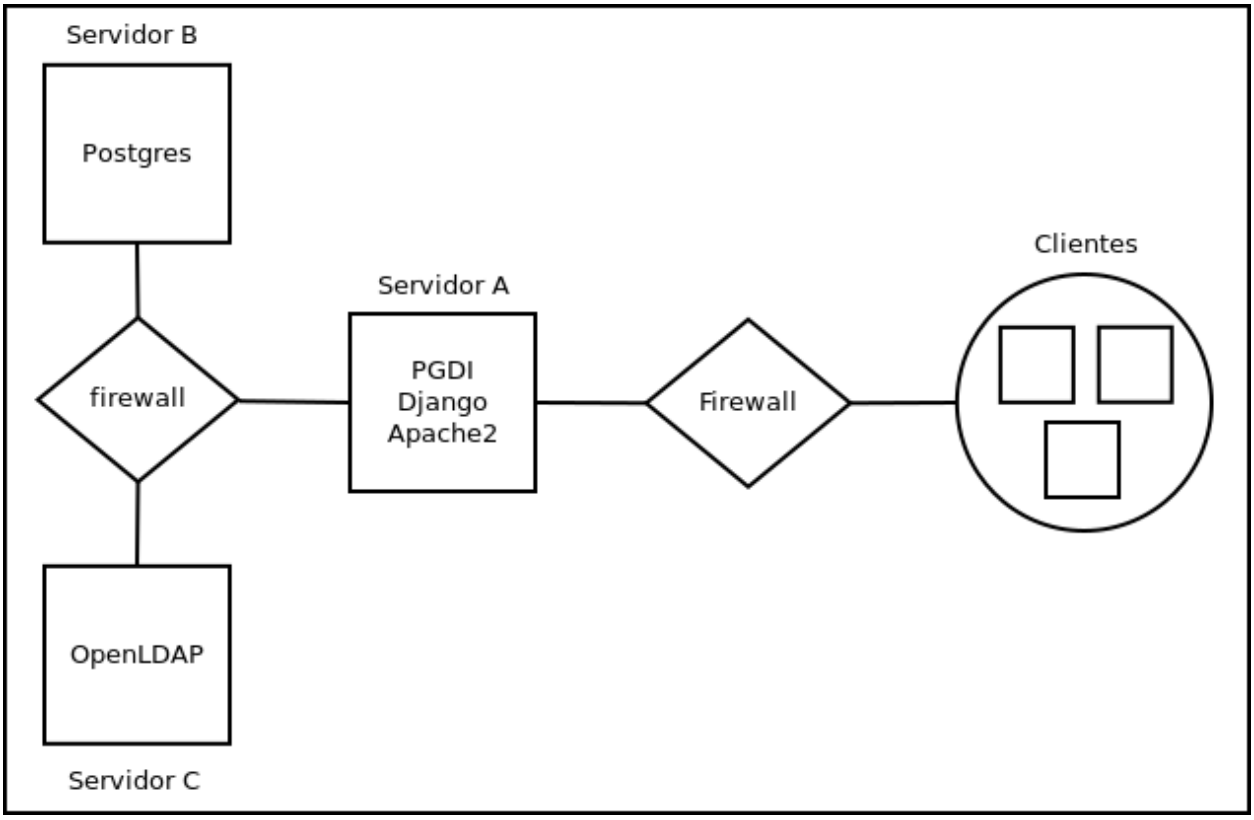
Para incrementar aun más la seguridad de la aplicación, se puede integrar la misma con tecnologías avanzadas como **MIT Kerberos**.

Posible escenario

A la hora de desplegar la aplicación, existen diferentes **posibilidades** en cuanto al numero de hosts y su distribución. Si bien es posible instalar todos los componentes en un único host, es **recomendable** separar los backends (Postgres y OpenLDAP) del servidor web con la aplicación.

Este tipo de distribución nos permitiría, entre otras ventajas, poder establecer reglas de **cortafuegos** mas permisivas en la red que comunica la aplicación con los clientes como si de una DMZ[1] se tratase.

A continuación se muestra un **ejemplo** de un escenario típico:



Clonado de la aplicación

La manera más sencilla de conseguir el **código de la aplicación** es utilizar git para descargar la aplicación desde su repositorio:

```
$ git clone git://github.com/Xpnosa/ProyectoGDI
```

Si no tenemos **git**, podemos instalarlo fácilmente:

```
# apt-get install git
```

---

(1) *Zona desmilitarizada. Zona segura que se ubica entre una red interna y una red externa.*

Django framework

El primer paso para comenzar con el despliegue de la aplicación PGDI es contar con el **framework django**, escrito en **python**.

La instalación de django + python ha de realizarse en el host donde estará alojado el código de la aplicación, junto con el servidor web **apache**.

Para la instalación de todos los **componentes necesarios** para hacer funcionar django necesitaremos un usuario con privilegios de administrador.

Comenzamos instalando **python**[2]:

```
# apt install python
```

También es recomendable instalar el interprete **ipython** (opcional):

```
# apt install ipython
```

A continuación instalamos **django**[3] con pip:

```
# pip install django
```

Por ultimo, instalamos la extensiones **django_extensions** y **django-suit**:

```
# pip install django-extensions
# pip install django-suit
```

Si no tenemos **pip**, podemos instalarlo con easy_install:

```
# easy_install pip
```

Otras dependencias

Aprovechamos para instalar los paquetes necesarios para la integración con **Postgres**:

```
# apt-get install python-dev libpq-dev
# pip install psycopg2
```

Y también los necesarios para la integración con **OpenLDAP**:

```
# apt-get install django-auth-ldap python-ldap
# apt-get install libssl-dev libsasl2-dev libldap2-dev
```

---

(2) *La versión de python usada en este manual es la versión 2.7*
(3) *La versión de django usada en este manual es la versión 1.10*

2. Apache

Apache es un **servidor web HTTP** de código abierto, para plataformas Unix, Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.

Apache será la solución elegida en este manual para la publicación de la aplicación PGDI.

Instalación

Ahora que ya tenemos correctamente instalado django, es el momento de instalar el servidor web **apache** en el host.

Para la instalación del **servidor web** apache necesitaremos un usuario con privilegios de administrador.

Comenzamos instalando el paquete **apache2**:

```
# apt-get install apache2
```

También necesitaremos el siguiente modulo **wsgi**[4]:

```
# apt install libapache2-mod-wsgi
```

En este punto, ya podemos **copiar** o **mover** el código de aplicación PGDI descargado en el paso anterior a una **ruta adecuada**, por ejemplo sobre "/var/www/html"

Certificación SSL

Para poder habilitar SSL en Apache es indispensable contar con los **certificados** que validarán la autenticidad del servidor. Así pues, puesto que mi caso no cuento con una **autoridad certificadora**, el primer paso será generar un **certificado auto-firmado.**

Primero, generamos una **clave** para el servidor:

```
$ openssl genrsa -out privkey.pem 4096
```

Y con esta clave, creamos el **certificado auto-firmado**:

```
$ openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095
```

Ahora guardamos los certificados en un **lugar seguro**, por ejemplo en "/etc/certs"

---

(4) *Especificación para interfaz simple y universal entre servidores web y aplicaciones web o frameworks para el lenguaje de programación Python*

Configuración del sitio

Pasamos ahora a configurar el **sitio virtual** para la aplicación. Para ello nos dirigimos al fichero "/etc/apache2/sites-enabled/000-default.conf" y lo editamos para añadir nuestro sitio:

```
<VirtualHost *:443>
        ServerName pgdi:443 ← A

        ServerAdmin admin@pgdi ← B
        DocumentRoot /var/www/html/pgdi ← C

        SSLEngine On
        SSLCertificateFile /etc/certs/cacert.pem ← D
        SSLCertificateKeyFile /etc/certs/privkey.pem ←  E

        WSGIScriptAlias /var/www/html/pgdi/wsgi.py ← F

        <Directory "/var/www/html/pgdi"> ← C
                <Files wsgi.py>
                        Require all granted
                </Files>
        </Directory>

        Alias /static/ /var/www/html/pgdi/pgdiapp/static/ ← G

        <Directory /var/www/html/pgdi/pgdiapp/static> ← G
                Require all granted
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log ← H
        CustomLog ${APACHE_LOG_DIR}/access.log combined ← H
</VirtualHost>
```

- **A**: Nombre del host y puerto en el que apache escuchará las peticiones HTTPS.
- **B**: Correo electrónico del administrador del sitio.
- **C**: Raíz de la aplicación PGDI, lugar donde se encuentra el código descargado.
- **D**: Ruta al certificado clave creado en el apartado "Certificación SSL".
- **E**: Ruta al certificado auto-firmado creado en el apartado "Certificación SSL".
- **F**: Ruta donde se encuentra el script wsgi.py de la aplicación PGDI.
- **G**: Alias para el directorio static de la aplicación PGDI.
- **H**: Rutas a los ficheros de log de apache.

Finalizando

Para terminar de configurar correctamente nuestro sitio con apache nos dirigimos al fichero "/etc/apache2/apache2.conf" y lo editamos para **añadir** la siguiente linea, la cual indica la ruta al **directorio raíz** de la aplicación PGDI:

```
WSGIPythonPath /home/espinosa/django/PASIR/pgdi
```

También nos aseguraremos de que apache **escuchará** en el puerto elegido (443) cuando **habilitemos el modulo SSL**.

Para ello, nos dirigimos al fichero "/etc/apache2/ports.conf" y, **si hiciese falta**, lo editamos para **añadir** el siguiente bloque para abrir el puerto elegido:

```
<IfModule ssl_module>
        Listen 443
</IfModule>
```

Ahora sólo tenemos que habilitar el módulos **wsgi** de apache:

```
# a2enmod wsgi
```

Y también el módulo **ssl** de apache:

```
# a2enmod ssl
```

Por último, reiniciamos el servicio de apache para aplicar todos los cambios:

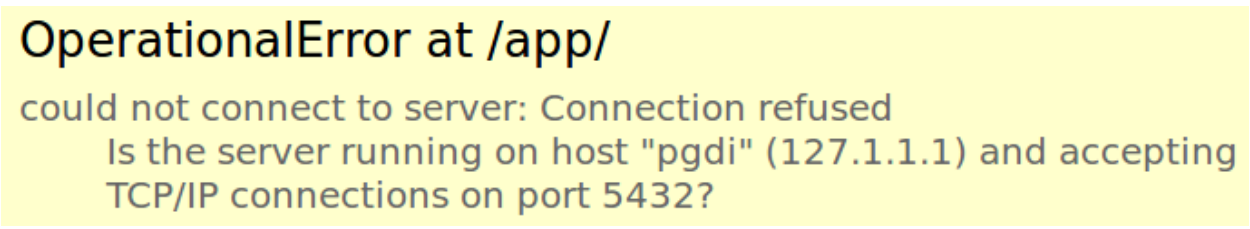
```
# systemctl restart apache2.service
```

Si todo los datos son correctos, el servicio debería **reiniciarse sin errores**.

Verificación

Por desgracia, aun es pronto para realizar **pruebas concluyentes**, pues no podremos acceder a la aplicación PGDI hasta haber configurado, al menos, el backend de **base de datos relacional**. (Postgres)

Sin embargo, si tratamos de acceder a la aplicación desde el **navegador** [https://pgdi] posiblemente veríamos (tras la aceptación del certificado auto-firmado) una pantalla de **depuración** similar a esta:



3. Postgres

PostgreSQL es un Sistema de gestión de bases de datos relacional orientado a objetos y libre, publicado bajo la licencia PostgreSQL, similar a la BSD o la MIT.

Instalación

Para este manual, se ha optado por utilizar el **contenedor Docker**[5] oficial para Postgres. Por supuesto, también podemos optar por instalar Postgres de forma tradicional desde el **repositorio oficial** para ubuntu:

```
# apt-get install postgresql postgresql-contrib
```

Para **iniciar el contenedor**, utilizamos el siguiente comando:

```
$ docker run --name postgreSQL -e POSTGRES_PASSWORD=<contraseña> \
  -p 5432:5432 -d postgres
```

Configuración

Con el contenedor iniciado, **accedemos** a el mismo con se siguiente comando:

```
$ docker exec -it postgreSQL /bin/bash
```

Una vez dentro seremos **root**, pero nos interesa cambiar al usuario **postgres**:

```
# su postgres
```

Ahora, usamos el cliente **psql** para crear y preparar la **base de datos** que usará la aplicación PGDI:

```
$ psql
```

```
CREATE DATABASE pgdi; ← A
CREATE USER pgdiuser WITH PASSWORD 'pgdipass'; ← B, C

ALTER ROLE pgdiuser SET client_encoding TO 'utf8';
ALTER ROLE pgdiuser SET default_transaction_isolation TO 'read committed';
ALTER ROLE pgdiuser SET timezone TO 'CET';

GRANT ALL PRIVILEGES ON DATABASE pgdi TO pgdiuser;
```

---

(5) *Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Linux.*

Integración con django

En este punto ya tenemos listo el backend Postgres, sólo nos queda **configurar la comunicación** de este con la aplicación PGDI.

Para ello, nos dirigimos al fichero *pgdi/settings.py* de la aplicación y editamos la opción "DATABASES" con los datos de nuestra **base de datos** Postgres:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'pgdi', ← A
        'USER': 'pgdiuser', ← B
        'PASSWORD': 'pgdipass', ← C
        'HOST': 'pgdi', ← D
        'PORT': '5432', ← E
    }
}
```

- **A**: Nombre de la base de datos para la aplicación.
- **B**: Usuario administrador (con todos los privilegios) en la base de datos.
- **C**: Contraseña de la cuenta del usuario administrador en la base de datos.
- **D**: Nombre del host donde se encuentra instalado Postgres.
- **E**: Puerto por el que escuchara las conexiones Postgres.

Finalizando

Para terminar con este apartado, volvemos al host que hospeda la aplicación web (donde tenemos instalado **django + apache**), para crear y poblar las diferentes tablas que componen el modelo.

Antes de empezar, nos situamos el el **directorio raíz** de la aplicación:

```
$ cd /var/www/html/pgdi
```

Primero migramos el **modelo** a Postgres:

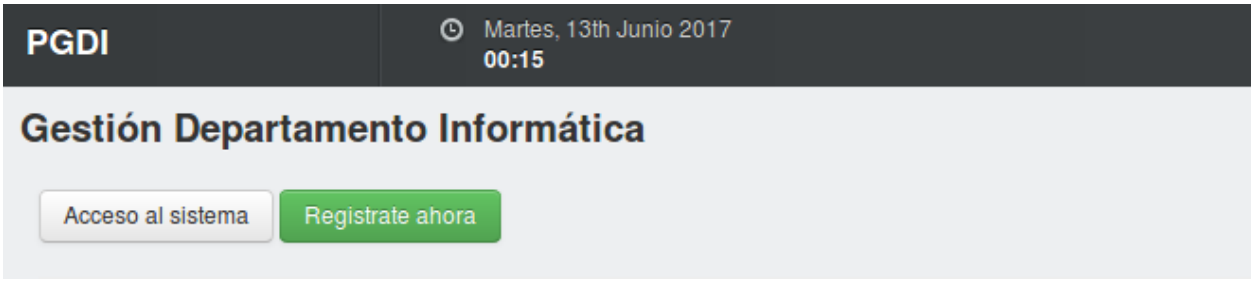
```
$ python manage.py makemigrations
$ python manage.py migrate
```

Y a continuación **poblamos** la base de datos:

```
$ python manage.py setup
```

Verificación

En este punto, y si hemos **configurado todo correctamente**, deberíamos ser capaces de poder ver (tras la aceptación del certificado auto-firmado) la **página de el inicio** de la aplicación si accedemos a ella desde el **navegador** [https://pgdi]



Sin embargo, aun no seremos capaces de **autenticarnos** ni **registrarnos**, pues nos falta un último componente, el backend LDAP.

Ficheros

pgdiapp/management/commands/setup.py

```
#-*- coding: utf-8 -*-

from django.core.management.base import BaseCommand, CommandError

from pgdiapp.models import Configuracion, Mensaje, Tipo

class Command(BaseCommand):
    help = 'Script que se debe ejecutar para completar la instalación de

    def handle(self, *args, **options):
        try:
            # Poblando Configuraciones
            Configuracion(clave='openregister',info='Si se establece a T
            Configuracion(clave='freeusername',info='Si se establece a T
            # Poblando Mensajes
            Mensaje(cod='WELCOME',asunto='Bienvenid@',cuerpo='Hola {{USU
            Mensaje(cod='BYEBYE',asunto='Aviso de baja',cuerpo='Tu usuar
            # Poblando Tipos
            Tipo(clase='error').save()
            Tipo(clase='info').save()
            Tipo(clase='success').save()
            Tipo(clase='warn').save()
            print "\nLa base de datos se pobló correctamente.\n"
        except:
            print "\nOcurrió un error al poblar la base de datos.\n"
            raise
```

El fichero *pgdiapp/management/commands/setup.py* mostrado justo arriba contiene el código de un **comando personalizado** de aplicación django.

Dicho comando se encarga de **poblar las tablas** “Configuración”, “Mensaje” y “Tipo” con los **registros necesarios** para el correcto funcionamiento de la aplicación PGDI.

4. OpenLDAP

OpenLDAP es una implementación libre, de código abierto y multiplataforma del protocolo Lightweight Directory Access Protocol (LDAP) desarrollada por el proyecto OpenLDAP bajo su propia licencia OpenLDAP Public License.

Instalación

Para este manual, se ha optado por utilizar el **contenedor Docker** para OpenLDAP. Por supuesto, también podemos optar por instalar OpenLDAP de forma tradicional desde el **repositorio oficial** para ubuntu:

```
# apt-get install slapd ldap-utils
```

Para **iniciar el contenedor**, utilizamos el siguiente comando:

```
$ docker run -v /data/slapd:/var/lib/ldap -e LDAP_DOMAIN=pgdi.inf \
  -e LDAP_ORGANISATION="PGDI" -e LDAP_ROOTPASS=toor \
  -p 389:389 -d --name slapd nickstenning/slapd
```

En *LDAP_DOMAIN* establecemos el **dominio** del árbol.
En *LDAP_ORGANISATION* establecemos el **nombre** de nuestra organización
En *LDAP_ROOTPASS* establecemos la **contraseña** del administrador del árbol (admin)

Configuración

Con el contenedor iniciado, **accedemos** a el mismo con se siguiente comando:

```
$ docker exec -it slapd /bin/bash
```

Una vez dentro seremos **root** y podremos gestionar el árbol con **comandos** como *ldapadd* (o podemos utilizar un **cliente** LDAP como *Apache Directory Studio*):

```
ldapadd -h localhost -p 389 -c -x -D cn=admin,dc=pgdi,dc=inf -W -f <file.ldif>
```

Como no sabemos cual es el usuario/contraseña del usuario **administrador** de LDAP (sólo conocemos el de nuestro árbol) vamos a establecerlo directamente en el fichero */etc/ldap/slapd.d/cn=config/olcDatabase={0}config.ldif*:

```
olcRootDN: cn=admin,cn=config
olcRootPW: <contraseña>
```

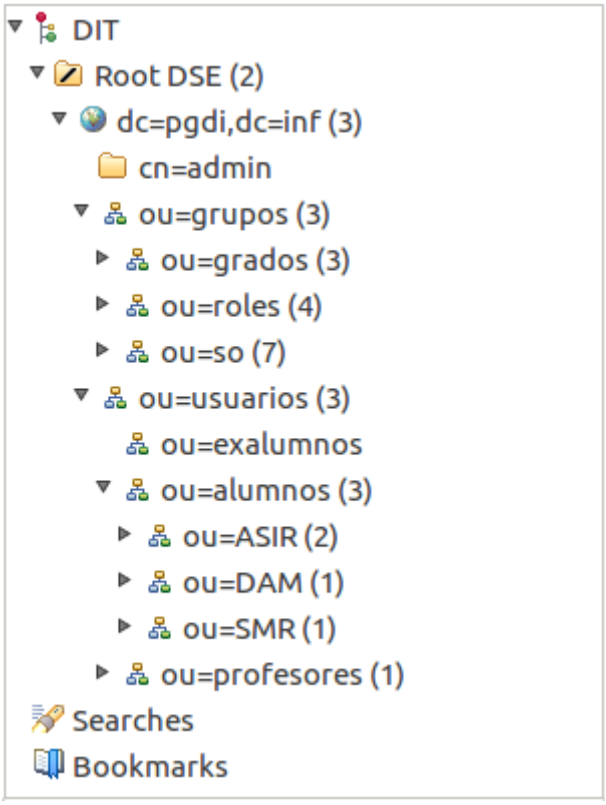
Reiniciamos el contenedor y ya podremos gestionar el árbol completamente con **comandos** como *ldapadd*:

```
ldapadd -h localhost -p 389 -c -x -D cn=admin,cn=config -W -f <file.ldif>
```

Frontend

En este apartado vamos a detallar el frontend del árbol que necesitamos para hacer funcionar la aplicación PGDI.

Para empezar necesitaremos crear las unidades organizativas (ou) **“usuarios”** y **“grupos “**.



Dentro de “usuarios” se encontrarán las unidades organizativas **“alumnos”, “profesores”** y **“exalumnos”**.

Dentro de “alumnos” se encontrarán las unidades organizativas de los diferentes **grados impartidos** por el departamento de informática del centro (ASIR, DAM, SMR…).

Dentro de “grupos” se encontrarán las unidades organizativas **“roles”, “so”** y **“grados”**.

En “grados” se encontrarán los grupos pertenecientes a **grados impartidos** en el departamento de informática del centro (ASIR, DAM, SMR…).

En “roles” se encontrarán los grupos **“activos”, “profesores”, “alumnos”** y **“administradores”**

En “so” se encontrarán los grupos de **sistema operativo**.

En las diferentes unidades organizativas dentro “alumnos” se encontrarán los usuarios pertenecientes a los **alumnos**.

En “exalumnos” se encontrarán los usuarios pertenecientes a los **antiguos alumnos**.

En “profesores” se encontrarán los usuarios pertenecientes a los **profesores**.

Todos los usuarios deben contener obligatoriamente los objectClass *posixAccount*, *shadowAccount, inetOrgPerson, organizationalPerson, person* y ***pgdi**** (creado en apartado backend).

Todos los usuarios deben contener obligatoriamente el objectClass posixGroup.

Con esto en cuenta podemos preparar un **fichero ldif** (ver frontend.ldif) y cargarlo en nuestro árbol LDAP con el **comando** *ldapadd*:

```
ldapadd -h localhost -p 389 -c -x -D cn=admin,dc=pgdi,dc=inf -W -f  frontend.ldif
```

Backend

En este apartado vamos a detallar las modificaciones que necesitaremos realizar en el backend de nuestro árbol para hacer funcionar la aplicación PGDI:

La aplicación PGDI necesita almacenar en LDAP los campos **fecha de nacimiento**, **DNI** e información sobre **cuotas de disco**. Puesto que no existen esquemas estandarizados que los incluyan, vamos a crear nuestro propio esquema (pgdi) con un **objectClass** que incluya los **atributos** que necesitaremos.

Para ello, vamos a preparar un **fichero ldif** con la siguiente estructura:

```
dn: cn={4}pgdi,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: {4}pgdi
olcAttributeTypes: <atributo para fecha de nacimiento>
olcAttributeTypes: <atributo para DNI>
olcAttributeTypes: <atributo para cuotas>
olcObjectClasses: <object class>
```

Para el atributo **fecha de nacimiento**, utilizaremos el siguiente **valor**:

```
{0}( 2.25.0001 NAME 'fnac' DESC 'Fecha de nacimiento' EQUALITY
generalizedTimeMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 SINGLE-VALUE )
```

Para el atributo **DNI**, utilizaremos el siguiente **valor**:

```
{1}( 2.25.0002 NAME 'dni' DESC 'Documento Nacional de Identidad' EQUALITY
caseIgnoreMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
```

Para el atributo con información sobre **cuotas**, utilizaremos el siguiente **valor**:

```
{2}( 2.25.0003 NAME 'quota'
DESC 'Quotas (FileSystem:BlocksSoft,BlocksHard,InodesSoft,InodesHard)'
EQUALITY caseIgnoreIA5Match SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{255} )
```

Y para el objectClass **pgdi**, utilizaremos el siguiente **valor**:

```
{0}( 2.25.0000 NAME 'pgdi' AUXILIARY MAY (fnac $ dni $ quota) )
```

Ahora que ya tenemos listo el **fichero ldif**, lo cargamos con el **comando** *ldapadd*:

```
ldapadd -h localhost -p 389 -c -x -D cn=admin,cn=config -W -f  pgdi.ldif
```

NAME: Nombre del atributo.
DESC: Descripción del atributo.
EQUALITY: Regla que dicta cómo se comportaran los valores en un filtro de búsqueda
SYNTAX: OID que define el tipo de datos y qué reglas (validaciones) se aplican a los mismos.
SINGLE-VALUE: Indica que el atributo sólo puede aparecer una vez en cualquier objectClass.
AUXILIARY: Complementa los atributos de un objeto que ya tiene un objectClass de tipo structural.
MAY: Define los atributos como opcionales.

Integración con django

En este punto ya tenemos casi listo el árbol LDAP, sólo nos queda **configurar la comunicación** de este con la aplicación PGDI.

Para ello, nos dirigimos al fichero *pgdi/settings.py* de la aplicación y **modificamos** o **verificamos** las siguientes opciones:

```
AUTHENTICATION_BACKENDS = (
    'django_auth_ldap.backend.LDAPBackend', # Backend de authenticación
)
```

```
AUTH_LDAP_SERVER_URI = "ldap://pgdi:389" # Protocolo, host y puerto LDAP
AUTH_LDAP_BIND_DN = "cn=admin,dc=pgdi,dc=inf" # Administrador del árbol
AUTH_LDAP_BIND_PASSWORD = "toor" # Contraseña del administrador
```

```
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=usuarios,dc=pgdi,dc=inf",
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)") # Acceso a los usuarios
AUTH_LDAP_GROUP_SEARCH = LDAPSearch("ou=grupos,dc=pgdi,dc=inf",
    ldap.SCOPE_SUBTREE, "(objectClass=PosixGroup)") # Acceso a los grupos
```

```
# Mapeado de los usuarios LDAP con los atributos del modelo User
AUTH_LDAP_USER_ATTR_MAP = {
    "first_name": "givenName",
    "last_name": "sn",
    "email": "mail"
}
AUTH_LDAP_USER_FLAGS_BY_GROUP = {
    "is_active": "cn=activos,ou=roles,ou=grupos,dc=pgdi,dc=inf",
    "is_staff": "cn=profesores,ou=roles,ou=grupos,dc=pgdi,dc=inf",
    "is_superuser": "cn=administradores,ou=roles,ou=grupos,dc=pgdi,dc=inf"
}
```

```
# Versión del protocolo LDAP y nombre del host donde se ubica el árbol
LDAP_VERSION = ldap.VERSION3
LDAP_SERVER_NAME = "pgdi"
```

```
# Mapeado de los dn del dominio del árbol y sus unidades organizativas
LDAP_DOMAIN_BASE = "dc=pgdi,dc=inf"
LDAP_USERS_BASE = "ou=usuarios," + LDAP_DOMAIN_BASE
LDAP_STUDENTS_BASE = "ou=alumnos," + LDAP_USERS_BASE
LDAP_TEACHERS_BASE = "ou=profesores," + LDAP_USERS_BASE
LDAP_EXSTUDENTS_BASE = "ou=exalumnos," + LDAP_USERS_BASE
LDAP_GROUPS_BASE = "ou=grupos," + LDAP_DOMAIN_BASE
LDAP_ROLES_BASE = "ou=roles," + LDAP_GROUPS_BASE
LDAP_GRADES_BASE = "ou=grados," + LDAP_GROUPS_BASE
LDAP_OS_BASE = "ou=so," + LDAP_GROUPS_BASE
```

Y con esto habríamos **terminado** de configurar la **integración** de LDAP con django.

Finalizando

Para terminar con este apartado, vamos a crear un usuario **administrador de la aplicación** (profesor) en nuestro árbol LDAP.

Dicho usuario debe estar ubicado bajo la unidad organizativa (ou) "**profesores**" y debe pertenecer a los grupos "**administradores**", "**activos**", "**profesores**" y los de los **grados que imparta** (bajo la unidad organizativa grados).

Para ello, vamos a preparar un **fichero ldif** (ver superuser.ldif) con la siguiente estructura:

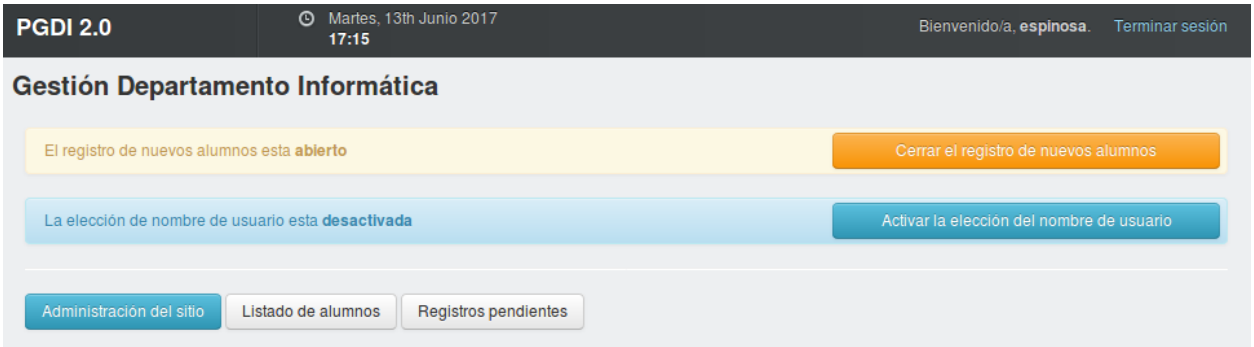
```
dn: cn=<nombre de usuario>,ou=profesores,ou=usuarios,dc=pgdi,dc=inf
changetype: add
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
mail: <correo electronico>
telephoneNumber: <número de teléfono>
givenName: <nombre>
sn: <apellidos>
cn: <nombre de usuario>
uid: <nombre de usuario>
uidNumber: <número uid en el so>
gidNumber: <número gid en el so>
homeDirectory: <directorio de casa>
loginShell: <shell de entrada por defecto>
userPassword: <contraseña del usuario>

dn: cn=<grado impartido>,ou=grados,ou=grupos,dc=pgdi,dc=inf
changetype: modify
add: memberUid
memberUid: <nombre de usuario>

dn: cn=activos,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: modify
add: memberUid
memberUid: <nombre de usuario>

dn: cn=administradores,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: modify
add: memberUid
memberUid: <nombre de usuario>

dn: cn=profesores,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: modify
add: memberUid
memberUid: <nombre de usuario>
```

Una vez listo, cargamos el **fichero ldif** en el árbol con el **comando** *ldapadd*:

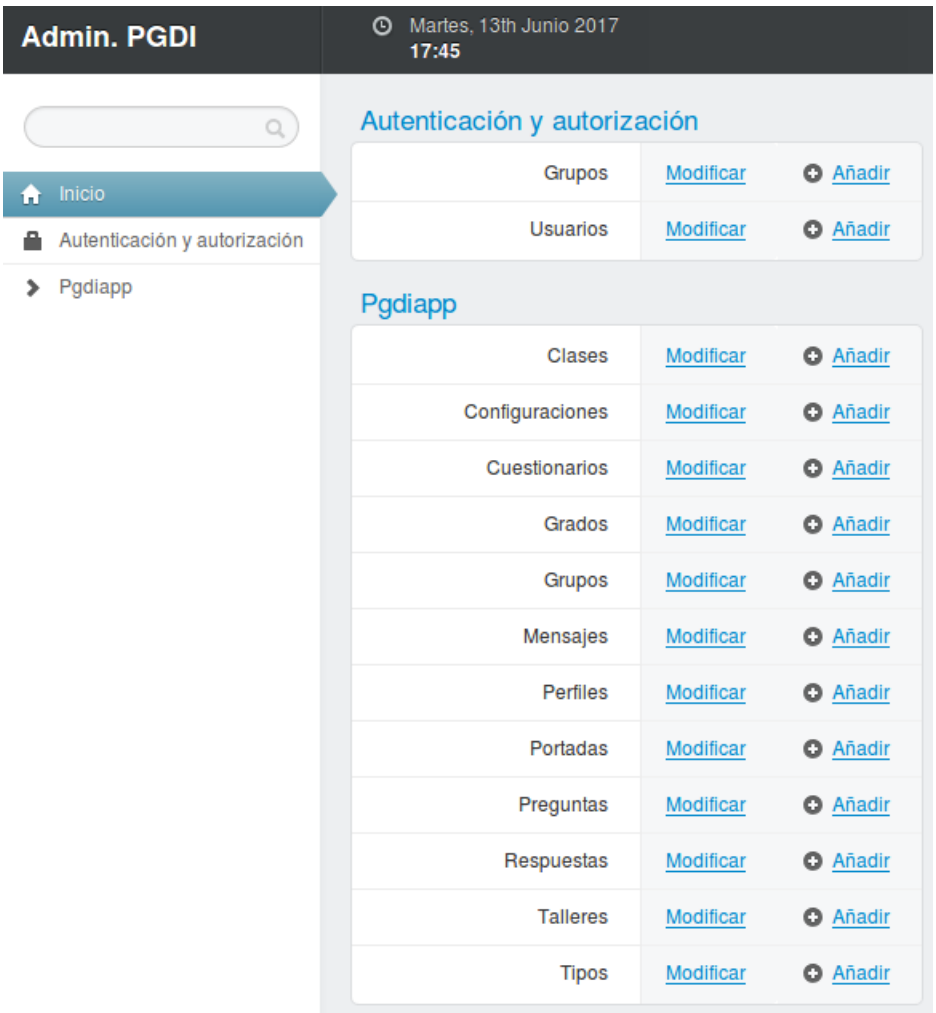
```
ldapadd -h localhost -p 389 -c -x -D cn=admin,dc=pgdi,dc=inf -W -f  superuser.ldif
```

Verificación

Esta vez, si accedemos a la aplicación desde el **navegador** [https://pgdi] deberíamos ser capaces de acceder a la aplicación con el usuario **administrador** que acabamos de crear:



También deberíamos poder acceder al **panel de administración**:



En realidad, en estos momentos la aplicación debería de **funcionar al 90%** (registros, altas, bajas, perfiles,…), solo nos quedaría por configurar la conexión con el **servidor NFS** para la consulta de cuotas y ficheros, el **servidor SMTP** para el envió de mensajes y otros pocos parámetros más que veremos en el siguiente punto (**5. Finalizando**).

Ficheros

frontend.ldif

```
#! Unidades Organizativas: Usuarios
dn: ou=usuarios,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: usuarios

dn: ou=alumnos,ou=usuarios,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: alumnos

dn: ou=exalumnos,ou=usuarios,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: exalumnos

dn: ou=profesores,ou=usuarios,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: profesores

#! Unidades Organizativas: Grupos
dn: ou=grupos,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: grupos

dn: ou=grados,ou=grupos,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: grados

dn: ou=roles,ou=grupos,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: roles

dn: ou=so,ou=grupos,dc=pgdi,dc=inf
objectClass: organizationalUnit
ou: so

#! Grupos de usuarios: Roles
dn: cn=activos,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: add
gidNumber: 4001
objectClass: posixGroup
objectClass: top
cn: activos

dn: cn=administradores,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: add
gidNumber: 4002
objectClass: posixGroup
objectClass: top
cn: administradores

dn: cn=profesores,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: add
gidNumber: 4003
objectClass: posixGroup
objectClass: top
cn: profesores

dn: cn=alumnos,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: add
gidNumber: 4004
objectClass: posixGroup
objectClass: top
cn: alumnos
```

superuser.ldif

```
dn: cn=espinosa,ou=profesores,ou=usuarios,dc=pgdi,dc=inf
changetype: add
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
mail: gm.mizuki@gmail.com
telephoneNumber: 679939948
givenName: Juan Antonio
sn: Espinosa
cn: espinosa
uid: espinosa
uidNumber: 10000
gidNumber: 10000
homeDirectory: /home/pub/espinosa
loginShell: /bin/bash
userPassword: {SHA}fNCy7S4AQyUV1pzrtB87f82wpCI=

dn: cn=activos,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: modify
add: memberUid
memberUid: espinosa

dn: cn=administradores,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: modify
add: memberUid
memberUid: espinosa

dn: cn=profesores,ou=roles,ou=grupos,dc=pgdi,dc=inf
changetype: modify
add: memberUid
memberUid: espinosa
```

5. Finalizando

¡Ya casi hemos terminado con el despliegue de la aplicación PGDI! Tan sólo nos quedan unos pocos pasos más...

NFS (SSH)

Es habitual que tengamos un servidor **NFS**(6) para proporcionar un a nuestros usuarios espacio de disco.

Si este es el caso y queremos que estos usuarios puedan consultar su **espacio** ocupado y disponible **en disco**, así como sus **ficheros más pesados**, deberemos configurar una conexión **SSH**(7) entre el host HTTP y el host NFS.

Sera necesario disponer de un **cliente SSH** en el host HTTP.

```
# apt-get install openssh-client
```

Así como contar con un **servidor SSH** en el host NFS.

```
# apt-get install openssh-server
```

Creamos un **par de claves** en el host cliente SSH (como root o usuario privilegiado):

```
# ssh-keygen -b 4096 -t rsa ← algoritmo de cifrado rsa, longitud 4096 bits
```

Esta clave será usada para ejecutar el comando ***repquota*** en el servidor SSH (host NFS), le ponemos un nombre adecuado. Por ejemplo: "id_rsa_repquota".

Creamos un nuevo par de claves con el comando anterior, esta vez para ejecutar el comando ***du*** en el servidor SSH. La nombramos, por ejemplo: "id_rsa_du".

Copiamos ambas **claves privadas** (id_rsa_repquota y id_rsa_du) a un directorio **accesible unicamente** por el usuario **www-data** (podemos crearle un directorio .ssh):

```
# cp  id_rsa_repquota /var/www/.ssh/
# cp  id_rsa_du /var/www/.ssh/
```

Establecemos al usuario/grupo www-data como **propietario** de las claves y nos aseguramos de que sólo tengan **permiso de lectura** para este usuario:

```
# chown www-data:www-data  id_rsa_du  id_rsa_repquota
# chmod 400  id_rsa_du  id_rsa_repquota
```

---

(6) Protocolo de capa de aplicación utilizado en sistemas de ficheros distribuido en red. Posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si de ficheros locales se tratasen.

(7) *Protocolo que sirve para acceder servidores privados a través de una puerta trasera.*

Nos dirigimos ahora al **host NFS** (servidor SSH) y creamos un usuario al que le daremos el **privilegio de ejecutar** los comandos r*epquota* y *du* **sin restricciones**:

```
# useradd <usuario> -m -s /bin/bash
```

Modificamos el fichero "/etc/sudoers" (¡con visudo!) para añadir al nuevo usuario:

```
# visudo
```

```
<usuario> ALL=(root) NOPASSWD: /usr/sbin/repquota
<usuario> ALL=(root) NOPASSWD: /usr/bin/du
```

Ahora copiamos ambas **claves publicas** (id_rsa_repquota.pub y id_rsa_du.pub) desde el **cliente SSH** al un directorio **accesible unicamente** por el **usuario privilegiado** que hemos creado y que ejecutara los comandos *repquota* y *du* en el **servidor SSH**.

Hacemos **login** con este usuario y creamos el directorio ".ssh" en su directorio de casa:

```
$ mkdir ~/.ssh
```

Creamos el fichero "authorized_keys" **dentro** del recién creado directorio ".ssh":

```
$ touch ~/.ssh/authorized_keys
```

Copiamos el contenido de las **claves publicas** dentro del fichero "authorized_keys":

```
$ cat id_rsa_repquota.pub >> ~/.ssh/authorized_keys
$ cat id_rsa_du.pub >> ~/.ssh/authorized_keys
```

Esto nos dejara **dos lineas** en el fichero "authorized_keys", una por cada clave. Ahora **asignaremos** a cada clave el comando correspondiente **editando** dicho fichero. Añadiremos al principio de la **primera linea** (id_rsa_repquota.pub) lo siguiente:

```
command="sudo repquota /home -O csv" <resto de la linea...>
```

También añadimos al principio de la **segunda linea** (id_rsa_du.pub) lo siguiente:

```
command="sudo du -s $SSH_ORIGINAL_COMMAND" <resto de la linea...>
```

Para terminar, nos dirigimos al **Host HTTP** (cliente SSH) y editamos el fichero *settings.py* de la aplicación para establecer la **ruta a las claves privada** mediante las opciones *SSH_KEY_REPQUOTA* y *SSH_KEY_DU*:

```
SSH_KEY_REPQUOTA = "/var/www/.ssh/id_rsa_repquota"
SSH_KEY_DU = "/var/www/.ssh/id_rsa_du"
```

¡Y esto es todo! Ahora los usuario pueden **consultar sus cuotas** de disco en su perfil. Para saber cómo consulte el **Manual de Usuario** de esta aplicación.

SMTP

Para que la aplicación sea capaz de **enviar correos** a los alumnos tras las altas/bajas de los mismos, necesitaremos establecer unos **valores apropiados** en el fichero *pgdi/settings.py* de la aplicación:

```
SMTP_HOST = <host del servidor smtp>
SMTP_USER = <usuario del host que enviará los eMails>
SMTP_PASS = <contraseña de usuario del host que enviará los eMails>
SMTP_NAME = <alias para el remitente de los eMails>
```

La plantilla de los mensajes puede cambiarse desde el panel de administración. Para más detalles consulte el **Manual Técnico** de esta aplicación.

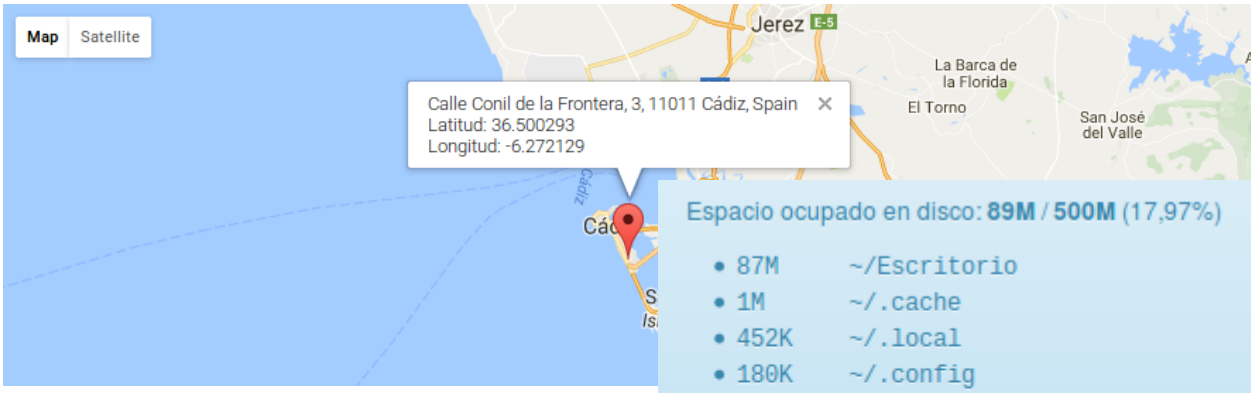
Google Maps

Si queremos usar la API de Google Maps en el **formulario de registro**, necesitaremos crear nuestra propia API key para Google Maps y establecer la misma en el fichero "pgdi/pgdiapp/templates/pgdiapp" de la aplicación:

```
...
<script src="https://maps.google.com/maps/api/js?key=<API key>"></script>
...
```

Verificación

¡Ahora sí! Si accedemos a la aplicación desde el **navegador** [https://pgdi] deberíamos ser capaces de acceder a todas las funcionalidades de la aplicación, incluyendo la consulta de **cuotas** y **ficheros más pesado**, el **envió de mensajes** de correo y el uso de la **API** de **Google Maps** en el formulario de registro.



Robot PGDI <smtp.segundo.asir@gmail.com>
para usuario

Hola usuariodepr, bienvenid@ al curso de ASIR

Espacio ocupado en disco: **89M** / **500M** (17,97%)

- 87M ~/Escritorio
- 1M ~/.cache
- 452K ~/.local
- 180K ~/.config
- 12K ~/examples.desktop
- 12K ~/.compiz
- 8K ~/.gnupg
- 4K ~/Videos
- 4K ~/Público
- 4K ~/Plantillas

Ficheros

Extracto: pgdi/settings.py

```
# Valores por defecto para los nuevos usuarios
LOGIN_SHELL = '/bin/bash'
HOME_DIRECTORY = '/home/pub/'

# Parametros para la conexion SSH con el servidor NFS
SSH_USER = "comandante"
SSH_HOST = "pgdi"
SSH_PORT = "22"

# Ruta a las claves para la conexion SSH
SSH_KEY_REPQUOTA = "/var/www/.ssh/id_rsa_repquota"
SSH_KEY_DU = "/var/www/.ssh/id_rsa_du"

# Numero de directorios de gran peso a mostrar
MAX_SIZE_FILE_LIMIT = 10

# Umbrales de alerta para disco ocupado (en %)
QUOTA_WARN1_LEVEL = 60
QUOTA_WARN2_LEVEL = 90

# Campos de "repquota" para limites usado/blando/duro
QUOTA_USED_DISK_FIELD = 3
QUOTA_SOFT_LIMIT_FIELD = 4
QUOTA_HARD_LIMIT_FIELD = 5

# smtp config
SMTP_HOST = 'smtp.gmail.com'
SMTP_USER = 'smtp.segundo.asir@gmail.com'
SMTP_PASS = 'pass.segundo.asir'
SMTP_NAME = 'Robot PGDI'

# Django Suit configuration
SUIT_CONFIG = {
    'ADMIN_NAME': 'Admin. PGDI',
}
```

¿Easter Egg?

Esta aplicación ha sido desarrollada por **Juan Antonio Espinosa Prieto**.