Universidad
de La Laguna

GScout
Desarrollo e implantación de una aplicación para la gestión de grupos Scout

Title in English .

José Daniel Juárez Dávila

Dpto. Nombre del Departamento

Escuela Técnica Superior de Ingeniería Informática

Trabajo de Fin de Grado

La Laguna, 3 de junio de 2013

D. **Nombre Apellido1 Apellido2**, con N.I.F. 12.345.678-X profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna

C E R T I F I C A

Que la presente memoria titulada:

"*Titulo del Trabajo.*"

ha sido realizada bajo su dirección por D. **Nombre Apellido1 Apellido2**, con N.I.F. 12.345.678-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 3 de junio de 2013

Agradecimientos

Equipo de scout Aguere por su colaboración en el proyecto.
XXX
XXX
XXX

Resumen

El objetivo de este trabajo ha sido crear una aplicación web para los scout de Aguere 70, la cual facilite la gestión de los socios de dicha organización.

Para ello usamos como framework Django, pero como tambien vamos a trabajar con la estructura de Google App Engine, utilizamos la version de django-nonrel que nos permite utilizar base de datos no relacionales ya que la que usa App Engine sigue la tecnología BigTable de Google. La ventaja del uso de este tipo de base de datos es que son mas escalable, y gracias al framework de Django se pueden manipular por medio de este con QuerySet de Django, salvo que no podemos usar joins ni many to many, etc ni ninguna relacion entre tablas que no sea de clave foranea.

Aparte del frameworky y de App Engine introducimos en la aplicación APIs de Google para poder iniciar sesión con una cuenta de google y obtener los datos de dicha cuenta sin necesidad de almacenarlo, ademas tambien utilizamos otra API para la exportación de información a Google Drive.

En cuanto a las templates de la aplicacion se utiliza la configuración CSS de nos brinda Bootstraps y alguno que otro codigo en jQuery para crear dinamismo en las paginas y con la iteracion del usuario y la aplicación.

Palabras clave: Palabra reservada1, Palabra reservada2, ...

Abstract

Here should be the abstract in a foreing language...

Keywords: *Keyword1, Keyword2, Keyword2, ...*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El objectivo de este proyecto es desarrollar una aplicación para la gestión de un grupo de Scout. La aplicación debera gestionar a los chicos asociados al grupo y mantener su información personal, incluyendo familiares, datos bancarios y medicos.

La tecnologia principal usada se basa en aplicaciones web con un entorno de desarrollo de alto nivel, en nuestro caso don el Framework de Django, y tambien con implantación en la nube, gracias a Google App Engine.

1.1. Antecedentes y estado actual del tema

El Escultismo es un movimiento educativo fundado en el año 1907 por Baden Powell en Inglaterra e instalado en España en 1912. Su musión es dejar este mundo mejor de como lo encontramos, una misión que es posible gracias a una gran labor diaria y educativa que realizan de manera voluntaria jóvenes de todo el mundo.

Hoy en día se pueden encontrar páginas web de organizaciones de grupos Scout, pero la mayoria son simlemente para uso informativo y darse a conocer, como es el caso del grupo Aguere 70, que carece de una aplicación para la gestión de los propios socuts con sus tareas, que es en lo que se ennfocó este proyecto. Existe una implementación, ya desactualizada, basada en tecnología PHP para las funciones básicas en la gestión de grupos Scout: GNU Scout [1]. Esta implementación aunque no fue usada, sirvio para darnos una idea general de como enfocar nuestra aplicación.

De modo que creamos una aplicación en la nube utilizando una versión de Django adaptada a Google App Engine, un servicio de alojamiento web que permite desarrollar aplicaciones online sin necesidad de administrar o mantener servidores dedicados. De esta forma, los usuarios podrán utilizarla lo antes posible, evitando tareas de gestión y administración de sistemas.

1.2. Caracteristicas de la Aplicación

Resumen de los principales recursos utilizados en el desarrollo del proyecto.:

- Framework Django-nonrel

- Despliegue en Google App Engine
- Google APIs (Google+ y Google Drive)
- GitHub
- jQuery
- JavaScript
- Bootstrap CSS

En los próximos capítulos se profundizará en las herramientas empleadas en la elaboración del proyecto.

1.3. Actividades

El desarrollo del proyecto se organizó de las siguientes tareas:

Tarea	**Actividad**
Tarea1	Analisis de la aplicación GNU Scout [1] y el modelo de datos utilizado.
Tarea 2	Entrevistas con los responsables de un grupo Scout para analizar las funcionalidades ya previstas según [1] y añadir/eliminar/modificar aquellas de interés.
Tarea 3	Montar un repositorio GIT [2] para alojar los codigos del proyecto. Definir la estrategia de branching.
Tarea 4	Definir u proyecto en Pivotal tracker[3] para el seguimiento del proyecto. Esta herramienta facilita el desarrollo siguiento metodologías ágiles.
Tarea 5	Desarrollo de un proyecto piloto, realizar la implantación en GAE[2] comprobando el funcionamiento básico de esta plataforma.
Tarea 6	Entrevistas de seguimiento. 2º reunión con los responsables del grupo Scout para mostrar el piloto de la aplicación, refinar diseños,etc
Tarea 7	Desarrollo de la aplicación: implementar las funcionalidades requeridas (posibles entrevistas a lo largo del proceso para comprobar si la implementación cumple los requisitos del cliente: se realizarán por lo menos 4 iteraciones completas: análisis, desarrollo, test, implantación)
Tarea 8	Puesta en producción (fase beta), formación de usuario y gestión de errores.

1.4. Período de Desarrollo del Proyecto

El período de elaboración del proyecto abarca desde el 11 de septiembre de 2012 cuando se presentaron las ofertas de los proyectos, hasta mediados de junio de 2013 que es cuando corresponden las respectivas defensas orales, pero lo que es la elaboración en sí de la aplicación del proyecto abarcó del 30 de Enero de 2013 hasta primeros de Junio incluyendo los retoques finales y puesta a punto de la aplicación.

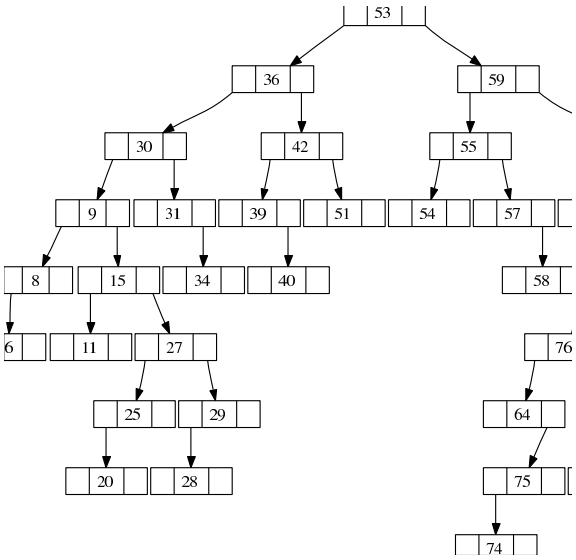

Figura 1.1: Ejemplo

Capítulo 2

Entorno de Desarrollo

En el capítulo anterior se ha introducido los antecedentes como el estado actual del proyecto, nombrado sus herramientas, actividades y periodos de desarrollo. Ahora nos vamos enfocar y describir lo que es el entorno de desarrollo de la aplicación

Partimos de una implementación en PHP, que poseía las funcionalidades básicas, pero el problema era que la base de datos no era compatile con nuestra tecnología, debido a que seguía el model entidad-relación, y nosotros al contrario, necesitábamos crear una base de datos no relacional. Por otro lado teníamos de ejemplo la aplicación web "online scout", la cual era muy completa pero era de pago, y nuestro objetivo era hacer una aplicación gratuita adaptada al cliente, en nuestro caso la organización de scout Aguere 70 de La Laguna.

2.1. Bases de la aplicación

La aplicación GScout esta programada principalmente en lenguaje python, bajo el framework de django-nonrel, ya que como se ha comentado antes se necesitaba este framework especifico para trabajar con bases de datos no relacionales, debido a que como usamos Google App Engine para el despliegue, un requerimiento que tiene esta tecnologia es que solo trabaja con bases de datos no relacionales. En un principio se habia propuesto el uso de pivotaltracker para el seguimiento de la aplicación, pero como solo habia un desarrollador en proceso, pues se descartó la idea y se limito a tener un repositorio gestor de versiones por medio de github, donde se van guardando los cambios oportunos, y segun la información de los "commit" se puede ver lo que se ha hecho.

2.2. Modelo Vista Controlador: Django

Django es un framework de desarrollo web de código abierto, escrito en Python, que cumple en cierta medida el paradigma del Modelo Vista Controlador. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el

framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt.

2.3. Google App Engine

Google App Engine te permite ejecutar tus aplicaciones web en la infraestructura de Google.Las aplicaciones App Engine son fáciles de crear, de mantener y de ampliar al ir aumentando el tráfico y las necesidades de almacenamiento de datos. Con App Engine no necesitarás utilizar ningún servidor: solo tendrás que subir tu aplicación para que los usuarios puedan empezar a utilizarla.

Puedes proporcionar a la aplicación tu propio nombre de dominio (como, por ejemplo, http://www.example.com/) a través de Google Apps. También puedes proporcionarle un nombre que esté disponible en el dominio appspot.com. Podrás compartir tu aplicación con todo el mundo o limitar el acceso a los miembros de tu organización.

Google App Engine admite aplicaciones escritas en varios lenguajes de programación.

Google App Engine permite desarrollar fácilmente aplicaciones que se ejecuten de forma fiable, incluso con pesadas cargas de trabajo y grandes cantidades de datos. App Engine incluye las siguientes funciones:

- Servidor web dinámico, totalmente compatible con las tecnologías web más comunes,
- Almacenamiento permanente con funciones de consulta, clasificación y transacciones,
- Escalado automático y distribución de carga,
- API para autenticar usuarios y enviar correo electrónico a través de Google Accounts,
- Un completo entorno de desarrollo local que simula Google App Engine en tu equipo,
- Colas de tareas que realizan trabajos fuera del ámbito de una solicitud web,
- Tareas programadas para activar eventos en momentos determinados y en intervalos regulares.

2.4. Google APIs

En la aplicación se usaron dos APIs de Google para aumentar y mejorar su funcionalidad.

2.4.1. Google Plus

Como GAE ya posee un modulo de autenticación por medio de Google Auth, en el proyecto se modifico para que se hiciera por medio de Google Plus, generando unas credenciales que se guardaran en el usuarios para que la aplicaccion pueda tener determinados

permisos y poder obtener información que proporciona Google Plus. El objetivo de la incorporación de esta API a nuestra aplicación es evitar almacenar datos personales de los integrantes de la organización que usa la aplicación en una base de datos, sino usar directamente los que nos proporciona Google Plus, como nombre, apellidos, foto de perfíl, dirección, etc.

2.4.2. Google Drive

Uno de los objetivos de la aplicación en cuanto a funcionalidad era poder exportar una tabla filtrada o no con los datos de los socios, como estamos utilizando tecnologia Google, decidimos que la mejor forma era utilizar la API de Google Drive, para crear documentos de hoja de calculo con la información que le pasara la aplicación, para ello fue necesario modificar las credenciales que se almacenaban en los usuarios, para darle permiso y poder utilizar las funciones que nos proporciona dicha API.

2.5. GitHub

GitHub es un software para alojar proyectos utilizando el sistema de control de versiones Git. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

Nuestra aplicación esta alojada en un repertorio público, cuyo repertorio se facilitara en la sección de enlances.

2.6. JavaScript y JQuery

2.6.1. JavaScript

JavaScript es un lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, en bases de datos locales al navegador, etc.

2.6.2. JQuery

jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones (FLV) y agregar interacción con la técnica AJAX a páginas web. Cuyas caracteristicas son:

- Selección de elementos DOM.
- Interactividad y modificaciones del árbol DOM, incluyendo soporte para CSS 1-3 y un plugin básico de XPath.

- Eventos.
- Manipulación de la hoja de estilos CSS.
- Efectos y animaciones.
- Animaciones personalizadas.
- AJAX.
- Soporta extensiones.
- Utilidades varias como obtener información del navegador, operar con objetos y vectores, funciones para rutinas comunes, etc.
- Compatible con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+.

2.7. CSS y Bootstrap

Para generar el estilo de nuestra aplicación utilizamos las tecnologías que se descriran a continuación.

2.7.1. CSS (Hojas de estilo)

Las hojas de estilo en cascada (Cascading Style Sheets, o sus siglas CSS) hacen referencia a un lenguaje de hojas de estilos usado para describir la presentación semántica (el aspecto y formato) de un documento escrito en lenguaje de marcas. Su aplicación más común es dar estilo a páginas webs escritas en lenguaje HTML y XHTML, pero también puede ser aplicado a cualquier tipo de documentos XML, incluyendo SVG y XUL.

2.7.2. Bootstrap

Para agilizar el maquetado de las páginas web de la aplicación utilizamos Twitter Bootstrap, que es una colección de herramientas de software libre para la creación de sitios y aplicaciones web. Contiene plantillas de diseño basadas en HTML y CSS con tipografías, formularios, botones, gráficos, barras de navegación y demás componentes de interfaz, así como extensiones opcionales de JavaScript.

Capítulo 3

Descripción de la aplicación

En el cápitulo 1 se describió brevemente la aplicación. En este cápitulo nos expandiremos y eplicaremos toda la funcionalidad de GScout.

3.1. Usuarios

La aplicación esta destinada para el uso de los empleados de la organización de scout Aguere 70, el inicio de sesión esta restringido, por tanto solo podrán acceder con su cuenta de Google perteneciente al dominio "gruposcoutaguere70.com". En principio todos los usuarios son genericos, pero la aplicación esta preparada para clasificarlos según su cargo, y que cada usuario pueda desempeñar una funcion y otras dependiendo del cargo asignado por el administrador.

[[Imagen del login]]

3.2. Gestión de socios

A continuación nombraremos las funcionalidades entorno a la gestión de socios.

3.2.1. Creación de Socios

Los usuarios podrán crear nuevos socios rellenando los formularios predefinidos en la aplicación, donde se clasifican los datos en personales, economicos, familiares y medicos.

[[Imagen de los formularios]]

3.2.2. Edición de Socios

Una vez creados los socios se pueden editar sus datos, de una manera similar a la de creación de socios, simplemente buscamos el socio y le damos a editar, en la pestaña del tipo de información que querramos cambiar.

[[Imagen de formularios de edición]]

3.2.3. Borrado de Socios

Tambien podremos borrar los socios, solo que estos socios no son borrados permanentemente con esta opción, sino que su estado pasará a inactivo y no estara visible a todo el mundo.

[[Imagen borrado y estado inactivo]]

3.2.4. Familiares

Es obligado que cada socio pertenezca a una familia, cuya familia tendra un responsable que sera el indentificador de la familia, a la que se le pueden asignar varios socios, padres y/o tutores. En cierto modo, podremos formar lo que seria un arbol de familia entre los socios, saber que socios conviven en la misma familia, saber quienes son los padres/tutores del socio, etc.

[[Imagen arbol familia]]

Tambien GScout nos da la opción de cambiar o crear una familia nueva, por cualquier circunstancia que pueda pasar.

[[Imagen edicion familia]]

3.2.5. Medicamentos

Muchos de los socios podrían estar bajo medicación de algun tipo, por tanto existe un modulo en el que podemos anexar a los socios los datos de sus medicamentos, como el nombre, la pauta y la dosis correspondiente.

[[imagen medicamentos]]

3.3. Cambios de unidad

A petición de la organización scout Aguere 70 se creó una función que llamada **Cambio de Unidad** que consistes en analizar todos los socios, verificar su edad, y si procede cambiarlo a la sección/unidad acorde a su edad.

[tabla de unidades por rango de edades]

3.4. Listados de información

Hay una sección en la aplicación en la que podemos listar los socios visualizando sus datos dependiendo del listado, es decir, existen un listado especifico para los datos personales y otro para los datos economicos.

En estos listados podremos interactuar filtrando los datos por medio de varios filtro, incluso ordenarlos alfabeticamente en orden creciente y decreciente si se desea.

[[Imagen de un listado ejemplo con algunos filtros]]

3.5. Busquedas por ID

Tambien se pueden realizar busquedas directas por el ID del socio, en la que nos redirecciona a directamente a la ficha técnica del socio.

[[Imagen de busquedas por ID]]

3.6. Exportaciones

Los resultados de los listados los podemos exportar a Google Drive en formato de hoja de calculo, con solo darle al botón de exportar debajo del listado. Veamos un ejemplo con imagenes:

[[Boton exportar]]

El resultado seria el siguiente:

[[Documento en drive]]

3.7. Importaciones

GScout brinda a la organización de scout Aguere 70 la posibilidad de migrar su base de datos en access a la propia aplicación, siempre y cuando este fichero se transforme en un archivo .csv, con la herramienta MDB Tools por ejemplo, y solo haría falta subirla a la aplicación que lo demás lo hace de manera automatica la aplicación. De esta manera tendremo los datos de la antigua base de datos en la nueva base de datos que pertenece a GScout.

[[Imagen de subida de archivo csv]]

Capítulo 4

Desarrollo de la aplicación

En el capitulo 3 se describieron las funcionalidades de la aplicación. A continuación hablaremos de todo lo que conllevó el proceso de elaboración del proyeto, desde su comienzo hasta el final, incluyendo las entrevistas, problemas, etc.

4.1. Primeros Pasos

Partimos de una implementación en PHP, que poseía las funcionalidades básicas, pero el problema era que la base de datos no era compatible con nuestra tecnología, debido a que seguía el modelo entidad-relación, y nosotros al contrario, necesitábamos crear una base de datos no relacional. Por otro lado teníamos de ejemplo la aplicación web "online scout", la cual era muy completa pero era de pago, y nuestro objetivo era hacer una aplicación gratuita adaptada al cliente, en nuestro caso la organización de scout Aguere 70 de La Laguna.

Para adaptarla lo mejor posible, lo primero que hicimos fue tener una reunión con el grupo scout Aguere 70, previamente a esta reunión se realizó una serie de tutoriales básicos de Django para refrescar conocimientos, ademas de hacer pruebas con la tecnologia de Google App Engine e incorporarlas al framework de django-nonrel.

Después de estos pasos previos, se realizo la primera reunión con el grupo Aguere 70, en la que se elaboró un análisis de requisitos, se estudiaron las posibles funcionalidades que tendría la aplicación, seleccionando las funciones fundamentales, debido al corto tiempo y personal que había para realizar la aplicación.

4.2. Requisitos Iniciales

Después de la reunión con los interesados se estableció un esquema compuesto por 7 aplicaciones, la cual una de ella era la principal y de esa dependerían el resto. Las aplicaciones son las siguientes:

- Socios: esta es la aplicación principal que tendrá los datos de los socios que usaran las demás aplicaciones. Contiene:
  - Mantenimiento de socios.
  - Cambios de Unidad
  - Informes (Familiares, Unidades, Médicos, Personales)
- Lista de espera
  - Entradas
  - Propuestas de entrada a grupo
  - Incidencias
- Biblioteca
  - Mantenimiento
  - Préstamos
- Intendencia
  - Mantenimiento
  - Préstamos
  - Informes
    - Prestados
    - Estados
  - Campamentos
- Recursos
  - Dinámicas
  - Talleres
  - Juegos
  - Resto de Actividades
- Tesorería
  - Control de Cuentas
  - Presupuestos
  - Emisión de recursos
  - Resúmenes
    - Anual
    - Trimestral

- Varios
  - Compañía de seguridad
  - Agenda
  - Recetario

Visto las siguientes aplicaciones se comprobó que el proyecto era ambicioso y extenso, por lo tanto nos enfocamos en el apartado de la app de Socios que era la mas importante. Se nos facilitó la ficha de inscripción en formato papel que han de cumplimentar los socios para inscribirse en la organización. Por tanto nos sirvió de modelo para generar una primera aproximación del modelo de datos que va a llevar la aplicación de Socios.

[[Espacio para el primer modelo de datos]]

4.3. Preparación para el proyecto

Primero que nada necesitamos instalar el framework de django-nonrel, para ello lo que se hizo fue crear un nuevo proyecto vacío con el programa "Aptana Studio 3", el cual debía de tener los archivos que nos proporciona la web "www.allbuttonspressed.com" que contienen todos los archivos necesarios para configurar un proyecto que funcione con django-nonrel y Google App Engine, de modo que el proyecto nos quedaría de la siguiente manera:

- django-nonrel/django =¿¡project¿/django
- djangotoolbox/djangotoolbox =¿¡project¿/djangotoolbox
- django-autoload/autoload =¿¡project¿/autoload
- django-dbindexer/dbindexer =¿¡project¿/dbindexer
- djangoappengine =¿¡project¿/djangoappengine

Importante: Al instalar el SDK de App Engine incluir el PATH en el archivo .profile de nuestra carpeta personal en Linux.

[[Archivo app.yaml]] Describir el archivo app.yaml

4.4. Inicio de sesión

Como se ha dicho en el capitulo 3 el inicio de sesión esta limitado al dominio del grupo de scout de Aguere 70, lo cual se accede a la aplicación por medio de sus cuentas de google asociadas a dicho dominio.

El archivo settings.py se configuró de la siguiente manera:

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

INSTALLED_APPS = (
    .
    .
    .
    .
    'plus',
    'gaeauth',
    'djangoappengine',
    .
    .
    .
)

.
.
.
.

AUTHENTICATION_BACKENDS = (
      'gaeauth.backends.GoogleAccountBackend',
)
ALLOWED_DOMAINS = ('gruposcoutaguere70.org',)
```

Listing 4.1: Ejemplo de listado desde archivo

Donde se puede apreciar las INSTALLE DAPPS necesarias para este apartado de la aplicación. Pasamos a describir cada una:

- **djangoappengine**: es un backend para poder utilizar la base de datos que nos brinda Google App Engine.

- **gaeauth**: es otro backend que nos facilita el inicio de sesión a traves de la interfaz de Google Accounts, para iniciar sesión con las cuentas de Google en nuestra aplicación.

  Es la ultima parte del archivo settings.py se ve claramente que esta seleccionado en AUTHENTICATION BACKENDS el backend de gaeauth y seguidamente elegimos en ALLOWED DOMAINS el dominio en el cual queremos restrigir el acceso, en nuestro caso solo los usuarios pertenecientes al dominio "gruposcoutaguere70.org" podran acceder a la aplicación.

- **plus**: Este es el modulo que nos permite obtener la información personaly foto de

perfil de los usuarios que acceden a la aplicación por medio de su cuenta de Google+, sin necesidad de guardar esos datos en nuestra base de datos.

Este fue uno de los modulos que más dio problemas, primero habia que acceder a la web de Google apis con la cuenta de desarrollador y activar la API de Google+ posteriormente tenemos que editar el acceso a las APIs, para introducir nuestra aplicación en el entorno, por lo tanto se nos generaria una serie de datos que podriamos introducirlos en nuestra aplicación uno a uno o bien en formato JSON, para que esta pueda acceder a los servicios que nos proporciona la API.

Dicho esto una vez que el usuario inicia sesión, en el proceso se accede a la vista de la app plus, que esta cargara la configuración de Google API por medio de un archivo JSON o introduciendo una a una de la siguiente manera la configuración necesaria para utilizar el servicio de Google+:

```
FLOW = OAuth2WebServerFlow (
    client_id ='480482329789-dj8a6lggtm9cpml3oib7imbvs3b9fv4q.apps.googleusercont
    client_secret = 'C5umC7xa_ML704wnmdT79Bh0',
    token_uri='https://accounts.google.com/o/oauth2/token',
    scope= ['https://www.googleapis.com/auth/drive','https://www.googleapis.com/a
    redirect_uri= 'http://localhost:8000/oauth2callback',
    access_type='offline',
    approval_prompt='force')
```

Listing 4.2: Ejemplo de listado desde archivo

Donde **client-id** y **client-secret** son los datos que apuntan al acceso de API que ha establecido el programador de la aplicación, **token-uri** lo delegamos para que lo ejetute oauth2, en **scope** se añaden las direcciones de las API que se van a usar, en nuestro caso la de Google+ y la de Drive(esta la explicaremos mas adelante), en **redirect-uri** establecemos la direccion de retorno a nuestra aplicación por medio de la llamada oauth2cabllback que la incorpora el modulo plus.

Con todo esto lo que logramos es que en el inicio de sesión se establezca una conexion entre el usuario y las APIs de google, en la cual se acepten una serie de permisos para posteriormente generar una credencial, que se seguardara en el usuario, con la cual podra acceder a los servicios activados previamente por el programador en Google APIs. Es importante señalar que el modelo de User que nos ofrece django se modifico para que se pudieran guardar las credenciales, ademas se añadio el campo de **cargo**, donde se le asignara a cada usuario un cargo especifico, por tanto las funciones que podra realizar en la aplicación estaran supeditadas a dicho cargo.

Nota: las credenciales suelen expirar a la hora por tanto, una vez expiradas el usuario no podra acceder a la aplicación, al menos que se actualiecen, una forma de automatizar esto es que en cada inicio de sesión se restablescan las credenciales, por eso añadimos dos campos mas a la variable FLOW, que son el **access-type='offline'** y **approval-prompt='force'** con esto logramos forzar el refrezco de credenciales una vez estas caduquen.

4.5. Creación de Usuarios

Una vez definida la primera aproximación del modelo de datos que vamos a emplear en la aplicación, empezamos con la creacion de la app de socios que va a gestionar todo lo referente a la información de los socios de la organización.

Esta tarea era simple pero tediosa a la hora de realizar templates formularios para que luego en la vista de la app de socios por medio de diferentes llamadas POST, se fueran contruyendo los objetos necesarios para la creacion de un nuevo socio, cada objeto almacenaba un tipo de información especifica segun el modelo de datos. Por tanto el modelo final de socios tenia las siguientes tablas:

```
from django.db import models


class Autorizaciones(models.Model):
    #3 autorizaciones de momento
    autorizacion_medica=models.BooleanField()
    autorizacion_actividades=models.BooleanField()
    autorizacion_fotografias=models.BooleanField()


class Familia(models.Model):
    nombre=models.CharField(max_length=100)
    apellidos=models.CharField(max_length=100)
    nif=models.CharField(max_length=100,primary_key=True, unique=True)


class Familiares(models.Model):
    nombre=models.CharField(max_length=100)
    apellidos=models.CharField(max_length=100)
    nif=models.CharField(max_length=100,primary_key=True, unique=True)
    telefono=models.CharField(max_length=100)
    movil=models.CharField(max_length=100)
    email=models.CharField(max_length=100)
    rol=models.CharField(max_length=100)
    familia_id=models.ForeignKey(Familia,null=True)


class Socio(models.Model):
    n_asociado=models.CharField(max_length=100, primary_key=True, unique=True)
    alta=models.BooleanField()
    autorizaciones=models.ForeignKey(Autorizaciones, null=True)
    familia_id=models.ForeignKey(Familia,null=True)

class D_Personales(models.Model):
    nombre=models.CharField(max_length=100)
    apellidos=models.CharField(max_length=100)
```

```
    dni=models.CharField(max_length=100)
    sexo=models.CharField(max_length=20)
    f_nacimiento=models.DateTimeField()
    direccion=models.CharField(max_length=200)
    c_postal=models.CharField(max_length=100)
    localidad=models.CharField(max_length=100)
    provincia=models.CharField(max_length=100)
    fijo=models.CharField(max_length=100)
    movil=models.CharField(max_length=100)
    f_ingreso=models.DateTimeField(blank=True, null=True)
    f_baja=models.DateTimeField(blank=True, null=True)
    seccion=models.CharField(max_length=100)
    estudios=models.TextField()
    profesion=models.TextField()
    deportes=models.TextField()
    aficiones=models.TextField()
    socio_id=models.ForeignKey(Socio)

class D_Medicos(models.Model):
    c_seguro=models.CharField(max_length=100)
    n_poliza=models.CharField(max_length=100)
    enfermedad=models.CharField(max_length=2, default='no')
    t_enfermedad=models.TextField()
    enfermedad_cfp=models.CharField(max_length=2, default='no')
    t_enfermedad_cfp=models.TextField()
    operado=models.CharField(max_length=2, default='no')
    t_operado=models.TextField()
    alergia=models.CharField(max_length=2, default='no')
    t_alergia=models.TextField()
    otras_alergias=models.CharField(max_length=2, default='no')
    t_otras_alergias=models.TextField()
    dieta=models.CharField(max_length=2, default='no')
    t_dieta=models.TextField()
    toma_medicamentos=models.CharField(max_length=2, default='no')
    info_adicional=models.TextField()
    socio_id=models.ForeignKey(Socio)


class D_Economicos(models.Model):
    titular=models.CharField(max_length=100)
    nif_titular=models.CharField(max_length=100)
    banco=models.CharField(max_length=100)
    sucursal=models.CharField(max_length=100)
    localidad=models.CharField(max_length=100)
    d_banco=models.CharField(max_length=100)
    d_sucursal=models.CharField(max_length=100)
    dc=models.CharField(max_length=100)
    n_cuenta=models.CharField(max_length=100)
    socio_id=models.ForeignKey(Socio)
```

```
class Medicamentos(models.Model):
    nombre=models.CharField(max_length=100)
    dosis=models.CharField(max_length=100)
    pauta=models.CharField(max_length=100)
    socio_id=models.ForeignKey(Socio)
```

Listing 4.3: Ejemplo de listado desde archivo

Como podemos ver el modelo esta compuesto por 8 tablas, debido a que no se pueden establecer relaciones en la base de datos de tipo no relacional, la unica forma de en cierto modo comunicar/unir una tabla con otra, es con lo que llamamos claves foraneas(ForeignKey), de esta manera podremos asosciar las tablas que nos interesen de manera unidimensional.

Se puede observar para que es cada tabla y el tipo de información que guarda, se dejo elaborada aunque no se usa todavía la tabla de **Autorizasiones** para que en un futuro los socios almacenen ahí las posibles autorizaciones/permisos cuando se realice una excursión, uso de fotografias de manera pública, etc.

Retomando a la creación de usuarios es necesario rellenar todos los campos importantes de cada objecto para conservar la integridad de la base datos. Los formularios comprueban que esto se cumpla y al finalizar lo manda como un POST al servidor donde es procesado y si todo esta correcto se guarda en la base de datos.

Por otro lado en la carpeta **templates/socios** están todos los archivos .html que se usan para la interacción del usuario desde el cliente con el servidor, las template que se usan para la creación de usuario se distinguen por empezar por **f-[nombreTabla].html**

4.6. Modificación de Usuarios

Un proceso similar al de creación de usuarios se implementó para la modificación de los respectivos datos de los socios, los archivos de las templates eran similares, salvo que tenian trozos de código en Javascript y variables que se pasaban de la vista a la template para autocompletar los campos de los formularios con los datos del socio, de esta forma si se va a editar un campo en concreto no es necesario rellenar todos los campos restantes.

Los archivos de templates relacionados con la edición de datos de los usuarios se caracterizan por tener la siguiente nomenclatura **f-edit-[nombreTabla]**

4.7. Listados de información

Se implementaron dos formas de visualizar los datos de los socios, la mas simple es una vez obtenido el ID del socio, se accede a las paginas de información del socio, donde cada template esta enfocada a una determinada tabla del modelo de datos, de esta forma los

datos los clasificariamos en personales, economicos, medicos y familiares. En la vista lo unico que se hace es volcar los datos del socio a la template, que es la que se encarga de la visualización de los repectivos datos.

```
################## Vista ########################

def economicos_socio(request, n_asociado):
    try:
        socio = Socio.objects.get(n_asociado=n_asociado)
    except:
        return  HttpResponseRedirect('/socios/search')

    economicos = D_Economicos.objects.get(socio_id=socio)

    return render_to_response('socios/datos_economicos.html', {'economicos': economic

################## Template ########################

{% block form %}
    <p><strong>Titular de la cuenta: </strong> {{economicos.titular}}</p>
    <p><strong>N.I.F. del Titular: </strong> {{economicos.nif_titular}}</p>
    <p><strong>Banco: </strong> {{economicos.banco}}</p>
    <p><strong>Sucursal: </strong> {{economicos.sucursal}}</p>
    <p><strong>Localidad: </strong> <time>{{economicos.localidad}}</time></p>
    <p><strong>Datos de la cuenta: </strong> {{economicos.d_banco}}-{{economicos.d_su
    <a class="btn btn-large btn-primary" href="/socios/{{ economicos.socio_id.n_asocia
{% endblock %}
```

Listing 4.4: Ejemplo de listado desde archivo

Por otro lado para tener un vistazo general de los socios, se elaboró una tabla donde se listan todos los socios mostrando sus datos acorde con el listado en el que se este, es decir existen varios listados para representar los datos de los socios, listados de datos personales, economicos, etc. Para el formato de la tabla se encontro un widget con la apariencia de bootstrap para mantener la escencia de la interfaz visual, en cuyas tablas se pueden realizar ordenaciones, filtrados y exportaciones que describirán detalladamente en los siguientes apartados.

4.8. Filtrado de tablas

Para poder utilizar las funciones de filtrado y ordenaciones en los listados, era necesario incorporar en nuestra aplicación los siguientes scripts:

```
<!-- jQuery -->
<script src="/static/js/jquery-1.9.1.js" type='text/javascript'></script>

<!-- Demo stuff -->
<link href="/static/css/bootstrap.css" rel="stylesheet">
```

```
<!-- Tablesorter: required for bootstrap -->
<link rel="stylesheet" href="/static/css/theme.bootstrap.css">
<script src="/static/js/jquery.tablesorter.js"></script>
<script src="/static/js/jquery.tablesorter.widgets.js"></script>

<!-- filter formatter code -->
<link rel="stylesheet" href="/static/css/filter.formatter.css">
<script src="/static/js/jquery.tablesorter.widgets-filter-formatter.js"></script>

  <!-- jQuery UI for range slider -->
<link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.0/ther
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.0/jquery-ui.min.js"><

<!-- Tablesorter: optional -->
<link rel="stylesheet" href="/static/addons/pager/jquery.tablesorter.pager.css">
<script src="/static/addons/pager/jquery.tablesorter.pager.js"></script>
```

Listing 4.5: Ejemplo de listado desde archivo

En la zona comentada de cada script se nombra para que se utiliza cada uno. Ademas de estos script es necesario introducir una función en Javascript que nos proporciona el propio widget donde se especifican la configuracion de tabla, como el tema de apariencia, los filtros, paginación etc.

Posteriormente la estructura de la tabla lleva la siguiente sintaxis:

```
<table class="tablesorter">
 <!-- Encabezado -->
 <thead>
  <tr>
   <th>ID</th>
   <th class="filter-select filter-exact" data-placeholder="Todos">Sexo</th>
   .
   .
   .
  </tr>
 </thead>
 <!-- Pie de tabla -->
 <tfoot>
  <tr>
   <th>ID</th>
   <th>Sexo</th>
   .
   .
   .
  </tr>
  <tr>
   <!-- Aqui se define el numero de columnas que tendra la tabla -->
   <!-- Y los intervalos de paginacion -->
   <th colspan="9" class="pager form-horizontal">
```

```
    <button class="btn first"><i class="icon-step-backward"></i></button>
    <button class="btn prev"><i class="icon-arrow-left"></i></button>
    <span class="pagedisplay"></span> <!-- this can be any element, including an inpu
    <button class="btn next"><i class="icon-arrow-right"></i></button>
    <button class="btn last"><i class="icon-step-forward"></i></button>
    <select class="pagesize input-mini" title="Select page size">
     <option selected="selected" value="10">10</option>
     <option value="20">20</option>
     <option value="30">30</option>
     <option value="40">40</option>
    </select>
    <select class="pagenum input-mini" title="Select page number"></select>
   </th>
  </tr>
 </tfoot>
 <!-- Por ultimo esta el cuerpo de la tabla que es el que va a contener los datos de
 <tbody>
  {% for s in socios %}
   <th><td>
     <!-- Aqui se introduce una etiqueda <td> por cada dato que se quiere representar
     <!-- segun el numero de columnas -->
    </td>
   </th>
  {% endfor %}
 </tbody>
 </table>
```

Listing 4.6: Ejemplo de listado desde archivo

4.9. Exportaciones a Google Drive

Para poder llevar las exportaciones a Google Drive primero que nada se tiene que establecer un permiso entre el usuario, la aplicación y la API de Google Drive para poder utilizar el servicio. Esto con las credenciales que se han comentado en la sección de **Inicio de Sesión**, añadiendo en el campo de textbfscope la ruta al servicio Drive antes de generar la credencial.

Ahora bien, los filtrados de los listados se realizan sobre la template en el cliente, por tanto si quieremos exportar una listado filtrado tenermos que en cierto modo, realizar el filtrado en el lado del servidor. Esto se logro modificando los archivos del tablesorter para que cuando generara la tabla, cada input de filtrado tenga un nombre, el cual se utilizara para por medio de una llamada POST, obtener los parametros de filtrado. y realizar el filtrado directamente con el modelo de datos.

Una vez realizado esto, nos queda el paso intermedio para poder exportar los datos a Google Drive, ya que para exportar necesitamos crear un fichero de extensión .csv para que la API de Google Drive lo interprete y lo procese. De modo que la construcción del

fichero se realizó de la siguiente manera:

```
#Indicamos la fecha de elaboracion del fichero
now = datetime.datetime.now()
now = now.strftime("%d-%m-%Y")
# "datos" almacena el contenido en formato string de todos los datos
# incluidas las comas y los retornos de carro, para facilitar la creacion del fichero
datos="ID, Titular, NIF Titular, Banco, Nombre Socio, Apellidos Socio, DNI Socio\n"
for d in range(len(lista)):
    datos += str(socios[lista[d]]['socio_id_id']) + "," + unicodedata.normalize('NFKD

# media_body prepararÃ¡ el contenido de datos para transformarlo al formato csv
# gracias a la funcion MediaIoBaseUpload, es importante poner la resumable=False
# para evitar problemas de carga y de formato.
media_body = MediaIoBaseUpload(StringIO.StringIO(datos), 'text/csv', resumable=False)

# body almacenara el titulo del fichero, la descripcion y el tipo de fichero.
body = {
  'title': 'Datos_Economicos_'+now+'',
  'description': 'A test document',
  'mimeType': "text/csv"
}

# Finalmente se llama al servicio de drive_service al cual le pasamos por parametros
# que tendra el contenido y convert=True para obligar la conversion del fichero.
file = drive_service.files().insert(body=body, media_body=media_body, convert="true")
pprint.pprint(file)
```

Listing 4.7: Ejemplo de listado desde archivo

4.10. Cambios de Unidad

Esta función fue bastante demandada por la organización y consiste en crear una función que calcule la edad de los socios y en según la edad los clasifique en la sección/unidad que le corresponda de acuerdo al rango previamente establecido.

El código de dicha función es el siguiente:

```
def cambio_unidad(request):

    socios = D_Personales.objects.all()

    date = datetime.datetime.now()
    date = int(date.strftime('%Y'))

    for s in socios:
        if (date - int(s.f_nacimiento.strftime('%Y'))) < 11 :
            s.seccion="Manada"
```

```
        if ((date - int(s.f_nacimiento.strftime('%Y')) >= 11) and (date - int(s.f_na
< 14)):
            s.seccion="Tropa"
        if ((date - int(s.f_nacimiento.strftime('%Y')) >= 14) and (date - int(s.f_na
< 17)):
            s.seccion="Esculta"
        if ((date - int(s.f_nacimiento.strftime('%Y')) >= 17) and (date - int(s.f_na
< 20)):
            s.seccion="Rover"
        if (date - int(s.f_nacimiento.strftime('%Y'))) >= 20 :
            s.seccion="Scouter"

        s.save()
```

Listing 4.8: Ejemplo de listado desde archivo

4.11. Importación de base de datos antigua

La organización de scout Aguere 70 tiene una base de datos donde hasta el momento guarda los datos de los socios, todos los datos están almacenados en una sola tabla y se nos propuso si era posible importar esta base de datos y volcarla en nuestra aplicación.

El principal problema era que esta toda la información en un sola tabla por tanto habria que implementar un mecanismo para separar la información y adaptarla a nuestro modelo de datos.

Por otro lado tenemos el formato del fichero de la base de datos que estaba en formato textbfaccess, de modo que era necesario convertirlo a un formato que pudiera procesar la aplicación de manera eficiente. Se pensó en el formato .csv separado por comas, ya que habia una herramienta llamada **MDB Tools** que convertia los ficheros accecc en csv.

Una vez obtenido el fichero csv, se implemento un mecanismo para subir el fichero a la aplicación sin que este se guardara en el servidor, sino que solo estuviera ahí mientras se este procesando. De modo que se creo una nueva app llamada **upload** especificamente para este proceso.

Esta app tiene un fichero **forms.py** que contiene el formato del form para datos de tipo fichero (FileField).

```
# -*- coding: utf-8 -*-
from django import forms

class DocumentForm(forms.Form):
    docfile = forms.FileField(
        label='Select a file',
        help_text='max. 42 megabytes'
    )
```

Listing 4.9: Ejemplo de listado desde archivo

Esto lo procesa la template y posteriormente con la llamada de POST se envia el archivo al servidor, que se tratara de la manera siguiente:

```
# Con la llamda a request.FILES['nombre'] se guarda el contenido del fichero
# en la variable paramFile
paramFile = request.FILES['docfile'].read()

# Con csv.reader(paramFile.splitlines()) dividimos en lineas el
# contenido del fichero para guardarlo en un array bidimensional
# compuestos por filas y columnas.
portfolio = csv.reader(paramFile.splitlines())

# Como no nos interesa la primera fila del documento que contiene los
# nombre de las columnas con el comanto .next() la omitimos.
portfolio.next()
```

Listing 4.10: Ejemplo de listado desde archivo

Después se procesa casilla a casilla el array y se introducen los datos en la base de datos, siempre cuidando la integridad de la base de datos, de modo que no queden ambiguedades, ni campos vacios que sean importantes en cada tabla del modelo.

Capítulo 5

Conclusiones y trabajos futuros

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir unas conclusiones y unas líneas de trabajo futuro

Capítulo 6

Summary and Conclusions

This chapter is compulsory. The memory should include an extended summary and conclusions in english.

6.1. First Section

Capítulo 7

Presupuesto

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir un presupuesto.

7.1. Sección Uno

Tipos	**Descripcion**
AAAA	BBBB
CCCC	DDDD
EEEE	FFFF
GGGG	HHHH

Tabla 7.1: Tabla resumen de los Tipos

Apéndice A

Título del Apéndice 1

A.1. Algoritmo XXX

```
*****************************************************************************
*
* Fichero .h
*
*****************************************************************************
*
* AUTORES
*
*
* FECHA
*
*
* DESCRIPCION
*
*
*****************************************************************************/
```

A.2. Algoritmo YYY

```
/*****************************************************************************
 *
 * Fichero .h
 *
 *****************************************************************************
 *
 * AUTORES
 *
 * FECHA
 *
 * DESCRIPCION
```

```
 *
 *
 ********************************************************************************/
```

Apéndice B

Título del Apéndice 2

B.1. Otro apendice: Seccion 1

Texto

B.2. Otro apendice: Seccion 2

Texto

Bibliografía

[1] ACM LaTeX Style. http://www.acm.org/publications/latex_style/.

[2] GitHub. http://www.github.com.

[3] FACOM OS IV SSL II USER'S GUIDE, 99SP0050E5. Technical report, 1990.

[4] D. H. Bailey and P. Swarztrauber. The fractional Fourier transform and applications. *SIAM Rev.*, 33(3):389–404, 1991.

[5] A. Bayliss, C. I. Goldstein, and E. Turkel. An iterative method for the Helmholtz equation. *J. Comp. Phys.*, 49:443–457, 1983.

[6] C. Darwin. *The Origin Of Species.* November 1859.

[7] C. Goldstein. Multigrid methods for elliptic problems in unbounded domains. *SIAM J. Numer. Anal.*, 30:159–183, 1993.

[8] P. Swarztrauber. *Vectorizing the FFTs.* Academic Press, New York, 1982.

[9] S. Taásan. *Multigrid Methods for Highly Oscillatory Problems.* PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1984.