Curso académico 2013/2014

APLICACIÓN DE GESTIÓN DE OFICINAS DE SERVICIOS DE ASISTENCIA LEGAL

AGOSAL

Autor: Francisco Borja Brisson Vega

Tutor: Zenón J. Hernández Figueroa

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Agradecimientos

Me gustaría aprovechar este pequeño rincón para agradecer a todas aquellas personas que han formado parte de mi vida durante estos años de lucha y esfuerzos.

En primer lugar me gustaría agradecerle a mi tutor Zenón J. Hernández Figueroa por todo el trabajo que ha hecho ayudándome, no solo en el actual proyecto sino desde que inicié este reto que empezó por 2007.

Agradecerle a mis amigos de toda la vida, el seguir ahí aún cuando a veces uno desaparece durante una temporada.

Agradecer a mis compañeros de carrera la alegría que me da poder decir que tengo amigos como compañeros y no al revés.

Agradecer especialmente el apoyo de mi familia en todos los momentos vividos, algunos buenos y otros no tanto. Especialmente a mi madre que ha estado ahí siempre que lo necesito, aunque a veces cueste verlo.

Y sobre todo a ti, mi otra mitad, gracias a ti por hacer más luminosos mis días y más cálidas las noches, gracias por no dejar que me rinda y hacer que me levante aunque solo quisiera lo contrario, gracias por darle valor al mañana.

A todos los que han formado parte de esta pequeña etapa, ¡muchas gracias!

Índice

Índice de Ilustraciones.

Índice de Tablas

Resumen

El presente proyecto se postula como una posible solución de los distintos problemas de gestión que se encuentran los profesionales del sector jurídico durante la realización su actividad laboral.

Se plantean diversas alternativas, eligiendo como solución final implantar un sistema en Nube que le permita al usuario acceder a su información en cualquier momento y lugar.

Dicha solución está constituida por cuatro componentes diferenciados (Servidor de Información, Aplicación de Escritorio, Aplicación móvil y Web), permitiendo al usuario poder utilizar el sistema en diversas plataformas. El diseño de los distintos elementos de la solución tendrá como características principales la seguridad, disponibilidad y usabilidad del sistema.

1. Introducción/Presentación del PFC:

AGOSAL (Aplicación de Gestión de Oficinas de Servicios de Asistencia Legal) es el resultado de un proyecto de fin de carrera (PFC) elaborado por Fco. Borja Brisson Vega y tutorizado por Zenón J. Hernández Figueroa. Este PFC se postula como una posible solución para los problemas administrativos y de gestión encontrados en la realización de la actividad de los profesionales del sector jurídico, como: abogados, procuradores, graduados sociales, etc.

La responsabilidad de estos profesionales es defender los intereses de sus clientes en casos asociados a una presunta infracción legal, respecto de la cual dichos clientes pueden ser denunciantes o denunciados.

Estos casos son gestionados por los profesionales mediante expedientes, que pueden albergar varios procedimientos judiciales diferentes, relacionados directa o indirectamente con el motivo inicial de la apertura del expediente.

Dichos procedimientos consisten en una secuencia de actos que referencian las normas y tramites de desarrollo realizadas durante la actividad jurisdiccional. Cada procedimiento suele tener múltiples documentos asociados, entre escritos y notificaciones, pudiendo llegar a alcanzar tamaños muy considerables.

Durante el desarrollo del procedimiento, los profesionales deberán hacer frente a diversos plazos de vencimiento, vinculados normalmente con los escritos ya mencionados, que exigen la presentación de documentación o realización de acciones.

Todo ello, junto a la escueta informatización existente, hace que la tarea administrativa del sector judicial sea muy costosa, ya que la cantidad de documentación y plazos pueden ocasionar graves problemas o incomodidades durante la realización de su actividad laboral.

Conociendo este problema, se ha planteado un proyecto con el objetivo de satisfacer los requisitos existentes mediante la utilización de un sistema informático. Este proyecto ha sido realizado aplicando las distintas etapas definidas en la metodología Métrica 3, añadiendo durante la fase de desarrollo la utilización de la metodología Ágil pseudo Scrum, versión alternativa al propio Scrum.

En esta memoria se aglutinan todos los aspectos técnicos y prácticos considerados durante la realización del proyecto. Para una mayor comprensión comenzaremos explicando los detalles más genéricos y abstractos hasta llegar a los más específicos de la solución.

En primer lugar se abordará el contexto social del problema, explicando las distintas peculiaridades existentes y los problemas a los cuales se desean dar solución. Inicialmente se estudiará el ámbito jurídico-legal actual, analizando las diversas entidades y roles existentes, así como la forma en la que desempeñan sus actividades y las restricciones legales a las que se ven afectados. Este estudio inicial concluye con la obtención de los requisitos y el planteamiento de una solución a desarrollar, tras una breve comparación con otras posibles alternativas.

Elegida la solución profundizaremos en ella, detallando las características y diseño de los distintos elementos que la componen. Comenzaremos por los componentes más influyentes en la solución, seguido por el resto de ellos. En esta fase se estudiará el diseño de cada uno, presentando varias alternativas planteadas junto a las tecnologías que lo componen. Estudiaremos en cada caso las responsabilidades y funcionalidades que deben realizar, detallando la composición y desarrollo de cada una de estas funcionalidades.

Tras mostrar los distintos componentes de la solución, abarcaremos posibles extensiones a la solución planteada, aumentando las funcionalidades diseñadas anteriormente. En este momento no solo propondremos nuevas alternativas o funcionalidades para el futuro, sino que además se contextualizará el porqué de la no realización de ciertas características instanciadas en el diseño del proyecto.

Seguidamente comenzaremos un apartado de conclusiones donde expondremos nuestra opinión global al respecto del proyecto. En esta sección realizaremos una reflexión sobre los distintos factores y características del grosso del proyecto. Centrándonos en las distintas fases realizadas, contextualizando las distintas conclusiones obtenidas en cada una de ellas.

Finalmente se presentará diversos anexos en los cuales encontraremos extensiones a las diversas explicaciones realizadas durante la memoria.

1.1. Descripción del problema

Como se ha comentado, la actividad administrativa de los profesionales del sector jurídico se fundamenta en la gestión de expedientes y de sus documentos asociados, a través de los diversos procedimientos existentes. Si generalizamos las tareas administrativas de los distintos profesionales podríamos resumirlas en el siguiente ejemplo.

"Un cliente se pone en contacto con el profesional para hacer uso de sus servicios jurídicos. Este hecho culmina en la creación de un caso, por lo que nuestro profesional instanciará un nuevo expediente. El caso se iniciará con la puesta en marcha de un primer procedimiento judicial, para el cual se deberán presentar los escritos y/o denuncias correspondientes.

Durante la realización del procedimiento, la actividad del profesional se incrementa, debido a que en este momento deberá gestionar y responder tanto las posibles notificaciones que el Juzgado solicite, como los Escritos propios que nuestro profesional necesite durante la ejecución del procedimiento. En esta etapa del procedimiento se puede alcanzar una carga administrativa importante, debido al número de documentos que pueden llegar a generarse.

Un factor a tener en cuenta, al margen de la carga procedimental, es el volumen de los propios documentos, ya que en la actualidad se realizan mayoritariamente en formato físico, dificultando la manipulación y gestión de los mismos.

Una vez finalizado el procedimiento puede suceder que se recurra la sentencia generando un nuevo Procedimiento, asociado al Expediente ya creado, y repitiendo todo el proceso de gestión descrito anteriormente. Otra posibilidad es que, debido a hechos ocurridos durante el Procedimiento actual o por la aparición de nuevas pruebas o testimonios, se instancie un nuevo Procedimiento, ajeno al ejecutado en este momento pero asociado al mismo expediente."

Además hay que recalcar otros aspectos administrativos ajenos a la propia a la actividad legislativa como tal. Por ejemplo, la gestión de contactos, administración contable y plazos de pagos del IRPF.

Contextualizada la actividad realizada por los operadores jurídicos, mediante el ejemplo presentado, explicaremos ahora formalmente los diversos conceptos y entidades existentes en el ámbito que nos encontramos. En la siguiente clasificación se mostrarán los distintos conceptos estructurados en 3 grupos principales: Entidades, Agentes y Elementos Vinculados.

a) <u>Entidades</u>:

1. Órganos Jurisdiccionales: aquellos órganos del Poder Judicial encargados de la función de juzgar y hacer ejecutar lo juzgado.

2. Jurisdicción: potestad que tiene el Estado de administrar justicia por medio de los órganos del Poder Judicial, de acuerdo con la Constitución y las leyes, por su independencia y sumisión a la Ley y al Derecho, que la soberanía nacional ha otorgado en exclusiva y, en consecuencia, expresamente les ha legitimado para la resolución jurídica, motivada, definitiva e irrevocable de los conflictos intersubjetivos y sociales, para la protección de los derechos subjetivos, el control de la legalidad y la complementación del ordenamiento jurídico.

Es la función pública, realizada por órganos competentes del Estado, con las formas requeridas por ley, en virtud de la cual, por acto de juicio, se determina el derecho de las partes, con el objeto de dirimir sus conflictos y controversias de relevancia jurídica, mediante decisiones con autoridad de cosa juzgada, eventualmente factibles de ejecución.

3. Órdenes jurisdiccionales: la organización judicial española ordinaria, se divide en cuatro órdenes jurisdiccionales:

a) **Civil**: examina los litigios cuyo conocimiento no venga expresamente atribuido a otro orden jurisdiccional. Por ello puede ser catalogado como ordinario o común.
b) **Penal**: corresponde al orden penal el conocimiento de las causas y juicios criminales.
c) **Contencioso administrativo**: trata del control de la legalidad de la actuación de las administraciones públicas y las reclamaciones de responsabilidad patrimonial que se dirijan contra las mismas.

d) **Social**: conoce de las pretensiones que se ejerciten en la rama social del Derecho, tanto en conflictos individuales entre trabajador y empresario con ocasión del contrato de trabajo, como en materia de negociación colectiva, así como las reclamaciones en materia de Seguridad Social o contra el Estado cuando le atribuya responsabilidad la legislación laboral.

e) Además de los cuatro órdenes jurisdiccionales, existe en España la **Jurisdicción Militar**.

4. **Partido Judicial**: demarcación judicial procedente de la unión de varios municipios limítrofes de una misma provincia. La determinación de los municipios que integran cada partido corresponde, a la Ley de Demarcación y Planta judicial

5. **Órganos unipersonales: Juzgados**: Son aquellos órganos en los que el personal jurisdicente consiste en una sola persona, un único juez y secretario (personal no jurisdicente).

6. **Órganos colegiados: Tribunal**: Son aquellos órganos jurisdiccionales cuyo personal jurisdicente lo integran varias personas que actúan colegiadamente, es decir, órganos jurisdiccionales cuyo titular es un colegio de jueces.

7. **Secretaría u oficina judicial**: Entidad calificada por la ley como unidad procesal de apoyo directo que existente en cada Juzgado y cuya principal responsabilidad es la revisión y validación de los diferentes eventos producidos durante un procedimiento judicial (notificaciones, escritos, autos, etc.).

b) <u>Agentes</u>:

1. **Juez / Magistrado**: personas que ejercen la función jurisdiccional de juzgar y hacer ejecutar lo juzgado.
2. **Secretario**: personal no jurisdicente en los órganos jurisdiccionales para ejercer tareas instrumentales. Son funcionarios públicos integrantes del

sistema judicial independientes al personal jurisdicente, con funciones propias como, que actúan como ministros de fe pública en los juzgados.

3. **Unidades procesal de apoyo directo y servicios comunes procesales**: unidades compuestas por funcionarios de carrera y pertenecientes a cuerpos nacionales que prestan sus servicios en los Juzgados y Tribunales.

4. **Abogado**: Licenciado o doctor en derecho que ejerce profesionalmente la dirección y defensa de las partes en toda clase de procesos o el asesoramiento y consejo jurídico.

5. Fiscal: órgano de relevancia constitucional y con personalidad jurídica propia integrado con autonomía funcional en el Poder Judicial, cuyo cometido es promover la acción de la Justicia en defensa de la legalidad, de los derechos de los ciudadanos y del interés público tutelado por la Ley, de oficio o a petición de los interesados, así como velar por la independencia de los Tribunales y procurar ante ellos la satisfacción del interés social.

El Ministerio Fiscal es un órgano único para todo el Estado y sus miembros son autoridad a todos los efectos, actuando siempre en representación de toda la Institución. Ejerce sus funciones por medio de órganos propios conforme a los principios de unidad de actuación y dependencia jerárquica y con sujeción, en todo caso, a los de legalidad e imparcialidad.

6. **Procurador**: Profesional del derecho que, en virtud de apoderamiento, ejerce ante juzgados y tribunales la representación procesal de cada parte.
7. **Graduado Social**: profesional que, estando en posesión del título oficial correspondiente y debidamente colegiado, asesora en materia laboral y fiscal y gestiona las relaciones laborales y la Seguridad Social de las empresas, trabajadores y ciudadanos en general, y les representa técnicamente ante la Administración y los Tribunales de Justicia.
8. **Cliente**: Persona física o jurídica que hace uso de los servicios legales ofrecidos por cualquiera de los profesionales que operan en el ámbito jurídico.

c) <u>Elementos Vinculados</u>:

1. **Procedimientos**: es concebido doctrinalmente como la forma en que se concreta la actividad jurisdiccional, constituye el elemento dinámico del proceso.

Hace referencia a las normas de desarrollo, de ritualidad, tramitación o formalidades para su realización. Están constituidos por la combinación y coordinación de varios actos jurídicos que, siendo procesalmente autónomos, tienen por objeto la producción del efecto jurídico final del propio proceso. Por ello, en su aspecto externo, aparece como una sucesión temporal de actos, donde cada uno de ellos es presupuesto del siguiente y condición de eficacia del anterior.

2. **Expedientes**: entidad administrativa utilizada para identificar los diversos servicios que, para su realización, sea necesaria su prolongación en el tiempo. Normalmente en parte de sus identificadores se refleja el marco temporal en el que se instanció.

   Generalmente dentro de un Expediente nos encontramos con un solo procedimiento. Sin embargo, es común que dentro del mismo existan varios procedimientos cuando se suscitan cuestiones secundarias o accesorias al asunto principal, cuya tramitación. En este caso, cada cuestión secundaria o incidente dará origen a un procedimiento distinto al principal, aunque dentro de un mismo proceso.

3. **Escritos judiciales**: solicitud escrita presentada ante un juzgado o tribunal. Existe una multitud de tipos de Escritos, diferenciados por el tipo de procedimiento y solicitud. La buena elaboración, argumentación y documentación es primordial ya que el Juez se basará en este documento para pronunciarse a favor o en contra de la solicitud realizada.

4. **Resoluciones Judiciales – Notificaciones** (autos, providencias, decretos y diligencias): acto de comunicación emitido desde un juzgado o tribunal. Este documento debe ser entregado a la persona o ser publicado a través de un edicto para que el destinatario conozca el lugar, la fecha y la hora en que debe presentarse a prestar declaración o intervenir por una causa judicial.

5. **Demanda**: acto procesal por el que se inicia un proceso, y se solicita del órgano jurisdiccional frente al demandado una tutela jurídica en forma de

sentencia favorable, mediante un escrito en el que expone los antecedentes del hecho del caso y sus razonamientos jurídicos.

6. **Denuncia**: noticia o aviso que se da a una autoridad de un delito o una acción que va contra la ley, o de su autor.

Además de conocer el ámbito de trabajo y de las actividades que lo componen, otro factor a tener en cuenta son las peculiaridades de los posibles usuarios finales del sistema a crear.

Este hecho se tuvo en consideración durante la fase de estudio, por lo que concretaron varias reuniones donde su principal objetivo era recabar información en este sentido. Por ello se diseñaron varias preguntas, escenarios y actividades que permitieran vislumbrar las características más intrínsecas del usuario.

Una vez recopilada toda la información de los distintos profesionales entrevistados, se realizó una tarea de análisis donde abstraer los requisitos generales existentes en los usuarios de este sector. Las características obtenidas tras este análisis, tomadas como base para la definición de requisitos, son:

1. ***Formación escasa en informática***: Tras el estudio se observó una formación muy superficial sobre conceptos y uso de la informática. Este hecho incluso se potenció al escuchar varios hechos sucedidos a compañeros de los entrevistados. Una vez conocido este hecho se focalizó en encontrar el por qué, lo cual nos llevó a varias conclusiones.
    a. Los profesionales de este gremio forman parte de la rama de Humanidades, donde el contacto con la informática no es muy extenso.
    b. Otro factor a considerar es la edad generacional mayoritaria del sector. En este punto se observó que las generaciones conocidas como “tecnológicas” son un pequeño porcentaje. En cambio, las personas entre 40-58 años forman el grupo mayoritario.

Estas conclusiones serán de gran importancia en el diseño del sistema, ya que deberá ser muy intuitivo y con la menor configuración posible de cara al usuario.

2. ***Difícil adaptabilidad***: En las reuniones donde se presentaron diversos diseños de la interfaz gráfica apreciamos un prediseño muy arraigado en los expertos.

Debido a su experiencia en Windows, los usuarios tienen muy interiorizado la estructuración de las aplicaciones en este entorno, por lo que nuestro diseño final deberá seguir estas pautas para una mayor aceptación.

3. ***Restricciones legales.*** Durante la explicación de varios escenarios se pudo apreciar como el usuario debía adaptar su forma de trabajar para satisfacer ciertos requisitos, procedentes de la legislación actual. Concretamente de la Ley Orgánica 15/1999 de Protección de Datos (***LOPD***), la cual obliga a todos profesionales del ámbito jurídico a realizar la gestión de la información sensible dentro de un marco controlado y seguro descrito por la ley. Esta ley no solo afecta en la administración de la información, sino también en su acceso y en la notificación de la propia, ya que los dueños de esta deberán tener conocimiento de su existencia y uso. En el apartado *1.2. Requisitos determinados por la LOPD.* se detallarán, en mayor profundidad, las características y restricciones de la LOPD.

1.2. Requisitos determinados por la LOPD.

Ley Orgánica 15/1999 de Protección de Datos de carácter personal (LOPD), publicada el 13 de Diciembre de 1999, tiene como objetivo principal regular el tratamiento de los datos y ficheros de carácter personal, independientemente del soporte en el cual sean tratados, así como los derechos de los ciudadanos sobre ellos y las obligaciones de aquellos que los crean o tratan.

Antes de hacer mención de las restricciones y obligaciones que imponen el cumplimiento de la LOPD, es necesario conocer diversos conceptos que se utiliza durante la definición de la misma.

A). ***Datos de carácter personal:*** cualquier información concerniente a personas físicas identificadas o identificables.

B). ***Fichero:*** todo conjunto organizado de datos de carácter personal, cualquiera que fuere la forma o modalidad de su creación, almacenamiento, organización y acceso.

C). ***Tratamiento de datos:*** operaciones y procedimientos técnicos de carácter automatizado o no, que permitan la recogida, grabación, conservación, elaboración, modificación, bloqueo y cancelación, así como las cesiones de

datos que resulten de comunicaciones, consultas, interconexiones y transferencias.

D). ***Responsable del fichero o tratamiento:*** persona física o jurídica, de naturaleza pública o privada, u órgano administrativo, que decida sobre la finalidad, contenido y uso del tratamiento.

E). ***Afectado o interesado:*** persona física titular de los datos que sean objeto del tratamiento a que se refiere el apartado c) del presente artículo.

F). ***Procedimiento de disociación:*** todo tratamiento de datos personales de modo que la información que se obtenga no pueda asociarse a persona identificada o identificable.

G). ***Encargado del tratamiento:*** la persona física o jurídica, autoridad pública, servicio o cualquier otro organismo que, solo o conjuntamente con otros, trate datos personales por cuenta del responsable del tratamiento.

H). ***Consentimiento del interesado:*** toda manifestación de voluntad, libre, inequívoca, específica e informada, mediante la que el interesado consienta el tratamiento de datos personales que le conciernen.

I). ***Cesión o comunicación de datos:*** toda revelación de datos realizada a una persona distinta del interesado.

Observando los conceptos encontramos la existencia de 3 tipos de roles (el *Afectado*, el *Responsable del fichero* y el *Encargado del tratamiento*). El conocimiento de estos roles es de vital importancia, ya que deberá identificarse a quienes se les atribuye durante la inscripción del fichero de datos de carácter personal, registrado en la Agencia de Protección de Datos (APD).

Este fichero, de obligado cumplimiento para el registro de la actividad, indicará el *Responsable del fichero*, la finalidad del mismo, su ubicación, el tipo de datos de carácter personal que contiene, las medidas de seguridad, con indicación del nivel básico, medio o alto exigible y el *Encargado del tratamiento* de los datos, entre otras cosas.

Si analizamos estos roles, concretamente el *Responsable del fichero* y el *Encargado del tratamiento*, podemos encontrar una concordancia entre ellos y nuestros usuarios y nosotros mismos. Los *Responsables del fichero* estarán formados por los usuarios de nuestro sistema, los cuales decidirán sobre la finalidad y contenido de los datos recolectados. En cambio, el rol de *Encargado del tratamiento* recaerá

sobre el administrador del sistema a crear, el cual proporcionará todas las operaciones necesarias para el tratamiento de los datos.

Por lo tanto no solo se deberá estudiar qué impedimentos poseen los operadores jurídicos, sino que además debemos analizar que restricciones y controles se nos atribuyen como *Encargados del tratamiento*. Debido a ello se dividirá el estudio en dos bloques: Restricciones referidas al Responsable del fichero y Restricciones asociadas al Encargado del tratamiento.

1.2.1. Restricciones referidas al Responsable del fichero

En este bloque se detallarán aquellas características más relevantes respecto al papel que desempeña el *Responsable del fichero* en la gestión de los datos, haciendo hincapié en las propias peculiaridades del ámbito en el que nos encontramos.

A continuación se presentarán aquellas obligaciones consideradas relevantes para la elaboración del proyecto, indicando el artículo y sección en el que se expresan.

a) Los datos de carácter personal solo podrán ser recabados siempre y cuando tengan relación con el ámbito y las finalidades determinadas en el registro realizado en la Agencia de Protección de Datos. **Artículo 4, sección 1.**

b) “Los datos de carácter personal objeto de tratamiento no podrán usarse para finalidades incompatibles con aquellas para las que los datos hubieran sido recogidos. No se considerará incompatible el tratamiento posterior de éstos con fines históricos, estadísticos o científicos.” **Artículo 4, sección 2.**

c) “Los datos de carácter personal serán exactos y puestos al día de forma que respondan como veracidad a la situación actual del afectado.” **Artículo 4, sección 3.**

d) “Los datos de carácter personal serán cancelados cuando hayan dejado de ser necesarios o pertinentes para la finalidad para la cual hubieran sido recabados o registrados”. **Artículo 4, sección 5.**

e) Los Interesados a los que se soliciten datos personales deberán ser previamente informados de modo expreso, preciso e inequívoco de la existencia del fichero, así como la finalidad de la recogida de datos y los destinatarios de estos. Además deberá ser informado de los derechos de acceso, rectificación y cancelación que posee, junto a la identidad y dirección del Responsable del fichero. **Artículo 5.**

f) Cualquier persona que intervenga en cualquier fase del tratamiento de datos está obligada al secreto profesional respecto de los mismos y al deber de guardarlos. Obligaciones que subsistirán aun después de finalizar sus relaciones con el titular del fichero o, en su caso, con el responsable del mismo. **Artículo 10.**

g) El interesado tendrá derecho a solicitar y obtener gratuitamente información de sus datos de carácter personal sometidos a tratamiento, el origen de dichos datos, así como las comunicaciones realizadas o que se prevén hacer de los mismos. El derecho de acceso a que se refiere este artículo sólo podrá ser ejercitado a intervalos no inferiores a doce meses, salvo que el interesado acredite un interés legítimo al efecto. **Artículo 15.**

h) El responsable del tratamiento tendrá la obligación de hacer efectivo el derecho de rectificación o cancelación del interesado en el plazo de diez días. La cancelación dará lugar al bloqueo de los datos, conservándose únicamente a disposición de las Administraciones públicas, Jueces y Tribunales, para la atención de las posibles responsabilidades nacidas del tratamiento, durante el plazo de prescripción de éstas. Cumplido el citado plazo deberá procederse a la supresión. **Artículo 16.**

1.2.2. Restricciones asociadas al Encargado del tratamiento.

En esta sección se abordarán las particularidades y obligaciones que deberá cumplir todo Encargado del tratamiento, centrándonos especialmente en aquellas particularidades relacionadas con el entorno a simular. Como en el bloque anterior, se indicará el artículo y sección de la ley en la que se definen.

- “El responsable del fichero, y, en su caso, el ***encargado del tratamiento*** deberán adoptar las medidas de índole técnica y organizativas necesarias que garantice la seguridad de los datos de carácter personal y evite su alteración, pérdida, tratamiento o acceso no autorizado, habida cuenta del estado de la tecnología, la naturaleza de los datos almacenados y los riesgos a que están expuestos, ya provengan de la acción humana del medio físico o natural.” **Artículo 9, sección 1.**

- “No se registrarán datos de carácter personal en ficheros que no reúnan las condiciones que se determinen por vía reglamentaria con respecto a su integridad y seguridad y a las de los centros de tratamiento, locales, equipos, sistemas y programas”. **Artículo 9, sección 2.**

- Cualquier persona que intervenga en cualquier fase del tratamiento de datos está obligada al secreto profesional respecto de los mismos y al deber de guardarlos. Obligaciones que subsistirán aun después de finalizar sus relaciones con el titular del fichero o, en su caso, con el responsable del mismo. **Artículo 10.**

- “Los datos de carácter personal objeto del tratamiento sólo podrán ser comunicados a un tercero para el cumplimiento de fines directamente relacionados con las funciones legítimas del cedente y del cesionario con el previo consentimiento del interesado”. **Artículo 11, sección 1.**

- “No se considerará comunicación de datos el acceso de un tercero a los datos cuando dicho acceso sea necesario para la prestación de un servicio al responsable del tratamiento”. **Artículo 12, sección 1.**

- “La realización de tratamientos por cuenta de terceros deberá estar regulada en un contrato que deberá constar por escrito o en alguna otra forma que permita acreditar su celebración y contenido, estableciéndose expresamente que el encargado del tratamiento únicamente tratará los datos conforme a las instrucciones del responsable del tratamiento, que no los aplicará o utilizará con el fin distinto al que figure en dicho contrato, ni los comunicará, ni siquiera para su conservación, a otras personas. En el contrato se estipularán,

asimismo, las medidas de seguridad a que se refiere el artículo 9 de esta Ley que el encargado del tratamiento está obligado a implementar". **Artículo 12, sección 2.**

- "Una vez cumplida la prestación contractual, los datos de carácter personal deberán ser destruidos o devueltos al responsable del tratamiento, al igual que cualquier soporte o documentos en que conste algún dato de carácter personal objeto del tratamiento". **Artículo 12, sección 3.**

- En el caso de que el encargado del tratamiento *destine los datos a otra finalidad*, los comunique o los utilice incumpliendo las estipulaciones del contrato, será considerado también **responsable del tratamiento**, respondiendo de las infracciones en que hubiera incurrido personalmente". **Artículo 12, sección 4.**

1.3. Requisitos finales del sistema.

Tras analizar los distintos elementos del problema a tratar se ha elaborado un listado de requisitos, de mayor a menor prioridad, que deberá satisfacer toda solución a plantear.

1. Cumplimiento de la Ley Orgánica 15/1999 de Protección de Datos (LOPD).
2. Alta seguridad del sistema.
3. Cubrimiento de todos los requisitos de información necesarios para la gestión de procedimientos judiciales.
4. Alta disponibilidad del sistema.
5. Buen sistema de respaldo.
6. Facilidad de uso.
7. Alta fluidez de las aplicaciones durante su ejecución.
8. Buena integración de la agenda.
9. Buen diseño de la interfaz gráfica.
10. Buen control de la compartición de datos.
11. Diversidad de dispositivos.
12. Fácil mantenimiento.

1.4. Soluciones Propuestas.

Considerando la información obtenida durante esta fase, se ha elaborado dos posibles sistemas que satisfagan todos los requisitos y necesidades de los usuarios finales.

Solución A: Aplicación gestora de información.

Aplicación gráfica de escritorio que permitirá el manejo y edición de la información de forma sencilla y cómoda. Esta solución le facilitará al usuario la gestión de expedientes, permitiéndole búsquedas selectivas y de mayor profundidad. Además incorporará una agenda con la que el usuario podrá gestionar los distintos eventos que desee. El acceso a esta información estará protegido por credenciales, por ejemplo una contraseña, por lo que cumplirá la LOPD.

Otro aspecto importante es que realizará copias automáticas, o indicadas por el usuario, permitiéndole al profesional restaurar en el punto temporal que desee.

- Pros
  1. Satisface los requisitos tanto personales como legales indicados.
  2. Rápida implementación. En unos pocos meses podrá alcanzarse un nivel suficientemente elaborado para comenzar con la fase beta del proyecto.
  3. Requisitos tecnológicos muy escasos.

- Contras
  1. La escasa formación tecnológica por parte del gremio hace un tanto problemática el uso automático de la copia de seguridad.
  2. La información sigue encerrada en el despacho del usuario, pero en estado digital.
  3. Posible riesgo en el mantenimiento de las base de datos al estar gestionadas por el usuario. Es posible que en el futuro se realice algún cambio en la estructura, por lo que el usuario podría utilizar procedimientos o copias de seguridad incompatibles con este nuevo diseño.

Solución B: Sistema de información basado en la nube.

Esta solución está formada por un sistema en nube, constituido por dos componentes principales. Un servidor de información, que contendrá toda la base de

datos del ámbito a emular y un cliente de conexión, que será el software utilizado por el usuario para acceder a su información. Además de la propia administración de expedientes, este sistema permitirá la gestión temporal al disponer de una agenda y la comunicación entre usuarios del sistema, ya que este nuevo paradigma permitirá la existencia de múltiples usuarios.

Para satisfacer la LOPD se utilizará un acceso basado en credenciales. Además todas las conexiones de los clientes se harán de forma segura (encriptadas) y se automatizarán las copias de seguridad del sistema.

- Pros.
    1. Satisface los requisitos tanto personales como legales indicados.
    2. Permite la comunicación entre usuarios, reduciendo aún más las distancias entre compañeros.
    3. Mayor flexibilidad de uso. Los usuarios tendrán varias alternativas para acceder a la información, y sobre todo, actualizada en cada momento.
    4. Se permite poder acceder a la información fuera del área de trabajo.
    5. La gestión y configuración de las copias de seguridad son totalmente transparentes para el usuario.
    6. Este nuevo paradigma le permite al usuario una mayor flexibilidad en el cambio de infraestructuras, ya que solo necesitará instalar el cliente en su nuevo equipo.
    7. Se logra un sistema de mayor adaptabilidad, se aumenta la facilidad de implementación de futuros paradigmas tecnológicos.
- Contras.
    1. Tiempo de realización de todo el sistema medio/alto.
    2. El acceso multiusuario del sistema hace más influyente la disponibilidad del mismo.
    3. Posibilidad de ataques al sistema.
    4. Es necesario un control más alto en el diseño de los distintos componentes, ya que la existencia de varios elementos hace más probable la aparición de errores.
    5. Uso de hardware específico para el componente servidor del sistema.

1.5. Solución Elegida.

La solución elegida es el sistema de información basado en la nube. Esta elección se ha hecho considerando varios aspectos desde la funcionalidad final para el usuario, como su alta flexibilidad y su facilidad de mantenimiento. Por todo ello, pensamos que este caso proporcionará un sistema más completo al usuario, permitiéndole una mayor funcionalidad y flexibilidad de uso.

A continuación se explicará en más detalle los distintos componentes que constituyen el proyecto a desarrollar, junto a las responsabilidades que se les atribuyen.

Como se indicó anteriormente, esta solución está formada por dos componentes funcionales básicos: ***Servidor de Información*** y el ***Cliente de Acceso***.

Ilustración 1. Estructura inicial de la solución selecionada

Este Cliente de Acceso estará constituido por diversos agentes que permitirán la utilización del sistema a través de diferentes plataformas, aumentando la flexibilidad y capacidad del sistema.

Por lo tanto, el sistema final basado en la nube estará constituido por 4 componentes bien definidos, los cuales son: ***Servidor de Información***, ***Aplicación de Escritorio***, ***Aplicación Móvil*** y ***Aplicación Web***. En el siguiente diagrama se podrá apreciar la estructura jerárquica del sistema en cuanto a la gestión de información se refiere, junto al flujo de información existente entre ellos.

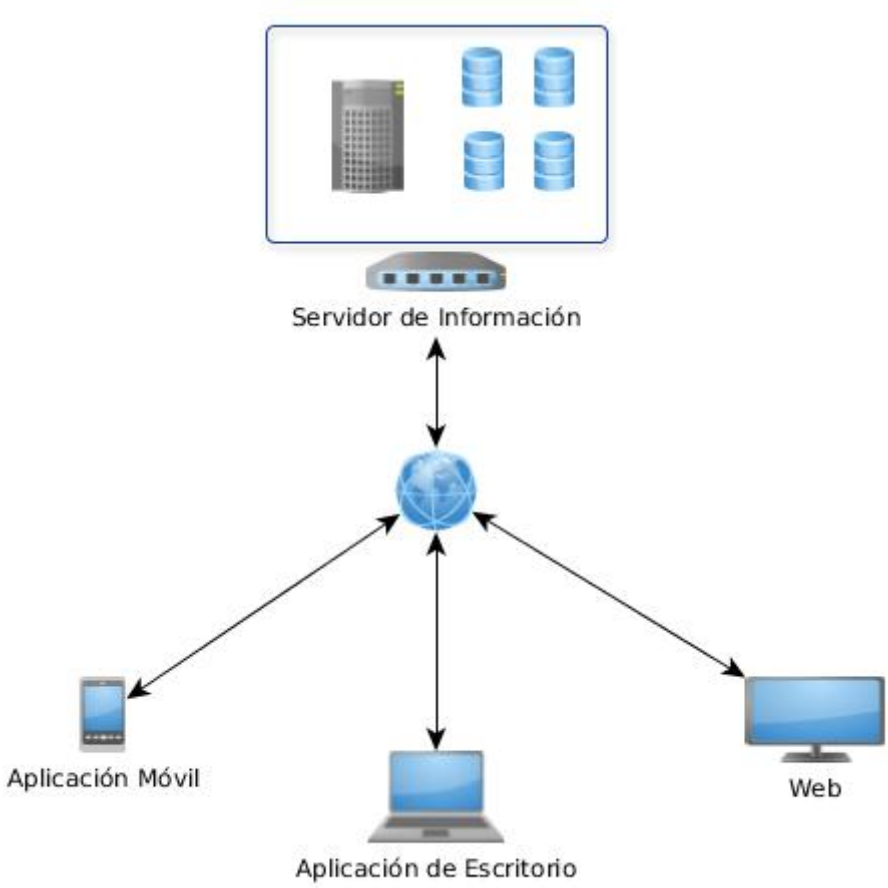

Ilustración 2. Estructura final de la solución elegida.

Servidor de información

Este componente será el pilar principal en el que se fundamenta todo el sistema de información. Estará formado por un ***servidor*** físico, que albergará toda la información almacenada por los usuarios, mediante un Sistema Gestor de Bases de Datos (SGBD). Aparte realizará, de forma segura, la comunicación con los distintos componentes, gestionando las transacciones de información según la demanda. Además posee la responsabilidad de realizar la copia de seguridad del sistema, gestionando toda la información que ello conlleva.

Debido a su rol en el sistema deberá poseer un mecanismo de seguridad que preserve la integridad del sistema.

Responsabilidades.

1. Almacenamiento y gestión de datos (BD)
2. Automatización de copias de seguridad.
3. Establecer un sistema seguro.
4. Alta disponibilidad.
5. Dar soporte para la realización de conexiones seguras (SSL).

Aplicación de Escritorio.

Subsistema constituido por una aplicación de escritorio ejecutada tanto en equipos de sobremesa cómo portátiles. Con esta aplicación se podrá acceder a la información almacenada en el servidor y permitirá modificarla de forma transparente. Además permitirá gestionar la agenda del usuario con sus notificaciones y comunicarse con el resto de usuarios.

Los datos serán transmitidos mediante comunicaciones seguras, y en ningún momento se guardará ningún dato de forma local en el equipo, excepto ciertos registros de log del propio sistema.

La aplicación deberá ser dinámica y enfocada para el tipo de usuario descrito. Tendrá que seguir una estructura uniforme en las distintas plataformas existentes y presentará toda la información a mostrar en un formato estructurado, facilitando la legibilidad de la misma.

Responsabilidades.

1. Ofrecer los distintos servicios del sistema.
2. Conexión segura.
3. Facilidad de uso.

Aplicación móvil

Diseñado para los dispositivos móviles como SmartPhone y Tablets. Con este sistema se podrá hacer uso de los distintos servicios que ofrece el sistema, tal y como

en el componente anterior. En este caso la información estará estructurada en un formato que permita explotar toda la potencia de los dispositivos móviles.

Los datos serán transmitidos mediante comunicaciones seguras, y en ningún momento se guardarán de forma local en el equipo.

Responsabilidades.

1. Ofrecer los distintos servicios del sistema.
2. Conexión segura.
3. Facilidad de uso.

Web

En este caso se trata de un sistema WEB que permitirá a los usuarios poder acceder a su información y a los servicios del sistema desde cualquier equipo. Seguirá los patrones de diseño utilizados en la actualidad, buscando una solución flexible e intuitiva.

Como en los componentes anteriores, permitirá acceder tanto a la información del usuario como a la agenda, mensajería, etc. Así mismo, las comunicaciones se harán de forma segura, utilizando en este caso el protocolo HTTPS.

Responsabilidades.

1. Ofrecer los distintos servicios del sistema.
2. Conexión segura.
3. Facilidad de uso.

2. Servidor de Información.

El servidor de información constituye la piedra angular del sistema, siendo el nexo de unión entre los distintos elementos que elaboran la solución, dando soporte a un gran número de usuarios.

Se encarga de suministrar a los diversos dispositivos toda la información necesaria para poder realizar sus tareas. Además, les proporciona una API de comunicación para el tratamiento de la información, eliminado de estos componentes la responsabilidad de manipular los datos correctamente.

El gran número de accesos a los que debe atender y a la propia sensibilidad de la información a tratar, conlleva que la disponibilidad y seguridad del sistema adquieren una vital importancia, sobre todo esta última, al vernos influidos por la LOPD.

Analizando todos los requisitos que debe cumplir el componente, podemos apreciar la existencia de 4 módulos independientes, tal y como se muestra en la figura Ilustración 3.

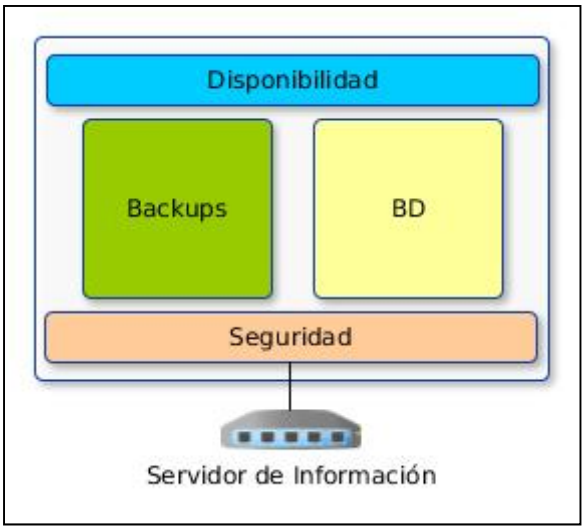

Ilustración 3. Composición del Servidor de Información

Estará formado por un servidor físico que dará soporte a los distintos servicios del sistema. En este caso se ha decidido utilizar la plataforma Linux como Sistema Operativo (S.O.) base, debido a la gran facilidad de configuración que ofrece, además

de poseer un gran número de servicios a nuestra disposición. Concretamente se ha elegido la distribución Ubuntu Server 12.04, al tratarse de una versión LTS.

Teniendo a Ubuntu como base, se estudiaron los distintos servicios que pueden agregarse al sistema para satisfacer las necesidades funcionales de los distintos módulos diseñados. Tras este análisis se decidió utilizar varias herramientas propias de Linux junto a la incorporación de algunos sistemas externos. La tecnología a utilizar es:

2.1. Tecnología empleada.

MySQL.

A día de hoy es uno de los Sistemas Gestores de Bases de Datos (SGBD) más populares y utilizados en la comunidad informática. Elegimos este componente por su alto rendimiento y estabilidad, además de ofrecer licencia de Software Libre.

Es capaz de soportar un gran número de accesos simultáneos, y nos ofrece múltiples opciones de configuración en diversas áreas como: encriptación de comunicación, soporte para copias de seguridad y restricción de accesos a los datos.

A. ***Encriptación de comunicación:***

MySQL nos permite configurar las conexiones utilizando el protocolo de cifrado SSL. Para ello necesitaremos un certificado digital, a compartir con el resto de clientes, y una clave privada que solo conocerá el servidor.

También podemos indicar que los usuarios estén obligados a utilizar conexiones seguras, lo que impedirá que se establezca alguna conexión sin cifrar. Además se le puede asociar a cada usuario un certificado digital único, que debe presentar siempre que quiera establecer la conexión, lo que nos garantiza que el remitente es quien dice ser.

En este caso no utilizaremos esta última opción, debido a que complicaría el diseño y funcionamiento de nuestro componente Móvil, tal y cómo se detallará más adelante. En cambio, sí obligaremos que las conexiones de todos los usuarios deban realizarse mediante el protocolo SSL, eliminando la posibilidad de que cualquier transmisión pueda producirse de forma inteligible.

B. *Soporte para copias de seguridad (Backups):*

Podemos configurar el SGBD para que generen unos ficheros de registro, llamados ficheros binarios de logs, donde se notificará cualquier acción finalizada con éxito. Estos ficheros permiten poder acceder al estado de la BD en cualquier instante registrado, por lo que podrán ser utilizados cómo copias de seguridad.

Los ficheros de logs están estructurados en forma binaria, con un formato exclusivo de la plataforma, logrando una gran compactación de la información. Por ello MySQL ofrece herramientas que permiten adquirir las diversas sentencias SQL que lo forman, además de poder acceder al estado del sistema en cualquier marco temporal registrado.

Otra de sus funcionalidades es la capacidad de reiniciar el contenido de estos ficheros en cualquier momento, lo que ayudará en la administración de las copias incrementales, como se mostrará en la sección *2.2.3. Backup.*

Al margen de la utilización de ficheros binarios, MySQL ofrece otros recursos para la gestión de backups. Por ejemplo, da soporte para obtener el estado actual de todo el sistema o de BD específicas, lo que nos permitirá generar estados completos del sistema.

C. *Restricción de accesos a los datos.*

Una de las características de administración de usuarios que posee es la de asignarle a cada uno el ámbito de trabajo al que tiene acceso. En la definición de privilegios podemos indicar a que BD o a que tablas de la misma puede acceder, e incluso restringir las acciones que puede realizar.

Además también haremos uso de su sistema de acreditación de usuarios, ya que permitiremos que sea el propio MySQL quien valide las credenciales indicadas por el usuario. Con ello nos evitamos tener que realizar esta gestión y sobre todo tener que crear un usuario genérico para la edición de datos, lo que produciría una disminución de la seguridad debido a que este usuario tendría acceso a los datos de todo el sistema, por lo que la existencia de una vulnerabilidad por esta parte haría caer la consistencia de todo el sistema.

Otros de los motivos de su elección, es la facilidad de integración en sistemas Linux. La expansión que tiene en la comunidad hace que existan diversos conectores, estructuras y sistemas ya implementados que nos faciliten su utilización.

Por otro lado, MySQL ofrece el uso de ***Stored Procedure***, procedimientos almacenados que permiten poder llevar a cabo tareas de manipulación de datos de forma totalmente transparente al usuario. Con ello eliminamos del resto de componentes la responsabilidad de editar los datos, ya que únicamente deberán hacer uso de estos procedimientos.

IPtables.

Se trata de un cortafuego perteneciente al framework Netfilter, disponible en el núcleo Linux de, prácticamente, cualquier distribución. Iptables permite una rápida y sencilla gestión del firewall del equipo mediante el filtro de paquetes, control de conexiones o traducción de direcciones de red (NAT).

Por ello se ha elegido como cortafuego del sistema, ya que utilizando este cortafuego gratuito conseguiremos el nivel de seguridad necesario para nuestro servidor sin incrementar el coste de su construcción.

SELinux.

Security-Enhanced Linux (**SELinux**) es un módulo de seguridad incorporado en Linux que proporciona el soporte de políticas de seguridad para el control de acceso.

Nos permitirá restringir el acceso de los usuarios a directorios así como, controlar que servicios podrán hacer uso de los recursos disponibles.

RSync.

Rsync es una aplicación libre para sistemas de tipo Unix y Microsoft Windows que ofrece transmisión y sincronización eficiente de datos incrementales, pudiendo operar además con datos comprimidos y cifrados.

Se ha elegido esta aplicación ya que nos permite sincronizar archivos y directorios entre dos máquinas de una red o entre dos directorios en una misma máquina, lo cual combinado con la utilización de los ficheros binarios de log de MySQL

y el uso de ciertos scripts nos permitirá poder llevar a cabo la realización de copias de seguridad.

También será de utilidad cuando se desee replicar las copias de seguridad en otras máquinas alejadas, debido a su capacidad de compresión y el establecimiento de comunicaciones cifradas.

CronTab.

Es un administrador de procesos en segundo plano incorporado en el sistema operativo Unix. Este servicio permite gestionar la ejecución de diversas tareas en marcos temporales concretos.

La utilización de Cron junto a Rsync y los ficheros de log de MySQL nos permitirá automatizar la creación y administración de los backups necesarios para garantizar la robustez del sistema.

2.2. Módulos del Subsistema.

A continuación se detallarán todos los aspectos y configuraciones realizadas para la construcción de los distintos módulos presentados en la imagen Ilustración 3.

2.2.1. Definición de las Bases de Datos (BD).

En este módulo se almacenará todo lo referente a la configuración y diseño del modelo de datos que representará al ámbito descrito, siendo el encargado además, de disponer todos los elementos necesarios para la edición de la información por parte del resto de subsistemas.

Para la creación del modelo de datos fueron necesarias varias reuniones con los profesionales donde nos centramos en la búsqueda de todas las entidades, elementos y restricciones existentes en la realización de su trabajo. Este proceso no fue momentáneo sino que se distribuyó en el tiempo, a modo de iteraciones, hasta que se obtuvo una descripción del sistema lo suficientemente consistente y flexible como para dar por finalizado esta etapa.

Una vez generado el modelo de datos de la actividad jurídica nos centramos en fusionarlo con nuestros modelos prediseñados del resto de funcionalidades del sistema, como la mensajería y la agenda. Finalizada la unión observamos que el

modelo final podría dividirse en dos grandes bloques, uno general que compartían todos los usuarios y otro específico para cada uno.

Debido a ello se decidió separar ambos modelos en dos tipos de bases de datos diferenciadas, una llamada *Agosal*, albergando toda la estructura necesaria para la realización de las características generales encontradas, y una base de datos por usuario utilizada para la información específica de cada uno. El nombre de estas BD seguirán el patrón *agosal_user_XXX*, donde XXX hace referencia al código numérico con el que el sistema identifica a cada usuario.

La utilización de 2 bases de datos nos permite incorporar un mayor control de acceso ya que podremos restringir la operatoria de cada usuario a su propia BD y a las tablas indicadas de la BD general, aumentando la seguridad y sobre todo, mejorando la administración de recursos del sistema.

Modelo de datos.

Para facilitar la comprensión del diseño de las bases de datos construidas, se dividirán las distintas entidades existentes en las bases de datos en la que se encuentra. Además se agruparán por la funcionalidad que representan, mostrando las relaciones y conceptos que conllevan.

BD propia del usuario:

a. *Contactos*:

Como se muestra en el diagrama de Entidad/Relación, todos aquellos conceptos que sean considerados como entidad de Contacto serán derivados de la entidad Supertipo. A partir de esta clase principal obtienen 3 grupos: Clientes, Abogados y Procuradores.

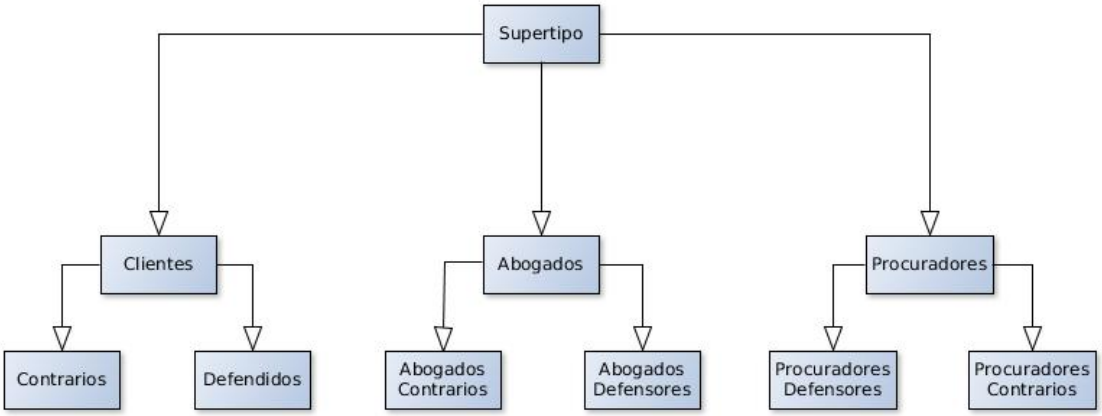

Ilustración 4. Diagrama E/R referente a los Contactos.

1. **Supertipo (SuperType)**: Hace referencia, de forma general, a cualquier elemento que pueda utilizarse para la identificación de una persona física o entidad legal.
2. **Clientes (Clients)**: Representa aquellas personas físicas, legales o jurídicas que realizan uso de los servicios judiciales de un experto jurídico.
3. **Abogados (Lawyers)**: Contendrá todos aquellos licenciados en derecho colegiados en el algún Colegio de Abogados del territorio español.
4. **Procuradores (Attorneys)**: Personas que desarrollan la actividad laboral de un Procurador de los Tribunales.

Además de la propia utilización como Contacto, se utilizarán estas entidades para la identificación de personalidades en el resto de componentes.

Cada uno de las subentidades resultantes, a partir del Supertipo, serán divididas en Contrarios y Defensores, utilizando como criterio para su clasificación el lugar que ocupen en la realización del procedimiento, teniendo como perspectiva el lado del usuario.

b. *Expedientes*:

1. **Expediente (Files)**: Unidad/Entidad mínima administrativa utilizada por el usuario para el gestión de la actividad laboral realizada o a realizar. Estos expedientes son referenciados por un identificador único, especificable por el usuario. Normalmente se utiliza el identificador "AÑO_de_creación/Número_de_tareas_realizadas", aunque algunos

profesionales pueden incluir leves matices como el trimestre o mes de su creación.

2. **Procedimientos (Judgments)**: Hacen referencia a los procedimientos judiciales resultantes de la actividad principal del Expediente asociado. Debido a esta relación, todo Procedimiento no puede estar asociado a más de un Expediente.

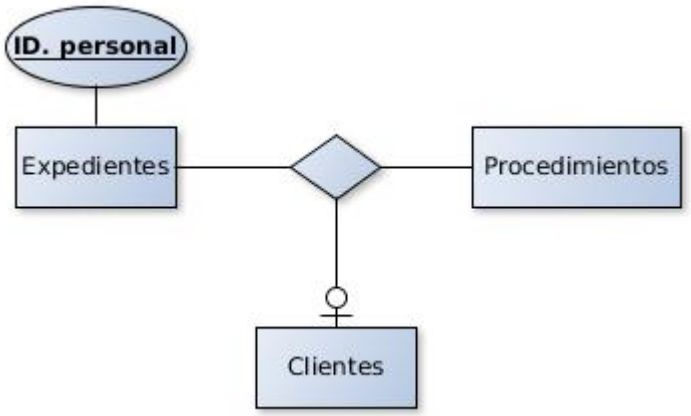

Ilustración 5. Diagrama E/R representativo de los Expedientes

c. *Procedimientos*:

1. **Recurso (Appeal)**: Refiere al acto de apelación de la sentencia de un Procedimiento previo, produciendo la creación de un nuevo Procedimiento a asociado al anterior
2. **Escritos (associatedDocuments)**: Documentos asociados a la actividad de un Procedimiento. Pueden estar relacionados tanto con Escritos judiciales como a Resoluciones o Notificaciones judiciales.
3. **Procedimiento Normal**: Procedimiento judicial realizado directamente por el servicio solicitado por el Cliente. No es derivado de un Procedimiento anterior.
4. **Procedimiento Recurso**: Procedimiento derivado o resultante por la realización de uno previo. Este tipo de Procedimiento se producen cuando se recurre la sentencia de un Procedimiento Normal.
5. **Defendidos (Defense)**: Dupla formada por el Cliente, Abogado Def., Procurador Def. y Procedimiento.
6. **Defendido (Opposite)**: Cliente que hace uso de los servicios del usuario.
7. **Abogado Defensor**: El usuario o abogado asignado a la misma parte de este.

8. **Procurador Defensor**: El usuario o procurador asignado a la misma parte de este.
9. **Contrarios**: Dupla formada por el Contrario, Abogado Cont., Procurador Cont. y Procedimiento.
10. **Contrario**: Antagonista al cliente del usuario.
11. **Abogado Contrario**: Abogado asignado al procedimiento por la parte contraria del usuario.
12. **Procurador Contrario**: Procurador asignado al procedimiento por la parte contraria del usuario.

El resto de entidades, a pesar de ser utilizadas para esta funcionalidad, se explicarán en la sección reservada para los elementos alojados en la BD general del sistema.

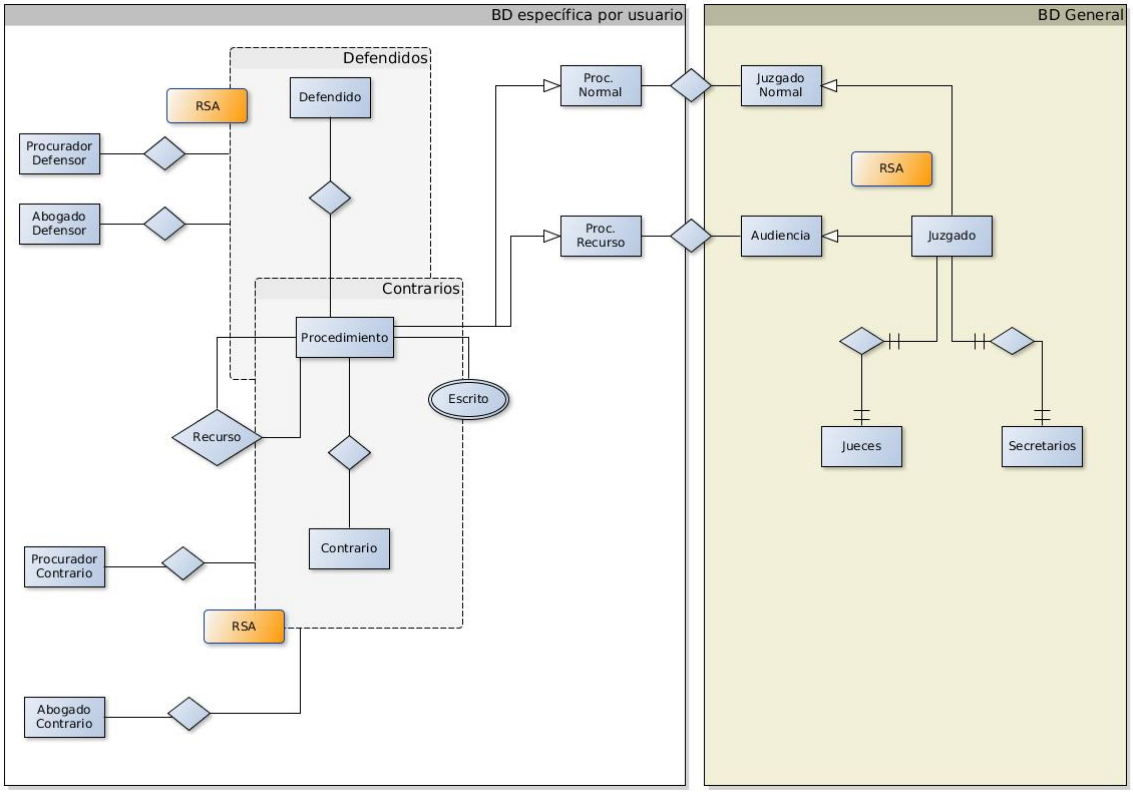

Ilustración 6. Diagrama E/R representativo de la actividad judicial

En este diagrama encontramos 3 RSA (Restricciones Semánticas Adicionales) las cuales deberán controlarse por mecanismos ajenos a la propia estructura SQL. Realmente existen 2 RSA, debido a que las existentes en las entidades Defendidos y Contrarios forman parte de la misma restricción global. Las RSA encontradas son:

- **RSA 1**: Debe garantizarse que ninguno de los miembros de la dupla Contrarios formen parte de los Defendidos para un mismo procedimiento.
- **RSA 2**: No puede permitirse que un Juzgado pertenezca tanto al grupo de Audiencia como a Juzgado Normal.

d. *Calendario*:

1. **Nota (Notebook)**: Refleja una cita o recordatorio asociado a una fecha y hora. Además podrá estar vinculado con la actividad de un Procedimiento existente.
2. **Comentario (Comment)**: Descripción de una observación específica atribuida a una nota determinada.

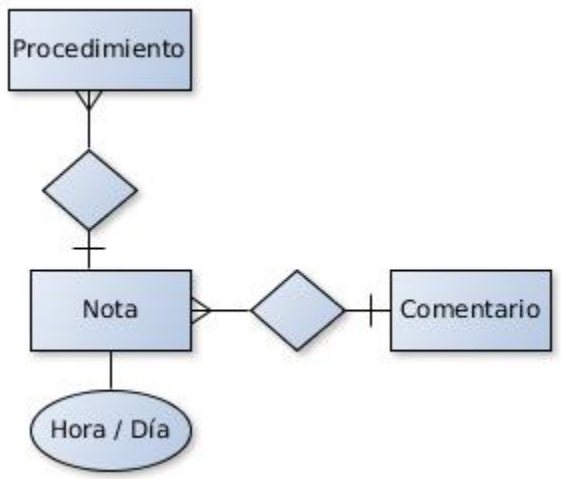

Ilustración 7. Diagrama E/R representativo del calendario

BD general:

a. *Mensajería*:

1. **Mensajes (Message)**: Constituye la comunicación entre el usuario remitente y un destinatario perteneciente al sistema. En esta entidad se aglutina el contenido de la comunicación, el remitente, el asunto del mensaje y fecha de realización.
2. **Notificación de Mensajes (Notify)**: Hace la similitud del buzón de correos. Esta entidad contendrá todas las notificaciones pendientes y atendidas por el usuario.
3. **Usuarios (Users)**: Hace referencia a todos los usuarios registrados del sistema. Además del nombre propio del usuario posee un identificador

único que será utilizado, entre otras cosas, para nombrar la BD específica de cada uno.

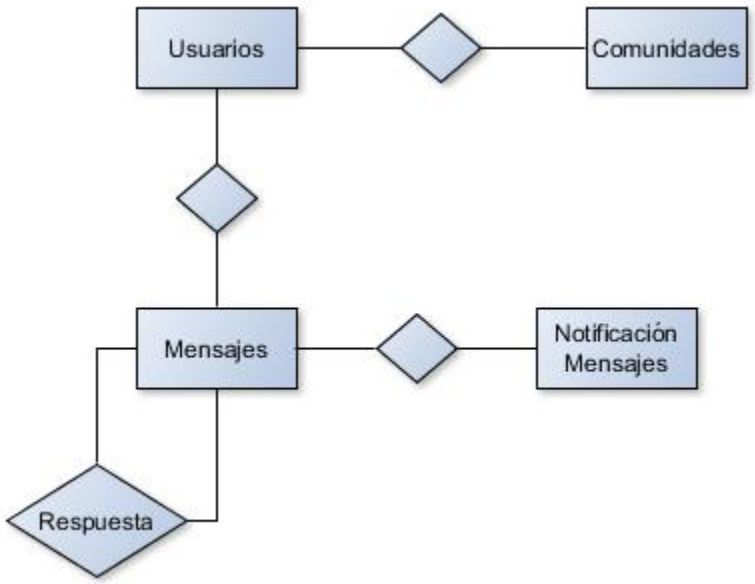

Ilustración 8. Diagrama E/R representativo de la mensajería.

b. *Uso General*:

1. **Órdenes Jurisdiccionales**: Refleja los tipos de órdenes existentes en la legislación española.
2. **Tipo de Juzgados (courtsType)**: Representa los diferentes tipos de Órganos Judiciales, como el caso Juzgados y Tribunales.
3. **Tipo de procedimientos (judgmentType)**: Simboliza los distintos tipos de procedimientos existentes, derivados del tipo de orden jurisdiccional al que pertenecen.
4. **Comunidad (Community)**: Alude a las comunidades autónomas del territorio español, utilizada para localizar distintos conceptos de la BD, como al usuario o el Juzgado.
5. **Jueces (Judges)**: Identifica a los Jueces del territorio español asignados a un Juzgado en particular.
6. **Secretarios (Secretaries)**: Personaliza los funcionarios que llevan a cabo la actividad de Secretario/a en los Juzgados españoles.
7. **Juzgados (Courts)**: Hace referencia a los órganos judiciales, de cualquier índole, en los que se realiza cualquier actividad procedimental. Estos órganos están asociados a la comunidad autónoma en la que se encuentra.

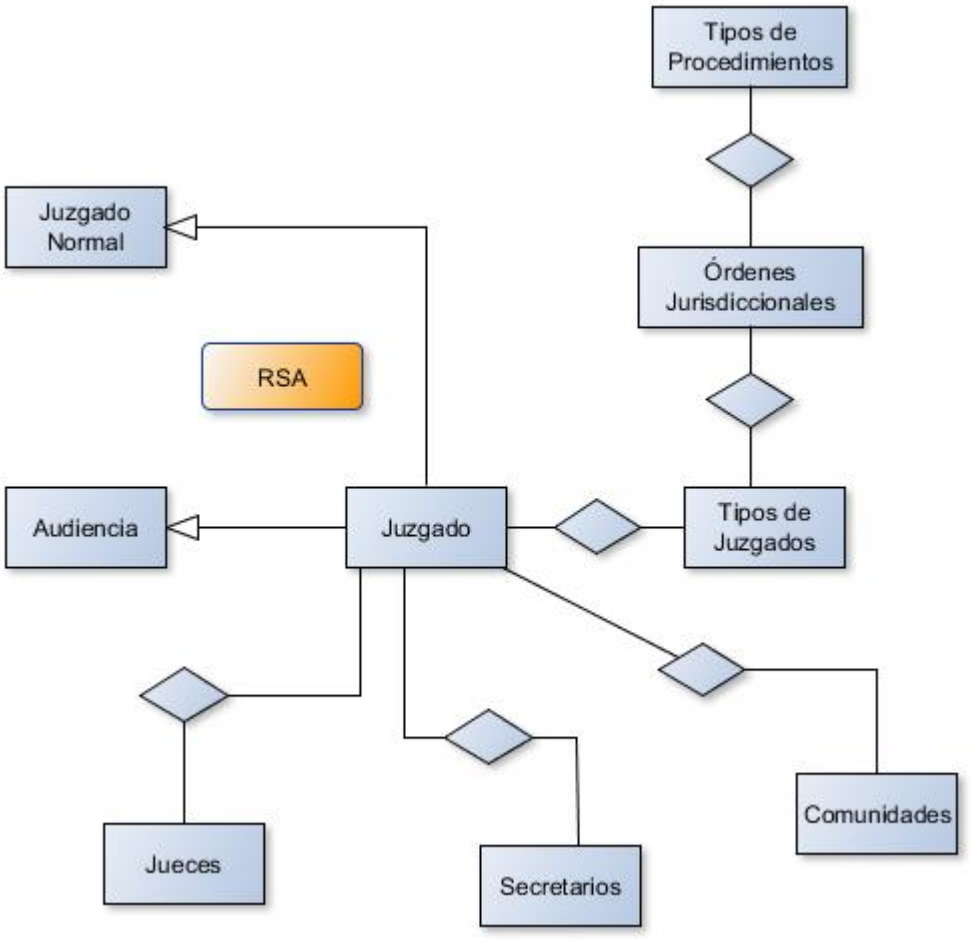

Ilustración 9.Diagrama E/R representativo de la estructura jurídica.

Para la edición de los datos de las bases de datos se decidió utilizar la herramienta Stored Procedures (SP) ofrecida por MySQL. Consiste en la ejecución de procedimientos almacenados a los cuales se les suministra un número de parámetros predefinidos, tanto en número como en tipo. Cada vez que se lanza un procedimiento se evalúa la concordancia de los parámetros enviados y se le traslada el control a las instrucciones que el desarrollador haya introducido en el procedimiento.

Con esto conseguimos abstraer al resto de componentes toda la estructura interna que tiene nuestro modelo de datos, eliminando en estos la responsabilidad de gestionar la correcta manipulación de la información. También logramos reducir la sensibilidad a cambios del sistema, ya que el resto de componentes solo conocen una API de edición, permitiéndonos cambiar la ejecución de los SP sin repercutir en el funcionamiento del sistema.

Una peculiaridad de utilizar una BD específica por usuario es que debemos replicar cualquier cambio en los SP en todas las BD creadas, perjudicando la gestión

de los mismos. Aún así, este tipo de diseño facilita el desarrollo de los propios SP ya que permite centrarnos en tareas individuales más sencillas y de un menor grado de gestión.

A continuación se presentarán los distintos *Stored Procedure* definidos para la manipulación del sistema, describiendo la funcionalidad que realizan.

2.2. Stored Procedures.

En esta sección se definirán los SP que constituirán el módulo de edición de las bases de datos diseñadas. Para una mayor claridad, se agruparán los procedimientos mediante la BD en la que se encuentran y por la entidad a la que hacen referencia.

BD específica para el usuario.

	Procedimientos	Funcionalidad
Expedientes	new_File edit_File delete_File close_File	Permitirán la creación, edición, borrado y cierre de expedientes, respectivamente.
Procedimientos	new_Judgment edit_Judgment delete_Judgment close_Judgment	Llevan a cabo la manipulación de los procedimientos.
	add_appeal_Judgment	Realiza la creación del recurso, crea un nuevo procedimiento asociado inicialmente a otro existente
	associate_appeal_Judgment	Le asocia a un procedimiento ya existente otro procedimiento como recurso de este.
Contrarios	add_Opposite delete_Opposite	Añaden y eliminar la dupla que forman los Contrarios del Procedimiento
Defensa	add_Defense delete_Defense	Añaden y eliminan la dupla que forman los Defendidos del Procedimiento.

	Procedimientos	Funcionalidad
Escritos	new_typeWrits edit_typeWrits delete_typeWrits:	Gestionan la creación, edicion y borrado de Escritos.
	new_associatedDocuments edit_associatedDocuments delete_associatedDocuments	Realizan la creación, edición y borrado de los tipos de Escritos.
Contactos	new_Company edit_Company delete_Company	Llevan a cabo la inserción, edición y borrado de los contactos de tipo Empresa.
	new_Lawyers edit_Lawyers delete_Lawyers	Idéntico al caso anterior pero orientado a los contactos del tipo Abogado.
	new_Attorneys edit_Attorneys delete_Attorneys	Idéntico al caso anterior pero orientado a los contactos del tipo Procurador.
	new_Clients edit_Clients delete_Clients	Idéntico al caso anterior pero orientado a los contactos del tipo Cliente.
Calendario	new_note edit_note delete_note	Se encargan de la gestión de recordatorios, permitiendo la creación, edición y borrado de los mismos.
	add_commentNote edit_commentNote delete_commentNote	Permiten tanto añadir como editar o borrar comentarios a las notas ya realizadas.

Tabla 1. Procedimientos asociados a las Entidades alojadas en la BD específica.

BD general.

	Procedimientos	Funcionalidad
Mensajería	new_message reply_message delete_message	Realizan las operaciones de envío, respuesta y borrado de mensaje.
	read_message	Indica que el usuario ha leído el mensaje.
	notify_message	Avisa que el usuario ha sido notificado del envío del mensaje.

Tabla 2. Procedimientos asociados a las Entidades alojadas en la BD general

2.2.2. Seguridad.

Es el encargado de garantizar el nivel de seguridad del sistema completo. Aunque esté enfocado para el servidor, debido a la importancia de la seguridad en nuestro entorno, es necesario ampliar su ámbito incluyendo al resto de componentes, por lo que podríamos decir que se trata de un módulo global.

Debido a la trascendencia que posee, es necesario saber qué requisitos debe cumplir para satisfacer todos los requerimientos de seguridad preestablecidos. Por ello se mostrará a continuación las responsabilidades que se le han otorgado a este módulo.

1. Todo usuario solo tendrá acceso a su propia información o, en el caso de existir, información compartida por otro usuario.
2. Cualquier acceso hacia el sistema deberá seguir ciertos protocolos de seguridad que garanticen que la comunicación no será inteligible por ningún intermediario malicioso.
3. El sistema deberá ser capaz de soportar el ataque malicioso de terceras personas, haciendo hincapié en las vulnerabilidades existentes en el lenguaje SQL.
4. El acceso físico al sistema deberá estar restringido por el personal autorizado y siguiendo unos pasos bien definidos.
5. En caso de caer ante un ataque de cualquier tipo, el sistema debe ser capaz de reducir al mínimo el daño ocasionado por dicho ataque.

Para poder satisfacer los requisitos es necesario construir una serie de servicios que le suministren al módulo toda la funcionalidad requerida. Por ello se analizaron las distintas tecnologías existentes en el mercado, buscando aquellos servicios que permitan construir un módulo que satisfaga los requisitos indicados, sin olvidar el mantenimiento y administración de los mismos. Los servicios resultantes de este estudio fueron: *Encriptación de comunicaciones, Restricción/Control de acceso y Protección ante ataques maliciosos.*

- ***Encriptación de comunicaciones.***

Ofrecerá la capacidad de establecer comunicaciones encriptadas mediante la utilización del protocolo TLS. Para ello se hará uso de la posibilidad que proporciona MySQL para establecer todas sus comunicaciones de forma segura. Este tipo de comunicación requiere que se modifique en los Subsistemas de acceso el modo de conexión, ya que en este momento deben de habilitar el protocolo TLS. El uso de este método hace que se tenga que crear en el Servidor un certificado digital que deberán conocer el resto de componentes y una clave privada con la que el Servidor se identificará durante la comunicación.

Para el caso de la página web o el WebService, como se verá más adelante, es necesario realizar este mismo proceso pero en la configuración de Apache, permitiéndonos así utilizar el protocolo HTTPS. En este caso debemos seguir los mismos pasos que antes, en cuanto a la utilización del certificado digital se refiere. Aunque teóricamente podríamos utilizar el mismo certificado es conveniente tener un certificado digital único por cada servicio suministrado.

- ***Restricción/Control de acceso:***

Este servicio está diseñado para gestionar el acceso a los distintos niveles que presenta el sistema: el acceso a la información y el acceso al propio sistema.

  - *Control de acceso a la información:*

Gestionamos el acceso a la información del sistema impidiendo que el usuario pueda acceder a información perteneciente a otros usuarios. Para ello hacemos uso de las restricciones de ámbitos y de acciones que ofrece MySQL. Con ellas indicamos que tablas y bases de datos tiene acceso el usuario, así como las acciones que puede realizar con ellas.

Además se decidió implantar un diseño de BD dividida donde cada usuario poseerá una BD propia, con sus datos privados, y otra BD a compartir con el resto de usuarios, en la cual se compartirán distintos registros de información global como el caso de la mensajería o la información referentes a los Juzgados.

La utilización de este patrón de BD nos facilitará la restricción de acceso a cada usuario, ya que toda la información relevante estará alojada en cada BD independiente. También facilitará la creación del modelo de datos al tratar las distintas entidades y relaciones de forma individual, eliminando la gestión de multiusuario en el proceso.

Por otro lado reduciremos la capacidad de cambios del modelo de datos, debido a que los cambios, tanto de estructura como de los SP, deberán ser propagados por todas las BD de los usuarios.

- *Control de acceso al sistema.*

El control del acceso al sistema es un proceso de vital importancia en un entorno de producción debido a que debe garantizar que solo tendrán acceso los administradores del sistema pero sin restringir su utilización.

Antes de explicar la configuración de acceso y el protocolo de actuación, debemos distinguir entre el acceso físico del equipo y el acceso virtual o lógico. El acceso físico consiste en manipular físicamente la propia máquina. El acceso lógico se trata de acceder al sistema operativo de la máquina, permitiéndonos configurar las distintas características de este.

Si se ha decidido utilizar una máquina propia como servidor, y no la alternativa de utilizar servicios de alojamiento, debemos establecer un protocolo de actuación si se desea manipular de cualquier forma la máquina física, ya sea un apagado, reseteo o comprobación.

Al margen de todas las características de seguridad indicadas anteriormente, debemos de utilizar un registro de acceso donde cada persona que desee acceder al sistema tendrá que firmar en el registro indicando: nombre, fecha, hora y motivo del acceso. A su vez, cuando salga del recinto controlado deberá repetir la notificación pero esta vez indicando: nombre, fecha, hora, Identificador del acceso previo.

Cada uno de los registros, tanto de entrada como de salida, tendrá un identificador único que puede consistir en una secuencia de números.

Si utilizamos un servicio de alojamiento no podremos acceder a la máquina física, pero sí podremos actuar sobre ella mediante las herramientas que nos ofrece la empresa en la que nos alojemos. Debido a la sensibilidad de su manipulación, se seguirá el mismo protocolo que en el caso físico cuando se desee gestionar el estado de la máquina, no el acceso a ella.

Respecto al acceso lógico, utilizaremos el protocolo SSH que nos permite acceder a maquinas remotas de forma encriptada. Para asegurarnos que se encuentra controlado su acceso eliminaremos la posibilidad de acceder al equipo desde el exterior mediante SSH, rechazando mediante IPTables cualquier acceso al puerto 22 desde el exterior. Solo permitiremos este acceso a equipos concretos dentro de nuestra red local.

Este diseño nos obliga a trasladarnos al área de trabajo siempre que deseáramos modificar el sistema. Si realmente fuera necesario acceder remotamente desde el exterior, podríamos tener una máquina al margen con la que nos conectamos y desde esta máquina realizar la conexión al propio servidor, y realizar las operaciones oportunas.

A pesar de facilitar la gestión del sistema y de ofrecer 2 niveles para poder acceder al sistema, no se recomienda este uso debido a que reduciría la seguridad del sistema, al tener una puerta trasera.

Una vez accedido al sistema, se restringirá el acceso a los distintos directorios y servicios, haciendo uso de la gestión de privilegios que ofrece Linux junto al servicio SELinux. Esta configuración será utilizará con el resto de elementos de seguridad para reducir el posible daño ante un acceso no permitido.

- ***Protección ante ataques maliciosos.***

    - *Defensa ante ataques contra el sistema.*

El disponer de un elemento accesible desde el exterior, desde internet, hace que el sistema pueda ser atacado explotando cualquier vulnerabilidad que posea. Para disminuir este peligro se han deshabilitado todos los servicios del sistema excepto los

explícitamente requeridos. Además mediante la configuración del cortafuego IPTables se rechazará el acceso a cualquier puerto de la máquina, a excepción de los puertos 22(SSH), 3306(MySQL), 80(HTTP) y 446(HTTPS).

Actualmente se ha incorporado el servidor Web en el mismo equipo que el Servidor de Información por falta de recursos. Lo ideal sería que cada servicio se encontrase en una máquina independiente, ya sea física o virtual. En el apartado *6.3. Sistema de alta* disponibilidad, de la sección 6. Perspectivas Futuras, se explicará una configuración alternativa que nos garantizará un mayor grado de seguridad y disponibilidad.

A pesar de todo, hay que considerar que cualquier sistema puede ser vulnerable, por lo que debemos configurar al sistema para que el daño producido sea el menor posible en el caso de que un atacante acceda.

Por ello se configurarán los permisos de los usuarios para restringir el acceso a los distintos directorios, haciendo hincapié en los relacionados con el funcionamiento del sistema. También se restringirá el arranque y parada de servicios a usuarios específicos, evitando que las acciones que pueda realizar un intruso con sus credenciales sean las menores posibles. Aún así no hay que olvidar que no existe el sistema perfecto y que todos tendrán vulnerabilidades a descubrir, por lo que es recomendable revisar cada 6 meses los sistemas de seguridad y buscar en la comunidad informática nuevos sistemas o diseños que puedan mejorar nuestra seguridad.

Un posible ataque ante el cual nuestro sistema actual sería vulnerable es la denegación de servicios (DDoS). Este tipo de ataque consiste en enviar un número tan alto de peticiones que nuestro sistema no sería capaz de gestionar, produciendo que el sistema no pueda atender las peticiones de nuestros usuarios reales, o incluso, producir la caída del sistema.

Estos ataque se realizan principalmente para suplantación de identidad, el atacante sustituiría nuestra IP en uno de sus equipos e intentaría atacar a nuestros usuarios. Otra motivo es para buscar alguna vulnerabilidad en nuestro sistema y así poder acceder o, menos frecuente, para simplemente probar tácticas de ataques que utilizarán en ataques a sistemas de mayor nivel.

Para evitar la suplantación de identidad utilizamos los certificados de seguridad, los cuales además de permitirnos la encriptación de datos nos garantizan la identificación del equipo. En el caso que el sistema cayera ante una suplantación, el resto de componentes no realizarían ninguna acción ya que este sistema no posee los certificados oportunos ni las claves privadas de estos.

Una posible solución para este tipo de ataque se encuentra al final del apartado *6.3. Sistema de alta* disponibilidad, mencionado anteriormente.

- *Defensa ante ataques Inyección SQL.*

Debido a las características del sistema no podemos controlar las solicitudes enviadas al servidor directamente, sería necesario filtrar las peticiones en un componente o módulo previo que realizara dicha comprobación.

Por ello se ha introducido en cada Cliente de Acceso, una etapa intermedia donde se revisa que las consultas SQL se encuentren libre de cualquier posible ataque de inyección SQL. En el caso del componente Móvil es el WebService, que se explicará más adelante, quien realiza esta revisión, introduciendo ese elemento intermedio ya mencionado.

La utilización del WebService, o elementos de similar funcionalidad, serían de gran utilidad para la mejora de la seguridad, sobre todo al respecto de ataques SQL o denegación de servicios. Inicialmente no se ha podido introducir al WebService como capa de abstracción global, al no disponer de suficientes herramientas para su utilización en todos los subsistemas, como en el caso del sistema de Escritorio. Por ello se ha decidido trasladar su desarrollo a una fase posterior, convirtiéndose en una de las posibles ampliaciones del sistema.

2.2.3. Backup.

Este módulo llevará a cabo toda la gestión y planificación para la correcta realización de las copias de seguridad del sistema. Para ello, se hará uso de distintas tecnologías que, coordinadas entre sí, permitirán crear un sistema automatizado que satisfaga todas nuestras necesidades.

Antes de describir la estructura del módulo analizaremos los requisitos que debe satisfacer el sistema de respaldo.

1. Debe garantizar que los datos obtenidos tras el backup corresponden con los reales, asegurando su no corrupción.
2. El proceso de backup debe ser capaz de recuperarse ante caídas del sistema. Además tendrá que asegurar la correcta ejecución de sus tareas.
3. Debido a la sensibilidad de la información a tratar, se debe generar varias copias del sistema, a poder ser en ubicaciones distintas, para disminuir el riesgo de corrupción o accidente físico.
4. La realización de las copias de seguridad y su administración no deben afectar al comportamiento ni disponibilidad del sistema.
5. Debe dar soporte para albergar las copias realizadas en lugares distantes al centro de datos.
6. El módulo debe seguir un plan de backups periódico.

Si analizamos los requisitos mostrados, podemos apreciar la necesidad de instanciar ciertos servicios que le proporcionen al sistema de respaldo toda la capacidad requerida para poder llevarlos a cabo. Los servicios diseñados para la construcción del módulo de Backups son:

1. ***Automatización***: Llevado a cabo por el servicio Cron suministrado por Linux. Con este servicio indicaremos la temporización y momentos concretos en los que se deberá realizar las distintas acciones programadas. Estas acciones consistirán en la sincronización de copias como en la administración de las mismas.

2. ***Registro de acciones***: Este servicio se llevará a cabo mediante la utilización de los ficheros binarios de log suministrados por MySQL. En estos ficheros registrarán toda la actividad realizada en el sistema.

3. ***Sincronización de ficheros***: Siempre existirá como mínimo 2 copias de los ficheros de logs, los originales y otra ubicada en el directorio /Ag_backups/año-semana_del_año, actualizada mediante el uso de Rsync. Además con Rsync podemos tener actualizado, de forma automatizada, el estado del sistema en otro equipo ajeno a este.

4. ***Copias y gestión de directorios***: Las copias, no sincronización, como la administración de procesos y directorios se realiza mediante el uso de scripts de

líneas de comando, haciendo uso de la gran potencia que presenta el terminal Bash de Linux.

El primer paso en el desarrollo de este módulo es seleccionar el modelo de copia de seguridad que se desea implementar, pudiendo elegir entre: incremental, diferencial, CDP, completa, completa sintético, etc. Antes decantarnos por un tipo comentaremos brevemente las características de cada modelo.

- ***Completa***: Consiste en generar una imagen completa del estado actual del sistema. La ventaja de su utilización es que permite disponer de la totalidad de la información en una única entidad. Ello permite que los tiempos de restauración sean mínimos pero obliga a una mayor capacidad de almacenamiento. Además el proceso de realización es mucho mayor que en otros modelos, pudiendo obligar al bloqueo del sistema para su realización.

- ***Incremental***: Sólo registra los datos modificados desde la última operación de backup, de cualquier tipo. Lo bueno de este modelo es que solo almacena los cambios producidos desde la anterior copia, ya sea otra incremental o completa, por lo que requiere un almacenamiento muy inferior a las copias completas. Para realizar su función se suele utilizar la fecha y hora de su realización, sabiendo así el punto final de la anterior copia. Este tipo de backup ofrece un tiempo menor para su realización que las copias completas, permitiendo además un mayor número de puntos de restauración.

- ***Diferencial***: Se trata de una derivación del modelo incremental. El backup diferencial registra todas las modificaciones realizadas desde la última copia completa del sistema. Como se puede apreciar, la diferencia entre el modelo incremental y diferencial es el punto de partida considerado. Cada ejecución de un backup diferencial almacenarán todos los datos diferentes a la anterior copia completa realizada, se hayan hecho o no, copias anteriormente de cualquier tipo. Aunque cada copia individualmente produzca un tamaño de almacenamiento mayor que en el caso incremental, la capacidad de aglutinar varias copias incrementales en una única copia diferencial hace que faciliten mucho más la gestión del proceso.

- ***Completa sintética***: Restaura la imagen del sistema completo utilizando modelos incrementales o diferenciales.

- ***Protección de datos continua (CDP)***: Este tipo de backup no solo almacena los datos en sí, si no todos los cambios que se realizan sobre ellos junto al momento de su realización. Este tipo de copias son las más completas de todas ya que no solo se obtiene el estado del sistema sino que además permite conocer que acciones se realizaron, en que momento y, según la configuración, hasta quien lo realizó. El aspecto negativo de este modelo es que requiere un espacio mayor a las copias completas, debido a que almacena tanto los datos como toda la información de su edición.

Para la elección del modelo de respaldo se tuvo en cuenta tanto la robustez del modelo cómo la facilidad de administración de la misma. Por ello no se decidió utilizar un único modelo, sino combinar varios de ellos, aprovechando las mejores características de cada uno y así obtener un sistema de mayor calidad.

El resultado de este modelo híbrido consiste en un sistema de respaldo elaborado por la colaboración de 4 modelos de backups, concretamente: el modelo completo, diferencial, incremental y CDP. El sistema parte de una copia completa mensual, o inicial en la puesta en marcha del sistema. Tras esto se utilizará los ficheros de log de MySQL para obtener un CDP, que a su vez se utilizará para generar una copia diferencial de la actividad semanal realizada. Es decir, generaremos un backup diferencial que se irá incrementando a lo largo de la semana y al final de esta, cerraremos la copia diferencial copiando y reiniciando los ficheros de log generado. Con ello conseguiremos toda la potencia de un sistema CDP pero con la facilidad de gestión de los sistemas diferenciales. Además, la copia y reinicio del registro de log nos deriva en un proceso incremental, ya que los datos de cada semana serán independientes al resto.

La combinación del modelo diferencial e incremental hace que se facilite la tarea administrativa sin incurrir en copias voluminosas. Con el modelo diferencial evitaremos tener un gran número de ficheros y el tipo incremental eliminará la posibilidad de tener copias diferenciales de gran tamaño.

A todo ello se le suma una capa más de seguridad para garantizar que se dispone de una imagen del sistema lo más cercana posible al estado real. Para ello se automatiza un proceso de sincronización cada 25 minutos donde se actualizará el contenido de los ficheros de log en otro directorio de la máquina, atribuido a la semana que corresponde tal y como se indicará posteriormente. Esta sincronización también podríamos realizarla con otras máquinas alejadas, aunque no se dispone en el sistema actual.

Además lanzamos otro proceso de control cada 5 horas, en el cual se copiarán, no sincronizar, todos los ficheros de log en una carpeta identificativa. Cada una de estas copias auxiliares se archiva en carpetas con la fecha del día, alojadas en el subdirectorio temp y a su vez en la carpeta asociada a la semana actual. La nomenclatura utilizada para estas copias es:

/Ag_backups/año-semana_del_año/temp/día/hora:minutos

Con este nuevo nivel garantizamos que, en un hipotético fallo, se posea una copia adicional del estado de la máquina hace 25 minutos, o en el peor de los casos, hace 5 horas.

En la figura Ilustración 10 se refleja el comportamiento y temporalidad de las distintas fases que posee el sistema de respaldo.

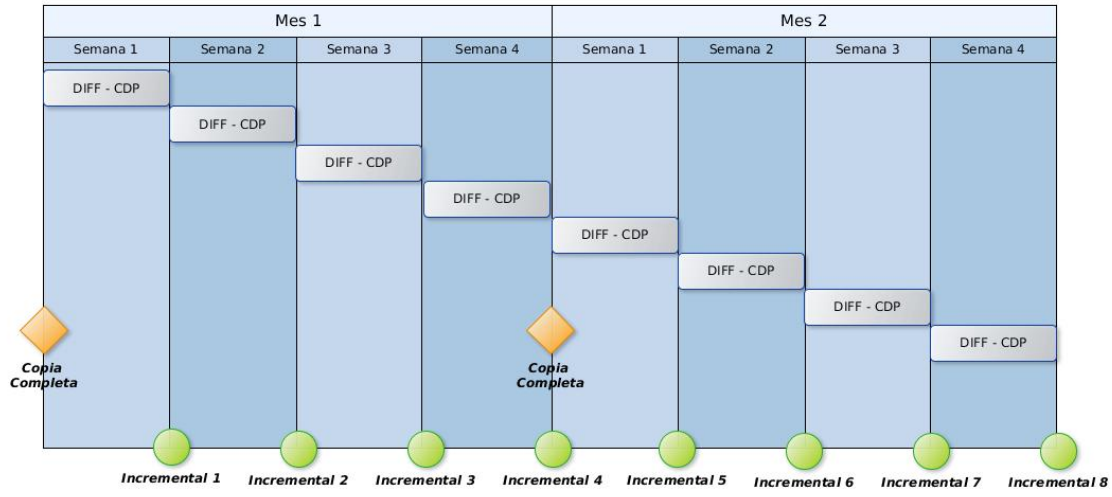

Ilustración 10. Planificación temporal del servicio Backup

Para facilitar la gestión de directorios y de nombres se ha utilizado una clasificación diseñada para diferenciar los distintos tipos de copias. Todas las copias

almacenadas se encontrarán en el directorio raíz */Ag_backups*. Dentro de este directorio encontraremos 3 tipos de elementos: carpeta *save*, copia completa del estado actual del sistema y registro histórico.

1. **carpeta *save*:** En esta carpeta se almacenarán todas las copias globales que se consideren importantes, por ejemplo la copia inicial o una anual.

2. **copia actual del sistema:** Fichero con la extensión .bck que contendrá el estado global del sistema obtenido en la última copia completa. El nombre de este fichero seguirá el patrón *Agosal.currentState.Año-mes*.

3. **registro histórico:** Se trata de un conjunto de carpetas utilizadas para almacenar todas las copias de seguridad realizadas semanal y diariamente. En este caso siguen el patrón *Año-Semana_del_año*. Dentro de estas carpetas se alojan tanto la imagen semanal final, actualizada cada 25 minutos, como todas las copias periódicas realizadas cada 5 horas. Este último tipo de copias se encuentran en el subdirectorio *temp*, organizadas por el día de su generación mediante la utilización de carpetas.

Aunque la creación de copias de seguridad se ha automatizado, se decidió no automatizar su borrado ya que es muy frecuente que los administradores de sistemas tengan criterios muy diversos al respecto. Por ello, es responsabilidad del administrador del sistema la decisión de borrar, o no, las copias seguridad realizadas.

Al margen de la diversidad de criterio, se tomó esta decisión para garantizar que los administradores participen en este proceso, validando la correcta función del mismo. Es muy común que estos procesos pasen desapercibidos en la administración hasta el momento en el que sea necesario recuperar datos anteriores. Por ello se desea que lo administradores formen parte activa de este proceso tan vital.

A pesar de todo se recomienda, según nuestra experiencia, para facilitar la administración, guardar el estado del sistema 6 meses hacia atrás, no más. Además también es recomendable eliminar las copias realizadas cada 2 horas de las semanas anteriores a la actual, ya que al final del día se genera la copia incremental del mismo.

2.2.4. Alta disponibilidad.

Debido a la importancia de este componente en el sistema hace que la disponibilidad del mismo sea una de las características más relevantes. Su NO disponibilidad produciría un error completo del sistema ya que el resto de componentes no serían capaces de realizar sus funciones, por lo que los usuarios no podrían acceder a su información.

Por ello es necesario atribuirle al servidor una infraestructura lo suficientemente flexible como para conseguir una alta disponibilidad del servicio. Aunque esta infraestructura no fue implementada, debido a que su carga de trabajo sobrepasaba el tamaño del proyecto y su funcionalidad se alejaba del núcleo del mismo, se diseñó una arquitectura que pudiera dar soporte a la cantidad de solicitudes previstas para este sistema.

La arquitectura planteada se encuentra definida por 5 capas funcionales, repartiendo entre ellas las distintas responsabilidades que deberá realizar la arquitectura, como son: la seguridad, la atención de peticiones y el acceso a la información. En el diagrama Ilustración 11 se muestra la estructura de nuestro diseño junto a los componentes que forman cada capa.

Capa 1: Firewall - Cortafuegos.

Este nivel es el principal encargado de la *seguridad* del sistema. Estará compuesto por un Firewall que le protegerá ante ataques maliciosos de diversa índole. Dicho Firewall bloqueará el acceso de cualquier petición que no sea por el puerto configurado para el uso del servicio.

Además se coordinará con los componentes de la *Capa 3* para denegar el acceso, de forma temporal o permanente, aquellas IP que podrían estar realizando un ataque de denegación de servicios (DDoS).

Capa 2: load balancer - Balanceador de Carga.

El balanceador de carga distribuirá todas las peticiones entrantes entre los distintos nodos de la capa inferior, distribuyendo la carga del sistema de una forma homogénea. El balanceo se realizará mediante el redireccionamiento del flujo TCP/IP, por lo que se le delegará a los nodos de la *Capa 2* la responsabilidad de revisar el contenido de estos paquetes.

El uso de este componente nos permitirá ofrecer una mayor disponibilidad, ya que dispondremos de la capacidad de respuesta de varios equipos, en lugar de solo uno. Además dispondrá de una carga dinámica, por lo que podrá aumentar o reducir el número de máquinas activas según la demanda existente en ese momento.

Capa 3: atención de peticiones - care requests

Se trata del nivel más relevante de la arquitectura en cuanto a la capacidad de respuesta se refiere. Formado por un número de máquinas a definir, ya sean virtuales o físicas, se encargará de atender todas las peticiones recibidas. Es por ello, por lo que se le deberá incluir a estas máquinas los elementos de red necesarios para la realización de su tarea, como tarjetas de red de alta capacidad, memoria RAM de gran tamaño, etc.

Las peticiones serán atendidas por un componente propio especialmente diseñado para esta funcionalidad. Podría utilizarse tanto un WebService alojado en cada máquina, bajo el servidor Web Apache, o un componente desarrollado completamente donde se establezca un protocolo de comunicación a modo de API. Una vez recibida la petición se analizará y realizará todas las acciones necesarias sobre el SGBD. Para limitar al máximo el tiempo de respuesta entre estos nodos y la capa inferior (SGBD) se utilizará un conexionado de alta velocidad entre ellos.

La utilización de este componente hace que todos los subsistemas de este proyecto desconozcan el SGBD utilizado, así como las características del mismo. Además se conseguirá un mayor control del sistema porque se podrá filtrar completamente las acciones y acceso al SGBD.

Al ser en este nivel donde se traducirá el contenido de los paquetes TCP/IP, se analizará el contenido de las peticiones, comprobando si sigue el protocolo diseñado o si existe algún tipo de ataque en él. Además durante la primera fase de la comprobación se registrará, de forma global, el acceso del usuario y su IP, permitiendo la detección de un posible ataque DDoS. Cualquier detección de un posible ataque será registrado en el sistema (SGBD) y transmitido a la primera capa para que realice el bloqueo de esta posible amenaza.

Las comunicaciones, como hasta ahora, se realizarán de forma encriptada, utilizando para ello un certificado digital global que poseerán todos los nodos de este nivel.

Capa 4: Cluster de Sistemas Gestores de Bases de Datos

Esta capa se caracteriza por el funcionamiento de un cluster de servicios, concretamente del SGBD. Dicho cluster estará formado por un grupo de nodos, ya sean físicos o virtuales, que ofrecerán el acceso a los datos almacenados en las bases de datos. Para facilitar la configuración del cluster, las BD se alojarán en un equipo externo donde los nodos de esta capa deberán conectarse. Este hecho hace que se deba controlar que únicamente existe un nodos suministrando el servicio, ya que de no ser así podrían corromperse los datos.

Debido a la importancia al acceso de la capa inferior se utilizará un tipo de conexión lo suficientemente rápido como para eliminar la latencia producida al utilizar almacenamiento distribuido.

En un inicio se considera utilizar MySQL cómo SGBD, como hasta ahora, pero deberá analizarse si existen otras alternativas para la implementación de este nivel.

Capa 5: Almacenamiento Distribuido

La última capa estará compuesta con un sistema de almacenamiento distribuido, como por ejemplo la exportación de discos iSCSI. Este almacenamiento será utilizado por los SGBD de la capa anterior para alojar las distintas BD, permitiendo que todos los nodos tengan acceso a los datos reales y actuales del sistema.

Por ello es de vital importancia, como se ha nombrado, que únicamente se encuentre en funcionamiento el servicio del SGBD en uno de los nodos al mismo tiempo, ya que podría corromperse los datos de no ser así.

En la sección *6. Perspectivas Futuras* se profundizará sobre el sistema de alta disponibilidad necesario, teniendo como base el ya mostrado. En dicho apartado se abordará tanto el diseño y funcionalidad cómo la tecnología necesaria para su elaboración.

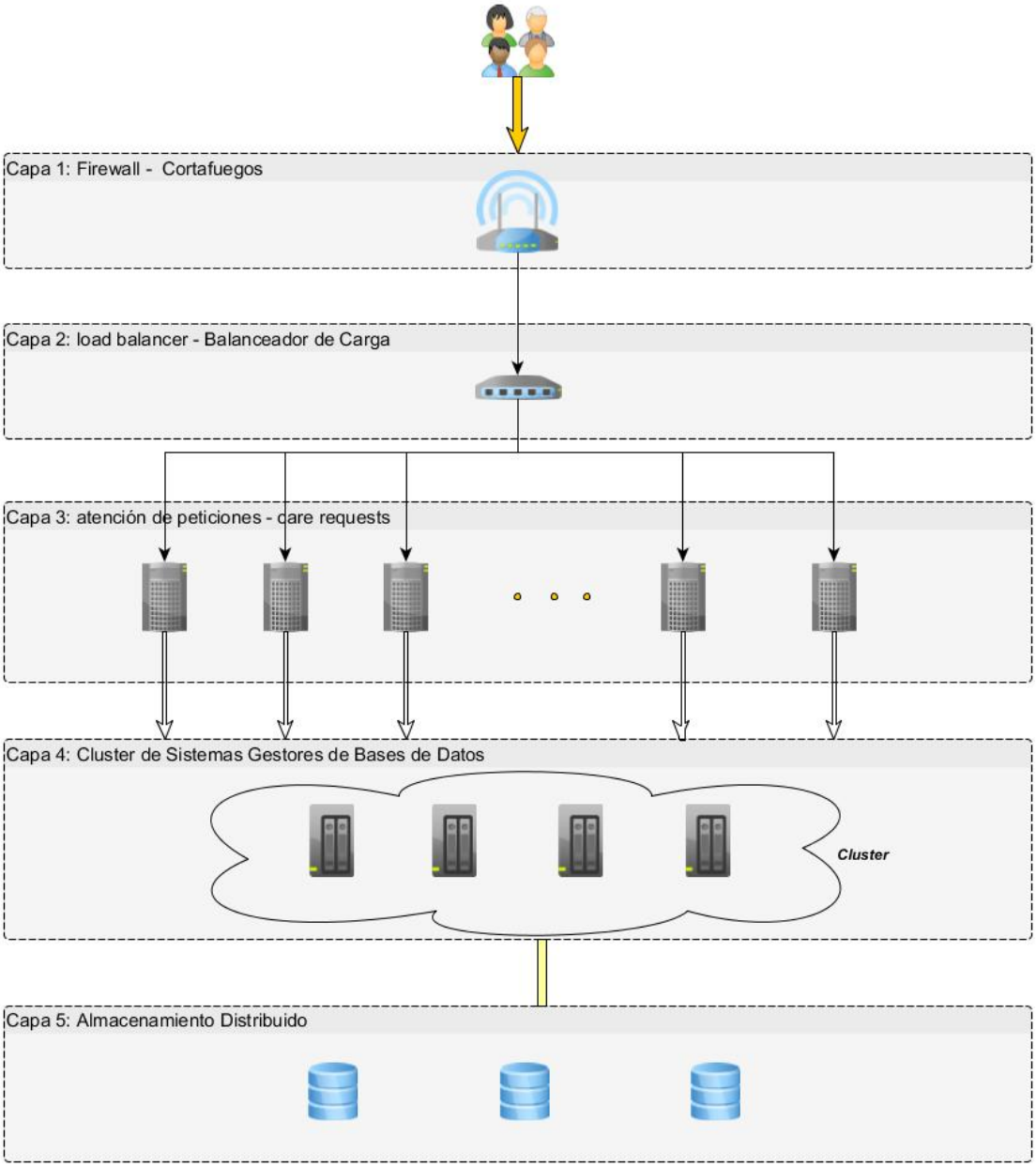

Ilustración 11. Diseño de plataforma de Alta Disponibilidad

3. Subsistema de Escritorio.

El subsistema de escritorio es un tipo de ***Cliente de Acceso*** específico para usarlo en entornos de escritorio gráfico. Como se indicó en la introducción, este componente permitirá al usuario acceder a las distintas funcionalidades ofrecidas, comunicándose con el ***Servidor de Información***.

Para lograr un producto de mayor expansión se ideó como un sistema multiplataforma, dándole la posibilidad al usuario de elegir la plataforma que prefiera. Al tratarse de una solución multiplataforma el diseño adquiere una mayor relevancia, ya que cualquier error o modificación se verá propagado hacia los distintos sistemas.

Por ello, se decidió crear un entorno de generalización que nos permita abstraernos del SO utilizado. Dicho entorno estará constituido por un diseño en capas, a modo de niveles, que suministrará los distintos servicios proporcionados por la mayoría de las plataformas. Con este diseño en capas se consigue un mejor mantenimiento y una mayor flexibilidad ante cambios, muy probables por la posibilidad de utilizar código específicos para cada plataforma, pero nos obliga a tener un especial cuidado con las API de uso de cada servicio.

Estos niveles formarán un componente base que permitirá elaborar aplicaciones multiplataforma, debido a que posee un nivel de abstracción sobre el cual pueden ejecutarse.

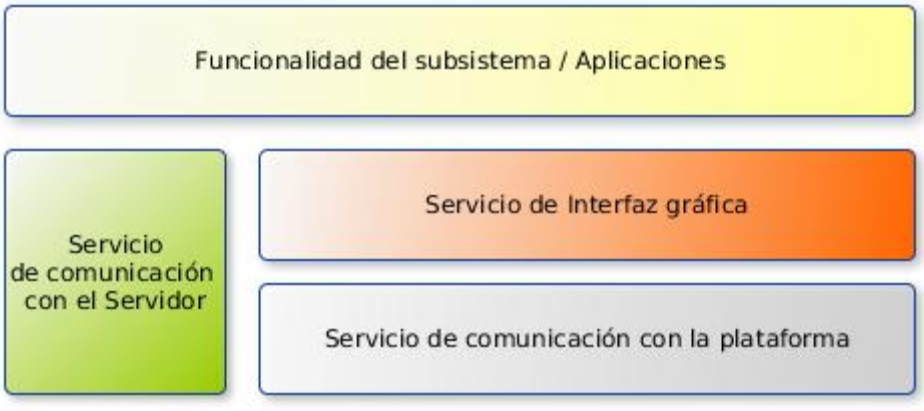

Ilustración 12. Componentes del Subsistema de Escritorio

En el gráfico Ilustración 12 se muestran los niveles de servicios prediseñados inicialmente, de forma generalizada, sin incorporar peculiaridades de las plataformas.

1. ***Funcionalidad del subsistema/Aplicaciones:*** Este nivel contiene toda la funcionalidad final del componente, es la parte visible por parte del usuario.
2. ***Servicio de interfaz gráfica:*** Abstrae las características gráficas de cada plataforma, permitiendo utilizar una interfaz gráfica estable y común para cada una de ellas.
3. ***Servicio de Comunicación con servidor***: Su principal responsabilidad será la comunicación con el ***Servidor de Información***.
4. ***Servicio de comunicación con la plataforma***: Se trata del nivel más básico del sistema. Se encarga de realizar la comunicación con el sistema, como la escritura o lectura de ficheros.

Además de un diseño en capas para el sistema multiplataforma base, sobre el que correrá la aplicación, también se introdujo el paradigma MVC (Modelo-Vista-Controlador) en el diseño de las funcionalidades de la aplicación final, las cuales harán uso de los servicios anteriores.

Este paradigma adicional permitirá disminuir el acoplamiento entre las distintas funcionalidades y la dependencia con el propio ***Servidor de Información***. Logrando aumentar, más aún, la tolerancia a cambios y la facilidad de mantenimiento, debido a que cada funcionalidad dependerá de sí misma, sin verse influida por el resto de ellas.

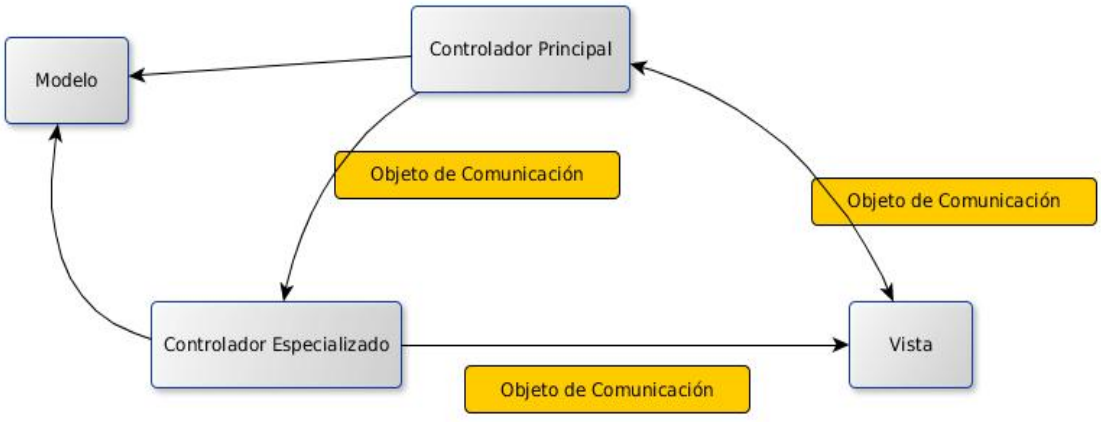

Ilustración 13. Diseño MVC planteado para Subsistema de Escritorio

Los elementos diseñados para este paradigma, junto a su relación, se presentan en la imagen Ilustración 13, los cuales son:

1. ***Controlador principal***: Encargado de gestionar las peticiones de las vistas hacia los controladores especializados y de la gestión global del programa. Este controlador tendrá asociado la correspondencia entre vista-controlador. Además, llevará a cabo la comprobación de credenciales durante el login, junto al bloqueo por inactividad o el control de refresco.
2. ***Controlador especializado***: Realizará la funcionalidad específica de cada servicio del programa. Recibirá los eventos lanzados durante la ejecución de la aplicación, a través de las vistas y decidirá qué acciones ejecutar ante cada petición, enviando a la vista emisora las acciones que deben llevar a cabo.
3. ***Objeto de Comunicación***: Contendrá los mensajes utilizados para sincronizar la comunicación entre las vistas y los controladores. Según el sentido de la comunicación algunos de sus atributos podrán cambiar de significado.
4. ***Modelo***: Este componente se encuentra en el ***Servidor de Información*** y será el encargado de gestionar la información del usuario. Los controladores especializados realizarán la edición de datos en la BD mediante la ejecución de *Stored Procedures*. En el caso de las consultas utilizarán directamente cláusulas SQL.
5. ***Vista***: Es la parte visible para el usuario. Se encargará de atender los distintos eventos producidos y transmitirlos a su controlador siempre que sea necesario.

Una vez finalizado el componente y atendiendo a lo descrito en este punto, como en la introducción de la memoria, los requisitos y responsabilidades que deberá satisfacer tras su finalización será:

1. Cumplimiento de la LOPD
    a. Conexiones seguras.
2. Ofrecer los distintos servicios del sistema.
3. Facilidad de uso.
4. Utilizable en distintas plataformas.
5. Rápido aprendizaje.
6. Alto nivel de mantenimiento.
7. Gran flexibilidad y tolerancia a cambios.

3.1. Estructura del componente.

Al tratarse de una solución multiplataforma hace que la selección tecnológica adquiera una gran importancia, debido a que es necesario que los sistemas faciliten y mejoren la capacidad de traslación de la aplicación. Debido a ello se analizaron varias alternativas tecnológicas para la creación de los diferentes servicios, siendo debatidos en posteriores reuniones y eligiendo una de ellas para cada nivel. Finalizada esta fase, se eligió como estructura tecnológica a:

3.1.1. Descripción tecnológica de la solución

A. *C++:*

Al encontrarnos en un entorno docente hace que podamos tomar decisiones basadas en criterios especulativos y de aprendizaje, ajenos a los utilizados en un entorno de producción. Es por ello por lo que se ha elegido a C++ como el lenguaje de programación con el que se implementará la aplicación resultante de este subsistema.

C++ es un lenguaje imperativo evolucionado a partir del lenguaje C. Este lenguaje permite la utilización de programación genérica, programación estructurada o programación orientada a objetos, por lo que es conocido como lenguaje de programación multiparadigma.

Se ha elegido este lenguaje en lugar de otros más obvios y usuales como es el caso de Java, Python o Perl, que ya soportan la multiplataforma, por el reto que supone y sobre todo por la reflexión que se puede obtener de ello, tras el movimiento actual.

A día de hoy se ha expandido en la comunidad informática la creencia de que los lenguajes multiplataforma Phyton y Java poseen una superioridad al resto de lenguajes. Esta superioridad viene asociada, generalmente, por su capacidad multiplataforma y sobre todo por la cantidad de módulos, de diversa índole, que poseen cada uno de ellos. Haciendo alusión a que el resto de lenguajes son inferiores debido al inexistente o escaso número de módulos adicionales.

En nuestra opinión, creemos que esta filosofía se tambalea por el hecho de que se le atribuyen a dichos lenguajes la potencia y capacidad de la comunidad de Software Libre que llevan detrás. Es decir, se defiende que un lenguaje es mejor que otro por los componentes adicionales que disponen y no por las características propias

de este. Por lo tanto, uno de los axiomas por los que se fundamenta esta idea es erróneo, ya que la comparativa correcta no sería entre lenguajes y sí entre comunidades informáticas.

Observando este criterio se puede comprender la infravaloración de lenguajes como ADA y C++, donde el desarrollo de elementos de Software Libre carece del control y distribución que poseen otros lenguajes. Un claro ejemplo lo encontramos entre Phyton y C++ donde la diferencia entre ambas comunidades es abismal. En el caso de Phtyon posee 2 comunidades de desarrollo, una oficial como es el caso ***Python Software Foundation*** y otra no oficial formada por el repositorio de software para aplicaciones de terceros ***Python Package Index (PyPI)***, pero considerada cuasi-oficial por los usuarios. En cambio, C++ no posee una comunidad organizada como tal, sino que los usuarios suben a los distintos portales de desarrollo, como GitHub, los proyectos que realizan, sin seguir ningún estándar ni control de calidad.

Es por ello, por lo que decidimos llevar a cabo el desarrollo de esta aplicación multiplataforma mediante la utilización del lenguaje NO multiplataforma C++. Con ello intentamos demostrar la capacidad real que posee C++, así como la necesidad imperiosa de crear una organización oficial que permita centralizar y homogeneizar todos los proyectos realizados con este lenguaje. Además se fomentará el uso y desarrollo de Software Libre, aportando a la comunidad componentes de mayor nivel y calidad.

Para facilitar la migración de la aplicación se utilizará, en todo lo posible, aquellas características que nos ofrece las bibliotecas estándar de C++, bajo el espacio de nombres conocido como ***STD***. Con la utilización de estas librerías se logrará reducir la necesidad de utilizar peticiones exclusivas para cada plataforma, facilitando el desarrollo y mantenimiento del producto. Entre los usos que podemos realizar con el *STD* de C++ podemos encontrar la manipulación de ficheros, estructuras de datos, gestión de strings, etc.

B. *WxWidgets.*

La realización de la interfaz gráfica se realizará mediante el proyecto WxWidgets, también conocido como Wx. Este proyecto consiste en una librería multiplataforma, y multilenguaje, que da soporte para la construcción de interfaces gráficas.

En su selección se analizó Wx cómo la librería Qt, el proyecto Qt se trata de otra librería multiplataforma utilizada para el desarrollo de interfaces de usuarios. La principal diferencias entre ellas es la forma en la que construyen los objetos gráficos. Wx utiliza las API's de las distintas plataformas para construir interfaces nativas de la plataforma utilizada. En cambio, Qt inserta una capa de abstracción entre el S.O. y la librería, obteniendo interfaces genéricas y similares en los distintos entornos.

El hecho de crear interfaces nativas y específicas para todas las plataformas fue el principal motivo por el que decidió la opción de Wx. El presentar diseños exactos para todos los entornos limita la adaptabilidad de la aplicación, pudiendo provocar en el usuario una sensación de rechazo, ya que encontrará una aplicación que puede entrar en conflicto con el diseño del sistema utilizado.

Otro factor que se consideró, es la cantidad de cambios a los que se ha visto sometido el proyecto Qt desde que Nokia vendió el producto a la empresa Digia. Temiendo que el desarrollo que se hiciera en este proyecto no tuviese soporte en un futuro, o incluso eliminarse la publicación de nuevas versiones liberadas, ya que siempre ha tenido una versión comercial de gran potencia que se ha ido liberando durante la construcción de nuevas versiones.

Por todo ello se eligió a WxWidgets como la mejor opción para la elaboración de nuestras interfaces de usuarios multiplataforma. Además con su utilización fomentamos la utilización del software libre, ya que este proyecto se encuentra totalmente publicado mediante la licencia *wxWindows Licence*, similar a la licencia GPL con la excepción de que el código binario realizado por el desarrollador puede ser de tipo propietario.

C. *Conector MySQL C.*

Para la comunicación con el SGBD se ha elegido el conector basado en el lenguaje C suministrado por MySQL. Este conector está formado por una librería multiplataforma y una API de uso que nos permitirá establecer conexiones con el SGBD, pudiendo utilizar las diversas características ofrecidas por MySQL. Además se podrá configurar, de forma muy sencilla, si se desea realizar estas comunicaciones de forma encriptada.

Se ha preferido la versión en C respecto a la disponible en C++ ya que aporta un mayor rendimiento, al tratarse la versión de C++ de un encapsulamiento del conector original en C.

Otra de las alternativas que se consideró fue utilizar el conector *ODBC* aportado también por MySQL, el cual nos podría permitir una abstracción del SGBD utilizado. En este caso se rechazó su uso debido a que no consideramos importante este factor de abstracción, ya que nosotros mismos realizamos esta abstracción mediante la incorporación del ***Servicio de Comunicación con servidor***. Además el conector *ODBC* ofrece esta abstracción a costa de la sencillez de uso y del rendimiento del conector, pudiendo perder alguna funcionalidad exclusiva de MySQL.

Al margen de ello, el cambio del SGBD no es un factor que se tenga en consideración, ya que se ha elegido este componente de alto nivel como una herramienta que nos garantice un buen funcionamiento a lo largo del tiempo.

D. *Plataformas disponibles:*

La aplicación se diseñará para que pueda ser utilizada tanto en la plataforma Linux y cómo en Microsoft Windows. Inicialmente será desarrollada en Linux, donde se dispondrá la configuración para el resto de componentes. Una vez que se obtenga la estructura base de la aplicación se comenzará la fase de emigración hacia Microsoft Windows.

E. *Make:*

Se trata de una herramienta de gestión de dependencias muy extendida en el ámbito informático para la compilación de proyectos. Esta herramienta nos facilitará la gestión de los módulos existentes, permitiéndonos configurarlos y recompilarlos de forma individual. Con ella podremos administrar las distintas dependencias existentes en el programa así como automatizar comprobaciones previas a la compilación, como la existencia de directorios y su creación.

Además será de gran ayuda para la emigración de la aplicación, debido a que podremos utilizar los mismos pasos en su compilación sin tener conocimiento de los ajustes internos que se realizarán a lo largo del proceso de compilación.

Obtenida la estructura tecnológica, nos centraremos en el desarrollo de los distintos niveles de servicios presentados en la imagen Ilustración 12. Estos servicios se diseñaron de tal forma que se aprovechara al máximo las capacidades de las tecnologías empleadas.

En el siguiente apartado se explicará en mayor detalle los diferentes elementos que componen cada módulo de este subsistema.

3.1.2. Descripción de los módulos de la solución

A. *Comunicación con la plataforma.*

Este módulo se encargará de realizar las diversas acciones que necesiten la participación de la plataforma en la que nos encontramos. Está diseñado para llevar a cabo tanto la manipulación de ficheros, como el uso de threads o semáforos.

Con este nivel se pretende abstraer a nuestra solución de las diferencias existentes entre las plataformas a utilizar. Para ello se hará uso, en todo lo posible, de la librería estándar STD y de la posibilidad que nos ofrecen las macros de C++, permitiendo seleccionar el código a utilizar según la configuración que indiquemos.

B. *Servicio de Comunicación.*

Será el encargado de ofrecer la comunicación con el ***Servidor de Información.*** Este procedimiento nos permitirá realizar tanto la ejecución SP cómo de consultas SQL. Está configurado para realizar todas las transacciones de forma segura, utilizando para ello certificados TLS.

Concretamente, se deberá utilizar el mismo certificado establecido en la configuración del servicio MySQL para el establecimiento de conexiones seguras. Esto se debe a que nuestro conector validará la conexión utilizando este certificado durante el protocolo de cifrado.

A parte de la propia comunicación, se encargará de revisar todas las sentencias SQL a enviar, en búsqueda de posibles ataques de *inyección SQL*. Cada vez que el *Controlador* lance una acción SQL, este módulo le devolverá un objeto de tipo CustomComunicate en el cual le devolverá el código de error del resultado, como el resultado de la misma.

El Servicio de Comunicación está formado por tres clases: sgbd_conectorBase, sgbd_conector y handleError. En la figura Ilustración 14 se puede observar la relación existente entre ellas.

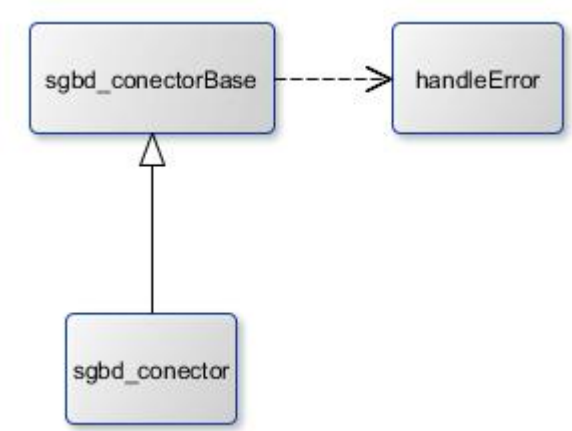

Ilustración 14. Diseño de clases del módulo de comunicación

- **sgbd_conectorBase**: Clase base en la comunicación con el Servidor de Información. Su funcionalidad consiste en la creación de una API que facilite la utilización del conector C proporcionado por MySQL. Esta clase es utilizada por los controladores específicos, ya que no permite la modificación de las credenciales de acceso, funcionalidad solo permitida por el controlador principal. Las acciones que permiten son:
  - ◊ Envío y recepción de consultas SQL.
  - ◊ Envío y recepción de ejecución de SP.
  - ◊ Manejo de errores.
  - ◊ Comunicación cifrada.

- **sgbd_conector**: Extiende de sgbd_conectorBase. Esta nueva clase posee las mismas funcionalidades que la clase anterior, añadiendo la modificación de los credenciales de acceso, funcionalidad solo permitida para el Controlador General del subsistema. Con esta diferenciación permitimos que todos los controladores del sistema tengan acceso a la información pero la edición de esta esté supervisada.
- **handleError**: Utilizada por las clases anteriores, es la responsable de traducir los diferentes errores ocurridos durante alguna fase de la comunicación. Diferencia los errores producidos en categorías, según el origen, para notificárselo al usuario de la mejor forma posible. Los errores que captura son:

Clasificación de errores.	
1. Error gestionado por el Stored Procedure	
(Cod. 1)	El SP ejecutado ha lanzado una notificación de error
2. Error de acceso	
(Cod. 20)	Error de contraseña o nombre. Error procedente del SGBD. Se produce cuando la pareja usuario-password no corresponde con ningún usuario del sistema.
(Cod. 21)	BD inexistente. Error lanzado por el SGBD. Se ha intentado acceder a una BD inexistente.
3. Error de conexión	
(Cod. 30)	Error conexión con el servidor. No se ha podido establecer la conexión con el SGBD
(Cod. 31)	Error en el uso SSL. Error producido durante la conexión SSL con el SGBD.
(Cod. 32)	Error en el autocommit. Procedente del SGBD. Se produce durante la deshabilitación del autocommit.
(Cod. 33)	Servidor desconocido. Desconocido el Host contenedor del SGBD. Este error es uno de los más relevantes, ya que en un entorno de desarrollo se debe diseñar para que nunca suceda.
4. Error de SQL	
4.1 Error de construcción de consulta.	
(Cod. 410)	Tabla inexistente.
(Cod. 411)	Campo inexistente.
(Cod. 412)	Expresión incorrecta.
4.2 Ataque inyección SQL	
(Cod. 420)	Se ha detectado un posible ataque de inyección SQL en la cláusula SQL enviada.
4.3 Error de construcción en la ejecución del SP	
(Cod. 430)	Error en el formato de parámetros
(Cod. 431)	SP no existente
(Cod. 432)	Error en el número de parámetros
(Cod. 433)	Error producido durante su ejecución

Tabla 3. Clasificación de errores del Subsistema de Escritorio

C. *Módulo de interfaz gráfica.*

Con este componente se permitirá utilizar diferentes elementos gráficos, abstrayéndonos de la plataforma que nos de soporte. Para facilitar su construcción, se decidió utilizar el proyecto WxWidgets, el cual consiste en bibliotecas multiplataforma para el desarrollo de interfaces gráficas.

Debido a la gran diversidad que posee Wx y el número de posibilidades de configuración, se decidió construir un framework que sirva como capa intermedia entre nosotros y Wx, creando así un módulo más sencillo en su utilización.

Nuestro framework ha sido llamado WxAF, *WxWidgets Application Framework*, el cual nos permitirá construir aplicaciones de escritorio multiplataforma de forma muy sencilla. Está orientado hacia el paradigma MVC, siguiendo nuestro patrón de diseño base y apoyándose en las distintas capas de servicios ya mencionados.

Con esta decisión fomentamos la filosofía del Software Libre, creando un producto que compartimos con el resto de la comunidad, enriqueciéndose con mayores recursos. Por ello se ha publicado dicho framework en el portal GitHub, bajo la licencia GNU General Public License version 3.0 (GPLv3) y con el nombre de WxAF, en el repositorio https://github.com/borjabrisson/WxAF.

Durante su diseño se decidió ir un paso más allá, e introducir en el framework toda la funcionalidad del resto de módulos, aglutinando toda nuestra funcionalidad de servicios en el propio framework. Esto nos permitirá crear un producto de mayor potencia y calidad, ya que no solo permitirá crear interfaces gráficas multiplataforma, sino aplicaciones completas.

En los siguientes apartados se profundizará en las diversas características y peculiaridades que posee tanto el framework como la propia solución diseñada.

WxAF, WxWidgets Application Framework

WxAF está formado por 4 componentes principales: base, conector, controladores y customService, entre los cuales se repartirán las distintas funcionalidades y responsabilidades descritas en los anteriores módulos.

La configuración del framework se realiza a través del fichero de cabecera ***config.h***, localizado en la carpeta *configuration*. Con este fichero se podrá indicar la IP

del servidor de información, la BD por defecto, el tiempo de refresco, así como la activación o desactivación del registro de errores y/o avisos.

Para su compilación se utilizará la herramienta *make*, haciendo uso de tres ficheros Makefile, diseñados para propósitos distintos pero interconectados entre sí. Existirán 2 variantes de estos ficheros, unos orientados para la plataforma Linux y otra para Microsoft Windows. Estos ficheros se encuentran en el directorio *build*, en el cual se alojan los distintos ficheros objetos resultantes del proceso de compilación. En el apartado *3.2. Adaptabilidad a Windows / Multiplataforma* se explicará en mayor profundidad las características de este proceso y las particularidades existentes entre ambas plataformas, en cuanto a la compilación se refiere.

En los siguientes apartados se abordará los distintos componentes y características que constituyen al framework WxAF.

I. Módulos WxAF.

El funcionamiento del framework se encuentra dividido en 5 módulos principales, atribuyéndole a cada uno una responsabilidad específica, estando relacionada, en algunos casos, con los servicios diseñados anteriormente,

A. Base:

Este módulo contendrá todas las clases que permitirán la creación gráfica de los distintos componentes disponibles. En la imagen Ilustración 15 se mostrará las clases que componen el módulo, dividiéndolas en grupos funcionales para una mayor claridad.

	Clases	Funcionalidad
Grupo Application	**skeletonApp**	Forma el armazón sobre el que se construirá la clase baseApp. Su utilización es necesaria para eliminar el conflicto de dependencias entre las clases del grupo panel y baseApp. Hace de API para las clases de dicho módulo, indicándole las funciones que pueden utilizar.
	baseApp	Se trata de la unidad básica sobre la que se desarrollará la aplicación final. Esta clase proporciona todos los mecanismos necesarios, desde el lanzamiento del login,

	Clases	Funcionalidad
		hasta el control del flujo de la comunicación.
Grupo Panel	**basePanel**	Forma la estructura base de todas las Vistas a generar. Implementa el mecanismo de comunicación con los Controladores, centralizando el proceso en esta clase.
	mypanel	Construye las estructuras necesarias para la representación visual de las Vistas. Es el punto de partida para construir el resto de componentes visuales.
	panelBaseGrid	Realiza un nivel de abstracción para todas las Vistas que utilicen una lista (grid) en su visualización. Esta clase captura todos los eventos producibles por una lista, ofreciendo las herramientas necesarias para capturar los eventos que se necesiten.
	panelGrid	Vista general que presenta una lista donde poder mostrar la información necesaria. Hace uso del resto de clases heredadas para facilitar la utilización de los diferentes recursos gráficos.
	panelCalendar	Esta Vista presenta dos recursos del sistema: una lista y un Calendario superior.
	panelCustomer	Esta clase permite poder crear Vistas personalizables. No posee ningún recurso visual (listas o calendarios), únicamente contiene las distintas herramientas para facilitar la creación y utilización de los diferentes recursos gráficos que se posee.
Grupo Ficha	**DialogBase**	Como ocurrió con la clase *skeletonApp*, se tuvo que incluir esta clase entre LookupBox y PopUp para eliminar la dependencia cíclica que existía. Esta clase forma parte de la base de PopUp, creando estructuras y métodos que faciliten la creación de los diversos elementos gráficos
	Lookupbox	Gestiona la creación de los Inputs que se presentarán en la ventana emergente. Estos elementos tendrán asociada una etiqueta e identificador que se utilizará para referenciar su contenido

	Clases	Funcionalidad
	popup	Construye una ventana emergente utilizada para mostrar un formulario previamente diseñado. Hace uso de la clase Lookupbox para facilitarle al usuario la inserción y edición de inputs
Grupo Aditional	**mygrid**	Lleva a cabo la construcción y gestión de las listas. Permite la inserción y edición de filas y columnas, pudiendo enviarle datasets completos. Además esta clase nos permitirá capturar los distintos eventos ocurridos sobre la lista, como: simples o dobles clicks, selección de filas, click de botón derecho, etc.
	sysEvent	Contiene los eventos globales que atenderá la aplicación
	timer	Implementa el temporizador con el que se controlará la gestión de inactividad del usuario

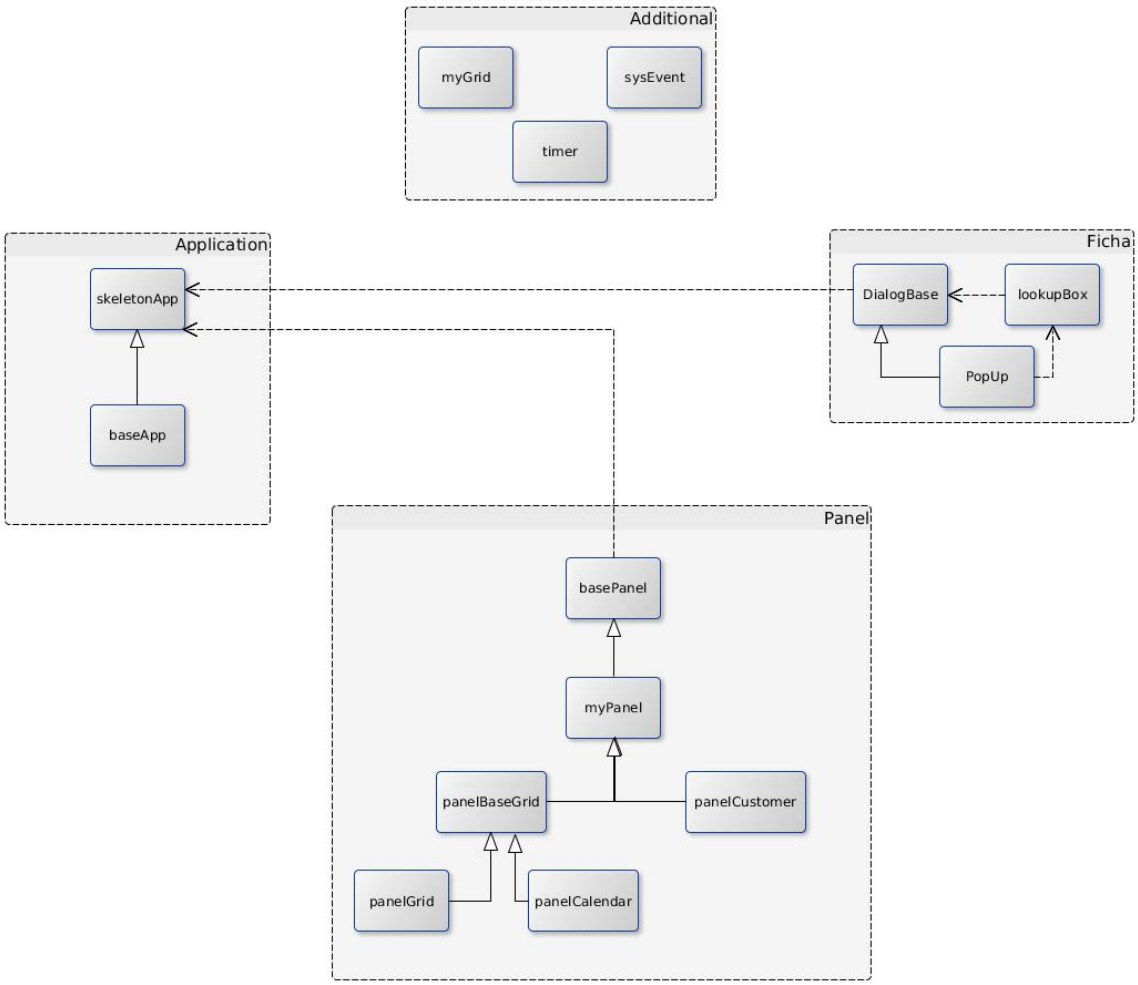

Ilustración 15. Diseño de clases del módulo base del framework WxAF

B. Conector:

Realizará el funcionamiento del módulo *Servicio de Comunicación*, utilizando para ello a las clases *sgbd_conectorBase, sgbd_conector* y *handleError*, tal y como se determinó anteriormente.

La localización del certificado digital a utilizar se encuentra en la macro DIR_CERT, dentro del fichero config.h. En nuestro caso, el certificado lo hemos alojado en el directorio *configuration*, bajo el nombre *ca-cert.pem*.

C. Controllers:

Llevará a cabo toda la funcionalidad necesaria por parte de los ***Controladores Específicos***, desde la creación del propio controlador hasta el protocolo de comunicación. Las clases utilizadas para ello son:

	Clases	Funcionalidad
Controllers	**CustomComunicate**	Lleva a cabo la creación de los objetos de comunicación. Los atributos que utiliza para ello pueden cambiar según el sentido de la comunicación. Los atributos que se utilizan son: ❖ **Type**: Indica de que tipo se trata la acción a realizar (Action_type,Query_type o Filter_type) ❖ **ActionID**: : Especifica tanto la acción como la consulta a utilizar ❖ **specificAction**: Se utilizará como subcategoría en los casos donde una acción pueda tener alternativas. ❖ **Data**: Contendrá los datos a utilizar en los casos de filtros, acciones, etc. ❖ **Dataset**: Conjunto de datos provenientes de una consulta.
	parser	Es utilizada para transformar la nomenclatura utilizada por el usuario para realizar los filtros en sentencias SQL válidas. Para ello se hizo uso de un autómata de estados con el cual poder capturar el comportamiento del parser resultante.
	ctrBase	Controlador Base a partir del cual se crearán los distintos Controladores. Ofrece herramientas pre-diseñadas para facilitar la realización de las distintas responsabilidades

D. CustomService.

Este módulo está diseñado para que el desarrollador que desee utilizar el framework, añada el código de los distintos servicios que realizará la aplicación resultante. Aunque no esté obligado, se recomienda agrupar estos servicios en directorios específicos para cada funcionalidad, hecho que facilitará la compilación de la aplicación.

La configuración de los ficheros makefile para la compilación de los componentes añadidos por el desarrollador se detallarán en el apartado XXX.

Al margen de estos módulos, existen otros directorios dentro del framework que se encargan de características secundarias del sistema.

A. Configuration:

Como ya se ha comentado, en este directorio se definen los distintos elementos configurables del sistema. Las entidades que encontraremos son:

	Clases	Funcionalidad
Configuration	***action***	Identifica los tipos de bloques de mensaje a utilizar, al igual que las posibles acciones que pueden indicar tanto los controladores como las vistas
	configuration	Contiene todos los parámetros configurables del sistema. En el se alojan los distintos flags que utiliza el sistema, como el tipo de plataforma en la que se encuentra

B. Build:

Se trata del lugar en el que se disponen todos los ficheros de configuración para la compilación en ambas plataformas. Además se alojará todos los ficheros objetos resultantes durante la compilación de la aplicación.

C. Docs:

En este directorio se encuentran todos los ficheros de información respecto al framework, desde el significado de cada componente hasta la configuración necesaria para la compilación.

II. Protocolo de comunicación - flujo de comunicación.

El protocolo de comunicación es un mecanismo vital para la correcta ejecución de las aplicaciones que utilicen el framework. El hecho de utilizar el paradigma MVC para la construcción de las aplicaciones, hace que los mecanismos de comunicación adquieran una mayor relevancia, ya que su buen funcionamiento garantizará la correcta sincronía entre los distintos componentes de la aplicación.

El mecanismo de comunicación consiste en un paso de mensajes centralizado y gestionado por la clase *customApplication* de la aplicación. Esta clase hará de

Controlador Principal, administrando el flujo de la información, permitiendo o no la finalización de la misma. Recibirá la solicitud de comunicación por parte de una Vista (Page) y comprobará si esa Vista tiene acceso, o no, al Controlador indicado, haciendo uso de la relación Controlador-Vista que posee. De tener acceso se le enviará el mensaje al Controlador asociado, el cual realizará las tareas oportunas. Una vez finalizada la actividad, el Controlador le devolverá un mensaje a la Vista con las acciones que deberá realizar.

Como se observa en el diagrama Ilustración 16 consta de 9 etapas, transcurriendo entre las distintas clases responsables de la comunicación.

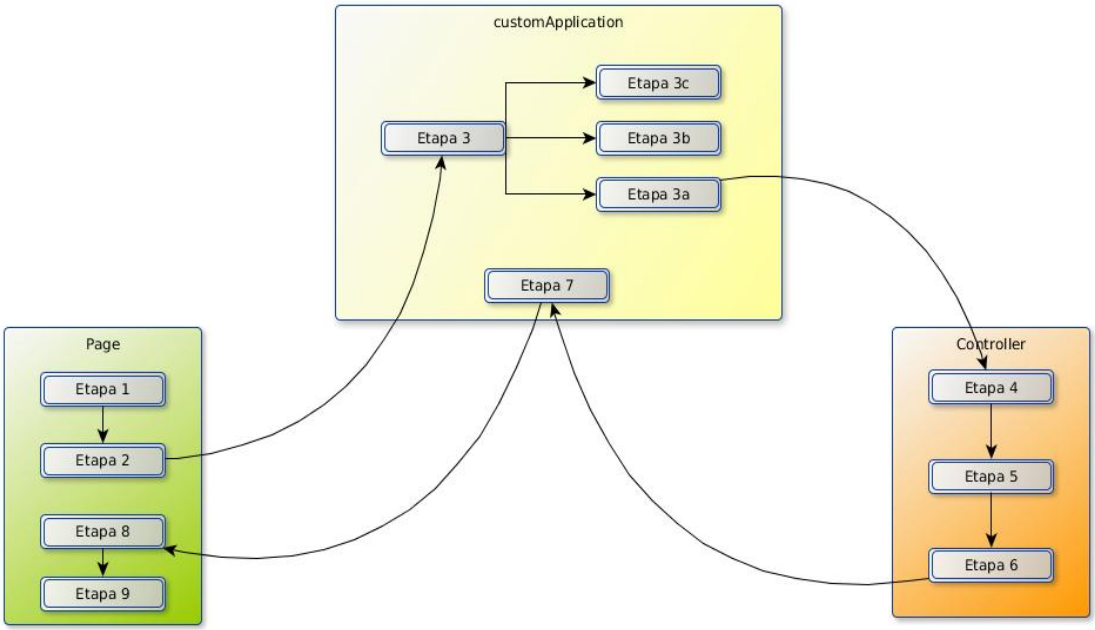

Ilustración 16. Flujo de comunicación entre las Vistas y Controladores

Etapa 1: La Vista (***Page***) recibe la acción del usuario, la cual se traduce en una petición hacia su Controlador.

Etapa 2: La Vista construye el objeto *CustomComunicate* con la información necesaria, indicando: el Controlador destinatario, el identificador de la Vista, el tipo de solicitud y la acción solicitada.

Etapa 3: ***CustomApplication*** recibirá la petición de la Vista y decidirá si es válida su solicitud. En este nivel nos encontramos con 3 posibles estados:

> **Etapa 3a**: El sistema se encuentra activo y la Vista se encuentra asociada con el Controlador indicado.

Etapa 3b: La Vista no tiene relación con el Controlador solicitado, por lo que se cancelará su petición.

Etapa 3c: La aplicación se encuentra bloqueada. Este bloqueo puede ser debido tanto por inactividad cómo o por no haber realizado la autenticación inicial. En este estado no debería producirse ningún envío de solicitud, al encontrarse el sistema bloqueado. Por ello se cancela cualquier petición hasta que el usuario inserte sus credenciales satisfactoriamente.

Etapa 4: El Controlador (***Controller***) recibe la notificación del *customApplication* con el mensaje de la Vista. En esta fase analiza el tipo de la solicitud y lo redirige al manejador correspondiente.

Etapa 5: Recibido el mensaje en el manejador oportuno se comenzará a su tratamiento. Se realizarán las distintas tareas necesarias para la realización de la solicitud.

Etapa 6: Una vez obtenido el resultado de la petición, se construirá un objeto customComunicate con las acciones que deberá realizar como consecuencia de la realización de su solicitud, enviándolo posteriormente a *customApplication*.

Etapa 7: *customApplication* recibe la respuesta del Controlador y la redirige a la Vista inicial.

Etapa 8: La Vista recibe la respuesta a su solicitud. Como en la fase 4, se analiza el tipo de mensaje y lo redirige al manejador correspondiente.

Etapa 9: Se realizan las distintas acciones indicadas en la respuesta a la petición de la Vista.

Como ya se ha comentado, la información enviada en los mensajes se almacena en los objetos de tipo customComunicate. Los tipos de mensajes que dispone el framework son:

- ***BKCM_Action***: Se solicita la edición de datos o una acción concreta del sistema. El identificador de la acción seleccionará la tarea a realizar dentro de todas las posibles opciones.
- ***BKCM_Query***: Se desea realizar una consulta. El identificador referente a la acción indica que consulta desea realizar dentro de todas las ofrecidas.

- ***BKCM_Filter***: La acción adjunta hace referencia a tareas de filtrado de datos.

Aunque se comparta el tipo de solicitud, los tipos de acciones que se utilizan para notificar a una Vista son totalmente diferentes a los utilizados en la comunicación con un Controlador. Las acciones base que suministra el framework de forma genérica son:

A. Mensaje dirigido hacia un Controlador:

WxAF solo establece dos acciones genéricas a enviar a los Controladores. Estas acciones son del tipo *BKCM_Filter* y se encargan de gestionar la utilización de los filtros.

- **V2C_onFilter**: Le indica al Controlador que el usuario desea poder filtrar el conjunto de datos mostrado.
- **V2C_setFilter**: Envía los datos con los que el usuario a indicado realizar el filtro.

B. Mensaje dirigido hacia una Vista.

Todas las acciones diseñadas para las Vistas son del tipo BKCM_Action, aunque el desarrollador puede incluir nuevas acciones que sean de tipos diferentes.

- **C2V_showList**: La vista deberá mostrar en la lista indicada los datos adjuntos al mensaje.
- **C2V_showMsg**: Mostrará en un cuadro de diálogo el texto indicado en el mensaje.
- **C2V_actionOK**: La solicitud enviada fue realizada con éxito.
- **C2V_actionDeny**: No se ha finalizado la acción solicitada.

III. Requisitos para la construcción de una aplicación.

Cualquier persona que desee utilizar el framework WxAF deberá instanciar las siguientes clases:

- ***Custom Application:*** Deberá construir una clase heredera de BaseApp en la cual instanciar la aplicación WxWidgets a crear. En esta clase se asociará cada controlador con su vista/s. Además se decidirá si se lanza el login y se construirán los distintos menús de la aplicación.

- ***Custom frameContainer:*** Tendrá que crear una clase extendida de frameContainer, indicando las vistas (Pages) que le desea añadir a este contenedor, referenciando a cada una con su propio identificador.
- ***Controller-Page:*** Se recomienda utilizar una pareja Vista-Controlador por cada funcionalidad que desee añadir. Aunque el desarrollador no esté obligado a realizarlo, se recomienda esta buena práctica para alcanzar productos de mayor calidad y mantenimiento. Hay que recalcar que será esta asociación la que hay que indicarle al Custom Application. Además, es recomendable utilizar un fichero de cabecera donde indicar las características de la funcionalidad, como el identificador de la Vista o si se desea añadir nuevas acciones al mecanismo de comunicación.

Una vez hecho, se deberá rellenar los ficheros Customfile con el contenido y formato que se indica. Como se comentó, en el apartado *3.2. Adaptabilidad a Windows / Multiplataforma* se detalla la información que debe añadirse a los ficheros de compilación de cada plataforma. Para facilitar este proceso se recomienda situar el código de todas las funcionalidades en el directorio *customService.*

D. *Funcionalidad del subsistema.*

En este módulo se realizará las funcionalidades que la aplicación ofrecerá al usuario final. Se apoyará en el resto de niveles, aglutinados en el framework WxAF. Como se ha comentado, toda servicio que vaya a ser implementado a través del framework debe estar formada por un Controlador (***Controller***) y una Vista (***Page***).

En nuestro caso, se ha construido 5 módulos, a modo de componentes, que realizarán las diversas funcionalidades diseñadas: General, Contactos, Mensajería, Calendario y Expedientes. En la figura Ilustración 17 se observa como queda jerarquizada la estructura final del ***Subsistema de Escritorio***.

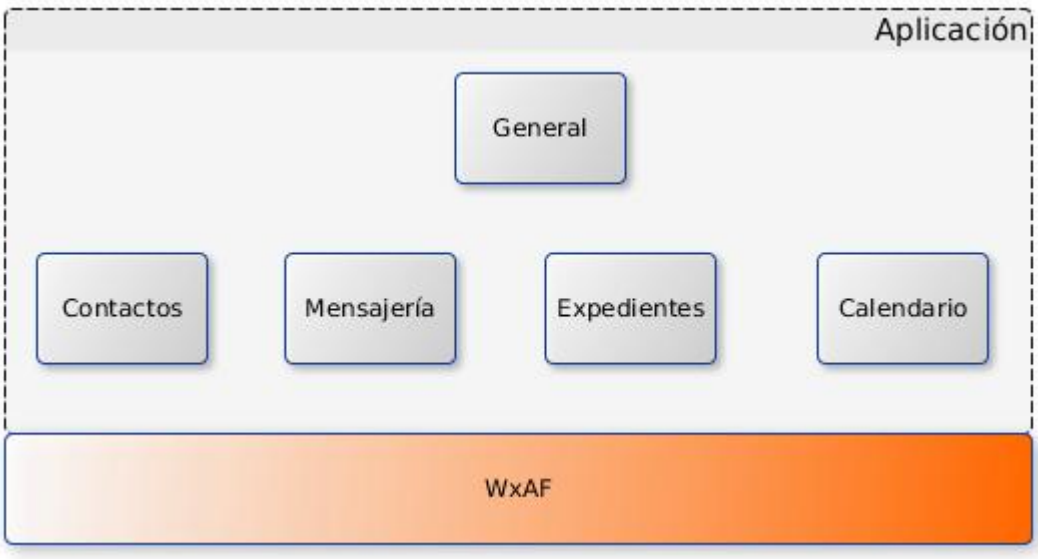

Ilustración 17. Funcionalidades desarrolladas sobre el framework

En los siguientes puntos se mostrarán las distintas clases que componen cada funcionalidad, así como las responsabilidades de cada una. En cada módulo encontraremos un fichero nombrado ***events.h***, en el cual se detallarán las acciones e identificadores utilizados para el correcto funcionamiento del servicio a implementar.

1. *General:*

Realiza la configuración y administración global de la aplicación.

- ***MyApp:*** Se trata de la implementación del *CustomApplication* necesario para el funcionamiento del framework. Esta clase se encargará de la validación del login, como de la correcta comunicación entre las Vistas y sus Controladores.
- ***MyFrame:*** Realiza el funcionamiento del ***Custom frameContainer***. Indicará que Vistas se deben mostrar, asignándole a pestaña del NoteBook a cada una.

2. *Contactos:*

Llevará a cabo la gestión y edición de contactos. Permitirá añadir nuevos contados, editarlos y borrarlos. Dispondrá de 4 tipos de contactos distintos: Abogados, Clientes, Empresas y Procuradores. Las clases utilizadas para su realización son:

- ***CompanyContactCard***: Se trata del formulario diseñado explícitamente para la captura de los datos de los contactos de tipo Empresa.

- ***ContactosCtr***: Se trata del Controlador responsable de la funcionalidad. Será quien reciba todas las peticiones enviadas por la Vista, tomando la decisión que tareas llevar a cabo.
- ***ContactosPage***: Vista de la funcionalidad, realizará la comunicación con el Controlador indicándole todas las acciones realizadas.
- ***ContactosFilter***: Formulario utilizado para realizar el filtro de las consultas activas.
- ***PeopleContactCard***: Complemento de ***CompanyContactCard***, mostrará el formulario para la manipulación de los contactos del tipo: Abogados, Clientes y Procuradores.

3. Mensajería:

La funcionalidad de Mensajería consistirá en el envió, respuesta y notificación de mensajes entre usuarios del sistema.

- ***MensajeriaCard***: Presentará el formulario con el que el usuario podrá insertar y editar sus notas.
- ***MensajeriaCtr***: Es el Controlador de la funcionalidad.
- ***MensajeriaFilter***: Muestra el formulario con el que el usuario podrá filtrar la consulta mostrada actualmente.
- ***MensajeriaPage***: Vista principal de la funcionalidad. Esta Vista será quien reciba las indicaciones del Controlador y quien le notifique las acciones realizadas por el usuario.

4. Calendario:

El uso del calendario le permitirá al usuario poder marcar eventos concretos, indicando el día, la hora, una descripción o incluso cuántos días se previos se desea nu notificación. Además se podrá añadir a cada cita un comentario que se añadirá a los ya existentes, si los hubiera.

- ***CalendarioCard***: Capturará los datos necesarios para la creación y edición de los eventos.
- ***CalendarioCtr***: Se trata del controlador de la funcionalidad, llevando a cabo toda la gestión y control de la misma.

- ***CalendarioPage***: Implementará la Vista de la funcionalidad, atendiendo todas las acciones realizadas por el usuario, enviándole al Controlador la información necesaria siempre que sea oportuno.

5. *Expedientes:*

- ***ExpedientesCard***: Se utiliza para capturar y mostrar los datos asociados a un Expediente.
- ***ProcedimientosCard***: Posee la misma responsabilidad que la clase anterior, solo que diseñado para la información contenida en los Procedimientos.
- ***ExpedientesCtr***: Realiza el control de la funcionalidad previamente dicha. Atenderá las peticiones de la Vista y decidirá qué acciones realizar.
- ***ExpedientesPage***: Lleva a cabo la gestión de la Vista de la funcionalidad. En este caso encontraremos 2 funcionalidades diferenciadas, una para la utilización de expedientes y otra para los procedimientos.

3.2. Adaptabilidad a Windows / Multiplataforma

Debido a las diferencias entre las plataformas utilizadas, se decidió implantar un procedimiento que le facilitara al desarrollador la tarea de compilación, homogeneizando el proceso.

Dicho proceso se lleva a cabo mediante la gestión de dependencia de la tecnología make, Se diseñaron 3 entidades a las cuales se le atribuyeron las responsabilidades de gestionar la compilación de los distintos elementos existentes: Aplicación Framework y funcionalidad del usuario.

La aplicación propiamente dicha será controlada por el fichero makefile, la cual construirá los distintos componentes de los que depende la aplicación. En este nivel se derivara la ejecución a las otras 2 entidades.

El Framework estará asociado al fichero de compilación Basefile.gcc donde se construirá los distintos objetos que constituyen al mismo. Este fichero no deberá ser tratado por el usuario, ya que su funcionamiento será independiente a la actividad del desarrollador.

Las funcionalidades que realizará el usuario deberán estar registradas en el fichero Customfile.gcc. Este fichero está diseñado para albergar todas las clases que deban ser compiladas en la construcción de la aplicación. Para ayudar en la tarea se diseñó el fichero de tal forma que se redujera esa actividad a una simple inscripción de clases. Esto fue llevado a cabo mediante la creación de diferentes variables que aglutinasen los flags de compilación, además de aprovechar los distintos recursos que ofrece la herramienta make.

Las diferencias existentes entre las dos plataformas, Linux y Windows, nos obligo a duplicar estas entidades especificando su contenido a las características de cada sistema. Debido a ello se crearon dos carpetas dentro del directorio build alojando en cada una de ellas la especificación de los ficheros indicados anteriormente.

Este hecho nos obligo a introducir un nuevo elemento que gestionase la bifurcación entre una plataforma y otra. Dicho elemento estará formado por un nuevo fichero Makefile llamado Mainfile, el cual permitirá compilar el sistema para una plataforma u otra según las dependencias indicadas, utilizando por defecto la plataforma Linux. En la siguiente imagen Ilustración 18 se muestra la actividad de esta entidad.

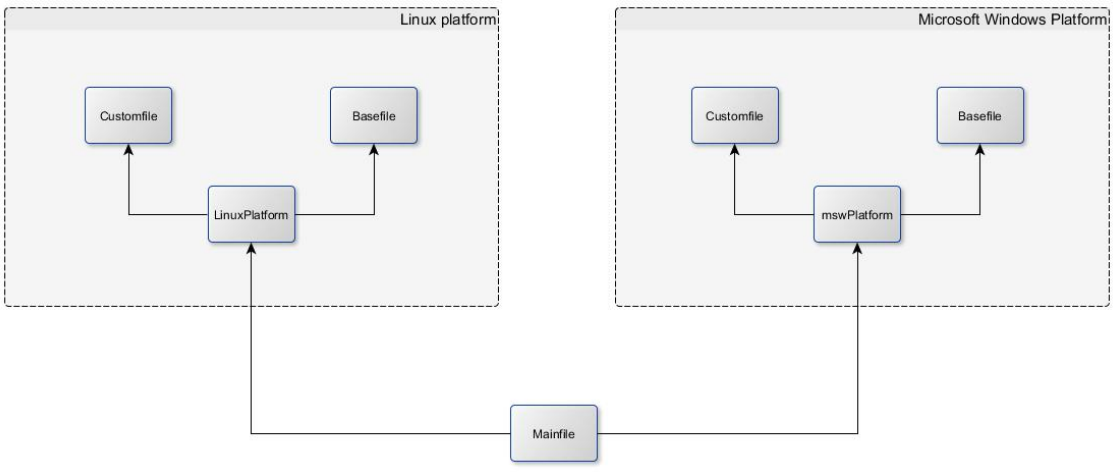

Ilustración 18. Distribución de los ficheros Makefile

Los ficheros Customfile están diseñados para utilizar como contenedor de las clases la carpeta *customService*, por lo que el usuario deberá cambiar dicha configuración si desea alojar las funcionalidades en otro directorio.

3.3. Conclusiones finales.

Durante la realización de este componente se observaron diferentes peculiaridades relacionadas con diversas características del subsistema, desde la propia realización de la aplicación hasta la migración de plataforma.

En un principio se encontraron varios problemas a la hora de documentarse sobre la utilización de la librería WxWidguets, a los cuales se le sumaron los problemas de ejecución debidos al mal entendimiento de las distintas entidades utilizadas por esta librería.

Una vez solventados estos problemas y ya con el 90% del componente realizado tuvimos el impedimento de realizar la migración del sistema. En este caso el conflicto se derivó, principalmente, por la falta de documentación para la compilación del conector C de MySQL. Además se encontraron numerosos imprevistos durante el proceso de adaptación de la librería Wx.

Atendiendo a todo ello, podemos sacar diversas conclusiones sobre las ineficiencias existentes en la comunidad y desarrollo de Software Libre. Si extendemos este análisis a la globalidad de la comunidad informática, encontraremos como cada vez se incrementa más y más aún el número de proyectos publicados

Debido a ello, creemos que es necesaria la creación de una organización o comunidad que desarrolle unos estándares de calidad y de documentación que regulen la publicación de los proyectos. Este estándar funcionaría como acreditación de la calidad del sistema, pudiendo estar estructurado en distintos niveles. Con ello se conseguiría productos de mayor calidad y una mayor expansión, debido a que se solucionaría uno de los problemas más frecuentes como es la falta o ineficiente documentación.

Además, estos proyectos podrían centralizarse en un repositorio global, organizado por tipo y características del mismo. Aunque a día de hoy existen múltiples organizaciones que proporcionan esta centralización, como GitHub, SourceFouece, Bitbucket, etc., no garantizan un nivel de adecuación suficiente como para garantizar su funcionalidad.

Otro factor que debe considerarse tras estos resultados es la imperiosa necesidad de estandarizar una arquitectura que facilite la migración de escritorio, sin la

obligatoriedad de utilizar un entorno virtualizado cómo en el caso de Java y Phyton, entre otros.

Este estándar puede estar formado por una estructura de directorios similares, junto a unas directrices de compilación comunes entre las diversas aplicaciones, así como la definición de diferentes variables de entorno. El momento en el que nos demos cuenta de la importancia y capacidad que tiene esta necesidad será cuando realmente comprendamos la filosofía “Less is more”.

4. Subsistema Móvil.

En la actualidad, la importancia de los dispositivos móviles ha ido aumentando año tras año y las previsiones auguran que esta tendencia seguirá in crescendo. Es por ello por lo que este subsistema adquiere una gran relevancia en el uso cotidiano del sistema. Con este componente se le permitirá, mediante el uso de dispositivos móviles, acceder al usuario en cualquier lugar del mundo, en cualquier momento.

Inicialmente se plantearon varias alternativas en el diseño del componente, utilizando diferentes modelos de aplicación.

En primer lugar, se consideró realizar una aplicación híbrida, este tipo de aplicación consiste en adaptar el diseño y configuración de nuestra página web al dispositivo de visualización. Con ello se consigue reducir el desarrollo de las aplicaciones de cada una de las distintas plataformas a un simple navegador. Además nos permite utilizar nuestra Aplicación Web como una aplicación nativa, logrando reutilizar componentes.

A pesar de todo se descartó su uso. La utilización de una interfaz generalizada nos prohíbe utilizar las distintas peculiaridades de cada plataforma, obteniendo una aplicación ajena a lo habitual en cada plataforma, lo que puede producir un cierto nivel de rechazo por parte de los usuarios habituados a una experiencia de usuario específica para cada plataforma.

Otra alternativa fue el apoyarnos en la utilización de un framework, como eMobc, Kendo UI, PhoneJS, etc, que nos permitieran construir aplicaciones multiplataforma de forma rápida y sencilla. Esta opción también fue rechazada por los mismos argumentos mencionados anteriormente. Además, debido a la capa intermedia que incorporan, el uso de este tipo de frameworks puede reducir el rendimiento, dañando a la imagen de nuestra aplicación. Otro aspecto a mencionar, es la dependencia que adquiriríamos a la hora de utilizar nuevos servicios que ofrecieran las plataformas, ya que estaríamos limitados por la capacidad de adaptación del framework utilizado.

Por todo ello, se decidió realizar *una* ***aplicación nativa*** *para cada plataforma*, que, a pesar de incrementar el tiempo de desarrollo, nos permitirá ofrecerle al usuario una aplicación con todas las características de su plataforma.

Aunque el subsistema móvil está definido para ser ejecutado en múltiples plataformas móviles, debido a la carga de trabajo, nos centraremos inicialmente en las plataformas Android e iOS, las cuales engloban, en 2012, el 87.6% de la cuota de mercado, 68.8% y 18.8% respectivamente.

A pesar de ello, debido a los requisitos necesarios para el desarrollo de aplicaciones iOS, licencias de desarrollo y equipos Apple, se decidió, para el presente proyecto, desarrollar este componente únicamente en Android. Aún así, se diseñará este subsistema de manera que nos permita poder adaptar su funcionalidad al resto de plataformas de la manera más sencilla posible.

Versión	Nombre Común	Nivel de API	Distribución
2.2	Froyo	8	1.6%
2.3.3 - 2.3.7	Gingerbread	10	24.1%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	ICS	15	18.6%
4.1.x	Jelly Bean	16	37.4%
4.2.x		17	12.9%
4.3		18	4.2%
4.4	KitKat	19	1.1%

Tabla 4. Distribución de versiones de Android según su nivel de API.

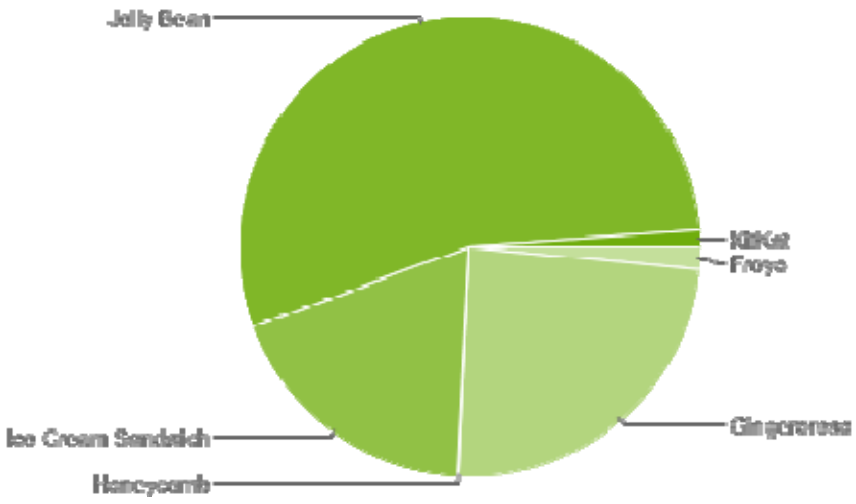

Ilustración 19. Distribución de Versiones proporcionada por Android

La aplicación Android tomará como base la versión 4.0 Ice Cream Sandwich (ICS), publicada en 2011. Se ha elegido esta versión debido al gran número de dispositivos activos que poseen esta versión o una superior. En la figura Ilustración 19 se puede observar las estadísticas publicadas por Google el pasado 2 de Diciembre de 2013, donde se muestra que el 74.2% de los dispositivos Android poseen una versión igual o superior a ICS. Además, si atendemos a los datos presentados en la gráfica Ilustración 20 se observa como esta tendencia ha ido aumentando a lo largo de este año, teniendo como versión límite la 4.0 (ICS). Por ello, a la vista de estos datos, se considera la versión 4.0 como un buen punto de inicio para desarrollar nuestra aplicación.

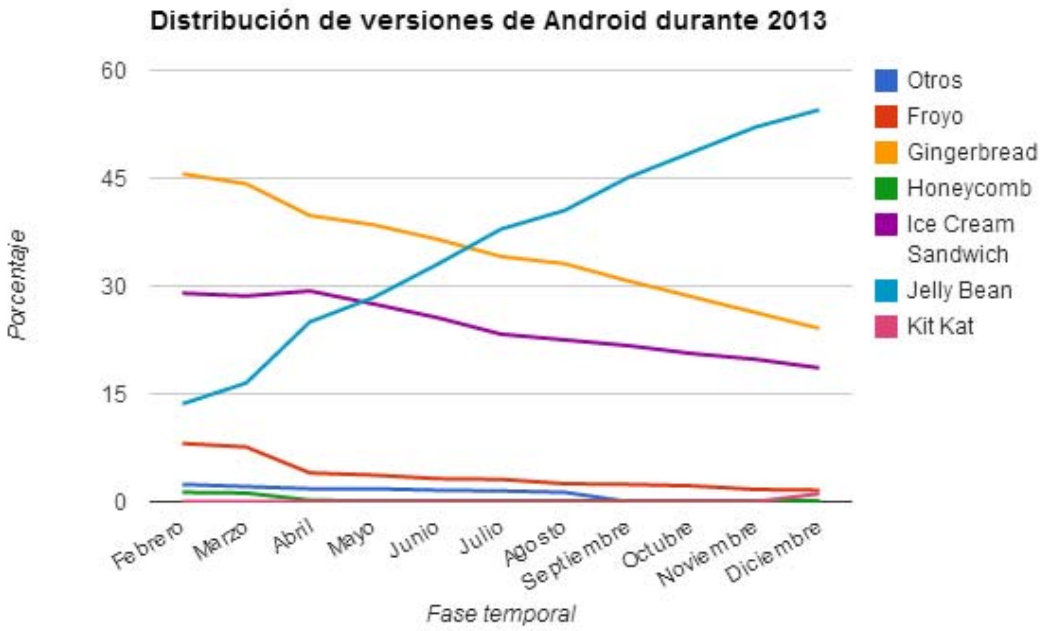

Ilustración 20. Distribución de las versiones de Android durante el año 2013

4.2. Diseño del componente.

El diseño del subsistema móvil debe tener en cuenta la posibilidad de acceso desde múltiples plataformas diferentes; por ello se decidió incorporar la utilización de un WebService, a fin de lograr un adecuado grado de abstracción y generalización.

Esta incorporación derivó en la modificación del flujo de comunicación mostrado al inicio de la memoria, introduciendo un nuevo camino al diagrama. La figura Ilustración 21 representa el nuevo flujo de comunicación del sistema global.

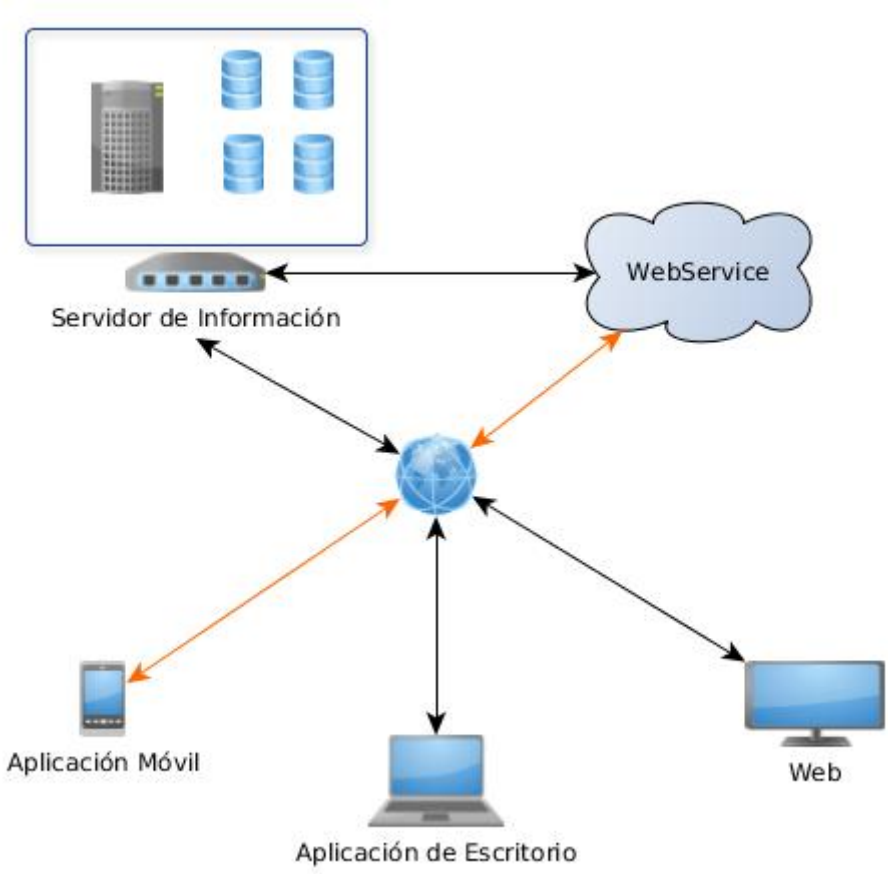

Ilustración 21.Modificación del flujo de comunicación producido por la inserción del WebService

Utilizando este WebService conseguimos abstraernos de la plataforma que utilizamos, permitiendo poder darle servicios a cualquier tipo de plataforma. Además, con este tipo de acceso facilitamos la creación de los conectores por parte de las plataformas móviles, al utilizar el protocolo HTTP/S. Por otra parte, eliminamos la necesidad de conocer el tipo de SGBD a utilizar, ya que los dispositivos móviles solo necesitarán conocer que acción desean hacer y cómo notificarlo al WebService.

Respecto a la aplicación Android, se utilizará el paradigma de diseño MVC que nos ofrece un alto grado de cohesión y mínimo acoplamiento. Esta solución está basada en la entidad Service que suministra Android para la realización de tareas en segundo plano.

En el apartado *4.4. Componente Android.* se detallarán los componentes y características de la solución realizada. A pesar de estar enfocada para la plataforma Android, se tomará como prototipo base para el resto de plataformas, marcando la filosofía de diseño para todas ellas.

4.3. Diseño WebService.

El servicio web diseñado está formado por 3 componentes: ***Cliente del servicio***, ***Manejador de peticiones*** y ***SGBD***. Cada uno de estos componentes cumple una función específica, que en coordinación con las demás partes implementan la funcionalidad total del servicio. La comunicación entre estos componentes se desarrolla a través de una serie de etapas, pudiendo derivar o finalizar la comunicación en varias de ellas.

En la figura Ilustración 22 se muestra las distintas etapas que constituyen este proceso de comunicación, así como los elementos que componen el diseño del WebService.

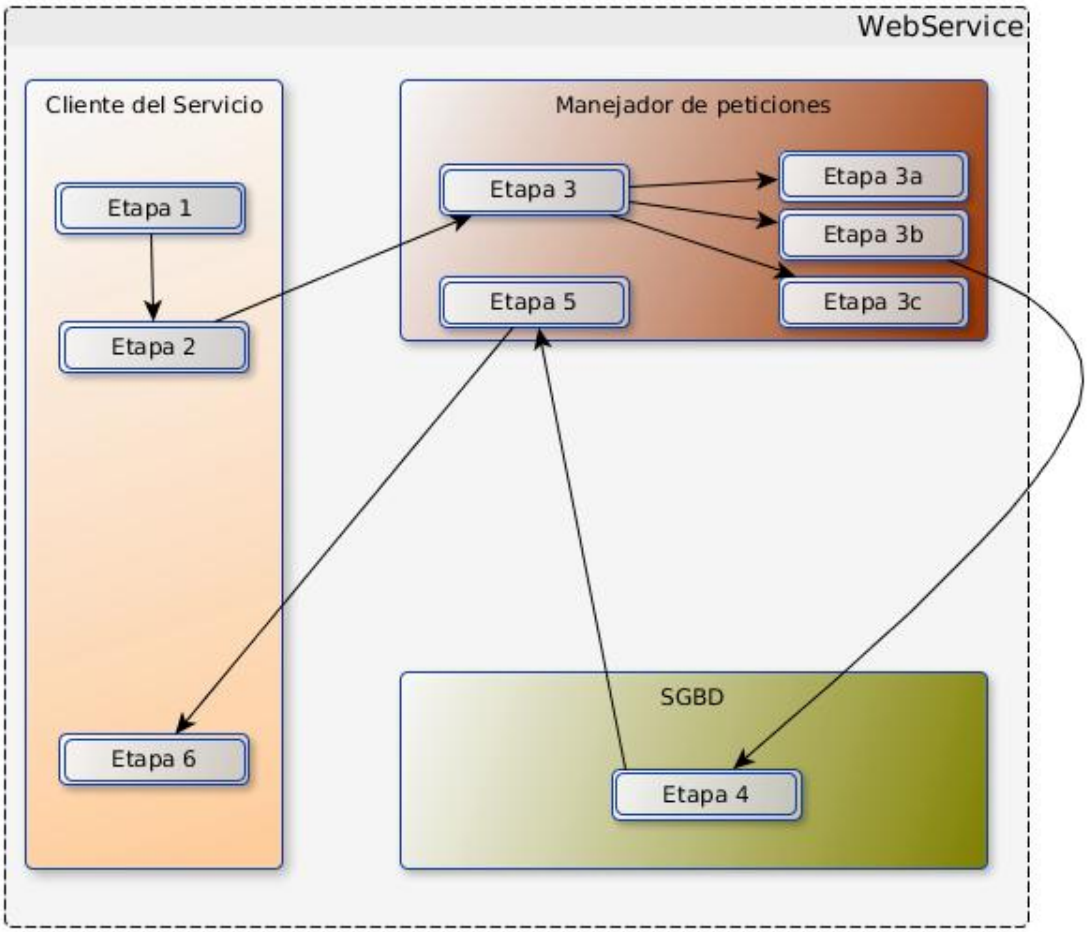

Ilustración 22. Protocolo de comunicación del WeService.

Etapa 1: El ***Cliente del Servicio*** analiza la solicitud del usuario y genera la estructura de datos necesaria para iniciar la petición de servicio.

Etapa 2: El cliente del dispositivo móvil realiza una petición al WebService. Previamente ha establecido comunicación con el servidor Web, iniciando el protocolo de comunicación HTTPS.

Etapa 3: En este momento se analiza la petición realizada. Se observa si su estructura corresponde con la establecida por la API del servicio y si no existe ningún error en los parámetros enviados.

***Etapa 3a*:** Se ha detectado algún tipo de error en la solicitud enviada. En este caso se finalizará el proceso, enviando un objeto JSON donde se indicará el tipo de error realizado junto a un mensaje descriptivo.

***Etapa 3b*:** La solicitud cumple con los requisitos establecidos. Se comunicará con el SGDB para realizar las acciones necesarias.

***Etapa 3c*:** Se ha detectado un error grave, la estructura del mensaje no corresponde con el establecido. Este caso se toma como un posible ataque de DDoS o un estudio de vulnerabilidades. En este caso no se devolverá ningún tipo de mensaje.

Etapa 4: El ***SGBD*** recibe y realiza las acciones indicadas por el ***Manejador de Peticiones***. Tras su realización le devuelve el resultado de las acciones enviadas. En este caso, la respuesta puede ser una notificación de error.

Etapa 5: El ***Manejador de Peticiones*** recibe la respuesta del ***SGBD***. Una vez obtenida, construye el objeto JSON oportuno según el tipo de respuesta y el tipo de servicio solicitado. Tras obtener el objeto JSON se le enviará al ***Cliente del Servicio*** como respuesta a su solicitud.

4.3.1. Componentes del WebService.

Cliente del servicio

Este componente estará alojado en las distintas plataformas móviles a utilizar. Será el encargado de transmitir las peticiones del usuario, convirtiéndolas en el formato correcto para la comunicación con el WebService.

Este caso en particular, estará elaborado por el módulo Android *PHPConector*, que hará de enlace con el sistema web. Este módulo lo hemos publicado en GitHub con el nombre PHPConector, bajo la licencia GNU General Public License, version 3.0 (GPLv3).

El paquete PHPConector está formado por 3 clases Java, cada una con una responsabilidad específica. Como sucedió con el *Subsistema de Escritorio*, se diseñaron estas clases a modos de capas o servicios. En la figura Ilustración 23. Diseño de clases pertenecientes al módulo PHPConectorIlustración 23 se muestra la relación existente entre dichas clases.

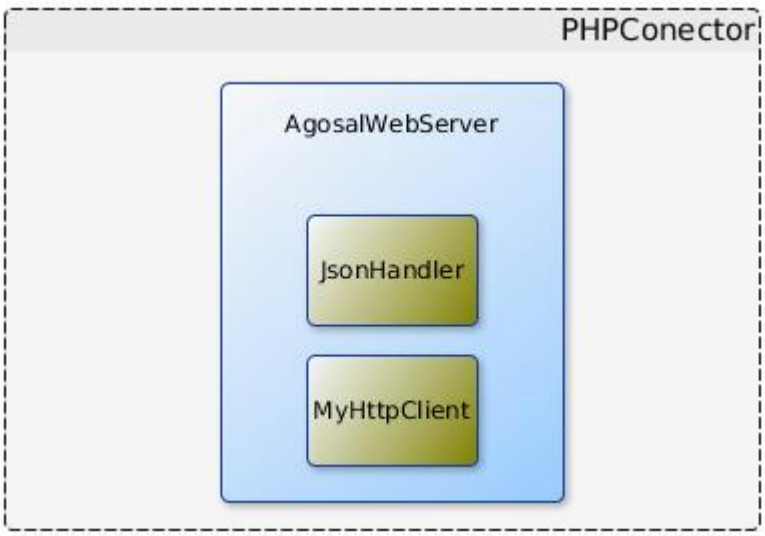

Ilustración 23. Diseño de clases pertenecientes al módulo PHPConector

- ***MyHttpClient***:

Proporciona un encapsulamiento del conector DefaultHttpClient, suministrado por Apache en su paquete org.apache.http, contenido en el núcleo de Android. Concretamente extiende de la clase DefaultHttpClient, la cual permite utilizar el protocolo HTTP sin ningún tipo de configuración, únicamente indicando la URL. MyHttpClient ofrece la posibilidad de transmitir datos mediante el mecanismo POST y en un futuro se añadirá la posibilidad de utilizar además el método GET.

Con MyHttpClient puedes alternar el uso de accesos HTTP o HTTPS, únicamente indicándolo en la URL a utilizar. Para el uso del protocolo HTTPS se deberá tener el certificado digital del dominio a utilizar. Este certificado debe alojarse en un contenedor de certificados, keystore, nombrado como "*mystore*" dentro del directorio /res/raw. Además la contraseña de acceso al contenedor se almacenará en la variable keyStore del fichero /res/values/strings.xml.

También facilita la obtención de la respuesta del servidor Web, proporcionando una función que nos devuelve un objeto String con todo el contenido recibido.

Debido a que nos encontramos en un entorno docente no disponemos de un dominio adquirido, por lo que tuvimos que indicarle al conector que únicamente validara si coincidía el certificado presente en la URL con uno de los almacenados en el contenedor, sin comprobar que realmente correspondiera con el dominio utilizado.

Una particularidad en la creación del keystore, es que debe cifrarse utilizando el módulo de cifrado BouncyCastle, debido a que Android solo da soporte para el formato BKS.

❖ ***JsonHandler***:

Es el encargado de la manipulación de los objetos JSON. JsonHandler le permitirá al desarrollador traducir los objetos JSON procedentes del WebService, además de construir objetos JSON con las parejas clave-valor que éste le indique.

Para su actividad, JsonHandler se apoya en el paquete de tratamiento de objetos JSON *JSON.simple,* suministrado por el repositorio de Software Libre de Google (*code.google.com/p/json-simple/).* Este paquete fue seleccionado tras analizar las distintas posibilidades que nos ofrecía la comunidad de Software Libre. Entre las distintas alternativas que consideramos están: org.json, JSON.simple, Jackson, etc.

Para la selección del módulo se tuvieron dos aspectos en consideración: la eficiencia y la complejidad de uso. En la figura Ilustración 24, se presenta una gráfica comparativa, proporcionada por la organización EclipseSource, donde se analiza el tiempo de ejecución tanto en la lectura cómo escritura de mensajes RAP.

El protocolo RAP describe la estructura de mensajes basados en JSON utilizando para el envío la comunicación HTTP. Este protocolo remplazó el envío de mensajes Javascript sin estructurar, experimentando un auge con la expansión actual de las plataformas móviles.

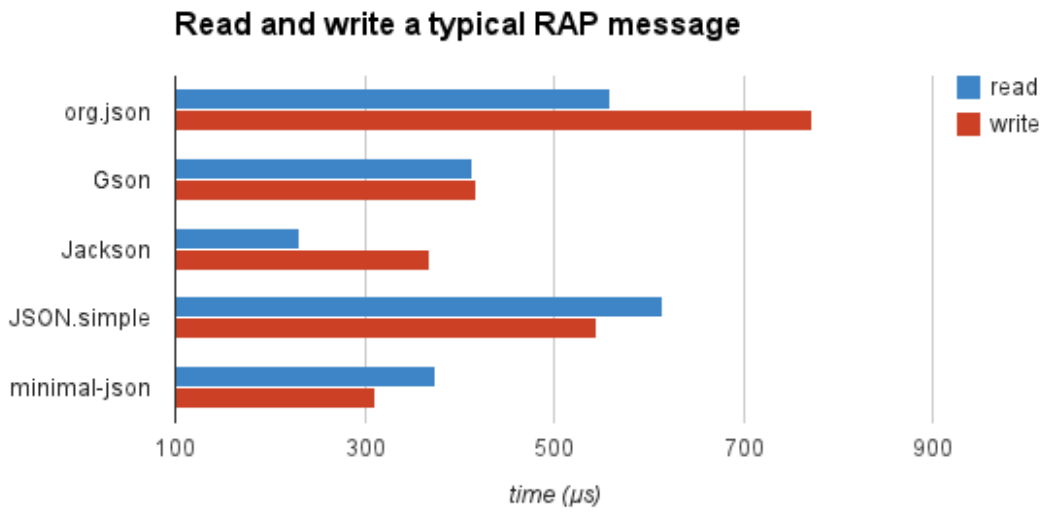

Ilustración 24. Comparativa entre distintos módulos JSON en el tratamiento de mensajes RAP.

A la vistas de estos datos se contempla como mejor opción el módulo Jackson, ya que posee un tiempo de ejecución en ambos casos muy inferior al resto. Aun así, todavía queda por analizar la complejidad de uso de la misma, y es en este punto donde se observa el porqué de sus resultados. La necesidad de configuración es tan alta que hace que su alto rendimiento se consiga a costa de una especificación muy detallada, hecho totalmente opuesto a nuestra metodología de trabajo.

Por ello, se estudiaron los distintos módulos atendiendo a su complejidad de uso. Combinando este estudio con el ya mostrado, observamos que la facilidad de uso del módulo JSON.simple era muy superior al resto y aunque su eficiencia fuese la segunda peor, no dejaba de estar por debajo de los 700 picosegundos, capacidad suficiente para los requisitos de nuestro sistema.

- **AgosalwebServer**:

Se trata del nexo de unión entre Android y el WebService. Hace uso de las clases anteriores para poder llevar a cabo las especificaciones indicadas por la API de nuestro servicio Web.

Para poder llevar a cabo su tarea necesita que el usuario introduzca sus credenciales de acceso al sistema. AgosalwebServer crea una capa de abstracción facilitando al desarrollador la utilización del WebService. Posee una función específica para cada tipo de servicio: Consulta, SP y chequeo de Credenciales.

A tratarse de conexiones vía HTTP/S el tiempo de respuesta podría alargarse, lo que podría producir que el gestor de procesos de Android notificará nuestra aplicación como suspendida. Debido a ello se tuvo que utilizar la clase *AsyncTask* de Android, la cual nos permite llevar a cabo tareas que pueden alargarse en el tiempo sin bloquear el funcionamiento de la aplicación.

El uso de AsyncTask hace que se introduzcan varias consideraciones que no se habían planteado previamente. Por ejemplo, todo objeto AsyncTask solo puede lanzarse una sola vez, por lo que es necesario clonar el objeto original si deseamos volver a utilizarlo. Debido a ello se introdujo la función *toClone*, la cual nos devuelve un nuevo objeto AgosalwebServer, imagen del primero.

Manejador de peticiones

Consiste en un módulo PHP situado en el servidor Web establecido. Es el responsable de atender las peticiones procedentes del ***Cliente del servicio***, implementando toda la funcionalidad y controles diseñados para el WebService.

Para su comunicación con el SGBD se apoya en el conector MySQLi ofrecido por la plataforma PHP, el cual consiste en una adaptación del propio conector suministrado por MySQL.

El manejador está constituido por dos clases PHP diseñadas en capas: HTTPD y lib. La clase Lib suministra la comunicación con el SGBD, abstrayendo todas las particularidades del conector MySQLi, y su configuración.

La clase HTTPD es quien implementa toda la funcionalidad de nuestro WebService. En esta clase se analiza las peticiones entrantes, devolviendo un objeto JSON con la información resultante de cada una.

HTTPD implementará el comportamiento e interfaces descritas en el punto *4.3.2. API WebServer.*, donde se acometerá, en mayor detalle, la especificación y comportamiento del WebService, y por lo tanto, de este componente.

SGBD

Elemento situado en el ***Servidor de Información*** y base para el funcionamiento del sistema. Cómo se ha indicado, alberga toda la información del sistema principal, ofreciéndole a los distintos subsistemas la información y capacidad de manipulación necesaria en cada momento.

4.3.2. API WebServer.

Debido a las características del sistema, el servicio se fundamenta en un nuevo tipo de paradigma en el desarrollo de WebServices, conocido como JSON-WSP. Este paradigma utiliza para la comunicación el protocolo HTTP/S y el método de petición POST, utilizando objetos JSON tanto en la solicitud (requests), cómo en la respuesta (responses) de peticiones HTTP.

A diferencia a lo que postula JSON-WSP, nuestro sistema solo enviará un objeto JSON como respuesta del servicio solicitado, utilizando para el envío de parámetros los objetos clave-valor del propio método POST. De esta forma se

consigue un sistema más eficiente y compacto, eliminado el proceso de análisis y conversión a objeto JSON en una de las etapas.

Los servicios diseñados inicialmente son 3: ***CheckCredentials***, ***lauchQuery*** y ***execSP***. En el siguiente punto se detallarán su funcionamiento y características individuales.

Servicios Ofrecidos.

- Realización de consultas (***lauchQuery***).

Descripción

Se ejecuta la consulta solicitada por el usuario. Si no se ha producido ningún error se devuelve el número de elementos resultantes, junto a los registros solicitados.

Estructura de la Solicitud

Se enviará un conjunto de elementos clave-valor, indicando las acciones a realizar. Los parámetros a utilizar son:

1. **typeAction**: indica el tipo de acción a realizar. En este caso deberá valer *lauchQuery***.** (Nota: Los tipos de acciones serán normalizados, por lo que no influye el uso o no de mayúsculas)
2. **user**: Usuario que desea realizar la acción
3. **pass**: Password utilizada como credencial de acceso.
4. **bd**: Base de datos a la que se desea acceder
5. **clause**: Consulta que se desea realizar.

Estructura de la Respuesta

En el caso de una ejecución correcta se enviará un objeto JSON, formado por un vector asociativo, cuyas claves primarias son:

1. **errorCode**: Indicará el número de filas que posee el resultado. En caso de una ejecución satisfactoria será un valor mayor o igual a 0.
2. **content**: Contendrá un vector de arrays asociativos, correspondiendo a las columnas de la consulta realizada.

Estructura del objeto JSON:

{ “errorCode”: [0-9]+,
“content”: <array_asociative_items> }

Caso de error

En caso de error se modificará levemente el objeto JSON enviado. En este caso se sustituirá el campo “content” por la clave errorMsg, cuyo contenido será el mensaje de error indicado por el sistema.

La especificación del objeto es:

{ “errorCode”: -[1-9]+,
“errorMsg”: <string>}

❖ Ejecución de procedimientos (***execSP***).

Descripción

Este servicio se encarga de ejecutar el Stored Procedure indicado por el usuario, utilizando los argumentos enviados. Antes de realizar la operación comprueba que el usuario tiene acceso al sistema y analiza los datos enviados para anular cualquier tipo de ataque de inyección SQL.

Estructura de la Solicitud

Los parámetros que se utilizarán en este método son:

1. **type**: Como ya se comentó, indica el tipo de acción a realizar. En este caso deberá valer execSP. (Nota: Los tipos de acciones serán normalizados, por lo que no influye el uso o no de mayúsculas)
2. **user**: Usuario que desea realizar la acción
3. **pass**: Password utilizada como credencial de acceso.
4. **bd**: Base de datos a la que se desea acceder
5. **SP**: nombre del Stored Procedure que se desea utilizar.
6. **args**: String con todos los parámetros utilizados por el SP, separados por comas. Esta string deberá estar correctamente formateada para su uso en el SP.

Estructura de la Respuesta

Si se ha ejecutado el SP correctamente se enviará un objeto JSON formado por varios campos, siendo algunos optativos, introducidos por la ejecución del SP.

1. **errorCode**: Indica el estado de la ejecución del SP, 0: satisfactorio, 0<: error.
2. **[msg]**: El procedimiento podrá indicar si desea enviar, o no, un mensaje predefinido al usuario tras la ejecución de este.
3. **[extra]**: Además el SP podrá devolver consultas junto a su estado. Para ello los procedimientos deberán añadir a sus consultas un campo nombrado "resultID" donde indicarán el nombre con el que desean que se indexe dentro del JSON. Para más información acuda al apartado X: Metodología de los SP.

Estructura del objeto JSON:

```
{  "errorCode": 0 | [1-9]+,
[msg]: <string>,
([extra<SP>]: <array_asociative_items>)*}
```

Caso de error

En el caso de error, se enviará un objeto JSON formado por 2 campos conocidos:

1. **errorCode**: Indica el código del error ocurrido durante la ejecución..
2. **msg**: Mostrará el mensaje descriptivo asociado al código de error.

Estructura del objeto JSON:

```
{  "errorCode": - [1-9]+,
"msg": <string> }
```

❖ Chequeo de Credenciales (***CheckCredentials***).

Descripción

Llevará a cabo el proceso de validación de las credenciales del usuario.

Estructura de la Solicitud

Los parámetros que se utilizarán en este método son:

1. **type**: Deberá ser CredentialsCheck. (Nota: Los tipos de acciones serán normalizados, por lo que no influye el uso o no de mayúsculas)
2. **user**: Usuario que desea realizar la acción
3. **pass**: Password utilizada como credencial de acceso.

Estructura de la Respuesta

Si se ha validado las credenciales del usuario se enviará un objeto JSON con el campo errorCode a 0.

1. **errorCode**: Indica el estado de la ejecución del SP, 0: satisfactorio, 0<: error.

Estructura del objeto JSON:

```
{  “errorCode”: 0 | [1-9]+}
```

Caso de error

En el caso de error, se enviará un objeto JSON formado por 2 campos:

1. **errorCode**: Indica el código del error ocurrido durante la ejecución..
2. **msg**: Mensaje descriptivo asociado al código de error.

Estructura del objeto JSON:

```
{  “errorCode”: - [1-9]+
,“msg”: <string> }
```

Clasificación de errores.

Para facilitar la identificación y gestión de errores se ha creado una clasificación que nos permita identificarlos de forma sencilla. A continuación, se mostrará una tabla jerarquizada donde se indicará los distintos errores capturados y los grupos a los que pertenecen.

Clasificación de errores.	
Error de protocolo	
(Cod. 10)	Falta de parámetros obligatorios.
(Cod. 11)	Servicio inexistente. Error producido cuando se solicita un servicio que no corresponde con los ofertados actualmente.
Error de acceso	
(Cod. 20)	Error de contraseña o nombre. Error procedente del SGBD. Se produce cuando la pareja usuario-password no corresponde con ningún usuario del sistema.
(Cod. 21)	BD inexistente. Error lanzado por el SGBD. Se ha intentado acceder a una BD inexistente.
Error de conexión	
(Cod. 30)	Error conexión con el servidor. No se ha podido establecer la conexión con el SGBD
(Cod. 31)	Error en el uso SSL. Error producido durante la conexión SSL con el SGBD.
(Cod. 32)	Error en el autocommit. Procedente del SGBD. Se produce durante la deshabilitación del autocommit.
(Cod. 33)	Servidor desconocido. Desconocido el Host contenedor del SGBD. Este error es uno de los más relevantes, ya que en un entorno de desarrollo se debe diseñar para que nunca suceda.
Error de SQL	
Error de construcción de consulta.	
(Cod. 410)	Tabla inexistente.
(Cod. 411)	Campo inexistente.
(Cod. 412)	Expresión incorrecta.
Ataque inyección SQL	
(Cod. 420)	Se ha detectado un posible ataque de inyección SQL en la cláusula SQL enviada.
Error de construcción en la ejecución del SP	
(Cod. 430)	Error en formato de parámetros
(Cod. 431)	Error SP no existente
(Cod. 432)	Error número de parámetros
(Cod. 433)	Error producido durante su ejecución

Tabla 5. Clasificación de errores gestionada por el WebService

4.4. Componente Android.

Antes de centrarnos en el diseño de la aplicación planteada, debemos conocer algunas de las características que posee Android, las cuales pueden afectar a nuestro diseño inicial planteado.

1. Toda aplicación de Android está constituida por un proceso en ejecución, por lo que si realizamos tareas ocultas al usuario bloquearemos la aplicación debido a que solo poseemos un hilo de ejecución. Otro aspecto derivado es el tiempo de ejecución de las tareas a realizar, ya que si este tiempo se alarga en el tiempo el administrador de procesos de Android podría señalar a nuestra aplicación como bloqueada.

2. Para la realización de tareas en segundo plano Android proporciona el objeto Service, el cual se ejecuta en un hilo propio perteneciente al proceso, evitando el bloqueo de la aplicación. Además también podría configurarse este objeto para que utilice un proceso propio, aunque implicaría utilizar una API de comunicación prediseñada por Android.

3. Android utiliza el término Activity para denominar a las Vistas de la aplicación. Aunque no es obligatorio su uso, Android recomienda definir su configuración gráfica en ficheros xml, situados en el directorio /res/layouts. La plataforma ofrece distintos tipos de Activities según el tipo de vista que se desea implementar.

4. Android suministra un método de comunicación entre la Vista activa y el Service, utilizando la interfaz Callback y la clase Message. Esta comunicación consiste en el envío y captura de objetos Message, cuyos atributos fijos poseen un significado conocido por ambas partes.

Considerando las características y herramientas que ofrece Android, se adaptó nuestro diseño MVC base, buscando un producto más eficiente y robusto que sea capaz de proporcionar toda la potencia y estabilidad necesaria, tal y como se muestra en la figura Ilustración 25.

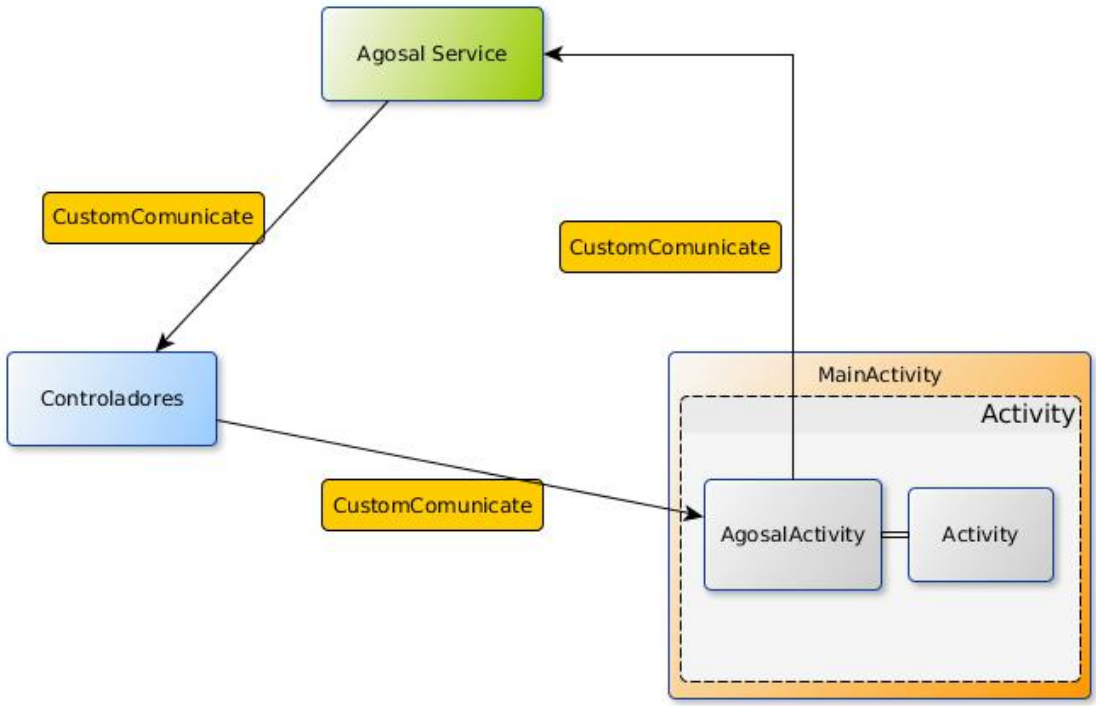

Ilustración 25. Paradigma MVC adaptado a las características de Android

Todas las funcionalidades que suministrará la Aplicación deberán estar basadas en esas entidades: Controllers, AgosalActivity y Activity. Teniendo que utilizar, de forma global y conjunta, el Service y MainActivity.

A continuación se detallarán las responsabilidades y características de cada entidad, así como las distintas clases que lo componen.

4.4.1. Componentes Base.

A. Service.

Se trata del punto de acceso de todas las peticiones realizadas por las Actividades (Vistas). Consiste en una extensión del objeto *Service* suministrado por Android, ofreciendo además de la ejecución en segundo plano, toda la gestión y control necesaria en los Controladores.

Posee la correspondencia entre Vistas y Controladores, notificando a estos últimos las peticiones solicitadas por las Vistas, rechazándola en los casos en los que no se encuentre un Controlador asociado. También posee el conector hacia el WebService (AgosalwebServer), ofreciendo una copia del mismo a cada Controlador cuando le notifica la llegada de una petición.

Además de esta funcionalidad reguladora, el Service se encarga de validar las Credenciales del usuario, cómo de bloquear la aplicación si se ha detectado inactividad.

Debido a que el Servicio necesita de una Vista con la que poder llevar a cabo estas acciones, el Servicio hace uso del objeto MainActivity, vista principal de la Aplicación y contenedora del resto de Vistas.

La entidad Service está constituida por dos clases: *ServiceBase* y *AgosalService*, siendo base una de la otra como se aprecia en la figura Ilustración 26.

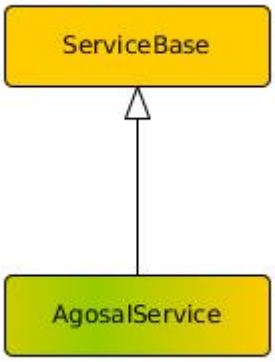

Ilustración 26. Diagrama de clases de la entidad Service

ServiceBase.java: Realiza toda la funcionalidad genérica del servicio. Se trata del prototipo base tras el cual se basará el Service de nuestra aplicación.

AgosalService.java: Extensión de la clase anterior, realiza toda la funcionalidad descrita del Agosal Service. Es en esta clase donde se debe instanciar la relación Controlador-Vista. Deberá capturar los mensajes procedentes de MainActivity y decidir qué hacer con ellos al margen del resto de peticiones.

B. Controladores.

Cada funcionalidad de la Aplicación estará controlada y gestionada por un Controlador en particular. Los controladores recibirán los eventos procedentes de las Vistas, a través del Service, realizando las tareas oportunas y notificando a la Vista emisora las acciones que debe hacer. Las acciones que realizará cada controlador se clasificarán en 3 tipos: Action,Query y Filter.

Las acciones de tipo Action harán mención a tareas procedimentales, como la edición de los datos. Si se trata de tipo Query se estará solicitando realizar una consulta y en el caso Filter, se deseará editar el filtro existente de la consulta actual. A parte del tipo de notificación se indicará la acción a realizar propiamente dicha, mediante un valor numérico. Se recomienda utilizar para estos valores las herramientas de configuración xml que ofrece Android, las cuales nos facilitarán la gestión de las acciones al tenerlas de forma estructuradas y bien ubicadas.

En la entidad Comunicate, se detallarán los distintos atributos y valores que participarán en el proceso de comunicación.

Los distintos controladores a utilizar deberán extender de la clase ControllerBase, la cual está prediseñada para facilitar el funcionamiento de los mismos. Se ha diseñado 3 funciones manejadoras, que recibirán la petición de la Vista según el tipo de solicitud (Action,Query y Filter). Esta clasificación previa nos facilita la gestión del controlador, ya que dispondremos de una mayor estructuración del código. Además posee funciones auxiliares para ayudar en la creación y envío de los objetos de comunicación, como en la gestión del conector del WebService.

C. MainActivity.

Se trata de la Vista contenedora del resto de Actividades de la aplicación. La visualización de las Vistas se realiza mediante el uso del recurso TabActivity suministrado por Android. A pesar de que este recurso pasó a estar obsoleto desde la versión 13 de la API de Android, hemos decidido utilizarlo ya que ofrece toda la capacidad que necesitamos.

Aun así, MainActivity se ha diseñado de forma escalar mediante diversas clases, ofreciendo funcionalidades independientes las unas de las otras. Con ello conseguimos que la sustitución futura del TabActivity requiera el menor trabajo posible, ya que solo se verán afectadas el menor número de ellas.

Otras de sus responsabilidades son: realizar todas las acciones indicadas por el Service, ofrecer al resto de Vistas el conector con el Service o instanciar, al comienzo de la aplicación, al propio servicio. Las acciones que serán enviadas por parte del Service son: AcceptedAccess, DenyAccess y ShowLogin.

En la figura Ilustración 27 se presentan la relación de las distintas clases que forman el componente MainActivity.

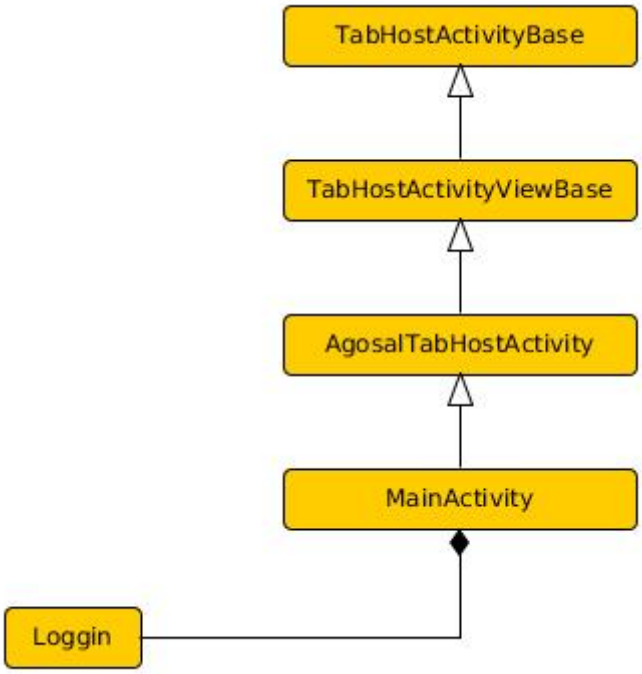

Ilustración 27. Diagrama de clases de la entidad MainActivity

TabHostActivityBase.java: Lleva a cabo la creación y arranque del Service, además de la construcción del protocolo de comunicación entre el Service y el MainActivity. Esta clase está diseñada a modo de interfaz, donde las clases extendidas deberán especificar ciertas características, como la clase que realizará de Service o las acciones a realizar como respuesta a las indicaciones del mismo.

TabHostActivityViewBase.java: Se le añadirá la capacidad de mostrarle al usuario acciones a realizar mediante el uso de un Slide. Aunque en nuestro caso no hicimos uso de esta característica, la introducimos por si algún desarrollador que utilice el módulo publicado pueda serle de utilidad.

AgosalTabHostActivity.java: Especificamos todas aquellas características necesarias para poder utilizar todas las herramientas ofrecidas por TabHostActivityBase. Concretamente se añadió la especificación de la clase que implementa nuestro Service (AgosalService), permitiéndonos así conectarnos y comunicarnos con ella.

Loggin.java: Actividad secundaria que mostrará la vista de Login. Esta vista no permitirá volver a la Actividad anterior, quedando la aplicación paralizada en este punto hasta que el usuario valide sus credenciales satisfactoriamente.

MainActivity.java: Vista principal del sistema que contendrá al resto de Actividades mediante el uso del TabActivity. En esta clase se debe definir las Actividades a utilizar, además de instanciar y conectarse con el Service. Será la responsable de realizar las distintas acciones indicadas por el Service, así como de suministrarle al resto de Vistas el conector para comunicarse directamente con él.

D. Activity.

Las Actividades (Activity) representan las Vistas de nuestro patrón MVC. Todas estas vistas están contenidas en MainActivity y solo podrá visualizarse una en cada momento. En este caso debemos separar el concepto Activity en dos niveles: Actividades nativas de Android y Actividades propias de Agosal.

- *Las Actividades nativas de Android*: son las utilizadas en los formularios de captura de datos o listados muy simples, los cuales no se diferencian de cualquier otra Actividad utilizada en Android.

- *Las Actividades propias de Agosal (AgosalActivity)*: se tratan de extensiones de las propias Actividades de Android. Estas Actividades constituyen la interfaz principal de cada funcionalidad, incorporando el protocolo de comunicación establecido entre ellas y los Controladores. Además gestionan los distintos eventos realizados por el usuario, notificando a los Controladores siempre que sea necesario.

Todas las AgosalActivity poseen unas acciones estándar que deberán realizar siempre que el controlador lo indique, las cuales son:

1. *ShowFilter*: Solicita que la Vista muestre el formulario para el filtro.
2. *ShowDialog*: Muestra un diálogo con el contenido enviado.
3. *AlertBox*: Indica que el mensaje enviado debe presentarse como alerta.
4. *ErrorBox*: Similar a AlertBox pero en este caso debe mostrarse como un error.
5. *ActionOk*: La acción enviada previamente se ha realizado correctamente.

6. *ActionDeny*: La acción enviada previamente NO se ha realizado correctamente.
7. *ShowList*: Se indica que muestra en la lista indicada el dataset enviado.

Este tipo de Vistas contienen un método específico para cada acción mostrada, en las cuales el desarrollador deberá indicar que acciones desea realizar. En el caso de no declarar una ejecución específica se llevará a cabo la funcionalidad estándar establecida.

Seguidamente se presentarán las distintas clases que componen a las Vistas AgosalActivity, haciendo referencia a sus características y responsabilidades.

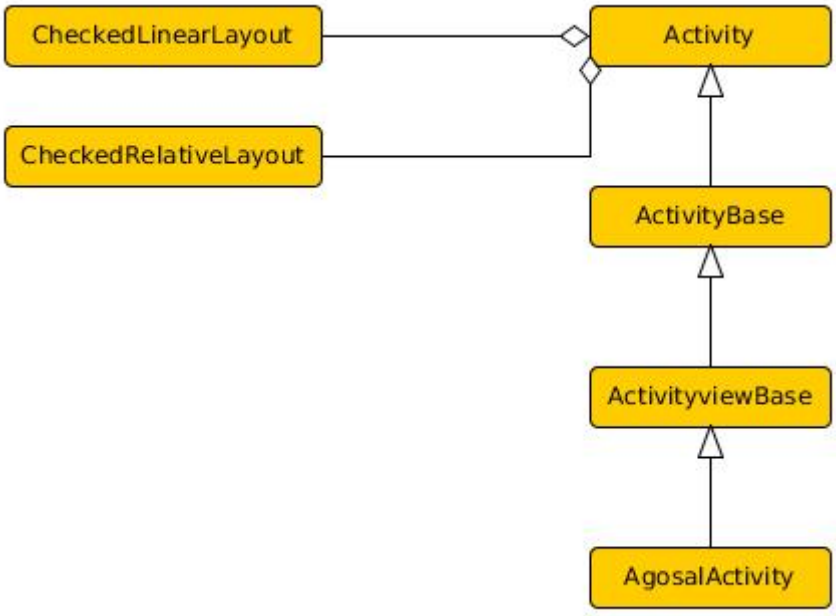

Ilustración 28. Diagrama de clases de la entidad AgosalActivity

ActivityBase.java: Extiende de la clase ACtivity de Android, implementando la interfaz Callback con la que se realizará la comunicación con el Service. En esta clase se define y realiza los controles y estructuras necesarias para la realización del protocolo de comunicación.

ActivityViewBase.java: Como ocurrió con *TabHostActivityViewBase*, incorporará el uso de un Slide con el que poder mostrar las acciones a realizar de una forma estructurada y atractiva. El uso de este componente lo realizamos

mediante la utilización del recurso SlidingDrawer, recientemente descatalogado desde la API 17. Al no disponer de alternativas para su sustitución y ser muy reciente su eliminación, se decidió continuar con su uso ya que se dispone del tiempo suficiente para cambiar el componente antes de que la plataforma Android no disponga de su compatibilidad. Por ello, uno de los futuros trabajos que se deberá realizar es la sustitución de este elemento por una solución soportada por la plataforma.

AgosalActivity: Representa la entidad que deberá instanciarse en cada funcionalidad a crear. Esta entidad será responsable de comunicarse con el Controlador y de proporcionarle al usuario las herramientas gráficas necesarias para la realización de las distintas acciones.

CheckedLinearLayout.java y CheckedRelativeLayout.java: Se tratan de extensiones de las clases LinearLayout y RelativeLayout, respectivamente. Son utilizadas en las interfaces gráficas para poder mantener reflejado (**Checked)** el item seleccionado de las distintas listas. Para ello se hizo de uso de la interfaz *Checkable*.

E. Comunicate

La comunicación entre las Vistas y los Controladores se fundamenta en la clase ***CustomComunicate***, la cual alberga en sus atributos toda la información necesaria para la correcta comunicación de peticiones. Algunos de sus atributos pueden cambiar su significado según el sentido de la comunicación, adaptándose al destinatario del mensaje. Los atributos que posee esta clase son:

Nombre	Tipo	Sentido: C -> V	Sentido: V -> C
actionID	Integer	Especifica la acción que debe realizar la Vista destinataria	Representa la acción que se le solicitará al Controlador
specificItemID	Integer	Se utilizará como subcategoría en los casos donde una acción pueda tener alternativas	
type	Integer	Asociado al tipo de acción (Action,Query y Filter).	

Nombre	Tipo	Sentido: C -> V	Sentido: V -> C
content	HashMap <String, Object>	Contendrá información adicional para que pueda finalizar la tarea indicada. Por ejemplo: el mensaje a mostrar o el código de error.	Contendrá la pareja clave-valor con los datos necesarios para realizar la acción solicitada.
dataset	ArrayList<Object>	Almacenará los datos que deberá mostrar la Vista en la lista.	No será utilizado

El proceso de comunicación se realiza mediante el envío de objetos Message, a través del canal de comunicación generado por los objetos Messenger. El objeto CustomComunicate a enviar se introduce en el atributo obj de la clase Message. Si el remitente del mensaje es una Vista, se añadirá al atributo replyTo, del objeto Message, una referencia hacia la instancia del objeto Messenger de la Vista, permitiendo que el Controlador pueda enviar su respuesta a la Vista.

4.4.2. Módulos Funcionales.

Como se mostró en la figura Ilustración 25, todas las funcionalidades de nuestra aplicación estarán basadas en el diseño MVC detallado, constituidas por los componentes Controllers, AgosalActivity y Activity.

Para facilitar su desarrollo, se agruparon las distintas funcionalidades en paquetes Java independientes entre sí. Con ello conseguimos tener mejor localizadas las clases y que las funcionalidades dependan únicamente de sí mismas y del Service, favoreciendo la tolerancia a cambios.

Antes de mostrar los paquetes generados, junto a las clases que los componen y sus responsabilidades, hay que indicar que cada módulo posee un componente derivado de la clase ***AdapterList***, la cual nos facilitará la gestión y personalización de las distintas listas a utilizar. Los elementos derivados de esta clase se nombrarán siguiendo el patrón XXAdapterList, donde XX reflejará la lista a gestionar. Al tratar las listas de forma individual es posible que algunos módulos posean más de un componente de este tipo si manipulan diferentes listas.

Calendario

Las clases de este paquete se encargarán de realizar las diversas funcionalidades atribuidas al uso del calendario.

- **CalendarCard**: Es el formulario con el que se obtendrán los datos para la creación y edición de notas.
- **CalendarController**: Realiza el control de la funcionalidad. Extiende de la clase Controller, heredando las responsabilidades de esta.
- **Calendario**: Constituye la Vista AgosalActivity de la funcionalidad. Esta clase gestionará la utilización del CalendarCard y se comunicará con el Controlador asociado.

Contactos.

Este módulo llevará a cabo la gestión y manipulación de los contactos del usuario. Para ello hará uso de las clases:

- **ContactCard**: Administra el formulario de datos utilizado para la creación y edición de los diversos contactos.
- **ContactController**: Se trata del Controlador de la funcionalidad, atendiendo las peticiones enviadas por parte de la Vista.
- **Contacts**: Es la Vista principal de la funcionalidad, gestionando las acciones realizadas por el usuario.

Mensajería.

El envío y recepción de mensajes se realizará a través del paquete Mensajería, gestionando los distintos estados que adquirirán los mensajes.

- **Messenger**: Implementa el componente AgosalActivity de la funcionalidad. Presentará los mensajes del usuario permitiendo elegir la acción que se desea realizar.
- **MessengerCard**: Vista auxiliar utilizada para mostrar y construir el contenido de los mensajes.
- **MessengerController**: Es el Controlador de la funcionalidad. En esta clase se centraliza el control y gestión de las acciones solicitadas por el usuario.

Expedientes.

Este paquete será el encargado de llevar a cabo la gestión y edición referida a la utilización de Expedientes.

- **Files**: Lleva a cabo la funcionalidad del componente AgosalActivity. Le permitirá al usuario realizar las distintas acciones pertenecientes a la gestión de expedientes, mostrándoles los expedientes registrados.
- **FilesCard**: Vista utilizada como formulario de datos en el que el usuario podrá rellenar los campos necesarios para realizar la acción solicitada.
- **FilesController**: Representa al Controlador del módulo. Gestionará todas las peticiones enviadas por la Vista, decidiendo que acciones se deberá realizar.

Procedimientos.

La creación y utilización de los procedimientos, como su asociación a Expedientes vendrá dada por los componentes de este módulo. Debido a la tarea que realiza, podría considerarse este paquete como un subcomponente del módulo *Expedientes*, ya que su funcionalidad vendrá gestionada por la utilización de dicho módulo.

- **Procedures**: Le permitirá al usuario visualizar los Procedimientos asociados a un Expedientes indicado previamente. Además le dará acceso al usuario a las distintas acciones permitidas.
- **ProceduresCard**: Construye el formulario de datos utilizado para insertar y visualizar la información perteneciente a un Procedimiento.
- **ProceduresController**: Implementa el Controlador de la funcionalidad, gestionando todas las acciones solicitadas por el usuario a través de la Vista.

5. Componente Web.

La responsabilidad de este componente será permitir el acceso al sistema mediante la utilización de un portal Web. Está página estará diseñada de tal forma que ofrezca una interfaz gráfica intuitiva, facilitando la utilización de las distintas características que ofrece.

A pesar de no haberse podido implementar, debido a las restricciones temporales del proyecto, sí se llevaron a cabo las distintas fases de análisis y diseño previas al desarrollo.

En los siguientes apartados se presentará la tecnología y diseño seleccionados para la creación del subsistema.

5.1. Arquitectura tecnológica seleccionada.

Bootstrap.

Framework Front-End diseñado para facilitar la construcción de páginas web, ofreciendo una estructura de HTML/CSS constituida por templates. Bootstrap permite adaptar la visualización de la Web según el dispositivo en el que se muestre, ajustándose a las dimensiones del mismo. Esto se debe a la utilización por parte del framework de la nueva tecnología CSS Responsive, la cual nos suministra dicha funcionalidad.

Debido a estas características se decidió utilizar Bootstrap como plataforma base para construir las distintas vistas que formarán nuestra Web.

LESS.

Se trata de un lenguaje orientado para la construcción de hojas de diseño dinámicas. Este lenguaje es utilizado para construir ficheros CSS de una forma sencilla y dinámica, permitiendo obtener los valores de los atributos a través de funciones o variables establecidas de forma global.

Se decidió utilizar esta herramienta para facilitar la adaptación de diversos templates encontrados en bootstrap. Además, dichos templates se encuentran

construidos mediante LESS, por lo que su conocimiento será de gran ayuda para su manipulación.

Framework Zend

Zend Framework (ZF) es un framework de código abierto para desarrollar aplicaciones web y servicios web con PHP 5. ZF es una implementación que usa código 100% orientado a objetos. La estructura de los componentes de ZF es algo único; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado. A menudo se refiere a este tipo de diseño como "use-at-will" (uso a voluntad).

Este framework, conocido también como ZF (Zend Framework), es un sistema de código abierto diseñado para facilitar la creación de aplicaciones y servicios web mediante la utilización de PHP 5.

La estructura de ZF es algo única ya que cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado.

Es debido a estas características por lo que se decidió introducir la utilización de ZF, además del alto rendimiento y estabilidad que proporciona.

PHP

PHP es un lenguaje de programación de uso general de código del lado del servidor, utilizado especialmente para el desarrollo de aplicaciones Web. La utilización de este lenguaje viene asociada por la selección del framework Zend, aunque sus características y módulos que posee lo hacen idóneo para nuestras necesidades.

jQuery

Es una librería de javascript publicada como software libre que permite poder realizar funcionalidades una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más esfuerzo. Debido a esta ayuda en el desarrollo se eligió utilizar esta tecnología, la cual además ofrece mecanismos preestablecidos para utilizar la tecnología AJAX.

Ajax

AJAX (**A**synchronous **J**avaScript **A**nd **X**ML) es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el navegador del cliente realizando una comunicación asíncrona con el servidor en segundo plano. El principal motivo por el que se decidió utilizar esta tecnología es que permite realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las mismas.

Además, Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos ya que está basada en estándares abiertos como JavaScript y Document Object Model (DOM).

5.2. Diseño planteado.

La utilización del framework Zend nos obliga a seguir el paradigma MVC, coincidiendo con nuestra filosofía de trabajo hasta ahora. Como en el resto de componentes, utilizaremos una pareja Vista-Controlador por cada funcionalidad a desarrollar.

El Modelo del paradigma se implementará mediante la creación de una librería que se incluirá al framework Zend, la cual simulará el comportamiento y responsabilidades del WebService. Se ha decidido utilizar este método, en lugar de las herramientas proporcionadas por Zend, para seguir una misma filosofía entre los distintos componentes del sistema.

Para mejorar el rendimiento de la Web, se establecerá una estructura visual base desde la cual se desarrollarán las distintas Vistas, adaptando su configuración mediante uso coordinado de AJAX y jQuery. Con la incorporación de esta tecnología conseguiremos reducir el número de componentes enviados desde el servidor Web, reduciendo considerablemente el tráfico de datos. Esta característica será de gran importancia durante la utilización de dispositivos con tecnologías móviles, como 3G o 4G/LTE.

A continuación se presentarán los prototipos de Vistas diseñadas para la realización del subsistema. Cómo se puede apreciar todas las Vistas parten de un diseño base, adaptando su presentación a la funcionalidad actual.

5.2.1. Prototipo de vistas.

Vista general.

En esta vista se presentará información general sobre la plataforma, indicando las diversas características que ofrece.

Su principal responsabilidad será la gestión de acceso al sistema, ya que deberá validar el loggin del usuario. Una vez aceptadas las credenciales del usuario se almacenarán en la sesión establecida, permitiendo que el resto de componentes puedan acceder a ellas. En este aspecto hay que tener en consideración la necesidad de encriptar estos datos almacenados, evitando que puedan ser utilizados por personas ajenas.

La vista prediseñada para esta funcionalidad es:

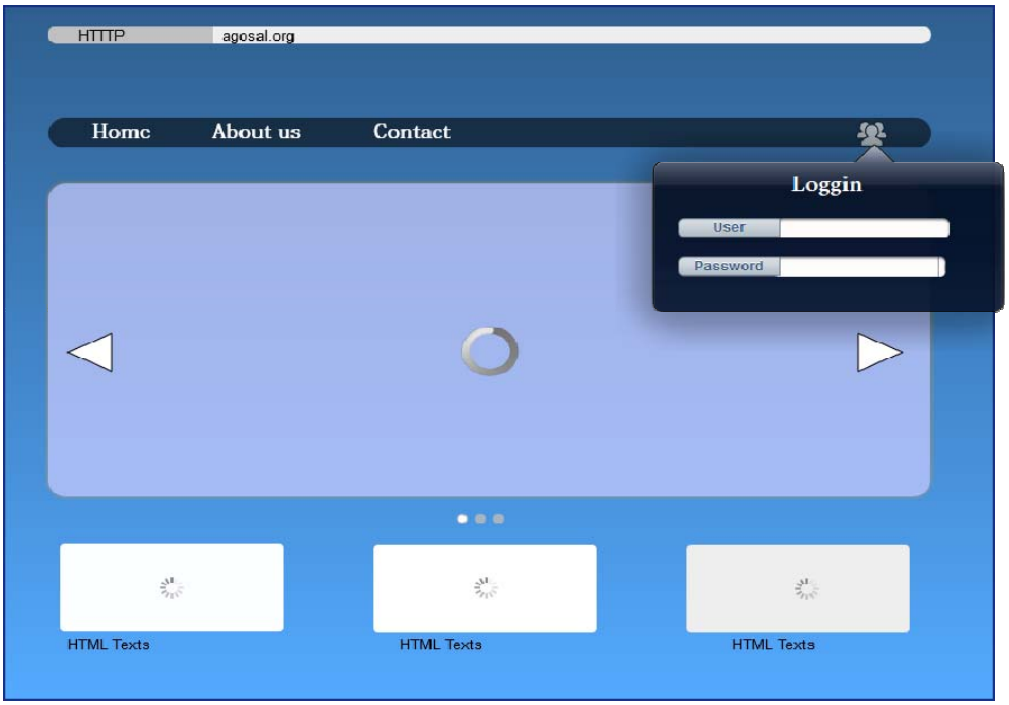

Ilustración 29. Vista asociada a la presentación inicial

Vista de mensajería.

Para este caso se mostrarán todos los mensajes referentes al usuario, tanto recibidos, enviados o eliminados. En esta Vista se le permitirá al usuario realizar todas las acciones relacionadas con el servicio de mensajería.

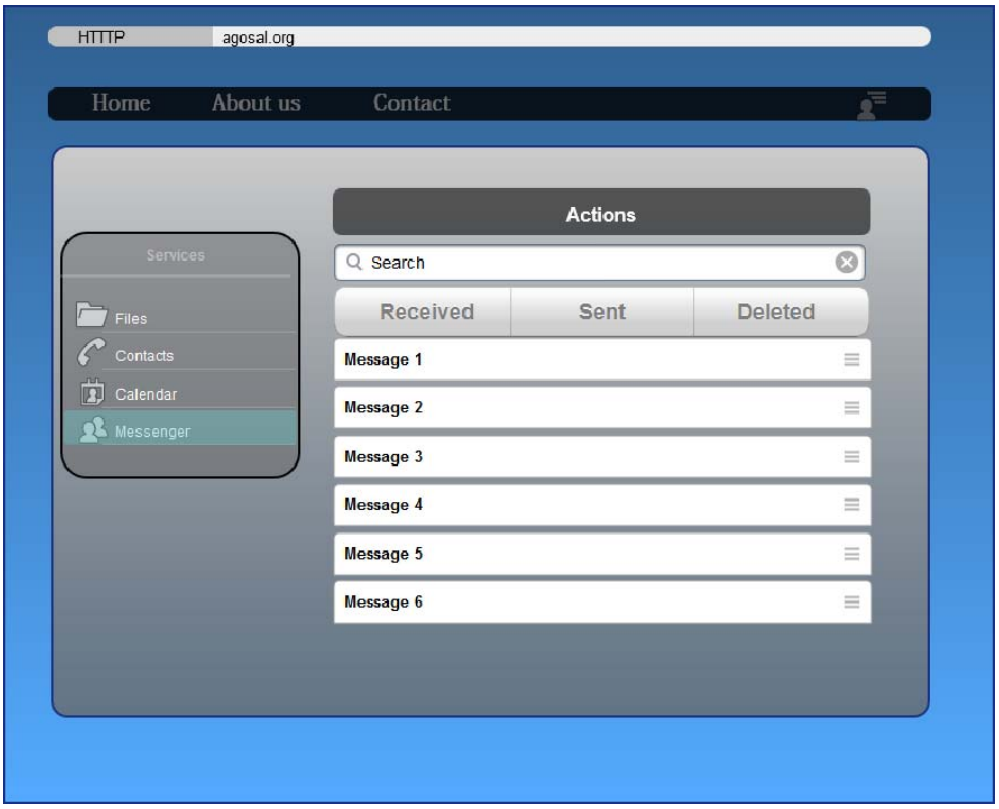

Ilustración 30. Vista asociada al servicio de mensajería.

Vista de contactos.

Responsable de mostrar los contactos del usuario, clasificándolos por el tipo al que pertenecen. El usuario podrá llevar a cabo todas acciones ofrecidas para la gestión de contactos.

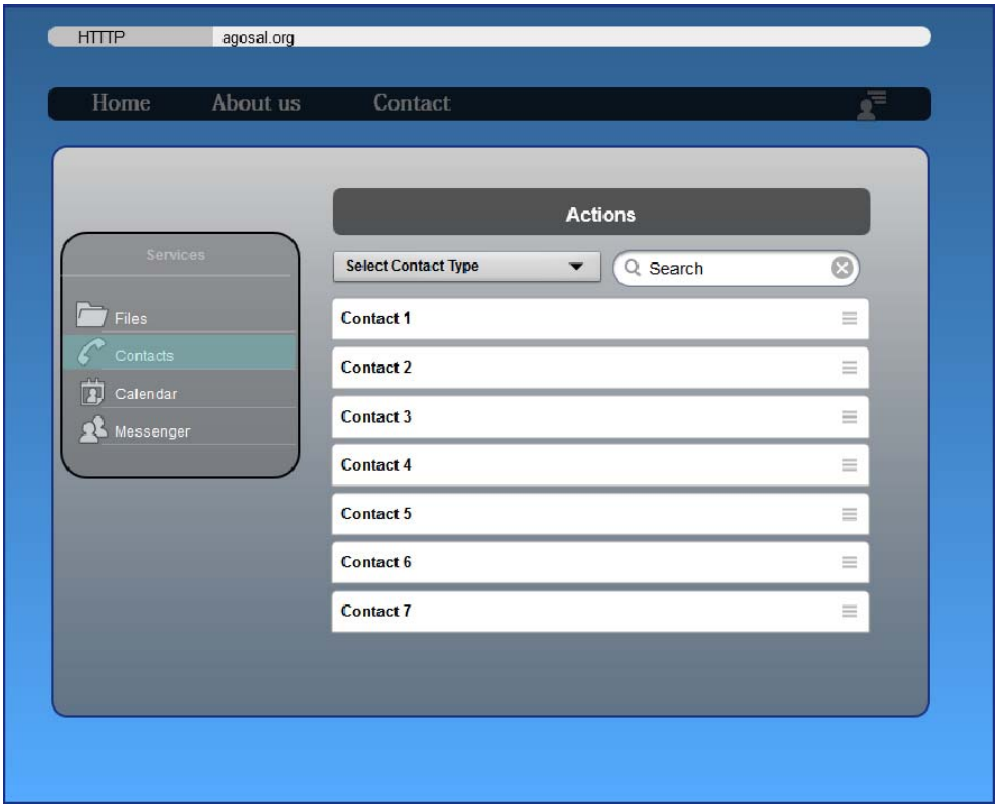

Ilustración 31. Vista asociada a la gestión de Contactos.

Vista de calendario.

Se presentará un Calendario del mes actual, en el cual se señalarán aquellos días que posean un recordatorio o nota en ellos. Justo debajo se indicará los eventos que posee el día señalado. En el caso de no señalar ningún día se mostrará aquellos recordatorios que se cumplirán en los próximos 7 días, a partir del día actual.

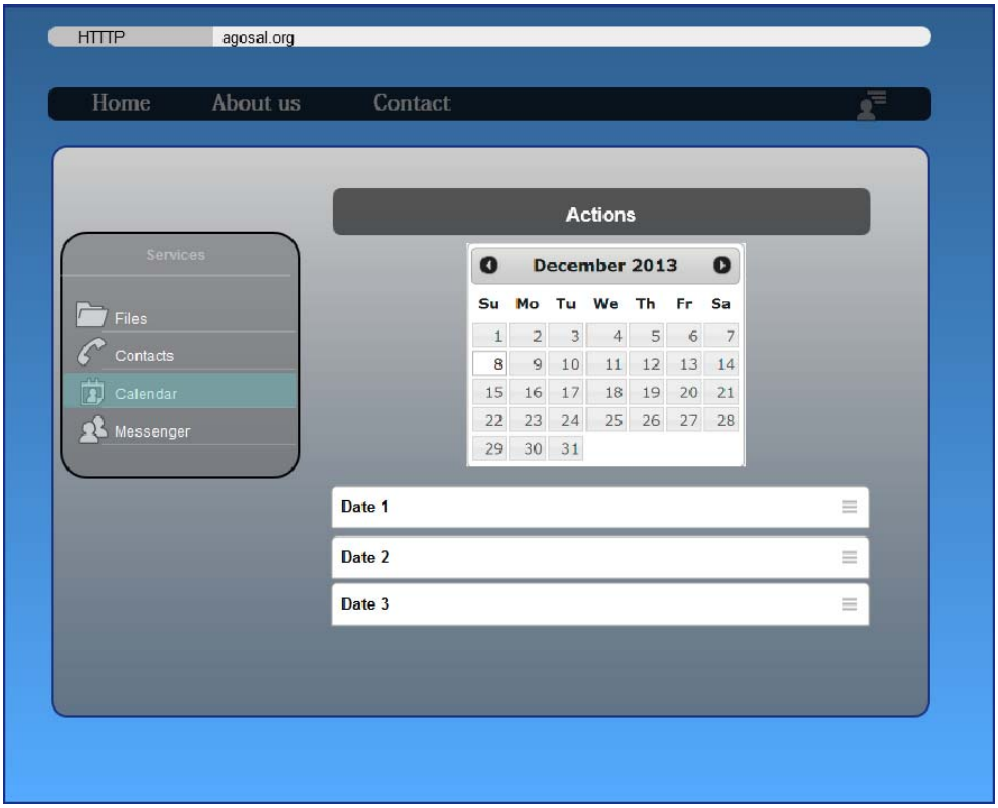

Ilustración 32. Vista asociada a la funcionalidad de Agenda.

Vista de expedientes.

La gestión de expedientes tendrá una Vista particular en la que se podrá realizar filtrar los expedientes presentados. Cuando se acceda a un expediente en concreto se mostrará toda la información y procedimientos asociada al expediente.

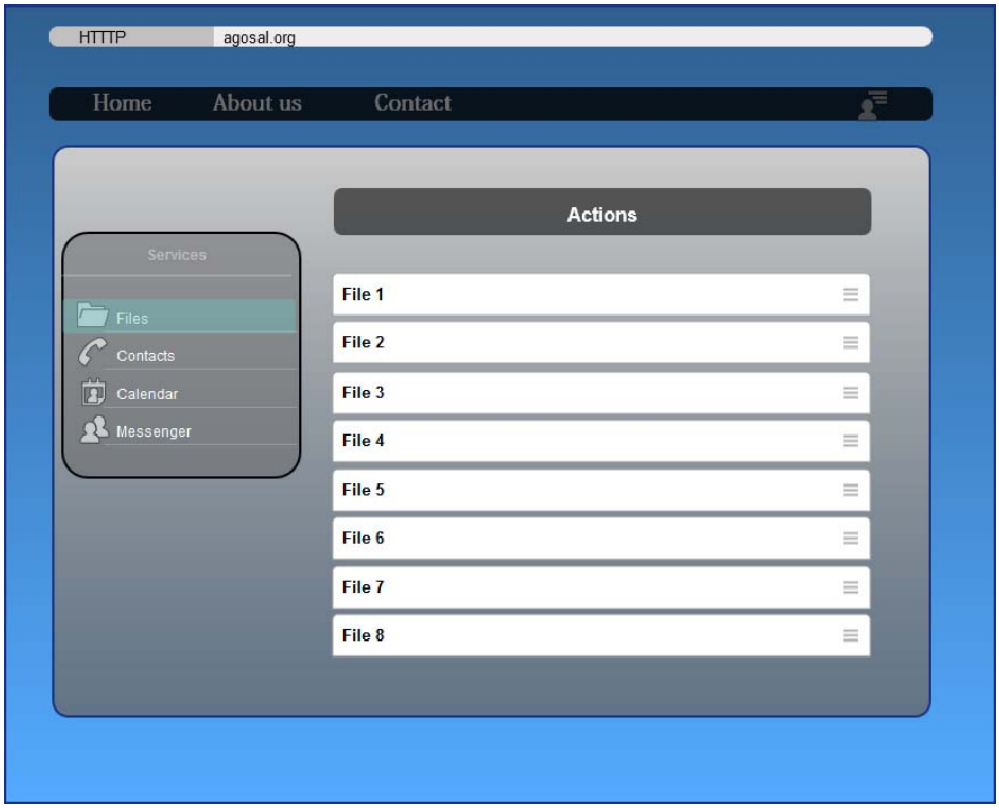

Ilustración 33. Vista asociada para la administración de Expedientes.

Vista de procedimientos.

Esta Vista puede considerarse una subvista de la anterior al estar supeditada su utilización a esta. Una vez seleccionado un procedimiento se mostrará toda la información relacionada con él, como: el tipo de procedimiento, el identificador, documentos asociados, etc.

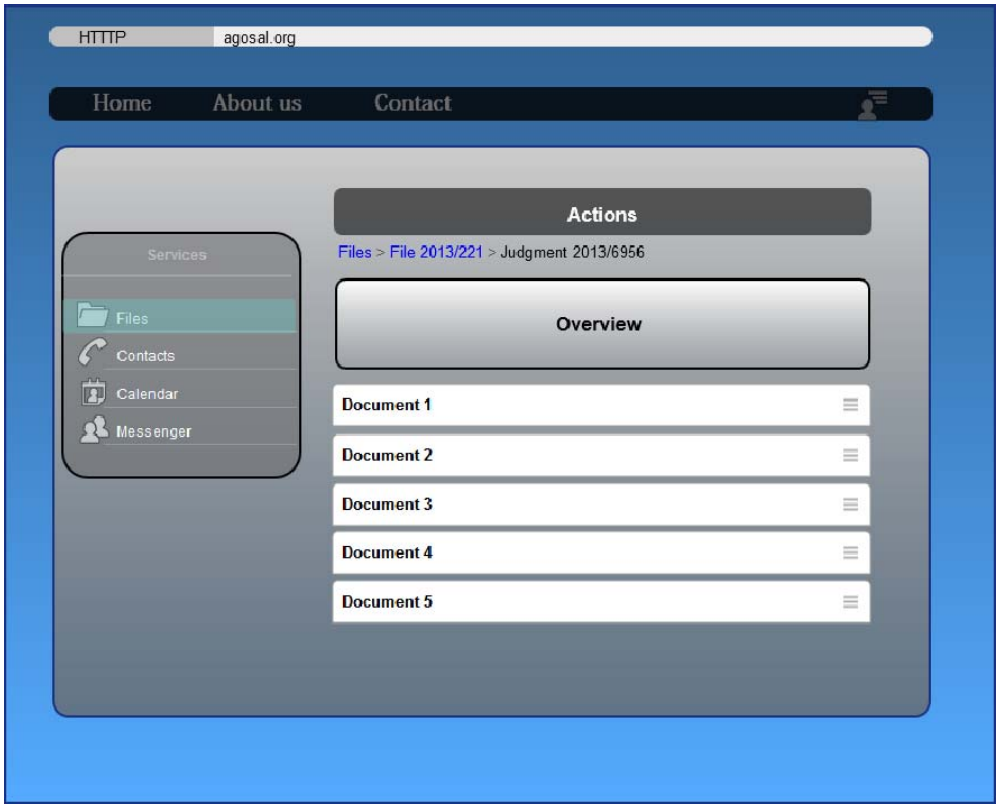

Ilustración 34. Vista asociada para la gestión de Procedimientos

6. Perspectivas Futuras

6.1. Añadir documentos

La capacidad del sistema podría aumentarse si introducimos la capacidad de adjuntar documentos reales. Este nuevo servicio consiste en alojar en la plataforma, aquellos archivos que el usuario ha vinculado a un procedimiento cualquiera.

Con ello se logrará que el usuario disponga del documento en cualquier componente del sistema, teniendo siempre la versión más actual. Dentro de las acciones que podrá realizar se encuentra:

1. La inserción de un nuevo Escrito (archivo)
2. La actualización de un Escrito existente, pudiendo sustituir el archivo existente por uno nuevo.
3. La sustitución de un archivo se reflejará en el historial del Escrito, pudiendo recuperar hasta las 15 últimas modificaciones.
4. Eliminar el Escrito o el documento asociado a este.

La realización de esta funcionalidad se verá favorecida por la estructura actual de la BD, ya que esta se encuentra prediseñada para la asociación de documentos. En estos momentos se vincula los Escritos con el directorio en el que se encuentra el archivo referenciado, es decir, se almacena la ruta y no el contenido de este. Por ello el cambio principal se vería realizado en este aspecto y en el propio envío del documento al Servidor central.

6.2. Gestión Contable

Además de la gestión administrativa, podría añadirse la gestión contable del usuario final. Con esta gestión se le permitirá al cliente poder llevar a cabo ciertos registros y acciones relacionadas con su actividad económica. Las utilidades que se le aportarían al usuario son:

1. Realización automática de facturas.
2. Registro temporal de la actividad realizada, pudiendo elegir diversas opciones como: trimestral, semestral, anual, etc.

3. Contabilización de pagos pendientes. En este caso el usuario podrá ver cuanto se ha ingreso por los servicios prestados, pudiendo poner recordatorios en aquellos pagos pendientes para una futura revisión.

6.3. Sistema de alta disponibilidad

La arquitectura de *Alta Disponibilidad* se fundamentara en el diseño presentado en el apartado *2.2.4. Alta disponibilidad.* Este diseño está constituido por 5 capas, a modo de niveles, donde cada una de ellas poseerá una responsabilidad independiente pero coordinada con el resto de ellas, permitiendo el correcto funcionamiento del *Sistema de Alta Disponibilidad*.

La plataforma tecnológica para su implementación estará enfocada en la utilización, en todo lo posible, de componentes publicados como Software Libre o licencias similares, como Open Source. Para facilitar su lectura se expondrá los distintos elementos utilizados según el nivel que lo contenga.

Capa 1: Firewall - Cortafuegos.

Esta capa estará formada por un servidor físico de tipo RAIC gestionado por el S.O. Ubuntu Server 12.4. Las medidas de seguridad se realizarán mediante la utilización del firewall IPTables, el cual nos permitirá poder bloquear el acceso de cualquier conexión entrante.

Cualquier tarea periódica que deba ejecutarse se realizará mediante la utilización del servicio Cron, como la comprobación del estado del sistema o la revisión de la BlackList, que se mostrará en los siguientes puntos.

La comunicación de las IPs que deban ser bloqueadas se llevará a cabo mediante la creación de un servidor TCP/IP interno con el cual se comunicará la Capa 1 y 3. Este servidor podrá ser construido con cualquier lenguaje que soporte la utilización socket. Siendo no fundamental la utilización del protocolo TLS, debido a que no es estrictamente necesario en este punto cifrar la comunicación, al encontrarse en el interior del sistema.

Capa 2: load balancer - Balanceador de Carga.

El balanceador de carga podrá estar formado por dos alternativas: por un dispositivo físico, que se encargará de repartir las conexiones entrantes o por el servicio IPS alojado en el Servidor anterior.

Respecto al dispositivo físico podemos elegir prácticamente cualquier solución existente en el mercado, desde modelos Barracuda, TP-Link, Edimax, etc.

La utilización del servicio IPS nos permitirá reutilizar el Servidor Cortafuego, reduciendo los costes del sistema, así como poder abstraernos de ciertas restricciones físicas derivadas de la utilización del componente anterior.

IPS proporciona la capacidad de configurar el funcionamiento del balanceo, pudiendo elegir el método de reenvío (NAT, Tunneling y Direct Routing) y entre 8 algoritmos de balanceo de cargas. Para controlar los casos en los que uno de los nodos de balanceo cae, se utiliza el servicio Keepalived conjuntamente con IPS, gestionando así estos casos.

Aunque en este momento no se ha decidido que opción utilizar, por lo que se realizará un estudio para ello, si es cierto que tenemos una preferencia a priori por la utilización del servicio IPS. Esta predilección es debida al control total que poseeríamos del servicio, además de la facilidad de configuración del balanceo ante la naturaleza de los nodos de la capa inferior.

Capa 3: atención de peticiones - care requests

Como se detalló en la sección *2.2.4. Alta disponibilidad.*, esta capa será la encargada de atender todas las peticiones enviadas por parte de los diversos componentes del sistema.

Por ello, para controlar los costes de construcción del sistema y aprovechar al máximo los recursos de los que se disponen, se ha decidido utilizar un sistema de virtualización con el que se creará los distintos nodos que formarán esta capa.

El sistema de virtualización base (HOST) podrá estar formado por 2 opciones:

A. Utilización del producto Citrix Xen: Consiste en un producto de la compañía Citrix que utiliza el hipervisor Xen sobre el sistema operativo CentOS con un kernel optimizado.

B. Utilizar el proyecto de virtualización KVM sobre cualquier plataforma Linux: Esta opción consistirá en utilizar las herramientas de virtualización del proyecto KVM, proporcionadas por cualquier distribución de Linux de la actualidad.

Al margen de la constitución del sistema Host, que deberá estudiarse en el futuro, nos centraremos en la composición de los nodos a virtualizar.

Estos nodos estarán formados por la distribución Ubuntu Server 12.04, la cual utilizará el servidor web Apache para dar soporte tanto al componente Web como al WebService.

Capa 4: Cluster de Sistemas Gestores de Bases de Datos

Esta capa estará formada por un cluster de máquinas virtuales, suministrando el servicio MySQL. El sistema Host de virtualización será el mismo que se implante en la capa anterior, utilizando también para los nodos virtualizados la distribución Ubuntu Server 12.04.

El cluster de este nivel estará elaborado por el sistema **High Availability** de RedHat, el cual nos permitirá disponer de hasta16 nodos de forma gratuita. Se ha decidido utilizar esta tecnología por su conocimiento previo y su buen funcionamiento, aunque no se ha descartado utilizar otras alternativas como el proyecto de Software Libre *Pacemaker*.

La utilización del cluster grantizará que siempre exista un nodo ofreciendo el servicio de MySQL, pudiendo elegir distintos métodos para los casos en los que se detecte la caída de un nodo. El servicio MySQL de cada nodo virtualizado deberá estar configurado para alojar toda la información y configuración del servicio en los discos exportados por la capa número 5.

Capa 5: Almacenamiento Distribuido

En este nivel se exportará los discos utilizados para almacenar la información procedente del SGBD.

Para ello se dispondrá de una o varias máquinas que exportarán discos iSCSI, apoyándose de la plataforma Linux para ello. El Sistema Operativo base podrá ser una distribución CentOS o la versión de Ubuntu Server 12.04.

Los discos exportados serán del tipo SATA de alta densidad y velocidad, capaces de soportar sistemas de alta frecuencia de acceso. Estarán configurados para llevar a cabo un sistema RAID 10, el cual distribuye y duplica la información entre todos los discos seleccionado, obteniendo así una baja latencia de búsqueda y una copia de seguridad a nivel físico.

En la siguiente imagen se presentará la adaptación del diseño previo tras la inclusión de las tecnologías seleccionadas.

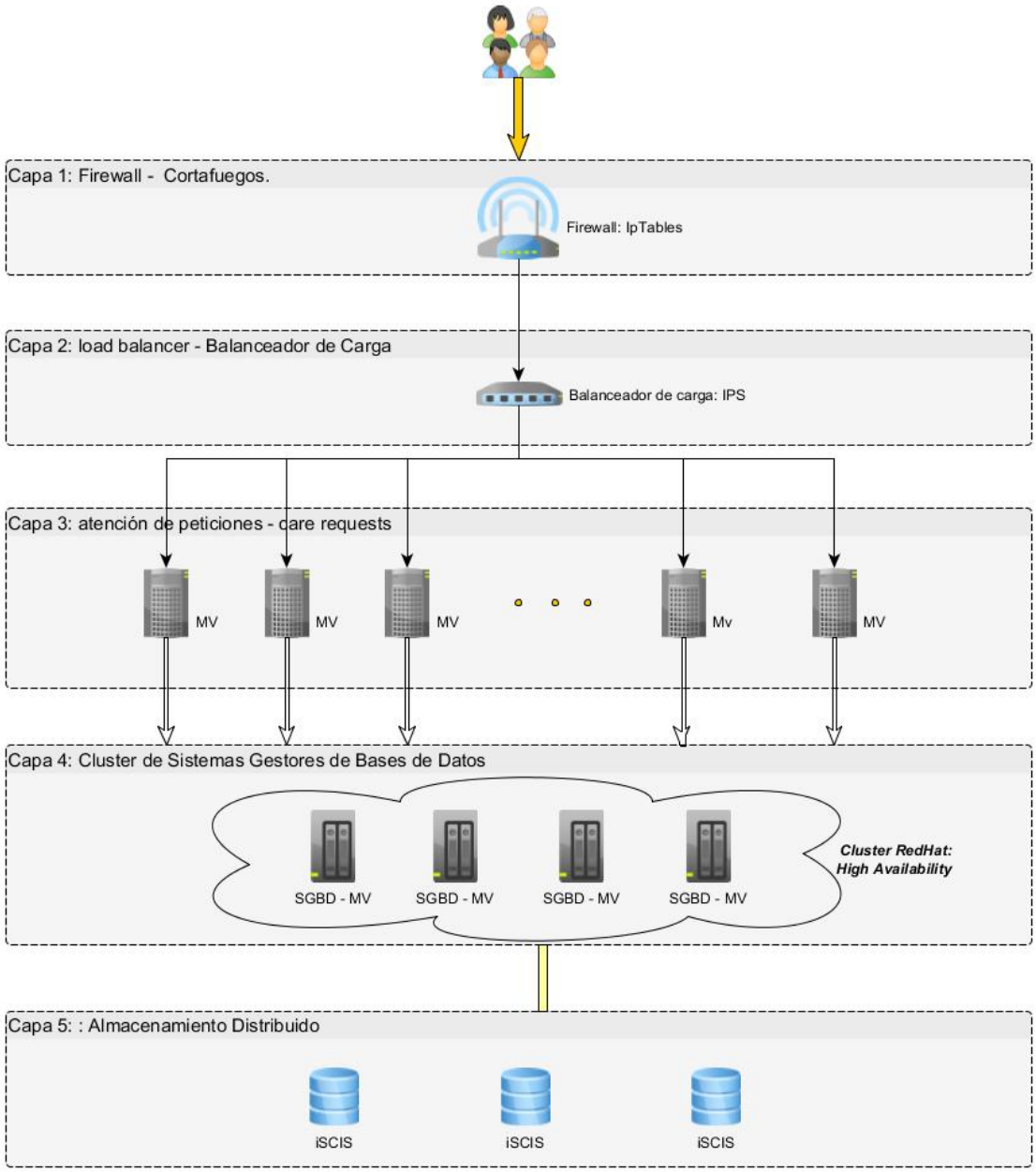

Ilustración 35. Inserción tecnológica a la arquitectura de Alta Disponibilidad diseñada

6.4. Utilización del WebService por parte del Subsistema de Escritorio

Una de las posibles mejoras que se puede realizar sobre el sistema es la sustitución del conector MySQL del *Subsistema de Escritori*o. Dicho conector podría sustituirse por una conexión con el WebService, permitiendo una mayor abstracción y generalidad del sistema, ya que el *Subsistema de Escritori*o desconocería el SGBD a utilizar.

Con ello se consigue un mayor control de los accesos a la BD, debido a que tenemos la capacidad de gestionar las propias conexiones, incluso defendernos ante posibles ataques de DDoS. Otra característica que podemos alcanzar es la utilización y edición de operaciones preestablecidas, eliminando en el subsistema la necesidad de conocer que tareas conlleva. Además, que el SGBD permanezca totalmente oculto para usuarios ajenos al sistema produce un aumento de la seguridad global del sistema.

Por lo tanto, con esta sustitución, el flujo de comunicación diseñado inicialmente evolucionaría tal y como se muestra en el diagrama Ilustración 36.

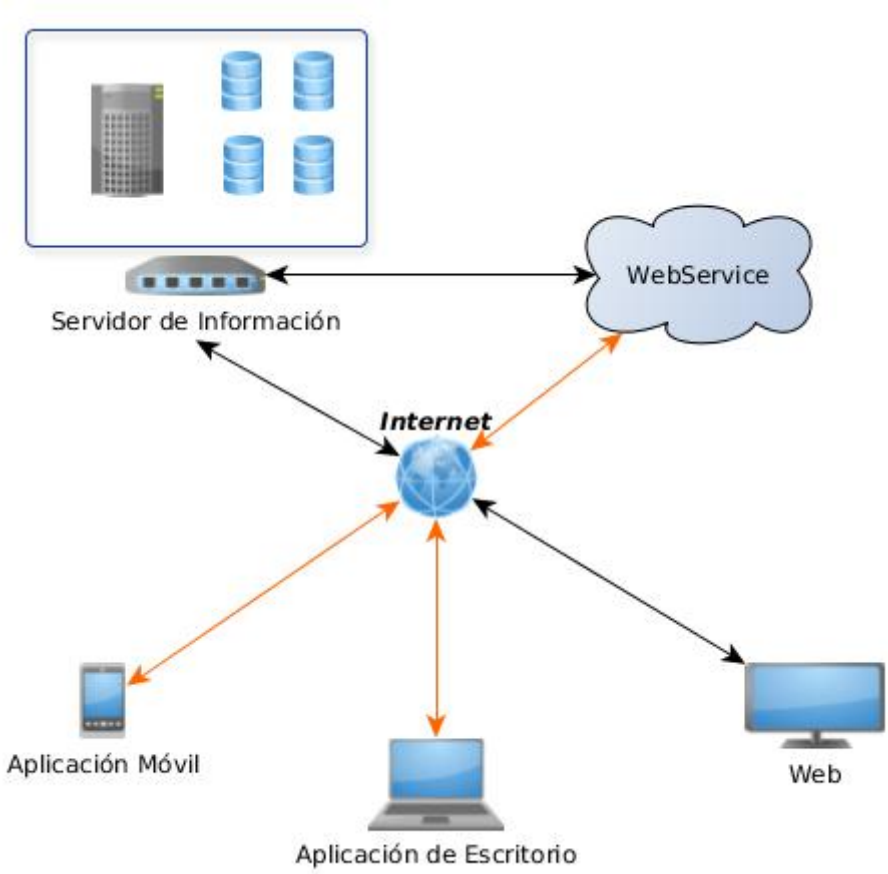

Ilustración 36. Adaptación del flujo de comunicación tras la utilización del WebService

6.5. Bloqueo automático de accesos (BlackList).

Un aumento de la seguridad del sistema vendría asociado por la utilización de un servicio de BlackList o lista negra. Este componente está fuertemente relacionado con el sistema de Alta disponibilidad mostrado anteriormente, y se encarga de bloquear el acceso a usuarios e IPs que hayan realizado una actividad considerada como errónea o imputable como ataque.

El BlackList se encontraría ubicado en la firewall de la capa 1 diseñada en sistema de Alta disponibilidad y utilizaría para la gestión de su funcionamiento tanto los nodos WebService de la capa 3 junto al propio firewall.

El funcionamiento prediseñado inicialmente para el BlackList consistirá en los siguientes pasos:

1. Arranque del sistema.

2. Se deniegan todos los tipos de accesos externos a la capa 1 (firewall), y por lo tanto al sistema.
3. Cuando el SGBD se encuentra arrancado, se pone en marcha en marcha el BlackList junto a los nodos que ofrecerán el WebService.
4. El BlackList situado en el firewall, observará la lista de IP denegadas antes del apagado del sistema y habilitará el acceso al mismo, denegando aquellas IPs que continúen prohibidas.
5. El control de acceso se rige por el propio servicio de BlackList:
    a. Cada 10 minutos se revisa el listado, reasignado aquellas IPs que hayan cumplido la penalización.
    b. Los nodos WebService notificarán inmediatamente al servicio BlackList aquellas IPs que se hayan realizado una actividad penalizable.

El procedimiento para seleccionar una actividad como no permitida, se fundamenta en 3 niveles, cada uno más restrictivo que el anterior.

Nivel 1: Posible error humano.

Este nivel se realiza cuando la comprobación de credenciales del usuario ha sido denegada en varias ocasiones en un corto plazo de tiempo. Este nivel está diseñado para defender a los usuarios de un posible ataque de fuerza bruta hacia sus credenciales.

Inicialmente se bloqueará el acceso del usuario, no de la IP, durante 10 minutos. Si tras este tiempo se repite el error se trasladará al nivel de error 11, en el cual se denegará el acceso durante 1 hora. De seguir produciéndose el error se irá trasladándose por los distintos niveles de error hasta bloquear permanentemente el acceso. Los niveles subyacentes a este tipo son:

1. Nivel 1 : Bloqueo 10 minutos
2. Nivel 11: Bloqueo de 1 hora.
3. Nivel 12: Bloqueo de 4 horas.
4. Nivel 13: Bloqueo permanente.

Nivel 2: Detectado posible ataque de denegación de servicios (DoS).

Se controlará el número estimado de accesos por minuto de la pareja usuario e IP. Si alguna pareja sobrepasa el límite establecida entrará en este nivel de control.

Como en el caso anterior existirán diversos subniveles que irán gestionando la evolución del elemento marcado.

1. Nivel 2 : Bloqueo 10 minutos
2. Nivel 21: Bloqueo de 3 horas.
3. Nivel 22: Bloqueo permanente.

Nivel 3: Uso incorrecto de la API establecida.

El error en la utilización de la API establecida se considera directamente un ataque malicioso al sistema, ya que este protocolo nunca deberá ser erróneo al ser utilizado únicamente por componentes construidos por nosotros mismos. Concretamente se considera un ataque de análisis, en el cual se intenta vislumbrar cualquier tipo de vulnerabilidad en nuestro sistema. Siempre que se detecte este tipo de error se bloqueará el acceso a dicha IP de forma permanente.

En el caso de que un usuario se vea afectado por la restricción de cualquier nivel, podrá notificárselo al administrador del sistema para, tras una identificación previa, eliminar dicho bloqueo.

El servicio de BlackList podrá verse evolucionado con el tiempo añadiendo nuevos controles de seguridad como puede ser la detección de ataques de análisis más sofisticados o recopilación y detección de patrones de ataques en un periodo más alargado.

6.6. Desarrollo del componente Web

Para completar el diseño inicial del proyecto es necesario que se desarrolle el componente Web planteado en la sección *5. Componente Web.*.

Con ello se enriquecerá al proyecto, añadiendo toda la potencia y flexibilidad que atribuye la utilización de portales Web. Su implementación le permitirá al usuario llevar a cabo la gestión de su actividad desde cualquier equipo y en cualquier parte del mundo, de una forma sencilla e intuitiva.

Además con la incorporación de componente conseguimos un efecto publicitario ya que existe la posibilidad de aparecer en los resultados de buscadores como Google, Yahoo, Bing etc.

El tiempo estimado para su realización, considerando las fases de desarrollo, de prueba y de documentación, se encuentra entre las 4 ó 6 semanas.

6.7. Componente Móvil para el resto de plataformas.

Una vez concluido el sistema, se iniciará una fase de expansión en la que se adaptará el componente Móvil para las diversas plataformas restantes. Concretamente se desarrollará para los sistemas iOS y Windows Phone, los cuales poseen la mayoría del mercado junto con Android. En esta fase se han descartado los sistemas Mozilla OS y Ubuntu Mobile debido a su reciente incorporación y sobre todo por la incertidumbre sobre su continuidad.

Debido a la necesidad de aprendizaje de cada plataforma, se tomará un tiempo de desarrollo mayor al planteado inicialmente. En este tiempo se abarcará desde el propio aprendizaje hasta las fases alfa y beta de las aplicaciones. El tiempo considerado para la realización de cada aplicación oscila entre las 15 y 16 semanas. En la figura Ilustración 37 se presenta la planificación de las distintas etapas planteadas.

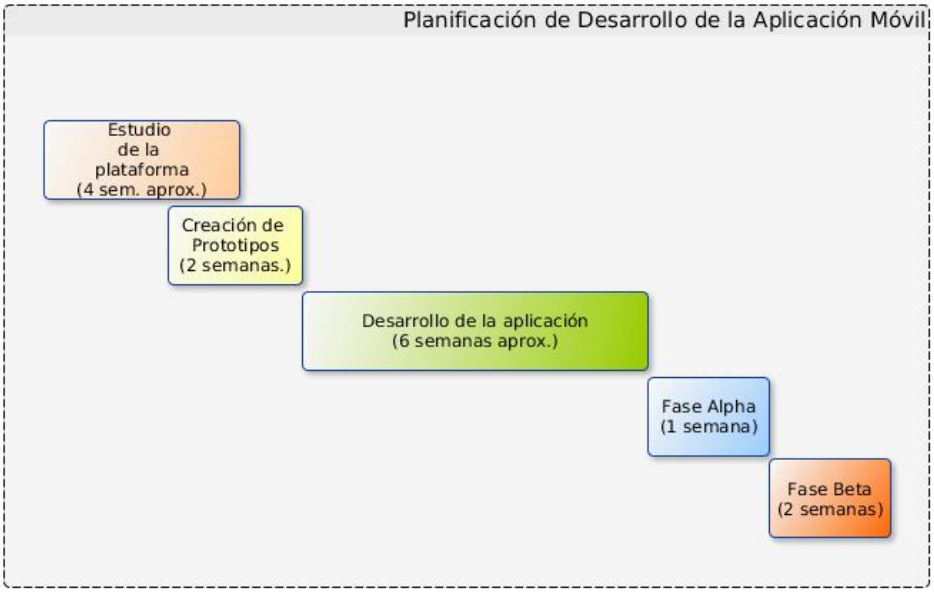

Ilustración 37. Planificación temporal del desarrollo por aplicación

6.8. Conector Propio

Una de las opciones que podría considerarse tras la puesta en marcha del sistema, es la sustitución del WebService, como capa de abstracción del resto de componentes, por un Servidor propio que emule el funcionamiento de este.

Este remplazo no proporcionará nuevas funcionalidades ni mejores controles, si no que mejorará el rendimiento de la plataforma. Es por ello, por lo que se deberá analizar si es necesaria o no la incorporación de este nuevo elemento, ya que es muy posible que con la utilización del WebService se consiga los niveles de rendimiento y de disponibilidad requeridos.

La mejora del rendimiento se consigue debido a la eliminación de todas las capas intermedias que conlleva la utilización del protocolo HTTP/S y del propio servidor Apache (Web Server). La utilización de este nuevo Servidor atenderá directamente de la conexión TCP/IP aplicando las capas exclusivas para la tramitación del protocolo establecido.

Este componente estaría formado por 2 entidades: un elemento servidor, que eliminaría al WebService y Apache, y un conector alojado en cada subsistema que permitirá la conexión con él. El servidor atenderá las peticiones entrantes, llevando a cabo toda la funcionalidad de control y de servicios que prestaba el WebService. La implementación de dicho servidor podrá estar diseñada con cualquier lenguaje actual que soporte el uso de socket y TLS.

La utilización de este conector propio además de mejorar el rendimiento del canal de comunicación del sistema, eliminando todas las capas intermedias existentes, permitirá abandonar los puertos asignados para los protocolos HTTP/S, dejando dichos puertos exclusivamente para la utilización Web.

Con ello conseguimos controlar aún más los servicios que ofrece el sistema, pudiendo incluso dividir el nivel 3 de nuestra plataforma de Alta Disponibilidad en 2 secciones según el servicio prestado: una para el uso de nuestro nuevo servidor y otra para la utilización del portal Web. En el diagrama Ilustración 38 se presenta el resultado final de dicha capa.

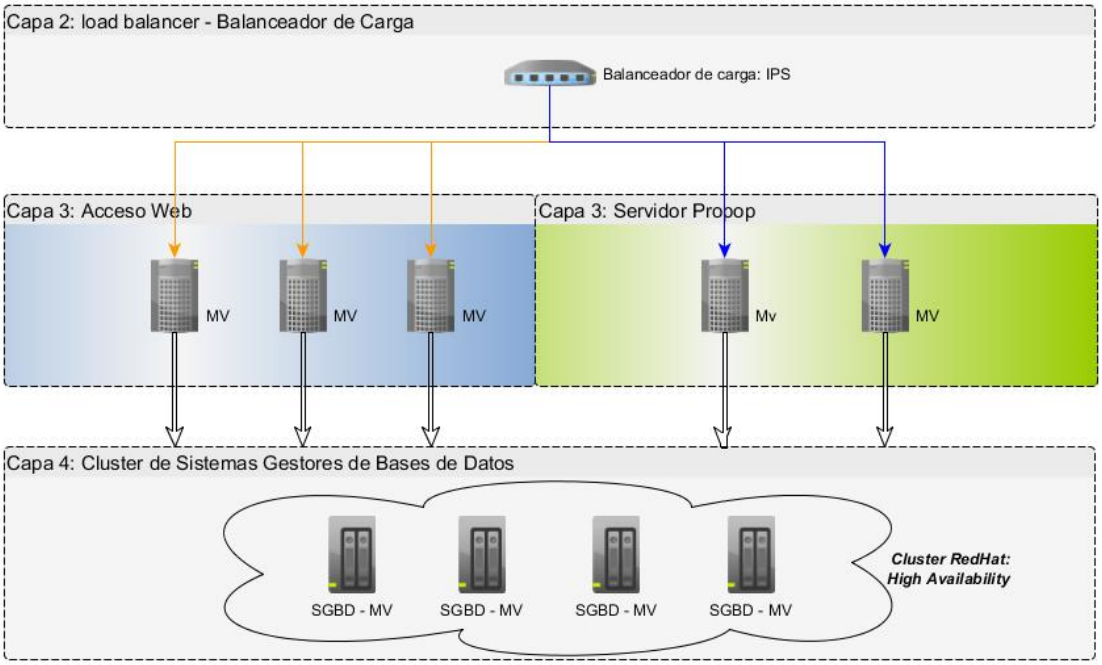

Ilustración 38. Inclusión del Servicio personalizado al sistema de Alta disponibilidad

7. Conclusiones

Se ha planteado la realización de un proyecto con el objetivo de estudiar el desarrollo de una herramienta distribuida y multiplataforma destinada a solucionar el problema de gestión de la actividad de los profesionales del sector jurídico. Como resultado se ha realizado un análisis y estructuración completas de las características que debería de reunir dicha herramienta y, además, se han implementado el 90% de las mismas, dentro de los límites temporales y de recursos del proyecto.

Se han obtenido interesantes conclusiones durante la realización del proyecto. Si analizamos las características del sistema postulado, podemos apreciar la capacidad que proporciona utilizar sistemas distribuidos o en nube, permitiéndole al usuario una diversidad de opciones impensables hace poco tiempo.

Además este hecho se intensifica con el auge que han ido adquiriendo los dispositivos móviles en los últimos años, tendencia que se espera continuar con la aparición de nuevos conceptos tecnológicos como son las SmartGlass y SmartWatch.

Debido a esta evolución en la conducta del usuario se hace más necesaria, aún si cabe, la aparición de proyectos y sistemas de Software Libre que faciliten y optimicen el aprovechamiento de estos nuevos nichos de mercado.

Para ello es de vital importancia hacer una reflexión sobre los puntos fuertes y débiles que presenta la estructuración actual de la comunidad Libre, y elaborar un adecuamiento de la misma, con vistas a construir una organización que pueda permanecer en el tiempo, adaptándose a los cambios introducidos en este sector tan dinámico.

La importancia que puede alcanzar el Software Libre, y por lo tanto la informática, la encontramos en la solución que se plantea en este proyecto. Se puede apreciar claramente la ayuda y reducción de trabajo que introduciría el sistema, produciendo una mejora de la actividad de los operadores jurídicos y por lo tanto una mayor calidad de los servicios prestados.

8. Referencias Consultadas.

1. http://www.phpframeworks.com/
2. http://www.developerphil.com/parcelable-vs-serializable/
3. http://www.tutorialandroid.com/basico/como-crear-pestanas-tabs-en-independientes-activities/
4. http://www.vicente-navarro.com/blog/2009/02/22/crear-los-certificados-ssl-para-nuestro-servidor-web-https-con-apache-openssl-y-debian-lenny/
5. http://en.wikipedia.org/wiki/JSON-WSP
6. http://en.wikipedia.org/wiki/Web_service
7. http://dev.mysql.com/doc/refman/5.0/en/ssl-connections.html
8. http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster.html
9. http://dev.mysql.com/doc/refman/5.5/en/c-api-prepared-call-statements.html
10. http://docs.wxwidgets.org/2.8/wx_appinifunctions.html#wxgetapp
11. http://searchdatacenter.techtarget.com/es/cronica/Copia-de-seguridad-completa-incremental-o-diferencial-como-elegir-el-tipo-adecuado
12. http://dev.mysql.com/doc/refman/5.0/es/binary-log.html
13. http://dev.mysql.com/doc/refman/5.0/es/backup.html
14. http://chuwiki.chuidiang.org/index.php?title=Lectura_http_y_https_desde_java
15. http://miguelangellv.wordpress.com/2011/09/14/creacion-de-servicios-con-comunicacion-bidireccional-con-un-activity/
16. http://nosoandroid.blogspot.com.es/2013/03/como-usar-php-para-conectar-android.html
17. http://capdroid.wordpress.com/2012/07/10/configuring-and-accessing-mysql-jdbc-driver-on-android-application/
18. http://stackoverflow.com/questions/4065379/how-to-create-a-bks-bouncycastle-format-java-keystore-that-contains-a-client-c
19. http://eclipsesource.com/blogs/2013/04/18/minimal-json-parser-for-java/
20. https://code.google.com/p/json-simple/
21. http://wiki.eclipse.org/RAP/Protocol#Message
22. El gran libro de Android. Autor: Jesús tomás Gironés. Editorial marcombo.
23. Cross-platform GUI programming with wxWidgets_book. Autor: Julian Smart and Kevin Hock. Editorial: Prentice Hall