



Universidad
de La Laguna

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

*Developmen and implementation of an application for Scout group
management*

José Daniel Juárez Dávila

Dpto. de Estadística, Investigación Operativa y Computación

Escuela Técnica Superior de Ingeniería Informática

Trabajo de Fin de Grado

La Laguna, 17 de junio de 2013

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Estadística, Investigación Operativa y Computación de la Universidad de La Laguna

C E R T I F I C A

Que la presente memoria titulada:

“**GScout: Desarrollo e implantación de una aplicación para la gestión de grupos Scout**”

ha sido realizada bajo su dirección por D. **José Daniel Juárez Dávila**, con N.I.F. 78.645.387-S.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 17 de junio de 2013

Agradecimientos

Equipo de scout Aguere 70 por su colaboración en el desarrollo del proyecto.

Resumen

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

Para ello utilizaremos el framework Django y realizaremos el despliegue en la infraestructura de Google App Engine. Un requerimiento que tiene esta infraestructura es que trabaja con bases de datos no relacionales, en concreto sigue el modelo no-relacional Big-Table, por tanto es necesario utilizar la versión de Django-nonrel que es compatible con este tipo de base de datos.

Una de las ventajas del uso de este tipo de base de datos es su escalabilidad, pues están diseñadas para manejar grandes volúmenes de datos. Gracias al modulo QuerySet de Django podemos manipular la información en estas bases de datos, con las restricciones de no poder usar “joins” ni “many to many”, ni ninguna relación entre tablas que no sea de clave foránea.

A parte del framework y el despliegue en Google 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, además de utilizar otra API para la exportación de información a Google Drive.

En cuanto al diseño de la aplicación se utilizan tecnologías CCS y Ajax, introduciendo los frameworks Bootstraps y jQuery que mejoran la usabilidad de la interfaz de usuario y descargan parte de la funcionalidad en el navegador.

Palabras clave: Scout, Django, Google App Engine, Base de datos no-relacional, Google Plus, Google Drive, CSS, AJAX, Bootstraps, jQuery

Abstract

The goal of this project is to develop a web application for the Aguere 70 Scout group. The applications facilitates the typical management issues in a Scout organization.

To implement the applications, we have been used the Django framework. The deployment has been carried out on the Google App Engine infrastructure. One of the requirements to work with this infrastructure is the use of a non-relational database, specifically following the non-relational BigTable model. Therefore it is necessary to use the Django-nonrel version of the framework that supports this type of database.

One of the advantages of these databases is its scalability. They are designed to manage BigData applications. Thanks to the Django framework, we can manipulate the data with Django QuerySet, but with the restrictions of not using joins, many to many, neither table relationship other than foreign key.

Besides the framework used and the deployment on Google App Engine, we introduced other Google technologies. For example, the use of Google+ API to login with a google account and get its data profile, or the Google Drive API to implement import/export facilities.

Regarding the design of the application using CCS and Ajax technologies, we introduce Bootstraps and jQuery frameworks. This technologies enhance the usability of the user interface as well as deploys part of the functionality in the browser.

Keywords: *Scout, Management, Django, Google App Engine, Database, non-relational, API, Google Plus, Google Drive, CSS, AJAX, Bootstraps, jQuery*

Índice general

1. Introducción	1
1.1. Antecedentes y estado actual del tema	1
1.2. Características de la Aplicación	2
1.3. Actividades	2
1.4. Período de Desarrollo del Proyecto	3
2. Entorno de Desarrollo	5
2.1. Bases de la aplicación	5
2.2. Arquitectura Software	6
2.2.1. Modelo Template View (MTV)	6
2.2.2. Django	6
2.3. Herramientas de Desarrollo	7
2.3.1. Despliegue: Google App Engine	7
2.3.2. Aumento funcionalidad: Google APIs	8
2.3.3. Gestión de versiones y repositorio: GitHub	8
2.3.4. Dinamismo en las plantillas: JavaScript y JQuery	8
2.3.5. Apariencia de la Aplicación: CSS y Bootstrap	9
2.4. Herramientas para Calidad de Código	10
3. Descripción de la aplicación	13
3.1. Usuarios	13
3.2. Gestión de socios	14
3.2.1. Creación de Socios	14
3.2.2. Visualización y Edición de Socios	15
3.2.3. Borrado de Socios	16
3.2.4. Familiares	17
3.2.5. Medicamentos	18
3.3. Cambios de unidad	19
3.4. Listados de información	19
3.5. Búsquedas por ID	20
3.6. Exportaciones	20
3.7. Importaciones	22

4. Desarrollo de la aplicación	25
4.1. Primeros Pasos	25
4.2. Requisitos Iniciales	25
4.3. Preparación para el proyecto	27
4.4. Inicio de sesión	29
4.5. Creación de Usuarios	31
4.6. Modificación de Usuarios	33
4.7. Listados de información	33
4.8. Filtrado de tablas	34
4.9. Exportaciones a Google Drive	35
4.10. Cambios de Unidad	36
4.11. Importación de base de datos antigua	37
4.12. Calidad de Código: Pylint y Pymetrics	38
5. Conclusiones y trabajos futuros	41
6. Summary and Conclusions	43
7. Presupuesto	45
Bibliografía	46

Índice de figuras

1.1.	Tabla del Cronograma del Proyecto de Fin de Grado.	4
3.1.	Ventana de login	14
3.2.	Ejemplo de Formulario	15
3.3.	Ficha del Socio	16
3.4.	Botón de Editar	16
3.5.	Borrado de Socios	17
3.6.	Ejemplo de Familia	18
3.7.	Edición de Familia	18
3.8.	Medicamentos	19
3.9.	Rango de edades	19
3.10.	Botón Cambio de Unidad	19
3.11.	Aplicación de algunos filtros al listado	20
3.12.	Busqueda ID	20
3.13.	Exportación de un listado	21
3.14.	Resultado de la Exportación	22
3.15.	Subiendo archivo csv para importar base de datos	23
3.16.	Botón Importar Base de Datos	23
4.1.	Propuesta del Grupo Aguere 70	26
4.2.	Modelo Inicial planteado en la primera reunión con los miembros de Aguere 70	27
4.3.	Panel de Control de administrador de Google	30
4.4.	Resultado del Pylint tras analizar un conjunto de ficheros con el script.	39
7.1.	Números de horas totales y de desarrollo que se llevaron a cabo en el proyecto.	45

Listings

4.1.	Archivo de configuración app.yaml	28
4.2.	Archivo settings.py, sección de autenticación	29
4.3.	Configuración del FLOW para utilizar los servicios de Google.	31
4.4.	Extracto del modelo de datos de la aplicación	32
4.5.	Vista que gestiona la visualización de los datos económicos	33
4.6.	Plantilla que muestra los datos económicos	33
4.7.	Configuración de la plantilla para usar el widget Tablesorter	34
4.8.	Sintaxis de la tabla	34
4.9.	Descripción de como se construye un fichero para exportar a Drive	36
4.10.	Función Cambio de Unidad	37
4.11.	Código del form para subir un archivo	38
4.12.	Ejemplo de listado desde archivo	38
4.13.	Script metrics.sh	38

Capítulo 1

Introducción

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

La tecnología principal usada se basa en aplicaciones web con un entorno de desarrollo de alto nivel, en nuestro caso con el Framework de Django, y también 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 misión 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 mayoría son simplemente 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 Scouts con sus tareas, que es en lo que se enfocó 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, sirvió 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, evitando tareas de gestión y administración de sistemas.

1.2. Características de la Aplicación

La aplicación esta creada para la gestión de socios Scout, donde se podrá interactuar con datos relevantes de los socios. En el Capítulo 3 se describe toda la funcionalidad de la aplicación, al igual que nos enseña como usarla.

En cuando al entorno de desarrollo tenemos como resumen el siguiente listado de tecnologías que se usaron en el desarrollo de la aplicación:

- Framework Django-nonrel [3]
- Despliegue en Google App Engine [6]
- Google APIs (Google+ [7] y Google Drive [8])
- GitHub [4]
- jQuery [11]
- JavaScript [10]
- Bootstrap [1] y CSS [9]
- Pylint [14] y Pymetrics [15]

En el Capítulo 2, se profundizará en la descripción de las tecnologías empleadas en el proyecto y en el Capítulo 4 veremos como se emplearon estas tecnologías y por que motivo se usaron en la aplicación GScout.

1.3. Actividades

El desarrollo del proyecto se organizó en las siguientes tareas:

Tarea	Actividad
Tarea 1	Análisis de la aplicación GNU Scout y el modelo de datos utilizado.
Tarea 2	Entrevistas con los responsables de un grupo Scout para analizar las funcionalidades ya previstas según y añadir/eliminar/modificar aquellas de interés.
Tarea 3	Montar un repositorio GIT para alojar los códigos del proyecto. Definir la estrategia de branching.
Tarea 4	Definir un proyecto en Pivotal Tracker para el seguimiento del proyecto. Esta herramienta facilita el desarrollo siguiendo metodologías ágiles.
Tarea 5	Desarrollo de un proyecto piloto, realizar la implantación en GAE 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 mediados de junio incluyendo los retoques finales y puesta a punto de la aplicación.

La siguiente tabla de la Figura 1.1 refleja detalladamente todas las actividades y horas empleadas en cada una de ellas durante todo el período de desarrollo del proyecto.

Fecha	Nº Actividad	Actividad de enseñanza aprendizaje	Horas de trabajo presencial	Horas de trabajo autónomo	Total
11 de septiembre de 2012	Actividad 1	Seminario de presentación de los proyectos de Trabajo de Fin de Grado a los estudiantes	2		2
17 al 21 de septiembre de 2012		Seminario preparatorio del proyecto de Trabajo de Fin de Grado	2		2
30 de enero de 2013 al 15 de junio de 2013	Actividad 2	Ejecución del proyecto del Trabajo de Fin de Grado (incluye elaboración de la memoria del proyecto)		430	430
4 al 8 de febrero de 2013	Actividad 3	Seminario de supervisión 1	1	1	2
30 de enero de 2013 al 15 de junio de 2013		Tutoría Semanal	24		24
30 de enero de 2013 al 15 de junio de 2013		Reuniones con la organización scout Aguere 70	3		3
4 al 8 de marzo de 2013	Actividad 4	Seminario de supervisión 2	1	1	2
8 al 12 de abril de 2013	Actividad 5	Seminario de supervisión 3	1	1	2
13 al 18 de mayo de 2013	Actividad 6	Seminario de supervisión 4	1	1	2
21 de mayo de 2013	Actividad 7	Taller de presentación de los Trabajos de Fin de Grado	2		2
22 al 19 de junio de 2013	Actividad 8 (convocatoria de junio)	Defensa oral del Trabajo de Fin de Grado	2	15	17
Totales			39	449	488

Figura 1.1: Tabla del Cronograma del Proyecto de Fin de Grado.

Capítulo 2

Entorno de Desarrollo

En el capítulo anterior se han introducido tanto los antecedentes como el estado actual del proyecto, se nombraron 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 (GNU Scout [5]), que poseía las funcionalidades básicas, pero el problema era que la base de datos no era compatible con nuestra tecnología (Google App Engine), 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 manager** [12], 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 específico para trabajar con bases de datos no relacionales, debido a que usamos Google App Engine para el despliegue, y esta tecnología tiene como requerimiento que solo trabaja con bases de datos no relacionales.

En un principio se había propuesto el uso de PivotalTracker para el seguimiento del desarrollo de la aplicación, pero como sólo había un desarrollador en proceso, 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 siguiendo la información de los “commit” se puede ver lo que se ha hecho.

Por otro lado el desarrollo del proyecto fue basado en metodologías ágiles, en el cual se realizaban una serie de iteraciones compuestas por diferentes fases, como las de planificación, análisis de requerimientos, diseño codificación, etc. De modo que al finalizar cada iteración se realizaba una evaluación y se repetía el ciclo una y otra vez, con el fin de

elaborar un producto que cumpla las expectativas y necesidades del cliente.

2.2. Arquitectura Software

En esta sección nos expandiremos en describir lo que es la arquitectura software de la aplicación, es decir el modelo y el framework de desarrollo.

2.2.1. Modelo Template View (MTV)

Empezaremos diciendo que la aplicación sigue el paradigma del modelo Template-View(**MTV**), que es una variante del modelo Vista-Controlador(**MVC**) que se deriva de la forma en la que está implementada el framework de Django. Por esta razón, para entender el modelo MTV, primero tenemos que tener claro lo que es el modelo MVC, el cual se define de la siguiente manera:

- **Modelo:** El modelo tiene como objetivo mapear a la base de datos de tal forma que crea una sincronización entre la base de datos y la aplicación, con el fin de mantener actualizada toda la información de las tablas, campos, y datos de nuestra base.
- **Controlador:** Responde a las peticiones y comunicaciones que existen entre los modelos y las vistas.
- **Vista:** Decide cómo se va a mostrar la información.

Visto esto, el modelo Template-Vista funciona de la siguiente manera:

- **Modelo:** Cumple la misma función que en el modelo anterior.
- **Template:** Es básicamente un archivo html que tiene como objetivo mostrar las respuestas enviadas por la vista.
- **Vista:** La vista tiene como objetivo recibir el requerimiento que es enviado por el navegador, procesar la información, si es necesario realizar una petición al modelo para extraer un valor de la base de datos, entre otras funcionalidades, para posiblemente devolver una respuesta a la template.

2.2.2. 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, pero específicamente el modelo que usa es el Modelo Template View. 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.

En la aplicación se usa una variante de este framework, **Django-nonrel**, debido a que la aplicación necesita gestionar una base de datos no relacional por requerimientos de la propia tecnología Google App Engine, no soportada por la versión normal de Django.

2.3. Herramientas de Desarrollo

En este apartado se describirán las herramientas que se usaron para el despliegue, aumento de funcionalidad, gestión de versiones y repositorio, dinamismo en las plantillas y apariencia de la aplicación.

2.3.1. Despliegue: Google App Engine

Google App Engine nos permite realizar el despliegue de la aplicación web en la infraestructura de Google, con el objetivo de no usar ningún servidor.

Las aplicaciones en App Engine son fáciles de gestionar. Podemos proporcionar nuestro propio nombre de dominio (como, por ejemplo, <http://www.example.com/>) a través de Google Apps. También podemos proporcionarle un nombre que esté disponible en el dominio appspot.com. Además podremos limitar el acceso a la aplicación a un determinado grupo de miembros.

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

Las funciones que nos brinda Google App Engine son las siguientes:

- 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.

2.3.2. Aumento funcionalidad: Google APIs

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

Google Plus

Como GAE ya posee un modulo de autenticación por medio de Google Auth, en el proyecto fue necesario modificarlo para que se hiciera por medio de Google Plus, generando unas credenciales que se guardaran en el gestor de usuarios para que la aplicación 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 datos que nos proporciona Google Plus, como nombre, apellidos, foto de perfil, dirección, etc.

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 tecnología Google, decidimos que la mejor forma era utilizar la API de Google Drive, para crear documentos en formato de hoja de calculo en el entorno de Google Drive. 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.

2.3.3. Gestión de versiones y repositorio: 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 repositorio público que se facilitara en la sección de bibliografía.

2.3.4. Dinamismo en las plantillas: JavaScript y JQuery

En ocasiones es necesario crear dinamismo en las plantillas, para que el usuario pueda desenvolverse mejor con la interfaz de la aplicación, esto los conseguimos con el uso de las dos tecnologías siguientes, JavaScript y jQuery, que además de mejorar la usabilidad de la aplicación, descargan parte de la funcionalidad de la aplicación en el navegador, liberando de esta manera al servidor de algunas tareas. A continuación de describirá cada una de estas tecnologías.

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.

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 características 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+.

2.3.5. Apariencia de la Aplicación: CSS y Bootstrap

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

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.

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.

2.4. Herramientas para Calidad de Código

Con el uso de este tipo de herramientas hacemos que nuestro código tenga mayor calidad y sea mas legible por lo demás desarrolladores. En este proyecto se usaron las siguientes:

Pylint

Pylint es una herramienta que básicamente tiene como función analizar código Python en busca de errores o síntomas de mala calidad en el código fuente. Cabe destacar que por omisión, la guía de estilo a la que se trata de apegar Pylint es la descrita en el PEP-8 [13].

Entre la serie de revisiones que hace Pylint al código fuente se encuentran las siguientes:

- Revisiones básicas:
 - Presencia de cadenas de documentación (docstring).
 - Nombres de módulos, clases, funciones, métodos, argumentos, variables.
 - Número de argumentos, variables locales, retornos y sentencias en funciones y métodos.
 - Atributos requeridos para módulos.
 - Valores por omisión no recomendados como argumentos.
 - Redefinición de funciones, métodos, clases.
 - Uso de declaraciones globales.
- Revisión de variables:

- Determina si una variable o **import** no está siendo usado.
- Variables indefinidas.
- Redefinición de variables proveniente de módulos builtins o de ámbito externo.
- Uso de una variable antes de asignación de valor.
- Revisión de clases:
 - Métodos sin **self** como primer argumento.
 - Acceso único a miembros existentes vía **self**
 - Atributos no definidos en el método **__init__**
 - Código inalcanzable.
- Revisión de diseño:
 - Número de métodos, atributos, variables locales, entre otros.
 - Tamaño, complejidad de funciones, métodos, entre otros.
- Revisión de imports:
 - Dependencias externas.
 - imports relativos o importe de todos los métodos, variables vía * (wildcard).
 - Uso de imports cíclicos.
 - Uso de módulos obsoletos.
 - Conflictos entre viejo/nuevo estilo:
 - Uso de **property**, **__slots__**, **super**.
 - Uso de **super**.
- Revisiones de formato:
 - Construcciones no autorizadas.
 - Sangrado estricto del código.
 - Longitud de la línea.
 - Uso de <> en vez de !=
- Otras revisiones:
 - Notas de alerta en el código como **FIXME**, **XXX**.
 - Código fuente con caracteres non-ASCII sin tener una declaración de encoding. PEP-263
 - Búsqueda por similitudes o duplicación en el código fuente.
 - Revisión de excepciones.
 - Revisiones de tipo haciendo uso de inferencia de tipos.

Pymetrics

Pymetrics es similar a la herramienta nombrada anteriormente, produce indicadores para los programas en Python. Estos indicadores incluyen el indicador de **Complejidad Ciclomática de McCabe** [2].

Capítulo 3

Descripción de la aplicación

En el capítulo 1 se describió brevemente la aplicación. En este capítulo explicaremos 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 genéricos, pero la aplicación esta preparada para clasificarlos según su cargo, y que cada usuario pueda desempeñar unas funciones u otras dependiendo del cargo asignado por el administrador.

En la Figura 3.1 se muestra la pantalla de acceso a la aplicación a través del login de Google.

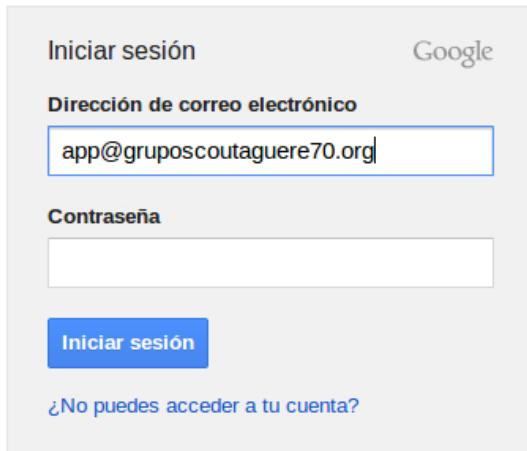


Figura 3.1: Ventana de login

3.2. Gestión de socios

Como se ha dicho a lo largo de la memoria, GScout está enfocada a la gestión de socios, concretamente Scouts del grupo Aguere 70, por tanto la aplicación brinda la posibilidad de crear socios, visualizar sus datos, incluso podemos editarlos y/o borrarlos. Además existe una funcionalidad que nos permiten crear familias dentro de la aplicación, para asociar unos socios con otros, creando de esta manera un árbol de familia, el cual nos permite tener un contacto directo con los representantes de dichos socios en caso de cualquier incidencia o necesidad. Por otro lado muchos de los socios podrían estar bajo medicación, de modo que la aplicación posee otra funcionalidad encargada de asociar medicación a los socios, en la que se señalan las pautas y dosis que se deben de seguir con cada medicamento.

A continuación se detallará cada una de estas funciones que nos proporciona la aplicación GScout.

3.2.1. Creación de Socios

Los usuarios podrán crear nuevos socios rellenando los formularios predefinidos en la aplicación, dónde se clasifican los datos en personales, económicos, familiares y médicos.

En la Figura 3.2 se puede apreciar una parte de estos formularios para la creación de socios.

The screenshot shows a web-based application interface for managing member data. At the top, there's a navigation bar with links: 'Aguere70' (highlighted in blue), 'Home', 'Nuevo Socio' (highlighted in red), 'Listados', 'Borrar Socios', and 'Editar Ficha'. Below the navigation is a title 'Datos Personales'. The form contains the following fields:

Nombre	Manuel
Apellidos	Lopez
DNI	785412Q
Sexo	Hombre
Fecha de Nacimiento	dd/mm/aaaa
Dirección	Calle La Viñita nº10
Código Postal	38300
Localidad	La Orotava
Provincia	Santa Cruz de Tenerife
Tel. Fijo	922324587
Tel. Móvil	Tel. Móvil
Sección	Esculta
Estudios	(empty)

Figura 3.2: Ejemplo de Formulario

3.2.2. Visualización y Edición de Socios

Una vez creados los socios podremos visualizar su información accediendo a la ficha de socio, donde la información se divide en 4 pestañas: Personales, Familia, Económicos y Médicos

En la Figura 3.3 se ve un claro ejemplo de como se visualizan los datos, en este caso los datos personales del socio.

Nombres:	Gerardo José	Apellidos:	Alvarez Rodríguez	DNI:	78563135B
Sexo:	none	Fecha de Nacimiento:	11-10-1985	Sección:	Scouter
Dirección:	C/ Adelantado nº 36 l 3º 3º			Código Postal:	38201
Localidad:	La Laguna	Provincia:	S/C de Tenerife		
Tel. Fijo:	922 250215	Móvil:	610347345		
Fecha de Ingreso:	16-10-1993	Fecha de Baja:	16-11-2009		
Estudios:					
Profesión:					
Deportes:					
Aficiones:					

Editar

Figura 3.3: Ficha del Socio

También se pueden editar sus datos, de una manera similar a la de creación de socios, simplemente buscamos el socio, nos situamos en la pestaña de la información que queramos editar y le damos al botón editar (botón de la Figura 3.4).



Figura 3.4: Botón de Editar

3.2.3. Borrado de Socios

También podremos borrar los socios, de momento los socios se borran permanentemente, pero en el modelo de datos esta configurado para poder implementar un método que en vez de borrarlos de manera definitiva, estos pasen a un estado de inactividad, el cual no los haría visible para cualquier usuario, sino solo para el administrador.

En la Figura 3.5 se ve la pantalla donde se seleccionan los socios que se desean borrar.

Listado de Socios (Datos Personales)									
ID	Nombre	Apellidos	DNI	Sexo	Sección	F. Nacimiento	Localidad	Provincia	
	ge			Toc ▾	Todos ▾	From To			
380700154	Gerardo José	Alvarez Rodríguez	78563135B	none	Scouter	Oct 11, 1985	La Laguna	S/C de Tenerife	
380700244	Miguel Ángel	Siverio Rodríguez		none	Scouter	Jun 05, 1988	La Laguna	S/C de Tenerife	
380700568	Angeles Daphne	Marichal Rosa	43840077f	none	Tropa	Apr 13, 2001	La Esperanza	S/C de Tenerife	
380700627	Jorge Javier	Dorta Gamez	51201055l	none	Tropa	Apr 03, 2001	La Laguna	S/C de Tenerife	
380700650	Jorge	Rodríguez Vilar	79083179W	none	Tropa	May 25, 2002	La Laguna	S/C de Tenerife	
ID	Nombre	Apellidos	DNI	Sexo	Sección	F. Nacimiento	Localidad	Provincia	
<input type="button" value="Borrar seleccionados"/>									
<input type="button" value=""/> 1 - 5 / 5 (160) <input type="button" value=""/> 10 <input type="button" value=""/> 1 <input type="button" value=""/>									

Figura 3.5: Borrado de Socios

3.2.4. Familiares

Es obligado que cada socio pertenezca a una familia, la cual tendrá un responsable que sera el identificador 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 árbol de familia entre los socios, saber que socios conviven en la misma familia, quienes son los padres/tutores del socio, etc.

Un ejemplo de familia dentro de la aplicación lo podemos ver en la Figura 3.6.

NIF	Nombre	Apellidos	Rol	Tel	Movil	Email
11405153M	Maria Nueves	Funes Lorenzo	Madre	922259601	638741642	none

NIF	Nombre	Apellidos	Sección
54116696L	Carlos Manuel	Verdugo Funes	Scouter

Figura 3.6: Ejemplo de Familia

También GScout nos da la opción de cambiar o crear una familia nueva, por cualquier circunstancia que pueda pasar, en la que se necesite modificar las familias ya existentes, como se muestra la Figura 3.7.

¿Familia nueva? No Si

Nota: El la primera persona que figure en la lista sera el identificador de la familia.

Nombre	Apellidos	NIF	¿Padre, Madre o Tutor?
Telefono	Movil	Email	<input type="checkbox"/> ¿Eliminar?

Figura 3.7: Edición de Familia

3.2.5. Medicamentos

Muchos de los socios podrían estar bajo medicación de algún tipo, por tanto existe un modulo en el que podemos anexar a los socios los datos de sus medicamentos, como el nombre, las pautas y las dosis correspondientes, como se muestra en la Figura 3.8.

Medicamentos		
Nombre	Pauta	Dosis
Ibuprofeno	2 veces al día	600mg
Jarabe para la tos	3 veces al día	50cc

Editar

Figura 3.8: Medicamentos

3.3. Cambios de unidad

A petición de la organización scout Aguere 70 se creó una función llamada **Cambio de Unidad** que consiste en analizar todos los socios, verificar su edad, y si procede cambiarlo a la sección/unidad acorde a su edad, cuyo rango de edades se muestra en la Figura 3.9.

Intervalo(Años)	Unidad
0-11	Manada
11-14	Tropa
14-17	Esculta
17-20	Rover
20+	Scouter

Figura 3.9: Rango de edades

Esta función se ejecuta al darle al botón de cambio de unidad que se encuentra en el menú principal (Figura 3.10).



Figura 3.10: Botón Cambio de Unidad

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 específico para los datos perso-

nales y otro para los datos económicos.

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

En la Figura 3.11 se muestra un ejemplo de una pantalla de listado.

Listado de Socios (Datos Personales)								
ID	Nombre	Apellidos	DNI	Sexo	Sección	F. Nacimiento	Localidad	Provincia
	Javier			Toc	Tropa	From		
380700604	Javier	Padilla Pio	54114502x	none	Tropa	Dec 26, 2002	La Laguna	S/C de Tenerife
380700627	Jorge Javier	Dorta Gamez	51201055l	none	Tropa	Apr 03, 2001	La Laguna	S/C de Tenerife
ID	Nombre	Apellidos	DNI	Sexo	Sección	F. Nacimiento	Localidad	Provincia
◀ ▶ 1 - 2 / 2 (160) ▶ ◀ 10 ▼ 1 ▼								
Export								

Figura 3.11: Aplicación de algunos filtros al listado

3.5. Búsquedas por ID

También se pueden realizar búsquedas directas por el ID del socio, como se muestra en la Figura 3.12, en la que nos redirecciona directamente a la ficha técnica del socio.

Busqueda de Socio

Figura 3.12: Busqueda ID

3.6. Exportaciones

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

imágenes:

En la Figura 3.13 se muestra el listado que se desea exportar.

Listado de Socios (Datos Personales)									
ID	Nombre	Apellidos	DNI	Sexo	Sección	F. Nacimiento	Localidad	Provincia	
				Toc ▾	Manada ▾	12/01/2003 09/30/2004			
380700586	Nerea	Martín Vazquez	51166855C	none	Manada	Dec 28, 2003	La Laguna	S/C de Tenerife	
380700641	Victor	Rodríguez Radius	54060666V	none	Manada	Sep 08, 2004	Guamasa La Laguna	S/C de Tenerife	
380700643	Tomás	Tamayo Bolaños	79099190M	none	Manada	Aug 13, 2004	Santa Cruz de Tenerife	S/C de Tenerife	
380700640	Sara	Sánchez de León	51165729K	none	Manada	Sep 08, 2004	La Laguna	S/C de Tenerife	
380700659	Valeria Valentina	Almarza Rodríguez	Y0691050Y	none	Manada	Feb 18, 2004	La Laguna	S/C de Tenerife	
ID	Nombre	Apellidos	DNI	Sexo	Sección	F. Nacimiento	Localidad	Provincia	
⏪ 1 - 5 / 5 (160) ⏩ 10 ⏴ 1 ⏴									
Export									

Figura 3.13: Exportación de un listado

Al darle al botón exportar, se crea un documento en la cuenta de Drive del usuario con los datos que se exportaron, como se muestra en la Figura 3.14.

Datos_Personales_10-06-2013 ☆

Archivo Editar Ver Insertar Formato Datos Herramientas Ayuda Última modificación hace 2 minutos

Comentarios Compartir

Fx ID

ID	Nombre	Apellidos	DNI	Sexo	Sección	F.Nacimiento	Localidad	Provincia
1	Nerea	Martin Vazquez	51166855	none	Manada	28/12/200	Laguna Guamasa	S/C de Tenerife
2	Victor	Rodriguez Radius	54060666	none	Manada	8/09/2004	La Laguna Santa Cruz de Tenerife	S/C de Tenerife
3	Tomas	Tamayo Bolanos Sanchez de Leon	79099190	none	Manada	13/08/200	La Laguna	S/C de Tenerife
4	Sara	Valeria Almarza Rodriguez	51165729	none	Manada	8/09/2004	La Laguna	S/C de Tenerife
5	Valentina		Y0691050	none	Manada	18/02/200	Laguna	S/C de Tenerife
6								
7								
8								
9								
10								

Añade 20 filas más al final.

Figura 3.14: Resultado de la Exportación

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, por ejemplo con la herramienta **MDB Tools**. Finalmente con el fichero csv con los datos de la base de datos de la organización, solo haría falta subir el fichero a la aplicación, que de lo demás se encarga la propia aplicación de manera automática. De esta manera tendremos los datos de la antigua base de datos en la nueva base de datos que pertenece a GScout.

En la Figura 3.16 se ve un ejemplo del proceso de importación.

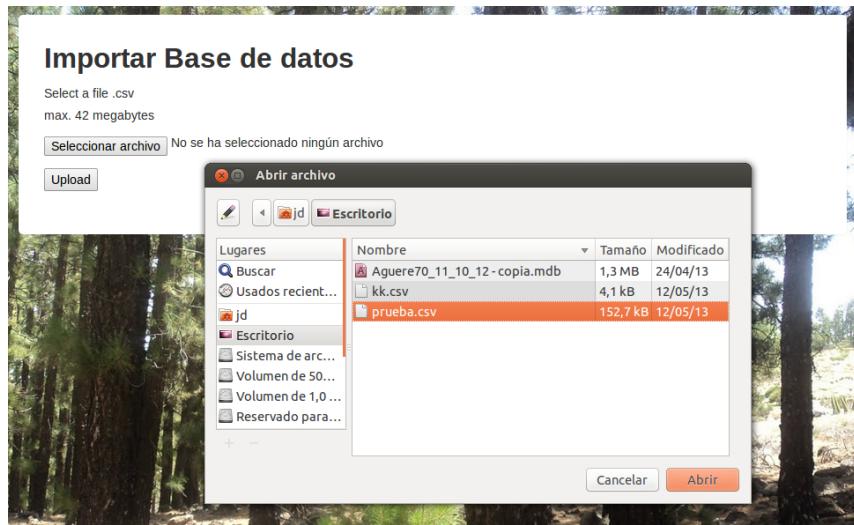


Figura 3.15: Subiendo archivo csv para importar base de datos

Esta función se ejecuta al darle al botón de la Figura 3.16 que se encuentra en el menú principal.

Importar Base de Datos

Figura 3.16: Botón Importar Base de Datos

Capítulo 4

Desarrollo de la aplicación

En el capítulo 3 se describieron las funcionalidades de la aplicación. A continuación hablaremos de todo lo que conllevó el proceso de elaboración del proyecto, 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 manager** [12], 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, además de hacer pruebas con la tecnología de Google App Engine e incorporarlas al framework de django-nonrel.

Después de estos pasos previos, se realizó 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 primera reunión con los interesados se estableció un esquema compuesto por 7 aplicaciones, la cual una de ellas era la principal (**App Socios**) y de esa dependerían el resto. El esquema inicial de aplicaciones se muestra en la Figura 4.1.

Aplicación	Enfoque
Socios	Mantenimiento de socios Cambios de Unidad Informes (Familiares, Unidades, Médicos, Personales) Exportaciones
Lista de Espera	Entradas Propuestas de entrada a grupo Incidencias
Biblioteca	Mantenimiento Préstamos
Intendencia	Mantenimiento Préstamos Informes(Prestamos y 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

Figura 4.1: Propuesta del Grupo Aguere 70

Visto las siguientes aplicaciones se comprobó que el proyecto era ambicioso y extenso, por tanto no se podía acaparar todo en este corto período de tiempo para el desarrollo del proyecto de fin de grado. De modo que simplificamos funcionalidad, con el fin de ocuparnos de las aplicaciones más importantes. Lo que originó el esquema de la Figura 4.2

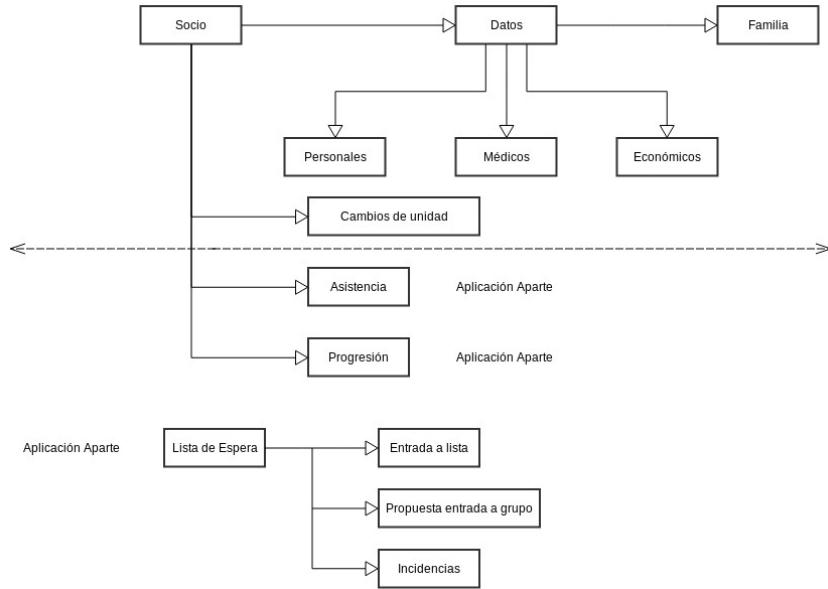


Figura 4.2: Modelo Inicial planteado en la primera reunión con los miembros de Aguere 70.

En la Figura 4.2 se puede apreciar este modelo aplicaciones inicial. En un principio se tenía previsto la realización de dos aplicaciones, una para la gestión de **socios** y otra para la gestión de la **lista de espera**. Esta última consistía en gestionar las solicitudes de los futuros socios, como también facilitar a la aplicación Socio, los datos personales de los socios, una vez se proceda a la creación definitiva del mismo. Pero al final se descartó su desarrollo por lo nombrado anteriormente, por lo tanto, nos enfocamos sólo en la app de Socios, que era la más importante, que a su vez se simplificó, omitiendo los módulos de **Asistencia** y **Progresión** del socio, ya que se clasificaron como dos aplicaciones más del proyecto fuera de la app Socios.

De modo que nos quedó solo por desarrollar la aplicación **Socio**, destinada a la gestión de los mismos. Para el desarrollo del modelo de datos se nos facilitó la ficha de inscripción (en formato papel), que han de cumplimentar los socios para inscribirse en la organización. Hablaremos de este modelo detalladamente en el Apartado 4.5.

4.3. Preparación para el proyecto

En primer lugar 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 [3] 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.

Por otro lado es importante hablar del archivo de configuración **app.yaml**, que especifica la manera en la que las rutas de URL se corresponden con los controladores de solicitudes y con los archivos estáticos en el entorno de Google App Engine. También debe contener información sobre el código de la aplicación como, por ejemplo, el ID de la aplicación, lenguaje de programación, el identificador de la última versión y el directorio de los archivos estáticos.

De modo que el archivo seria el que aparece en el Listado 4.1.

```
application: ctstprueba
version: 1
runtime: python27
api_version: 1
threadsafe: yes

builtins:
- remote_api: on

inbound_services:
- warmup

libraries:
- name: django
  version: latest

handlers:
- url: /_ah/queue/deferred
  script: djangoappengine.deferred.handler.application
  login: admin

- url: /_ah/stats/.*
  script: djangoappengine.appstats.application

- url: /media/admin
  static_dir: django/contrib/admin/media
  expiration: '0'

- url: /static
  static_dir: static
```

```
- url: /.*
  script: djangoappengine.main.application
```

Listing 4.1: Archivo de configuración app.yaml

4.4. Inicio de sesión

Como se ha dicho en el capítulo 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 como muestra el listado 4.2.

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

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

.

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

Listing 4.2: Archivo settings.py, sección de autenticación

Donde se puede apreciar las **INSTALLED APPS** 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 través 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 está seleccionado en **AUTHENTICATION BACKENDS** el backend de **gaeauth** y seguidamente elegimos en **ALLOWED DOMAINS** el dominio en el cual queremos restringir el acceso, en nuestro caso solo los usuarios pertenecientes al dominio **gruposcoutaguere70.org** podrán acceder a la aplicación.

- **plus:** Este es el modulo que nos permite obtener la información personal, foto de perfil y demás datos 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 módulos que más dio problemas. Primero había 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 generaría una serie de datos que podríamos introducirlos en nuestra aplicación uno a uno o bien en formato JSON, este paso es importante ya que sino la aplicación no podría acceder a los servicios que nos proporciona la API.

A parte de esto como se utiliza las API de Google Plus para la obtención de datos, es necesario que antes de iniciar sesión el usuario posea un perfil en Google Plus. Por defecto este servicio esta deshabilitado cuando se crea un dominio dentro de Google, por tanto se necesita de una cuenta administradora del dominio para activar este servicio.

Importante: Antes de activar el servicio de Google Plus es necesario activar el servicio de Talk.

En la Figura 4.3 se muestra que ambos servicios están activos en el dominio **gruposcoutaguer70.org**.

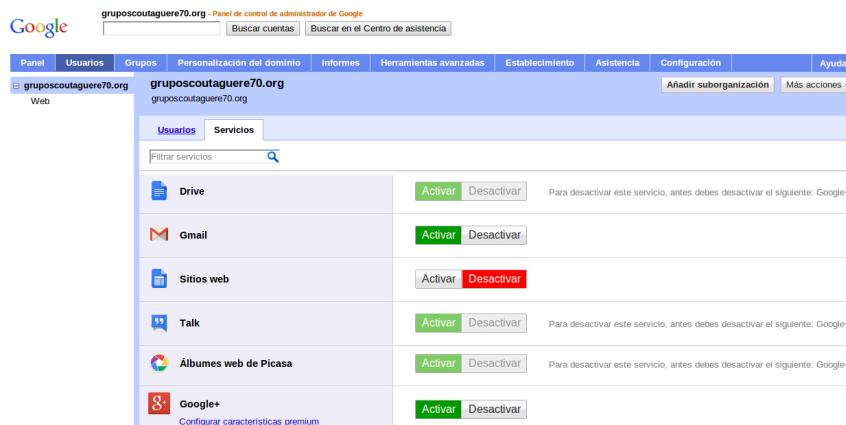


Figura 4.3: Panel de Control de administrador de Google

Dicho esto, una vez que el usuario inicia sesión, en el proceso se accede a la vista de la app **plus**, que está cargará la configuración de Google API por medio del mencionado archivo JSON o introduciendo una a una como muestra el Listado 4.3 la configuración necesaria para utilizar el servicio de Google+:

```
FLOW = OAuth2WebServerFlow (
    client_id = '480482329789-dj...',
    client_secret = 'C5umC7xa.ML704wnmdT79Bh0',
    token_uri='https://accounts.google.com/o/oauth2/token',
    scope= [ 'https://www.googleapis.com/auth/drive',
             'https://www.googleapis.com/auth/plus.me'],
    redirect_uri= 'http://localhost:8000/oauth2callback',
    access_type='offline',
    approval_prompt='force')
```

Listing 4.3: Configuración del FLOW para utilizar los servicios de Google.

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 ejecute 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 más adelante), en **redirect-uri** establecemos la dirección de retorno a nuestra aplicación por medio de la llamada oauth2cabllback que la incorpora el módulo plus.

Con todo esto lo que logramos es que en el inicio de sesión se establezca una conexión entre el usuario y las APIs de Google, en la cual se acepten una serie de permisos para posteriormente generar una credencial, que se guardará en el usuario, con la cual podrá 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 modificó para que se pudieran guardar las credenciales, además se añadió el campo de **cargo**, donde se le asignará a cada usuario un cargo específico, por tanto las funciones que podrá realizar en la aplicación estarán supeditadas a dicho cargo.

Nota: las credenciales suelen expirar a la hora por tanto, una vez expiradas el usuario no podrá acceder a la aplicación, al menos que se actualicen. Una forma de automatizar esto es que en cada inicio de sesión se restablezcan 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 refresco 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 creación 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 plantillas, formularios, etc, para que luego en la vista de la app de **Socios** por medio de diferentes llamadas POST, se fueran construyendo los objetos necesarios para la creación de un nuevo socio, cada objeto almacenaba un tipo de información específica según el modelo de datos. Por tanto un extracto de lo que seria el modelo de datos y la sintaxis se refleja en el Listado 4.4.

```

from django.db import models

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)

```

Listing 4.4: Extracto del modelo de datos de la aplicación

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 única forma de en cierto modo comunicar/unir una tabla con otra, es con claves foráneas(ForeignKey), de esta manera podremos asociar las tablas que nos interesen de manera unidimensional.

Cada tabla posee un nombre característico que define para que se usa y el tipo de información que guarda, se dejó elaborada, aunque no se usa todavía la tabla de **Autorizaciones** para que en un futuro los socios almacenen ahí las posibles autorizaciones/permisos cuando se realice una excursión, uso de fotografías de manera pública, etc.

Retomando a la creación de usuarios es necesario llenar todos los campos importantes de cada objeto 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 dónde es procesado y si todo está 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 plantillas 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 plantillas eran similares, salvo que tienen trozos de código en Javascript y variables que se pasaban de la vista a la plantilla 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 llenar todos los campos restantes.

Los archivos de plantillas 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 más simple es una vez obtenido el ID del socio, se accede a las páginas de información del socio, dónde cada plantilla está enfocada a una determinada tabla del modelo de datos, de esta forma los datos los clasificariamos en personales, económicos, médicos y familiares. En la vista (Listado 4.5) lo único que se hace es volcar los datos del socio a la plantilla (Listado 4.6), que es la que se encarga de la visualización de los receptivos datos.

```
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': economicos})
```

Listing 4.5: Vista que gestiona la visualización de los datos económicos

```
{% 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_sucursal}}-{{economicos.dc}}-{{economicos.n_cuenta}}
</p>
<a class="btn btn-large btn-primary"
    href="/socios/{{ economicos.socio_id.n_asociado }}/edit_economicos">
    Editar
</a>
{% endblock %}
```

Listing 4.6: Plantilla que muestra los datos económicos

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 esté, es decir existen varios listados para representar los datos de los socios, listados de datos personales, económicos, etc.

Para el formato de la tabla se encontró un widget con la apariencia de Bootstrap para mantener la esencia 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 scripts que se muestran en el Listado 4.7.

```
<!-- 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/themes/cupertino/jquery-ui.css">
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.0/jquery-ui.min.js">
</script>

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

Listing 4.7: Configuración de la plantilla para usar el widget Tablesorter

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

Posteriormente la sintaxis de la tabla se muestra en el Listado 4.8.

```
<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>
<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 los socios. -->
<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 4.8: Sintaxis de la tabla

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 se realiza de forma similar que con las credenciales que se han comentado en la sección **Inicio de Sesión 4.4**, añadiendo en el campo de **scope** la ruta al servicio Drive antes de generar la credencial y también en la página de administración de las APIs de Google, activar la API **Google Drive SDK**.

Ahora bien, los filtrados de los listados se realizan sobre la plantilla en el cliente, por tanto si queremos exportar un listado filtrado tenemos que en cierto modo, realizar el filtrado en el lado del servidor. Esto se logró modificando los archivos del **tablesorter** para que cuando se generara la tabla, cada input de filtrado tenga un nombre (**name**), el cual se utiliza cuando se realiza una llamada POST, para poder obtener los parámetros de filtrado y realizar el filtrado directamente con el modelo de datos.

Una vez realizado el filtrado en el lado del servidor, nos queda el paso intermedio para poder exportar los datos a Google Drive, ya que para exportar datos necesitamos crear un fichero de extensión “.csv” para que la API de Google Drive lo pueda interpretar y procesar. De modo que la construcción del fichero se puede apreciar en el Listado 4.9

```
#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 csv
datos="ID, Titular, NIF Titular, ... "
for d in range(len(lista)):
    datos += str(socios[lista[d]][ 'socio_id_id ']) + "," ...
# media_body preparara 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 body, mediabody (que tendra el contenido del fichero) y
# convert=True para obligar la conversion del fichero.
file = drive_service.files().insert(body=body, \
                                      media_body=media_body, \
                                      convert="true").execute()

pprint.pprint(file)
```

Listing 4.9: Descripción de como se construye un fichero para exportar a Drive

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 aparece en el Listado 4.10.

```

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_nacimiento.strftime('%Y')) < 14):
            s.seccion="Tropa"

        if ((date - int(s.f_nacimiento.strftime('%Y'))) >= 14) and
           (date - int(s.f_nacimiento.strftime('%Y')) < 17):
            s.seccion="Esculta"

        if ((date - int(s.f_nacimiento.strftime('%Y'))) >= 17) and
           (date - int(s.f_nacimiento.strftime('%Y')) < 20):
            s.seccion="Rover"

        if (date - int(s.f_nacimiento.strftime('%Y'))) >= 20 :
            s.seccion="Scouter"

    s.save()

```

Listing 4.10: Función Cambio de Unidad

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 está toda la información en una sola tabla, por tanto habría 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 **Access**, 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 había una herramienta llamada **MDB Tools** que convertía los ficheros Access en csv.

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

Esta app tiene un fichero **forms.py** (Listado 4.11) que contiene un formulario específico 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 4.11: Código del form para subir un archivo

Este formulario lo procesa la plantilla y posteriormente con la llamada de POST se envía el archivo al servidor, que se manipulara como indica el Listado 4.12.

```
# Con la lladma 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 comando .next() la omitimos.
portfolio.next()
```

Listing 4.12: 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 ambigüedades, ni campos vacíos que sean importantes en cada tabla del modelo.

4.12. Calidad de Código: Pylint y Pymetrics

Para finalizar el período de desarrollo del proyecto, mejoramos la calidad de código de los principales archivos escritos en Python de la aplicación. Para ello creamos un script llamado **metrics.sh** (Listado 4.13).

```
#!/bin/bash
# Borra los reportes existentes y crea de nuevo la carpeta vacia
rm -rf pymetrics/
mkdir pymetrics
rm -rf pylint/
mkdir pylint

# Indicadores para el Modulo Socio
echo "Generating metrics for User module..."
pymetrics -i simple:SimpleMetric,mccabe:McCabeMetric
    socios/views.py socios/models.py > pymetrics/Socios.txt
pylint -d C0103,E1101,C0301 -f html socios/ > pylint/Socios.html
```

```
# Borra los archivos temporales
echo "Cleaning temp files..."
rm -f metricData.*
```

Listing 4.13: Script metrics.sh

Este script crea dos directorios dentro del directorio raíz de la aplicación (/pymetrics y /pylint), los cuales guardan los resultados que se obtiene al ejecutar ambas herramientas.

Se logró obtener una nota media en los archivos principales de 8.70 en cuanto a calidad de código (Figura 4.4).

Global evaluation

Your code has been rated at 8.70/10 (previous run: 8.70/10)

Figura 4.4: Resultado del Pylint tras analizar un conjunto de ficheros con el script.

Capítulo 5

Conclusiones y trabajos futuros

Mis experiencias a la hora de elaborar el Trabajo de Fin de Grado han sido muy satisfactorias, a pesar de que al principio (antes de empezar con el proyecto) tenía miedo de como afrontarlo. Al estar acostumbrado a trabajar en grupo y ayudarnos mutuamente para salir del paso ante un problema, el hecho de hacerlo yo sólo me angustiaba. No fue el caso, con ayuda del tutor, y varias horas de investigación e indagación buscando soluciones a los problemas propuestos hicieron que el proyecto saliera adelante.

Por otro lado el proyecto que elegí, el de crear una aplicación para la gestión de grupos Scout, me pareció muy completo, a lo largo del proyecto se tocan muchos contenidos que se han impartido en la carrera de Grado en Informática. Dentro de lo que es la arquitectura de software, se utiliza el modelo de Template-View que nos ofrece Django. Trabajamos con una versión peculiar, Django-nonrel, para poder utilizar base de datos no relacionales ya que el despliegue se efectuará con Google App Engine, y esta tecnología solo soporta este tipo de base de datos, basadas en el modelo BigTable. A parte de esto se utilizaron también integración con APIs de Google, como Google+ para autenticación y obtención de datos personales de los usuarios, y Google Drive para extender las funciones de la aplicación y poder exportar datos a documentos de hoja de cálculos en Drive. Además, en el diseño web se utilizan hojas de estilo CSS para las personalizaciones, fragmentos y códigos en Javascript y JQuery para crear dinamismo en la apariencia de interfaz de las páginas web, a la vez que descargan algunas funcionalidades en el cliente.

Otra cosa que me pareció interesante, es que el proyecto se está elaborando para cumplir las expectativas y necesidades de una organización Scout real, por tanto teníamos que realizar varias reuniones con ellos, para obtener requisitos, mostrar avances, adaptarlo al cliente, volver a enfocar el proyecto si fuese necesario, etc. Esto me aporto mucha experiencia a la hora de ver más o menos cómo son los ciclos de desarrollo de software en un entorno real.

En cuanto a trabajos futuros, como bien se ha comentado en los capítulos anteriores, el desarrollo de la aplicación se enfocó en cubrir las funciones básicas referentes a la gestión

de los socios scout, pero conservando una visión de futuro con el fin de ir ampliando las funcionalidades de GScout para satisfacer las necesidades que en un principio la organización Scout Aguere 70 nos dio, como por ejemplo incorporar módulos para la tesorería, biblioteca, listas de espera, etc.

Por lo tanto podremos decir que GScout es una aplicación abierta y está totalmente preparada para añadirle más funcionalidades, con sólo el hecho de anexar módulos a la aplicación.

Capítulo 6

Summary and Conclusions

My experiences during the time of preparing this Master Thesis have been very successful. At the beginning (before starting the project) I was afraid how to face it. I used to work in groups and with the help of each other we can address any problem. Now I have to do it by myself. With the assistance from my tutor, and several hours of research and investigation looking for solutions to the proposed problems caused the project to go on forward.

On the other hand, the project I chose, developing an application for managing Scout groups, is very complete. During the project development, touched many contents that were taught in Computer Science degree. Regarding the project features, I want to mention this is a typical MVC application. As we used Django as the developing framework, our design was conducted in a MTV (model-template-view) software architecture: a small variation of the MVC software architecture model. We work with a tunned version of the framework, Django-nonrel, to use non-relational database because the deployment is made on Google App Engine. This technology only supports non-relational databases, based on the BigTable model. We also use integration with Google APIs like Google+ for authentication and access to users personal data, and Google Drive to extend application functionality, exporting data to spreadsheet documents in Drive. Regarding web design, we used CSS for customizations, and code fragments in JavaScript and jQuery to create dynamism in the interface of the website, while deploying some features on the client.

Another thing I found interesting is that the project is being developed to achieve the requirements and needs of a real Scout organization. I have the oportunity to have several meetings with them, to collect requirements, show progress, adapt to the client, refocus the project if necessary, etc. This me I bring a lot of experience when it comes to see more or less how are the cycles of software development in a real environment.

As it has been commented in future work chapter, the application development is focused on achieving the basic functionality concerning scout members management, while retaining a vision in order to gradually increase GScout functionality to meet the needs

that originally Aguere 70 Scout organization gave us. For example, modules to incorporate the treasury, library, waiting lists, etc.

Therefore, we can say that GScout is an open application, fully prepared to add more functionality, with only the fact of annexing modules to the application.

Capítulo 7

Presupuesto

La siguiente tabla de la Figura 7.1 nos indica cuantas horas de trabajo e investigación abarcó cada características o funcionalidad de la aplicación, al igual que el número total de horas (346 horas) que duró la elaboración del proyecto incluyendo ambos procesos.

Funcionalidad	Horas de trabajo totales (Investigación y Desarrollo)	Porcentaje de Investigación (%)	Horas de Implementación
Usuarios (Inicio de sesión, credenciales, cargos, etc.)	50	60	20
Módulo de Gestión de Socios	Creacion de Socios	35	50
	Visualización de Socios	25	40
	Edición de Socios	20	50
	Borrado de Socios	5	20
	Familiares	30	40
	Medicamentos	30	40
	Cambios de Unidad	15	50
	Listados de Información	20	60
	Busquedas por ID	4	10
	Exportaciones a Drive	35	60
	Importaciones de la Base de datos Antigua	32	60
	Calidad de código, interfaz y apariencia de la aplicación	45	60
Total	346	Total de horas exclusivas de Desarrollo	166,4

Figura 7.1: Números de horas totales y de desarrollo que se llevaron a cabo en el proyecto.

Por otro lado, destacamos el porcentaje de investigación que tuvo cada funcionalidad, de esta manera podemos ver que funcionalidad fue más complicada de desarrollar según su porcentaje de investigación. Y para finalizar se descontaron estas horas de investigación para obtener el número neto de horas de desarrollo de cada funcionalidad, cuya suma será el total de horas exclusivas de desarrollo, que son las horas que se le facturarán al cliente (166,4 horas).

Dicho esto la tarifa de desarrollo establecida en este proyecto es de **12 euros** la hora.

Por tanto la aplicación **GScout** tendrá un precio de: **1996,90 euros**

Bibliografía

- [1] Bootstrap. <http://twitter.github.io/bootstrap/>.
- [2] Complejidad Ciclomática de McCabe. <http://cnx.org/content/m17455/latest/>.
- [3] Django-nonrel con Google App Engine. <http://www.allbuttonspressed.com/projects/djangoappengine>.
- [4] GitHub (Repositorio). <https://github.com/Jalcubo/GScout>.
- [5] GNU Scout (Implementación en PHP). <http://sourceforge.net/projects/gnuscout>.
- [6] Google App Engine. <https://developers.google.com/appengine/?hl=es>.
- [7] Google Plus API. <https://developers.google.com/+/api/>.
- [8] Google Plus SDK. <https://developers.google.com/drive/>.
- [9] Hojas de estilo CSS. <http://www.w3schools.com/css/>.
- [10] JavaScript Tutorial. <http://www.w3schools.com/js/>.
- [11] jQuery. <http://jquery.com/>.
- [12] Online Scout Manager. <https://www.onlinescoutmanager.co.uk/>.
- [13] PEP-8. Guía de estilo para códigos en Python. <http://www.python.org/dev/peps/pep-0008/>.
- [14] Pylint Tutorial. <http://docs.pylint.org/tutorial.html>.
- [15] Pymetrics Tutorial. <http://mindref.blogspot.com.es/2010/06/python-pymetrics.html>.