Escuela Universitaria de Ingeniería Técnica Industrial de Bilbao

Grado en Ingeniería Informática en Gestión y Sistemas

Trabajo fin de grado

2015/2016

Proyecto Akeko

Memoria del Trabajo Fin de Grado

Datos de la Alumna o del Alumno	Datos del Director o de la Directora
Rubén Mulero Martínez	**Mikel Villamañe Gironés**
Fdo:	Fdo:
Fecha:	Fecha:

Resumen

Fruto de muchas horas de investigación, de estudiar muchos lenguajes de programación y sobre todo, de poner un gran empeño, se presenta un gran proyecto nacido de una humilde forma de hacer las cosas.

Éste trabajo fin de grado ha consistido en la consecución del estudio de 3 lenguajes de programación diferentes, del aprendizaje de nuevas formas distintas de programar, de saber hacer una búsqueda de información, de escuchar al usuario para obtener una mejor adaptación visual, de encontrar los diferentes caminos que llevan a una solución, de entender que cada lenguaje tiene su forma de pensar y de hacer y sobre todo, de la dificultad que acarrea integrar una aplicación compleja en varios entornos operativos con distintos tipos de arquitecturas.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Planteamiento inicial

1.1. Introducción

Hoy en día, ser profesor de informática es duro. Ya no es cuestión simplemente de realizar la tarea educadora, si no de tener preparadas las herramientas necesarias para poder facilitar dicha labor.

Las tecnologías cambian y cada día aparecen más, los alumnos deben hacer uso de diferentes tipos de herramientas para poder practicar y sobre todo, para poder adquirir los conocimientos necesarios para poder desarrollarse como personas.

El uso de diferentes herramientas trae un problema. Dicho problema es la creación de usuarios y contraseñas para que cada alumno pueda acceder a éstas y puedan desarrollar su labor. Ésta tarea, resulta un tanto tediosa si por cada asignatura un profesor tiene un elevado número de alumnos.

¿Cuántas veces se ha podido ver la situación en la que un profesor debe crear por cada alumno entradas a más de un servicio distinto para cada asignatura? Seguramente muchas y ésto, al final, acarrea un problema serio.

Para ello, el profesorado necesita una herramienta, un sistema capaz de automatizar el proceso de creación de usuarios en los distintos servicios para hacer un poquito más sencilla la labor educadora.

1.1.1. Propósito

El propósito fundamental de éste trabajo fin de grado es la creación de una herramienta que permita a un profesor poder gestionar los distintos servicios a los que puede acceder un alumno.

Entendemos como servicios, aquellas herramientas que permiten desarrollar de forma efectiva las competencias necesarias en cada asignatura. Por ejemplo, servicios como bases de datos “MySQL” o servicios de control de versiones como “GIT”.

1.1.2. Ámbito

El proyecto está enfocado al profesorado de la UPV/EHU, el cual debe disponer al menos de unos mínimos de conocimientos informáticos para poder hacer un uso correcto de la herramienta.

La parte administrativa deberá ser gestionada por un administrador competente que tenga conocimientos avanzados del uso de “GNU/Linux” y “bashscripting”.

La creación de la parte administrativa y de la parte del usuario, será competencia directa del ingeniero informático, el cual deberá diseñar y crear los componentes necesarios para que los administradores y profesores puedan usar la herramienta sin dificultad alguna.

1.1.3. Contexto de negocio

Ésta aplicación está exclusivamente realizada para pueda ser usada por el profesorado de la UPV/EHU por lo que el uso en otros ámbitos no es posible debido a que está preparada para un fin muy determinado.

El intento de poder usar la aplicación en otros contextos podría ser factible, siempre y cuando se hagan unos pequeños ajustes en la aplicación para adaptarla de forma adecuada y que el uso final sea la gestión de alumnos.

1.1.4. Razones de la elección del TFG

La elección de éste proyecto ha sido motivada por una buena serie de razones. Debido a que son bastantes, se han plasmado únicamente las más determinantes.

- Poder realizar una aplicación en "GNU/Linux". Una de las premisas del proyecto es la gestión de los usuarios haciendo uso del sistema operativo del pingüino, lo que a nivel personal del desarrollador resulta una buena manera de profundizar más en los conocimientos que ya se tienen sobre éste sistema.

- La capacidad de crear algo nuevo. El TFG da carta blanca para crear el sistema como más guste, no existen limitaciones específicas o la necesidad de hacer uso de determinadas herramientas. La elección y hasta donde se desea llegar, es una puerta abierta que no todos los TFG son capaces de ofrecer.

- La posibilidad de crear algo propio. Una de las ventajas del TFG es la capacidad de saber que lo que se está creando es algo nuevo y que no se tiene la necesidad de ajustarse a algo que se ha hecho anteriormente por otra persona. Uno es dueño de lo que crea y da un grado de libertad de no sentir las ataduras de recorrer un camino impuesto por una tercera persona.

- Es software libre. "GNU/Linux" es sinónimo de software libre, una filosofía con la cual el desarrollador se siente identificado, por lo que da un buen empujón psicológico que ayuda a querer crear algo nuevo sabiendo que tiene unas bases con un cierto grado de moralidad de cara hacia el usuario final.

- Podría ser una realidad en un futuro. Gracias a que es una aplicación para "GNU/Linux" y gracias a las bondades del software libre, una de las cuestiones positivas es la apertura de una puerta que permita en un futuro, realizar modificaciones al proyecto para convertirlo en una herramienta que pueda ser usada en un entorno real y permita solventar situaciones muy específicas.

- Sirve de referencia para futuros TFG. El proyecto sienta unas bases que permiten que otros alumnos, puedan apoyarse para despegar otros TFG que expandan o complementen las funcionalidades desarrolladas.

1.1.5. Definiciones, acrónimos, abreviaturas

- **MySQL:** *My Structured Query Language* consiste en una base de datos relacional multiusuario y multihilo creada por "MySQL AB" en 1995. La idea consistía en crear una potente base de datos de código abierto que fuese un referente dentro del mundo informático. MySQL ofrece dos tipos de licencia, una de pago para los proyectos comerciales y una gratuita bajo la licencia "GPL" para apoyar la creación de software libre. En el año 2008, la empresa fue comprada por "Sun Microsystems"(creadores de Java) que a su vez, fueron comprados por "Oracle Corporation" en 2009. El impacto de la compra por parte de "Oracle" propició la creación de variaciones del proyecto(llamados *forks*) para desligarse del camino de la empresa comercial y mantener la filosofía de código abierto con la que fue creada. [12]

- **SQL:** o *Structured Query Language* consiste en un lenguaje estandarizado para realizar diversas consultas y obtener información en una base de datos. La version original llamada SEQUEL (*Structured English Query Language*) fue diseñado por IBM en 1974 y 1975. SQL fue introducido de forma comercial por vez primera de la mano de "Oracle Corpotarion" en 1979. [21]

- **GIT:** Es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones.

- **UPV/EHU:** Universidad del País Vasco / Euskal Herriko Unibertsitatea.

- **Bashscripting:** Consiste en la programación de scripts haciendo uso de "Bash".

- **Bash:** *Bourne again shell* es un programa informático cuya función consiste en interpretar órdenes. Dichas órdenes están basadas en el intérprete de comandos del sistema Unix.

- **Unix:** Es un sistema operativo portable, multitarea y multiusuario desarrollado por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie y Douglas McIlroy. [17]

- **Python:** Es un lenguaje de programación interpretado, multiplataforma, con tipado y fuertemente tipado que apela a la creación de una sintaxis legible y ordenada. Su principal característica es que fuerza al usuario a realizar una indentación del código(colocar los bloques de texto debidamente ordenados) para conseguir que éste sea perfectamente legible y sobre todo funcional. Python es una lenguaje multiparadigma por lo que puede ser usado por la programación orientada a objetos, por la programación imperativa, por la programación orientada a funciones y por la programación orientada a aspectos. Fue creado por Guido van Rossum en 1991 y su nombre fue atribuido por los cómicos ingleses "Monty Python".[27]

- **QT:** Consiste en un potente *framework* con una serie de herramientas avanzadas que permite la creación de interfaces de usuario multiplataforma y nativas para cada sistema operativo. QT fue creado en 1994 y hoy en día es usado en todo tipo de plataformas, ya sean de escritorio o móviles. [11]

- **IDE:** *Integrated Development Environment* o *Ambiente de Desarrollo Integrado* consiste en una potente aplicación informática que permite a un desarrollador crear y editar nuevos programas informáticos a través de una serie de herramientas gráficas e intuitivas. Un IDE permite generar y ordenar proyectos de programas informáticos facilitando su gestión y proporcionando ciertas utilidades que ayudan en la sintaxis y compilación del código generado. Algunos IDE contienen incluso herramientas que ayudan a la generación de diagramas de clases para identificar y ordenar la programación orientada a objetos.

- **Librería:** Una librería o biblioteca es un Kit de herramientas software pequeño y autónomo que ofrece una funcionalidad muy específica al usuario. Normalmente se usa junto con otras librerías y herramientas para hacer una aplicación completa ya que, por lo general, las bibliotecas no son ejecutables pero pueden ser usadas por ejecutables que las necesiten para poder funcionar. [6]

- **Binding:** Consiste en la asignación de un o una serie de comandos Unix a un acceso directo, normalmente un acceso por teclado.

- **GPL:** *General Public License*, consiste en una licencia gratuita y libre para licenciar software con el amparo de la "Free Software Foundation", una fundación que busca la creación y distribución de software libre por todo el mundo. Dicho software debe de ser libre (que no gratuito) y su código debe ser fácilmente accesible al resto de las personas que deseen verlo. La idea de esta licencia es crear software que sea libre y que se pueda compartir con otro software libre y se puedan generar interesantes proyectos en un futuro. [29]

- **Latex:** Creado a principios de los 80 por Leslie Lamport en SRI International, LaTeX es un lenguaje de alto nivel (más fácil de usar para los usuarios) para acceder al poder de TeX, que es de muy bajo nivel para poder realizar documentos. Se basa en que el autor se concentre en el contenido de lo que escribe, en lugar de la presentación visual. De ésta forma, éste último decide de forma libre la estructura lógica que debe de poseer el documento a escribir. [1]

- **C++:** El lenguaje de programación C++ es un lenguaje orientado a objetos creado por Bjarne Stroustrup en 1980. La intención del creador era extender la funcionalidad y mecanismos del lenguaje original C dotándole de manipulación de objetos y permitiendo que el lenguaje fuese híbrido para poder tener la elección de programar haciendo uso del "estilo C" o del "estilo orientado a objetos". El nombre de C++ fue concedido tres años después para representar el incremento de la funcionalidad del lenguaje original C. C++ es un lenguaje multiparadigma que permite la programación genérica, la programación estructurada y la programación orientada a objetos. [34]

- **Diagrama de Gantt:** Desarrollado por Henry Laurence Gantt a principios del siglo XX, el diagrama de Gantt es una herramienta que se emplea para planificar y programar tareas a lo largo de un período determinado de tiempo. Gracias a una fácil y cómoda visualización de las acciones a realizar, permite realizar el seguimiento y control del progreso de cada una de las etapas de un proyecto. Reproduce gráficamente las tareas, su duración y secuencia, además del calendario general del proyecto y la fecha de finalización prevista. [10]

- **Plasma WorkSpace:** Consiste en un entorno de escritorio compuesto de librerías, *frameworks* y una serie de aplicaciones todo ello integrado. Plasma se usa bastante como entorno de escritorio gráfico en los sistemas basados en *Nix(Linux, Unix, etc...). [13]

- **Visual Basic:** Consiste en un lenguaje de programación desarrollado por Alan Cooper en 1991 y que tiene como objetivo programar contenidos informáticos de manera simple y eficaz gracias a una serie de herramientas visuales integradas en el lenguaje.

- **SDK:** *Software Development Kit* o kit de desarrollo de software. Es un conjunto de herramientas que ayudan a la programación de aplicaciones para un entorno tecnológico particular. Es decir, las aplicaciones desarrolladas sobre el SDK estarán destinadas a algún sistema operativo, plataforma hardware, consola de videojuegos o paquete de software en especial. [7]

- **SSL:** Según la propia DigiCert, entidad certificadora, SSL o *Secure Sockets Layer* consiste en un protocolo que está diseñado para transmitir información de ida y de vuelta de una forma segura. Las aplicaciones que transmiten dicha información tienen las claves necesarias para poder cifrar o descifrar los mensajes que envíen o reciban. [3]

- **Socket:** Consiste en un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador. Desde el punto de vista de programación, un socket no es más que un "fichero" que se abre de una manera especial. [4]

- **KDE:** *K Desktop Environment*, llamado el escritorio K, es el predecesor de *Plasma WorkSpace*. Cuando el entorno de escritorio K migró sus librerías a la versión 4 de las librerías gráficas QT, éste fue renombrado como *Plasma WorkSpace*.

- **Unicode:** Unicode es el estándar de codificación de caracteres universal utilizado para la representación de texto en el procesamiento del equipo. Unicode proporciona una manera consistente de codificación de texto multilingüe y facilita el intercambio de archivos de texto internacionales. [5]

- **SQLInjection:** Es una vulnerabilidad que permite a un atacante realizar consultas a una base de datos, se vale de un incorrecto filtrado de la información que se pasa a través de los campos y/o variables que usa un sitio web. Es, por lo general, usada para extraer credenciales y realizar accesos ilegítimos, práctica un tanto neófita, ya que un fallo de este tipo puede llegar a permitir la ejecución de comandos en el servidor, subida y lectura de archivos, o peor aún, la alteración total de los datos almacenados. [28]

- **Pythonic Way:** Es el llamado "camino de Python". Representa la forma por la cual se debe programar dentro de éste simbólico lenguaje de programación. El camino consiste en las metodologías que se deben aplicar a la hora de programar en código Python y que son consideradas como *buenas prácticas* a los ojos del resto de programadores y de la comunidad en general. Éste camino posee una serie de principios que se podrían traducirse como el *Zen*

de Python y fueron descritos por el mismísimo Tim Peters que es uno de los desarrolladores del lenguaje. [32]

1. Bello es mejor que feo.
2. Explícito es mejor que implícito.
3. Simple es mejor que complejo.
4. Complejo es mejor que complicado.
5. Plano es mejor que anidado.
6. Disperso es mejor que denso.
7. La legibilidad cuenta.
8. Los casos especiales no son tan especiales como para quebrantar las reglas.
9. Lo práctico gana a lo puro.
10. Los errores nunca deberían dejarse pasar silenciosamente.
11. A menos que hayan sido silenciados explícitamente.
12. Frente a la ambigüedad, rechaza la tentación de adivinar.
13. Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
14. Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
15. Ahora es mejor que nunca.
16. Aunque nunca es a menudo mejor que ya mismo.
17. Si la implementación es difícil de explicar, es una mala idea.
18. Si la implementación es fácil de explicar, puede que sea una buena idea.
19. Los espacios de nombres (namespaces) son una gran idea – ¡hagamos más de esas cosas!

- **Portar aplicación:** Consiste en la acción por la cual un programa es adaptado y modificado para conseguir hacer que funcione de manera deseada en un sistema operativo diferente para el cual fue pensado en una primera instancia. La portabilidad puede ir desde la modificación del código de programación, hasta la recompilación de las distintas librerías que se hayan usado para crear el programa.

- **Binario:** Un binario es un paquete que contiene un software que está compilado y preparado para ser usado en un entorno operativo determinado. Un binario puede contener uno varios paquetes de software que permiten, de forma sencilla, preparar un programa con todas sus dependencias para ser usado en cualquier ordenador. La finalidad de un binario es facilitar al usuario la instalación o desinstalación del software deseado.

- **Debian Package:** Consiste en un binario que permite la instalación de software en un sistema operativo que contenga un gestor de paquetes basado en Debian.

- **Debian:** Es una comunidad formada por voluntarios y usuarios que desarrolla software libre. Debian mantiene un sistema operativo bajo el mismo nombre basado en el kernel de "GNU/Linux". La comunidad fue formada en 1991 por Ian Murdock y hoy en día es una importante organización con muchos desarrolladores en sus espaldas que ayudan a mejorar la filosofía del software libre y el propio sistema operativo. [19]

- **Configure:** Es un script usado principalmente en sistemas operativos basados "GNU/Linux" para preparar un entorno de compilación deseado. Cuando se usa *configure* el script revisa si en el sistema existen las dependencias necesarias por el programa a compilar y establece las acciones que se deben realizar. Dichas acciones son los *makefiles*.

- **Makefile:** Consiste en un archivo que se genera después de realizar el lanzamiento del comando *configure*. Un *makefile* es un fichero generado que contiene las directrices de compilación de un programa. Establece qué se va a compilar y cómo se va a compilar y dónde debe de salir el resultado de la operación.

- **Make:** Consiste en un comando típico en sistemas operativos "GNU/Linux" que permite la ejecución de las instrucciones contenidas en un archivo *makefile*.

- **SUS**: Llamado *System Usability Scale* y creado por John Brooke en 1986. Un SUS consiste en una herramienta simple y rápida para poder hacer una medición sobre la usabilidad de servicios o productos incluyendo cuestiones como hardware, software, servicios móviles, páginas web y aplicaciones informáticas. [16]

- **Script:** Un guión o script es un fichero de texto que contiene una serie de instrucciones que se pueden ejecutar en la línea de órdenes y que se ejecutarán seguidas. Generalmente un script se lanza desde una consola de Bash y ésta interpreta línea a línea cada instrucción. La idea de un script es automatizar una serie de comandos para facilitar ciertas labores. [31]

- **Tag:** El término tiene muchas acepciones y formas de interpretarse, en el caso de éste proyecto un tag es una etiqueta que aglutina una serie de scripts en su interior. La idea es automatizar el proceso de varios scripts a la vez cuando se invoca a un tag determinado.

1.2. Descripción del proyecto

El proyecto es un espacio realizado en el lenguaje de programación "Python" y en las librerías visuales "QT" en el que se pueden realizar una serie de funciones importantes definidas en un sistema informático complejo actual(realizar administraciones determinadas, añadir elementos, eliminar, asignar variables a objetos, eliminar variables etc..). El sistema desarrollado se llama **"Akeko"** que la traducción desde el idioma Yoruba significa "estudiante".

Lo que se desea solventar es la gestión del alumnado de una forma eficiente y rápida, ahorrando tiempo, costes y mejorando el rendimiento del profesorado.

1.3. Objetivos del proyecto

Los objetivos principales del proyecto son:

1. *Ofrecer una herramienta de gestión útil:* Se quiere crear una herramienta que permita, de forma muy sencilla, gestionar a los alumnos para que éstos puedan hacer uso de los distintos servicios disponibles.

2. *Mejorar la eficiencia del profesorado:* Mediante la automatización de los procesos en la gestión; el profesorado tardará menos tiempo en asignar a los alumnos a los distintos servicios existentes, lo cual, permitirá un aumento de su productividad.

3. *Hacer un sistema intuitivo y sencillo de usar:* Se quiere un sistema en el que no sean necesarios extensos conocimientos informáticos para poder manejarlo de forma óptima.

4. *Satisfacción personal:* El hecho de crear una aplicación sólida haciendo uso de distintas tecnologías diferentes y nuevas para el desarrollador. Obteniendo, de ésta forma, nuevos conocimientos en otros lenguajes en los que no se ha dado docencia.

1.4. Arquitectura

La arquitectura utilizada es la arquitectura "cliente-servidor". La razón por la que se ha elegido en contra de otros modelos es la siguiente:

- *Generar una conexión segura entre el cliente y el servidor:* El profesor manejará datos sensibles por lo que es imperativo crear una pasarela encriptada para que dichos datos no se vean expuestos.

- *Permitir al programa conectarse a distintos destinos:* Permitir a la aplicación conectarse a otros destinos para hacer la gestión en cualquier servidor que esté soportado y cumpla las condiciones del contexto de negocio de la aplicación.

1.5. Herramientas

Las herramientas utilizadas en éste trabajo fin de grado se detallarán a continuación:

1. **Pycharm:** IDE de desarrollo específico para "Python" desarrollado por la empresa "JetBrains". Tiene dos versiones, una de pago para entornos profesionales y una edición gratuita comunitaria. En éste caso se ha hecho uso de la edición comunitaria para poder desarrollar el proyecto haciendo uso del lenguaje "Python".

2. **Visual Paradigm:** Entorno de desarrollo para lenguaje de modelado "UML" desarrollado por la empresa "Visual Paradigm International". Éste programa permite la creación de diagramas "UML" para representar los casos de uso, el diseño y el modelo de dominio.

3. **PyQT:** Se trata de una librería, considerada un binding de "QT". Está desarrollada por "RiverBank Computing". El uso de ésta librería ayuda a interpretar el código "Python" para construir interfaces en "QT" de forma semiautomática y sin mucho esfuerzo.

4. **TextMaker:** Es un editor multiplataforma que contiene una serie de herramientas para facilitar el desarrollo de documentos con Latex. Está desarrollado por Pascal Brachet y es un programa gratuito de software libre bajo la licencia "GPL". Con éste programa se ha podido editar toda la memoria del proyecto.

5. **Firefox:** Navegador web multiplataforma desarrollado por la fundación "Mozilla". El uso de éste software ha sido esencial para la búsqueda de información sobre herramientas y manuales que han ayudado en la consecución del proyecto.

6. **Dropbox:** Consiste en un servicio de alojamiento de archivos en la "nube". Contiene una aplicación de sincronización automática entre dispositivos. Dropbox está desarrollado por "Dropbox Inc". El uso de éste servicio ha sido indispensable para tener copias de seguridad de la documentación del proyecto y facilitar la actualización de los documentos entre los diferentes dispositivos utilizados.

7. **Pencil:** Es un software de prototipado de interfaces creado por a empresa "Evolus Co, LTD". Es gratuito y está bajo la licencia "GPLv2". El uso de esta herramienta ha sido primordial a la hora de obtener una visión inicial de cómo debería de ser el diseño de las interfaces.

8. **MySQL:** Es un sistema de gestión de bases de datos relacionales multihilo y multiusuario desarrollado por "MySQL AB". Actualmente pertenece a "Oracle coporation" y ofrece dos versiones de uso; una comercial de pago y una gratuita bajo la licencia "GPL". El uso de éste sistema ha sido esencial para manejar y guardar los datos que necesita la aplicación para su correcto funcionamiento.

9. **Works:** Programa informático que pertenece a la suite ofimática de "Office". Está desarrollado por "Microsoft". El uso de éste programa viene dado por la razón de conseguir una mayor facilidad a la hora de establecer las tareas a realizar en el proyecto y la creación de los diagramas temporales para representar dichas tareas.

10. **Cacoo:** Es una herramienta online para la creación de diagramas de todo tipo, desde diagramas "UML" hasta diagramas de flujo. Está desarrollado por "Nulab Inc". El uso de ésta herramienta tiene como objetivo la creación sencilla de un diagrama de descomposición de trabajo (EDT) para facilitar la relación de las tareas que tiene el proyecto.

11. **QtDesigner:** Consiste en una herramienta que permite el diseño y la creación intuitiva y gráfica de interfaces basadas en "QT". Designer forma parte de una serie de herramientas aglutinadas con el nombre de "QTProject". "QTProject" pertenece a la empresa "Digia" aunque antiguamente fue propiedad de "Nokia". El uso de éste programa ha sido únicamente orientado en la creación de interfaces gráficas en "QT".

12. **SSMTP:** Es un programa que envía un correo desde un ordenador local hacia un servidor de correo electrónico ya configurado. La aplicación únicamente se encarga de enviar e-mails, no puede ni recibirlos ni gestionarlos y ni mucho menos hacer de servidor. Depende de forma exclusiva de un servidor externo previamente ya configurado. La utilización de éste programa ha sido clave para poder conseguir enviar correos electrónicos a los distintos usuarios para informarles sobre diversas cuestiones relacionadas con el sistema.

13. **Checkinstall:** Consiste en un programa de línea de comandos para los sistemas operativos basados en "GNU/Linux". Ésta herramienta permite crear un paquete binario de *Debian* (extensión *.deb) de una forma simple y sencilla. Normalmente no suele ser aceptado por la comunidad por la simplicidad y la poca seguridad que ofrece debido a que no permite firmar los paquetes creados ni ofrece opciones más avanzadas que son requeridas en algunas distribuciones de "GNU/Linux". El uso de ésta herramienta ha sido necesario para crear una serie de paquetes para poder instalar las librerías necesarias por el proyecto en la distribución "GNU/Linux" *Ubuntu*.

14. **Microsoft Visual Studio 2012:** Este potente IDE, creado por la empresa "Microsoft", permite programar en varios lenguajes de programación, desde "C++" hasta "Python" pasando por "Visual Basic". Además de la capacidad de IDE integrado, Visual Studio provee a un sistema operativo "Windows" una serie de herramientas para compilar librerías en "C++". El uso de éste software ha sido clave para poder compilar "PyQT" en un entorno operativo "Windows" y así hacer compatible y ejecutable la aplicación en dicho sistema operativo.

15. **Cuestionario de usabilidad:** Se trata de un cuestionario que está basado en un estudio en el cual se establecen una serie de preguntas determinadas y se genera una escala de adjetivos/notas para los distintos valores del SUS. Éste cuestionario ha sido usado para medir la calidad de las interfaces de la aplicación. [20]

1.6. Alcance del proyecto

Éste trabajo fin de grado está realizado por un sólo componente. Debido a ello no es necesario describir qué personas han realizado qué tareas, por lo que se asume que la ejecución de todas las tareas han sido realizadas por una sola persona.

A continuación, se va a presentar un índice con todo el desglose de las tareas realizadas y seguido se va a mostrar, paso a paso, toda la información referente a cada tarea con sus características asociadas.

1. Organización del Proyecto.
   - 1.1. Captura de requisitos.
   - 1.2. Planificación de las tareas a realizar.
   - 1.3. Creación de los prototipos de interfaces.
   - 1.4. Aprendizaje del lenguaje de programación Python.
   - 1.5. Aprendizaje del lenguaje gráfico QT.

1.6. Aprendizaje del uso y manejo de TextMaker.

1.7. Aprendizaje de Microsoft Project.

1.8. Aprendizaje del lenguaje BashScript.

1.9. Creación del DOP.

2. Diseño del proyecto.

2.1. Diseño de los módulos.

2.1.1. Diseño de la parte cliente.

2.1.1.1. Diseño de casos de uso.

2.1.1.2. Diseño de modelo de dominio.

2.1.1.3. Diseño de diagrama de clases.

2.1.1.4. Diseño de diagramas de secuencia.

2.1.2. Diseño de la parte servidor.

2.1.2.1. Diseño de casos de uso.

2.1.2.2. Diseño de modelo de dominio.

2.1.2.3. Diseño de diagrama de clases.

2.1.2.4. Diseño de diagramas de secuencia.

2.2. Diseño de la base de datos.

2.3. Diseño de las pruebas.

3. Implementación del proyecto.

3.1. Generación de la base de datos.

3.2. Implantación módulos de la parte cliente.

3.3. Implantación módulos de la parte servidor.

3.4. Pruebas de funcionamiento general.

4. Documentación

4.1. Manual técnico.

4.2. Manual de usuario.

1.6.1. Diagrama de descomposición EDT

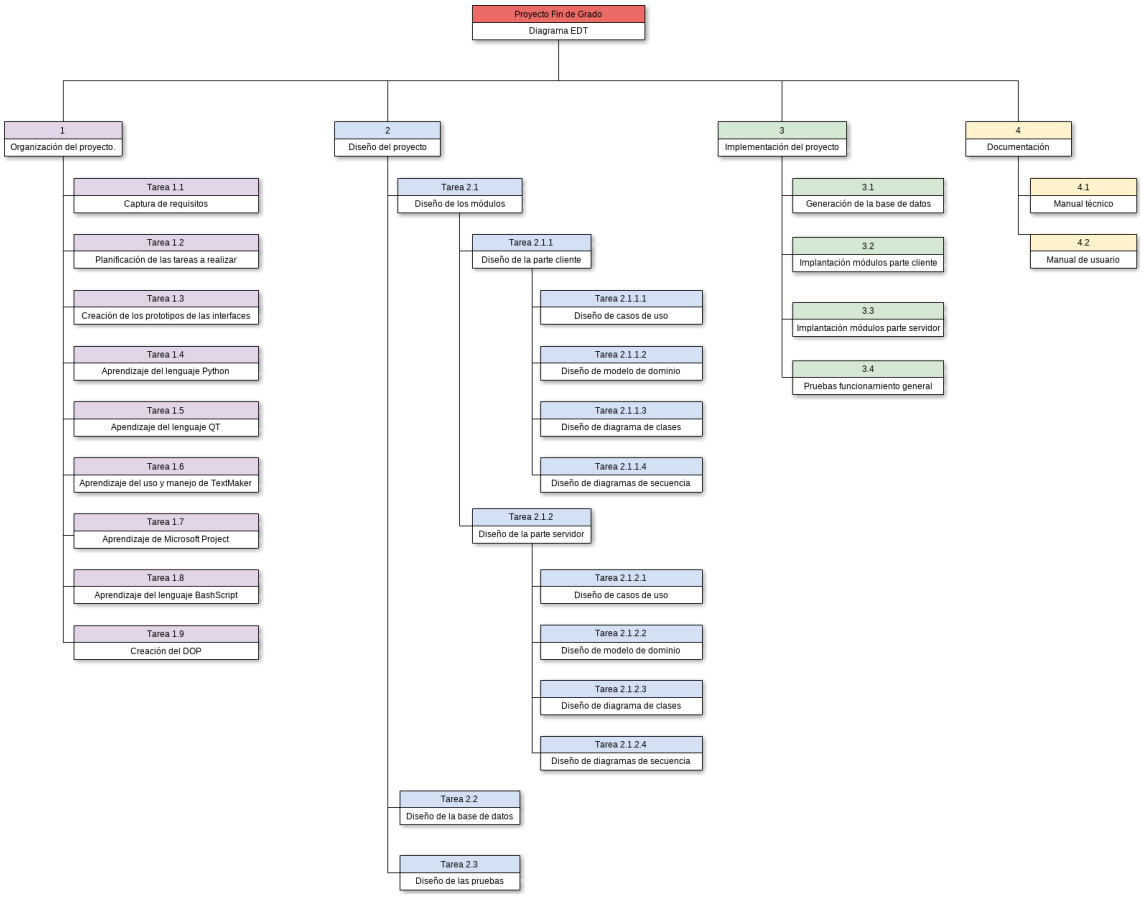


Figura 1.1: Diagrama de descomposición de trabajo

1.6.2. Descripción de los paquetes de trabajo

1.6.2.1. Organización del Proyecto

1. Organización del proyecto.

Paquete de trabajo: 1.1- Captura de requisitos.
Duración: 1 hora.

Descripción

Consiste en un tarea por la cual el programador debe identificar las necesidades exigidas por la persona que le propone el proyecto. Se podría traducir como la primera toma de contacto que se hará con el programa a desarrollar.

En ésta tarea es imperativo conocer exactamente cuáles son los objetivos que se quieren desarrollar y cuál es el alcance que podría tomar el proyecto.

Entradas

En éste apartado se producirá la entrada inicial de información por parte de la persona que propone el proyecto. Lo importante, es conocer exactamente las funcionalidades necesarias, así como el tipo de arquitectura que se va a tomar y sobre todo qué tipo de tecnologías se van a utilizar.

Durante la toma de datos, es necesario concretar todas las restricciones existentes para poder determinar qué tipo de sistema operativo se debe de utilizar, si el proyecto es monousuario o multiusuario y sobre todo, sobre qué plataforma debe o no funcionar.

Salidas/Entregables

La salida de ésta tarea consistirá en un documento que tendrá toda la información requerida para poder realizar el planteamiento, organización y consecución del proyecto.

Recursos necesarios

Los recursos serán únicamente la persona que va a desarrollar el proyecto y el cliente(en éste caso un profesor) el cual expondrá sus necesidades y la visión que tiene de lo que necesita.

Es importante concretar que en éste caso, al ser un TFG, el cliente es un profesor con alta experiencia informática para expresar las cualidades del proyecto y que por lo tanto no hace falta que el programador deba hacerse una idea de lo que debería ser el proyecto. Por lo que muchas incongruencias que se pudieran presentar durante la consecución del trabajo se resolverán de forma inmediata.

Precedencias

En éste caso no existe precedencias al ser una tarea inicial.

1. Organización del proyecto.

Paquete de trabajo: 1.2- Planificación de las tareas a realizar.
Duración: 4 horas.

Descripción

Éste apartado engloba la definición concreta de las tareas que se deben llevar a cabo. Se busca concretar qué tareas se deben hacer, cómo se han de hacer y el número de personas implicadas en cada una de ellas.

Además, también se debe planificar qué tipos de tecnologías se van a aplicar y se debe considerar cuál debe ser la manera más eficiente de alcanzar los objetivos que desea el cliente.

Entradas

Como entrada es necesario tener una idea de lo que se debe de realizar. Para ello, se necesitan las directrices que haya pedido el cliente sobre cómo ha de ser el programa a completar.

Éstas directrices se obtienen mediante una captura de requisitos realizada anteriormente.

Salidas/Entregables

La salida que se obtendrá, consistirá en un documento por el cual se pueda ver todas las tareas con una serie de propiedades determinadas. Dichas propiedades serán:

- El nombre de la tarea.
- La duración que se estima.
- Una descripción que deje claro sobre qué trata.
- Las entradas que tenga. Por ejemplo, si necesita de determinadas tareas anteriores para poder realizarse.
- Qué salida ofrecerá cada una de ellas, considerando salida como una información en un formato determinado.
- Los recursos que han sido necesarios para completar la tarea de forma satisfactoria.
- Las tareas de las cual precede.

Además, también se especificará qué tipo o tipos de lenguajes de programación se van a realizar y qué tipos de tecnologías se tiene pensado aplicar.

Recursos necesarios

En éste caso, los recursos necesarios serán dos personas. Por un lado, estará el programador que realizará la planificación que considere oportuna y por el otro, estará el profesor director del proyecto que evaluará la calidad de la planificación propuesta.

Precedencias

Es obvio que para poder realizar ésta tarea, es necesario haber completado con anterioridad la tarea de captura de requisitos dado que sin ésta es completamente imposible llegar a cabo una planificación concreta.

1. Organización del proyecto.

Paquete de trabajo: 1.3- Creación de los prototipos de interfaces,
Duración: 3 horas.

Descripción

Consiste en la realización de los prototipos de las interfaces. En ésta tarea se intenta buscar la mejor manera de representar conceptualmente cómo debería ser el aspecto que debe de tener las interfaces del proyecto.

Entradas

Como entrada, se debe tener una planificación temporal fijada en la cual se especifique la ejecución de ésta tarea.

Salidas/Entregables

La salida que se generará será un documento gráfico o un fichero dentro de un programa de modelado de interfaces, con la descripción de las interfaces que serán necesarias implantar en el proyecto y qué aspecto deberán de tener, explicando en qué zonas irán los elementos de la interfaz y cómo estarán colocados.

Recursos necesarios

Para la consecución efectiva de la tarea, sólo es necesaria la intervención de programador, el cual deberá establecer los conceptos de las interfaces.

Precedencias

Ésta tarea precede directamente de la planificación de las tareas.

1. Organización del proyecto.

Paquete de trabajo: 1.4- Aprendizaje del lenguaje de programación Python.
Duración: 20 horas.

Descripción

Ésta tarea consiste en el aprendizaje y entendimiento del lenguaje de programación "Python" mediante la búsqueda de información en diferentes medios, ya sea Internet, libros o gente con experiencia en la materia.

Entradas

Como entrada para ésta tarea, es necesario saber exactamente qué cantidad de interfaces necesitamos programar y con que características, para saber cuán de profundo es necesario el aprendizaje de éste lenguaje de programación.

Salidas/Entregables

En ésta tarea no se produce una salida física como tal por lo que no podríamos decir que se produce un documento o un fichero determinado. Lo que si se produce, es una adquisición de conocimientos que serán vitales para poder realizar el proyecto de forma satisfactoria.

Recursos necesarios

Para que ésta tarea se desarrolle de forma efectiva es necesaria la intervención del programador que va a desarrollar todo el proyecto y de agentes externos que le den cierta ayuda para poder aprender el lenguaje de programación. Dichos agentes pueden ser otros alumnos, profesores, personas experimentadas e incluso libros(Orientándonos a recursos).

Precedencias

Las precedencias de ésta tarea es la consecución exitosa de la creación de los prototipos de interfaces.

1. Organización del proyecto.

Paquete de trabajo: 1.5- Aprendizaje del lenguaje gráfico QT.
Duración: 3 horas.

Descripción

La tarea en cuestión comprende el aprendizaje de las técnicas necesarias para poder hacer uso del lenguaje de programación "QT" y su vinculación con "Python" para realizar interfaces gráficas funcionales. Dicho aprendizaje es esencial debido a las características del proyecto.

Lo más importante de ésta tarea, no es el mero hecho de aprender las técnicas de modelado y diseño que ofrece "QT", si no la capacidad de vincular éste lenguaje con "Python" para conseguir una interacción satisfactoria en diferentes sistemas gráficos y operativos.

Entradas

Como entradas serán usados los documentos relacionados con el prototipo de las interfaces para tener una idea conceptual de lo que se debe desarrollar y por ende, qué técnicas se deben aprender.

Además, también será necesario tener los conocimientos adecuados sobre el lenguaje de programación "Python" para saber qué técnicas se han de aprender para adaptar las interfaces a dicho lenguaje y que éstas sean plenamente compatibles y funcionales(PyQT).

Salidas/Entregables

De la misma forma que otras tareas de índole similar, la salida que se produce no es algo físico o algo que se pueda entregar. En éste caso es el aprendizaje y la dominación de las técnicas para crear las interfaces compatibles con "Python" adaptadas a las necesidades del proyecto.

Recursos necesarios

Para ésta tarea es necesario tener como recursos al programador que va a realizar el proyecto y a las personas que puedan aportar determinados conocimientos sobre las experiencias en éste campo.

Precedencias

Las precedencias de ésta tarea es el aprendizaje del lenguaje de programación "Python".

1. Organización del proyecto.

Paquete de trabajo: 1.6- Aprendizaje del uso y manejo de TextMaker.
Duración: 1 hora.

Descripción

Consiste en el aprendizaje y uso de la herramienta de edición "TextMaker" para el lenguaje "Latex". Ésta tarea es necesaria para que el programador pueda aprender las opciones avanzadas que ofrece "Latex" a través de una herramienta que permite la edición avanzada de textos.

Entradas

Ésta tarea no requiere de ninguna entrada adicional.

Salidas/Entregables

La salida que se produce es la adquisición de los conocimientos para poder hacer un uso correcto de "TextMaker" y poder editar sin problemas ficheros en el lenguaje "Latex".

Recursos necesarios

Los recursos que hacen falta para desarrollar ésta tarea es el propio programador del proyecto y la información que pueda obtener sobre el uso del programa, ya sea mediante la ayuda de personas ajenas al proyecto, recursos en Internet u otras fuentes externas como libros etc.

Precedencias

Ésta tarea no tiene ninguna precedencia.

1. Organización del proyecto.

Paquete de trabajo: 1.7- Aprendizaje de Microsoft Project.
Duración: 3 horas.

Descripción

La ejecución, aborda el aprendizaje efectivo sobre el manejo y uso de "Microsoft Project". La tarea tiene como finalidad conseguir la adquisición de conocimientos necesarios para poder usar de forma adecuada las capacidades de Project y poder ser capaz de exponer una planificación temporal de forma gráfica y sencilla, para presentar como parte de la documentación del proyecto.

Entradas

Las entrada de esta tarea es el documento que contiene la planificación de las tareas a realizar, clave para poder saber exactamente qué se necesita aprender y así poder poner la información necesaria dentro de Project.

Salidas/Entregables

La salida de ésta tarea será la adquisición de los conocimientos para poder manejar de forma correcta Project y ser capaz de plasmar de forma efectiva la información requerida para presentar en la documentación del proyecto.

Recursos necesarios

Como recursos será necesaria la participación del programador del proyecto y de entidades externas como personas con conocimientos sobre el uso de Project e información obtenida a través de medios como Internet o libros.

Precedencias

Ésta tarea tiene como precedencia haber realizado anteriormente una planificación temporal del proyecto.

1. Organización del proyecto.

Paquete de trabajo: 1.8- Aprendizaje del lenguaje BashScript.
Duración: 15 horas.

Descripción

Abarca el aprendizaje de los métodos para programar de forma efectiva en BashScript. La realización de ésta tarea es vital para poder conseguir que la parte servidor de la aplicación funcione de forma correcta.

Ésta tarea tiene como objetivo ampliar los conocimientos del programador en materia de uso del lenguaje BashScript para conseguir realizar comandos avanzados dentro de un servidor "GNU/Linux" y así satisfacer los requisitos necesarios para la ejecución de scripts en el sistema servidor.

Entradas

Como entrada para ésta tarea es necesario que el programador posea algunos conocimientos sobre en qué consiste el lenguaje BashScrip y su lógica de funcionamiento.

Salidas/Entregables

La salida que produce la realización exitosa de ésta tarea será la adquisición de conocimientos para satisfacer la demanda que solicita la realización de la parte servidor de la aplicación.

Recursos necesarios

Los recursos necesarios para que ésta tarea se pueda completar son el propio programador con sus conocimientos básicos. También es importante incluir elementos como recursos externos que facilitan el aprendizaje como libros, internet o personas con experiencia.

Precedencias

Ésta tarea no tienen precedencias.

1. Organización del proyecto.

Paquete de trabajo: 1.9- Creación del DOP.
Duración: 25 horas.

Descripción

Como bien indica, esta tarea se enfoca en la creación del DOP del proyecto. Lo que se quiere hacer, es plasmar por escrito toda la información posible acerca de las características que tiene el proyecto y qué rumbo se desea tomar para su consecución.

La generación del DOP es crítica para poder expresar todas las virtudes del proyecto a personas que sean ajenas al programador. Por ende, se puede deducir que se intenta hacer un documento que permita que otras personas puedan entender las intenciones del programador y puedan usar de forma exitosa su proyecto.

Entradas

Como entrada, será necesario tener la planificación temporal bien definida, los prototipos de interfaces definidos y sobre todo, haber usado "Microsoft Project" para poder extraer un buen gráfico Gantt que permitirá plasmar la planificación temporal.

Salidas/Entregables

La salida que produce ésta tarea es un extenso documento en el que se expone todas las características del proyecto y toda la información posible sobre su construcción, manejo y tecnologías utilizadas.

Recursos necesarios

En ésta tarea sólo es necesaria la intervención del programador del proyecto.

Precedencias

La tarea de Aprendizaje de "BashScript" y todas las tareas anteriores que preceden a la descrita, ya que al final, se engloban todos los conocimientos adquiridos para poder crear el proyecto más las planificaciones e ideas de cómo debería de ser.

1.6.2.2. Diseño del Proyecto

2.1.1 Diseño de la parte cliente.

Paquete de trabajo: 2.1.1.1- Diseño de casos de uso.
Duración: 5 horas.

Descripción

En ésta tarea, se capturan los casos de uso de la parte cliente de la aplicación. Se intenta conseguir plasmar qué casos de uso pueden aparecer a la hora de hacer uso de la aplicación por un cliente.

Entradas

Ésta tarea tiene como entradas las interfaces desarrolladas en tareas anteriores. Con dichas interfaces se puede saber exactamente los casos de uso que serán necesarios.

Salidas/Entregables

Como salida se creará un documento en "Visual Paradigm" que mostrará los casos de uso definidos.

Recursos necesarios

Los recursos que se necesitan en ésta tarea, es únicamente el programador que realizará el proyecto.

Precedencias

La precedencia de ésta tarea será haber realizado el DOP del proyecto, para tener un punto de partida. Además, será necesario disponer de las interfaces de la parte cliente ya fijadas para poder saber exactamente qué funcionalidades son posibles de realizar.

2.1.1 Diseño de la parte cliente

Paquete de trabajo: 2.1.1.2- Diseño de modelo de dominio.
Duración: 7 horas.

Descripción

El diseño del modelo de dominio consiste en realizar el esquema relacional de la parte cliente. Identificando qué datos se van a guardar y dónde se deben guardar. Ésos datos son la información necesaria para que la parte del cliente sea funcional y se pueda manejar la información necesaria para un correcto funcionamiento.

Entradas

Como entrada, será necesario disponer tanto de las interfaces definidas en tareas anteriores, como tener unos casos de uso que nos ayuden a identificar exactamente qué funcionalidades tenemos para saber qué tipos de datos necesitamos manejar.

Salidas/Entregables

La salida que produce ésta tarea es un documento en "Visual Paradigm" que provee información visual sobre cómo es el modelo de dominio.

Recursos necesarios

Los recursos que se necesitan en ésta tarea, es únicamente el programador que realizará el proyecto.

Precedencias

La precedencia de ésta tarea, será el diseño de de los casos de uso de la parte cliente y los prototipos de las interfaces bien definidos.

2.1.1 Diseño de la parte cliente

Paquete de trabajo: 2.1.1.3- Diseño del diagrama de clases.
Duración: 5 horas.

Descripción

El diseño de los diagramas de clases es una tarea por la cual el programador debe definir exactamente cuántas clases necesita el programa que va a realizar.

Hay que destacar que el concepto "clase" viene definido por las clases en "Java", pero en "Python" es exactamente igual dado que es un lenguaje de programación orientado a objetos por lo que también se debe realizar un diseño de clases.

La realización de ésta tarea es importante para saber cuántas clases se deben de programar, qué tipo de datos se tienen que manejar y qué métodos se van a utilizar. De ésta forma cuando se tenga que implementar el proyecto, todo será más simple y fácil.

Entradas

La entrada que condiciona ésta tarea es un fichero con el modelo de dominio y un fichero con los casos de uso.

Salidas/Entregables

La salida que produce ésta tarea es un fichero en "Visual Paradigm" que consistirá en un mapa que contenga todas las clases con los tipos de datos y sus métodos necesarios para poder programar de forma eficiente la parte cliente de la aplicación.

Recursos necesarios

Sólo será necesario la intervención del programador para realizar la tarea.

Precedencias

Como precedencia, hay que nombrar a la tarea que engloba el diseño de los diagramas de clases.

2.1.1 Diseño de la parte cliente

Paquete de trabajo: 2.1.1.4- Diseño del diagrama de secuencia.
Duración: 8 horas.

Descripción

Consiste en el diseño de los diagramas de secuencia de la parte cliente que mostrarán cuál será el funcionamiento exacto de los métodos que se hayan definido en etapas anteriores.

El diseño de los diagramas de secuencia dejarán patente qué pasos realizará un método y con qué tipos de datos jugará hasta lograr su objetivo final.

Entradas

Como entrada será necesario el diagrama de clases para disponer de un punto de referencia a la hora de realizar la tarea.

Salidas/Entregables

Como salida se crearán una serie de documentos que mostrarán diferentes diagramas de diseño que describirán el funcionamiento de distintas funcionalidades, especificando qué tipo de métodos serán necesarios y qué tipo de datos se manejarán.

Recursos necesarios

Para que ésta tarea se realice de forma exitosa sólo será necesaria la intervención del programador del proyecto.

Precedencias

Esta tarea precede directamente del diseño de los diagramas de clases de la parte cliente.

2.1.2 Diseño de la parte servidor.

Paquete de trabajo: 2.1.2.1- Diseño de los casos de uso.
Duración: 7 horas.

Descripción

En ésta tarea, se capturan los casos de uso de la parte servidor de la aplicación. Se intenta conseguir plasmar qué casos de uso pueden aparecer a la hora de hacer uso de la aplicación por un administrador.

Entradas

Ésta tarea tiene como entradas las interfaces desarrolladas en tareas anteriores. Con dichas interfaces se puede saber exactamente los casos de uso que serán necesarios.

Salidas/Entregables

Como salida se creará un documento en "Visual Paradigm" que mostrará los casos de uso definidos.

Recursos necesarios

Los recursos que se necesitan en ésta tarea son únicamente el programador que realizará el proyecto.

Precedencias

La precedencia de ésta tarea será haber realizado el DOP del proyecto para tener un punto de partida. Además, será necesario disponer de las interfaces de la parte servidor ya fijadas para poder saber exactamente qué funcionalidades son posibles de realizar.

2.1.2 Diseño de la parte servidor.

Paquete de trabajo: 2.1.2.2- Diseño del modelo de dominio.
Duración: 9 horas.

Descripción

El diseño del modelo de dominio consiste en realizar el esquema relacional de la parte servidor. Identificando qué datos se van a guardar y dónde se deben guardar. Ésos datos son la información necesaria para que la parte del cliente sea funcional y se pueda manejar la información necesaria para un correcto funcionamiento.

Entradas

Como entrada, será necesario disponer tanto de las interfaces definidas en tareas anteriores, como tener unos casos de uso que nos ayuden a identificar exactamente qué funcionalidades tenemos para saber qué tipos de datos necesitamos manejar.

Salidas/Entregables

La salida que produce ésta tarea es un documento en "Visual Paradigm" que provee información visual sobre cómo es el modelo de dominio.

Recursos necesarios

Los recursos que se necesitan en ésta tarea son únicamente el programador que realizará el proyecto.

Precedencias

La precedencia de ésta tarea será el diseño de de los casos de uso de la parte servidor y los prototipos de las interfaces bien definidos.

2.1.2 Diseño de la parte servidor.

Paquete de trabajo: 1.1- 2.1.2.3- Diseño del diagrama de clases.
Duración: 6 horas.

Descripción

El diseño de los diagramas de clase es una tarea por la cual el programador debe definir exactamente cuántas clases necesita el programa que va a realizar.

Hay que destacar que el concepto "clase" viene definido por las clases en "Java", pero en "Python" es exactamente igual dado que es un lenguaje de programación orientado a objetos por lo que también se debe realizar un diseño de clases.

La realización de ésta tarea es importante para saber cuántas clases se deben de programar, qué tipo de datos se tienen que manejar y qué métodos se van a utilizar. De ésta forma cuando se tenga que implementar el proyecto, todo será más simple y fácil.

Entradas

La entrada que condiciona ésta tarea es el fichero con el modelo de dominio y el fichero con los casos de uso.

Salidas/Entregables

La salida que produce ésta tarea es un fichero en "Visual Paradigm" que consistirá en un mapa que contenga todas las clases con los tipos de datos y sus métodos necesarios para poder programar de forma eficiente la parte servidor de la aplicación.

Recursos necesarios

Sólo será necesario la intervención del programador para realizar la tarea.

Precedencias

Como precedencia, hay que nombrar la tarea que engloba el diseño de los diagramas de clases.

2.1.2 Diseño de la parte servidor.

Paquete de trabajo: 2.1.2.4- Diseño del diagrama de secuencia.
Duración: 8 horas.

Descripción

Consiste en el diseño de los diagramas de secuencia de la parte servidor que mostrarán cuál será el funcionamiento exacto de los métodos que se hayan definido en etapas anteriores.

El diseño de los diagramas de secuencia dejarán patente qué pasos realizará un método y con que tipos de datos jugará hasta lograr su objetivo final.

Entradas

Como entrada será necesario el diagrama de clases para disponer de un punto de referencia a la hora de realizar la tarea.

Salidas/Entregables

Como salida se crearán una serie de documentos que mostrarán diferentes diagramas de diseño que describirán el funcionamiento de diferentes funcionalidades, especificando qué tipo de métodos serán necesarios y qué tipo de datos se manejarán.

Recursos necesarios

Para que ésta tarea se realice de forma exitosa sólo será necesaria la intervención del programador del proyecto.

Precedencias

Ésta tarea precede directamente del diseño de los diagramas de clases de la parte servidor.

2. Diseño del proyecto.

Paquete de trabajo: 2.2- Diseño de la base de datos.
Duración: 6 horas.

Descripción

El diseño de la base de datos consiste en la creación en papel de la estructura general que debe tener la base de datos que se va a utilizar en el proyecto.

Dicha estructura debe contener las tablas con los campos necesarios para poder albergar en su corazón todos los datos necesarios para que la aplicación pueda ejecutarse sin problemas y sobre todo, pueda manejar los datos que requiera en cualquier momento.

Entradas

Para que ésta tarea sea efectiva necesita tener dos tipos de entradas:

- El diseño del modelo de dominio de la parte cliente.
- El diseño del modelo de dominio de la parte servidor.

Con éstos diseños se puede realizar una conversión exitosa a tablas reales para la base de datos que se vaya a utilizar.

Salidas/Entregables

La salida generada por ésta tarea es un papel que contendrá un mapa esquemático de todas las tablas que tendrá la base de datos con sus columnas definidas y sus tipos de datos asociados.

Recursos necesarios

Ésta tarea sólo requiere la intervención del programador del proyecto.

Precedencias

Ésta tarea precede directamente de las tareas del diseño del diagrama de clases de la parte cliente y la parte servidor.

2. Diseño del proyecto.

Paquete de trabajo: 2.3- Diseño de las pruebas.
Duración: 6 horas.

Descripción

La tarea engloba una de las partes más importantes del proyecto, que es el diseño de las pruebas. La importancia de ésto, radica en que unas pruebas bien diseñadas ayudan a saber si la aplicación desarrollada funciona como debería de esperarse y no da errores indeseados durante su uso.

Un mal diseño puede dar cabida a entregar un software en mal estado que antes o después acabará dando errores serios y que por lo tanto comprometerá su validez como su utilidad real.

Entradas

Como entradas serán necesarias los diagramas de diseño de la parte cliente y de la parte servidor para poder analizar la ejecución de los métodos y valorar qué pruebas serían las más importantes de diseñar.

Salidas/Entregables

La salida que produce ésta tarea es un documento por el cual se expondrá los tipos de pruebas que se van a realizar, qué resultados serían los esperados y sobre todo a qué clases se les va a realizar.

Es importante dejar patente, que también se debe reflejar las posibles observaciones personales para poder incluir información precisa en el caso en el que alguna prueba de un resultado erróneo y se necesite una explicación clara para solventarlo.

Recursos necesarios

En la realización de ésta tarea únicamente es necesaria la participación del programador del proyecto, ya que éste debe definir qué pruebas se han de realizar conforme al código que tiene pensado programar y al diseño que ha generado en tareas anteriores.

Precedencias

Las precedencias de ésta tarea son los diagramas de secuencia de la parte cliente y la parte servidor.

1.6.2.3. Implementación del Proyecto

3. Implementación del proyecto.

Paquete de trabajo: 3.1- Generación de la base de datos.
Duración: 3 horas.

Descripción

La generación de la base de datos consiste en una serie de acciones que implementan una estructura ordenada en una base de datos relacional atendiendo a los parámetros definidos en tareas anteriores(Diseño de la base de datos).

Entradas

Para la consecución efectiva de ésta tarea es necesario el documento en el cual se expone el diseño de la base de datos.

En dicho documento es necesario que se muestre la información necesaria sobre qué nombres tienen las tablas, qué campos tiene cada tabla con sus nombres y sus tipos de datos asignados, y sobre todo, qué relaciones tienen las tablas entre sí.

Salidas/Entregables

La finalización de ésta tarea hace que se cree una bases de datos en un entorno real, lista para poder albergar todos los datos que necesita el proyecto para su correcto funcionamiento.

Recursos necesarios

La realización de esta tarea sólo requiere de la intervención del programador del proyecto para que se pueda hacer de forma satisfactoria.

Precedencias

Ésta tarea precede del diseño de las pruebas, ya que llegados a ese punto, ya estaría implementado de forma correcta el diseño de la base de datos.

3. Implementación del proyecto.

Paquete de trabajo: 3.2- Implantación de los módulos de la parte cliente.
Duración: 35 horas.

Descripción

Consiste en la programación de la parte cliente. En ésta tarea se programará el código necesario para que la parte cliente sea real y funcional.

En ésta tarea se aplicarán todas las herramientas y tecnologías aprendidas para que entre ellas se consiga la funcionalidad deseada.

Entradas

Las entradas para ésta tarea son varias. Algunas son tangibles y otras no lo son debido a que son conocimientos adquiridos. Por lo tanto, dichas entradas son:

- El diseño de las interfaces que se tiene pensado implementar.
- Los conocimientos en materia de programación en interfaces "QT"
- Los conocimientos sobre el uso del lenguaje de programación "Python".
- Los conocimientos sobre el uso y manejo de "BashScrip".
- Los diagramas de secuencia.
- Los diagramas de clases.
- El diseño de la base de datos.
- Conocimientos sobre el uso y manejo de sentencias "SQL" para usar la base de datos.
- Una base de datos ya preparada.

Salidas/Entregables

La salida que genera la tarea será la parte cliente de la aplicación completamente implementada y funcional.

Recursos necesarios

Como en varias tareas anteriores, sólo es necesaria la intervención del programador para que ésta tarea se pueda realizar.

Precedencias

La precedencia es la implementación de la base de datos, clave para saber exactamente qué sentencias y conexiones se deben de realizar.

3. Implementación del proyecto.

Paquete de trabajo: 3.3- Implantación de los módulos de la parte servidor.
Duración: 50 horas.

Descripción

La tarea consiste en la programación de la parte servidor. Aquí se implantará todo el código necesario para que la parte servidor sea funcional.

Entradas

Las entradas para ésta tarea son varias. Algunas son tangibles y otras no lo son debido a que son conocimientos adquiridos. Dichas entradas, son las siguientes:

- El diseño de las interfaces que se tiene pensado implementar.
- Los conocimientos en materia de programación en interfaces "QT".
- Los conocimientos sobre el uso del lenguaje de programación "Python".
- Los conocimientos sobre el uso y manejo de "BashScrip".
- Los diagramas de secuencia.
- Los diagramas de clases.
- El diseño de la base de datos.
- Conocimientos sobre el uso y manejo de sentencias "SQL" para usar la base de datos.
- Una base de datos ya preparada.

Salidas/Entregables

La salida de la tarea es la implementación completa y funcional de la parte servidor que deberá comunicarse de forma correcta con la parte cliente.

Recursos necesarios

Únicamente será necesaria la intervención del programador para la realización de la tarea.

Precedencias

La precedencia de la tarea es la implementación de la parte cliente de la aplicación.

3. Implementación del proyecto.

Paquete de trabajo: 3.4- Pruebas de funcionamiento general.
Duración: 6 horas.

Descripción

En ésta tarea se intenta realizar una serie de pruebas de funcionamiento para saber si los módulos del cliente y del servidor han sido implementados de forma correcta y su funcionamiento produce unas salidas deseables para el contexto de la aplicación.

Entradas

Como entradas, es necesario disponer de la aplicación completada y de un documento que tenga las pruebas de funcionamiento que se desean realizar.

Salidas/Entregables

La salida que genera ésta tarea es un documento en el que se exponen las pruebas que se han realizado y el resultado que se ha arrojado.

En dicho documento se debe reflejar qué tipo de prueba se ha hecho, sobre qué clase, con qué variables y sobre todo, qué resultado se ha obtenido poniendo especial atención a las observaciones que se hayan descrito.

Recursos necesarios

Para la realización de ésta tarea es importante la implicación de las máximas personas posibles. No sólo el programador debe probar el funcionamiento de su programa si no que otros programadores pueden tomar parte para identificar posibles fallos de funcionamiento que no se hayan podido advertir.

Precedencias

La precedencia de ésta tarea es la implantación de los módulos de la parte servidor. De ésta forma se tendrá todo implementado para aplicar las pruebas de funcionamiento.

1.6.2.4. Documentación

4. Documentación.

Paquete de trabajo: 4.1- Manual técnico.
Duración: 3 horas.

Descripción

Consiste en la realización del manual técnico por el cual se expondrán los conceptos del funcionamiento del programa.

Dicho manual debe exponer cómo funciona toda la aplicación y qué funciones tiene para su correcta implantación y ejecución en un servidor. El objetivo de éste manual es que un administrador de sistemas pueda instalar la aplicación y gestionarla para su correcto uso.

Entradas

Como entradas será necesario tener la aplicación correctamente terminada y documentada.

Salidas/Entregables

La salida producida por la tarea es un manual que contiene todas las descripciones de las secciones de funcionamiento de la aplicación.

Recursos necesarios

Sólo es necesaria la intervención del programador de la aplicación de la tarea para terminarla con éxito aunque de forma opcional se puede contar con algún sujeto de pruebas que pueda probar el manual para identificar si está correctamente realizado.

Precedencias

Haber realizado de forma correcta la aplicación y tener toda la documentación realizada.

4. Documentación.

Paquete de trabajo: 4.2- Manual de usuario.
Duración: 3 horas.

Descripción

Ésta tarea engloba la realización de un manual de usuario para que cualquier persona con unos mínimos básicos de informática pueda hacer uso de la aplicación ya implementada y preparada por el administrador de sistemas.

Entradas

Como entradas es necesario disponer de la aplicación completada y de su documentación.

Salidas/Entregables

Como salida se generará un documento que expondrá los pasos necesarios para el uso del proyecto de forma sencilla para un usuario medio.

Recursos necesarios

Se necesita el programador del proyecto para la realización del manual y de forma opcional algún usuario medio para probar que es lo suficientemente explicativa para que pueda ser entendida por cualquier persona.

Precedencias

La precedencia de ésta tarea es haber haber realizado previamente el manual técnico.

1.7. Planificación temporal

A continuación se mostrará una tabla con la descripción detallada de las horas necesarias para completar el proyecto:

	Duración	Actividad siguiente	Tipo relación	Tiempo demora
Tarea 1.1	1 hora	Tarea 1.2	Comienzo/Fin	0
Tarea 1.2	4 horas	Tarea 1.3	Comienzo/Fin	0
Tarea 1.3	3 horas	Tarea 1.4	Comienzo/Fin	0
Tarea 1.4	20 horas	Tarea 1.5	Comienzo/Fin	0
Tarea 1.5	3 horas	Tarea 1.6	Comienzo/Fin	0
Tarea 1.6	1 hora	Tarea 1.7	Comienzo/Fin	0
Tarea 1.7	2 horas	Tarea 1.8	Comienzo/Fin	0
Tarea 1.8	15 horas	Tarea 1.9	Comienzo/Fin	0
Tarea 1.9	25 horas	Tarea 2.1.1.1	Comienzo/Fin	0
Tarea 2.1.1.1	5 horas	Tarea 2.1.1.2	Comienzo/Fin	0
Tarea 2.1.1.2	7 horas	Tarea 2.1.1.3	Comienzo/Fin	0
Tarea 2.1.1.3	5 horas	Tarea 2.1.1.4	Comienzo/Fin	0
Tarea 2.1.1.4	8 horas	Tarea 2.1.2.1	Comienzo/Fin	0
Tarea 2.1.2.1	7 horas	Tarea 2.1.2.2	Comienzo/Fin	0
Tarea 2.1.2.2	9 horas	Tarea 2.1.2.3	Comienzo/Fin	0
Tarea 2.1.2.3	6 horas	Tarea 2.1.2.4	Comienzo/Fin	0
Tarea 2.1.2.4	8 horas	Tarea 2.2	Comienzo/Fin	0
Tarea 2.2	6 horas	Tarea 2.3	Comienzo/Fin	0
Tarea 2.3	6 horas	Tarea 3.1	Comienzo/Fin	0
Tarea 3.1	3 horas	Tarea 3.2	Comienzo/Fin	0
Tarea 3.2	35 horas	Tarea 3.3	Comienzo/Fin	0
Tarea 3.3	50 horas	Tarea 3.4	Comienzo/Fin	0
Tarea 3.4	6 horas	Tarea 4.1	Comienzo/Fin	0
Tarea 4.1	3 horas	Tarea 4.2	Comienzo/Fin	0
Tarea 4.2	3 horas	—	Comienzo/Fin	0

Tabla 1.1: Tareas del proyecto.

El computo total de horas calculado es de **241** horas efectivas, lo que viene siendo aproximadamente unos 10 días de trabajo continuado.

Si tenemos en cuenta, los descansos de los fines de semana, que en un día se trabaja unas horas determinadas, fiestas etc... el tiempo completo de demora de días puede situarse alrededor de los **31,13** días.

1.8. Riesgos

Durante la consecución del proyecto pueden darte una serie de incidencias. Por ello, es importante enumerarlas y categorizarlas de forma correcta para poder generar un plan de contingencia efectivo.

En ésta sección se va a enumerar una serie de posibles riesgos que pueden aparecer durante la realización del proyecto y cuáles son los mejores métodos para poder reconocerlos, evitarlos y subsanarlos en caso de que se mostraran.

1.8.1. Falta de conocimiento del desarrollador en alguna tecnología

1.8.1.1. Descripción

Es un problema muy serio que puede aparecer en cualquier momento. El programador se ve capaz de hacer todo un proyecto sin ningún problema y por ello subestima la complejidad del mismo.

Subestimar la dificultad de un proyecto o simplemente sobrestimar las capacidades personales, puede llegar a ser un grave error debido a que en algún punto del proyecto puede darse la circunstancia de no poder avanzar debido a la falta de determinados conocimientos que se supone que debían estar adquiridos o consolidados.

1.8.1.2. Plan de prevención

Para prevenir la aparición de éste fatídico momento es imperativo tener en cuenta las siguientes cuestiones:

1. Tener una mentalidad muy humilde. Es importante tomarse en serio la medida de dificultad que acarrea el proyecto. Por ello, aunque se tenga determinada experiencia no está de más ir con cuidado y hacer de forma correcta todos los pasos necesarios de una forma seria y rigurosa para evitar sustos durante la realización de cada tarea.

2. Es necesario que el programador obtenga conocimientos en materia de las aplicaciones que necesita aprender. Sobre todo repasar las materias que ya se saben e intentar ampliar un poco más para que, con los conocimientos avanzados adquiridos, puedan resolverse situaciones puntuales o simplemente facilitar la consecución de ciertas áreas que puedan dar problemas en un futuro.

3. Escuchar y aprender sobre personas experimentadas. No está de más, ir a charlas o hablar con docentes y profesionales de la informática para obtener ciertos consejos sobre tecnologías y sus posibles problemas y soluciones asociadas para que en un futuro, si se diera el caso, poder estar formado adecuadamente para encarar un problema que pudiera aparecer de forma no esperada.

1.8.1.3. Probabilidad

La probabilidad de aparición de un riesgo de éste calibre es variable y viene ligada a los conocimientos que tenga el programador en base a las tecnologías que use en el proyecto y su experiencia sobre ellas.

1.8.1.4. Impacto

El impacto sobre el proyecto es muy grande debido a que la falta de conocimiento sobre una herramienta es altamente perjudicial y puede provocar problemas graves a la hora de la realización del proyecto.

Una falta de conocimiento se puede traducir a la paralización de una parte o partes del proyecto e incluso al abandono definitivo del mismo.

1.8.1.5. Plan de contingencia

Si éste riesgo se sucediera, la única solución posible sería la búsqueda externa de ayuda, ya sea mediante Internet, libros o personas que puedan escuchar qué ha sucedido y puedan dar una solución que ayuden a superar el muro que impide al desarrolador avanzar en su proyecto.

1.8.2. Baja moral y/o desmotivación

1.8.2.1. Descripción

Un posible peligro dentro de a ejecución del proyecto es la falta de motivación o una baja moral. Puede ocurrir durante el ciclo del proyecto y es importante subsanarlo para poder obtener los mejores resultados posibles.

La desmotivación suele aparecer cuando el programador no es capaz de completar de forma eficaz las competencias necesarias de su proyecto o cuando por determinadas circunstancias el proyecto pierde el atractivo que inicialmente tenía.

1.8.2.2. Plan de prevención

Una buena forma de prevenir un riesgo tan problemático, es hacer una rotación entre las diferentes tareas que se deben realizar. Por ejemplo, rotar entre tareas más fáciles y tareas más complejas, de éste modo, el proyecto se hará más dinámico y no se caerá en una rutina asfixiante.

1.8.2.3. Probabilidad

La probabilidad, en general, suele ser bastante baja, debido a que el programador escoge el proyecto que más interés le suscita por lo que es complicado que pueda llegar a desmotivarse y no querer seguir con él.

1.8.2.4. Impacto

El impacto de éste riesgo es demasiado alto. Una falta de motivación puede repercutir en la calidad del proyecto e incluso puede hacer que éste no se finalice. Además, un proyecto realizado por un programador con una moral baja, es susceptible a no ser estable ni estar debidamente hecho.

1.8.2.5. Plan de contingencia

Si por algún motivo se disparara éste riesgo, las mejores opciones serían:

- Dejar la tarea de inmediato y dedicarse a una tarea que sea más motivadora o más sencilla de realizar.
- Parar automáticamente la realización de la tarea y tomarse un descanso para recuperar la motivación necesaria.
- Estudiar alternativas "¿Es necesario hacer ésto en mi proyecto?" "¿Podría gustarme más si aplico otros criterios?"

1.8.3. Enfermedades

1.8.3.1. Descripción

Es un riesgo muy general y bastante conocido a la hora de planificar los riesgos. Cualquier persona se puede enfermar en cualquier momento del ciclo de realización del proyecto, es algo no controlable pero se puede prever fácilmente.

Normalmente, se sabe qué enfermedades pueden aparecer dependiendo de las épocas del año por lo que se puede planificar fácilmente los plazos de entrega sin que una o dos enfermedades afecten a dichos plazos.

1.8.3.2. Plan de prevención

Éste riesgo no se puede prevenir. No existe manera de adivinar cuando el programador se pone enfermo o no pero, lo que si se puede hacer, es incluir en los diagramas temporales una pequeña holgura para subsanar el impacto de una posible enfermedad.

1.8.3.3. Probabilidad

La aparición de éste riesgo suele ser, en la mayoría de las veces, bastante baja. Todo depende mucho de la persona y de la época del año.

Se debe tener en cuenta si el programador es muy propenso a contraer determinados tipos de enfermedades y basado en ello, se debe estudiar la época del año para hacerse una idea general de la probabilidad de que el riesgo se produzca.

1.8.3.4. Impacto

El impacto que puede ocasionar en éste TFG es muy grande. Al ser un proyecto realizado por un único programador, la incapacidad de éste para trabajar en el proyecto debido a una enfermedad ocasiona en un 100 % la detención del mismo.

1.8.3.5. Plan de contingencia

Si éste riesgo aparece, en el caso de éste proyecto en particular, no se podría aplicar un plan de contingencia adecuado.

La razón es simple, al ser un proyecto realizado por un sólo programador no existe forma posible de que otra persona pueda continuar con su labor mientras éste se encuentra indispuesto. Por ello no se puede aplicar un protocolo específico.

1.8.4. Fallo en el disco duro

1.8.4.1. Descripción

Todos los equipos informáticos son susceptibles a tener algún error de hardware. Entre ellos y sin duda uno de los más peligrosos es el fallo del disco duro.

Un disco duro es un elemento muy delicado y que se degrada fácilmente con el uso. Por ello, si no se tiene preparado un buen plan de contingencia se puede llegar a perder información muy valiosa y sobre todo, se puede perder muchas horas de trabajo.

1.8.4.2. Plan de prevención

Es prácticamente imposible prevenir un fallo en el disco duro, pero si es muy fácil mitigar su aparición y sobre todo evitar la pérdida de datos que el riesgo puede ocasionar.

Para ello es recomendable realizar las siguientes operaciones:

- Evitar golpear los equipos informáticos, especialmente las zonas donde se encuentra el disco duro. Los discos duros son muy sensibles.
- Dejar siempre que el sistema operativo se apague completamente. Ésto sobre todo se debe aplicar a los ordenadores portátiles, si se apaga el equipo informático de forma incorrecta la aguja del disco duro magnético no se posiciona de forma correcta(la aguja debe estar completamente retraída) y por ello al mover el disco duro, existe una posibilidad de que la aguja raye los platos y deteriore el disco magnético.
- Realizar copias de seguridad. Es una de las opciones más usadas. Hacer copias de seguridad en un medio externo al disco duro, ayuda a recuperar los datos del proyecto en caso que la unidad actual quede totalmente inoperativa.

1.8.4.3. Probabilidad

La probabilidad de aparición de ésta riesgo, suele ser entre media y baja. Un fallo en un disco duro no suele ser algo común y suele ser algo aleatorio dependiendo de la carga a la que se le someta y el tiempo de uso del mismo.

1.8.4.4. Impacto

El impacto es bastante alto. Si un disco duro deja de funcionar, las herramientas necesarias para el programador se quedan completamente fuera de juego. Además, existe un riesgo de perder el trabajo realizado y dicho riesgo sólo se podrá subsanar dependiendo del plan de prevención que se haya diseñado.

1.8.4.5. Plan de contingencia

Si por alguna cuestión el riesgo se produce se deberían seguir los siguientes pasos:

1. Identificar si el fallo producido es debido al disco duro y calcular la magnitud del fallo(el disco duro podría estar estropeado pero funcionar parcialmente).
2. Mediante el uso de programas especiales; intentar recuperar los datos de interés, en éste caso, los datos referentes al proyecto.
3. Obtener la última copia de seguridad(si existiera) y cotejarla con los datos recuperados del disco duro(si ha sido posible).
4. Unir los datos que falten para recabar la información más actualiza posible.
5. Obtener un nuevo medio de almacenamiento y volcar los datos.
6. Valorar las pérdidas de datos y rehacer el trabajo que se haya perdido.
7. Realizar una nueva copia de seguridad con los nuevos datos y las partes faltantes agregadas.

1.8.5. Fallo de la conexión de internet

1.8.5.1. Descripción

Uno de los riesgos más comunes que pueden aparecer es un fallo de la conexión a Internet. Dicho fallo se reproduce en la imposibilidad de que el programador pueda conectarse a Internet para obtener información relativa al trabajo fin de grado o información determinante para el uso de las herramientas.

Además, durante la vigencia del error, el programador no puede hablar con su profesor para solventar las dudas que tuviera y mucho menos conseguir copias de su proyecto.

1.8.5.2. Plan de prevención

Una forma efectiva de prevenir éste riesgo es portando un dispositivo externo que tenga acceso a Internet. Por ejemplo, un pincho móvil(módem conectado al puerto "USB" de un ordenador) o un "smartphone" que sirva de punto de acceso Wifi.

De ésta forma, en caso de que el riesgo aparezca y sea muy necesaria una conexión a Internet, el programador tendrá una vía de emergencia para resolver cualquier necesidad.

1.8.5.3. Probabilidad

Por norma general, la probabilidad de que éste riesgo sea una realidad es baja. Pero todo depende del tipo de conexión y de la fiabilidad y calidad del proveedor de acceso a Internet.

1.8.5.4. Impacto

El impacto no es muy severo ya que la falta de acceso a Internet no interrumpe el proceso de creación del proyecto. Si el riesgo se produce en un espacio de tiempo corto el impacto puede ser prácticamente nulo.

1.8.5.5. Plan de contingencia

Si por algún motivo, el acceso a Internet se produjera y fuera necesario una conexión urgente las posibles soluciones serían las siguientes:

- Si es posible, desplazarse con el equipo de trabajo a un lugar donde exista una conexión funcional y conectarse a ella.
- Si se tiene un dispositivo externo de conexión a Internet, configurarlo y hacer uso de el.
- Si se tiene un smartphone, crear un punto de acceso wifi para conectarse a internet.

1.8.6. Errores humanos producidos sin intencionalidad

1.8.6.1. Descripción

Durante la realización del proyecto se pueden producir errores humanos sin intención. Esto, por ejemplo, se produce cuando un programador no adopta una serie de medidas para proteger los datos de su proyecto sobre personas ajenas.

Dejarse el ordenador encendido sin supervisión, dejar contraseñas puestas o sesiones abiertas, olvidarse de guardar ciertos ficheros etc... Todo ésto son errores que pueden producirse sin intención y que pueden peligrar el trabajo realizado.

1.8.6.2. Plan de prevención

Los errores humanos no se pueden prevenir, no existe un mecanismo ni una serie de pautas que eviten que se produzcan. Pero si que es posible establecer una serie de criterios para mitigar la aparición de éste riesgo.

- Nunca dejar contraseñas abiertas ni divulgar información referente a ellas.
- Proteger con sistemas de control(encriptación, verificación en dos pasos, contraseñas....) todos los datos y equipos informáticos en los que se tenga acceso.
- Guardar una copia de seguridad en un equipo seguro y externo por si se produce una alteración no deseada de datos.
- Concienciarse de la importancia del trabajo realizado y la necesidad de seguir una serie de pautas para evitar peligrar su integridad.
- Cambiar cada cierto tiempo las contraseñas de los accesos a los servicios.
- Establecer rutinas de control para verificar que todos los pasos realizados se están haciendo de forma segura.
- Verificar en determinados intervalos temporales, la integridad completa del proyecto y los accesos que se han hecho al mismo.

1.8.6.3. Probabilidad

La probabilidad de la aparición del riesgo es media-alta debido a que un programador puede no darse cuenta de que ha cometido un error y todo dependerá del alcance del error cometido.

1.8.6.4. Impacto

El impacto es variable dependiendo de cuan serio haya sido el error cometido por el programador. Siendo, por ejemplo, un impacto grande si se ha dejado un ordenador encendido y sin ninguna supervisión de todos los programas y accesos abiertos.

1.8.6.5. Plan de contingencia

Si el riesgo se produce es recomendable realizar los siguientes pasos:

1. Evaluar que partes del proyecto o de las herramientas han sido accedidas sin permiso.

2. Cambiar todas las contraseñas y claves de acceso.

3. Respaldar con una copia de seguridad, aquellas partes que se creen que han sido manipuladas.

4. Estudiar las razones del fallo ¿Por qué se ha producido?

5. Establecer nuevos protocolos para evitar que en un futuro cercano vuelva a producirse el riesgo.

1.8.7. Cambio de versión en Python

1.8.7.1. Descripción

Consiste en un riesgo que aparece cuando se produce un salto de versión mayor. Dicho salto tiene como consecuencia el cambio de una o varias partes de la lógica del lenguaje de programación, haciendo que las variables o funciones usadas en una versión anterior, no sean reconocidas en versiones posteriores creando una incompatibilidad con la aplicación y produciendo una necesidad de reescribir varias partes del código ya generando.

1.8.7.2. Plan de prevención

Existen dos formas fundamentales de prevenir un riesgo de ésta característica:

1. Estar al corriente de noticias en torno al lenguaje de programación para ver qué futuros cambios se van a realizar y adoptar las medidas necesarias para evitar cualquier problema que pueda producirse a la hora de realizar el salto de versión.

2. Tener un programa bien diseñado e implementado, con funciones muy estándares para evitar que a la hora de realizar el salto de versión, no se vean afectadas. Por ejemplo, existen variables o funciones que son comunes en todos los lenguajes de programación y éstas no van a variar por mucho que un lenguaje de programación avance de versión.

1.8.7.3. Probabilidad

La probabilidad de la aparición de un riesgo de ésta importancia es baja ya que el salto de versión mayor de un lenguaje de programación no es algo que se haga de forma continuada. Pueden pasar muchos años desde la realización del proyecto hasta que el salto se produzca.

1.8.7.4. Impacto

El impacto de éste riesgo es muy alto debido a las consecuencias que puede ocasionar el cambio brusco de más de alguna funcionalidad.

1.8.7.5. Plan de contingencia

Si por algún motivo el proyecto realizado sufre problemas con una nueva versión se pueden seguir las siguientes recomendaciones:

- Obtener unas librerías antiguas compatibles con la versión del lenguaje de programación usado y hacer uso de éstas para hacerlo funcionar.
- Identificar los puntos de error en el proyecto y, haciendo uso de la documentación, modificar dichos puntos para que converjan con las directrices establecidas por la nueva versión del lenguaje de programación.

1.8.8. Cambio de la versión de las librerías gráficas QT

1.8.8.1. Descripción

El cambio de versión de las librerías gráficas es un problema que puede aparecer durante la realización del proyecto. Actualmente, durante la realización del mismo, se está produciendo un cambio de versión mayor de las librerías gráficas "QT" de la versión 4 a la versión 5.

Éste salto introduce un cambio profundo en el funcionamiento y políticas de las librerías que podrían echar por tierra, todo el trabajo realizado con las librerías de la versión 4.

¿Y a qué es debido ésta razón? A que ciertas herramientas como "PyQT" podrían dar problemas al estar pensadas para ser usadas con la versión 4 en vez de la versión 5 por lo que quedarían inútiles y entorpecerían la finalización del proyecto.

1.8.8.2. Plan de prevención

La única forma de prevenir la aparición de un riesgo de ésta índole, es estar al tanto de las noticias que genera la comunidad de desarrollo de "QT" para saber si se tiene pensado sacar una versión mayor de éstas y qué cambios puede conllevar.

Por tanto, la única forma de prevenir un error así, es actualizando las interfaces con los nuevos estándares que se definan en un futuro y estar al tanto de las actualizaciones de las herramientas que dependen de las interfaces.

1.8.8.3. Probabilidad

La probabilidad de aparición de un riesgo de ésta índole es bastante baja ya que las actualizaciones de librerías se hacen cada ciertos años y el soporte suele durar algo más.

1.8.8.4. Impacto

El impacto que produce éste riesgo es muy alto. El cambio de librerías a una nueva versión hace que las antiguas se queden sin soporte y que por lo tanto con el tiempo se puedan producir incompatibilidades con algunos entornos de escritorio.

1.8.8.5. Plan de contingencia

En caso de que por alguna razón éste riesgo aparezca se deberían hacer los siguiente:

1. Repasar la documentación de las librerías gráficas para saber exactamente qué cambios nuevos traen las versiones posteriores y a qué funciones afectan.
2. Buscar en el código implementado aquellas partes que den error con las nuevas librerías y anotarlas.
3. Realizar los cambios correspondientes comprobando que la adaptación sea correcta.

Pero en ocasiones, éste "modus operandi" puede llegar a no ser efectivo. Pudiera suceder que por cuestiones de la nueva versión de deban de rediseñar completamente todas las interfaces del proyecto y volverlas a adaptar al lenguaje Python. En ese caso, se debería de rescatar los diagramas de diseño y las clases asociadas con las interfaces para poder modificarlas y hacerlas compatibles con las nuevas interfaces que se realicen.

1.8.9. Riesgos en la implantación del proyecto en otros sistemas operativos

1.8.9.1. Descripción

Un riesgo muy peculiar dentro de éste TFG es la aparición de problemas a la hora de querer adaptar el proyecto en otros sistemas operativos.

Al tratase de un programa informático tradicional que se instala dentro de un entorno operativo, es necesario que el programa cumpla una serie de requisitos para que funcione como se espera. Requisitos como, la arquitectura, el tipo de sistema operativo, la forma de instalar el software, las librerías y dependencias que necesita e incluso la lógica del sistema de ficheros.

Por ello, un riesgo bastante importante es tener la capacidad para poder hacer que una misma aplicación funcione en distintos entornos operativos sin tener que estar rehaciendo parte de su código para adaptarla a un sistema determinado.

Incluso, se debe tener al menos una noción para saber que las librerías o herramientas utilizadas tienen una versión preparada para los otros sistemas operativos y que se puede hacer uso de las mismas si ningún tipo de restricción.

1.8.9.2. Plan de prevención

La mejor manera de prevenir la aparición de un riesgo de éstas características es haciendo un estudio previo del tipo de aplicación que se desea desarrollar y de la compatibilidad de las librerías y herramientas que se pretenden usar en los distintos sistemas operativos en los que se desee hacer funcionar la aplicación.

Además, también es importante comprobar que el rendimiento que sufra la aplicación a la hora de cambiar de sistema operativo no se vea afectado. Incluso también es importante verificar que el uso de ciertas librerías o herramientas en otros entornos operativos no obliguen a reprogramar ciertas áreas del proyecto para hacer el sistema compatible.

1.8.9.3. Probabilidad

La probabilidad de la aparición de un riesgo de éste tipo es media-baja. Si se tiene constancia de antemano en que entornos operativos debe funcionar el proyecto y si se hace un estudio previo satisfactorio de las compatibilidades de las librerías o herramientas a usar, es poco probable que el riesgo pueda llegar a reproducirse.

No obstante, aún sabiendo que todo puede ir bien hay dejar abierta la posibilidad de que por decisiones de cambios en la naturaleza del programa, éste error sea susceptible de aflorar.

1.8.9.4. Impacto

El impacto de éste riesgo de muy alto debido a que su aparición puede hacer que se produzcan graves retrasos en las planificaciones establecidas para el proyecto e incluso puede hacer que el proyecto se quede bloqueado por no existir una serie de librerías adaptadas y funcionales para un sistema operativo en particular.

1.8.9.5. Plan de contingencia

Si debido a ciertas decisiones o cuestiones inesperadas el riesgo apareciese, se pueden tomar los siguientes pasos para intentar subsanarlo:

1. Identificar qué librerías están dando problemas a la hora de adaptar el proyecto en otro sistema operativo.

2. Identificadas dichas librerías, intentar encontrar documentación referente a las mismas ¿Están integradas en el sistema operativo? ¿Se pueden instalar de forma oficial desde la página web del autor? ¿El sistema operativo las ofrece como parte de la paquetería de instalación?

3. En caso de no encontrar nada, buscar el código fuente de las librerías afectadas. Ir a la página web del autor de cada librería y obtener las fuentes.

4. Si se poseen las fuentes de cada librería, buscar información sobre cómo compilarlas para cada sistema operativo teniendo muy en cuenta las arquitecturas en las que se tiene pensado hacer funcionar la aplicación.

5. Incluir las librerías compiladas y comprobar que el programa funciona de manera óptima y sin dar problemas.

1.9. Evaluación económica

La realización del proyecto tiene una serie de costes asociados. Dichos costes se dividen en costes fijos y costes variables.

Por tanto, en primer lugar, se va a exponer una tabla la cual va a reflejar los valores de los costes para tener una referencia básica. Con ésta referencia se podrá tener una idea global de los costes que pueden generarse en cada hora de trabajo realizada.

En segundo lugar, se hará una evaluación económica que estime el coste total de la realización del proyecto y que calcule si el proyecto pudiera ser o no viable.

A continuación, se va a mostrar la tabla con los costes:

Costes fijos		Costes variables	
Concepto	Precio(hora)	Concepto	Precio(global)
Luz	0,07 €	Consumibles	50 €
Agua	0,02 €	Transporte	42,19 €
Telefonía	0,04 €		
Internet	0,07 €		
Gas	0,01 €		
Salario	25 €		
Total/hora	25,21 €	**Total/global**	92,19 €

Tabla 1.2: Costes fijos y costes variables

Una vez vista la relación de los costes fijos y variables podemos por lo tanto estimar el conste total del proyecto.

Sabiendo que el proyecto tiene una duración aproximada de unas 241 horas el coste total del proyecto asciende a la siguiente cantidad:

6167,8 €

Un profesor necesita aproximadamente una media de unos 45-50 minutos de su tiempo de trabajo para dar de alta a 40 alumnos de una única asignatura en varios servicios determinados. Éste tiempo, es oro y además teniendo en cuenta que un profesor cobra una media de unos 20 € la hora se puede vislumbrar de una manera sencilla que no es para nada rentable que un docente dedique su tiempo efectivo de trabajo a dar de alta a los alumnos en una serie de sistemas determinados.

Todo éste análisis está enfocado a un único profesor que al menos posee una serie de conocimientos informáticos que le hacen capaz de dar de alta a los alumnos en distintos servicios.

Ahora extrapolemos el caso a varios profesores, muchos de ellos sin conocimientos informáticos y sobre varias asignaturas con un número indeterminado de alumnos. El tiempo medio comentado anteriormente puede dispararse e incluso se puede correr el riesgo que el profesor sea incapaz de de dar de alta a los alumnos en los distintos sistemas y por ende deba de acudir a donde otro profesor para pedir ayuda. Ésto se traduce en una perdida sustancial de tiempo y sobre todo a una perdida indiscriminada de dinero.

Si reflexionamos entonces, es visible que el resultado es más que escalofriante y por ello se puede deducir que el coste final del programa es insignificante teniendo en cuenta las pérdidas que pudiera ocasionar el tiempo invertido por un docente para dar de alta a una serie de alumnos.

Por lo tanto se puede decir que el proyecto en cuestión es viable. El coste total es justificado y coherente, de hecho, es hasta muy barato considerando la magnitud del trabajo a realizar y el ahorro que supone a la larga para la universidad sin descontar la liberación que le supone a un profesor en su carga de trabajo, ganando éste en satisfacción y mejorando sensiblemente su nivel de productividad.

Capítulo 2

Antecedentes

2.1. Descripción de la situación actual

Hoy en día, para que un profesor pueda dar de alta a un alumno en un servicio determinado necesita realizar una buena serie de pasos:

1. Obtener los datos del alumno deseado; nombre, apellidos, Dni, etc...
2. Conectarse en el servicio con sus credenciales que le den permisos de registro.
3. Añadir los datos del nuevo alumno al servicio en cuestión.
4. Cercionarse que todos los datos introducidos son correctos.
5. Hacer llegar al alumno todos los datos necesarios para que pueda hacer uso del registro que se le ha realizado.

Si nos ponemos en situación, nos damos cuenta que hacer todos éstos pasos por cada alumno puede llegar a ser un problema. Ya de por si, hacerlo con uno resulta un poco complejo, hacerlo con muchos resulta una odisea. Además, si hacemos balance del tiempo empleado, nos damos cuenta que al final el problema no es sólo la fatiga generada al tener que dar de alta a tantos alumnos, si no que el tiempo que lleva hacerlo hace que otras tareas no se puedan realizar.

Ésta situación requiere de un programa que automatice el proceso y ayude a ahorrar tiempo. Aquí es donde entra "Akeko", se intenta conseguir que éste proceso sea automático y que a la vez sea fácil de gestionar porque en un futuro, las necesidades pueden cambiar y por lo tanto es importante tener un buen orden de las decisiones que se hayan tomado.

Un programa de éstas características es único. Es cierto que pueden existir programas similares pero, en éste caso, nos encontramos con un desarrollo especifico que busca un fin particular y personalizado y ello no lo da cualquier programa.

Podrían existir alternativas que ayudaran con la gestión de los usuarios. Por ejemplo un ERP(Sistema de planificación de recursos empresariales), pero aún así, el juego que da la automatización de los procesos de alta y baja de servicios de "Akeko" no lo puede dar ninguna aplicación ya que la idea es bastante innovadora.

2.2. Estudio de las alternativas posibles

Hacer un programa de ésta índole tiene un punto fuerte muy positivo: la capacidad de poder tener un amplio abanico de posibilidades.

Éste TFG se puede realizar de muchas maneras, bajo muchos lenguajes de programación y bajo muchas metodologías de trabajo. Podríamos enumerar varias alternativas posibles.

- **Aplicación para "smartphones"**. Se puede hacer una aplicación para móviles inteligentes, a decidir entre los distintos sistemas operativo móviles actuales. En dicho sistema, sólo es necesario hacer un desarrollo fácil e intuitivo que permita desde cualquier lugar, conectarse al programa y hacer las gestiones que sean necesarias.

- **Aplicación web**. Quizás lo más requerido y popular hoy en día es la realización de una página web. La página web ofrecería una interfaz modular que permitiría al usuario interactuar de forma directa con el servidor y realizar las gestiones necesarias. Una punto muy positivo de ésta alternativa sería la posibilidad de hacer funcionar el proyecto en casi cualquier sistema operativo, ya sea de escritorio o móvil.

- **Aplicación de terminal**. Y porque no, para aquellos puristas de la consola se podría hacer una aplicación completamente factible y funcional que permita hacer la gestión mediante sencillas interfaces en una consola virtual. El punto negativo de ésta alternativa es que sería poco agradable a la vista de cara al usuario más novel en cuestiones informáticas.

- **Aplicación clásica cliente-servidor**. Es el método escogido dentro de éste TFG pero incluso dentro de éste método se pueden hacer distintas variaciones. Por ejemplo, se pueden elegir distintos tipos de lenguajes de manipulación de interfaces para dejar una puerta abierta a cambios visuales, como por ejemplo, los colores o las tipografías. También se puede establecer las gestiones con el servidor, se pueden pedir conexiones síncronas o asíncronas y un manejo avanzado de hilos de ejecución para cada usuario conectado al servidor.

Además dependiendo del método elegido, se puede aplicar una metodología de trabajo determinada:

1. **Modelo en cascada**. Se puede establecer de forma rigurosas cada etapa con un tiempo determinado y con un orden fijo. Cada vez que se complete una etapa se empezará con la siguiente y así sucesivamente.

2. **Modelo en V**. Es una variación del modelo en cascada en la cual indica que las pruebas deben hacerse lo antes posible en el ciclo de vida de la aplicación y completarse de forma paralela con la etapa actual. La idea es que en cada etapa se desarrolle a la vez un plan de pruebas para verificar el correcto funcionamiento de la aplicación antes de continuar con la siguiente etapa.

3. **Modelo iterativo**. Es otra variación del modelo en cascada, se trata de iterar varias veces sobre un mismo modelo en cascada. En cada iteración se produce una nueva versión del software que deberá ser evaluada por el cliente, el cual establecerá que cuestiones deben mejorarse.

4. **Modelo de desarrollo incremental**. Éste modelo combina las ideas del modelo en cascada, con la filosofía interactiva de la construcción de prototipos. La idea de éste modelo es hacer un ciclo en cascada y producir un software sencillo, básico y funcional. Una vez hecho se repite el ciclo pero añadiendo nuevos componentes que mejoren el software. Al final de varias repeticiones, se obtiene el software completo.

5. **Modelo en espiral**. Las actividades se distribuyen en una espiral en la cual cada bucle representa un conjunto de dichas actividades. Las actividades no se fijan de antemano si no que se van ajustando dependiendo del análisis de riesgos que se realice y siempre comenzando desde el bucle anterior. Se podría decir que la metodología es sencilla, se hace un análisis de riesgos de las mejores alternativas posibles y se escoge la que aparentemente tenga menos riesgos y se hace una vuelta en la espiral. Tras haberlo hecho, se repite el proceso de nuevo y se da una nueva vuelta a la espiral. Se repite sucesivamente hasta que el software quede libre de posibles riesgos.

6. **Modelo de prototipos**. Éste modelo consiste en una conexión directa entre cliente y desarrollador por el cual se establecen unos criterios rápidos y se genera una versión sencilla y visual del software. La idea es que el cliente vea de antemano el posible funcionamiento y haga una evaluación. Con dicha evaluación, se repite el proceso realizando un nuevo prototipo e incluyendo los cambios pedidos. El proceso se repite varias veces entregando cada nuevo prototipo al cliente para que éste decida cómo se deben hacer las cosas y que rumbo debe tomar la aplicación. Tras varios prototipos, el programa se da como bueno y se entrega. [26]

7. **Metodologías ágiles**. Consiste en un nuevo modelo por el cual se desarrolla software haciendo uso de unas técnicas más innovadoras. La idea consiste en juntar al equipo de trabajo cada semana para hacer una evaluación del software a completar, realizar una evaluación de lo que se ha hecho, proponer nuevas ideas y sobre todo, fijar qué se va a hacer durante la siguiente semana de trabajo. De ésta forma, se va creando un software incremental y en cada reunión el equipo de trabajo toma las decisiones de lo que se debe hacer en los próximos días. Éste modelo apuesta por la comunicación directa y cara a cara más que en la documentación y busca resolver los cambios y las incidencias posibles de una manera veloz.

Al final, todo depende de cómo se quiera realizar el proyecto y del resultado esperado. Quizás, para un proyecto de éstas características, una alternativa de gran peso sería la aplicación web pero como ya se ha expuesto anteriormente, el interés de hacer un programa que sea una aplicación clásica cliente-servidor viene dada por cuestiones educativas. Además, debido a las características del proyecto y el tipo de TFG se ha optado por un modelo clásico en cascada que en éste caso, representa de una mejor manera la idea de cómo hacer las cosas debido a la naturaleza del propio proyecto a realizar.

Capítulo 3

Captura de requisitos

3.1. Introducción

En éste apartado se va a realizar el estudio de los casos de uso que contiene el proyecto. Para ello se va a dividir la labor en dos apartados: Uno destinado a la parte cliente y otro destinado a la parte servidor.

3.2. Casos de uso del sistema (Parte Cliente)

En ésta sección se va a desengranar los casos de uso que afectan a las funcionalidades de la parte cliente de la aplicación.

3.2.1. Diagrama de casos de uso

A continuación se va a mostrar el diagrama completo de casos de uso de la parte cliente de la aplicación:

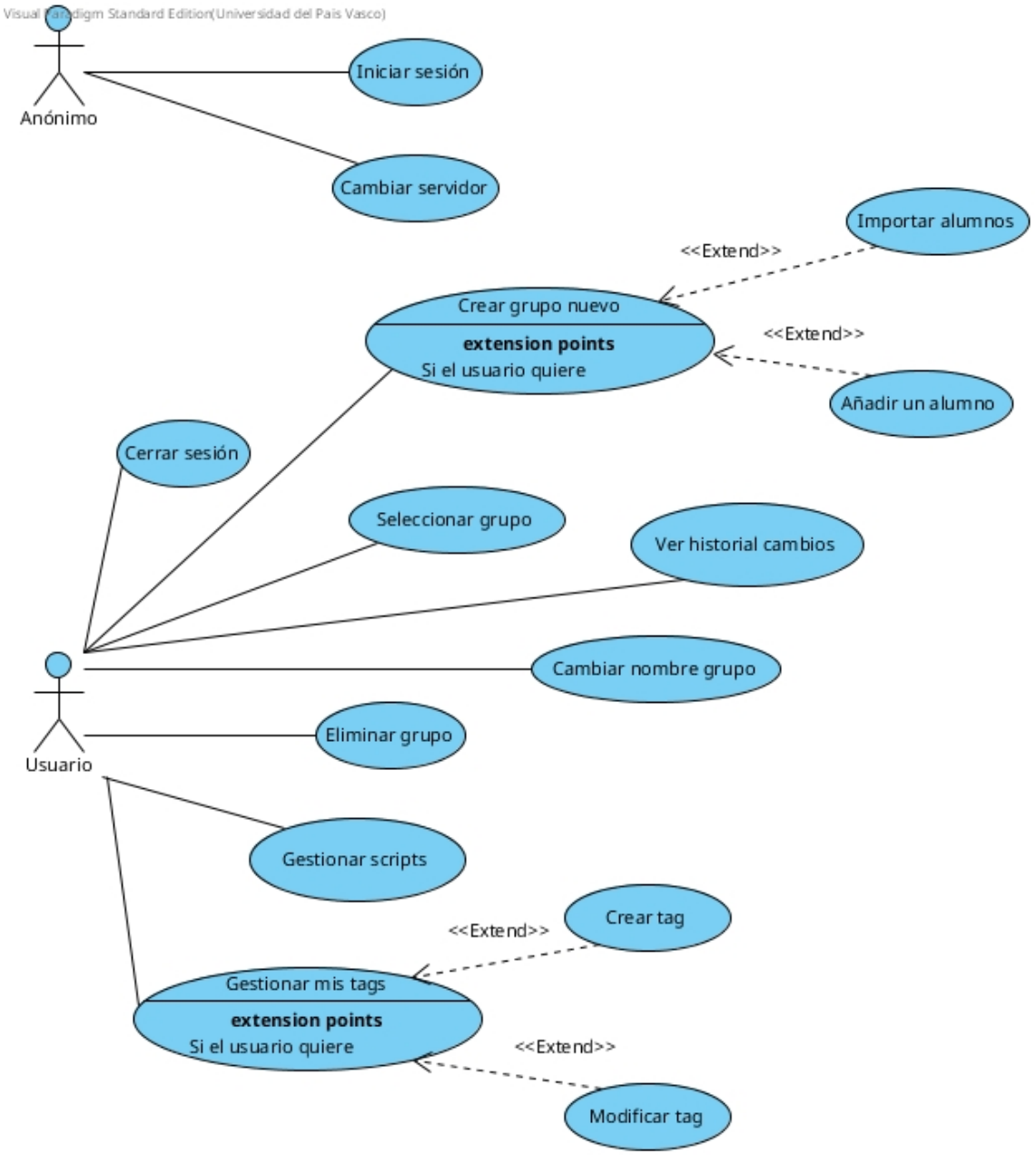


Figura 3.1: Diagrama de casos de uso de la parte cliente

3.2.2. Jerarquía de actores

A continuación de describen los diferentes actores:

1. **Anónimo:** Consiste en el usuario que no está dentro del sistema. Éste anónimo tendrá una serie de acciones muy limitadas a la hora de interactuar con la aplicación.

2. **Usuario:** Éste rol simboliza el usuario que se ha introducido en el sistema por la parte cliente de la aplicación. Obtendrá el privilegio de usar las opciones disponibles para la gestión de los alumnos.

3.2.3. Casos de uso

Documentación de los casos de uso de acuerdo a la funcionalidad actual.

3.2.3.1. Iniciar sesión

Permite al rol Anónimo tener acceso al sistema. Iniciar sesión comprende la funcionalidad para verificar la identidad del rol que desea tener acceso al sistema y cambiarla por la del Usuario en caso de que los datos de verificación introducidos sean correctos. Una vez que el usuario está validado, éste tiene acceso al sistema.

3.2.3.2. Cambiar servidor

Le da la capacidad al rol Anónimo de poder cambiar los datos de conexión hacia el servidor destino. De ésta forma, puede conectarse a una versión distinta de la aplicación que le de otra serie de privilegios o funcionalidades.

3.2.3.3. Crear grupo nuevo

Da al rol Usuario la capacidad de crear un grupo nuevo en el sistema. Con ésta funcionalidad, el Usuario dispone de las herramientas necesarias para poder registrar en el sistema un nuevo grupo con una serie de alumnos asociados al mismo.

3.2.3.4. Seleccionar grupo

Permite al Usuario poder seleccionar uno de los grupos que tenga creados anteriormente. Con ésta funcionalidad el Usuario tiene la capacidad de seleccionar un grupo para poder visualizar la información del mismo y activar los controles que le pueden permitir realizar otras funcionalidades que afecten únicamente al grupo que tenga actualmente seleccionado.

3.2.3.5. Gestionar Scripts

Da al rol Usuario la capacidad de modificar los Scripts/Tags que posee el grupo actualmente seleccionado. Ésta funcionalidad es el punto de partida que permite alterar los scripts y los tags que poseen todos los alumnos que están contenidos en el grupo que se haya seleccionado.

3.2.3.6. Gestionar Tags

Permite al Usuario poder crear, modificar y eliminar sus propios Tags incluyendo o eliminando una serie de Scripts para acelerar el proceso de gestión de Scripts en un grupo. Además, también permite que durante la modificación de un Tag éste pueda ser donado a otro usuario del sistema siempre y cuando no existan problemas de colisiones(que el Tag tenga el mismo nombre que el que posee el usuario destino).

3.2.3.7. Cambiar nombre grupo

Da permisos al rol Usuario para poder realizar un cambio en el sistema que permita modificar el nombre de un grupo determinado.

3.2.3.8. Ver historial cambios

Permite mostrar al Usuario una lista que contiene los cambios que ha ido realizando a lo largo del tiempo, desde la primera vez que accedió a la aplicación hasta los cambios más recientes y actuales todos ellos ordenados por fecha de modificación para una mejor comprensión. Además, le permite poder realizar una serie de filtros para poder acceder de una forma más cómoda a la información que se requiera en un momento dado.

3.2.3.9. Eliminar grupo

Da la capacidad al rol Usuario de borrar un grupo del sistema. El Usuario deberá dar el visto bueno y confirmar la ejecución de la acción.

3.2.3.10. Cerrar sesión

Da la opción al rol Usuario de poder salir de forma segura del sistema. Al hacerlo, su rol pasará a ser el de Anónimo.

3.3. Casos de uso del sistema (Parte Servidor)

3.3.1. Diagrama de casos de uso

A continuación se va a mostrar el diagrama completo de casos de uso de la parte servidor de la aplicación:

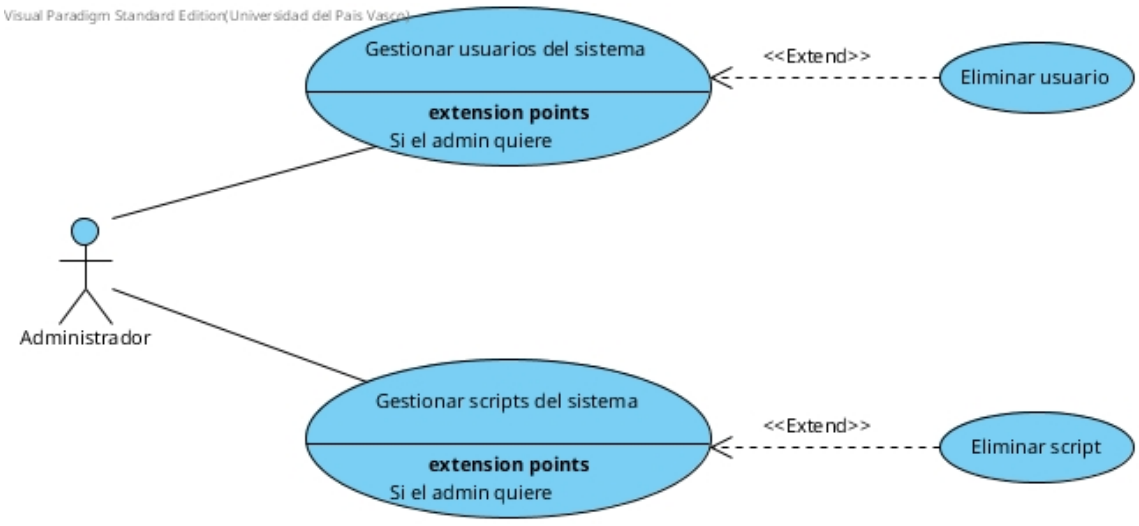


Figura 3.2: Diagrama de casos de uso de la parte servidor

3.3.2. Jerarquía de actores

En la parte servidor sólo existe un único rol:

- **Administrador:** El administrador principal del sistema. Contiene todos los privilegios que le permiten modificar el sistema para incluir/eliminar scripts e incluir/eliminar usuarios en el sistema.

3.3.3. Casos de uso

Documentación de los casos de uso de acuerdo a la funcionalidad actual.

3.3.3.1. Gestionar Usuarios del sistema

Permite al rol Administrador hacer una gestión avanzada sobre los usuarios que tienen derecho a acceder a la aplicación. Ésta funcionalidad permite ver una lista de usuarios registrados en el sistema y permite activar una serie de funcionalidades diversas para poder hacer una manejo coherente de los usuarios que pueden entrar en la aplicación.

Dichas funcionalidades comprenden la creación, modificación y eliminación de los usuarios en el sistema con las repercusiones que ello trae(al eliminar un usuario del sistema debe borrarse todo rastro de sus acciones dentro de él).

3.3.3.2. Gestionar Scripts del sistema

Da al rol Administrador la capacidad de hacer una gestión eficiente sobre los Scripts que contiene el sistema. Con ésta funcionalidad el Administrador puede ver una lista de los Scripts registrados y puede activar una serie de herramientas diversas que le ayudan a manejarlos de una forma simple.

Dichas herramientas comprenden la creación, modificación y eliminación de los Scripts en el sistema con las repercusiones que ello trae(al eliminar un Script éste debe desaparecer de todos los grupos y Tags del sistema).

3.4. Modelo de dominio (Parte Cliente)

A continuación se va a visualizar el modelo de dominio para la parte cliente de la aplicación

3.4.1. Diagrama de modelo de dominio de la parte cliente

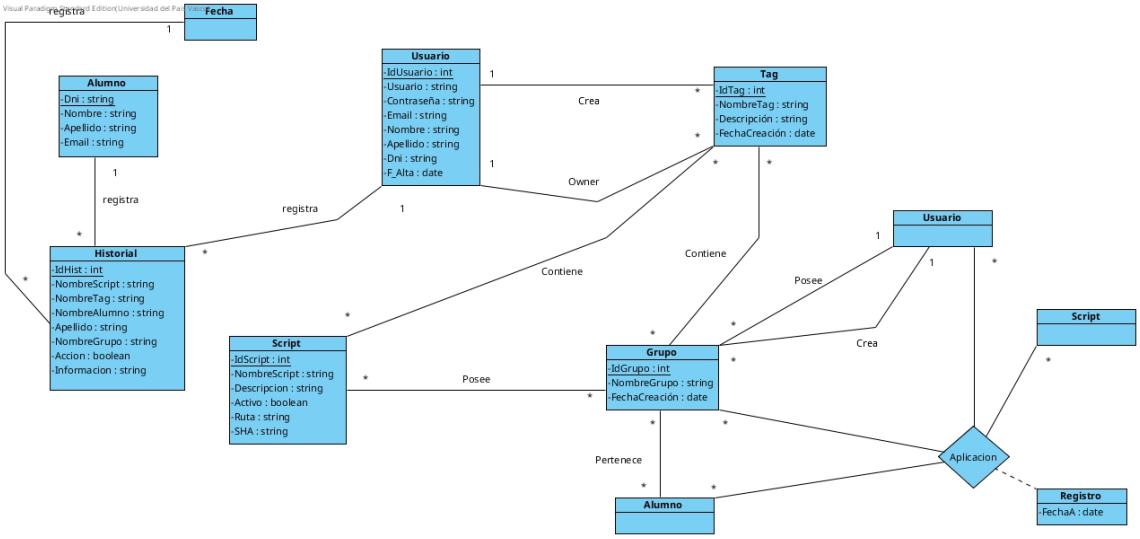


Figura 3.3: Modelo de dominio de la parte cliente.

A continuación se hará una explicación del modelo de dominio. Para ello, éste apartado se va a dividir en dos partes: La primera tratará sobre la explicación de las entidades y la segunda justificará las razones por las cuales se han establecido las relaciones entre dichas entidades.

3.4.1.1. Explicación de las entidades

1. **Alumno:** Representa un alumno en el sistema. Su clave principal es el Dni y tiene unos atributos bastante característicos como el nombre, apellido etc... Es muy importante el registro del Email ya que es el dato necesario para que éste pueda recibir las instrucciones que le permitirán acceder a un servicio en caso de que se le haya aplicado un Scritp/Tag

2. **Usuario:** El usuario es la representación empírica de un profesor. Su clave principal es el identificador de usuario y tiene como atributos sus datos personales, su email y sobre todo, el usuario y contraseña para poder acceder al sistema. El usuario es la entidad que representa básicamente el rol de Usuario del sistema y que permite hacer uso de todas las herramientas disponibles para hacer una gestión eficiente de los los Scripts/Tags a aplicar a los alumnos.

3. **Script:** Ésta entidad guarda las características de los Scripts. Tiene como clave principal el identificador del Script y como atributos tiene un nombre(para ayudar a identificarlo), una descripción(qué es lo que hace) y un avisador que indica si está disponible o no para que pueda usarlo un Usuario en el sistema. El Script representa la utilidad que permite automatizar los procesos del sistema.

4. **Tag:** Un Tag es una visualización de una aglutinación de Scripts. Tiene como clave pricipal el identificador del Tag y como atritubos su nombre(para identificarlo), su descripción(ayuda a saber qué hace) y una fecha de creación. Un Tag representa un conjunto de Scripts que se aplican de forma automática y que facilitan la labor del usuario a la hora de manejar los Scripts en el sistema.

5. **Grupo:** En la entidad de Grupo se guardan los datos relativos a los Grupos del Usuario. La clave principal es el identificador del Grupo y tiene como atributos el nombre del Grupo y la fecha de creación. Un Grupo es un lugar donde se aglutinan los alumnos y permite asignar los Scripts/Tags deseados. Resumiendo, un Grupo es el sitio donde qué alumno tiene asignado qué Script/Tag por qué profesor.

6. **Historial:** Representa el espacio donde se guardan los registros del usuario. Tiene como clave principal el identificador del historial y como atributos los nombres de los Scripts/Tags, alumnos, grupos y usuarios que han intervenido en un momento determinado del sistema. El historial guarda la información relativa a los cambios que se producen dentro de la aplicación, indicando si dichos cambios son agregaciones, eliminaciones o modificaciones. Además, el historial guarda una justificación para ayudar a identificar exactamente qué ha pasado y por qué. Una de las cuestiones más llamativas ha sido que a la hora de diseñar el historial no se ha querido dar la oportunidad de introducir registros haciendo uso de identificadores. Ésto tiene su justificación y es que era necesario que los datos del historial persistieran durante el tiempo, incluso si los elementos que existían(un alumno, un script, un tag o un grupo) ya no estuvieran en el sistema por causas como un borrado por parte del Usuario o el Administrador del sistema. Por ello, se ha pensado en una forma poco común pero efectiva de conseguir que los registros no desaparezcan en caso de borrado y dicha forma, como se puede discernir, es insertar directamente los datos de los nombres de los elementos afectados en vez de introducir los identificadores (que sería lo esperado en otro contexto).

3.4.1.2. Explicación de las relaciones

A continuación se va a exponer las explicaciones entre las relaciones. Para que se puedan entender, se hará uso de la siguiente nomenclatura:

RelaciónA - RelaciónB : Explicación de cardinalidad y razonamiento

- **Fecha - Historial:** La relación entre Fecha e Historial es de 1 a N. Una Fecha estará registrada en uno o varios historiales(día, mes, año) mientras que un historial tendrá una Fecha determinada. Con ésta relación se puede establecer las diferencias entre las entradas y razonar los duplicados, alegando que son de una Fecha distinta.

- **Usuario - Historial:** La relación entre Usuario e Historial es de 1 a N. Un Usuario puede estar registrado en uno o varios Historiales mientras que un Historial sólo puede albergar el registro de un único Usuario. Con ésta relación se establece que cada registro en el Historial tiene un único Usuario y no varios y que un Usuario puede estar en varios Historiales porque el mismo Usuario ha realizado una serie de acciones diferentes sobre distintos Scripts/Tags etc...

- **Usuario - Tag**: La cardinalidad de la relación entre Usuario y Tag es de 1 a N. El razonamiento es debido a que un Usuario puede crear y ser propietario

de uno o más Tags distintos mientras que un Tag sólo puede pertenecer a un único Usuario. De ésta forma, se puede saber qué Tags posee un único Usuario e independizar los accesos a los Tags entre unos y otros.

- **Tag - Script:** La cardinalidad entre las entidades Tag y Script es de N a M. Ésto es debido a que un Tag puede contener uno o varios Scripts y un Script puede ser accedido por uno o varios Tags. Con ésto, podemos hacer que cualquier Usuario del sistema pueda hacerse sus propios Tags agregando los Scripts necesarios con total independencia de lo que hagan el resto de los Usuarios.
- **Script - Grupo:** La relación entre éstas dos entidades es de N a M. La razón es porque un Script puede estar en niguno o varios Grupos distintos(Tanto del mismo Usuario como de otros) y un Grupo puede tener ninguno o varios Scripts aplicados. Ésta relación permite una total libertad de que cualquier Script pueda ser usado en cualquier Grupo y un Grupo pueda tener cualquier Script aplicado.
- **Tag - Grupo:** La cardinalidad de la relación existente entre éstas dos entidades es de N a M. El caso es similiar al descrito anteriormente en la relación entre Script y Grupo. Un Grupo contiene niguno o varios Tags mientras que un Tag está contenido en ninguno o varios Grupos.
- **Usuario - Grupo:** La relación entre Usuario y Grupo es de 1 a N. Ésto es sencillo de explicar dado que un Grupo sólo puede pertenecer a un Usuario y un Usuario puede tener ninguno o varios Grupos distintos. Ésta cardinalidad busca como objetivo saber la identidad del propietario de cada Grupo.
- **Grupo - Usuario - Alumno - Script:** Ésta relación múltiple de N a M tiene el nombre de "Aplicación". Una Aplicación representa la relación que engloba la respuesta a: qué Alumno se le ha aplicado qué Script por qué Usuario en qué Grupo. De ésta forma, queda un registro único(la entidad asociación que contiene la fecha) que indica la existencia de la acción que tomó un Usuario cuando quiso aplicar un Script a uno de sus Alumnos que se encontraba en un Grupo que previamente había creado para tener una organización completa sobre sus acciones.

3.5. Modelo de dominio (Parte Servidor)

A continuación se va a visualizar el modelo de dominio para la parte servidor de la aplicación

3.5.1. Diagrama de modelo de dominio de la parte servidor

Figura 3.4: Modelo de dominio de la parte servidor.

3.5.1.1. Explicación de las entidades y relaciones

A diferencia de la parte cliente, aquí sólo existe una única entidad y que no necesita de dato alguno ya que nunca queda registrada en el sistema. Además, las acciones que toma no son registradas en ningún lado por lo que, en éste caso, no poseemos ninguna relación con otras entidades para tener que guardar datos.

- **Administrador:** Ésta entidad comprende al Administrador del sistema. El Administrador accede desde un panel especial el cual no requiere de ningún tipo de permiso adicional. Todas las acciones que realiza no quedan registradas en el sistema y siempre son directas contra la base de datos.

Capítulo 4

Análisis y diseño

4.1. Introducción

En el capítulo de análisis y diseño se va a responder a las preguntas que engloban el cómo y de qué forma se ha conseguido hallar una solución final eficiente para poder satisfacer el planteamiento inicial.

A continuación, se van a desarrollar dos secciones. Una de ellas va a contener el diagrama de clases de la parte cliente y de la parte servidor de la aplicación mientras que la otra va a contener el diagrama de entidad-relación que a va a mostrar la lógica que contiene la base de datos.

4.2. Diagrama de clases

4.2.1. Parte cliente de la aplicación

El diagrama de clases actual de la aplicación es el que se muestra a continuación:

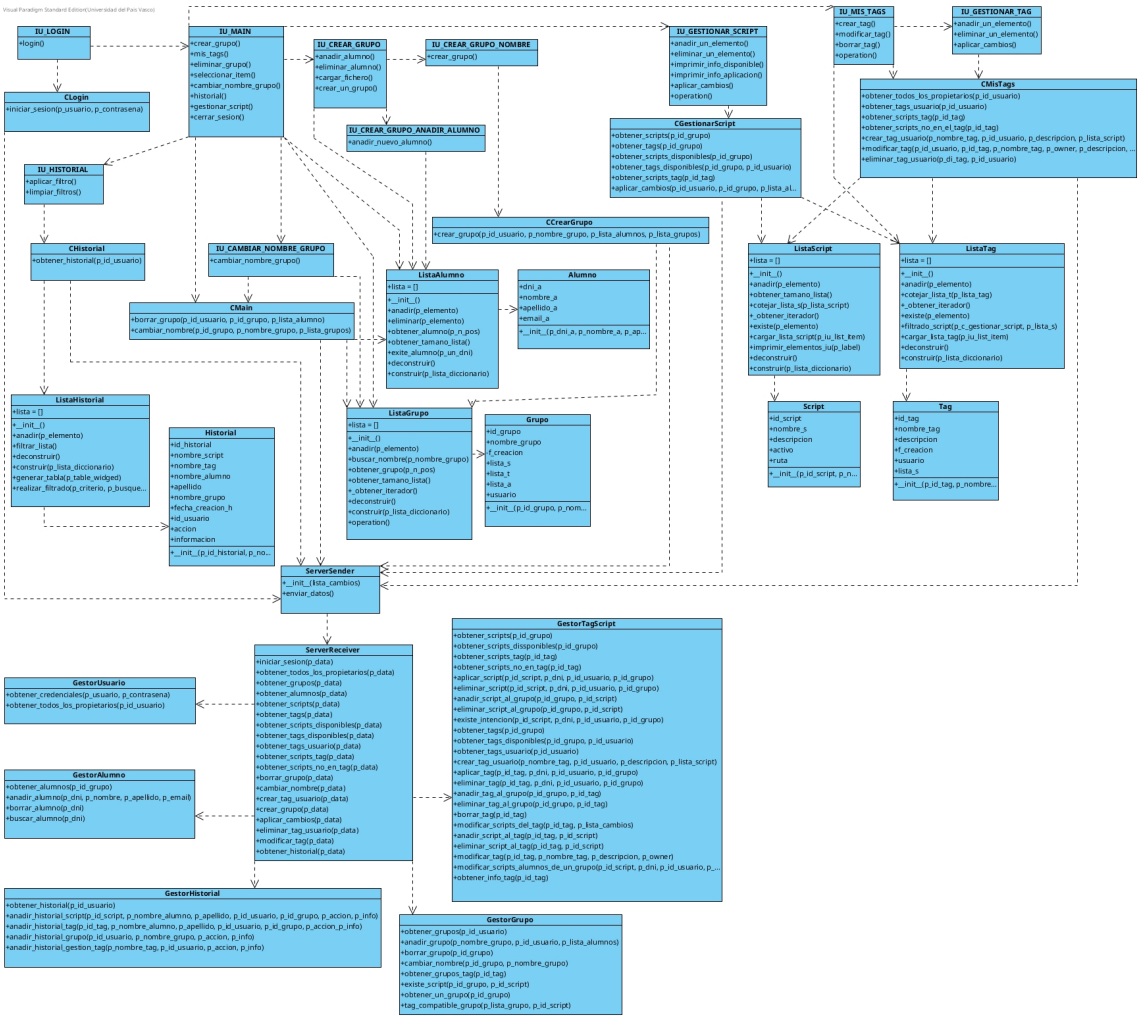


Figura 4.1: Diagrama de clases de la parte cliente.

Como se puede discernir en éste completo diagrama de clases, se ha hecho uso del patrón "modelo-vista-controlador". Hay que recordar que en "Python" todo son objetos y que no existe la encapsulación como tal por lo que los método son públicos y los atributos(donde se hayan usado) también lo son.

El orden de ejecución comienza desde la interfaz de login(IU_LOGIN) y a partir de ésta se salta a la interfaz principal del sistema(IU_MAIN), desde ahí se realiza toda la lógica del sistema con todas las opciones disponibles.

Todas las interfaces(las que comienzan por "IU") tienen asignadas un controlador(que comienzan por "C") que recoge las órdenes con los parámetros necesarios para ejecutar y controlar el código dependiendo de lo que se necesite. Los métodos de las interfaces no necesitan parámetros de entrada porque los propios métodos tienen la capacidad de recoger los datos directamente de la interfaz.

Los controladores terminan desembocando en la clase *ServerSender* la cual se encarga de generar un *Socket* que contendrá las órdenes y los datos necesarios para que el servidor, donde está ubicada la base de datos, sepa qué acciones debe de tomar.

La clase *ServerReceiver* es la encargada de escuchar las peticiones que le envía *ServerSender*. Cuando se envía una petición, la clase *ServerReceiver* procesa dicha

petición y ejecuta, haciendo uso de un diccionario, el o los métodos necesarios.

ServerReceiver gestionará las peticiones haciendo uso de los gestores existentes, los cuales, tendrán los accesos directos a la base de datos. Cuando los gestores le den un resultado, *ServerReceiver* mandará de vuelta aquellos datos que le haya procesado él o los gestores implicados en la petición y se cerrará la conexión.

La razón de realizar una estructura de éstas características es debido a dos razones:

1. *Proporciona un código ordenado.* Guarda un buen sentido con los paradigmas de la programación y ayuda a modularizar cada apartado de tal forma que la estructura sigue una lógica que tiene una clara orientación de cara a que un futuro programador pueda estudiar y cambiar ciertas características.

2. *Es una estructura segura.* La base de datos está completamente aislada de accesos no autorizados y la aplicación espera un tipo de estructura determinada. Ésto es importante de cara a manejar datos sensibles que no se quieran exponer a riesgos innecesarios.

4.2.2. Parte servidor de la aplicación

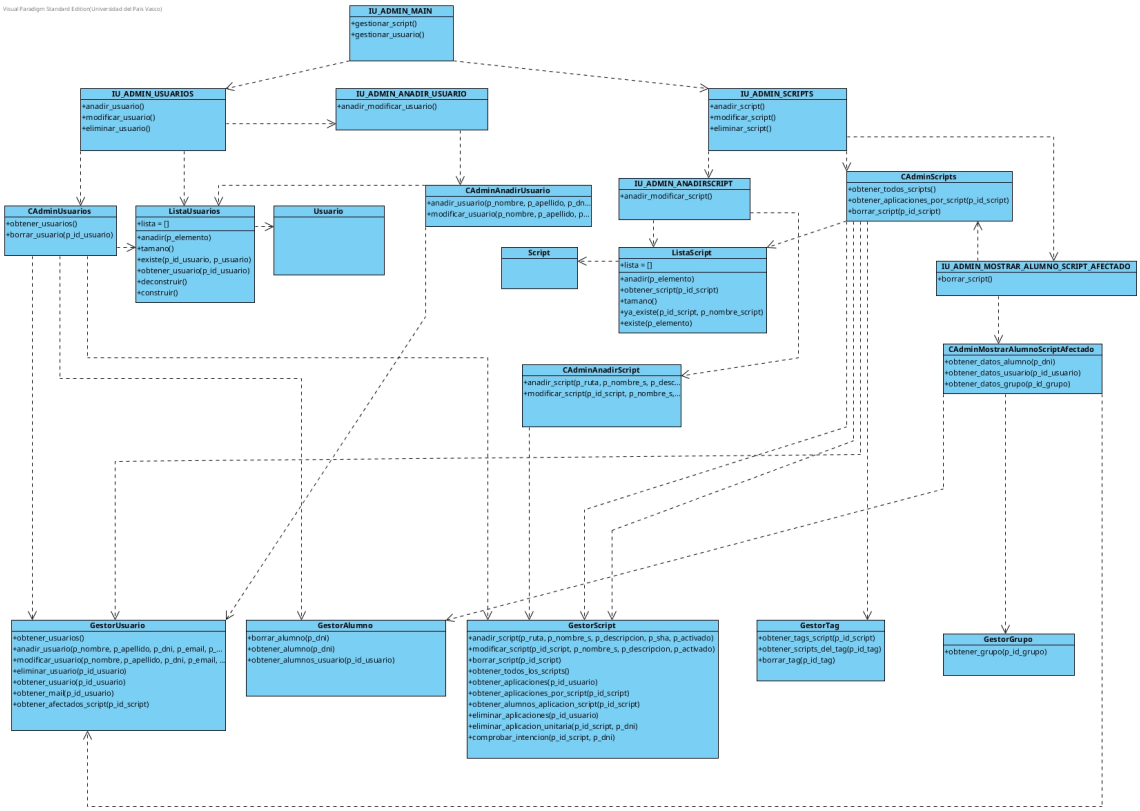


Figura 4.2: Diagrama de clases de la parte servidor.

En ésta ocasión se puede ver que éste diagrama es completamente distinto al anterior aunque se guarda el mismo patrón de diseño(modelo-vista-controlador). Aquí se emplea un sistema más clásico ya que el programa actúa directamente contra la base de datos del servidor y no necesita de permisos adicionales para ser manejado.

La aplicación comienza desde la interfaz principal del panel de administrador (IU_ADMIN_MAIN) y desde ahí da dos posibles opciones: Ir a la pantalla de gestión de usuarios (IU_ADMIN_USUARIOS) o ir a la pantalla de gestión de scripts(IU_ADMIN_SCRIPTS).

En ambas pantallas existen las opciones para añadir, modificar o eliminar elementos de las tablas. Lás únicas diferencias se hayan cuando se intenta eliminar un usuario o un script.

- Si se intenta eliminar un usuario, el sistema elimina todos los grupos con todas las aplicaciones de Scripts/Tags que hubiera. Además vacía todo el historial que tuviera que ver con el usuario borrado.
- Si se intenta eliminar un script el sistema detecta a qué alumnos afecta y muestra (si es necesario) una interfaz adicional de aviso mostrando a qué alumnos les afecta el borrado de un Script. En caso de validarse la operación el Script es borrado y su aplicación en cada alumno también.

4.3. Diagrama de Entidad-Relación

A continuación, se muestra el diagrama de Entidad-Relación que representa las relaciones lógicas existentes en la base de datos utilizada para el proyecto.

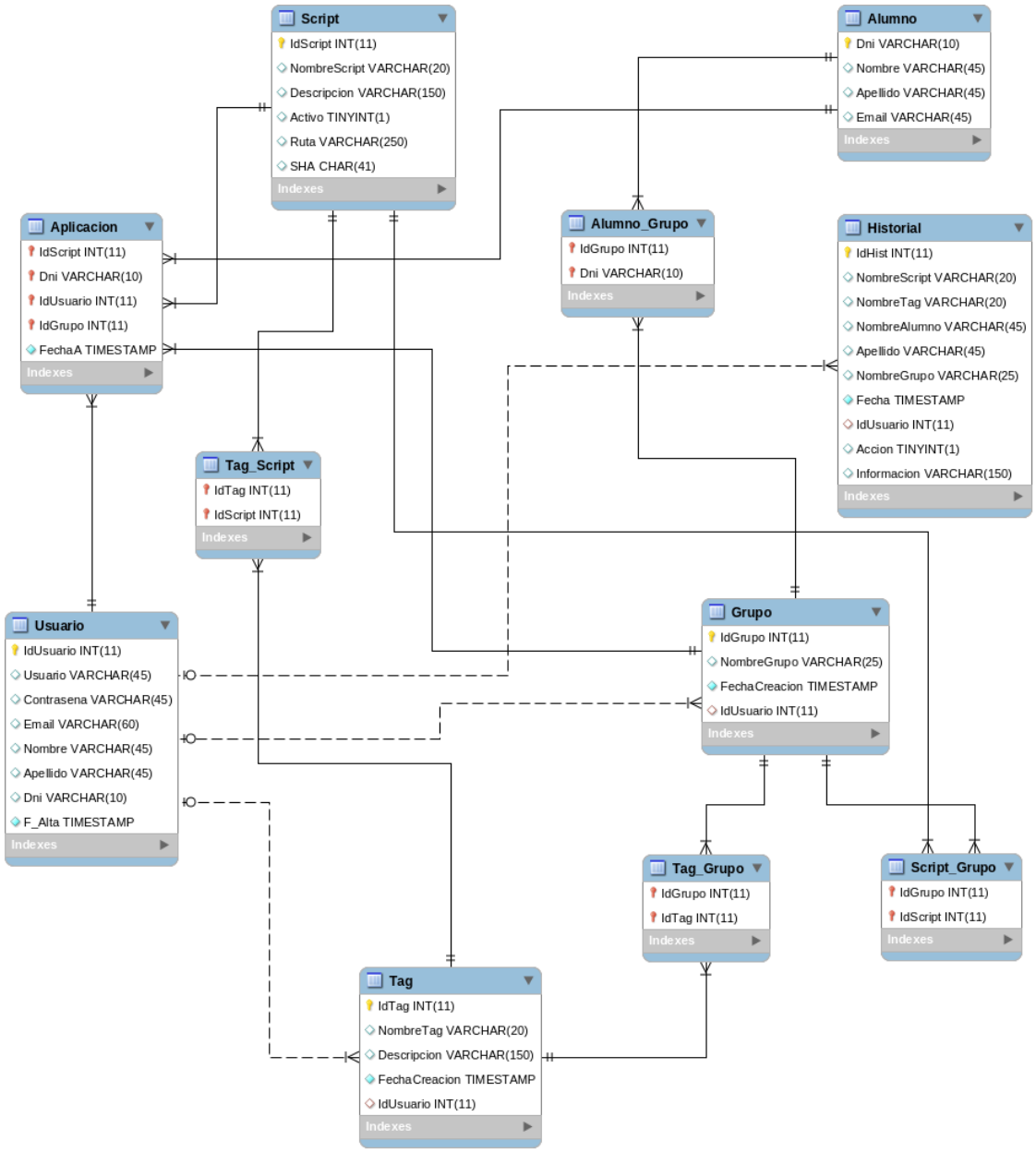


Figura 4.3: Diagrama de Entidad-Relación usado en el proyecto.

Como se puede apreciar, existen algunas entidades que no se han visto en el modelo de dominio. Éstas entidades en su mayoría son tablas intermedias que responden a las necesidades de las relaciones varios a varios. A continuación se va a describir dichas tablas y su necesidad:

- **Alumno-Grupo:** Ésta tabla guarda las relaciones entre los identificadores de los Alumnos y los Grupos. Con ello se puede saber por ejemplo, qué alumno o alumnos están contenidos en qué Grupos.

- **Script-Grupo:** Con ésta tabla se puede saber qué relación existe entre los Scripts y un Grupo determinado. Ayuda principalmente a saber cuándo un mismo Script ha sido aplicado en diferentes Grupos.

- **Tag-Grupo:** Ésta situación es similar a la descrita anteriormente con Script. La necesidad de ésta tabla viene dada por los mismos factores que con los Scripts.
- **Tag-Script:** En este lugar se almacenan las relaciones entre los Scripts y los Tags. Gracias a ésta tabla es posible saber por ejemplo que Scripts contiene un Tag.
- **Aplicación:** En la tabla Aplicación se guarda la relación múltiple entre Usuario, Grupo, Alumno, Script y la entidad asociación Registro(que contiene una Fecha). La necesidad de tener ésta tabla radica en poder guardar las aplicaciones de los Scripts a cada Alumno separándolos de los distintos Usuarios y Grupos para mantener un orden y una coherencia que evite que las decisiones tomadas por otros Usuarios generen conflictos con las que tome el Usuario actual.

Capítulo 5

Desarrollo

5.1. Introducción

En éste apartado se va a explicar qué se ha realizado en éste trabajo y cómo se ha querido realizar. Se darán una serie de justificaciones y se comentarán los problemas que se hayan encontrado y cómo se han conseguido resolver de forma exitosa.

5.2. Inicio: Una historia que contar

La presentación de éste proyecto fue un hecho fascinante. Y no fue debido a que el proyecto fuera a ser algo que cambiase el mundo, más bien fue la capacidad de personalización que ofrecía y la libertad que daba de hacerlo como a uno le gustase.

Éste proyecto sólo tenia un requisito fundamental importante: Tener un conocimiento determinado sobre sistemas operativos basados en "GNU/Linux", el resto, dependía enteramente de la imaginación y las ganas que le quisiera poner el programador.

Fácilmente se podría haber hecho una aplicación basada en código Java(lenguaje más que visto en la carrera) o haber hecho una aplicación web(que es la moda de la época de éste proyecto). Pero, una oportunidad así requería un reto mucho mayor.

Las decisiones que se tuvieron que tomar principalmente fueron:

- ¿Qué tipo de aplicación debería llegar a ser?
- ¿Qué lenguaje de programación se debería usar?
- ¿Qué cosas me gustaría aprender realmente con éste proyecto?
- ¿El esfuerzo merecerá la pena realmente?

Y con esas cuestiones, empezó el reto de hacer todo éste proyecto.

5.2.0.1. ¿Qué tipo de aplicación debería llegar a ser?

Lo primero de todo, y es lo que a cualquiera no le deja pegar ojo, es qué tipo de aplicación se busca realizar. Existen muchas formas de solventar éste problema pero en éste caso se decidió realizar una aplicación clásica de cliente-servidor, donde el servidor fuese un programa en ejecución constante que aceptara una serie de

conexiones de varios clientes. El cliente, por su lado, se instalaría una aplicación de escritorio y tendría la capacidad de conectarse al servidor para enviar y recibir la información necesaria. Además, existiría una panel de control de administración para que una persona experimentada pudiera hacer una serie de ajustes adicionales al sistema.

La construcción es clásica en comparación de los servicios web actuales y hace uso de sockets para hacer una comunicación rápida entre el cliente y el servidor. Es cierto que hoy en día una aplicación de éstas características tiene una mejor implantación siendo un servicio web pero se ha preferido hacer así debido a una razón.

La razón determinante de hacer un programa de éstas características vino dada por motivos académicos. El objetivo era crear un programa que emvolviese el uso de diferentes tipos de tecnologías que si bien quizás hoy en día no son altamente populares, no dejan de ser interesantes para aprender cómo hacer desde cero una aplicación real de escritorio pudiendo en un futuro, abrir puertas profesionales o académicas en cuestiones de implantación de aplicaciones personalizadas para ciertos fines determinados.

Si ponemos en contexto qué ha sido necesario aprender para hacer ésto nos encontramos con:

1. Aprender un lenguaje de programación completamente nuevo.
2. Aprender un lenguaje de modelado de interfaces completamente nuevo.
3. Aprender uso de scripting avanzado para sistemas operativos "GNU/Linux".
4. Tomar un conocimiento de los funcionamientos de los diferentes sistemas operativos.
5. Aprender a integrar los distintos lenguajes de programación consiguiendo conectar las características de cada uno de ellos.
6. Saber cómo relacionar un lenguaje de programación, con uno de modelado de scripts, con uno de scripting etc.. Básicamente, aprender a que se "entiendan" entre ellos.
7. Saber cómo hacer una aplicación que sea portable. Ésto es, que sea multiplataforma y funcione en más de un sistema operativo.
8. Obtener conocimientos de usabilidad para que la aplicación final sea lo más simple posible.

Todas éstas razones son sin duda una buena excusa académica para poder decantarse por hacer un proyecto orientado a una aplicación clásica cliente-servidor.

5.2.0.2. ¿Qué lenguaje de programación se debería usar?

Decidido qué es lo que se debía de hacer, lo siguiente que se tuvo que pensar fue qué tipo de lenguaje de programación se debería usar. En el caso de éste proyecto, la respuesta fue bien clara. "Python". [33]

En el mundo de "GNU/Linux", "Python" junto con "C" y "C++" es uno de los lenguajes más usados y respetados. Es un lenguaje más completo que "Java"(lo que

más se ha visto en la universidad), es más flexible, permite herencia múltiple(no existe problema de diamante), no tiene encapsulado(el programador decide todo) y además posee una amplia documentación y comunidad que ayuda bastante a poderlo aprender y utilizar como un profesional.

"Python" es un lenguaje de programación interpretado y multiparadigma. Con ésto tenemos un lenguaje que no necesita estar haciendo compilaciones y que da una posibilidad de elegir si uno desea hacer uso de los paradigmas de la programación orientada a objetos, programación imperativa o programación funcional.

Resumiendo, "Python" es un lenguaje que da libertad de moldear una aplicación a la manera que uno desee y esto tiene mucha relación con la filosofía de los sistemas operativos de "GNU/Linux".

5.2.0.3. ¿Qué cosas me gustaría aprender realmente con éste proyecto?

Es una pregunta sencilla pero a la vez compleja. Realmente hay que tener muy claro qué se quiere aprender cuando el deseo es hacer un proyecto que, en un principio lógico, va a contracorriente de lo que se considera lo "normal". Por ello es necesario tener muy claro qué se quiere aprender.

En éste caso, la respuesta es clara, aprender a usar nuevos lenguajes de programación y aprender a integrarlos entre ellos. De ésta forma, se puede abrir la mente a cosas nuevas y aprender nuevas metodologías que puedan llegar a ser, en un futuro indeterminado, útiles para abordar nuevos proyectos.

5.2.0.4. ¿El esfuerzo merecerá la pena realmente?

Por último llega una de las más importantes cuestiones. ¿Vale la pena el esfuerzo? Cierto es que es complejo saber a ciencia cierta si en un futuro se presenta la ocasión de hacer un programa de éstas características, máxime cuando hoy en día proliferan las "apps" y todo lo relacionado con el mundo de la movilidad, los smartphones, las tablets y los *Icacharros* existentes y por existir.

Pero abordándolo desde una perspectiva más funcional, al final la respuesta es un sí rotundo, básicamente porque es posible que se presenten ocasiones similares o que los tiempos cambien y se vuelvan a hacer prácticas como las que se han realizado en éste proyecto.

Por ello al final, la satisfacción personal de conseguir un hito haciendo una aplicación de éstas características y el hecho de haber aprendido distintas tecnologías nuevas y útiles, hace que al final merezca la pena y uno se sienta gratificado por haber sido capaz de superar éste hito.

5.3. Primeros pasos: La casa no se construye por el tejado.

Teniendo claro qué se tenia en mente y sabiendo más o menos qué herramientas se debían utilizar, empezó la materialización del proyecto, esto es, cómo se iba a mostrar la aplicación al mundo.

Para ésta hazaña era necesario aprender un lenguaje de programación que permitiera pintar interfaces. Además, era muy importante hacer una selección acertada debido a que uno de los requisitos del proyecto es que éste fuese multiplataforma.

En el mundo de "GNU/Linux" existen varios tipos de librerías gráficas para poder escoger. Las más populares, son "GTK", "QT" y "wxWidgets".

"Python" no tiene problema a la hora de entenderse con éstas librerías gráficas por lo que sólo restaba saber cuál de ellas se quería escoger.

En un principio la idea era hacer uso de "GTK" y la librería "PyGtk", que hace que "Python" y "GTK" se entiendan perfectamente. Ésta combinación se usa bastante dentro de las aplicaciones en "GNU/Linux", un gran número de ellas hacen uso de ésta dualidad y el resultado final es formidable. Pero al final la idea se terminó descartando por una serie de motivos.

1. *Que no existía un amplio soporte.* Aprender algo nuevo y que el soporte no sea el mejor, no era una gran ayuda para empezar a integrar un tipo de interfaz con el lenguaje de programación deseado. Si sumamos encima la falta de experiencia con ambos lenguajes, el resultado no terminaba siendo nada prometedor.

2. *Que no era muy portable para otros sistemas operativos.* "GTK" no tiene tantas compatibilidades con otros sistemas operativos a la hora de querer portar la aplicación. Es cierto que para "Windows", por ejemplo, sí que tiene una buena integración, pero si se pensará en ampliar más adelante el proyecto para otras plataformas, "GTK" no terminaría siendo la mejor elección posible.

3. *Las herramientas de diseño no convencen.* Por alguna razón, las herramientas para diseñar las interfaces no resultaban a nivel personal de gran agrado. Debía existir una mejor manera de hacer las cosas o al menos algo para personas de nivel bajo(o nulo en el caso personal) en el diseño de interfaces.

4. *La evolución de las librerías de "GTK" parece estancada.* Las librerías de "GTK" no parecen haber avanzado de forma espectacular durante el tiempo, mientras que otras opciones han mostrado grandes mejoras en cada versión que han ido lanzando.

Por éstas razones, se desechó la idea de hacer uso de "GTK" y los ojos se tornaron hacia "QT", la librería gráfica estrella en "Plasma WorkSpace"(anteriormente llamado "KDE").

Las interfaces basadas en "QT" son bastante mejores. "QT" consume bastante pocos recursos, tiene una amplia compatibilidad con diferentes sistemas operativos, tiene integración nativa con sistemas como "Windows", las herramientas para desarrollarlo son muy completas y existe mucha documentación que explica perfectamente cómo se debe desarrollar cada tipo de interfaz.

La documentación que ofrece "QT" en su página oficial, contiene bastantes ejemplos y primeros pasos para aprender el uso y manejo correcto del lenguaje. [15]

El problema que surgió a la hora de elegir "QT", era que "QT" está pensado y diseñado para ser usado con "C++." De hecho, "QT" tiene su propio editor que permite diseñar y programar las interfaces como si fuera un "Visual Basic" pero dicho editor sólo admite lenguaje "C++".

Ésto chocaba directamente con la intención de hacer una combinación entre "Python + QT". Afortunadamente, el problema fue fácilmente solventado gracias a "PyQt". "PyQt", de la misma forma que "PyGtk", permite un entendimiento entre

"Python" y "QT" y gracias a ello se pudo solventar la barrera que suponía conectar "Python" con una interfaz basada en "Qt".

Para que el funcionamiento fuese efectivo, se encontró una manera sencilla de hacer funcionar todo el conjunto. El procedimiento se basaba en:

1. Se diseñaba la interfaz completa haciendo uso de "QtDesigner"(Parte de las herramientas oficiales del "SDK" de "QT").

2. Se usaba "PyQt" para convertir la interfaz en un fichero "Python" que fuera ejecutable.

3. Se programaba la interfaz para hacer que cada elemento funcionase de manera correcta.

Todo en papel era bastante bonito, pero la realidad era muy distinta y bastante complicada. Surgieron varios problemas.

Es cierto que programar la interfaz era sencillo. Tan simple como arrastrar y colocar los botones, pero no se disponía de los conocimientos necesarios para que la interfaz tuviera un aspecto "usable". Las interfaces costaban bastante hacerlas y no terminaban de quedar bien(sobre todo si se reescalaba a determinados tamaños).

El problema, se hallaba a que no se disponía un entendimiento del uso de los denominados "layout" o lo que es lo mismo, los contenedores que ordenan los elementos de una determinada forma. Buscando en la documentación de "QT", se encontró los diferentes tipos de contenedor que existían para organizar los elementos de una forma u otra. Gracias a los documentos y a algún que otro vídeo explicativo, se pudo aprender a usar éstos "layouts" y a conseguir realizar unas interfaces dignas y bien organizadas.

Otro problema que surgió fue la conversión de las interfaces diseñadas a código "Python". Se supone que al usar "PyQt" todo va sobre ruedas, pues no, es cierto que el código era funcional pero no se explicaba que había que hacer ciertas "modificaciones" para que la interfaz fuese correctamente instanciada y ejecutada. Ésta parte resulto bastante complicada porque no existían gran cantidad de ejemplos útiles, o al menos, ejemplos que hiciesen entendible el código para que en un futuro fuese revisado. Al final, se terminó encontrando una solución factible y funcional y ya de paso, un buen puñado de ejemplos para aprender. [30]

El siguiente gran problema, se halló a la hora de querer conectar los botones con los métodos deseados. Sobre el papel, el procedimiento era simple, se llamaba al objeto que contenía un botón de la interfaz y se le mostraba el "camino" hacia el método objetivo para que se produjeran los eventos necesarios. El problema era que el funcionamiento que enseñaban los manuales eran para "C++" y por lo tanto inválidos para "Python".

Tras mucha indagación se descubrió que había que añadir en todas las conexiones un método llamado *"connect()"* que curiosamente el editor no lo detectaba como existente, pues bien, existía y precisamente era el requisito indispensable para que se pudiera conectar cualquier elemento de una interfaz con un método determinado. Tela marinera.

Tras conseguir que las interfaces fueran ejecutables y que éstas al menos funcionasen se empezó a realizar pequeños proyectos para conseguir medir la funcionalidad. Se hicieron proyectos como una calculadora, un convertidor de unidades y una interfaz que permitía obtener códigos Red Green Blue(RGB) para colores personalizados.

5.4. Construyendo el proyecto: Mira por dónde pisas

Superado la barrera de las interfaces llego el momento de empezar a construir el proyecto. La idea inicial fue realizar una serie de prototipos de interfaces y conseguir que unas personas hicieran una evaluación de su usabilidad. Para ello, se tomó la decisión de hacerles pasar un test de usabilidad para que puntuasen con los valores que estimasen oportunos. Esto ayudaba mucho a refinar la experiencia de usuario.

Tras varias iteraciones y evaluaciones se consiguió consensuar una idea aproximada de cómo debían de ser las interfaces a realizar. Ésto ayudo a decidir cómo se debía de diseñar el proyecto final.

El diseño de las interfaces fue bastante complicado y de ello se pudo sacar una cosa muy clara: Hacer que una interfaz sea simple y útil para el usuario final requiere un gran esfuerzo por detrás y muchas horas para deliberar qué es posible mejorar o dónde colocar los elementos de forma estratégica para que quien vaya a usarlo sea capaz de desenvolverse sin problemas.

Durante el diseño del proyecto, se estuvo deliberando el hacer uso de una base de datos distinta de lo habitual, una opción destacada fue "PostgreSQL", pero al final y a petición del profesor director del proyecto, se usó "MySQL" ya que es más popular y funciona perfectamente.

Uno de los grandes problemas que han surgido de forma continuada en el proyecto, ha sido la falta de experiencia en el lenguaje de programación "Python".

La experiencia en "Java", hacia que se intentara solventar cualquier situación haciendo uso de la lógica de "Java", cuando "Python" no es para nada parecido a "Java" y tiene sus propias maneras de hacer las cosas. De hecho, todo ésto tiene un nombre y es el llamado *Pythonic Way*. "Python" tiene grandes diferencias respecto a "Java" y dentro de esas diferencias hay principios que en "Java" son sinónimo de malas prácticas mientras que en "Python" es sinónimo de eficiencia y buen código.

Se podrían enumerar muchos casos diferentes pero quizás uno destacable es el recorrido de las listas. Mientras que en lenguajes como "Java" se hace uso de la clase *Iterator*, en "Python" no es necesario nada de eso, ya que un *For* en cada iteración obtiene un elemento de la lista, en vez de su posición y con ello, se puede hacer lo que sea necesario. Puestos a ponderar más ésta cuestión se podría decir que los *Iterator* de "Python" producen excepciones cuando no son capaces de encontrar un nuevo elemento de una lista mientras que en lenguajes como "Java" tienen un método que es capaz de saber si existe o no un siguiente elemento(*hasnext*). En resumen, el funcionamiento no es igual, por lo tanto, la lógica tampoco lo es.

En el diseño del proyecto, se pensó que la mejor manera de hacer el programa usable fuese realizando una aplicación que se conectara a un servidor externo que tuviera una base de datos preparada. Para ello, se pensó en los "Sockets" en "Python" que trajo consigo un auténtico quebradero de cabeza.

La información y tutoriales para el uso de sockets en "Python" es buena y hay varios ejemplos, de hecho el propio lenguaje posee clases que ayudan a configurar un servidor de manera muy sencilla con múltiples configuraciones y gestión automática de hilos para procesas varias peticiones [14]. Pero, el problema serio aparecía cuando se deseaba aplicar una capa de encriptación haciendo uso de "SSL".

Para usar sockets con "SSL" era necesario tener certificados instalado tanto en la parte local(llamada parte Cliente en la aplicación) como en la parte servidor. El

problema, era que prácticamente no existían ejemplos para implementar la encriptación "SSL" y muchos de ellos no funcionaban como debían. Al final, después de mucho indagar se recopilaron varias soluciones de diversos lugares en Internet y se consiguió hacer una clase de envío y de recepción 100 % funcionales. [24]

Una vez que los "Sockets" eran funcionales se presentaba otro reto. Cómo debían viajar los datos entre Cliente y Servidor. Para éste menester existían 3 formas distintas de serializar los datos y enviarlos mediante los "Sockets":

- **Utilizar JSON.** Si duda una de las opciones más interesantes y conocidas. El problema era que no era muy rápida a la hora de serializar/deserializar los datos, pero es lo que actualmente más se utiliza y por lo tanto tiene mucho soporte por detrás.

- **Utilizar Pickle.** Sin duda es una alternativa muy buena y bastante rápida. No tan usado hoy en día pero bastante funcional y simple.

- **Utilizar cPickle.** Básicamente ésta opción es una implementación en "C" de Pickle lo que se traduce en que hace las mismas cuestiones pero a una velocidad de serialización/deserialización de datos mucho mayor.

Al final se decantó por JSON. El problema de Pickle y cPickle era la **seguridad**. Aun siendo opciones muy interesantes y muy superiores en términos de velocidad, ambas tenían un agujero de seguridad que podría comprometer los datos de la aplicación. Sabiendo que en ésta aplicación se haría uso de datos sensibles como el nombre, apellido Dni etc. no era de recibo usar un modo de transmisión de datos que no fuese 100 % seguro

Decantarse por JSON fue una decisión interesante, se podían enviar datos al servidor y recibir una respuesta, los tiempos eran optimistas y además, en Internet había infinidad de ejemplos para implementar de forma correcta un "Socket" con JSON. Todo un acierto, si, sobe todo cuando uno descubre que cuando JSON serializa/deserializa los datos, éstos se convierten en Unicode lo cual agrega al comienzo de todas las cadenas de texto el siguiente elemento **u'**.

Realmente era algo incómodo ya que no se podían manejar los textos de forma adecuada. Al servidor no llegaba la misma información que se enviada desde el cliente por lo que se convertía en un serio impedimento a la hora de querer trabajar con la estructura que se quería plantear.

Muchas horas invertidas en indagar, al final, se encontró en una página web una forma de hacer que ésto funcionase [2]. Y fue sencilla, básicamente era reimplementar una clase que hiciera una serialización/deserialización personalizada para conventir los caracteres Unicode a caracteres normales, corrientes y entendibles para el sistema. La clase que se encargaba de realizar ésta "magia" fue bautizada por el nombre de *Decoder*.

5.5. Dando forma al proyecto: Ni contigo, ni sin tí

Sabiendo como hacer interfaces, teniendo una clara idea de cómo debían de ser, y sobre todo, sabiendo que la estructura pensada para conseguir que un cliente y un servidor se entendieran estaba ya definida; sólo faltaba saber qué datos se iban

a manejar y cómo se deberían querer manipular. Llegó la divertida parte de diseñar las primeras clases.

Se manejaron varias formas de hacer las cosas. Por un lado, existía la forma de hacer que la aplicación del cliente no guardase prácticamente datos, para así obtener siempre datos frescos del servidor y actualizados. Otra forma, era cargar todos los datos en local y luego, una vez que se producían una serie de cambios que se consideraran más que suficientes, se conectaría al servidor para actualizar dichos cambios.

Pero ni una idea ni la otra fueron convincentes. Por un lado, no guardar ningún dato en local y llamar continuamente al servidor, hacía que la aplicación fuese demasiado dependiente del servidor y a veces ocurre que las conexiones de Internet no son los suficientemente estables, por lo que podría terminar siendo un lastre. Por el otro lado, guardar toda la información en local tenia un riesgo, dicho riesgo era que cuando un usuario fuese a realizar una acción determinada, los datos hubieran cambiado, un ejemplo simple sería el siguiente:

- Imaginemos que tenemos un Tag con dos Scripts asociados, por algún motivo, el administrador de la aplicación decide borrar un Script del sistema, el usuario aplicará el Tag sin tener conocimiento alguno de que uno de sus Scripts ya no existe. Ésto crea una situación poco deseable.

Por todo ello, se decidió usar una estructura mixta. Parte de la información se guarda en el servidor, pero otra parte se va refrescando en determinadas llamadas clave. De ésta forma si el servidor sufre algún cambio, el usuario de manera rápida podrá visualizar dicho cambio y deliberar si las acciones que desea tomar merecen o no la pena.

La integración de “Python” con “MySQL” fue tediosa. Y de hecho, aparecieron bastantes problemas. Teniendo la costumbre de usar una conexión única en “Java”, en “Python”, aparece la interesante situación de descubrir que se tiene un abanico de varias librerías distintas para poderse conectar a “MySQL”. De hecho podríamos enumerar algunas de ellas:

1. MySQLdb
2. Oursql
3. MySQL Connector
4. PyMySQL
5. Pypy + PyMySQL
6. .....

El primer problema con ésto fue obvio, ¿qué librería se debía elegir? Todas parecían buenas opciones pero cada una de ellas tenían cosas buenas y cosas malas. Por ello, hubo que buscar algo de información sobre su rendimiento y al final un simple artículo de un blog arrojó un poco de luz sobre éste dilema [23].

En el artículo se hacían una serie de pruebas para demostrar el rendimiento en cada driver. Las pruebas son sencillas y los resultados dan a la vista que

MySQLdb(propietario de "MySQL") era el driver que mejor rendimiento ofrecía al sistema.

Por ese rendimiento y porque existia bastante información sobre el uso y manejo de dicho driver [9], se terminó eligiendo MySQLdb para poder conseguir conectar "Python" con "MySQL" de manera eficaz.

5.6. Creando el proyecto: Be Python my friend

Tras conseguir "Sockets" funcionales y una conexión a la base de datos estable ya sólo faltaba hacer el proyecto en sí, lo cual fue más o menos sobre ruedas contando de que se estaba haciendo uso de una metodología en cascada por lo que muchas de las cosas que se implementaban ya estaban pensadas y escritas en papel.

El problema apareció cuando se intentó importar ficheros desde otros paquetes. No funcionaba nada, extraño, raro, la pregunta era ¿qué pasaba? Bien, básicamente lo que ocurría era una cosa simple pero de aplastante lógica: Falta de experiencia con "Python".

"Python" a diferencia de otros lenguajes tiene una forma peculiar de crear paquetes con clases dentro. Para crear un paquete, hay que poner al lado del directorio que tiene los ficheros a querer introducir, una clase vacía con el nombre de *init.py*. De ésta forma, el intérprete de "Python" sabe que el contenido de la carpeta o carpetas forman un paquete que se puede importar de forma externa a otro fichero "Python" ubicado en una dirección distinta del sistema.

Son cuestiones que surgen debido a la inexperiencia con el lenguaje, pero no sólo ésto fue un problema porque cada vez que se avanzaba y se intentaban hacer ciertas cosas, surgían problemas continuados. A continuación se listarán unos pocos ejemplos que parecen cosas básicas pero que luego se convierten en problemas:

- **No existe el Swich/Case:** En "Python" no existe ésta estructura típica en los lenguajes de programación más comunes. Por su parte, para cubrir ésta carencia, es necesario hacer uso de diccionarios para ejecutar métodos cuando se detecta una palabra clave.

- **No hay getters ni setters:** Una cosa muy común en "Java" y algunos otros lenguajes es el uso de métodos llamados getters y setters que permiten obtener o fijar los atributos de una clase mediante un encapsulado seguro que mantiene la integridad de los datos. En "Python" no existe eso, se puede hacer uso de unos "decoradores"(Un método dentro de otro método) llamados *properties* para conseguir un proceso similar.

- **En general no existe la encapsulación:** No hay métodos privados, ni protegidos ni nada, la integridad de un programa está definida sólo por el programador y éste debe tener la batuta para elegir qué clases serán privadas o no. Cierto es que existe una "trampa" para indicar que una clase es privada haciendo uso de la barra baja antes del nombre de un método para indicar a otros programadores que ese método tiene intenciones privadas.

- **Usar "break" no es un delito:** En la carrera se nos pide encarecidamente que el uso del "break" debe ser nulo, pues bien, en "Python" es el pan de cada día y además es parte de las buenas prácticas del *Pythonic Way.* 1.1.5

Podríamos enumerar muchas cosas, pero todo ésto fueron problemas que surgieron poco a poco mientras se intentaba realizar el proyecto. El problema no sólo era la inexperiencia con el lenguaje, el problema también era que por culpa de no saber ciertas cosas los diseños pensados en un comienzo no eran válidos y se tenía que volver a rediseñar y replanificar varias partes por lo que los tiempos pensados para hacer cada apartado del proyecto eran imposibles de cumplir.

5.7. Integrando todo y acabando: No existe un final, sólo un principio.

Una vez programado todo el núcleo del proyecto e implementadas las funcionalidades, sólo quedó construir las interfaces e integrarlas con el código en "Python" que se había programado.

Integrar las interfaces y conectarlas fue bastante sencillo. Usando la metodología citada anteriormente 3 se programó una a una cada interfaz y se conectó a los controladores adecuados para hacer posible las funcionalidades. Pero no todo era oro y obviamente hubieron varios problemas. El más serio fue la implantación de los tipos de ventanas en "QT".

"QT" tiene distintos tipos de ventanas. Cada una de ellas tienen unas funcionalidades diversas y descienden de una superclase determinada. Buscando en la documentación oficial se halló la solución al problema y se aprendió algo nuevo. ¡Existen varios tipos de ventanas!

Nos encontramos por lo tanto con los siguientes tipos de ventanas:

1. **QWidget:** Es la clase más básica. Si no tiene padre alguno, se convierte en una ventana principal. Se usa generalmente para hacer ventanas simples o para ser integrado en un tipo de ventana más complejo.

2. **QMainWindow:** Es un tipo especial de QWidget diseñado parar ser una ventana principal en un sistema. A diferencia de un QWidget, éste tiene una barra de estado, una barra de menús, una barra de herramientas y la posibilidad de añadirle widgets. Además tiene la capacidad de contener QWidgets distintos en su interior.

3. **QDialog:** Es un tipo especial de QWidget diseñado para ser una ventana secundaria que muestre una determinada información simple. Suele usarse para hacer aparecer diálogos comunes como la confirmación de alguna acción o el error en algún proceso. [18]

Sabido ésto, había que modificar cada interfaz para que descendieran de diferentes clases y así determinar qué tipo de ventana era interesante programar en cada apartado del proyecto. Ésto trajo consigo un cambio en el diseño de las interfaces haciendo que éstas mejorasen sensiblemente.

Además de aprender algo tan importante, surgieron también otros problemas que fueron bastante problemáticos. Los problemas se centraban principalmente en la falta de experiencia en la programación con librerías gráficas en "QT".

Muchos de los elementos que había en las interfaces tenían sus propias metodologías y había que hacer bastantes ajustes para que hiciesen lo que se esperaba

de ellos. Por ejemplo, las tablas de información eran editables y permitían selecciones múltiples, o las listas de Scripts/Tags sólo permitían introducir un tipo de dato por lo que a la hora de querer obtener identificadores únicos era un problema. A continuación voy a dejar una pequeña lista de problemas y soluciones:

1. *Las tablas son editables y permiten selección múltiple.* Éste problema fue sencillo de solventar, bastaba con introducir cuatro comandos de propiedades que forzaran a un usuario a realizar una selección de una sola fila y no pudiera editar elementos.

2. *La listas no permiten introducir identificadores.* Para solventar ésto, se usaron las propiedades de "QT" llamadas *User Roles* las cuales permitían insertar información extra a los elementos de una lista para cuestiones internas del programa.

3. *Las ventanas no salen centradas:* Las ventanas aparecían donde querían y ésto era un incordio. La solución fue insertar un comando que forzara a las ventanas a ponerse en la posición 0,0 de la resolución de la pantalla actual.

4. *Cada elemento de una interfaz tiene su forma de conexión.* Un problema frecuente era que cada elemento tenía su forma de conectarse a una método, los botones hacían uso del *clicked*, las barras de acciones hacían uso del *triggered*, los combo box hacían uso del *indexChanged* etc. Por ello, en más de una ocasión ha sido necesario consultar en Internet cuáles eran los métodos que permitían capturar la acción parar hacer posible la conexión entre el elemento y el método.

Diseñadas las interfaces y conectadas de forma correcta hacía que el proyecto ya fuese funcional y descartando los obvios fallos que fueron apareciendo posteriormente(errores de programación), el proyecto se pudo completar con éxito.

5.8. Portando y finalizando: Sweet /home

Una vez implementado todo el código faltaba una última acción: Portar el programa a distintas plataformas y hacer el empaquetado de los binarios. Esto, sin duda, ha sido un grave problema ya que la integración en determinadas plataformas no ha podido ser precisamente un camino de rosas.

Debido a que éste último apartado es bastante complejo y que muchos elementos implicados en él no tienen mucho que ver con el desarrollo propiamente dicho de una aplicación, se ha deseado crear un capítulo aparte 7 para hacer un análisis más exhaustivo acerca de las implicaciones que ha tenido conseguir que el proyecto funcionara de manera adecuada en otros entornos operativos.

5.9. Conclusión final

El proyecto ha sido un cúmulo de decisiones y duros golpes. Más o menos y en general se ha podido implementar lo que se tenía en mente y cómo se quería hacer pero, la inexperiencia ante nuevos lenguajes de programación ha sido un factor muy determinante a la hora de hacer lo que uno deseaba y en el tiempo que deseaba.

Como cuestión positiva, se puede decir que por lo menos se ha conseguido obtener unas pautas de desarrollo con lenguajes nuevos y además gracias a los "baches" se han podido mejorar algunas partes que en un principio iban a ser más simples.

La necesidad de investigar para obtener información sobre ciertas partes del proyecto, ha hecho que en términos finales se haya podido mejorar la idea inicial y se haya podido aplicar más cosas de las que se tenían planteadas en un principio.

Como cuestión negativa, comentar que ha sido en ocasiones bastante duro y poco motivador tener que estar continuamente revisando manuales e información para poder hacer ciertas acciones que en otros lenguajes de programación se saben realizar y además se hacen de forma automática. Además uno siente a nivel personal que sólo ha tocado la punta del iceberg y que aún falta mucho camino que recorrer para poder hacer algo realmente importante y digno.

Finalmente, hay que mencionar también que la implantación de la aplicación en distintos sistemas operativos ha resultados un trabajo bastante grande ya que muchas de las dependencias para que la aplicación funcionase bien no estaban disponibles y se debía hacer un "montaje propio" quizás ésto podría tratarse como una mala decisión a la hora de elegir las herramientas adecuadas pero también se podría decir que la inexperiencia y el jugar en terrenos nuevos tienen su parte de culpa ya que al fin y al cabo nunca se tiene pleno conocimiento para saber hasta dónde se puede llegar o no.

Capítulo 6

Verificación y evaluación

6.1. Introducción

Existen muchos métodos para evaluar una aplicación y sobre todo para hacer una buena serie de pruebas pero en éste proyecto se ha querido hacer algo un poco distinto a lo habitual y para ello ha hecho falta una colaboración muy importante: La del usuario final.

No existe mejor forma de probar una aplicación que, el usuario que debe usarla, la pruebe y la valore de forma directa. Cierto es, obviamente, que es necesario un buen plan de pruebas para probar las partes criticas y hacer que todo vaya fluido y sin problemas. Pero al fin y al cabo, la experiencia del usuario final es la que predomina y marca la calidad de una aplicación para que ésta pueda ser lo más acertada posible.

Para ello se ha hecho una serie de pasos que han llevado a intentar ofrecer la mejor experiencia de usuario posible.

6.2. Evaluación inicial de los prototipos de las interfaces

La primera parte de las pruebas del sistema ha consistido en una evaluación de los prototipos de interfaces. Para darle la mejor experiencia posible al usuario se ha usado una muestra de población en la que se han juntado individuos con conocimientos informáticos avanzados(estudiantes de ingeniería informática) e individuos con conocimientos básicos(albañiles, fontaneros, transportistas etc...).

A todos los individuos se les sometió a una prueba por la cual se les mostraba una serie de papeles con una interfaz impresa(hecha con Pencil) y se les mostraba la posible reacción que podía suceder al interactuar con alguno de los elementos de la misma.

Los individuos no tenían conocimiento alguno sobre cómo funcionaba la aplicación. Ni siquiera se les había dado instrucciones sobre qué hacía o para qué servían los elementos, cada sujeto debía intentar darle sentido a la interfaz que tenía delante suyo y debía ser capaz por si mismo de intentar interactuar con la aplicación sin perderse y teniendo una mínima idea de lo que al menos estaba haciendo.

Una vez terminada la sesión se les hacía una explicación completa del funcionamiento del sistema y se les pedía que rellenasen una encuesta de usabilidad para, a

posterior, hacer una medición haciendo uso de una escala SUS 1.1.5 para que pudieran dar una valoración a los prototipos de interfaz con los cuales acababan de experimentar.

El objetivo era llegar a saber una serie de cuestiones:

1. Saber si la interfaz era lo suficientemente intuitiva. Una interfaz muy bien hecha hace que un usuario sepa usar un programa sin necesitar de un asesoramiento por parte de nadie.

2. Saber si la información que se mostraba era necesaria o no. Gracias a las pruebas se podía determinar qué información demandaba un usuario a la hora de hacer uso de la aplicación.

3. Saber qué cambios se debían hacer a las interfaces para poder mejorar su comprensión y usabilidad.

Los resultados de la encuesta fueron interesantes y se mostrarán a continuación separada en dos grupos para diferenciar los conocimientos informáticos.

	Informáticos	No Informáticos
Porcentaje evaluación	57,92 %	23,33 %
Porcentaje total	81,25 %	

Tabla 6.1: Porcentaje de usabilidad de los prototipos de las interfaces

Con éstos resultados se puede sacar como conclusión que la aplicación no tiene una buena aceptación en aquellos sujetos que tienen una falta de conocimientos informáticos. La aplicación les es difícil de usar, no tiene mucha ayuda y sobre todo rompe con los esquemas de "lo esperado" esto es, no representa algo con lo que el usuario medio está acostumbrado.

En cambio, si nos fijamos en el otro lado de la balanza, la aplicación tiene mejor aceptación en usuarios que poseen mucha más experiencia en cuestiones informáticas y que pueden ver el potencial global del proyecto en si.

Si sumamos todas las consultas realizadas se obtiene una puntuación aproximada del 81 % lo cual dentro de los valores del SUS representa que la aplicación está cerca de tener un rango de excelente. Esto da a entender que las interfaces son bastante buenas y fáciles de entender, aún así, se necesita mejorarlas para que el usuario más inexperto, pueda utilizarlas sin problema alguno.

Tras una deliberación, se llegó a la conclusión que uno de los problemas de las interfaces no era simplemente que fueran poco eficaces, si no que el contexto del proyecto está orientado a un público quizás algo más experto en materia informática y que por lo tanto se necesita pensar alguna forma de hacer una curva de aprendizaje sencilla para que la aplicación resulte atractiva a cualquier usuario.

Además de la evaluación de los prototipos de las interfaces, se pidió como broche final de la encuesta, poner una serie de observaciones. Gracias a ellas, se pudo obtener una visión más amplia de las carencias que poseían las interfaces. Algunas de las propuestas realizadas por los sujetos fueron las siguientes:

- *"Poner descripciones en cada pantalla de cara al usuario más novel."*

- *"Poner el selector de los grupos en la parte superior de la tabla que muestra los alumnos de un grupo."*
- *"Resolver de alguna forma que no se añada dos veces un mismo alumno a la hora de crear un grupo."*
- *"Indicar la existencia de campos obligatorios de alguna forma (mensajes, asteriscos....)."*
- *"Poner advertencias si se intentan crear tags sin introducir script alguno."*
- *"Añadir opciones adicionales a la hora de hacer un filtrado en el historial. Filtrar por grupo, script etc.."*
- *"En el panel de administración, conseguir que se generen contraseñas automáticas para cada nuevo usuario pero sin perder la opción de hacerlo también manual."*
- *"Realización de campos dinámicos que sombreen en rojo aquellos elementos que no están correctamente introducidos."*
- *"Poner un botón de interrogación (?) para tener alguna ayuda sobre algún elemento de la interfaz."*
- *"No se para que sirve gestionar scripts. Cambiar la forma de decirlo"*
- *"Al añadir o quitar scripts, hacer alguna identificación con colores, por ejemplo, rojo para quitar y verde para añadir."*
- *"Poner un texto mejor explicativo para entender que son mis tags."*
- *"No entiendo que significa owner de un tag, poner otra cosa más sencilla de entender."*
- *"Estaría bien introducir imágenes con las acciones más relevantes, así se entiende mejor."*
- *"Encuentro difícil la ventana de mis tags, no la entiendo."*
- *"Se podría poner alguna función para donar un tag a otro profesor."*
- *"En el historial, hacer filtrado sólo por una cosa y no por varias a la vez."*

La observación más demandada fue sin duda la inclusión de algún botón de ayuda que diese una información adicional para poder entender mejor el funcionamiento de la aplicación cuando se tuviesen dudas al respecto.

Con todo ésto se refuerza la idea de hacer una curva sencilla de aprendizaje y una buena información para que los usuarios más inexpertos tengan oportunidad de entender al menos cuáles son las bondades de la aplicación que están usando.

Gracias a éstas encuestas se pudo hacer una serie de iteraciones de prototipos hasta alcanzar las interfaces finales con botones de ayuda e informaciones flotantes al pasar sobre elementos críticos para entender la razón de la existencia de un elemento en la interfaz.

6.3. Evaluación de la aplicación

En ésta sección se van a desgranar las pruebas realizadas en la aplicación en la parte cliente y en la parte servidor. La mayoría de éstas pruebas son de caja negra pero algunas de ellas, sobre todo las más críticas, se han realizado en caja blanca ya que se quería saber si los registros en la base de datos y la lógica de funcionamiento esperado era el correcto.

6.3.1. Pruebas en la parte cliente de la aplicación

Cod	Descripcción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
1	El usuario decide elegir un grupo distinto en la lista de selección de grupos disponibles.	Pulsa el selector de la lista de grupos.	Se muestran correctamente la lista de todos los grupos que ha creado ese usuario y que le pertenecen.	Al seleccionar el comboBox, éste muestra todos los grupos disponibles.	X		
2	El usuario selecciona uno de los grupos disponibles en su lista.	Pulsa en un grupo en la lista desplegada de grupos.	Se actualiza de forma automática todos los nombres de los alumnos mostrando únicamente los alumnos que componen el grupo seleccionado.	La aplicación realiza la acción esperada y el servidor responde de manera rápida.	X		
3	El usuario intenta realizar una acción de grupo cuando no tiene ningún grupo seleccionado actualmente.	Intenta pulsar los botones: "Gestionar Scripts" – "Cambiar nombre" – Eliminar	El usuario debe ser incapaz de hacerlo ya que los botones están desactivados cuando no existen grupos seleccionados.	El resultado es el esperado.	X		
4	El usuario intenta seleccionar un grupo determinado cuando éste aún no ha sido creado uno previamente.	Pulsa la lista desplegable de grupos.	El usuario no puede ser capaz de selecciona ningún elemento ya que sólo existe el mensaje de que no tiene ningún grupo creado.	La aplicación funciona de manera correcta quitándose posibilidad alguna al usuario de poder elegir nada.	X		
5	El usuario decide cambiar los scripts de un grupo cuando no tienen ninguno seleccionado.	Pulsa el botón "Gestionar Scripts".	El usuario debe ser incapaz de pulsar el botón porque dicho botón debe estar desactivado al no tener grupo alguno seleccionado.	No es posible pulsar el botón ya que el usuario no tiene grupo alguno seleccionado por lo que los botones no son virtualmente accesibles.	X		
6	El usuario decide cambiar los scripts de un grupo cuando tiene al menos un grupo seleccionado.	Pulsa el botón "Gestionar Scripts.	El usuario accede a una interfaz que contiene todos los scripts y tags disponibles para gestionar el grupo y ninguno de ellos aplicado.	La interfaz se le muestra de manera correcta, con los scripts y tags disponibles para aplicar y con la lista de scripts y tags aplicados vacía.	X		Puede darse el caso que en la columna de "disponibles" no se muestren ningún Tag/Script debido a que estén desactivados por el administrador.
7	El usuario decide modificar los scripts de un grupo que previamente ya ha modificado	Pulsa el botón "Gestionar Scripts"	El usuario accede a una interfaz que contiene los scripts y tags disponibles y muestra además cuales ya ha aplicado anteriormente en el grupo seleccionado.			X	No mostraba de forma correcta los Tags y los Scripts. A veces ocurria que daba por hecho que un Tag y un Script a la vez estaban aplicados
7.1				Se muestra de forma correcta aquellos Tags/Scripts que estaban previamente aplicados en la columna de aplicados y además se muestran los Tags/Scripts en la columna de disponibles.	X		Se corrigió el error insertando rutinas de control en el *ServerReceiver*. Se hizo el recorrido haciendo prueba de caja negra.

Tabla 6.2: Pruebas de la parte cliente - 1

Cod	Descripcción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
8	El usuario decide aplicar un script a un grupo que previamente no tenía ningún Tag/Script aplicado.	Elige un Script de la columna de disponibles y lo pasa a la columna de aplicados y pulsa “Aplicar”.	El Script se aplica de forma correcta y se cierra la interfaz de gestionar Scripts.			X	Los Scripts/Tag no se aplicaban de forma correcta. Ocurrían errores
8.1				La acción se desarrolla de forma correcta. El script se aplica correctamente.	X		Se arregló la lógica de decisión de aplicar un Script y registrar la acción
9	El usuario decide aplicar un script a un grupo que previamente tenia algún Tag/Script aplicado.	Elige un Script de la columna de disponibles y lo pasa a la columna de aplicados y pulsa “Aplicar”.	El Script se aplica de forma correcta, sólo si éste no ha sido aplicado previamente por algún Tag y se cierra la interfaz.			X	No se hacía de forma correcta las operaciones de filtrado y se producían entradas duplicadas en la BD
9.1				El funcionamiento es correcto.	X		Se mejoró la capacidad de filtrado para hacer que si un Script ya había sido aplicado por un Tag éste no pudiese hacerlo de forma normal.
10	El usuario decide aplicar un Tag a un grupo que previamente no tenía ningún Tag/Script aplicado.	Elige un Tag de la columna de disponibles y lo pasa a la columna de aplicados y pulsa “Aplicar”.	El Tag se aplica de forma correcta y se cierra la interfaz de gestionar Scripts.	El Tag se aplica de forma correcta y se cierra la interfaz de gestionar Scripts.	X		Se realizaron pruebas de caja negra para comprobar el correcto funcionamiento.
11	El usuario decide aplicar un Tag a un grupo que previamente tenía algún Tag/Script aplicado.	Elige un Tag de la columna de disponibles y lo pasa a la columna de aplicados y pulsa “Aplicar”.	Si el Tag a aplicar contiene un Script que ya está en algun otro Tag aplicado previamente, el sistema dará error y no le dejará añadirlo. Si no, aplicará el Tag sin problema.			X	Ocurrieron ciertos problemas a la hora de conseguir funcionar éste apartado
11.1				El funcionamiento es correcto tal y como se espera.	X		Se consigió adoptar un Script al añadir un Tag que lo contenía.
11.2				El funcionamiento es correcto tal y como se espera.	X		Se consigió evitar que se produjeran colisiones entre Tags con el mismo script.
12	El usuario intenta cambiar el nombre del grupo sin tener seleccionado ningún grupo en la lista de grupos.	Pulsa el botón “Cambiar nombre grupo”	El usuario no puede ser capaz de acceder a cambiar el nombre del grupo porque no tiene ningún grupo seleccionado por lo que el botón se halla desactivado.	El funcionamiento es correcto tal y como se espera.	X		El sistema es sensible a la introducción de caracteres extraños, sólo deja letras normales.

Tabla 6.3: Pruebas de la parte cliente - 2

Cod	Descripcción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
13	El usuario selecciona un grupo y decide cambiar su nombre pero introduce uno que ya existe.	Pulsa el botón "Cambiar nombre grupo". Introduce un nombre y acepta	El sistema le devuelve una advertencia, indicando que no se puede realizar la acción.	El sistema funciona de forma adecuada.	X		El sistema le advierte de un error a la hora de cambiar el nombre y le ofrece una descripción detallada. En ella le explica que la razón es debido a una duplicación(nombre existente) o a que ha habido un error general de conexión.
14	El usuario selecciona un grupo y decide cambiar su nombre introduciendo un nombre válido.	Pulsa el botón "Cambiar nombre grupo". Introduce un nombre válido y pulsa aceptar.	El sistema devuelve un aviso indicando que el nombre del grupo se ha cambiado de forma satisfactoria y actualiza la lista de grupos de la pantalla principal con el nuevo nombre del grupo introducido.			X	El nombre se cambiaba de forma adecuada pero la pantalla principal no se refrescaba como debía.
14.1				El sistema funciona de manera adecuada.	X		Se arregló la interfaz forzando a la pantalla principal a recargar la tabla.
15	El usuario selecciona un grupo y decide cambiar su nombre introduciendo un nombre en blanco.	Pulsa el botón "Cambiar nombre grupo". Introduce un nombre en blanco y pulsa aceptar.	El sistema le devuelve un error indicándole que al menos debe introducir un nombre válido.	El sistema devuelve un error de manera correcta.	X		
16	El usuario intenta eliminar un grupo sin haber seleccionado uno previamente.	Pulsa el botón "Eliminar grupo"	El usuario es incapaz de realizar la acción debido a que el botón está deshabilitado y por ende es incapaz de hacer la acción de borrado.	El sistema funciona de manera adecuada.	X		
17	El usuario selecciona un grupo, decide borrarlo y confirma la acción. Éste grupo no tenía ningún Script o Tag asignado previamente.	Pulsa el botón "Eliminar grupo". Confirma el borrado pulsando "borrar".	El sistema elimina el grupo seleccionado y actualiza la lista de los grupos de la pantalla principal eliminando el grupo borrado.	El sistema elimina el grupo del sistema y actualiza la pantalla principal mostrando los grupos que queden.	X		Si los alumnos dejan de pertenecer a algún grupo, ésta acción los elimina del sistema. Se realizó pruebas de caja negra.
18	El usuario selecciona un grupo y decide borrarlo y confirma la acción. Éste grupo tenía asociados algún Tag/Script.	Pulsa el botón "Eliminar grupo". Confirma el borrado pulsando "borrar".	El sistema elimina el grupo seleccionado y actualiza la lista de los grupos de la pantalla principal eliminando el grupo borrado. Además elimina los Tags/Scripts aplicados sobre todos los alumnos afectados en dicho grupo.			X	El sistema no respondía de forma adecuada ya que no registrada bien el proceso en el historial y además por alguna razón no hacia de forma correcta el borrado. Aparecían problemas a la hora de eliminar correctamente los Scripts/Tags asociados al grupo.

Tabla 6.4: Pruebas de la parte cliente - 3

Cod	Descripcción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
18.1				El sistema funciona de manera adecuada. Borrar el grupo y remueve todas las aplicaciones de Scripts/Tags si las hubiera.	X		Se corrigió el fallo modificando el método de creación del historial para un grupo.
18.2				El sistema funciona de manera adecuada. Borrar el grupo y remueve todas las aplicaciones de Scripts/Tags si las hubiera.	X		Se corrigió el fallo comprobando la ejecución y modificando la lógica del borrado. Se hizo uso de pruebas de caja negra.
19	El usuario pulsa el botón de crear un nuevo grupo cuando aún no ha creado grupo alguno en el sistema.	El usuario pulsa el botón "Crear grupo nuevo".	Se abre la interfaz que le permite crear un nuevo grupo.	El sistema funciona según descripción.	X		
20	El usuario pulsa el botón de crear un nuevo grupo cuando ya tiene uno o varios grupos creados anteriormente.	El usuario pulsa el botón "Crear grupo nuevo".	Se abre la interfaz que le permite crear un nuevo grupo.	El sistema funciona según descripción.	X		
21	El usuario desde la interfaz de añadir un nuevo grupo decide añadir un nuevo alumno de forma manual cuando no existen alumnos aún añadidos en el grupo a crear.	El usuario pulsa el botón "añadir alumno".	Se le abre una interfaz que le permite introducir de forma manual todos los datos de un alumno.	El programa funciona según lo descrito.	X		
22	El usuario desde la interfaz añadir un nuevo grupo decide añadir un nuevo alumno cuando ya existen varios añadidos.	El usuario pulsa el botón "añadir alumno".	Se le abre una interfaz que le permite introducir de forma manual todos los datos de un alumno.	El programa funciona de forma correcta.	X		
23	El usuario desde la interfaz de añadir de forma manual un nuevo alumno, introduce de forma incorrecta alguno de sus datos	El usuario introduce(o no) los datos y pulsa "Añadir alumno"	Se le muestra un mensaje advirtiéndole que le falta un campo determinado	La aplicación muetra la ventana de manera correcta.	X		El mensaje que se le muestra al usuario dependerá del campo que haya introducido mal.
24	El usuario desde la interfaz de añadir de forma manual un nuevo alumno, introduce de forma correcta todos los datos de un alumno.	El usuario introduce todos los datos y pulsa "Añadir alumno".	Se le muestra una ventana de información para indicarle que todo ha ido bien y se le devuelve a la ventana de creación de grupo.			X	El control de repetición de alumno no funcionaba de forma adecuada y se podían añadir dos alumnos iguales.
24.1				La aplicación responde perfectamente, avisando de la creación exitosa y enviando al usuario a la pantalla correcta.	X		Se corrigió el problema y se incluyó una pantalla de aviso que mostrará al usuario que había intentado introducir dos veces el mismo alumno.

Tabla 6.5: Pruebas de la parte cliente - 4

Cod	Descripcción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
25	El usuario accede al panel de los Tags cuando no tiene ninguno creado previamente.	Pulsa el botón “Gestionar mis Tags”.	Se abre una interfaz que no debe contener ningún Tag a excepción de que le haya sido concedido por algún otro usuario.	El sistema funciona perfectamente de acuerdo con el resultado esperado.	X		
26	El usuario accede al panel de los Tags cuando tiene creado uno o varios Tags distintos.	Pulsa el botón “Gestionar mis Tags”.	Se abre una interfaz que muestra que Tags tiene actualmente creados aportándole la información pertinente.	El sistema muestra correctamente los Tags que tiene el usuario.	X		Puede ocurrir que por cuestiones de borrado por parte del administrador. No se le muestre ningún Tag creado con anterioridad.
27	El usuario decide crear un Tag pero no incluye ningún Script en su creación.	El usuario pulsa “Aplicar” cuando no tiene ningún Tag “En el Tag”.	Se le muestra al usuario una pantalla de información que le indica que al menos debe añadir un Script.	El sistema funciona de manera correcta.	X		
28	El usuario decide crear un Tag pero no incluye alguno de sus datos	El usuario pulsa “Aplicar” cuando no ha introducido todos los datos del Tag	Se le muestra una advertencia que le pide que debe de introducir determinados datos	El sistema muestra la advertencia de manera adecuada.	X		El tipo de advertencia varía dependiendo del dato que falte.
29	El usuario decide crear un Tag e introduce todos los datos requeridos para su creación.	El usuario pulsa “Aplicar “ cuando ha introducido correctamente todos los datos.	El sistema le muestra una pantalla de información indicando al usuario que el Tag se ha creado de forma correcta.			X	El Tag no se creaba de forma correcta debido a fallos en la asignación de Scripts en el Tag.
29.1					X		Se corrigió el error modificando el algoritmo de creación de un Tag en la parte del gestor de los Scripts y Tags
30	El usuario decide modificar un Tag ya creado previamente, modificando únicamente el nombre del Tag e insertando un nombre de otro Tag ya existente.	El usuario pulsa el botón “Aplicar” cuando ha introducido un nuevo nombre ya existente.	El sistema muestra una ventana de error indicando el fallo de la creación del Tag e informando al usuario que la causa puede ser debida a que ha puesto un nombre que ya existe en otro Tag.			X	Se duplicaban los Tags. Algo no iba bien.
30.1				El sistema muestra de forma correcta el error.	X		Se obtuvo la lista de todos los Tags con sus nombres y se creo en el local una función que comprobara los nombres antes de hacer cualquier llamada al servidor.

Tabla 6.6: Pruebas de la parte cliente - 5

Cod	Descripcción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
31	El usuario modifica un Tag cambiando el propietario a una persona que no sea el.	El usuario selecciona un usuario que no sea el mismo y pulsa el botón “Aplicar”	El sistema cambia el Tag de propietario y notifica al usuario del éxito de la operación.			X	El cambio de propietario no se hacía de forma correcta. El Tag no era eliminado correctamente en todos los grupos del usuario y el usuario destino podía recibir un Tag con cualquier nombre.
31.1				El sistema realiza la función sin problema alguno.	X		Se modificó el método de borrado de Tag para el usuario actual y se intentó hacer una integración con el borrado de un Tag por parte del usuario actual. Además se corrigieron algunas sentencias SQL para la obtención correcta de los datos.
31.2				El sistema realiza la función sin problema alguno.	X		Se insertó una metodología de reconocimiento de nombres de los Tags antes de hacer el envío y se corrigieron algunas sentencias SQL que no obtenían el usuario de forma correcta por su identificador.
32	El usuario modifica un Tag cambiando uno o varios Scripts del mismo.	El usuario añade o elimina Scripts del Tag y pulsa el botón “Aplicar”	El sistema revoca en todos los Grupos afectados por el Tag aquellos Scripts que el usuario haya eliminado del Tag y añade (si es posible) los Scripts que haya añadido.			X	Surgieron varios errores debido a que a la hora de introducir un Script que ya estaba incluido en otro Tag que estaba ya aplicado en un grupo, se sucedían colisiones. En otras ocasiones, no se modificaban los Scripts de forma correcta porque los borrados no se hacían como se debía de hacer.
32.1				El resultado es el esperado.	X		Se modificó el orden de borrado de los Scripts para evitar problemas y se añadío un control para la correcta aplicación.
32.2				El resultado es el esperado.	X		Se introdujo un control que evitaba aplicar un Script en un grupo si éste ya tenía un Tag que lo contenía. Además se programó un algoritmo que adoptaba el Script para darle preferencia al Tag y hacer así compatible la modificación en el grupo.

Tabla 6.7: Pruebas de la parte cliente - 6

Cod	Descripcción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
33	El usuario accede al historial cuando nunca antes ha realizado ningún tipo de acción en el sistema	El usuario pulsa el botón "Mostrar Historial"	Se le muestra una nueva interfaz que no contiene ninguna entrada determinada a no ser que por cuestiones de sistema se haya modificado algo al usuario.	La interfaz no muestra ningún tipo de resultado ya que nunca antes se ha realizado cambios en el sistema que deban ser registrados en el historial.	X		
34	El usuario accede al historial cuando ha realizado una serie de movimientos como aplicar Tags/Scripts, crear grupos, modificar grupos, crear/modificar/borrar Tags etc.	El usuario pulsa el botón "Mostrar Historial"	Se le muestra una nueva interfaz que le informa sobre qué cambios se ha hecho en el sistema, indicándole los componentes afectados, los razonamientos y si se ha borrado o añadido.			X	El historial no mostraba de forma correcta todos los elementos. Algunos apartados no estaban correctamente programados o simplemente no funcionaban como se esperaban. La pantalla daba problemas a la hora de saber cuándo debía pintar un Tag o un Script
34.1					X		Se añadieron nuevas entradas para registrar de forma correcta los eventos que no se mostraban y se cambiaron ciertas lógicas en los gestores para poder insertar de una manera más eficiente y limpia las inserciones en el historial.
34.2					X		Se modificó el funcionamiento de la lista de historiales en el modelo de datos para mejorar una función que permitiera diferenciar entre Tag y Script y pintara de forma correcta los elementos en la tabla.
35	El anónimo introduce un usuario y contraseña inválidos y pulsa aceptar.	El anónimo pulsa el botón "Logi"n.	Aparece una ventana indicando que los datos introducidos son inválidos.	Al introducir de forma incorrecta las credenciales. Una ventana nos avisa de ello.	X		
36	El anónimo introduce un usuario y contraseña válidos y pulsa aceptar cuando no tiene previamente creado ningún grupo.	El anónimo pulsa el botón "Login".	Se le muestra la venta principal de la aplicación y se le indica que no posee grupo alguno.			X	En ocasiones sucedía algún fallo puntual por el cual la interfaz mostraba datos cuanto el usuario no tenía grupos.
36.1				La aplicación muestra un combo box vacío de elementos con un mensaje indicativo diciendo que no se posee grupos.	X		Se modificó la interfaz principal forzándola a hacer un prerefresco siempre que entrara un nuevo usuario.

Tabla 6.8: Pruebas de la parte cliente - 7

Cod	Descripción	Acción	Resultado esperado	Resultado real	OK	KO	Comentarios
37	El anónimo introduce un usuario y contraseña válidos y pulsa aceptar.	El anónimo pulsa el botón "Login"	Se le muestra la ventana principal de la aplicación con un grupo pre-seleccionado en caso de tener alguno			X	La funcionalidad de la identificación era correcta. Se cargaba la interfaz principal y todo iba sobre ruedas. No obstante, los grupos no se mostraban de forma ordenada por la fecha de creación.
37.1				El Login funciona de forma correcta, mostrando la pantalla principal y los grupos de forma ordenada	X		Se modificó las sentencias de la base de datos corrigiendo la llamada y forzando a una ordenación por fecha de creación.

Tabla 6.9: Pruebas de la parte cliente - 8

Capítulo 7

Instalación

7.1. Introducción

Debido a la naturaleza especial que posee éste proyecto se ha deseado realizar un capítulo para poder hablar acerca de todo el proceso que ha conllevado conseguir implantar el proyecto en los sistemas operativos "GNU/Linux" y "Windows".

En comparación con otro tipo de proyectos, se podría llegar a la conclusión por la cual en éste caso ha sido necesario invertir una buena cantidad de horas para poder conseguir que la aplicación funcionase de manera deseable en otros entornos operativos y para ello, se ha necesitado hacer una serie de pasos que no tienen nada que ver con la programación directa de un proyecto.

El desarrollo ha sido portado para "Windows" y "GNU/Linux". Cada sistema operativo tiene sus peculiaridades y por ende cada uno de ellos ha presentado un reto diferente para poder conseguir que todo funcione de forma correcta.

El proyecto ha sido programado dentro de un entorno "GNU/Linux" pero debido a la gran gama de distribuciones existentes y a las diferencias que existen entre unas y otras, se ha tenido que realizar unos cuantos ajustes para que el proyecto tuviese un funcionamiento digno y deseable.

7.2. Portando el proyecto a GNU/Linux

Para comenzar, la primera tarea ha consistido en portar la aplicación a la distribución "GNU/Linux" *Ubuntu* en su versión 14,04. Ésta versión es muy estable y es la usada actualmente dentro de la universidad, pero, por casualidades de la vida y por cuestiones extrañas ésta versión no dispone de los binarios de dos de los paquetes necesarios para que el programa funcionase de forma adecuada. De hecho, los binarios existen para la versión 3 de "Python" pero no para la versión 2,7 que es la usada en el proyecto. Ahora bien, resulta impactante ver como en versiones posteriores de la distribución si aparecen dichos paquetes que dan soporte completo, lo cual resulta un tanto confuso y a la vez frustrante.

Los binarios que faltaban eran *PyQt5* y *Sip*. El primero, ya es conocido debido a que es la herramienta que permite crear interfaces "QT" a través de código "Python". El segundo es una dependencia necesaria para que *PyQt5* funcione de forma adecuada.

¿Cómo solucionar todo ésto? Básicamente el remedio ha sido la compilación manual de las dependencias que eran necesarias, el problema era saber exactamente

cuál era el método más adecuado. La solución se halló en una página web que explicaba el proceso para una versión anterior de *Ubuntu* pero que, afortunadamente, funcionaba correctamente en la versión 14,04. [8]

La idea es sencilla, para compilar *PyQt5* se necesitan dos herramientas, la primera es *Sip* en una versión superior a la que viene en *Ubuntu* y la segunda es tener *Qmake* que es una herramienta propia de las librerias de "QT" para la compilación de programas basadas en dicha plataforma. Con ésto, e instalando una serie de dependencias básicas para compilar aplicaciones (*python-dev, gcc, librerías optativas para "QT" y más*), se consigue lograr un entorno de compilación compatible para que *PyQt5* y *Sip* puedan ser creados sin problema alguno.

La cuestión es que una vez superada la barrera de conseguir las librerías necesarias viene otro nuevo problema: El empaquetamiento de dichas librerías para crear binarios compatibles con la distribución *Ubuntu*.

Empaquetar una aplicación consiste en la acción de agrupar los directorios creados por la compilación en un único archivo que contenga toda la información necesaria para poder ser instalado en un entorno operativo "GNU/Linux". Existen muchas formas de empaquetar una aplicación y por norma general cada distribución "GNU/Linux" tiene la suya propia. En el caso de Ubuntu, el tipo de empaquetación usado son los binarios de *Debian Package* que son paquetes de instalación usados por una distribución de "GNU/Linux" muy famosa llamada *Debian*.

Para conseguir dicha empaquetación se ha hecho uso de *checkinstall* que es una herramienta sencilla que permite generar los paquetes con extensión *.deb. Obviamente existen mejores alternativas que permiten, entre otras cosas, firmar los paquetes para verificar la autenticidad de quien los ha creado(Un usuario reconocido por la comunidad) pero, en éste caso, al ser un proyecto más simple se ha optado por una herramienta rápida y sencilla que resuelva el problema de integración de la forma más automática posible.

Con todo ello se ha podido crear cuatro paquetes binarios(dos para entornos de 32 bits y otros dos para entornos de 64 bits) que tras ser instalados, permiten la ejecución correcta de la aplicación dentro de la distribución "GNU/Linux" *Ubuntu*.

7.3. Portando el proyecto a Windows

La instalación y adecuación para un entorno Windows ha sido más o menos similar al caso de "GNU/Linux" pero con varios matices.

7.3.1. Preparación de un entorno de compilación

En un primer lugar, no existían problemas para obtener "Python" en "Windows" pero algunas librerías usadas en el proyecto no estaban para "Windows" en determinadas arquitecturas. Por ejemplo, "PyQT" no estaba disponible para versiones de "Python" 2,7 haciendo uso de "QT5". Por lo tanto, podríamos decir que se estaba ante un caso similar al anterior. Pero el problema iba a encaminarse a algo más complejo.

De la misma forma que en el caso anterior, es nuevamente necesaria la compilación de las librerías *SIP* y *PyQt5* pero además, se tiene que sumar la instalación manual de "QT".

Instalar "QT" es relativamente sencillo en "Windows", basta con bajar un instalador propio de la web oficial y elegir la licencia deseada(en éste caso la licencia GPL). El problema aparece cuando se necesita saber qué componentes se deben instalar ya que el instalador muestra una serie de versiones de "QT" y cada una de ellas tiene su propio conjunto de paquetes. Dicho conjunto de paquetes consiste en la elección de cómo se quieren compilar las interfaces dentro de "Windows" ya que existen varios métodos efectivos para ello y cada paquete representa el método a elegir.

Por defecto, un sistema operativo "Windows" no posee los comandos para hacer *make* por lo que es virtualmente imposible hacer la compilación de *SIP* y *PyQt5*. Sólo existe una manera y fue encontrada en un blog. [22]

Para conseguir una compilación exitosa es necesario instalar el programa "Visual Studio 2012". Dicho programa, aparte de realizar una gran serie de tareas interesantes, tiene una herramienta fundamental: La consola de desarrollador.

Ésta consola consiste en un símbolo del sistema típico de "Windows" (cmd.exe) pero con una serie de comandos agregados que permiten compilar código *C++*. De entre esa serie de comandos se encuentran dos que son necesarios:

- **nmake:** Consiste en un comando que hace el análogo del *make* en los sistemas "GNU/Linux". Gracias a éste comando se pueden ejecutar los *makefiles* y compilar el programa deseado.
- **cl:** Es un comando que permite hacer la compilación de un fichero determinado escrito en *C++*. Se podría decir que es el compilador *gcc* que poseen los sistemas operativos "GNU/Linux".

Por lo tanto, sabiendo qué método de compilación se iba a usar, sólo faltaba decirle al instalador de "QT" qué paquetes eran necesarios descargar para su instalación. En éste caso, el conjunto de paquetes son los llamados *msvc2012* que representan a "Visual Studio 2012".

Con "Visual Studio 2012" preparado y "QT" instalado correctamente, sólo faltaba seguir los pasos del blog para preparar bien los path del sistema y tener los comandos de la consola de "Python" y el compilador de "QT"(qmake) accesibles desde un terminal de "Windows".

Satisfechos todos éstos requisitos, ya era posible desde un terminal hacer prácticamente los mismos pasos que en un sistema operativo "GNU/Linux" para la compilación exitosa de *SIP* y *PyQt5*. De hecho, dichos pasos eran exactamente iguales, pero haciendo uso de *nmake* para la compilación y la instalación de las librerías en el sistema.

7.3.2. Adecuación de la compilación

Evidentemente la cosa no termina en éste punto porque al ejecutar el código y hacer uso de las librerías recién creadas, el intérprete de "Python" adolecía de la falta de los ficheros *.dll de dichas librerías(extraño cuando se supone que todo está compilado y funcionando). El problema no era que las cosas estuviesen mal compiladas, el problema era que se hacía necesaria la integración de los ficheros *.dll de la instalación de "QT" dentro de la paquete "PyQt5" recién creado.

¿Cómo resolver ésto? básicamente el paso consistía en copiar todo el directorio *bin* de la instalación de "QT" dentro del directorio donde se encuentra "PyQt5"(dicho directorio está dentro de la carpeta donde se instala el intérprete de "Python").

Una vez completados estos pocos y sencillos pasos, la aplicación era completamente funcional dentro de un entorno "Windows" aunque, aún eran necesarios algunos retoques más para conseguir que "PyQt5" fuera funcional en cualquier ordenador con "Windows".

Para que otros ordenadores con un sistema operativo "Windows" puedan usar correctamente el paquete compilado de "PyQt5" es necesario que tengan instalados "Visual Studio 2012" ya que son necesarias unas librerías externas a la instalación de "PyQt5" que permiten la ejecución de las interfaces. Obviamente, instalar "Visual Studio 2012" en cada ordenador que quiera hacer uso del proyecto no es para nada deseable y se necesita una solución inmediata y mucho más simplista. Dicha solución fue encontrada en el mismo blog que explicaba los pasos para la compilación de "PyQt5" en "Windows".

Según el autor, cuando se instala "Visual Studio", en la carpeta *system* de "Windows", que es una carpeta que contiene archivos importantes relacionados con el sistema operativo, se crean varias librerías. De entre ellas, algunas son usada para que "PyQt5' pueda funcionar como debe pero para ello es necesario saber cuál o cuáles son. El autor del blog expone que para identificarlas, hizo uso de una herramienta llamada *depends.exe* y además hizo una comparativa de la carpeta *system* de un "Windows 7" recién instalado contra la misma carpeta de un "Windows 7" con la compilación exitosa realizada. Gracias a ello descubrió dos archivos que resultaban ser las dependencias buscadas.

Los archivos descubiertos tienen el nombre de *msvcp110.dll* y *msvcr110.dll* y es necesario hacer una copia de ellos y enviarlas a la carpeta de "PyQt5" junto con las librerías de "QT" que habían sido copiadas anteriormente.

Con todos estos pasos necesarios se obtiene un interprete de "Python" que contiene todas las librerías necesarias para que la aplicación funcione de forma correcta en cualquier equipo que haga uso de "Windows", ahora bien, cabe recalcar que como cualquier aplicación actual de escritorio, hace falta hacer la compilación dos veces: Una para la arquitectura de 32 bits y otra para la arquitectura de 64 bits.

7.3.3. Creación de un archivo de instalación

Con "Python" y sus dependencias preparadas en un paquete con la capacidad de ejecutarse en cualquier máquina y la parte cliente de la aplicación completada, sólo falta añadir el último eslabón para cerrar ésta gran cadena: la creación de un instalador.

La forma más simple para que un usuario pueda hacer uso del programa, es entregándole un instalador que se encargue de introducir en el sistema toda la aplicación junto con el interprete de "Python", para que todo pueda funcionar de la forma más automática posible. La cuestión es ¿Cómo se hace un instalador? Quizás es una de las preguntas que más controversia ha generado debido a que un instalador es hablar de palabras mayores, es acercar el proyecto a una auténtica aplicación nativa de escritorio, o mejor dicho, hacer que el proyecto sea todo un programa real y serio.

Para conseguir una instalación efectiva se necesita completar tres sencillos pasos:

1. Se necesita un intérprete de "Python" que sea portable en distintos sistemas operativos y que ejecute, de forma correcta, el código generado en el proyecto. Además, éste debe tener compiladas todas las librerías necesarias para la arquitectura a la que pertenezca.

2. Se debe poner el proyecto completo junto con el intérprete anteriormente mencionado para generar un gran paquete que contenga todo el proyecto con todos los requisitos para hacerlo funcionar

3. Es necesario utilizar una herramienta que genere un ejecutable y que diga cómo y dónde se debe instalar todos los datos en cada nuevo ordenador donde sea ejecutado.

Para conseguir todo ésto se realizó una búsqueda de información. Había una gran cantidad de herramientas y métodos distintos para conseguir hacer un ejecutable. Algunos eran simples y demasiado sencillos, nada deseables para un usuario novel. Al final se optó por el método utilizado por un investigador científico que parecía solventar la situación de un modo más que razonable. [25]

Lo que éste investigador expone, es hacer uso de un intérprete de "Python" portable llamado *WinPython* que contiene una serie de herramientas adicionales para poder instalar o desinstalar paquetes para "Python" que extienden su funcionalidad. En éste caso, sólo se necesitaba la posibilidad de que el intérprete pudiera ejecutarse en otras plataformas sin problema alguno.

Con *WinPython* preparado, se instaló las librerías compiladas de *Sip* y *PyQt* y se procedió a juntar el proyecto para crear un gran paquete. Gracias a ésto se tuvo la capacidad de hacer funcionar el proyecto en cualquier ordenador bajo el sistema operativo "Windows".

El último paso era simplemente hacer uso de *Inno setup* para crear un instalador clásico *.exe para sistemas operativos "Windows'. Por desgracia, no ha sido posible aún escribir un script que automatice en un 100 % la capacidad de hacer una instalación completa, sencilla y eficiente con unos pocos clicks. Ésto es algo que se dejará como una línea futura ya que por lo menos, el proyecto es funcional en "Windows" y es usable en cualquier ordenador con arquitecturas de 32 y 64 bits.

Capítulo 8

Conclusiones y trabajo futuro

8.1. El final no es más que el principio

Después de toda la experiencia que ha sido realizar una aplicación de ésta índole, llega el final de todo en el que se debe al menos recapacitar y recapitular todos los acontecimientos que se han ido produciendo en cada avance.

Por ello, se va a dividir ésta conclusión en tres apartados distintos para enfatizar mejor cada aspecto de la finalización de éste TFG.

8.2. ¿Qué se ha desarrollado?

Aunque se ha comentado varias veces en el proyecto y se ha mostrado toda la información referente a ello, se va a resumir un poco qué se ha desarrollado.

En éste TFG se han desarrollado dos aplicaciones; una orientada a la experiencia del usuario(parte cliente) y otra orientada a la experiencia del administrador del sistema(parte servidor). La idea es que ambas formen un completo programa de gestión que ayude a los profesores a realizar una serie de labores de una forma más rápida y eficiente. Por tanto hablaremos de éstas dos "experiencias"

8.2.1. La parte Cliente

La primera de ellas es una aplicación que se ha considerado como "parte cliente" y que consiste en un programa funcional que se instala en el ordenador de un usuario concreto y de unos archivos que se instalan en una máquina servidor. El cliente, mediante una interfaz intuitiva realiza una gestión de los alumnos asignándolos a grupos para después, por cada grupo, tener la capacidad de poder asociarles unas acciones llamadas scripts. Éstos scripts representan unos archivos que contienen una serie de comandos que automatizan una serie de procesos. Por ejemplo, un script puede permitir que un alumno sea registrado en una base de datos y se le pueda enviar un e-mail de forma automática con sus credenciales de acceso.

Como gestionar tantos scripts puede llegar a ser un incordio, se ha implementado una funcionalidad llamada tags que permite agrupar uno o varios scripts a la vez. De ésta forma, el proceso anteriormente descrito se hace más dinámico y genera un mayor automatismo.

El usuario puede crear tantos grupos como desee y poner o quitar tantos scripts o tags como quiera, siempre y cuando no existan problemas de colisiones. Entendemos

por colisión al problema que surge cuando un mismo script intenta ser aplicado dos veces en un mismo grupo. La lógica de todo ello es debido a que no tiene sentido dar de alta en un servicio dos veces a un mismo alumno. En caso de querer dar de alta dos veces a un mismo alumno en un mismo servicio, como mínimo, debería existir algún elemento diferenciador que justifique una operación de éste tipo(por ejemplo, que demos dos accesos a una base de datos para dos tipos distintos de proyectos).

Como el usuario, puede ser muy olvidadizo se le ha concedido la capacidad de tener una herramienta que le permita "recordarle" qué cosas ha hecho. Para ello, existe lo que se llama historial. El historial permite al usuario ver las acciones que ha hecho para que éste sepa que hizo y porque razón.

El sistema permite ser muy solidario, por eso se ha implementado la capacidad de hacer "donaciones" de los tags entre los distintos usuarios. Ésto se ha pensado así porque hoy un profesor da una asignatura de un tipo y el siguiente curso dicha asignatura pasa a manos de otro profesor, por lo tanto y como buenos compañeros, se pasan los tags como si se trataran de cromos.

Ésta parte tiene implementada una lógica que permite al usuario que use la aplicación, conectarse con un servidor y enviar sus decisiones. Dicho servidor, que se encuentra en una máquina externa a la suya, tiene unos archivos que permiten "escuchar" las peticiones del cliente y hacer las órdenes que haya recibido. Una vez que las ha hecho, le devuelve al usuario unos resultados.

Para hacer ésta parte, ha sido necesario hacer una combinación de un lenguaje de programación llamado "Python" y de un lenguaje gráfico llamado "QT". Como dichos lenguajes no se entienden de forma directa, se ha pedido ayuda a un intérprete llamado *PyQT* que ha traducido las instrucciones de "Python" para que "QT" le entienda y juntos puedan trabajar para darle la mejor experiencia posible al usuario.

Como el mundo informático actual hay gente que piensa muy mal, se ha implementado una conexión segura haciendo uso de una tecnología llamada *socket en ssl*. Con ésta tecnología se ha podido dar una protección de privacidad a los usuarios para que puedan manejar los datos sensibles de los alumnos sin que ningún mal pensado intente hacer algo que no debería hacer.

La parte Servidor

La segunda de ellas, se la ha considerado como la "parte servidor". Ésta, consiste en un simple programa que se ejecuta desde la propia máquina servidor, por lo tanto, es necesario tener un acceso directo a los datos físicos.

Con ésta parte se ha querido dar a un usuario con unos permisos muy especiales, el poder para decidir quiénes pueden usar la herramienta y qué scripts pueden aplicar. Es una forma segura de saber que cualquiera no puede entrar como si estuviera en su casita.

Se ha implementado dos funcionalidades:

1. La primera de ellas ha consistido en una gestión de quiénes pueden acceder al sistema. En ella se les da al administrador la autoridad de crear, modificar y borrar usuarios en el sistema. Él decide quién entra y nadie le puede molestar ni decir lo contrario.

2. La segunda, ha consistido en una gestión de qué scripts se pueden usar. El administrador, con su sapiencia profunda e infinita, es quien escribe los scripts

y él mismo los pone a disposición de cualquiera que quiera usarlo. Obviamente de la misma forma que puede crear, también puede destruir y la idea precisamente es que él tenga siempre la última palabra.

De una forma tan simple, se ha podido cubrir y controlar cómo es el funcionamiento que se le quiere dar al programa. Cabe reseñar que al igual que en la "parte cliente" se ha hecho de la combinación de "Python" + "QT" y de su mejor amigo, el traductor *PyQT*.

8.3. ¿Qué objetivos se han cumplido?

En las primeras líneas de ésta memoria se mostraban unos objetivos. Ahora es el momento de recordarlos y de reflexionar si realmente se han llegado a cumplir o no. De una forma u otra se debe dar una justificación que muestre las razones por las cuales se hace la afirmación o negación de que un objetivo particular ha sido cumplido. Recordemos:

1. *Ofrecer una herramienta de gestión útil:* Éste objetivo trataba de conseguir crear una herramienta útil y que funcionase bien. Se puede decir sin problema alguno que sí se ha cumplido. El razonamiento es que el proyecto funciona; hace una gestión, da de alta a los alumnos y envía notificaciones. El proyecto realiza una labor más interesante que ir dando de alta a mano a cada alumno en cada servicio. Además la herramienta es sencilla, un panel gráfico, atractivo a la vista y con opciones mínimas que simplifican las labores a realizar. No posee interfaces muy avanzadas, con muchas opciones, ni dispone de un montón de funcionalidades que terminen siendo un auténtico caos. El programa es simple y hace lo que debe de hacer.

2. *Mejorar la eficiencia del profesorado:* Éste objetivo trataba de enfocar el ahorro de tiempo que se produce cuando el primer objetivo se ha cumplido de forma exitosa. Si el programa hace lo que debe de hacer, es obvio que el profesorado mejorará sus tiempo de trabajo porque tendrá más tiempo para dedicarlo a otros menesteres. Éste objetivo de momento no se puede dar por cumplido. La razón es que la evaluación de utilidad viene dada por el profesorado. Ellos son lo que debe valorar si realmente la herramienta les ahorra tiempo o no.

3. *Hacer un sistema intuitivo y sencillo de usar:* Aquí se trataba de hacer que el programa fuese tan intuitivo como hacer uso de un programa para un teléfono móvil. Interfaces claras, sencillas, con pocas opciones y todo claramente diferenciado. Éste objetivo se puede dar como cumplido. La razón es el test de usabilidad comentado anteriormente 6.2. En dicho test se ha podido ver la aceptación de los usuarios, por lo tanto se puede decir que al ser una experiencia positiva, las interfaces también lo son por lo que podemos decir que realmente el sistema es intuitivo y sencillo de usar. No obstante, sería conveniente volver a hacer el mismo test pero ésta vez exponiendo las interfaces finales para que el usuario pueda dar una evaluación más veraz y acorde con lo que es el programa actual.

4. *Satisfacción personal:* Aquí se intentaba plasmar la satisfacción personal que se producía al conseguir un hito en el cual se hacía funcionar una aplicación haciendo uso de distintas tecnologías e implantándola de forma exitosa en distintos entornos operativos. Éste objetivo se puede dar por cumplido. El razonamiento es que se ha conseguido aprender mucho de nuevos lenguajes de programación y se ha podido ahondar en los conocimientos que envuelven en la creación de aplicaciones de escritorio. Además se ha podido disfrutar de una cierta diversión cuando el proyecto no funcionaba ni en broma en ningún sistema operativo y se ha tenido que compilar a mano ciertas librerías para conseguir hacerlo funcionar. Por todo ello se puede decir que aunque a veces haya sido una auténtica pesadilla, la satisfacción final es más que merecedora.

8.4. Planificación y su cumplimiento

Llegamos a la parte en la que toca reflexionar qué ha pasado con la planificación y si ésta ha sido cumplida de forma completa o no.

En éste proyecto la planificación temporal no ha podido ser cumplida de forma completa. Las razones se van a comentar a continuación.

1. En varios apartados ha sido imposible cumplir con los tiempos establecidos. En algunos se ha hecho en menos tiempo, pero en muchos otros directamente el tiempo se ha superado en mucho más tiempo de lo que se tenia preparado. Se van a mostrar algunos de los paquetes de trabajo afectados por una duración excesiva de tiempo.

    a) *Creación de los prototipos de interfaces.* Éste apartado ha costado mucho más tiempo que del que se tenía pensado inicialmente. Diseñar interfaces, conseguir una ayuda por parte de un usuario final y realizar varias iteraciones ha costado unas 10 horas adicionales del tiempo que se tenía pensado.

    b) *Aprendizaje del lenguaje de programación "Python".* El tiempo para aprender el lenguaje en sí fue muy corto pero después, si se suma todo el tiempo adicional en investigación y consultas para implantar ciertas cosas, nos da un resultado de unas 8 horas más de lo que se tenía pensado.

    c) *Aprendizaje del lenguaje gráfico "QT".* Aquí se ha demorado al menos unas 6 horas más de lo deseado debido a la constancia de tener que revisar manuales y la documentación para conseguir hacer funcionar algunas cuestiones como los **User Roles** o los conectores a los botones.

    d) *Creación del DOP.* La creación de la memoria ha llevado mucho más tiempo que el establecido. Repasar todos los contenidos e incrustar todos los elementos más introducir todas las tablas ha llevado al menos 8 horas más de lo que se tenía pensado.

    e) *Diseño del modelo de dominio.* El diseño del modelo de domino de la parte cliente ha sido un cúmulo de cambios en más de una ocasión. El modelo ha recibido como unas 3 o 4 iteraciones para poder conseguir ser un modelo eficaz. Todo ésto, ha generado al menos unas 3 horas más de trabajo de lo que se tenía estimado.

f) *Implantación de los módulos de la parte cliente.* Éste apartado ha sufrido un duro revés en más de una ocasión. Se ha tenido que reimplementar muchas partes, se ha tenido que modificar algunos ficheros e incluso se ha tenido que unificar algunos gestores para hacer más sencillo la lectura del código. Si a todo ésto sumamos el tiempo invertido en corregir errores de compilación y poner el código bien legible, podemos decir que se ha excedido el tiempo en unas 18 horas más de lo que se tenía estimado.

g) *Pruebas de funcionamiento general.* Las pruebas del sistema han costado más tiempo del que se tenía estimado. En total se ha tardado 4 horas más, ésto ha sido debido a la complejidad de realizar pruebas de caja negra en algunos apartados.

2. Se ha tenido que replanificar el proyecto en dos ocasiones. Las razones para ésto han sido que por un lado no se tenía pleno conocimiento de cómo se iba a conectar las interfaces con "Python" y cómo se debían guardar todos los datos necesarios para luego mostrarlos de forma correcta. Por el otro lado, no se tuvo en cuenta que la implantación del proyecto en otros sistemas operativos fuese tan problemática. Además, se tuvo que aprender nuevas herramientas adicionales que en un principio no estaban pensadas, sobre todo a la hora de querer compilar el proyecto en otras plataformas.

   a) Se tuvo que aprender a hacer uso los métodos de compilación de los paquetes para "GNU/Linux".

   b) Se tuvo que aprender a manejar los comandos de compilación de *nmake* para "Windows".

   c) Se tuvo que aprender las distintas formas de empaquetamiento para poder generar binarios.

3. No se han podido completar todos los paquetes. No se ha podido generar la documentación específica al manual de usuario y al manual técnico para ayudar a los usuarios más inexpertos y enseñar al administrador del sistema a implantar de forma correcta todo el proyecto.

Con el incumplimiento de algunos de los tiempos y de la necesidad de replanificar el proyecto se puede decir que ha sido imposible cumplir con la planificación establecida por lo que los costes iniciales se han disparado un poco más de lo esperado. Aún así, el proyecto sigue siendo viable.

8.5. Riesgos y sus apariciones

Durante la consecución de éste proyecto sólo ha aparecido uno de los riesgos que se han documentado. En éste caso, ha sido el riesgo derivado a la implantación del proyecto en otros entornos operativos.

El plan de contingencia ha sido efectivo aunque obviamente se ha tenido que hacer bastante investigación y muchas pruebas para conseguir que todo fuese aceptable. Por lo que podemos decir que ha sido efectivo y el riesgo ha sido subsanado correctamente.

Otros riesgos que han aparecido y no han sido documentados han sido los siguientes:

- **Cambios en la BD cuando el proyecto está implantado.** En los últimos ciclos del proyecto, se han tenido que incluir algunos parámetros adicionales para mejorar la experiencia del programa. Dichos parámetros necesitaban nuevas entradas en la base de datos(los parámetros eran SHA y PATH para un script). Obviamente no ha habido que tener que replanificar el proyecto, pero podría haber sido un riesgo serio en caso de que hubiera afectado a la funcionalidad básica del sistema.
- **Cambios de la lógica de guardado de alguna clase.** Una de las cosas que se ha cambiado en el proyecto, ha sido la forma en la que se guardaba el historial dentro de la base de datos. En un principio se guardaban los identificadores de los elementos que debían ser registrados pero luego, al borrar esos elementos desaparecían también de la base de datos. Como una de las cosas interesantes del proyecto era tener un historial que perdurase sobre el tiempo, se tuvo que cambiar la forma de guardar los datos en la base de datos y además se tuvo que modificar varias funciones y varios tipos de datos. Un riesgo así resulta peligroso porque si no tiene un buen plan de contingencia es posible que al final se termine modificando una gran cantidad del proyecto. Por lo tanto, hubiera sido interesante haber tenido al menos unas pautas para medir el alcance y el impacto de una modificación de éstas características.

8.6. Líneas futuras

Como guinda final de ésta conclusión se va a mostrar una serie de líneas futuras que podrían mejorar las capacidades que ofrece el programa. Además se valorará si realmente merece la pena cada una de ellas y qué beneficios podría traer de cara a que la aplicación sea aún más perfecta.

1. **Cifrado de los datos.** Una cuestión muy interesante sería poder filtrar todos los datos que maneja el cliente en su aplicación. De ésta forma se podrían tener a salvo los datos personales de los alumnos en caso de que ocurriera algún percance. El programa generaría una clave de cifrado la primera que el usuario del sistema haría login y después éste requeriría al usuario algún tipo de identificación cada que vez que abriría el programa. Una idea, sería hacer uso de las tarjetas que tiene los profesores con su chip. La carga de trabajo sería bastante grande debido a que habría que hacer una incursión en el campo de la criptografía y su implantanción en Python. Además habría que estudiar el impacto a nivel de rendimiento del sistema para saber si realmente sería o no factible poder usarlo. En resumen, se necesita hacer mucho trabajo pero el resultado sería altamente beneficioso porque mejorarían las condiciones de seguridad del programa.
2. **Seguridad en dos pasos.** Otra idea bastante interesante sería una implantación de una seguridad en dos pasos. La ides que el la primera vez que un usuario se conecte a un sistema, éste le pida una contraseña adicional o le pida que haga uso de la tarjeta que tiene como profesor en la UPV/EHU para poder identificarse correctamente en el sistema. La implantación de éste método seria sencilla ya que sólo necesario conectar la aplicación a un servidor de autentificación externo y validar la respuesta que se de. Los beneficios que

traería ésta mejora serían muy buenos de cara a evitar que cualquier usuario pudiese entrar al sistema porque ha conseguido de un modo u otro los accesos de un profesor.

3. **JSON Schemas.** Los JSON Schemas son modelos de datos en JSON que fuerzan a éste a tener una estructura de datos determinada. Cuando se programa un JSON Schema se decide qué formato tiene que tener el diccionario y por lo tanto si éste recibe un JSON que no cumple ni con los campos especificados ni con una serie de datos pre-introducidos, el JSON Schema invalida los datos y no los usa. Ésto es interesante de cara a mejorar la eficiencia de los "sockets". Si *ServerReceiver* recibe un JSON que no contiene la estructura programa en el JSON Schema éste lo invalida y previene la obtención no autorizada de datos en el servidor. La implantación de ésta funcionalidad no supone una gran carga de trabajo porque únicamente se tienen que definir ciertos JSON Schemas y modificar cada envío del cliente para que coincidan con estos. Los beneficios serían muy considerables en cuestiones de seguridad por lo que merecería la pena implantarlo.

4. **Bandejas de mensajes.** Una idea de línea futura sería la implantación de bandejas para enviar y recibir mensajes directos. Ésto sería interesante, por ejemplo, si se querría enviar algún mensaje para avisar a algún usuario de que se ha donado algún tag o para pedir una ayuda específica. También sería interesante para poder recibir mensaje de parte del administrador del sistema, o enviar errores en caso de que algún script no funcione de manera adecuada. La carga de trabajo que produciría sería bastante grande porque habría que modificar las interfaces y se tendrían que crear nuevas. Además seria necesario modificar la base de datos y los modelos de dominio para poder hacer hueco a la nueva funcionalidad. Al final, el beneficio final no sería tan grande y realmente no terminaría mereciendo tanto la pena el esfuerzo ya que, por ejemplo, se puede hacer uso de los mails para notificar a los usuarios de ciertas cuestiones.

5. **Mejorar las interfaces.** Una de las cosas que sería necesario mejorar serían las interfaces en "QT" para que éstas brindasen al ususario una mejor experiencia. Para ello se tendría que estudiar y ahondar más en el uso y manejo de "QT" y su conexión con "Python" por lo que la carga de trabajo sería muy considerable. Ahora bien, el resultado final sería satisfactorio porque se mejoraría la experiencia de cara al usuario con unas interfaces más claras y avanzadas.

6. **Mejorar el filtrado en los historiales**. El filtrado en los historiales es bastante pobre y simple. Es cierto que realiza bien su trabajo pero termina siendo demasiado simplista. Una buena forma de mejorar ésto sería conectando la interfaz de "QT" directamente a la base de datos para que pudiera hacer búsquedas dinámicas. El usuario simplemente iría introduciendo el texto y la tabla se iría actualizando de forma dinámica mostrando los resultados deseados. La carga de trabajo de ésta funcionalidad sería media ya que lo único que haría falta es saber implantar la clase de "QsqlQuery" y hacer un pequeño refresco dinámico de la tabla. Los beneficios serían medios pero aún así sería visualmente interesante para el usuario.

Bibliografía

[1] Leando Alegsa. Definición de latex. http://www.alegsa.com.ar/Dic/latex.php, 2010. Accedido 09-09-2015.

[2] Mark Amery. How to get string objects instead of unicode ones from json in python. http://stackoverflow.com/questions/956867/how-to-get-string-objects-instead-of-unicode-ones-from-json-in-python, 2012. Accedido 12-05-2015.

[3] Varios Autores. Capa de conexión segura ssl. https://www.digicert.com/es/ssl.htm, 2003. Accedido 09-09-2015.

[4] Varios Autores. Programación de sockets en c de unix/linux. http://www.chuidiang.com/clinux/sockets/sockets_simp.php#sockets, 2007. Accedido 09-09-2015.

[5] Varios Autores. Descripción general de unicode. http://docs.oracle.com/cd/E26921_01/html/E27143/glmgn.html, 2012. Accedido 09-09-2015.

[6] Varios Autores. ¿qué es una librería o biblioteca en informática? http://desarrollomovilmultiplataforma.blogspot.com.es/2012/08/aspectos-teoricos-libreria-biblioteca.html, 2012. Accedido 09-09-2015.

[7] Varios Autores. Qué es un kit de desarrollo de software (sdk)? http://www.4rsoluciones.com/que-es-un-kit-de-desarrollo-de-software-sdk/, 2013. Accedido 09-09-2015.

[8] Varios Autores. Building pyqt5 for python2.7 on a clean ubuntu 13.10 build machine. https://plashless.wordpress.com/2014/03/26/building-pyqt5-for-python2-7-on-a-clean-ubuntu-13-10-build-machine/, 2014. Accedido 11-09-2015.

[9] Varios Autores. Mysql python tutorial. http://zetcode.com/db/mysqlpython/, 2014. Accedido 09-04-2015.

[10] Varios Autores. ¿qué es un diagrama de gantt y para qué sirve? http://www.obs-edu.com/blog-project-management/diagramas-de-gantt/que-es-un-diagrama-de-gantt-y-para-que-sirve/, 2014. Accedido 09-09-2015.

[11] Varios Autores. Background of qt. http://www.qt.io/about-us/, 2015. Accedido 09-09-2015.

[12] Varios Autores. Definición de mysql. https://www.techopedia.com/definition/3498/mysql, 2015. Accedido 28-10-2015.

[13] Varios Autores. Kde. https://wiki.archlinux.org/index.php/KDE, 2015. Accedido 09-09-2015.

[14] Varios Autores. Python documentation. https://docs.python.org/2/library/socketserver.html, 2015. Accedido 20-05-2015.

[15] Varios Autores. Qt documentation. http://doc.qt.io/, 2015. Accedido 5-05-2015.

[16] Varios Autores. System usability scale (sus). http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html, 2015. Accedido 04-10-2015.

[17] Varios Autores. What is unix. http://www.unix.org/what_is_unix.html, 2015. Accedido 29-10-2015.

[18] Varios Autores. Window and dialog widgets. http://doc.qt.io/qt-4.8/application-windows.html, 2015. Accedido 20-06-2015.

[19] Varios Autores. ¿qué es debian? https://www.debian.org/releases/jessie/mips/ch01s01.html.es, 2015. Accedido 28-10-2015.

[20] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. June 2008.

[21] Vangie Beal. Sql - structured query language. http://www.webopedia.com/TERM/S/SQL.html, 2015. Accedido 09-09-2015.

[22] Abstract Factory Blog. Pyqt5.1.1 for python 2.7 on windows 7+. http://blog.abstractfactory.io/pyqt5-1-1-for-python-2-7/, 2014. Accedido 15-09-2015.

[23] Benoit Chabord. Comparing python/mysql drivers. http://mypysql.blogspot.com.es/2013/05/comparaison-of-pythonmysql-connectors.html, 2013. Accedido 03-02-2015.

[24] Manel Clos. adding ssl support to socketserver. http://stackoverflow.com/questions/8582766/adding-ssl-support-to-socketserver, 2013. Accedido 15-06-2015.

[25] PhD Cyrille Rossant. Create a standalone windows installer for your python application. http://cyrille.rossant.net/create-a-standalone-windows-installer-for-your-python-application/, 2013. Accedido 13-09-2015.

[26] Laboratorio Nacional de Calidad del Software de INTECO. Ingeniería del software: Metodología y ciclos de vida. https://www.incibe.es/file/N85W1ZWFHifRgUc_oY8_Xg, 2009. Accedido 31-10-2015.

[27] Raúl González Duque. *Python para todos*. Autoedición, first edition, 2010.

[28] Eljach. Cómo funciona sql injection, seguro eres vulnerable. http://www.cristalab.com/tutoriales/como-funciona-sql-injection-seguro-eres-vulnerable-c113268l/, 2014. Accedido 09-09-2015.

[29] Free Software Foundation. Gnu general public license. http://www.gnu.org/licenses/gpl-3.0.en.html, 2007. Accedido 09-09-2015.

[30] Piotr Maliński. Introduction to pyqt4. http://www.rkblog.rk.edu.pl/w/p/introduction-pyqt4/, 2008. Accedido 09-05-2015.

[31] Esteban Manchado. Tutorial de programación en shell. http://www.demiurgo.org/doc/shell/shell-2.html, 1998. Accedido 28-10-2015.

[32] Tim Peters. Pep 20 – the zen of python. https://www.python.org/dev/peps/pep-0020/, 2004. Accedido 08-09-2015.

[33] Zed A. Shaw. Learn python the hard way. http://learnpythonthehardway.org/book, 2013. Accedido 20-11-2014.

[34] Techopedia. Definición de c++. https://www.techopedia.com/definition/26184/c-programming-language, 2015. Accedido 28-10-2015.

Anexos

Apéndice A

Anexo I: Casos de uso extendido

En éste primer anexo se hará una extensión detallada de los casos de uso mencionados en la documentación. Para ello, se mostrará de forma explícita toda la información relativa al funcionamiento del caso de uso y a los roles que participan en él. También, se mostrará qué tipo de decisiones lógicas suceden y cómo se gestionan.

El anexo consta de dos apartados, uno englobado a la parte cliente de la aplicación y otro englobado a la parte servidor.

A.1. Parte cliente

A continuación se muestran los casos de uso detallados para la parte cliente de la aplicación.

A.1.1. Iniciar sesión

Nombre: Iniciar sesión
Descripción: Permite al rol anónimo tener acceso al sistema
Actores: Anónimo
Precondiciones: Ninguna
Requisitos no funcionales: Si el usuario tuviera alguna lista de alumnos ya introducida en el sistema, se le mostaría la primera de ellas con los datos relacionados a la misma. En caso contrario, no se vería ninguna lista seleccionada.
Flujo de eventos:

1. El anónimo, desde la interfaz de login, introduce un usuario, una contraseña y un servidor válido A.1.

a) Si el usuario o contraseña son incorrectos, se le muestra una pantalla de error.

b) Si el servidor introducido no es correcto o no responde se le muestra un error.

2. Se le muestra la pantalla principal del sistema con todas las opciones disponibles A.2

Postcondiciones: Ninguna
Comentarios: Es necesario que el servidor destino introducido exista y tenga la parte servidor de la aplicación.

A.1.2. Crear grupo nuevo

Nombre: Crear grupo nuevo
Descripción: Permite al usuario crear un nuevo grupo en el sistema
Actores: Usuario
Precondiciones: Ninguna
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El usuario pulsa el botón “Archivo” -> “Crear nuevo grupo” de la pantalla principal

2. Se le mostrará una interfaz que le permitirá realizar una serie de acciones para añadir alumnos al nuevo grupo o la posibilidad de importar un fichero con los alumnos deseados.A.3

3. Si el usuario quiere:

   a) Puede importar los alumnos desde un fichero (ver subcaso de uso importar alumnos). A.1.10

   b) Puede añadir manualmente un nuevo alumno(ver subcaso Añadir alumno). A.1.11

Postcondiciones: Ninguna
Comentarios: Ninguno

A.1.3. Seleccionar grupo

Nombre: Seleccionar grupo
Descripción: Permite al Usuario poder seleccionar uno de los grupos que haya creado anteriormente.
Actores: Usuario
Precondiciones: Que como mínimo el Usuario haya creado previamente un grupo.
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El usuario desde la interfaz principal del sistema, pincha el desplegable que contiene la lista de los grupos y elige un grupo distinto al actual.

2. Al elegir un grupo se actualiza de forma automática la lista de los alumnos que lo contiene.

Postcondiciones: Se muestra en pantalla los nombres de los alumnos que conforman el grupo seleccionado.
Comentarios: Ninguno

A.1.4. Gestionar scripts

Nombre: Gestionar scripts
Descripción: Permite al Usuario del sistema modificar los scripts que contiene el grupo seleccionado.
Actores: Usuario
Precondiciones: Tener seleccionado un grupo.
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El usuario desde la interfaz principal del sistema, pulsa el botón “Gestionar Scripts”.

2. Se le muestra una interfaz por la cual podrá realizar una serie de gestiones para asignar los scripts o tags deseados al grupo actualmente seleccionado. A.4

Postcondiciones: Si el grupo ya tenía algún script o tag asignado previamente, se le mostrarán cuáles están aplicados y cuáles tiene por ende disponibles.

Comentarios: Si el usuario no había aplicado script alguno, la columna de la izquierda estará completamente vacía. Pero en cambio, si el grupo ya había sido modificado previamente, aparecerán los scripts o tags ya aplicados y sólo podrá gestionar los que estén disponibles actualmente.

A.1.5. Gestionar mis tags

Nombre: Gestionar mis tags.

Descripción: Permite al Usuario del sistema realizar una gestión de sus tags en el sistema.

Actores: Usuario

Precondiciones: Ninguna

Requisitos no funcionales: Ninguno

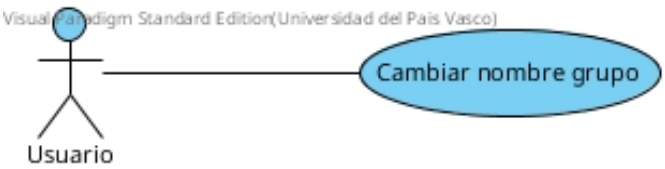
Flujo de eventos:

1. El usuario desde la interfaz principal del sistema, pulsa el botón "Gestionar mis Tags".
2. Se le muestra una interfaz por la cual podrá realizar una serie de gestiones para crear, modificar o eliminar un tag. A.6
3. Si el usuario quiere:
   a) Puede crear un nuevo tag (Ver subcaso "Crear tag"). A.1.12
   b) Puede Modificar un tag (Ver subcaso "Modificar tag"). A.1.13

Postcondiciones: Ninguna.

Comentarios: Ninguno.

A.1.6. Cambiar nombre grupo



Nombre: Cambiar nombre grupo
Descripción: Da la opción de poder cambiar el nombre de un grupo.
Actores: Usuario
Precondiciones: Tener seleccionado un grupo.
Requisitos no funcionales: Ninguno.
Flujo de eventos:

1. El usuario desde la interfaz principal del sistema pulsa el botón "Cambiar nombre".
2. Se le muestra una interfaz por la cual podrá cambiar el nombre del grupo. A.5
3. El usuario introduce el nuevo nombre del grupo:
   a) Si el usuario pulsa el botón "Cambiar":
      1) Si el nombre del grupo introducido ya existe se le muestra una advertencia indicándole que el nombre ya existe en el sistema. A.11
      2) Si el nombre es correcto se le muestra una advertencia indicándole el cambio satisfactorio del nombre del grupo.
   b) Si el usuario pulsa el botón "Cancelar" volverá a la interfaz principal.

Postcondiciones: El nuevo nombre del grupo, aparecerá actualizado en la lista de selección de grupos de la interfaz principal.
Comentarios: Ninguno.

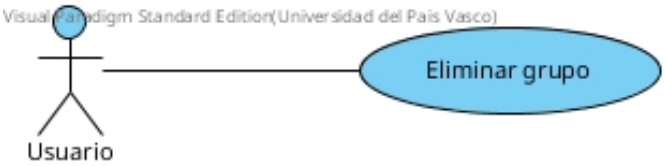
A.1.7. Ver historial cambios

Nombre: Ver historial cambios.
Descripción: Muestra al usuario los cambios realizados en el sistema para tener un feedback de las acciones que haya realizado.
Actores: Usuario
Precondiciones: Ninguna.
Requisitos no funcionales: Ninguno.
Flujo de eventos:

1. El usuario desde la interfaz principal del sistema pulsa el botón "Herramientas" -> "Ver historial de cambios".
2. Se le muestra una interfaz en la cual se muestra información relativa a los cambios realizados a cada alumno, mostrando información de interés. A.7

Postcondiciones: Ninguna.
Comentarios: Si el usuario nunca ha cambiado nada en el sistema o todos sus cambios fueron borrados, no se mostrará información alguna.

A.1.8. Eliminar grupo



Nombre: Eliminar grupo.
Descripción: Permite al usuario borrar un grupo previamente creado.
Actores: Usuario
Precondiciones: Tener al menos seleccionado un grupo.
Requisitos no funcionales: Ninguno.
Flujo de eventos:

1. El usuario desde la interfaz principal del sistema pulsa el botón "Eliminar".
2. Se muestra un aviso, pidiendo una confirmación sobre la acción a realizar. A.8
   a) Si el usuario pulsa el botón "Aceptar" el grupo será eliminado del sistema y se le mostrará una pantalla de confirmación.
   b) Si pulsa el botón "Cancelar" se volverá a la pantalla principal del sistema.

Postcondiciones: Al eliminar un grupo el sistema, de forma automática, eliminará de los alumnos de dicho grupo, todos los scripts/tags que tuvieran aplicados.
Comentarios: Si un alumno registrado en el sistema se queda sin grupo alguno(huérfano), entonces éste será borrado de la aplicación.

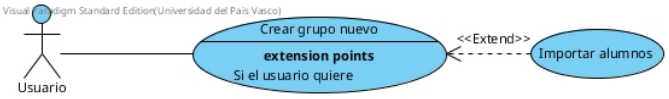
A.1.9. Cerrar sesión

Nombre: Cerrar sesión.
Descripción: Consiste en la salida segura del sistema al usuario autentificado.
Actores: Usuario
Precondiciones: Ninguna.
Requisitos no funcionales: Ninguno.
Flujo de eventos:

1. El usuario desde la interfaz principal del sistema pulsa el botón "Cerrar sesión".
2. El usuario sale del sistema y se le muestra la pantalla de login.

Postcondiciones: Ninguna.
Comentarios: Ninguna.

A.1.10. SUBCASO: Importar alumnos

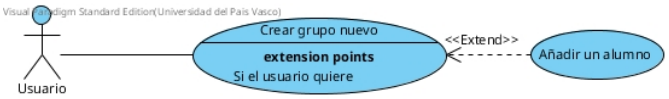


Nombre: Importar alumnos.
Descripción: Consiste en la adición de usuarios a través de un fichero con extensión *.txt.
Actores: Usuario
Precondiciones: Caso de uso "Crear grupo nuevo".
Requisitos no funcionales: Ninguno.
Flujo de eventos:

1. El usuario desde la interfaz de "Crear grupo nuevo" pulsa el botón "..."
2. Se le abre una interfaz que le permite añadir un fichero con extensión *.txt
   a) Si el archivo que intenta introducir es un archivo *.txt pero contiene entradas inválidas. El sistema le devolverá un error indicándole que el archivo no es correcto.
   b) Si el archivo es correcto y contiene entradas válidas, se le mostrarán aquellos alumnos que tendría el fichero.

Postcondiciones: Se mostrará en la lista de usuarios de la interfaz de "Crear grupo nuevo" la lista de los alumnos importados filtrando únicamente aquellos que hayan sido detectados como repetidos.
Comentarios: Ninguna.

A.1.11. SUBCASO:Añadir un alumno



Nombre: Añadir un alumno.
Descripción: Permite al usuario de manera manual añadir un alumno en la tabla de creación de un nuevo grupo.
Actores: Usuario
Precondiciones: Caso de uso "Crear grupo nuevo".
Requisitos no funcionales: Ninguno.
Flujo de eventos:

1. El usuario desde la interfaz de "Crear grupo nuevo" pulsa el botón "+"
2. Se le abre una interfaz que le permite introducir manualmente los datos de un nuevo alumno.

3. El usuario introduce los datos necesarios y pulsa aceptar.

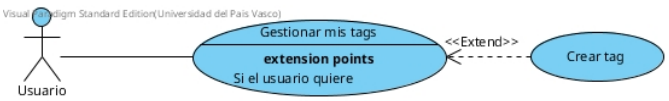
   a) Si los datos de un nuevo alumno son incorrectos, el sistema le envía un error indicándole cuál ha sido el fallo y en qué tipo de dato se halla.

   b) Si los datos del nuevo alumno son correctos se le muestra una ventana de confirmación y al aceptarla se le devuelve a la interfaz de crear grupo nuevo.

   c) Si el Dni introducido coincide con alguno de los alumnos que ya habían sido introducidos previamente durante la creación del grupo, el sistema le devuelve un error indicándole la duplicidad.

Postcondiciones: Ninguna.
Comentarios: Ninguno.

A.1.12. SUBCASO: Crear tag



Nombre: Crear tag.
Descripción: Permite al Usuario del sistema realizar una gestión de sus tags en el sistema.
Actores: Usuario
Precondiciones: Gestionar mis tags.
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El usuario desde la interfaz de gestionar mis tags pulsa “nuevo”.

2. Se le muestra una interfaz similar a la de Gestionar Scripts que le permite seleccionar aquellos scripts para crear un nuevo tag.

3. El usuario realiza una serie de acciones sobre la interfaz y pulsa en “Aplicar”:

   a) Si no introduce ningún script, el sistema le devolverá un error indicándole que debe al menos introducir un script.

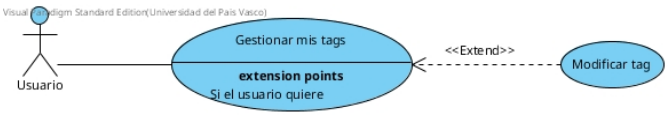
   b) Si no introduce ninguno de los datos asociados al nombre del tag y su descripción, se le devolverá un error indicándole la falta de los datos.

   c) Si introduce el nombre de un tag que ya existe, el sistema le devuelve un error indicándole que el nombre utilizado ya existe en otro tag.

   d) Si introduce correctamente todos los datos, el sistema le devolverá una ventana de confirmación y cerrará la interfaz.

Postcondiciones: Ninguna.
Comentarios: Ninguno.

A.1.13. SUBCASO: Modificar tag



Nombre: Modificar tag.
Descripción: Permite al Usuario del sistema realizar modificaciones en los tags creados con anterioridad.
Actores: Usuario
Precondiciones: Gestionar mis tags.
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El usuario desde la interfaz de gestionar mis tags pulsa "modificar tag".
2. Se le muestra una interfaz similar a la de Gestionar Scripts que le permite seleccionar aquellos scripts para modificar el tag actual
3. El usuario realiza una serie de acciones sobre la interfaz y pulsa en "Aplicar":
   a) Si deja vacía la columna "en el tag", el sistema le devuelve un error indicándole que debe al menos introducir algún script.
   b) Si olvida modificar de forma correcta los datos asociados con el tag, el sistema le avisa y le recuerda qué datos no son correctos.
   c) Si intenta modificar el tag, introduciendo un nombre de otro ya existente, el sistema le devuelve un error indicándole que el nombre ya existe.
   d) Si decide donar el tag a otro usuario y éste tiene un tag con el mismo nombre, el sistema devolverá un error indicando que el usuario destino tiene un tag ya asociado con el mismo nombre.
   e) Si introduce correctamente todos los datos, el sistema le devolverá una ventana de confirmación y cerrará la interfaz.

Postcondiciones: Ninguna.
Comentarios: Ninguno.

A.1.14. Imágenes

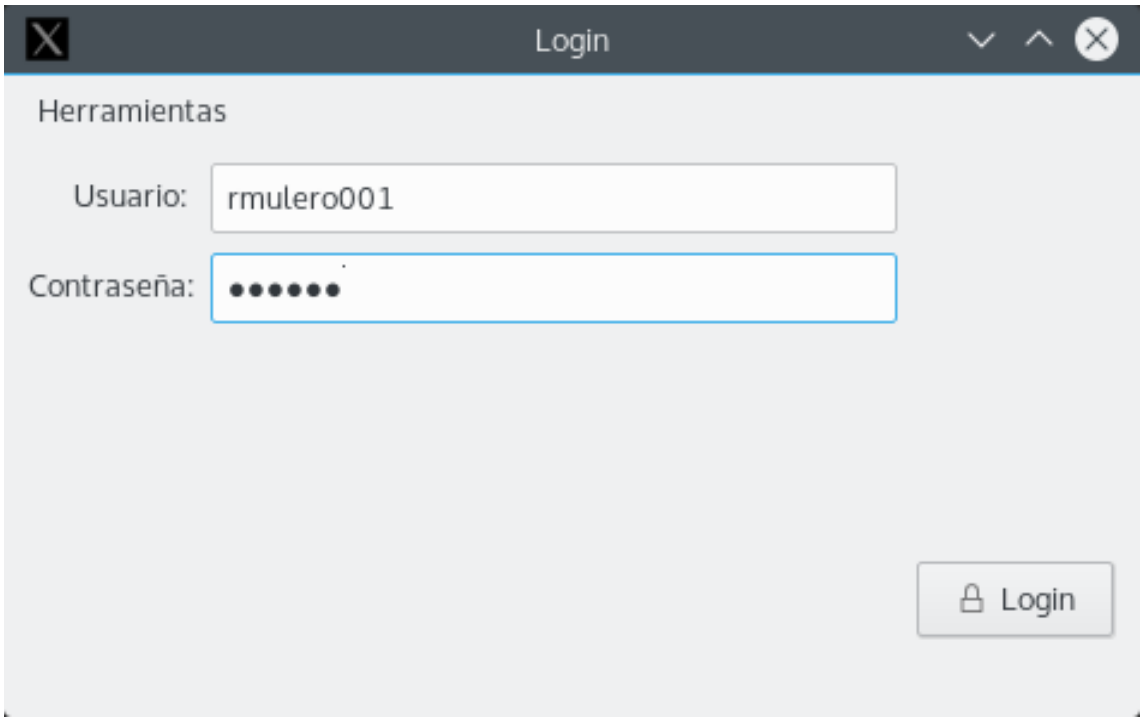


Figura A.1: Pantalla de identificación del sistema.

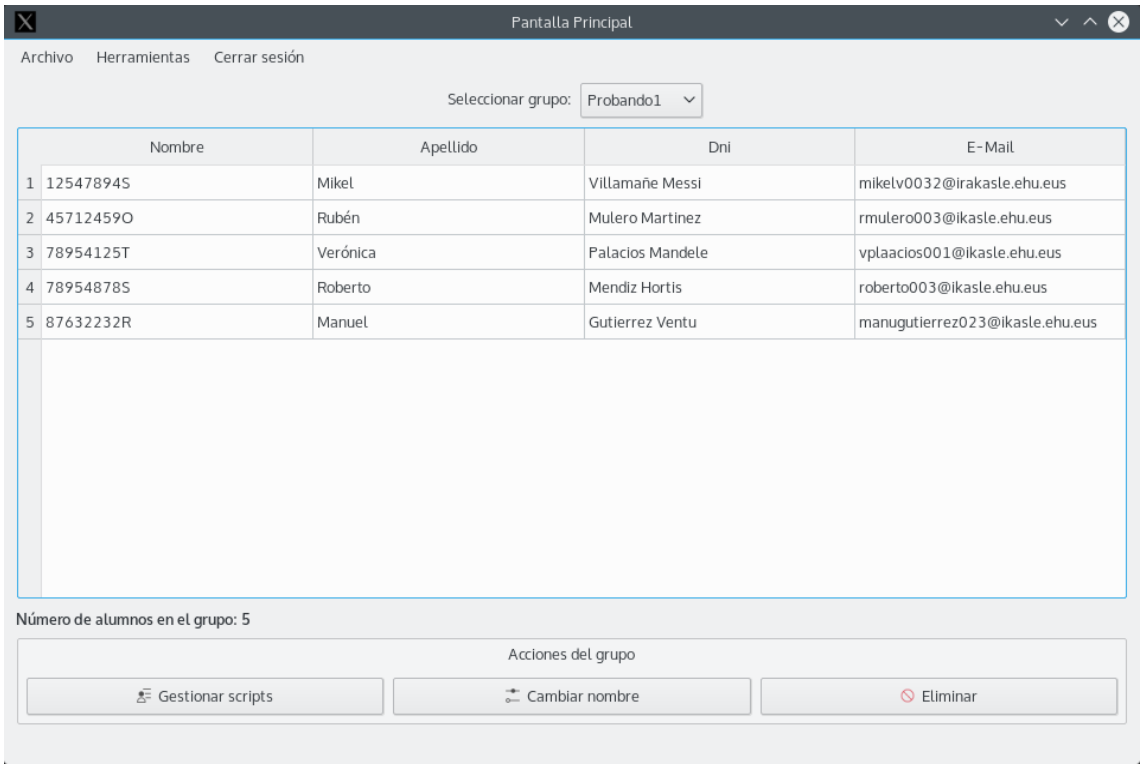


Figura A.2: Pantalla principal del sistema.

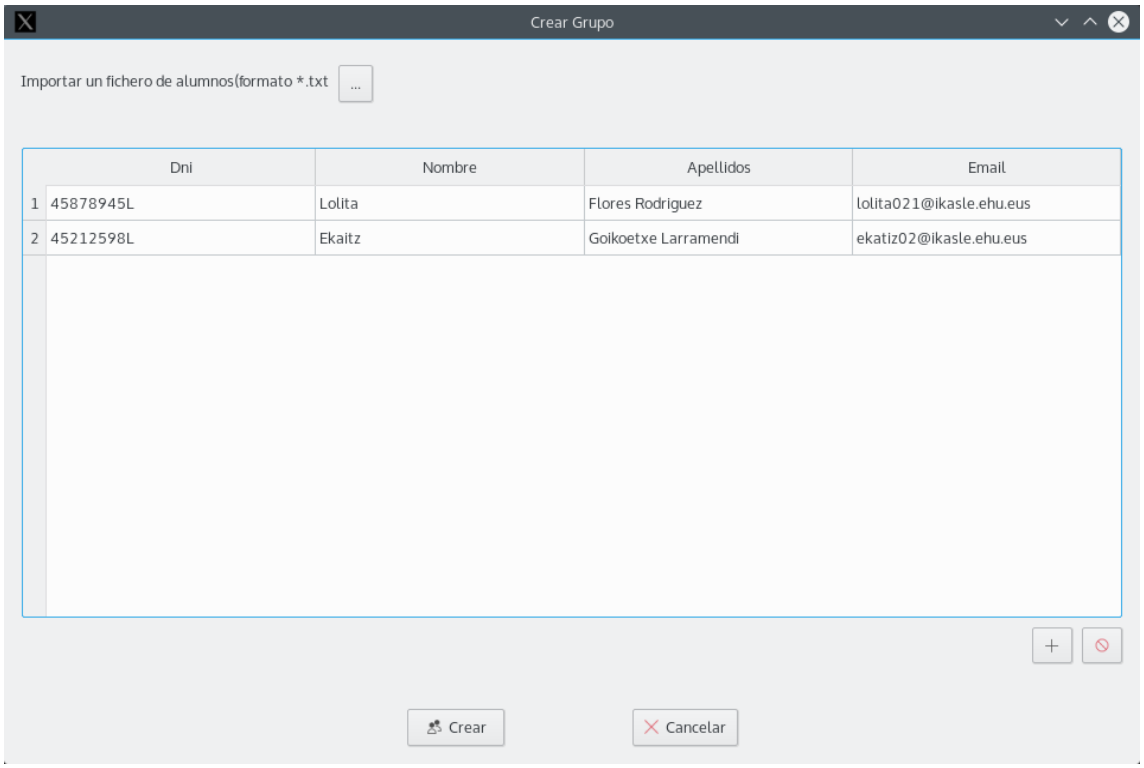


Figura A.3: Pantalla de creación de un nuevo grupo.

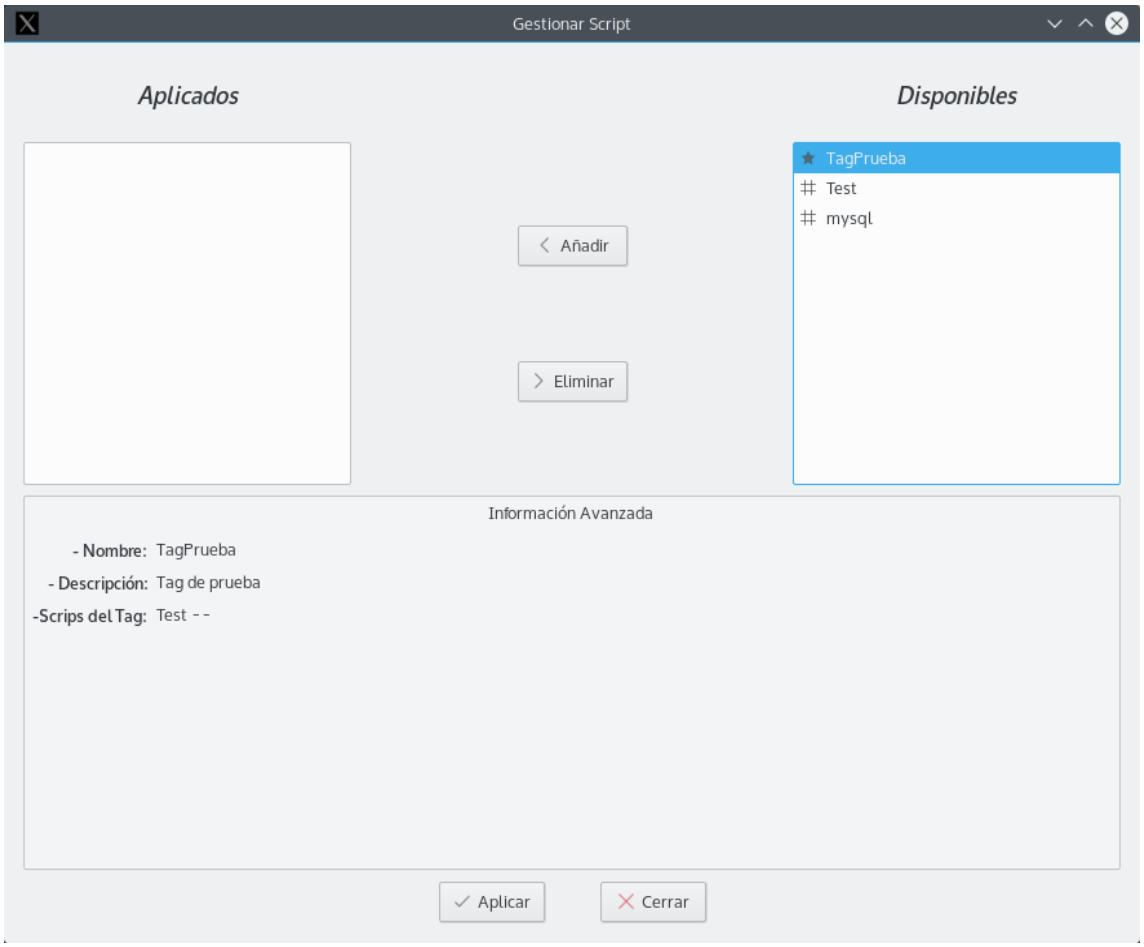


Figura A.4: Pantalla de gestión de Scripts/Tags de un grupo.

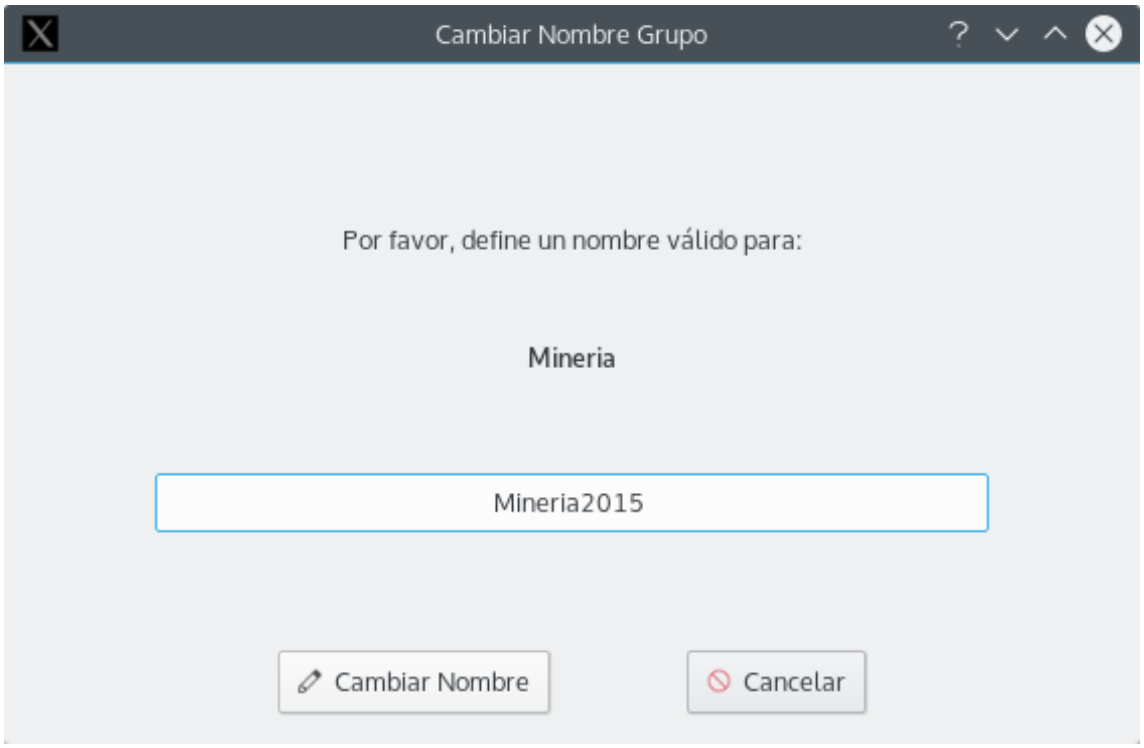


Figura A.5: Interfaz para el cambio del nombre de un grupo.

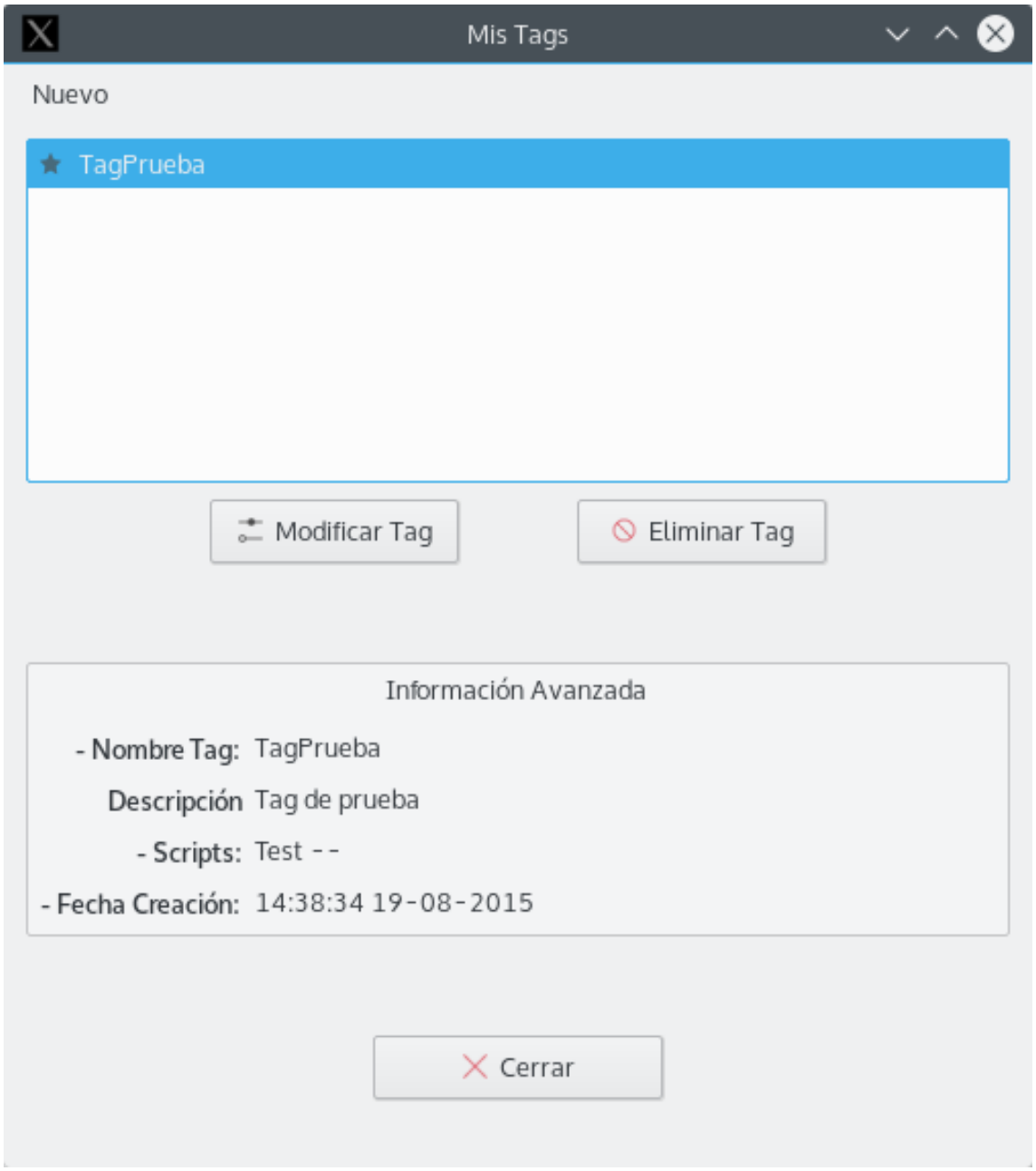


Figura A.6: Interfaz de gestión de los Tags del usuario actual.

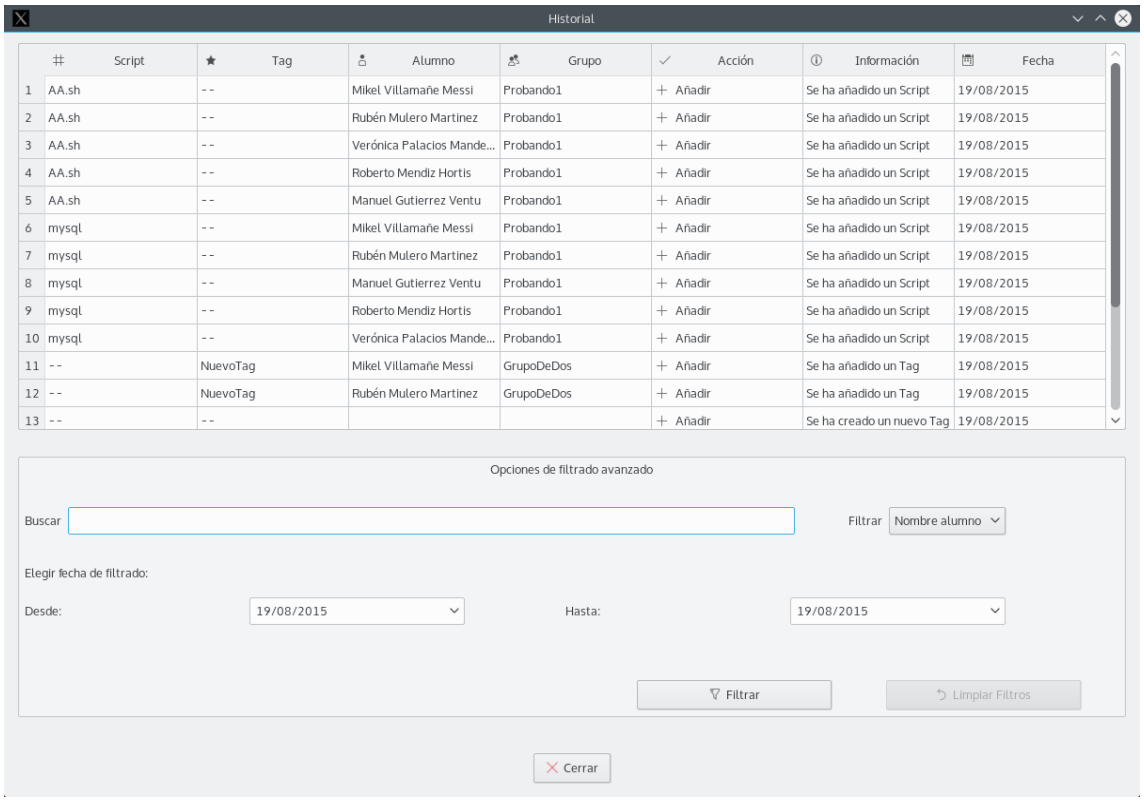


Figura A.7: Interfaz del historial de los cambios realizados.

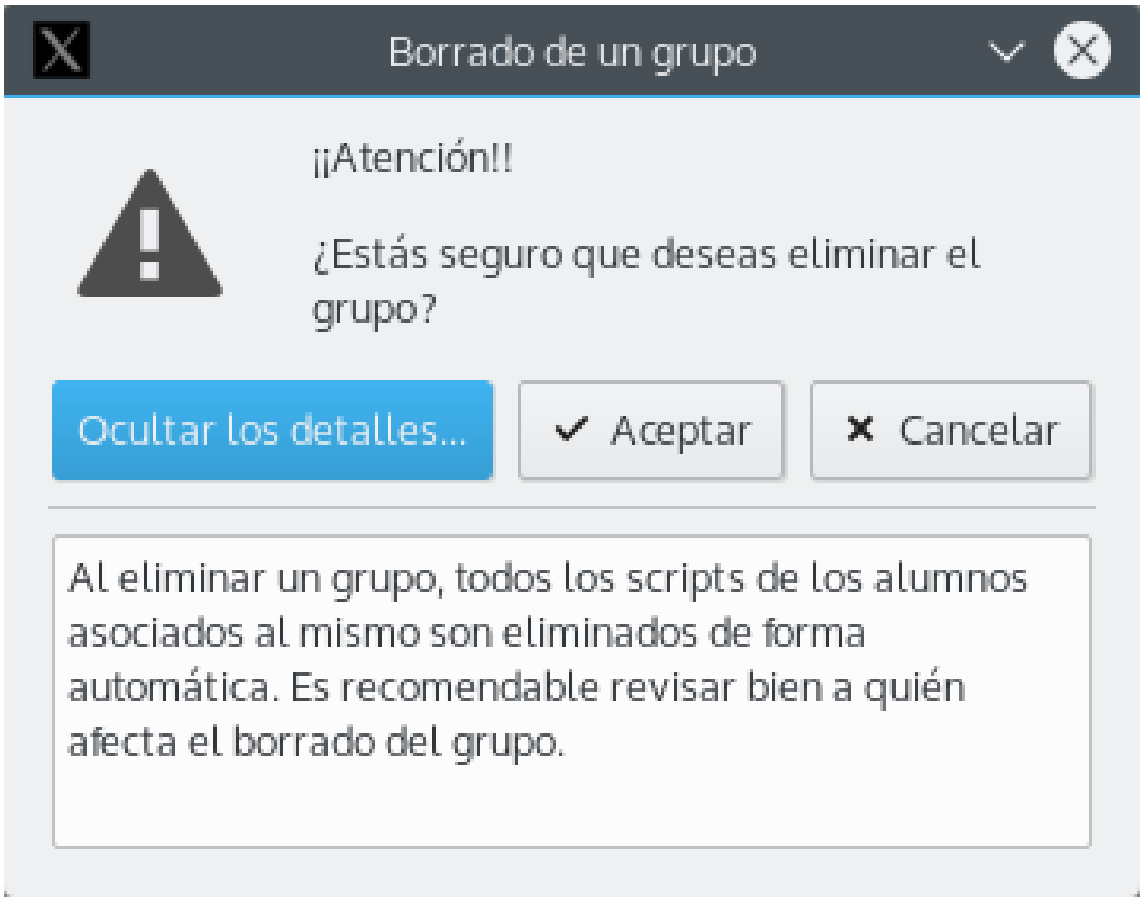


Figura A.8: Pantalla de aviso de eliminación de un grupo.

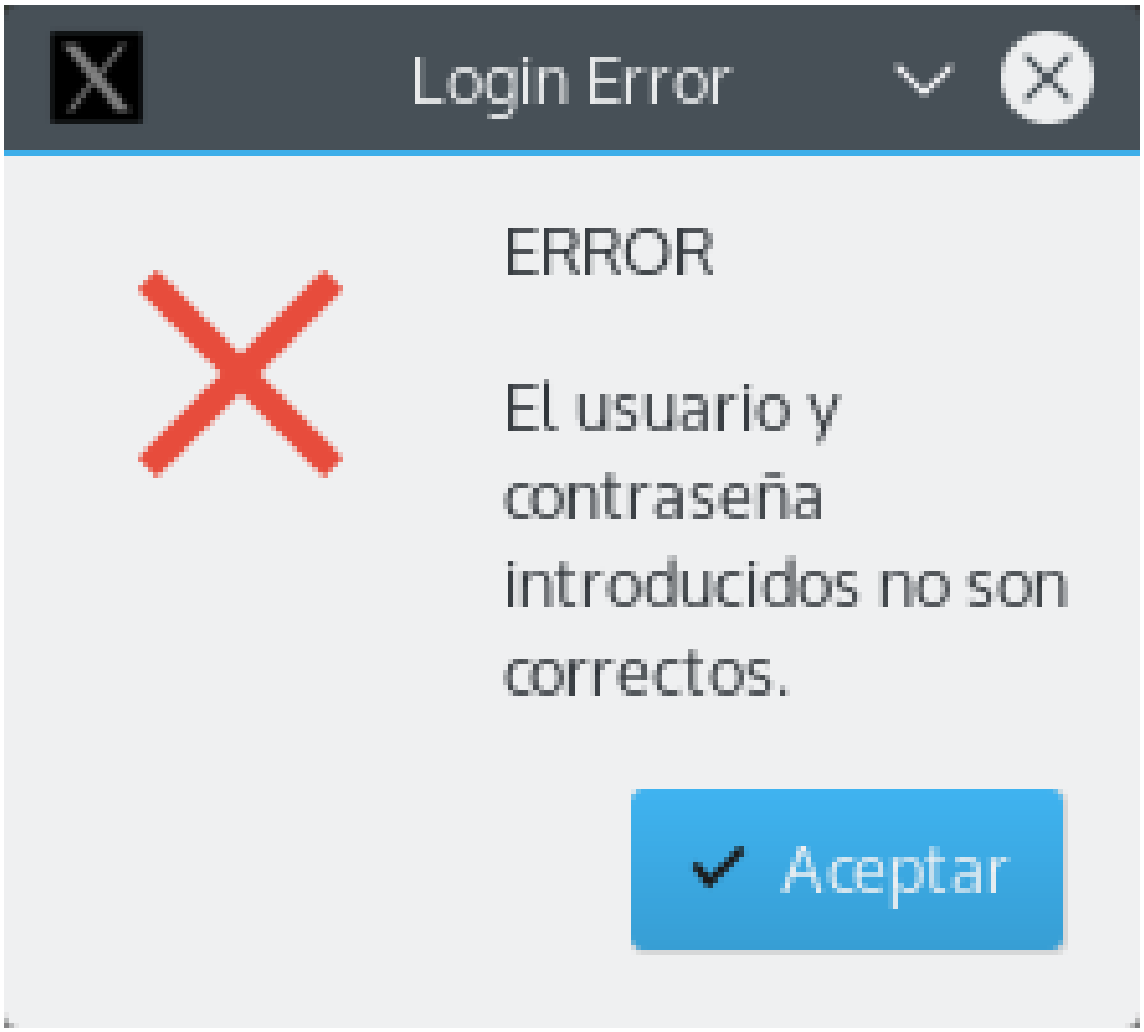


Figura A.9: Pantalla de error de credenciales incorrectas.

Figura A.10: Pantalla de error de servidor no encontrado.

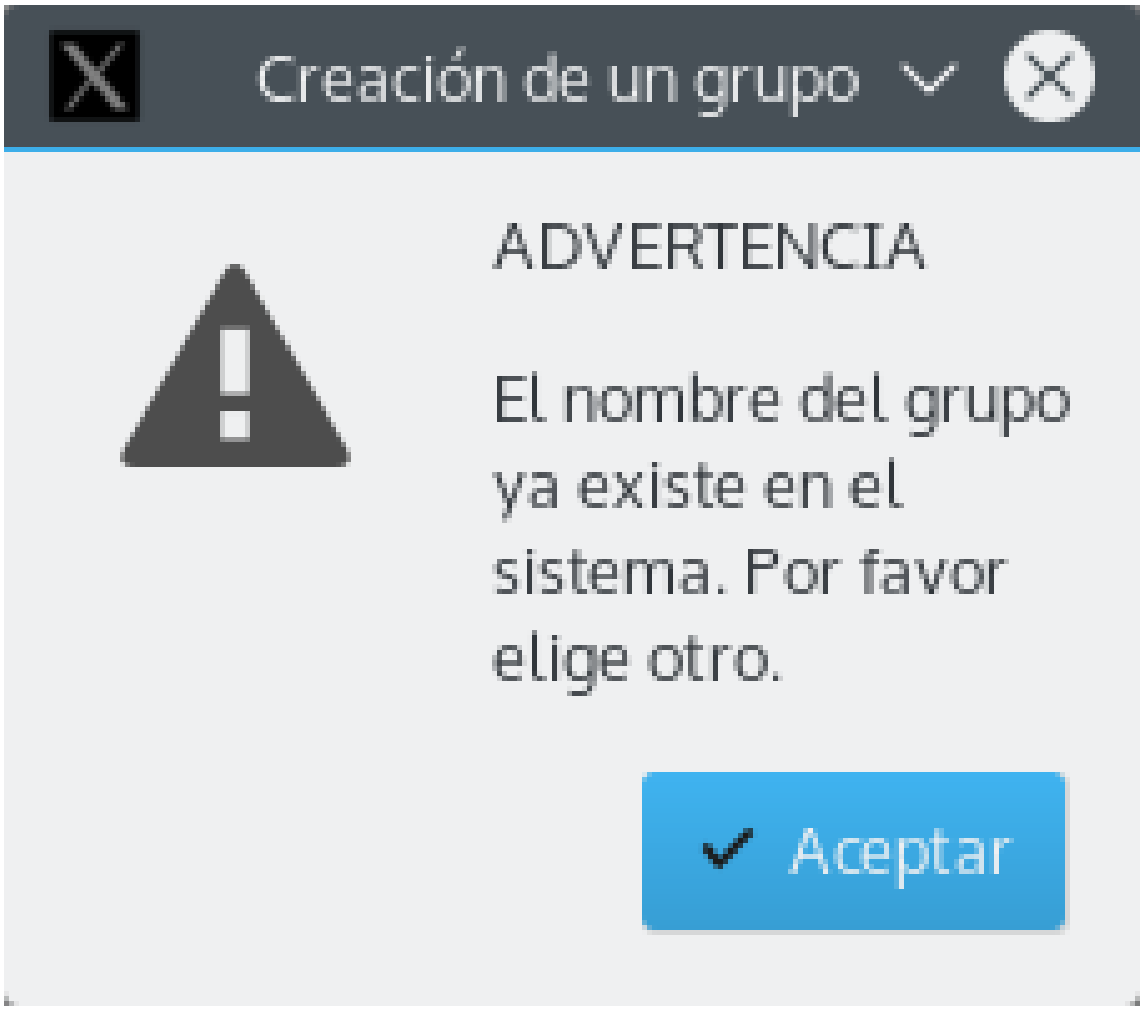


Figura A.11: Interfaz de aviso de nombre de grupo ya existente en el sistema al crear/modificar el nombre de un grupo.

A.2. Parte servidor

A continuación se muestran los casos de uso detallados para la parte servidor de la aplicación.

A.2.1. Gestionar Usuarios del sistema

Nombre: Gestionar Usuarios del sistema.
Descripción: Permite al administrador del sistema hacer una gestión de los usuarios que pueden acceder al sistema.

Actores: Administrador
Precondiciones: Ninguna
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El administrador desde su interfaz principal pulsa el botón "Gestionar usuarios del sistema". A.12

2. Se le mostrará una interfaz que le permitirá administrar los usuarios que quiera en el sistema. A.13

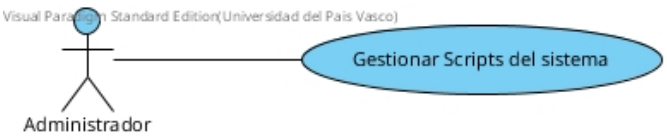
   a) Si el administrador pulsa sobre el botón "añadir usuario", se le mostrará una intefaz que le permitirá añadir los datos personales de un nuevo usuario y generarle (o no) una nueva contraseña.

   b) Si el administrador pulsa sobre el botón "modificar usuario", se le mostrará una interfaz que le permitirá modificar los datos personales del usuario.

   c) Si el administrador pulsa sobre el botón "eliminar usuario" (ver subcaso eliminar usuario). A.2.3

Postcondiciones: Ninguna
Comentarios: Cuando el administrador modifica los datos del usuario, si éste decide cambiar el usuario o contraseña de acceso al sistema, se le enviará de forma automática un e-mail para notificarlo.

A.2.2. Gestionar Scripts del sistema



Nombre: Gestionar Scripts del sistema.
Descripción: Permite al administrador realizar una gestión de los scripts del sistema.
Actores: Administrador
Precondiciones: Ninguna
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El administrador desde su interfaz principal pulsa el botón "Gestionar scripts del sistema". A.12

2. Se le mostrará una interfaz que le permitirá administrar todos los scripts que están registrados en el sistema actualmente. A.14

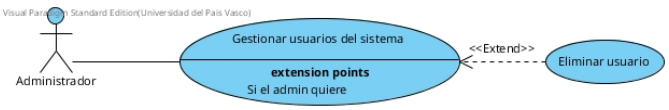
   a) Si el administrador pulsa sobre "añadir script", se le mostrará una interfaz que le permitirá introducir los datos para poder añadir un nuevo script en el sistema, indicando si éste debe estar activo o no.

b) Si el administrador pulsa sobre un script y luego pulsa el botón "modificar script", se le mostrará una interfaz que le permitirá cambiar el nombre y la descripción de un script.

c) Si el administrador pulsa sobre un script y luego pulsa el botón "eliminar script" (ver subcaso de uso Eliminar script). A.2.4

Postcondiciones: Ninguna
Comentarios: Al modificar un script, el administrador no tiene permisos para poder modificar el path donde está contenido el script. Ésto es así para evitar inconsistencias con los usuarios que ya tengan aplicado el script a modificar.

A.2.3. SUBCASO: Eliminar usuario

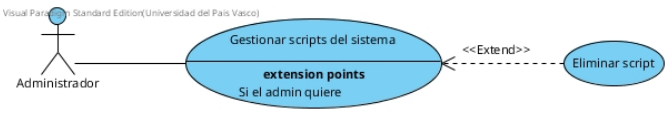


Nombre: Eliminar usuario
Descripción: Permite al administrador eliminar un usuario del sistema.
Actores: Administrador
Precondiciones: Gestionar usuarios del sistema.
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El administrador pulsa sobre el botón "eliminar usuario".
2. Se le muestra una pantalla de confirmación para realizar o no la acción.
   a) Si el administrador pulsa sobre el botón "cancelar", se le devuelve a la interfaz de gestionar usuarios del sistema.
   b) Si el administrador pulsa sobre el botón "aceptar", se procederá al borrado del usuario y se le devolverá a la pantalla de gestionar usuarios del sistema.

Postcondiciones: Al borrar un usuario, se borrarán todos los grupos con sus alumnos asociados, todos los scripts aplicados en cada grupo, todos sus tags y todo el historial que tuviera almacenado en el sistemas hasta la fecha.
Comentarios: Ninguno

A.2.4. SUBCASO: Eliminar script



Nombre: Eliminar script

Descripción: Permite al administrador eliminar un script del sistema.
Actores: Administrador
Precondiciones: Gestionar scripts del sistema.
Requisitos no funcionales: Ninguno
Flujo de eventos:

1. El administrador desde la interfaz de gestionar script del sistema, selecciona un script y pulsa el botón "eliminar script".

2. Se le muestra una ventana de confirmación que le pregunta si está seguro de la acción a tomar.

   a) Si el administrador pulsa "aceptar", el sistema procederá al borrado del script el cual realizará dos acciones dependiendo de ciertas condiciones.

      1) Si el script no estaba asociado a ningún grupo éste se eliminará directamente y mostrará al administrador una pantalla de confirmación.
      2) Si el script estaba asociado a algún grupo se le mostrará al administrador una pantalla que le indicará qué alumnos están afectados por la eliminación del script.

         a′ Si el administrador pulsa en "confirmar eliminación" el sistema borrará por completo el script y quitará la aplicación a todos los alumnos que estén afectados por el script.

         b′ Si el administrador pulsa "cancelar" el sistema le devolverá a la pantalla de Gestionar scripts del sistema.

   b) Si el administrador pulsa "cancelar" se le devolverá a la pantalla de Gestionar Scripts del sistema.

Postcondiciones: Ninguna
Comentarios: Ninguno

A.2.5. Imágenes

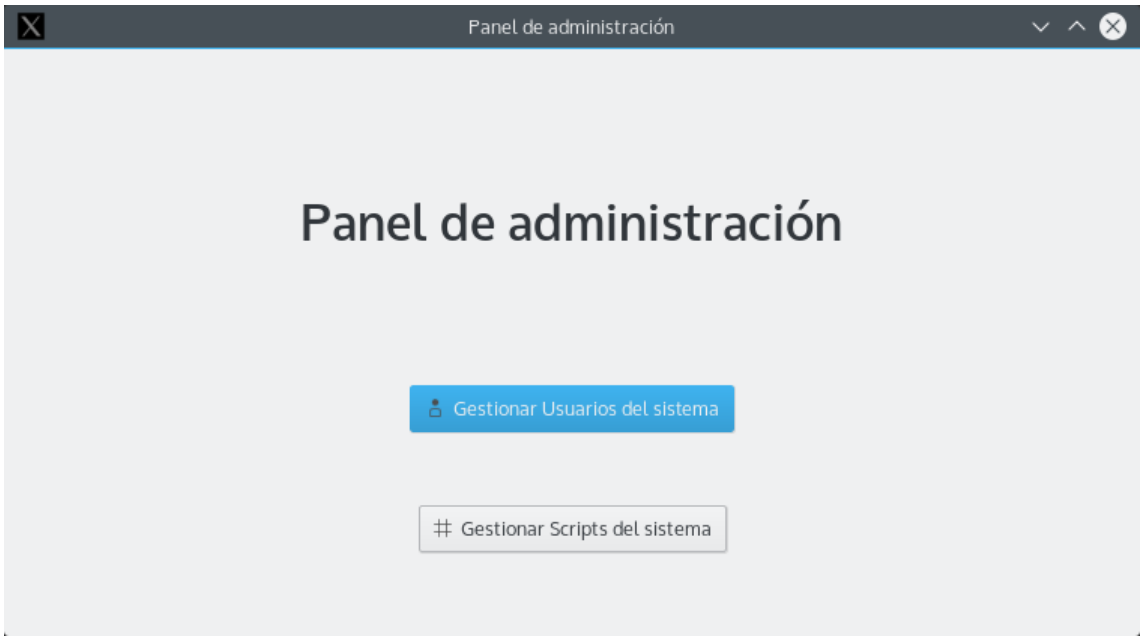


Figura A.12: Panel principal del administrador del sistema.

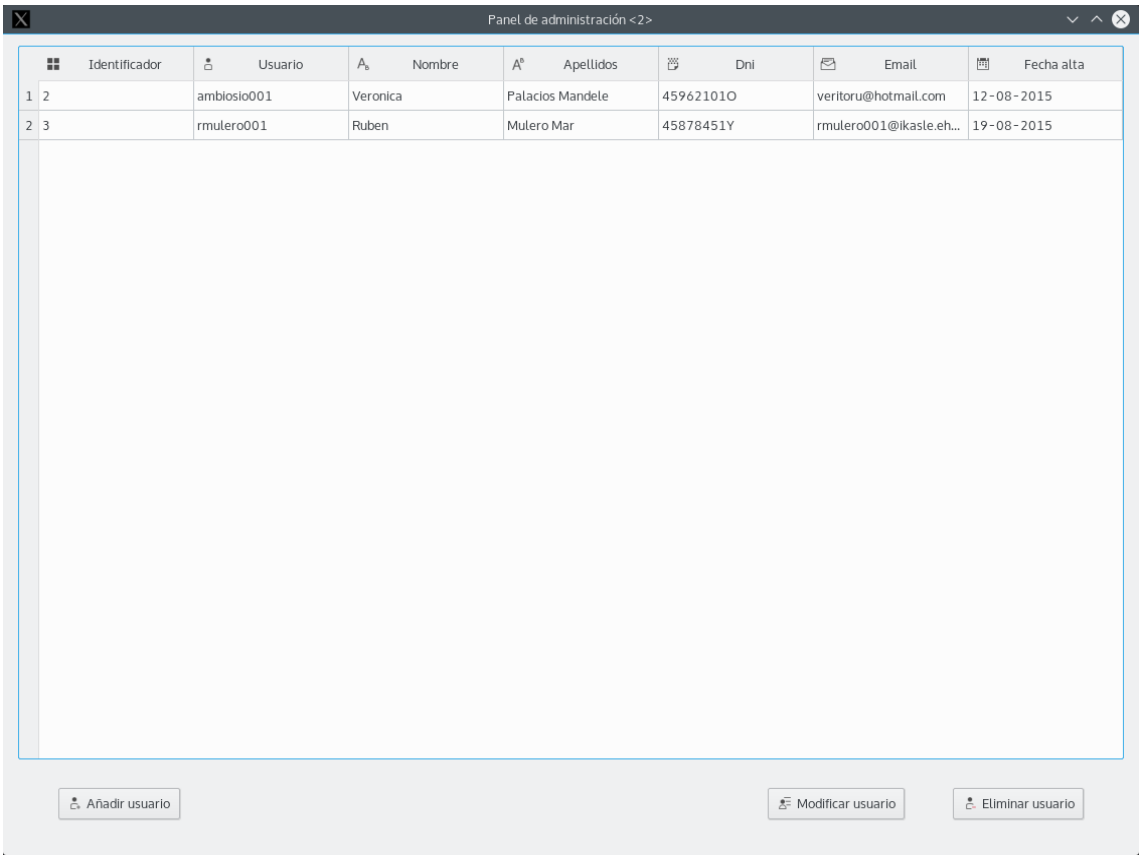


Figura A.13: Interfaz de gestión de los usuarios del sistema.

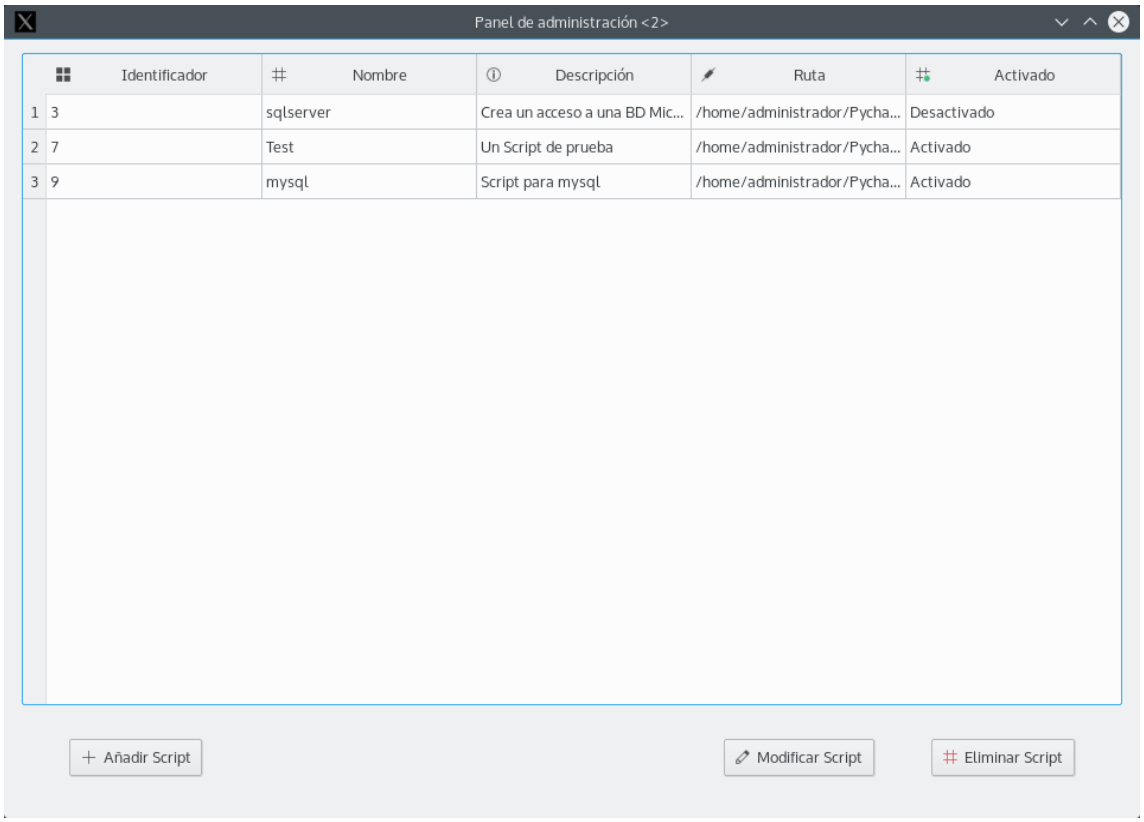


Figura A.14: Interfaz de gestión de los scripts del sistema.