RegistAB. Software de gestión documental.

Proyecto Ciclo Formativo Superior Desarrollo de Aplicaciones Multiplataforma.

Marzo 2014

TABLA DE CONTENIDO

1 PLANIFICACIÓN.

1.1 MARCO DE REFERENCIA.

El contexto en el que se encuadra el proyecto es el siguiente. Se trata de una administración pública (un organismo autónomo) de carácter nacional pero con oficinas en todas las provincias. Este organismo tiene cerca de 4.000 empleados en todo el territorio nacional, si bien, en la oficina provincial que nos ocupa, el número de empleados es de unos 40, de los cuales, somos dos los que nos encargamos del departamento de informática. Este departamento se encarga de tareas tan diversas como son el mantenimiento de los equipos y de la red, apoyo a usuarios, gestión de inventarios, gestión de bases de datos e incluso el desarrollo de aplicaciones.

En esta oficina provincial se utilizaban frecuentemente diversas bases de datos de tipo ACCESS, en concreto, una de estas bases de datos se utilizaba para gestionar toda la información referente al registro de entrada y salida de documentos. No obstante, debido a las restricciones presupuestarias, desde los servicios centrales se eliminaron la práctica totalidad de las licencias OFFICE, con la consiguiente desinstalación de la aplicación ACCESS de casi todos los equipos de la dependencia. Si bien, las aplicaciones WORD y EXCEL fueron relativamente fácilmente sustituidas por el paquete ofimático LIBREOFFICE, en un primer momento no se encontró una solución satisfactoria al problema generado por la desinstalación de la aplicación ACCESS.

Por otro lado, un elevado porcentaje de los equipos de este centro de trabajo funcionan con Windows XP, cuyo soporte finaliza el 8 de Abril de 2014. Más allá de esta fecha no puede haber ningún equipo funcionando con Windows XP, por lo que está previsto que alguno de estos equipos pase a funcionar con una distribución GNU/Linux, siendo necesario por lo tanto, encontrar una alternativa que sea multiplataforma.

Desde la dirección provincial me solicitan que realice un estudio para buscar y poner en funcionamiento una alternativa al uso de ACCESS y que, a grandes rasgos, cumpla con los siguientes requisitos:

- Software libre o en su defecto, bajo coste.
- Multiplataforma.
- Facilidad de uso. Debe poder ser utilizado por todos los usuarios independientemente de sus conocimientos informáticos.

- La migración de datos del sistema antiguo al nuevo debe ser rápida y a ser posible, sin coste alguno.
- El nuevo sistema gestor de bases de datos debe ser capaz de funcionar en red y admitir concurrencia. No obstante, al ser una dependencia pequeña, la carga de trabajo que deba soportar el sistema no será muy elevada.

Una vez realizado el estudio correspondiente, se valoran distintas alternativas, como son el simple uso de una hoja de cálculo almacenada en el servidor, utilización del programa BASE incorporado en LIBREOFFICE o algún otro SGBD libre, desarrollo de una aplicación web migrando los datos a MySQL o PostgreSQL, etc.

Finalmente, la solución adoptada es la migración de los datos a un SGBD de tipo SQLite, almacenado localmente en el servidor, al que cada usuario puede acceder por medio de un sencillo cliente desarrollado en Java.

Las ventajas e inconvenientes de esta solución frente al resto de alternativas propuestas, así como los motivos por los que nos decidimos por esta alternativa, se expondrán en los siguientes apartados.

1.2 EL PROYECTO.

1.2.1 Objetivos.

El objetivo principal es la implantación de un sistema de registro informático en el que quede constancia de toda entrada y salida de documentos en una dependencia administrativa. Este objetivo se divide a su vez en distintos sub-objetivos:

- En un contexto de restricciones presupuestarias, el coste debe ser el menor posible, utilizando soluciones de software libre siempre que sea posible. La migración de datos del sistema antiguo al nuevo debe ser sencilla y de bajo coste.
- No todos los empleados estaban capacitados para utilizar el anterior sistema, debido a su relativa complejidad. Se pretende aprovechar el cambio para que el nuevo sistema sea intuitivo y fácil de usar, y de esta manera hacer su uso extensible a todos los empleados de la dependencia.
- Cada apunte que se guarde en el sistema debe registrar de forma automática y transparente para el usuario una serie de datos necesarios para su posterior auditoría, almacenándose el código del usuario que realiza el apunte, así como fecha y hora.

- El sistema debe soportar la concurrencia y garantizar la integridad de los datos, puesto que serán varios los usuarios que utilicen el sistema simultáneamente.
- Se debe garantizar la seguridad de los datos, estableciendo para ello un mecanismo de copias de seguridad automáticas y periódicas.
- El sistema debe incluir un generador de informes, mediante el que se pueda extraer la información en base a una serie de parámetros definidos por el usuario (por ejemplo, registros entre fechas, registros por usuario entre fechas, registros de un determinado negociado, etc). Estos informes deben ser generados en formato pdf.
- Es necesario establecer diversos perfiles de usuario, de forma que todos los usuarios puedan guardar registros, pero solo algunos previamente determinados tengan control total sobre la base de datos.

1.2.2 Ciclo de vida.

Para este proyecto se estima que lo más conveniente es utilizar una **metodología ágil** de desarrollo de software, lo que en la práctica conlleva que se sigan las siguientes premisas:

- Se valora el software que funciona por encima de la documentación exhaustiva.
- Integración activa del "cliente" en el proyecto, puesto que es el verdadero conocedor de las necesidades que se pretenden solucionar.
- Se realizarán entregas tempranas y continuas de valor, en iteraciones de no más de 15 días, intentando aportar el mayor valor posible en cada iteración del software.
- Se valorará la adaptación al cambio por encima del seguimiento de un plan, es decir, anticipación y adaptación frente a planificación y control. Los requisitos cambiantes no son un inconveniente sino que son bienvenidos.

El producto funcional, es decir, software operativo, será la principal medida de progreso del proyecto. Centraremos el interés en el grado de finalización funcional y no en el tiempo transcurrido contra el planificado.

Existen diversas metodologías ágiles (SCRUM, Crystal_Clear, Programación Extrema, etc), no obstante, debemos encontrar aquella que mejor se adapte a la dimensión reducida del proyecto. No debemos olvidar que será un solo técnico el que se encargue del desarrollo del proyecto, que además es un empleado de la misma “empresa cliente" que ha encargado el desarrollo.

En este caso, nos decantaremos por utilizar una adaptación de la metodología ágil conocida como Programación Extrema *o* ***eXtreme Programming (XP)***, formulada por *Kent Beck*. Esta metodología se fundamenta en los siguientes valores:

- Simplicidad. Es la base de la programación extrema. Se buscará la simplicidad tanto en el código como en la documentación (el código debe documentarse en su justa medida y debe ser autodocumentado, es decir, eligiendo adecuadamente los nombres de variables, métodos y clases).
- Comunicación. Tanto entre los programadores como con el cliente. En este caso, solo hay un programador, pero todavía sigue siendo necesaria la comunicación con el "diente". Esta comunicación con el cliente debe ser fluida ya que es quien debe determinar que características son las que tienen prioridad y solucionar las dudas que vayan surgiendo.
- Retroalimentación. La comunicación descrita en el apartado anterior tiene que derivar necesariamente en una retroalimentación o feedback entre el cliente y el desarrollador. Como el cliente forma parte activa del equipo de desarrollo, su opinión sobre el estado del proyecto se conoce prácticamente en tiempo real.
- Valentía para, por ejemplo, reconstruir o desechar código cuando sea necesario, sin importar cuánto esfuerzo o tiempo se invirtió en ese código.
- Respeto entre los miembros del equipo.

De estos valores descritos, se derivan una serie de características fundamentales de la Programación Extrema, que son:

- Desarrollo iterativo e incremental, es decir, pequeñas y continuas mejoras.
- Pruebas unitarias continuas y automatizadas.
- Programación en parejas. Esta característica no es aplicable a este proyecto puesto que será un solo programador el encargado de llevarlo a cabo.
- Integración del equipo de programación con el cliente.
- Refactorización del código.
- Propiedad del código compartida. De nuevo, esta característica no es aplicable por el mismo motivo descrito anteriormente.
- Simplicidad en el código.

1.3 ORGANIZACIÓN DEL TRABAJO.

1.3.1 Análisis de requisitos.

El desarrollo e implantación de la solución debe cumplir los siguientes requisitos:

1. Ser económicamente viable.
2. Estar al alcance técnico del personal de la empresa que va a desarrollar el proyecto.
3. La migración de datos debe ser rápida y sin coste.

La solución finalmente adoptada debe, a su vez, cumplir con estos requisitos:

1. Multiplataforma.
2. Curva de aprendizaje reducida.
3. Admitir concurrencia.
4. Garantizar la seguridad de los datos almacenados.
5. Permitir la auditoría de los datos almacenados (qué usuario y cuando realiza modificaciones)

En base a estos requisitos elaboraremos un lista de comprobación que nos servirá posteriormente para seleccionar la mejor solución de entre las distintas alternativas (tabla 1).

Tabla 1. Lista de comprobación alternativas/requisitos.

	Alternativa 1	**Alternativa 2**	**Alternativa 3**	**Alternativa 4**
Económicamente viable				
Personal cualificado				
Migración de datos eficiente				
Multiplataforma				
Curva de aprendizaje reducida para los usuarios				
Concurrencia				
Seguridad				
Auditoría				

1.3.2 Análisis de las distintas alternativas.

1.3.2.1 Alternativa 1. LIBREOFFICE CAL

La solución más sencilla de implantar sería diseñar una hoja de cálculo utilizando LIBREOFFICE CALC. Este archivo estaría almacenado en el servidor, de tal forma que todos los usuarios puedan acceder y realizar las anotaciones correspondientes.

Las únicas ventajas de esta solución son su sencillez e inmediatez de implantación, pero sin duda, son más numerosos e importantes los inconvenientes. La seguridad es casi inexistente, el archivo se podría borrar de forma accidental. Por otro lado, la concurrencia es muy limitada, un usuario no podría modificar el archivo mientras otro lo tenga abierto. Además de que sería difícil realizar una auditoría.

1.3.2.2 Alternativa 2. LIBREOFFICE BASE.

Utilización de un sistema gestor de bases de datos open-source de escritorio. En concreto, se pensó en la posibilidad de utilizar LIBREOFFICE BASE. Esta alternativa es la que supone una mayor continuidad con respecto al sistema anterior, puesto que BASE es compatible con los archivos de tipo .mdb utilizados por ACCESS. No obstante, sería necesario desarrollar los formularios e informes, y sigue teniendo las mismas desventajas que en el caso anterior.

1.3.2.3 Alternativa 3. Aplicación Web.

Desarrollo de una aplicación web conectada con un sistema gestor de bases de datos open-source. Probablemente la solución más elegante sería el desarrollo de una aplicación web a la que pudieran acceder todos los usuarios de la red local por medio de su navegador. Se utilizaría un SGBD libre como MySQL o PostgreSQL.

El principal inconveniente es que el personal de la empresa encargado del desarrollo del proyecto no está formado en desarrollo web, con lo cual, los tiempos de desarrollo se alargarían puesto que los programadores tendrían que ir formándose “sobre la marcha”, o bien, sería necesario contratar personal externo que se encargase del proyecto por lo que los costes se elevarían notablemente.

1.3.2.4 Alternativa 4. Cliente JAVA

Desarrollo de aplicación JAVA conectada a SGBD open-source. Esta solución consiste en el desarrollo de un cliente JAVA-SWING, que se ejecutaría localmente en cada equipo, pero que conectaría con un SGBD distribuido (MySQL, PostgreSQL, SQLite, etc).

Esta solución satisface todos los requisitos descritos en el apartado anterior. Es económicamente viable, el personal que lo tiene que desarrollar está técnicamente cualificado para ello, la migración de datos es eficiente, se puede desarrollar un cliente JAVA con una curva de aprendizaje asequible para los usuarios. El SGBD garantizará las condiciones concurrencia, integridad y seguridad en los datos.

1.3.3 Selección de la alternativa a desarrolar.

En base a los requisitos que debe cumplir la solución, descritos anteriormente, se rellena la lista de verificación, obteniendo el resultado mostrado en la tabla 2.

Tabla 2. Lista de comprobación alternativas/requisitos cumplimentada.

	Alternativa 1	**Alternativa 2**	**Alternativa 3**	**Alternativa 4**
Económicamente viable	**X**	**X**		**X**
Personal cualificado	**X**	**X**		**X**
Migración de datos eficiente		**X**	**X**	**X**
Multiplataforma	**X**	**X**	**X**	**X**
Curva de aprendizaje reducida para los usuarios	**X**	**X**	**X**	**X**
Concurrencia			**X**	**X**
Seguridad			**X**	**X**
Auditoría			**X**	**X**

Una vez analizados los requisitos y las diferentes alternativas, se llega a la conclusión consensuada con la dirección provincial, de que la solución adecuada es la número 4. Se procederá, por tanto, a desarrollar un cliente JAVA que actuará de front-end conectado con un SGBD aún por determinar que actuará de back-end.

1.3.4 Implantación de la alternativa adoptada.

Como ya se ha mencionado, para el desarrollo general del proyecto se ha optado por utilizar una metodología de desarrollo ágil (en concreto, la metodología *eXtreme Programming*). Esta metodología permite trabajar con mucha mayor flexibilidad que una metodología clásica o en cascada.

No estamos sujetos al cumplimiento estricto de unas fechas de entrega previamente definidas, sino que por el contrario, realizaremos entregas sucesivas y continuas (iteraciones). Cada una de estas iteraciones supondrá la entrega de una versión plenamente funcional y mejorada con respecto a la anterior. Como

ya se ha dicho anteriormente, se valorará la adaptación al cambio por encima del seguimiento de un plan, es decir, anticipación y adaptación frente a planificación y control.

Por lo tanto, la implantación de la aplicación se realizará mediante sucesivas iteraciones, cada una de las cuales ha de suponer un aumento de las capacidades de la aplicación. Podemos definir previamente una serie de capacidades que la aplicación debe tener implementadas una vez que esté finalizada. Sin olvidar, que partimos de un modelo de desarrollo en el que prima la adaptación sobre la planificación y de que es indispensable la comunicación permanente con el cliente (en este caso con la dirección de la dependencia).

Una lista previa de estas funcionalidades es la siguiente, no necesariamente por orden:

- Añadir registros a la base de datos.
- Generar informes en pdf en base a los siguientes parámetros:
    - Registros por fechas.
    - Registros por operador.
    - Registros por negociado.
- Implementar distintos roles de acceso.
    - Rol de usuario.
    - Rol de administrador.
- Gestión gráfica de la base de datos (consulta y modificación de registros)
- Gestión de copias de seguridad.
- Auditoría de datos.

Las sucesivas iteraciones deberán ir añadiendo estas funcionalidades hasta que finalmente la aplicación esté terminada, teniendo en cuenta, que nuevas funcionalidades pueden ser requeridas durante el proceso de desarrollo o incluso desechar algunas de las previamente definidas.

Corresponde al programador junto con la persona designada por la dirección establecer cuáles serán las funcionalidades a implementar en las siguientes iteraciones.

El proceso de implantación se explica con mayor detalle en el apartado 2.4.

1.3.5 Herramientas.

1.3.5.1 Software.

Este apartado se enumeran y describen brevemente las herramientas software que se utilizarán en el proyecto. No obstante, en cuanto al SGBD, se ha realizado una preselección de tres alternativas. Más adelante, durante la etapa de diseño será el momento de valorar y decidirse por el SGBD a utilizar.

- **Sistema Gestor de Bases de Datos**. Se seleccionará una de entre las siguientes opciones:
    a. **MySQL**. Sistema Gestor de Bases de Datos relacional de tipo cliente-servidor y código abierto bajo licencia GNU-GPL para el desarrollo de productos no privativos.
        i. WampServer. Nos permite instalar de forma sencilla las herramientas necesarias para administrar y trabajar MySQL.
        ii. phpMyAdmin.
    b. **PostgreSQL**. SGBD relacional orientado a objetos y código libre bajo licencia BSD. Al igual que MySQL también es de tipo cliente-servidor.
        i. PgAdmin. Entorno de escritorio visual que facilita la gestión y administración de bases de datos PostgreSQL.
    c. **SQLite**. SGBD relacional de tipo embebido, es decir, a diferencia de los dos anteriores que son de tipo cliente-servidor, SQLite no es un proceso independiente con el que el programa principal se comunica sino que toda la base de datos está contenida en un único archivo y el programa accede a la base de datos a través de llamadas simples a subrutinas y funciones. Varios procesos o hilos pueden acceder a una misma base de datos. La lectura puede hacerse en paralelo, pero la escritura de datos no admite concurrencia. Se deberá prever y controlar esta circunstancia.
        i. SQLite Studio. Entorno visual de gestión y administración de bases de datos SQLite. Gratuito y de código abierto.
- **Entorno Integrado de Desarrollo (IDE):**
    a. **NetBeans**. Entorno Integrado de Desarrollo especializado en el lenguaje Java y con un potente diseñador de interfaces gráficas de usuario. Gratuito y de código abierto.
    b. Sistema de Control de Versiones. **Git**
- **Informes**:
    a. Librería **JasperReport**. Permite generar informes y exportarlos a múltiples formatos, entre ellos pdf. Se distribuye bajo licencia GNU por lo que es software libre.

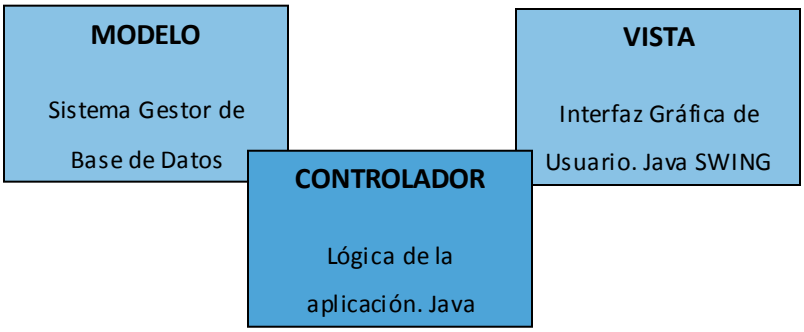
b. **I-Report**. (Diseño de informes). Para el diseño y la edición de los informes que posteriormente serán generados por la librería JasperReport. I-Report permite diseñar de forma sencilla estas plantillas de informe, que son archivos de tipo XML.

1.3.5.2 Hardware.

Para la fase de desarrollo de la aplicación no será necesario ningún hardware especial más allá de un equipo PC de sobremesa. Posteriormente, durante la fase de implantación puede ser necesario, dependiendo de si finalmente utilizados un SGBD de tipo cliente-servidor, un equipo que realice las funciones de servidor.

1.3.6 Patrón de diseño.

El desarrollo de la aplicación seguirá un patrón 'Modelo-Vista-Controlador' (MVC). Este modelo se basa en separar los datos (MODELO), la interfaz del usuario (VISTA) y la lógica interna (CONTROLADOR). Utilizando este patrón de diseño optimizaremos el trabajo mediante la reutilización de código y la separación de conceptos, facilitando tanto el desarrollo de la aplicación como su posterior mantenimiento.



1.3.7 Formularios.

Se considera importante la participación de los usuarios en el desarrollo del nuevo software. Esta participación se articulará fundamentalmente mediante la recogida de opiniones, sugerencias y reporte de errores. Para llevar a cabo esta participación se prevé la posibilidad de utilizar formularios para recoger la opinión y sugerencias de forma previa al inicio del desarrollo (formularios de planificación). También se deja abierta la posibilidad de utilizar formularios de evaluación, con el fin de conocer posibles fallos en el software y nuevas sugerencias y opiniones.

A continuación se recogen dos posibles modelos de formularios de planificación y evaluación:

FORMULARIO DE PLANIFICACIÓN	
FECHA	
NOMBRE Y APELLIDOS	
PUESTO DE TRABAJO	
Por parte de la dirección se ha decido implantar un nuevo programa informático para la gestión del registro de entrada/salida de documentos administrativos. Este nuevo software va a ser desarrollado por técnicos propios de esta dependencia. Como usuario activo que es del actual sistema de gestión documental, usted ha sido seleccionado para participar en la planificación del nuevo software. Tan solo tiene que contestar a las siguientes preguntas.	
RESPONDA BREVEMENTE LAS SIGUIENTES CUESTIONES:	
¿Cuál es su opinión sobre el actual sistema de gestión de documentos?	
¿En qué aspectos cree que puede ser mejorado?	
Sugerencias para el nuevo software de registro de documentación?	

FORMULARIO DE EVALUACIÓN	
FECHA	
NOMBRE Y APELLIDOS	
PUESTO DE TRABAJO	
Durante los últimos días usted ha participado en el periodo de evaluación previo a la puesta en funcionamiento del nuevo programa informático de gestión de entrada/salida de documentos. Es el momento de sacar conclusiones y realizar las modificaciones necesarias y subsanación de los posibles errores.	
RESPONDA BREVEMENTE LAS SIGUIENTES CUESTIONES:	
¿Cree que se han reflejado correctamente las sugerencias y especificaciones llevadas a cabo en el formulario de planificación anterior?	
¿Ha encontrado algún 'bug' (error) en el software? De ser así, explíquelo de la forma más detallada posible.	
Sugerencias (Si se deja en blanco, se entiende que queda satisfecho con el nuevo software)	

1.3.8 Actividades del proyecto.

Actividad 0: **Planificación del trabajo y estudio de las herramientas.**		**Actividad 1:** **Estudio del sistema existente.**		**Actividad 2:** **Análisis del sistema.**		**Actividad 3:** **Diseño e implementación del sistema.**		
Tarea 01: Planificación genérica del proyecto.	**Tarea 02:** Análisis y selección de las herramientas a utilizar.	**Tarea 11:** Análisis genérico del sistema existente.	**Tarea 12:** Análisis de la estructura de la base de datos.	**Tarea 21:** Realización de encuestas y entrevistas.	**Tarea 22:** Estudio de las propuestas y resultados obtenidos en la tarea anterior.	**Tarea 31:** Instalación de las herramientas necesarias (SGBD, WAMP, etc).	**Tarea 32:** Diseño de la estructura de la base de datos.	**Tarea 33:** Diseño de un primer prototipo del cliente JAVA.
7 horas	3,5 horas.	3,5 horas.	3,5 horas.	3,5 horas.	7 horas	3,5 horas.	7 horas.	10 horas.
Una persona y un equipo.								

<table>
<tr><td colspan="4">Actividad 4:

Implantación por iteraciones según metodología eXtreme Programming.</td><td>Actividad 5:

Mantenimiento y actualizaciones.</td></tr>
<tr><td>Iteración 1:

Implantación parcial en usuarios preseleccionados.</td><td>Iteración 2:

Implantación parcial en usuarios preseleccionados.</td><td>Iteración 3:

Implantación generalizada.</td><td>Iteración 4:

Implantación generalizada.</td><td rowspan="2">Tarea 51:

Realización de modificaciones y reparaciones que sean necesarias.</td></tr>
<tr><td colspan="4">En cada iteración será plenamente funcional y supondrá un aumento de capacidades con respecto a la iteración anterior además de solucionar los fallos que pueda tener la versión anterior.</td></tr>
<tr><td>30 horas.</td><td>15 horas.</td><td>15 horas.</td><td>15 horas.</td><td>14 horas.</td></tr>
<tr><td colspan="5">Una persona y un equipo.</td></tr>
</table>

1.3.9 Valoración económica.

Partimos de la base de que el desarrollo de este software no se trata del encargo de una empresa externa, sino que es la propia empresa 'cliente' la que va a dedicar sus propios recursos (humanos y técnicos) al desarrollo de la aplicación. Por lo tanto, no procede hablar de la elaboración de un presupuesto sino más bien de una valoración económica de la aplicación.

1.3.9.1 Costes de personal:

Según la planificación realizada en el punto anterior, se estima que serán necesarias 137,5 horas de trabajo. Queda, por tanto, determinar el coste por hora de trabajo. Según datos de la encuesta salarial de programadores Java realizada por la asociación JavaHispano.org, el suelo medio de un programador Java en España es de 25,910€ (datos de 2012). Teniendo en cuenta que en cómputo anual, la jornada laboral ordinario son 1780 horas de trabajo, podemos calcular un precio por hora de trabajo de 14,56€.

Por lo tanto en base a este criterio, la estimación de costes de personal del proyecto es de 2002€.

1.3.9.2 Costes de software:

A falta de seleccionar el software que se utilizará en el desarrollo de entre las herramientas descritas en el punto 1.3.5.1, lo que sí está claro es que todas ellas son gratuitas y tienen licencias de tipo 'open source'. Por lo tanto, en principio no existen costes derivados del uso del software.

1.3.9.3 Otros costes indirectos:

No es viable estimar los costes indirectos tales como el consumo eléctrico, calefacción y en definitiva costes asociados al funcionamiento de la oficina de trabajo, por lo que este aspecto no será tratado en el proyecto. En cualquier caso, estos costes indirectos de funcionamiento de la oficina no se verán afectados por el desarrollo del proyecto, por lo que se pueden omitir.

Por lo tanto, puesto que no existen costes derivados del uso de software y no se van a tener en cuenta los costes indirectos, los únicos costes asociados al proyecto son los de personal, que han sido estimados en 2002€.

2 DESARROLLO E IMPLANTACIÓN.

Una vez finalizada la fase de planificación del proyecto comenzamos con la fase de diseño e implantación. En esta fase realizaremos el análisis del sistema, se determinará el SGBD a utilizar y se diseñará un prototipo. Posteriormente comenzará la implantación del software en distintas iteraciones.

2.1 ANÁLISIS DEL SISTEMA.

Se debe crear un software en el que los usuarios guarden registro de todos los documentos que se reciben (entradas) y que se envían (salidas) en una dependencia de una administración pública. A su vez, en cada uno de estos registros se deben almacenar los siguientes datos: sección a la que se dirige el documento o sección que remite el escrito (según sea una entrada o una salida), procedencia/destino del documento, breve extracto del contenido del documento, fecha y hora de anotación del registro e identificación del usuario que realiza el apunte.

Además de lo anterior, cada registro debe tener un código identificativo con el siguiente formato: ENT/SAL + AAAA + nº de orden con 4 cifras. Por ejemplo, la entrada número 415 del año 2014 tendría el siguiente código identificativo: ENT20140415. Es necesario resaltar que la aplicación debe reconocer el cambio de año y poner el contador a cero de forma automática.

Habrá dos tipos de usuarios, grabadores y administradores. Los usuarios grabadores solo podrán añadir nuevos registros a la base de datos, mientras que los usuarios administradores podrán añadir nuevos registros y editar la información de la base de datos. Los usuarios estarán identificados por un solo dato que será su nombre de usuario en el LDAP de la red.

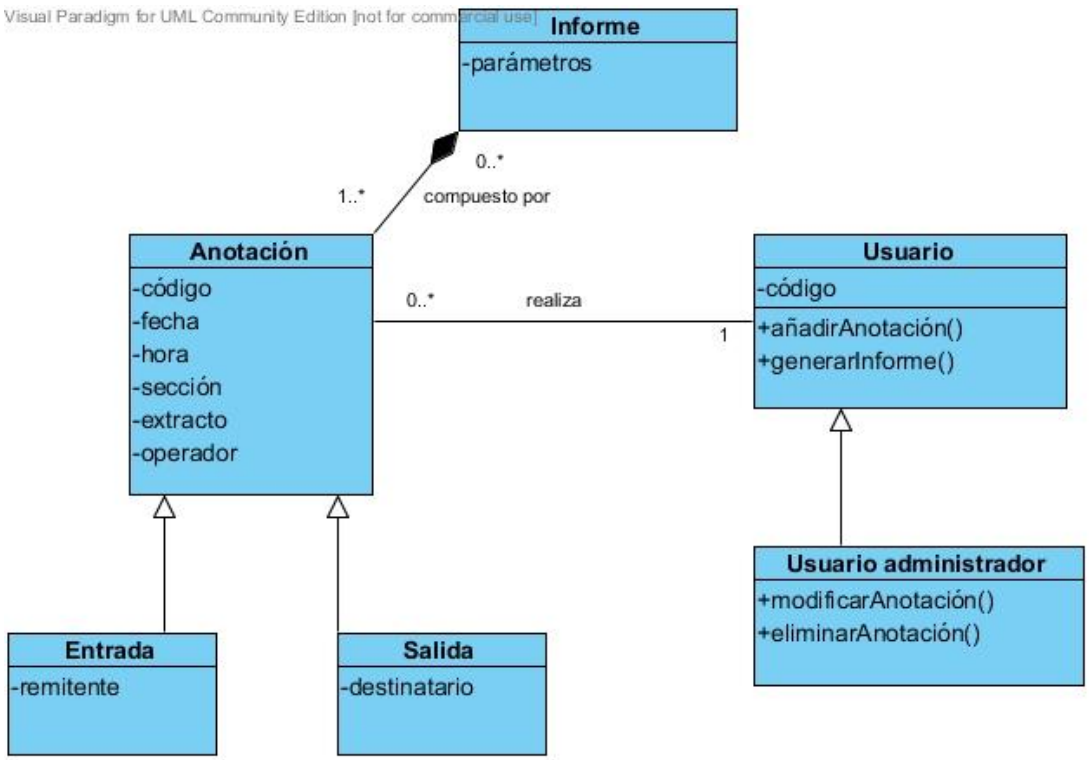
Por otro lado, la aplicación debe incluir un generador de informes (resúmenes) que admita los siguientes parámetros: entrada/salida, fechas de inicio y fin, operador que realiza el registro y sección. Estos informes deben ser generador en formato PDF.

Un requisito fundamental que debe cumplir el sistema es garantizar en la medida de lo posible la integridad y seguridad de los datos. Este asunto se abordará más en profundidad una vez determinado el SGBD a utilizar, puesto que dependiendo del SGBD el enfoque será distinto.

Para finalizar, la gestión de la base de datos se realizará, en un primer momento, mediante software externo, dependiendo del SGBD podrá ser con phpMyAdmin (paraMySQL), pgAdmin (para PostgreSQL),

SQLite Studio (para SQLite), etc. Se valorará más adelante si es necesario desarrollar una funcionalidad nueva añadida a la aplicación para la administración de la base de datos.

En base a este análisis del sistema, desarrollamos el siguiente diagrama de clases:



2.2 Determinación del Sistema Gestor de Bases de Datos.

Las herramientas necesarias para el proyecto ya se describieron en el punto 1.3.5.1. No obstante quedaba pendiente de determinar el Sistema Gestor de Bases de Datos a utilizar. En un principio, las alternativas eran tres: MySQL, PostgreSQL y SQLite.

En los dos primeros casos estamos hablando de SGBD de tipo cliente-servidor, es decir, el SGBD mantiene un proceso en ejecución de forma permanente en el equipo de la red local que hace la función de servidor. Los procesos cliente conectan con el proceso servidor para realizar las transacciones.

Las ventajas de este tipo de arquitectura son que el propio SGBD se encarga de gestionar la seguridad, concurrencia y gestión de usuarios (roles de acceso). Aspectos que requieren cierto tratamiento manual en el caso de las bases de datos de tipo embebido como es el caso de SQLite.

PostgreSQL funciona más rápido que MySQL en bases de datos grandes. No obstante, la velocidad de respuesta que ofrece PostgreSQL con bases de datos relativamente pequeñas es menos eficiente que la que ofrece MySQL.

Por el contrario, SQLite no es un proceso independiente sino que la base de datos se encuentra embebida en un archivo con el que el programa cliente se comunica. De esta forma se reduce la latencia en el acceso a la base de datos. Como desventaja podemos señalar que si bien varios procesos pueden acceder en paralelo para realizar lecturas, un acceso para escritura solo puede realizarse si no existe otro acceso de escritura simultáneamente. No obstante, los accesos de escritura apenas llevan unos pocos milisegundos por lo que en la práctica, para el volumen de uso esperado, esto no supone un problema, aunque debe ser convenientemente tratado mediante la gestión de las excepciones.

Otra gran desventaja de SQLite es que no admite el cifrado de la base de datos ni la gestión de usuarios. Los distintos roles de acceso a la base de datos deben ser establecidos mediante el sistema operativo, es decir, la base de datos es un archivo como otro cualquiera en el cual se deben establecer los correspondientes permisos de lectura o escritura según el usuario que acceda al archivo.

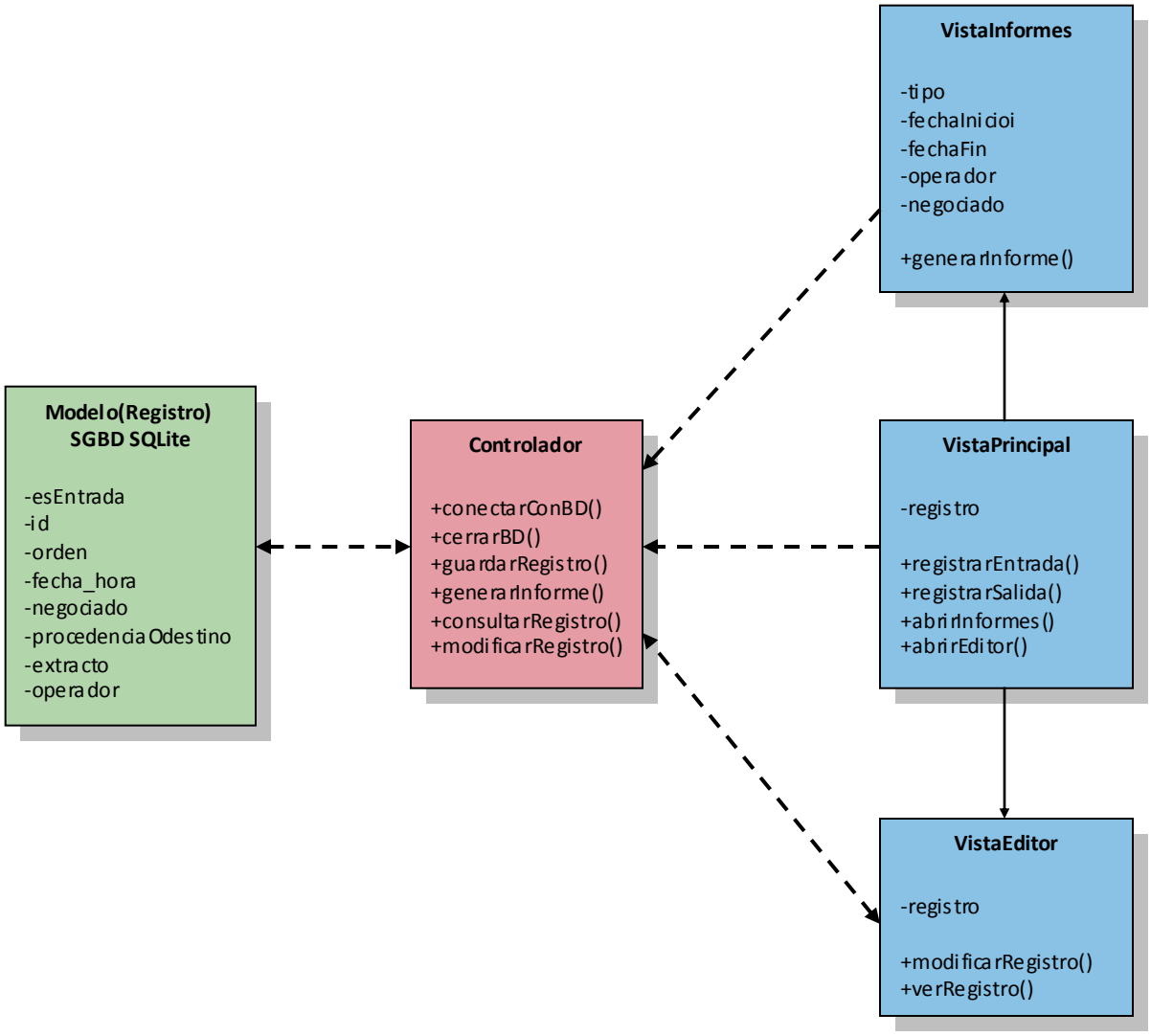
Estimamos que el SGBD más adecuado para este proyecto es MySQL. No obstante, una vez llegados a esta conclusión, nos encontramos con otro problema. Las políticas de seguridad establecidas desde los servicios centrales impiden la ejecución de software no autorizado en el servidor local, por lo que de utilizar este SGBD debería correr desde otro equipo de la red que haga las veces de host. No obstante, esto tiene otra serie de inconvenientes.

Finalmente, se decide utilizar SQLite por su sencillez y simplicidad de uso, que acortaría el tiempo de desarrollo, y porque se considera que sus desventajas son subsanables.

2.3 DISEÑO.

2.3.1 Diseño de un prototipo.

En base a los requisitos (apartado 1.3.1) y funcionalidades (apartado 1.3.4) solicitados por la dirección, y teniendo en cuenta que se ha optado un patrón de diseño de tipo Modelo-Vista-Controlador se puede realizar un prototipo de la aplicación en forma de diagrama. Este diagrama es una previsión esquemática de cómo debe ser la aplicación una vez terminada la última iteración, es decir, una vez el programa esté terminado.



En azul las clases que formarán la interfaz gráfica (VISTA), un formulario principal en el que se podrán insertar nuevos registros y un menú desde el que se podrá acceder al formulario de edición de registros y al del generador de informes. En rojo la clase CONTROLADOR que se encargará de gestionar la lógica de la aplicación. Todas las llamadas al SGBD (MODELO) se realizarán a través del controlador, es decir, la interfaz gráfica no conectará directamente con la base de datos sino que lo hará siempre a través de los métodos de la clase controlador.

Con este patrón de diseño facilitamos tanto el mantenimiento como la escalabilidad del código.

2.3.2 Diseño de la estructura de la base de datos.

No es necesaria una base de datos compleja para el proyecto que nos ocupa. En principio, es suficiente con dos tablas, una para registrar las entradas y otra para registrar las salidas. No hay ninguna relación entre ambas tablas. Cada una de las tablas tendrá los siguientes campos: id, orden, fecha_hora, sección, remitente/destinatario, extracto y operador. Todos los campos son de tipo texto (SQLite no tiene un tipo fecha) excepto el campo id que es numérico autoincremental (generado automáticamente por SQLite) que es también la clave primaria.

Por otro lado, se crearán dos tablas auxiliares: entradasXaño y salidasXaño con dos campos cada una, que son año y orden. Estas tablas tienen un solo registro, que es reescrito con cada nueva anotación en la tabla principal (entradas o salidas, según corresponda). El campo año recoge el año actual, y el campo orden el número del siguiente registro a anotar.

El uso de estas tablas auxiliares permite trabajar más cómodamente con el campo "orden" de las tablas principales. Por ejemplo, de esta manera es sencillo comprobar un cambio de año y por lo tanto, poner el contador de registros a cero. Recordemos que el campo "orden" tiene el siguiente formato ENT/SAL + AAAA + Nº de orden, donde el nº de orden debe empezar desde cero en cada inicio de año.

2.4 IMPLANTACIÓN.

Como ya se ha mencionado anteriormente, para el desarrollo de este proyecto se ha optado por la utilización de una metodología ágil, y de entre la gran variedad de metodologías que pueden definirse como ágiles optamos por la llamada metodología *eXtreme Programming*, con la que podemos trabajar con mayor flexibilidad que con una metodología clásica o en cascada.

De esta forma, no estamos sujetos al cumplimiento estricto de fechas de entrega previamente definidas, sino que por el contrario, realizaremos entregas sucesivas y continuas llamadas iteraciones. Cada iteración supone la entrega de una versión plenamente funcional y mejorada con respecto a la anterior, es decir, cada iteración debe suponer un aumento de las capacidades de la aplicación.

Partimos de un modelo de desarrollo en el que prima la adaptación sobre la planificación y, por lo tanto, es fundamental la comunicación permanente con el cliente (en este caso con la dirección de la dependencia).

2.4.1 Iteración 1.

La primera versión funcional de la aplicación tiene como única funcionalidad la de permitir a los usuarios introducir registros en la base de datos. Para ello se diseñó una interfaz gráfica de usuario que, a grandes rasgos, será la definitiva y se utilizará en las siguientes iteraciones.

Esta iteración cumple con los siguientes requisitos definidos durante la fase de planificación:

- Conectar con la base de datos y permitir la introducción de nuevos registros.
- Gestionar la posible concurrencia en las operaciones de escritura.
- Interfaz gráfica de usuario sencilla y consistente (imagen 1).

Imagen 1. Interfaz principal.

De momento, para la gestión de la base de datos (modificación o eliminación de registros, etc.) se utilizará una herramienta externa, habiéndose optado por la aplicación SQLite Studio.

2.4.2 Iteración 2.

Solución de errores encontrados en la iteración previa y adopción de mejoras propuestas por los usuarios:

- Esta nueva versión soluciona un error que consistía en que la base de datos quedaba bloqueada si no se pulsaba “aceptar” en el cuadro de diálogo que aparece tras insertar un registro, con lo que ningún otro usuario podía insertar nuevos registros mientras otro usuario tuviera el cuadro de diálogo abierto.
- Se adopta una mejora propuesta por algunos usuarios, que consiste en que añada una etiqueta donde se pueda ver en la interfaz cuál ha sido el código del último registro, puesto que si no lo apuntaban en el momento se les olvidaba y ya no podían consultarlo de ninguna otra forma salvo pidiendo el dato al administrador de la base de datos.

La aplicación ahora incorpora un sistema de copias de seguridad para prevenir la pérdida accidental de datos.

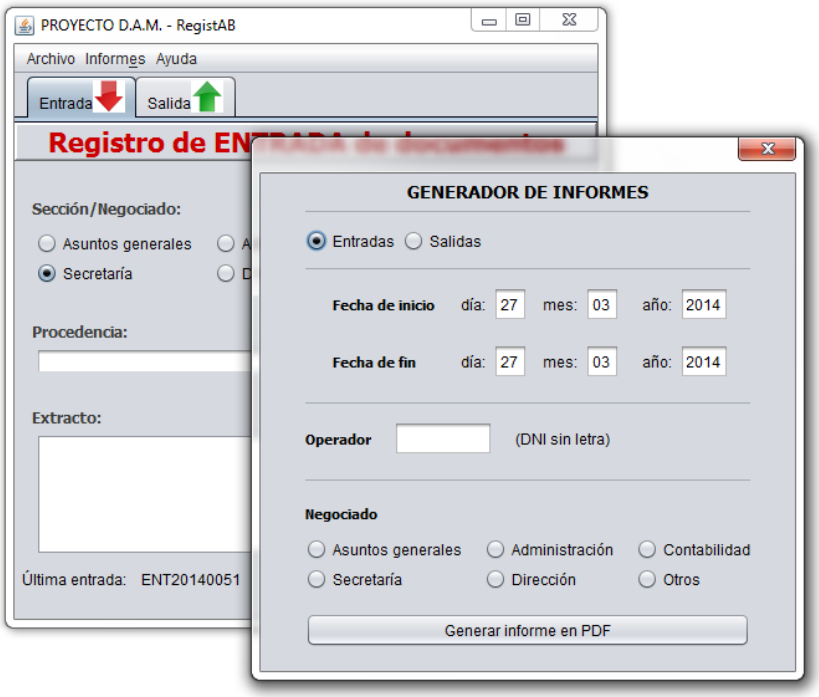
Se implementa un generador de informes capaz de exportar el informe a formato pdf y que admite generar informes en función de los siguientes parámetros introducidos por el usuario:

- Registros por fechas.
- Registros por operador.
- Registros por negociado.

El diseño del informe se ha realizado con la herramienta i-report, la cual genera una plantilla en formato XML que es la que se utiliza por la aplicación para generar el informe, utilizando para ello la librería jasperReport. Por lo tanto, de ser necesario modificar el diseño del informe, es suficiente con editar con i-report los archivos “informeEntradas.jrxml” e “informeSalidas.jrsml” situados dentro del directorio “plantillasInformes”.

La interfaz gráfica del generador de informes sigue el estilo definido en el resto del proyecto. Se puede ver en la imagen 2.

Imagen 2. Interfaz del generador de informes.



2.4.3 Iteración 3.

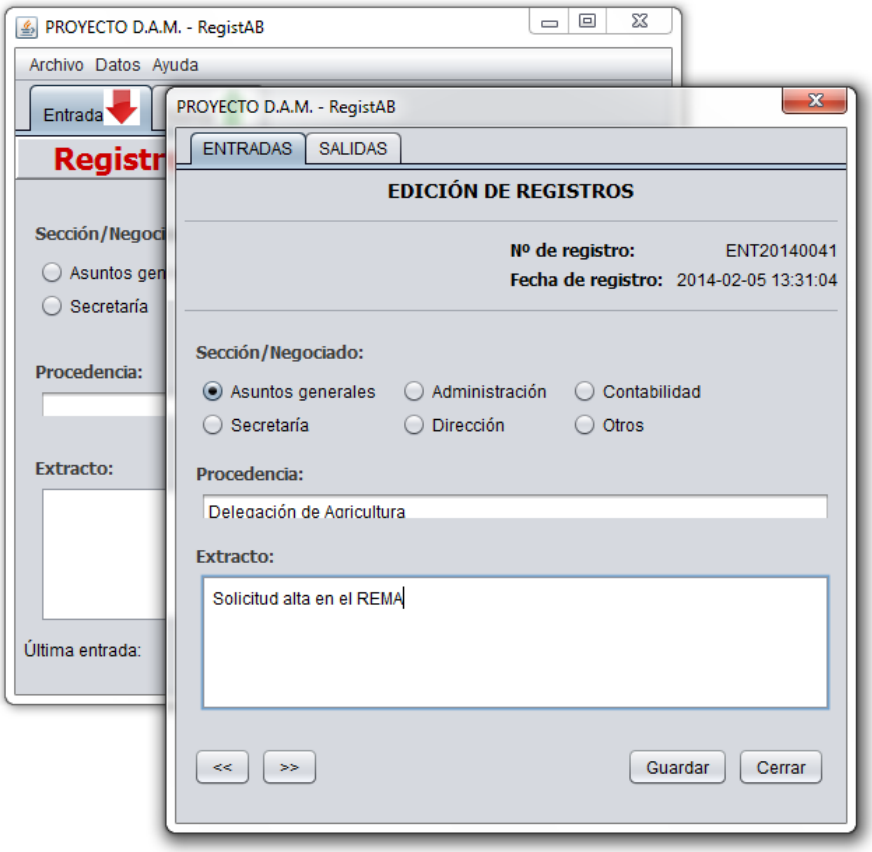
La principal novedad de esta versión es que incluye un formulario para la gestión de la base de datos (modificación de registros), donde cada usuario pueda modificar sus propios registros pero no los registros realizados por otros usuarios. La eliminación de registros no estará permitida, así como la modificación de los campos fecha_hora, orden y operador.

El formulario para la modificación de registros sigue el estilo definido en el resto del proyecto, como se puede apreciar en la imagen 3.

Por otro lado, se corrige un error reportado por los usuarios que consistía en que al insertar un apostrofe (‘) en los campos “procedencia” o “extracto” se producía un error que impedía realizar el registro.

Otro error subsanado en esta versión, en este caso reportado por la dirección, es que se podían añadir registros aunque alguno de los campos estuviera vacío, produciéndose anotaciones incompletas. En esta nueva iteración salta un mensaje de error en caso de intentar hacer un registro incompleto.

Imagen 3. Formulario de edición de registros.



2.4.4 Iteración 4.

En esta versión la aplicación reconoce dos tipos de usuarios. Por un lado, los usuarios sencillos, que pueden insertar registros y modificar sus propios registros, y por otro lado, los usuarios administradores que pueden además de insertar registros, modificar registros realizados por cualquier otro operador. Para ello, puesto que SQLite no admite la gestión de usuarios, se ha creado una nueva base de datos denominada *users.sqlite* en la cual están registrados los códigos de usuario de los administradores. Esta base de datos se encuentra en un archivo de solo lectura, para evitar de esta forma modificaciones no autorizadas.

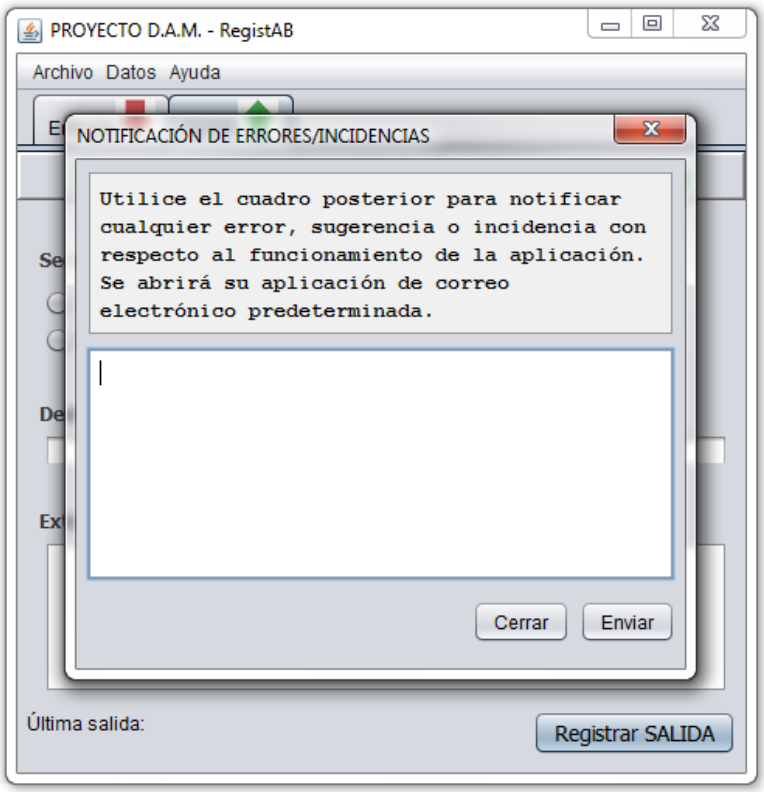
Cuando la modificación de un registro solo pueda ser realizada por un administrador, el programa buscará en *user.sqlite* para comprobar si el usuario forma parte del grupo de administradores.

Por último, ya que no están previstas nuevas iteraciones, se incluye un nuevo formulario (imagen 4) al que se accede desde el menú Ayuda, con el que los usuarios podrán notificar errores o incidencias no detectadas previamente y que surjan durante el uso de la aplicación. Al pulsar el botón enviar se abrirá

el programa de correo electrónico predeterminado con un correo electrónico nuevo en el que los campos dirección, asunto y cuerpo del mensaje ya estarán rellenados.

Al introducir este formulario la aplicación no queda "abandonada" sin más, sino que queda abierto un canal de comunicación entre los usuarios y el desarrollador. Se podrán corregir los erros que vayan surgiendo y que son inevitables en todo desarrollo de software y además se recogerá "feedback" en forma de sugerencias para una posible nueva versión mayor de la aplicación, tal y como se recoge en el apartado 4.

Imagen 4. Formulario de notificación de errores e incidencias.



3 EVALUACIÓN.

Con la cuarta iteración se da por concluida la fase de desarrollo e implantación. Es el momento de evaluar si tanto el proyecto en su conjunto, como la aplicación finalmente desarrollada han cumplido con los requisitos que se plantearon al comienzo.

3.1 EVALUACIÓN DEL PROYECTO.

Los requisitos que debía cumplir el desarrollo del proyecto en su conjunto eran los siguientes:

1. Ser económicamente viable.
2. Estar al alcance técnico del personal de la empresa que va a desarrollar el proyecto.
3. La migración de datos debe ser rápida y sin coste.

Tal como se planificó, no se ha incurrido en costes de software ni ha sido necesario adquirir hardware que no estuviera previamente disponible. Por otro lado, el proyecto se ha desarrollado satisfactoriamente sin necesidad de contratar personal adicional.

En cuanto a la migración de los datos, no supuso ningún problema puesto que existen varias herramientas libres diseñadas para convertir archivos Acces (.mdb) a SQLite. En concreto se optó por la aplicación open-source *mdb-sqlite-1.0.2*.

Se puede concluir por tanto, que el desarrollo del proyecto ha cumplido con los requisitos planteados al inicio del mismo.

3.2 EVALUACIÓN DE LA APLICACIÓN.

En el apartado 1.3.1 se establecen una serie de requisitos que debe cumplir la aplicación:

1. Multiplataforma.
2. Curva de aprendizaje reducida.
3. Admitir concurrencia.
4. Garantizar la seguridad de los datos almacenados.
5. Permitir la auditoría de los datos almacenados (qué usuario y cuando realiza modificaciones)

La aplicación ha sido desarrollada íntegramente utilizando tecnologías multiplataforma, además, el código ha sido realizado teniendo en cuenta las posibles diferencias dependiendo del sistema operativo (por ejemplo, utilizando *java.io.File.separator* en lugar de "/" o "\" en las rutas a ficheros).

Con respecto a la dificultad de uso, la opinión recogida tanto de los usuarios de la aplicación como de la dirección, es que la aplicación es sencilla de utilizar, no siendo necesario realizar ningún tipo de acción formativa sobre su uso.

En lo que respecta a los requisitos de concurrencia, seguridad y auditoría de los datos, partíamos del inconveniente de no poder utilizar un SGBD de tipo cliente-servidor que nos facilitara el cumplimiento de estos requisitos. No obstante, se consiguió superar estos inconvenientes como ya se ha ido explicando durante la fase de desarrollo. Resumidamente:

- La **concurrencia** es improbable (poca carga de trabajo además de que las operaciones de escritura apenas duran unos pocos milisegundos) pero es tratada si fuera necesario mediante la gestión de las excepciones.
- La **seguridad** es gestionada a partir de la iteración 2 mediante la incorporación de un sistema de copias de seguridad del archivo con los datos.
- La **auditoría** de datos se consigue mediante el registro en cada anotación de una serie de datos (código de usuario, fecha y hora del registro) de forma automática e independiente de la voluntad del usuario y que no pueden ser modificados.

Por otro lado, en el punto 1.3.4 se establecieron una serie de funcionalidades que la aplicación debía incluir. Todas esas funcionalidades han sido incorporadas a la aplicación en sus distintas versiones, tal como se puede ver en la tabla 3.

Tabla 3. Adopción de funcionalidades por versiones.

Funcionalidad	Versión en la que se incluye
Añadir registros a la base de datos.	Versión 1.1
Auditoría de datos.	Versión 1.1
Gestión de copias de seguridad.	Versión 1.2
Generador de informes en pdf.	Versión 1.2
Gestión gráfica de la base de datos.	Versión 1.3
Distintos roles de acceso.	Versión 1.4

En conclusión, el proyecto se ha desarrollado conforme a las condiciones planteadas en el inicio y la aplicación cumple con todos los requisitos y funcionalidades planteadas desde la dirección.

4 REGISTAB 2.0

Si bien la aplicación es completamente funcional, cumple con los requisitos y condiciones planteados desde la dirección y después de un uso continuado durante unos meses no se han encontrado errores significativos, hay una serie de mejoras que podrían ser implementadas en una nueva versión de la aplicación.

Por un lado, desde el punto de vista de la lógica interna de la aplicación, sería conveniente la migración a un SGBD de tipo cliente-servidor por las múltiples ventajas que este sistema ofrece (ya detalladas en el aparado 2.2). En concreto, propondría a la dirección migrar la base de datos de la aplicación a MySQL, aunque para ello es necesario solicitar un cambio en la política de seguridad informática establecida desde los servicios centrales. Además, habría que valorar las ventajas e inconvenientes de utilizar mapeador objeto-relacional (OMR) como puede ser *Hibernate*.

Por otro lado, hay una serie de funcionalidades, desde el punto de vista del usuario, que podrían ser interesantes:

- Incluir la posibilidad, directamente desde la aplicación, de escanear documentos y vincularlos con una anotación en el registro, además de que la propia aplicación se encargue de gestionar el almacenamiento de los documentos escaneados.
- Autocompletado de datos en los formularios.
- Otras funcionalidades que puedan ser propuestas por los usuarios a través del formulario de errores/incidencias incluido en la aplicación.

Todas estas cuestiones serían el punto de partida en caso de que se decidiera desde la dirección iniciar el desarrollo de una nueva versión de la aplicación, si bien, a día de hoy no se considera necesario.