ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

GESTIÓN DE INFORMACIÓN ADAPTABLE MEDIANTE DISPOSITIVOS MÓVILES Y PÁGINAS WEB

Sandra López Navarro
Jon Legarrea Oteiza
Pamplona, 30 de enero de 2014

Índice:

1- Introducción 4
- 1.1 – Introducción 4
- 1.2 – Motivación 6
- 1.3 – Objetivos del PFC 8
- 1.4 – Organización de la memoria 9

2- Estado del arte 10
- 2.1 – Estado de las tecnologías móviles 10
  - 2.1.1. Tipos de desarrollo de aplicaciones móviles 10
  - 2.1.2. Solución adoptada en este proyecto sobre el tipo de desarrollo 12
  - 2.1.3. Frameworks para desarrollo de aplicaciones híbridas 12
  - 2.1.4. Solución adoptada sobre el framework para desarrollo de aplicaciones híbridas 14
- 2.2 – Tecnologías web 15
- 2.3 - Bases de datos 16

3- Modelo 18
- 3.1 – Introducción 18
- 3.2 - Descripción del modelo 18
- 3.3 - Entrada y salida de datos 21
- 3.4 - Representación de la información 21
- 3.5 - Resumen del modelo 22

4- Requisitos 24
- 4.1 – Introducción 24
- 4.2 - Requisitos generales 24
- 4.3 - Requisitos hardware 24
- 4.4 - Requisitos funcionales 25
- 4.5 - Requisitos no funcionales 25

5- Diseño 27
- 5.1- Introducción 27
- 5.2- Diagramas de flujo de datos 27
  - 5.2.1. Diagrama de contexto 28
  - 5.2.2. Diagrama de nivel 1 30
  - 5.2.3. Diagrama de flujo de datos 2: Crear formularios 32
  - 5.2.4. Diagrama de flujo de datos 2: Listar formularios 33
  - 5.2.5. Diagrama de flujo de datos 2: Listar registros 35
  - 5.2.6. Diagrama de flujo de datos 2: Comunicación con el servidor. 37
  - 5.2.7. Diagrama de flujo de datos 3: Almacenar registro 39
- 5.3- Diagrama de estructura 40
- 5.4- Base de datos 42

6- Implantación 43
6.1- Introducción 43
6.2- Necesidades para la implantación en el cliente 43
6.3- Necesidades para la implantación en el servidor 44

7- Pruebas 45
7.1- Introducción 45
7.2- Proceso de pruebas 45
7.3- Resultados según el feedback de los usuarios 49
7.4- Resultados según la funcionalidad 50

8- Conclusiones 55

9- Líneas futuras 59
9.1- Introducción 59
9.2- En la parte servidor 59
9.3- En la parte cliente 60

10- Anexos 62
10.1- Instalación de Phonegap 62
10.2- Instalación de Firefox y complemento Firebug 62
10.3- Creación de un dispositivo Android emulado 63
10.4- Actualización de la versión de Phonegap 63
10.5- Instalación de servidor Apache 64
10.6- Instalación de PHP 65
10.7- Instalación de phpmyadmin 67
10.8- Creación de repositorios en GitHub 69

1- Introducción:

1.1 Introducción:

En la actualidad existe un uso cada vez más extendido de los dispositivos móviles. Estos dispositivos ya no se limitan a las capacidades específicas de la telefonía, sino que permiten acceder a todos los recursos de Internet y concretamente de la web ampliando enormemente sus capacidades. Compartir la información es una acción que puede realizarse desde cualquier lugar si se dispone de conexión de datos en el dispositivo. El empleo masivo de los dispositivos móviles supone un cambio de paradigma en la inmediatez de la información así como en la independencia del lugar donde se recogen los datos.

Previamente a la gestión de la información nos encontramos con la problemática de la recogida de datos. Es un hecho que a través de los dispositivos móviles se pueden tomar datos en cualquier lugar, sin embargo no siempre podemos predecir qué tipo de datos vamos a necesitar recoger ni la cantidad de los mismos.

Para este proyecto fin de carrera colaboran la Universidad Pública de Navarra y Cruz Roja Navarra planteando la creación de un sistema de gestión de datos adaptable. Ello supone que el sistema desconoce a priori qué tipos de datos va a ser preciso recoger, su cantidad y su finalidad. Por otra parte, se desconoce el escenario sobre el que se va a realizar la recogida de datos, por lo que este proceso debe realizarse de forma independiente a la existencia de una conexión a Internet en el momento.

Aunque la mayor parte de dispositivos móviles poseen conexión a Internet, son múltiples los casos en los que podemos no disponer de esta conexión. Puede ser que el dispositivo no posea conexión a Internet y dependa de las conexiones a redes inalámbricas abiertas para poder acceder a la web. También en algunos lugares no existe la posibilidad de conectar a la red de datos. Es por ello que se plantea un sistema de recogida de datos que no dependa inmediatamente de la disponibilidad de una conexión a Internet en el momento.

Una característica del mercado de los dispositivos móviles es la variedad de plataformas existentes, lo que supone distintos sistemas operativos para cada dispositivo. Esto hace que en el momento en que se plantea el desarrollo de una aplicación para este tipo de tecnologías se deban tener en cuenta factores como la plataforma sobre la que debe funcionar la aplicación. Cada plataforma posee unas características propias y un lenguaje de programación nativo a la misma sobre el que están desarrolladas sus aplicaciones.

La mejor opción desde el punto de vista de aprovechamiento del dispositivo es desarrollar una aplicación en el lenguaje de la plataforma (también llamado lenguaje nativo) para la que se ha decidido programar la aplicación. Esto supone que si deseamos que la aplicación funcione sobre más de una plataforma necesitaremos realizar tantas versiones del desarrollo como plataformas sobre las que queremos que nuestra aplicación funcione, lo que supone un coste de desarrollo por cada plataforma. Además, debemos tener en cuenta que el lenguaje de programación es distinto, por lo que se requiere de un tiempo de aprendizaje y adaptación a cada lenguaje.

En este proyecto se ha trabajado de forma que en el futuro se minimice el coste de desarrollo por plataforma. La forma empleada para minimizar este coste se basa en la posibilidad de crear aplicaciones que funcionen utilizando la vista web de la plataforma del dispositivo. Todas las plataformas móviles poseen un motor de renderizado web, por lo que parece que la mejor forma de asegurarse la compatibilidad con el mayor número de plataformas posibles sería hacer aplicaciones que funcionen en la vista web. Para hacer aplicaciones de este tipo existen unos frameworks que permiten programar aplicaciones que realmente están basadas en código web (HTML5, CSS3 y

JavaScript) y que se hacen funcionar en una vista web embebida en código de la plataforma.

El hecho de que las aplicaciones funcionen directamente sobre la vista web del dispositivo hace que sean compatibles con el mayor número de plataformas posible, pero tiene otras características asociadas que pueden constituir limitaciones para la aplicación que va a ser desarrollada.

Por una parte, muchas aplicaciones diseñadas específicamente para dispositivos móviles requieren de acceso a las capacidades hardware del dispositivo. Como hemos mencionado nada más comenzar la memoria, las tecnologías de los dispositivos móviles no se limitan tan sólo a las relacionadas con la telefonía, sino que añaden cada vez más características. Podemos poner por ejemplo la cámara fotográfica, el GPS o el acelerómetro. La mejor forma de acceder a todas las características disponibles para un dispositivo móvil es a través de la programación en el lenguaje de la plataforma, pero los frameworks destinados al desarrollo de aplicaciones multiplataforma proveen de APIS destinados a facilitar el acceso a capacidades nativas.

Por otra parte, una característica ampliamente apreciada por los usuarios de aplicaciones es la rapidez de ejecución de las mismas. Normalmente los usuarios sentimos frustración en el uso de aplicaciones lentas, por lo que la rapidez es altamente valorada. La forma de crear aplicaciones que funcionen con mejor rendimiento es a través del uso del lenguaje de la plataforma, pero como veremos es posible que la aplicación no requiera de un gran consumo de recursos y que por tanto sea más provechoso su desarrollo en forma de aplicación multiplataforma.

Como vemos, el desarrollo de aplicaciones móviles es un proceso que consta de varias características a tener en cuenta antes de decidir factores como la tecnología a utilizar, por lo que se hace necesario un estudio de lo que se desea conseguir de modo que las decisiones tomadas contribuyan a un menor coste y a obtener la mejor solución para el usuario.

1.2 -Motivación:

En el entorno anteriormente descrito se plantea un sistema de gestión de información utilizando dispositivos móviles. En primer lugar hemos de entender cómo se realiza la recogida de los datos. Este proceso se realiza a través de una variedad de herramientas que ya en su empleo establecen una organización de la información y la definición del tipo de la información que se va a recoger. Podemos pensar por ejemplo en: datos de usuarios en un registro, informes, encuestas, cuestionarios... todas ellas son herramientas utilizadas en la recogida de datos.

Por ejemplo, estos sistemas se utilizan frecuentemente en ciencias sociales como forma de estudiar características de un grupo humano. Sin embargo, la recogida de información según estas técnicas se puede emplear en todo tipo de ámbitos. Un formulario para resolución de un problema técnico, un formulario destinado a facilitar el diagnóstico de un caso clínico son otros ejemplos de uso de técnicas de recogida de información.

Todos estos ejemplos mencionados tienen en común una característica: la creación de estas herramientas se basa en la generación de un formulario. Los formularios se componen de varios campos de distintos tipos: campos de texto, campos numéricos, campos de texto largo, listas en las que es posible escoger una única opción, listas de selección múltiple... Podría decirse que un formulario se define como una plantilla que indica qué datos se van a recoger y qué tipo tienen esos datos.

El uso de una aplicación que genere herramientas para recoger información utilizando los campos individuales de que se compone hace que no sea necesario conocer en un principio la finalidad de la herramienta que se va a crear ni la cantidad de datos que se van a recoger. Sería posible crear una herramienta destinada a la obtención de estadísticas y a la vez, usando el mismo programa, crear otra herramienta destinada al diagnóstico médico.

Es por ello que hablamos de Gestión de información adaptable: es el usuario el que decide qué información desea recoger y la forma que tendrá su herramienta de recopilación de información.

Otra característica importante en el proyecto es la capacidad de funcionamiento sin conexión para sin embargo poder acceder a otras características cuando sea posible conectar a Internet. Muchas veces la información no está destinada al uso por parte de la persona que la ha recogido, sino que esta información debe llegar a otra persona que será la encargada de realizar una acción en consecuencia. En este proyecto se envía la definición de formularios o plantillas y la información recogida según estas plantillas a una base de datos alojada en un servidor, al cual se puede acceder de forma global permitiendo la explotación de los datos allí donde sea necesaria.

Podemos imaginar el servidor como una central a la que se conectan los dispositivos para centralizar la información. Esto hace que la información sea fácil de compartir y por tanto hace posible que llegue a otras personas que podrían ser las responsables de una actuación posterior a la recogida de los datos. Estas características van a dotar al proyecto de dos problemas a resolver: una forma de almacenar los datos en el dispositivo cuando se encuentra en un escenario de desconexión de una red de datos y una forma de almacenarlos en una base de datos externa.

Finalmente, se pretende que la aplicación sea compatible con el mayor número de plataformas posible. Por ello se opta por hacer que la base de la aplicación sea HTML5, CSS y JavaScript. La aplicación se va a ejecutar sobre la vista web del dispositivo. Tal y como va a ser desarrollada, su funcionamiento se probará en sistemas Android, sin embargo sería posible adaptarla en un período corto de tiempo para su funcionamiento en otras plataformas sin tener que invertir los recursos que serían necesarios si se pretendiera desarrollar la aplicación para cada una de las plataformas.

Existe un precedente de gran interés en aplicaciones de este tipo, capaz de generar

formularios para aplicaciones diversas, de recoger datos según esos formularios y de enviarlos, todo ello adaptado al funcionamiento en dispositivos móviles. Esta aplicación se llama Open Data Kit, herramienta gratuita y open source destinada a la recolección de datos.

Dado el interés que posee esta aplicación, se pretende hacer que una parte del proyecto sea compatible con ella. Open Data Kit es capaz de crear formularios en formato XML según el estándar de Java Rosa. Estos formularios son denominados Xforms y son definidos como una forma de crear formularios web.

La aplicación resultado del proyecto es capaz de crear también formularios en forma de Xform y de leer formularios sencillos en forma de Xform creados por la aplicación Open Data Kit.

La finalidad del proyecto es obtener una aplicación más sencilla que la existente para facilitar la generación de herramientas de recogida de la información. Se pretende que la aplicación sea sencilla de manejar por usuarios con niveles de conocimientos tecnológicos heterogéneos. La aplicación Open Data Kit es muy completa pero es de difícil configuración si además se tiene en cuenta que el usuario puede no estar familiarizado con el ámbito de la informática. Con este proyecto se pretende obtener una aplicación más sencilla de manejar incluso por usuarios no expertos.

1.3 - Objetivos del PFC:

La aplicación tiene por objetivos los siguientes:

- Obtener un sistema para su uso en dispositivos móviles con independencia de la plataforma, con lo que se obtiene una independencia con respecto al lugar de uso.
- Generar formularios de forma dinámica, posibilitando que el usuario de la aplicación genere un formulario adaptado a cada situación para recoger la información relevante en la misma.
- Generar registros para los formularios para que el usuario de la aplicación pueda realizar la recogida de datos adaptados a sus necesidades.
- Almacenar de forma local tanto los formularios como los registros, posibilitando de este modo el uso de la aplicación en ámbitos sin conexión a Internet. Ello hace más pronunciada la independencia de la aplicación con respecto a su lugar de uso, ya que no es necesario para el usuario disponer de conexión en el momento de recabar los datos.
- Posibilitar el posterior envío tanto de definiciones de formularios como de los registros creados a una base de datos alojada en un servidor, con el objetivo de facilitar compartir los datos y su almacenamiento externo.
- Posibilitar la obtención de definiciones de formularios desde el servidor, de modo que si existe en el mismo una definición de formulario adaptada a las necesidades del usuario, pueda ser utilizada por éste evitando que se deba crear un formulario desde cero, con lo que se permite la reutilización de definiciones.
- Asimismo, posibilitar la obtención de definiciones de registros desde el servidor para su posterior edición o revisión.
- Posibilitar la exportación de definiciones de formularios a un formato compatible con la aplicación Open Data Kit según el estándar de Java Rosa.

1.4 - Organización de la memoria:

El presente documento consta de los siguientes apartados:

1- Introducción: Tras una breve introducción se hace una presentación del proyecto en la que se destacan sus características principales y se explica la motivación que lleva al desarrollo de la aplicación que se presenta.

2- Estado del arte: Se realiza una explicación de las tecnologías que se van a aplicar en el desarrollo de la aplicación, haciendo una breve comparativa entre aplicaciones que poseen finalidades parecidas. Después se establecen los requisitos concretos de la aplicación que finalmente se convertirán en las características de la misma.

3- Modelo: Se presenta una explicación del modelado del sistema observándolo en su forma más abstracta.

4- Requisitos: Se realiza una enumeración de los requisitos que determinan las condiciones que debe satisfacer la aplicación.

5- Diseño: Se presentan los esquemas que definen la arquitectura del sistema (diagramas de flujo de datos, diagrama de estructura, diagrama entidad-relación de la base de datos en el servidor). Así se puede comprender más fácilmente el camino que siguen los datos desde su entrada a su salida, pudiendo ver la transformación de los mismos. También se hacen más comprensibles las relaciones entre los procesos que finalmente forman la aplicación.

6- Implantación: Necesidades para poder realizar la implantación del sistema.

7- Pruebas: Proceso de pruebas de la aplicación con sus resultados.

8- Conclusiones: Explicación de los objetivos cubiertos durante el proyecto, mostrando imágenes de la aplicación ya construida.

9- Líneas futuras: Mejoras que se podrían realizar para futuras actualizaciones de la aplicación y que contribuirían a la obtención de una aplicación más completa y con más funcionalidades.

10- Anexos: Proceso de instalación de los programas utilizados para el desarrollo de la aplicación.

2- Estado del arte:

2.1- Estado de las tecnologías móviles

Tal y como se ha explicado en la introducción, los dispositivos móviles poseen un amplio abanico de posibilidades diferentes del uso convencional telefónico. Poseen capacidades de procesamiento de datos y posibilidad de conexión a redes. Por estas características, el mercado de aplicaciones móviles es muy amplio y sigue creciendo a una gran velocidad.

La plataforma para la que se va a desarrollar es un factor de importancia. Ello condiciona el lenguaje de programación que se va a utilizar. La plataforma es el sistema operativo sobre el que se va a ejecutar la aplicación.

Android es la plataforma móvil más utilizada seguida de la plataforma iOS. Sin embargo estas no son las únicas plataformas existentes: aunque en un porcentaje mucho menor también se encuentran Windows Phone y Blackberry así como otras plataformas.

Actualmente los dos mayores mercados de venta de aplicaciones son Play Store de Google y App Store de Apple. En estos mercados se ponen a la venta las aplicaciones creadas por los desarrolladores, pudiendo descargarlas los usuarios a sus terminales.

Las aplicaciones de Android se venden en el Play Store, mientras que las aplicaciones iOS se venden en el App Store. Además, cada mercado de aplicaciones tiene unas condiciones distintas que se deben cumplir en el desarrollo de la aplicación. Por ejemplo, para poder hacer que una aplicación desarrollada para la plataforma iOS aparezca en el mercado App Store se deben cumplir unos requisitos de interfaz (esta debe generar una experiencia de usuario nativa a la plataforma).

Por otro lado, en la actualidad existen frameworks que proveen la capacidad de crear aplicaciones compatibles con más de una plataforma. Estas aplicaciones se basan en las nuevas características implementadas por HTML5, CSS3 y JavaScript. Las aplicaciones generadas con estos frameworks se ejecutan en un programa nativo que presenta una vista web embebida y además implementan ciertas funcionalidades que dan acceso a capacidades nativas del dispositivo. Es decir, no son simplemente páginas web con aspecto de aplicación móvil, sino que se puede hacer uso de algunas capacidades hardware como la cámara fotográfica, así como acceder a algunos componentes como la libreta de contactos del dispositivo.

Esto da una mayor flexibilidad al desarrollo de aplicaciones móviles. Cada modalidad de desarrollo posee sus ventajas e inconvenientes, por lo que se hace necesario un estudio de las características que deseamos que tenga la aplicación para decidirnos por un método u otro de desarrollo.

2.2.1- Tipos de desarrollo de aplicaciones móviles.

Existen 3 tipos de aplicaciones móviles, según su proceso de desarrollo.

Aplicaciones nativas: Estas aplicaciones son escritas íntegramente en el código nativo de la plataforma sobre la que se va a ejecutar la aplicación. Son las más rápidas de los 3 tipos que vamos a ver. Asimismo, las aplicaciones nativas no tienen limitaciones en el acceso a las capacidades y hardware del dispositivo. Sin embargo, hemos de pensar que una aplicación desarrollada para una plataforma es incompatible en las demás. Es por ello que si deseamos que nuestra aplicación esté disponible para más de una plataforma necesitaremos realizar tantas versiones de la aplicación como plataformas para las que deseamos que esté disponible. Esto supone unos mayores costes en tiempo de desarrollo. A estos costes hay que sumarles el tiempo necesario para aprender cada

lenguaje de programación nativo a la plataforma y el periodo de tiempo necesario para adaptarse a la misma.

Aplicaciones web: Son aplicaciones a las que se accede desde el navegador del dispositivo. Realmente son páginas web optimizadas para ser visualizadas desde dispositivos móviles. Las tecnologías utilizadas son las mismas que las del desarrollo web típico, es decir, HTML5, CSS3 y JavaScript. Una gran ventaja es la reutilización del código, por lo que no se hace necesario más que un único desarrollo dando compatibilidad a muchas plataformas de una sola vez. Además es más sencillo y rápido de crear, ya que es fácil que un programador posea conocimientos previos de tecnologías web. Sin embargo, en muchos casos no es posible utilizar la aplicación en modo sin conexión, lo cual puede ser una desventaja muy grande. Estas aplicaciones tampoco pueden acceder a gran número de funcionalidades nativas del teléfono. Por último, estas aplicaciones no pueden distribuirse desde los mercados de aplicaciones, ya que realmente son páginas web y no se instalan en el dispositivo.

Aplicaciones híbridas: Son aplicaciones que están construidas con una parte de código nativo y una parte de código web. Debido a la parte que tienen de código nativo pueden acceder a un buen número de funcionalidades nativas del dispositivo. El código web se ejecuta en una vista web que está embebida en la aplicación. Esto hace que las aplicaciones híbridas sean multiplataforma. Como prácticamente todas las plataformas poseen un motor de renderizado web embebible, estas aplicaciones pueden ejecutarse en todas las plataformas. Tan sólo habría que adaptar las funciones asociadas al acceso a capacidades nativas del dispositivo, con lo que se reducen ampliamente los costes de tiempo que estarían asociados a un desarrollo diferenciado para cada plataforma como sucede en las aplicaciones nativas. Las aplicaciones híbridas pueden encontrarse en los mercados de aplicaciones móviles con lo que su distribución es más sencilla que en el caso de las aplicaciones web. Además pueden funcionar sin conexión a Internet, ya que se instalan en el dispositivo. Por otro lado, mencionar que poseen una limitación en su velocidad de ejecución asociada a la velocidad del motor de renderizado web sobre el que se está ejecutando. Esto hace que las aplicaciones híbridas no puedan compararse en rendimiento con las nativas. La experiencia de usuario puede estar bastante alejada de la lograda con una aplicación nativa. Para generar una experiencia de usuario cercana a la nativa es necesario el uso de frameworks y librerías orientadas al desarrollo móvil.

Se procede a realizar una pequeña comparativa entre las distintas formas de desarrollar aplicaciones en atención a su desempeño según distintas capacidades:

- Acceso a las capacidades nativas del dispositivo: En este caso, aunque algunas aplicaciones web pueden acceder a algunas capacidades nativas, las aplicaciones puramente nativas y las híbridas son las más indicadas si en la aplicación es necesario el uso de varias características nativas del dispositivo.
- Velocidad: Las aplicaciones más veloces son las desarrolladas enteramente en código nativo.
- Mantenimiento: El mantenimiento de una aplicación en código nativo puede ser complicado si además se disponen de varias versiones de la aplicación para varias plataformas. Sin embargo, el mantenimiento de una aplicación web o de una aplicación híbrida es mucho más rápido y sencillo.
- Dependencia de la plataforma: Si es importante que la aplicación sea independiente de la plataforma, lo más útil es hacer una aplicación híbrida o una aplicación web ya que gran parte del código puede reutilizarse, no ocurriendo así con las aplicaciones nativas.

- Costes: Dadas algunas de las características de las aplicaciones web e híbridas que hemos descrito, como la posibilidad de reutilizar gran parte del código y la sencillez de las tecnologías utilizadas, estas aplicaciones son mucho menos costosas que las aplicaciones nativas.
- Interfaz de usuario: Si la prioridad es crear una interfaz de usuario consistente con el sistema operativo del usuario, de forma que la aplicación tenga unos gráficos similares a los que el usuario puede estar acostumbrado al utilizar una plataforma, lo más indicado entonces es el desarrollo de una aplicación nativa.

2.2.2- Solución adoptada en este proyecto sobre el tipo de desarrollo.

Como puede comprobarse, cada método posee su propio conjunto de ventajas e inconvenientes. En este proyecto, hemos decidido optar por una aplicación híbrida. Los motivos para tomar esta decisión son los siguientes:

- No hay un gran acceso a capacidades nativas del dispositivo: Esto hace que no sea necesario implementar la aplicación en código nativo, ya que no hay acceso a capacidades especiales.
- Es necesario que la aplicación funcione sin conexión: La aplicación posee algunas funcionalidades que requieren que pueda usarse en un momento en que no se disponga de conexión a Internet. Ello invalida el uso de una aplicación web pura, ya que se debe poder acceder a la aplicación independientemente de la conexión a red de datos o inalámbrica.
- Velocidad: No hay un gran uso de gráficos o de interfaz que haga que el rendimiento se vea notablemente afectado por la ejecución de la aplicación en la vista web de la plataforma.
- Plataforma: Se desea que la aplicación se pueda ejecutar en el mayor número de plataformas posibles con el menor coste de tiempo en su adaptación de una plataforma a otra.

2.2.3- Frameworks para desarrollo de aplicaciones híbridas.

Una vez nos hemos decidido por el desarrollo de una aplicación híbrida, la siguiente decisión a tomar es ¿Qué framework se va a utilizar para crear la aplicación?

Existen varios frameworks en el mercado para crear aplicaciones híbridas. Los más conocidos son los siguientes:
1- Titanium: es un framework para la creación de aplicaciones nativas para dispositivos móviles que utiliza íntegramente JavaScript para el desarrollo.
2- Sencha Touch: permite la creación de aplicaciones como si fueran nativas para Android o iOS utilizando HTML5, CSS3 y JavaScript.
3- Phonegap: permite la construcción de aplicaciones para móviles usando HTML5, CSS3 y JavaScript posibilitando el aprovechamiento de algunas características hardware de los teléfonos.
4- Rhodes: sirve para la creación rápida de aplicaciones nativas posibilitando el aprovechamiento de algunas características hardware de los dispositivos móviles..
5- Xui: permite la construcción de aplicaciones web simples. Es útil para crear sistemas poco complejos.

Los frameworks más utilizados son dos: Titanium y Phonegap. Se muestran a continuación las características de uno y otro.

- Titanium:

Las aplicaciones con Titanium son escritas íntegramente en JavaScript. Este código JavaScript es traducido a código nativo en el momento de la ejecución. No se hace uso de HTML ni de CSS ya que no se está utilizando una vista web, por lo que el conocimiento previo de JavaScript debe ser sólido. Ello hace que sea más complicado que los programadores provenientes del desarrollo web exporten todos sus conocimientos para el desarrollo de aplicaciones móviles utilizando esta plataforma.

Con este framework se obtiene una interfaz nativa, ya que no hay interfaz web en la aplicación, sino que la API de Titanium es la que se encarga de las necesidades de interfaz. El hecho de que se programe la aplicación enteramente en JavaScript supone que no se está utilizando HTML, por lo que no existe DOM. Ello hace que definir los componentes visuales sea más complicado. Además no se puede hacer uso de jQuery ni de librerías que sirvan para manipular el DOM.

Las aplicaciones obtenidas con este framework tienen un rendimiento bastante bueno, y es mejor si se compara con Phonegap por ejemplo, ya que las llamadas de JavaScript a la API de Titanium se convierten en código nativo. Sin embargo, y aunque a comparación de las aplicaciones obtenidas con Phonegap las escritas con Titanium poseen un rendimiento mejor, esta rapidez sigue sin ser igual a la obtenida a través del uso de código nativo.

Este framework también genera aplicaciones multiplataforma, pero está limitado a iOS y Android.

- Phonegap:

Las aplicaciones construidas con Phonegap se basan en código escrito en HTML, CSS y JavaScript. Se incluyen varias funciones JavaScript que permiten acceder a componentes hardware del dispositivo. Al final, la aplicación se ejecuta en una vista web del dispositivo, por lo que puede decirse que lo que se programa es una aplicación web con acceso a algunas capacidades nativas.

Las ventajas del uso de este framework residen en su comodidad para aquellas personas que dispongan de conocimientos web previos, lo cual dota de una mayor rapidez en el desarrollo de la aplicación sin hacer necesario el aprendizaje de un lenguaje distinto.

Además, las aplicaciones generadas con Phonegap son aplicaciones multiplataforma. Tan sólo es necesario adaptar entre distintas plataformas aquellas funciones que dan acceso a las características nativas del dispositivo, ya que el resto de la aplicación se basa en HTML, CSS y JavaScript. Al ser aplicaciones que se ejecutan en una vista web, Phonegap genera aplicaciones compatibles con más plataformas.

Como la aplicación que se crea se ejecuta en una vista web se percibe un descenso del rendimiento notable a comparación de las aplicaciones escritas enteramente en código nativo. Este rendimiento también es menor que el obtenido con Titanium, ya que recordemos que Titanium genera la interfaz y la funcionalidad en código nativo (aunque esto último tampoco es igual en rendimiento que el desarrollo puramente nativo). Hay que tener en cuenta que lo que en realidad se ha desarrollado es una aplicación que se está ejecutando utilizando el motor de renderizado web, lo cual supone una limitación en la rapidez alcanzada.

Como la aplicación es una página web embebida en el código de la plataforma, ello condiciona notablemente el aspecto de la interfaz de usuario. Es necesario el uso de varios frameworks HTML móviles si se desea que la aplicación tenga un aspecto similar al de una aplicación nativa.

2.2.4- Solución adoptada sobre el framework para desarrollo de aplicaciones híbridas.

En nuestro caso, nos hemos decidido por el uso de Phonegap frente al de Titanium u otros frameworks de desarrollo de aplicaciones híbridas. Los motivos son los siguientes:

- Uso de conocimientos de desarrollo web: Una gran ventaja de Phonegap es la posibilidad de aprovechar los conocimientos previos en materia de desarrollo web. Al estar más extendidos los conocimientos de HTML y y JavaScript, parece una mejor opción optar por apoyar el desarrollo basado en el conocimiento de HTML y CSS.
- Rendimiento: La aplicación a desarrollar no hace un gran consumo de recursos ni de acceso a capacidades nativas del dispositivo, por lo que la ventaja en rendimiento obtenida a través del uso de Titanium no nos parece relevante en este caso.
- Aplicación multiplataforma: Con Titanium las aplicaciones funcionan sobre iOS y Android, sin embargo con Phonegap el número de plataformas compatibles es mayor. Aunque el resto de plataformas sean de uso bastante minoritario, la compatibilidad es un gran punto a favor del desarrollo de la aplicación con Phonegap.

2.2- Tecnologías web:

HTML es el lenguaje fundamental de la web. Está formado por un conjunto de etiquetas (HTML es un lenguaje de marcado). Es de especial importancia resaltar la aparición de HTML5 que especifica nuevas funcionalidades a las ya conocidas del lenguaje aportando grandes avances aplicables al desarrollo de sitios web.

Los avances que más nos interesan por las características del proyecto son:

- Nuevo sistema de almacenamiento local: anteriormente, el almacenamiento local de HTML era gestionado por las cookies. Las cookies son pequeños ficheros con porciones de información. La capacidad de almacenar datos en cookies estaba muy limitada por el tamaño. Sin embargo, HTML5 ofrece un nuevo sistema de almacenamiento local basado en los pares "clave"-"valor". La información guardada en estos pares clave-valor es persistente aunque se cierre la aplicación o se reinicie. El sistema de almacenamiento clave-valor aumenta la funcionalidad de las páginas web. Esta característica de HTML5 es la que en este proyecto hace posible la funcionalidad sin conexión.
- Se añaden nuevos tipos de datos para su uso en formularios.

HTML5 está todavía en desarrollo, por lo que antes de utilizar sus funcionalidades hay que tener en cuenta que pueden no estar soportadas por todos los navegadores.

HTML define el contenido de una página web. A este contenido se le pueden aplicar unos estilos que definen la apariencia de los elementos HTML que se visualizan en el navegador. Para definir los estilos y por tanto la apariencia de un sitio web, se utiliza el lenguaje CSS. La última versión de este lenguaje de estilos se llama CSS3.

Se pueden ejecutar funciones en el lado del cliente para ofrecer páginas web dinámicas. El lenguaje de programación más utilizado para esto es JavaScript. Este lenguaje ofrece la posibilidad de mejorar la interfaz de usuario y de generar contenido dinámico en el sitio web.

HTML, CSS y JavaScript se pueden utilizar conjuntamente en el mismo fichero. Es decir, se puede crear un único fichero HTML que contenga además CSS y JavaScript embebido. Sin embargo, lo ideal suele ser separar contenido de apariencia (HTML de CSS) y contenido de funcionalidad (HTML de JavaScript) por lo que suele ser recomendable emplear contenido, apariencia y funciones en ficheros separados. Esto hace que el código sea más fácil de reutilizar y hace más sencilla la mantenibilidad de la página. Por ejemplo, si se desea modificar un aspecto del diseño de todas las páginas web, acudiríamos simplemente al fichero donde está definida la apariencia (CSS) y haríamos las modificaciones necesarias, evitando tener que modificar todas las páginas que componen el sitio web por separado.

Como ya se ha mencionado anteriormente, las tecnologías web (HTML5, CSS, JavaScript) se han empleado en este proyecto para realizar una aplicación Android mediante el framework Phonegap.

Queda destacar en este apartado de tecnologías web el lenguaje de programación PHP. Este lenguaje suele ser empleado para generar contenido web dinámico, generalmente almacenado en servidores web que se visitan en Internet.

A lo largo del proyecto se ha desarrollado una interfaz móvil/servidor (en adelante API), que realiza la función de trasladar a la base de datos almacenada en el servidor la información enviada por la aplicación Android. Para realizar el API se ha empleado PHP.

2.3- Bases de datos:

En este apartado vamos a ver el estado de las tecnologías relacionadas con las bases de datos para finalmente entender cómo se aplican en el proyecto.

Las bases de datos se utilizan para el almacenamiento de la información. Existen varios tipos de bases de datos atendiendo a distintos parámetros. Una forma de clasificar las bases de datos es según el modelo de datos en el que se basan, es decir, según los métodos que siguen para el almacenamiento y la recuperación de la información.

El modelo de datos más utilizado en bases de datos es el modelo relacional. Este modelo se basa en los conceptos de relación y tablas. Las tablas almacenan registros, donde estos vienen representados por cada fila de la tabla. Los registros se componen de campos, representados por las columnas de la tabla. Las relaciones representan las conexiones entre las tablas. La información se recupera a través del uso de consultas. El lenguaje más habitual en la creación de consultas es SQL.

Aunque el modelo de datos más utilizado es el relacional, este no es el único modelo utilizado. Los modelos que difieren del modelo relacional se denominan modelos No SQL y se caracterizan porque no utilizan el esquema de relaciones y tablas que define al modelo relacional. Al no estar sometidos a la misma estructura de relaciones y tablas son sistemas más flexibles, pudiendo almacenar la información en forma de pares clave-valor o de tablas con columnas dinámicas. Se suelen utilizar cuando el volumen de datos a almacenar es muy grande. Dos tipos de bases de datos NoSQL son por ejemplo las bases de datos clave-valor o las bases de datos documentales. Las bases de datos clave-valor asocian valores a claves únicas. Las bases de datos documentales asocian a la clave un valor que es un documento, donde este documento puede ser por ejemplo un JSON.

JSON es un formato de almacenamiento e intercambio de datos. Es un formato especialmente ligero y utiliza una sintaxis sencilla de entender, por ello es especialmente utilizado en el envío de datos, ya que estos pueden enviarse como una simple cadena de texto. Esta característica hace que mediante el uso de JSON se puedan enviar grandes cantidades de información. Sin embargo, en el uso de JSON para almacenamiento de datos hemos de tener en cuenta que si en la base de datos se introduce la información en este formato, lo que realmente estamos almacenando es una cadena de texto. Es decir, no podremos realizar consultas sobre partes del JSON para obtener información de una parte de la cadena.

Las bases de datos relacionales utilizan una estructura de tablas y relaciones para almacenar los datos. Pero si necesitamos de campos dinámicos para las tablas, o no conocemos previamente el tamaño de estas tablas, puede ser más adecuado el uso de otro tipo de base de datos.

La aplicación consta de dos enfoques respecto al almacenamiento de datos a resolver: por un lado el almacenamiento local de los datos para entornos sin conexión a red de datos y por otro lado el almacenamiento externo de los datos, una vez se ha accedido a un entorno con conexión.

Para el almacenamiento local se emplean las características de almacenamiento local provistas por HTML5 en forma de pares clave-valor.

Para el almacenamiento centralizado de los datos se va a utilizar una base de datos alojada

en un servidor en Internet.

En este proyecto vamos a utilizar una base de datos relacional MySQL para almacenar las cadenas JSON que se generan con la información de definición de formularios y la información que posteriormente se almacena al crear registros utilizando como base dichos formularios. Sin embargo tenemos que tener en cuenta que posteriormente no vamos a poder realizar consultas sobre estos datos en base a unos campos. Es decir, no vamos a poder realizar consultas en base al nombre de los formularios, o en base al número de campos que definen el formulario, o a todos los registros que contengan una determinada fecha. Esto sucede porque el JSON se almacena como una cadena de texto, y esa cadena de texto se almacena en un campo de la base de datos SQL, por lo que no podremos acceder separadamente a los campos que definen al JSON.

Se ha utilizado la base de datos MySQL por comodidad y porque en este punto inicial del proyecto no era necesario probar con otro tipo de base de datos que almacene los datos de una forma diferente. No vamos a necesitar realizar consultas sobre los campos del JSON por ejemplo. Sin embargo, en mejoras futuras, podría ser interesante adaptar la aplicación a otro tipo de bases de datos NoSQL como por ejemplo MongoDB que estén diseñadas especialmente para el almacenamiento de datos de tipo JSON. Quizá de esta forma se puedan conseguir mejoras en el rendimiento de la aplicación o el almacenamiento de los datos de una forma más manejable.

Estas cuestiones quedan definidas como posibilidades para su estudio en líneas futuras, considerándolas de gran interés.

3- Modelo:

3.1- Introducción:

Una de las prioridades centrales del modelo que se busca está en la capacidad de funcionar en todo tipo de entornos independientemente de la posibilidad de acceso a red de datos. Ello va a hacer que nos enfrentemos al problema del almacenamiento en los dos escenarios posibles: uno en el que existe conexión a red de datos y otro en el que no es posible realizar conexión.

Además del problema en si mismo que supone el almacenamiento de datos según el escenario en el que nos encontremos, derivado de éste aparece el problema de cómo pasar de un escenario al otro. Es decir, tendremos que poder transferir los datos que hemos almacenado en un escenario sin conexión al almacenamiento externo accesible en entornos con conexión si de repente la conexión a red de datos es posible. Y también tendremos que poder proceder de forma inversa: debe existir algún modo de transferir los datos que hemos podido visualizar teniendo conexión al dispositivo para trabajar con ellos si de repente hemos de trasladarnos a una zona donde perderemos la conexión a red de datos.

Otra de las prioridades en este modelo es obtener la mayor independencia posible de la plataforma del dispositivo móvil sobre el que se esté ejecutando la aplicación. Esto lo obtendremos a través del uso de las herramientas adecuadas para el desarrollo de este tipo de aplicaciones. De esta forma obtendremos la mayor compatibilidad posible con todo tipo de dispositivos.

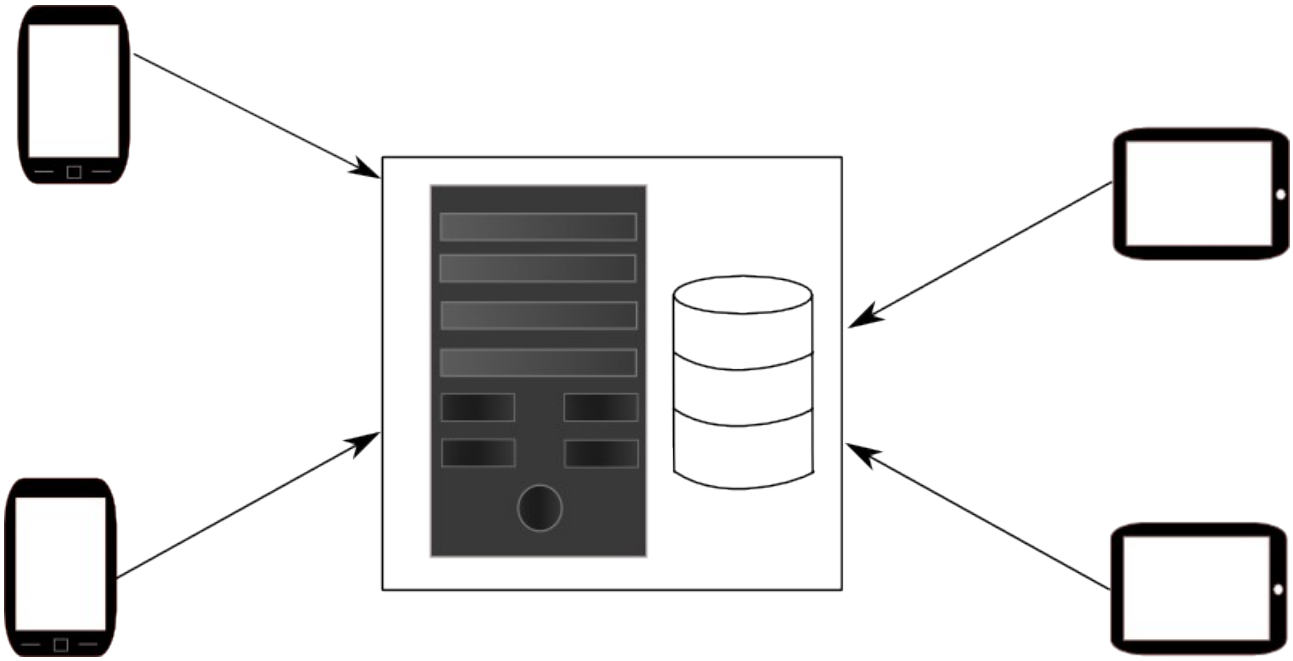
Finalmente, hemos de recordar que el objetivo final es desarrollar una aplicación suficientemente sencilla como para que pueda ser usada por un gran número de usuarios de experiencia heterogénea en el manejo de herramientas informáticas. Como comentamos en la introducción, existe una aplicación de propósito parecido al de la que presentamos, pero mucho más complicada de utilizar, sobre todo si tenemos en cuenta los distintos niveles de conocimientos informáticos de los usuarios. Sin embargo y dado el interés de dicha aplicación, se pretende la compatibilidad de ficheros generados con ella con ésta aplicación que presentamos y al revés, que formularios generados con ésta aplicación puedan almacenarse en ficheros que podrán ser utilizados por la otra aplicación existente.

Teniendo en consideración estos puntos, que constituyen la base de la problemática a resolver, vamos a presentar el modelo de la aplicación.

3.2- Descripción del modelo:

Una de las características principales del modelo es la posibilidad de generar datos y almacenarlos en forma que no requiera conexión a red y almacenarlos en una base de datos externa si acaso la conexión es posible.

Esta característica la vamos a conseguir utilizando un modelo cliente-servidor, en la que el cliente va a ser capaz de almacenar datos en el dispositivo que estarán disponibles incluso si la aplicación se reinicia o se cierra y se vuelve a abrir. El almacenamiento de los datos debe ser lo más liviano posible. Recordemos que estamos trabajando en dispositivos móviles, para los que el tamaño de los datos es una cuestión fundamental, por ello se trabajará de forma que los datos generados ocupen el menor espacio posible.



El almacenamiento de datos en el dispositivo en el que se encuentra el cliente se gestiona utilizando una característica nueva añadida en la especificación de HTML5: el almacenamiento local. Antes de que se creara esta característica HTML era capaz de guardar pequeñas porciones de datos en el lado del cliente almacenándola en el navegador. Estas porciones se denominan cookies. Las cookies son pequeños ficheros de texto donde un sitio web almacena información para facilitar por ejemplo el manejo de sesiones de usuario, haciendo más cómoda la navegación al no tener que estar pidiendo continuamente nombre de usuario y contraseña. Asimismo las cookies también pueden utilizarse para almacenar otros tipos de información relacionadas con la navegación del usuario por un sitio web, etc.

Las cookies sin embargo poseen un tamaño muy limitado. Además se envían constantemente al servidor con cada petición del cliente al servidor, esto tiene sentido por ejemplo para las cookies relacionadas con la sesión de usuario.

HTML5 incluye una función de almacenamiento local distinta de la de las cookies. El almacenamiento local ofrecido por HTML5 permite un mayor espacio de almacenamiento (de unos 5MB dependiendo del navegador). Además, el contenido del almacenamiento local no se reenvía constantemente al servidor reduciendo de este modo el tráfico a través de la red. La posibilidad de almacenar un mayor volumen de datos en el cliente ofrece unas posibilidades mayores de trabajo en entornos sin conexión, ya que podemos almacenar datos en el cliente.

El almacenamiento de datos en el almacenamiento local que ofrece HTML5 se realiza siempre en cadenas de texto. Consecuentemente hay que realizar una conversión ya que no se pueden almacenar otros tipos de datos como números o listas. Por otra parte, el almacenamiento de estas cadenas de texto está asociado a una clave. Es por esto que se dice que el formato de almacenamiento es clave-valor: a una clave única se le asocia una cadena de texto que contiene datos.

La forma que se ha utilizado para crear un formato de datos capaz de ser almacenado en el almacenamiento local ha sido crear unos JSON en un formato que nos es válido para guardar definiciones de formularios y otros JSON válidos para guardar la información que se desea almacenar una vez se han rellenado los formularios.

Como explicamos anteriormente, JSON es un formato ligero de datos en el que la información aparece estructurada. Estos JSON se tratan en JavaScript como objetos. Su uso se

asemeja al de las tablas, lo cual hace que sea sencillo de crear y de utilizar. Posteriormente, se pueden convertir los JSON a formato cadena de texto para su almacenamiento. Estas cadenas se podrán recuperar más tarde y volver a transformar en JSON para trabajar con ellas.

En resumen: En el lado del cliente vamos a utilizar el almacenamiento local que ofrece HTML5. Esta forma de almacenamiento nos servirá para guardar los datos en el cliente sin necesidad de que dispongamos de conexión a Internet y además la información no se perderá aunque la aplicación se cierre. En ese almacenamiento local guardaremos cadenas de texto asociadas a claves. Las cadenas de texto serán en realidad datos almacenados en forma de JSON que contendrán definiciones de formularios y la información de los registros.

La siguiente cuestión a resolver es el sistema de almacenamiento que se utilizará en entornos que dispongan de conexión y que puedan transmitir datos a través de la red.

Para ello pasamos a explicar cómo funciona la parte servidor de nuestro modelo.

El servidor contiene una base de datos MySql en la que se almacenarán las definiciones de formularios y los registros que el cliente haya generado. Una vez el usuario puede establecer conexión, podrá acceder a la base de datos tanto para introducir datos como para obtener datos de la misma para guardarlos en su dispositivo.

La base de datos consta de dos tablas: una para las definiciones de los formularios y otra para los registros. Los formularios se identifican por un hash md5 que habrá sido calculado previamente en el cliente. Los registros se identifican asimismo por su hash md5 pero además llevan asociados la clave md5 del formulario del que proceden, de forma que nunca quedan registros que no posean un formulario padre, sino que ambos están interrelacionados.

En nuestro caso es muy básica la función del servidor ya que tan sólo se encarga de dar acceso a los usuarios a la base de datos que contiene únicamente estas dos tablas.

Una vez hemos visto cómo se almacena la información de forma local y como se almacena de forma externa, hemos de explicar cómo se realiza el paso de la información de una a otra forma. Para facilitar la gestión de los datos que se encuentran en cliente y en servidor se ha diseñado una forma de “sincronización” que permite conocer qué datos se encuentran en servidor y en cliente a la vez o qué datos se encuentran tan sólo en uno de los lados. Así, el usuario podrá escoger para enviar al servidor tan sólo aquellos datos que no se encuentren previamente en él y viceversa: podrá descargar al dispositivo sólo los datos que no se encuentren ya en él. Esta forma de sincronización facilita al usuario el manejo de los datos porque le permite obtener una visión de qué datos se encuentran en qué lados del sistema.

Esta sincronización se obtiene haciendo un envío al servidor de una lista de los md5 de formularios y de registros que se encuentran en el dispositivo. Con esta información y tras hacer el servidor su propia lista de md5 alojados en él, se puede obtener la lista de md5 que se encuentran en las dos listas a la vez.

Aunque este método es muy sencillo, nos sirve para obtener una interfaz que orienta al usuario en su gestión de los datos en el dispositivo.

Posteriormente, si se desean enviar datos al servidor, se crean unas tablas que contienen los JSON con la información seleccionada por el usuario para enviar y se crea un objeto AJAX que realiza un envío a través de POST. Este envío es recogido y procesado por el servidor para la inserción en la base de datos.

Igualmente sucede para la recogida de datos desde el servidor. El usuario selecciona la información que desea obtener en su dispositivo. El servidor procesa esta petición y obtiene de la base de datos los formularios y registros pedidos, preparando dos tablas que contienen la información. Estas tablas son recogidas por el cliente y procesadas para terminar almacenándolas en

el dispositivo utilizando el almacenamiento local de HTML5.

3.3- Entrada y salida de datos:

Al describir las componentes más importantes del sistema ya hemos descrito la principal vía de entrada de datos al mismo: el usuario define a través de la aplicación la forma de los formularios y los datos que desea recoger. También puede obtener datos del servidor, haciendo que se puedan compartir datos entre distintos dispositivos. Sin embargo disponemos de otra forma de alimentar al sistema definida por el uso de ficheros .XML que contienen un formulario en forma de Xform.

Asimismo, la salida de datos del cliente se produce como ya hemos descrito realizando una conexión al servidor para el envío de datos almacenados en el dispositivo. Otra forma de salida de los datos es almacenando la información que define un formulario en un fichero .XML con formato de Xform.

Para poder realizar estas entradas y salidas de datos al sistema en forma de ficheros .XML tendremos que hacer uso de la capacidad de crear ficheros y almacenarlos en la tarjeta de memoria del dispositivo, así como de la capacidad de lectura de ficheros desde el dispositivo. Esta es una capacidad nativa del dispositivo y Phonegap es el que nos ofrecerá las funciones para el manejo de estas situaciones que no podríamos realizar en caso de que nuestra aplicación no fuera una aplicación híbrida.

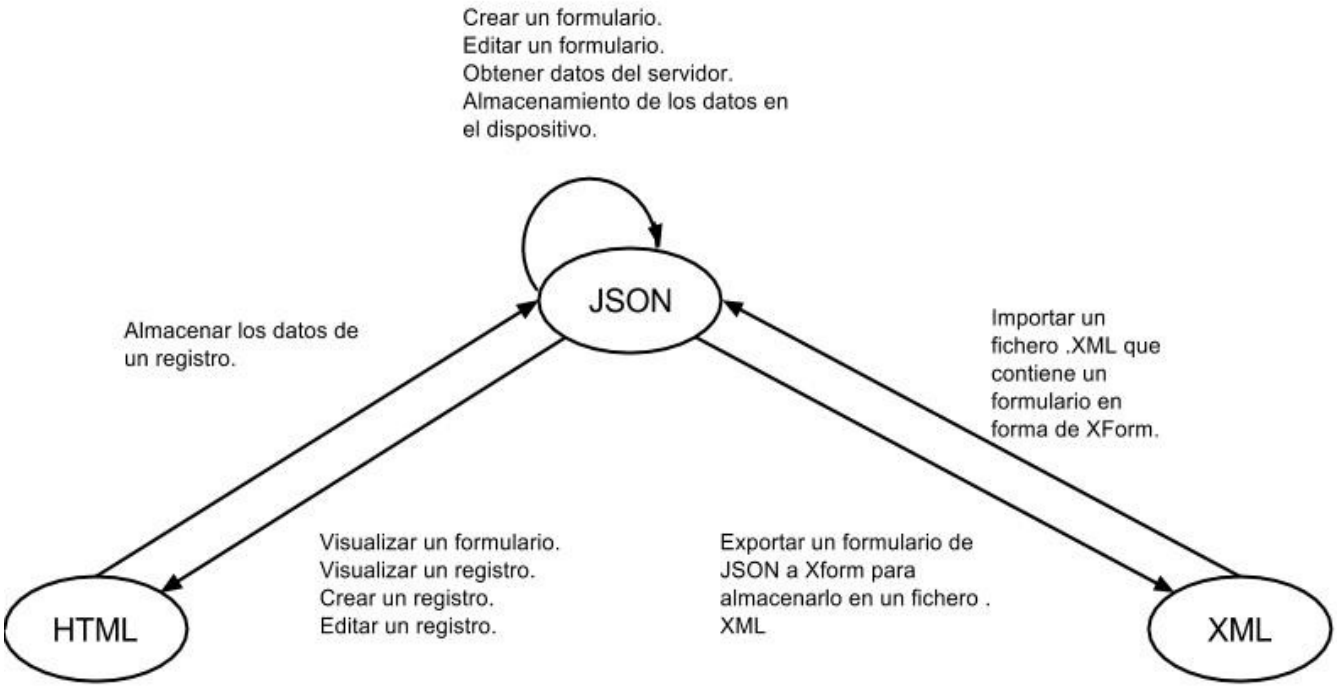
La entrada-salida de datos al sistema se verá de nuevo más adelante al tratar el diseño de la aplicación en el Diagrama de Flujo de Datos de Contexto.

3.4- Representación de la información:

Como podemos deducir de lo que ya se ha ido viendo, necesitamos hacer varias transformaciones en los datos para poder adaptarlos a la acción que se va a realizar sobre ellos.

Por ejemplo: hemos creado un formulario con la aplicación. Este formulario se guarda en formato JSON en el dispositivo. Para poder visualizarlo, se debe convertir a código HTML que genere el formulario en la interfaz de usuario. Así, el usuario puede introducir datos. Estos datos, se deben recoger de la interfaz para poder enviarlos. Para ello, se convierten a JSON. Si se desea almacenar el formulario en formato .XML se debe convertir el JSON en el código Xform correspondiente. Y si se desea leer un fichero .XML se debe convertir este código a JSON. Después se podrá convertir a HTML o a XML de nuevo o transmitirlo a través de la red.

Vamos a ilustrar estas transformaciones de la información con un diagrama:



Aquí podemos ver las conversiones que se realizan de los datos para poder llevar a cabo las distintas acciones que componen la aplicación. Para ello se han debido construir los conversores correspondientes que traducen de un formato a otro.

Para construir estos conversores se han utilizado las funciones del Modelo de Objetos del Documento (DOM) con JavaScript, que se aplica a documentos HTML y XML bien construidos. Este modelo permite programar estos documentos definiendo su estructura lógica en forma muy similar a la estructura en árbol. Cada elemento constituye un nodo y la estructura del documento es un conjunto de nodos ordenados de forma lógica. Accediendo a los elementos del DOM se puede acceder también a sus atributos y realizar las conversiones entre las distintas formas de representarlos en cada formato.

3.5- Resumen del modelo:

El modelo de la aplicación es el modelo cliente-servidor. Los dispositivos móviles constituyen los clientes sobre los que funciona la aplicación. Desde ellos, el usuario puede crear herramientas para la recogida de la información utilizando la función específica para ello que ofrece la aplicación. También puede utilizar formularios creados por ésta u otra aplicación en formato Xform que se encuentren almacenados en ficheros .XML y que estén en la memoria del dispositivo. Para esto se utilizan funciones nativas del dispositivo.

Desde la aplicación cliente se pueden crear formularios, modificarlos, rellenarlos y crear registros. Todo ello puede almacenarse en el dispositivo utilizando la capacidad de almacenamiento local, por lo que no es necesario encontrarse en un entorno que disponga de conexión a red.

Algunas operaciones relacionadas con el manejo de formularios y registros requieren que los datos se manipulen cambiando su formato. Para ello se han construido varios conversores que se encargan de transformar formularios desde su formato JSON al ser creados a HTML para visualizarlos o a XML. Lo mismo sucede para los registros: éstos pueden pasarse de JSON a HTML

para editarlos y se pasan de HTML a JSON al ser creados.

En el lado del servidor tenemos una base de datos MySQL donde se almacenan formularios y registros. Los clientes pueden comunicarse con el servidor tanto para enviar información como para descargarla al dispositivo para trabajar de forma local. Estas operaciones requieren de un entorno con conexión a la red.

Para facilitar la gestión de la información en ambos lados del sistema se utiliza un sencillo sistema de sincronización basado en el intercambio de las claves que definen formularios y registros al establecer la conexión cliente-servidor.

A continuación, se define el diseño del sistema en el que podrá verse de forma detallada el camino que siguen los flujos de datos entre los procesos que componen la aplicación desde su entrada en el sistema a su salida del mismo.

4- Requisitos:

4.1- Introducción:

En este apartado vamos a recoger los requisitos que debe cumplir la aplicación. Estos requisitos vienen dados tanto por la voluntad de los usuarios finales (aquellos que en base a unas necesidades detectan la necesidad de la aplicación) como por las exigencias propias del modelo de aplicación que se desea desarrollar.

Distinguiremos varios tipos de requisitos:

- Requisitos generales: especificarán de forma general la funcionalidad del sistema.
- Requisitos hardware: aquellos que vienen dados por el modelo de la aplicación, definen las necesidades hardware que serán claves en la implantación del sistema.
- Requisitos funcionales: describirán de forma detallada las funciones que debe realizar el sistema, es decir, el comportamiento que se espera de él.
- Requisitos no funcionales: aquellos que no afectan directamente a la funcionalidad del sistema pero que vienen dados por restricciones impuestos al mismo o por lo que el usuario debe percibir de él.

4.2 - Requisitos generales:

- El sistema implantará una forma sencilla de crear formularios y gestionarlos.
- El sistema implantará una forma sencilla de recoger datos utilizando los formularios creados con ese fin. Además implantará una forma sencilla de gestionar esta información.
- El sistema dispondrá de un sistema de almacenamiento local de la información persistente aunque la aplicación se reinicie.
- La aplicación podrá enviar datos a una base de datos situada en un servidor externo.
- La aplicación podrá recibir datos desde una base de datos situada en un servidor externo.

4.3 - Requisitos hardware:

En esta aplicación se aplica el modelo cliente-servidor, por lo que los requisitos hardware implican la implementación del sistema siguiendo este modelo. Las características específicas para cada parte del sistema son las siguientes:

- Cliente:
  - La aplicación debe funcionar independientemente del lugar. Por ello ha de ser desarrollada específicamente para dispositivos móviles.
  - La independencia del lugar implica independencia de posibilidad de conexión a redes de datos. Su funcionamiento no debe estar condicionado por la posibilidad de conexión a red de datos.
  - La aplicación debe estar disponible para varios tipos de dispositivo móvil. Esto incluye independencia de la plataforma y del tamaño del dispositivo.

- Servidor:
  - No se establecen requisitos previos de soporte de peticiones por unidad de tiempo, aunque se presupone que el servidor debe soportar un número medio de peticiones.

Este requisito se irá adaptando dependiendo de las necesidades que se vayan detectando tras su aplicación en la organización.
- La máquina servidor debe disponer de Apache.
- La base de datos alojada en el servidor debe ser de tipo MySQL.

4.4- Requisitos funcionales:

Los requisitos funcionales establecen la funcionalidad que debe proveer la aplicación. Define el comportamiento del sistema, sus funciones o componentes.

- Se debe poder crear un formulario abstracto con independencia del número de campos.
- Los formularios deben ser capaces de contener distintos tipos de datos (texto, número, fecha, listas de selección única, listas de selección múltiple, texto largo, decimales).
- Se deben poder crear registros para cada formulario para recabar los datos necesarios.
- La aplicación debe poder utilizarse en modo sin conexión, esto significa que debe existir una forma de almacenamiento en modo local tanto para las definiciones de formularios creadas como para los registros que contengan la información recogida.
- La aplicación debe poder enviar esos datos a un servidor si tiene conexión a Internet.
- Debe existir alguna forma de sincronización entre cliente y servidor para facilitar la gestión de información entre ambos extremos.
- Los datos deben poder ser almacenados remotamente en el servidor.
- Se deben poder obtener datos almacenados por el usuario o por otros usuarios en el servidor para almacenarlos de modo local y trabajar con ellos cuando no se disponga de conexión a red de datos.
- La aplicación tiene que generar formularios que puedan ser exportables a formato .XML para poder usarse en la aplicación Open Data Kit del proyecto Java Rosa.
- La aplicación tiene que ser capaz de procesar formularios creados en la aplicación Open Data Kit en formato .XML.
- La aplicación tiene que proveer de una forma de gestión de los formularios creados (es decir, de las tareas de creación, visualización, edición y eliminación de los mismos).
- La aplicación tiene que proveer de una forma de gestión de los registros creados (es decir, las tareas de visualización, edición creación y eliminación de los mismos).

4.5- Requisitos no funcionales:

A diferencia de los requisitos funcionales, los no funcionales no describen funciones que deba realizar el sistema. Estos requisitos definen restricciones en el comportamiento del sistema como por ejemplo los derivados del rendimiento. También describen aspectos que el usuario percibe del sistema, pero que no constituyen parte de la funcionalidad del mismo, como aquellos relacionados con las interfaces de usuario.

Framework de desarrollo de la aplicación:

- La aplicación deberá desarrollarse utilizando el framework Phonegap para desarrollo multiplataforma para dispositivos móviles.

Requisitos de conectividad:

- La aplicación debe funcionar tanto en entornos en los que se disponga de conexión a red de datos como en entornos donde no sea posible establecer conexión.
- La aplicación se comunicará con un servidor especificado por el usuario para enviar o recibir datos.

Requisitos de implementación:

- La aplicación debe ser compatible con el mayor número posible de plataformas.
- La aplicación debe ser compatible con el mayor número de dispositivos móviles, independientemente de factores como el tamaño de la pantalla.

Requisitos de rendimiento:

- El tamaño de los datos que se envían a través de la red debe ser el menor posible.
- No se establecen requisitos específicos de rendimiento aunque es importante que la aplicación funcione con la mayor fluidez posible.

Requisitos de usabilidad:

- La interfaz de usuario debe ser intuitiva para que el usuario se familiarice con ella lo antes posible independientemente de sus conocimientos tecnológicos.
- La interfaz debe adaptarse a distintos tamaños de pantalla dependiendo del dispositivo móvil sobre el que se instale.

5- Diseño:

5.1- Introducción:

Como se ha detallado en apartados anteriores, las tecnologías que se han utilizado para realizar el desarrollo de la aplicación propuesta han sido HTML5, JavaScript y CSS3 para la parte del cliente, y PHP Y MySql para la parte del servidor.

Estos lenguajes pertenecen a la familia de lenguajes del paradigma estructurado. Este paradigma de lenguajes de programación posee unas características propias y diferentes de otros paradigmas como es por ejemplo el Paradigma Orientado a Objetos.

En el desarrollo con este tipo de lenguajes se disponen de unas herramientas específicas para guiar la fase de análisis y diseño del sistema.

En primer lugar vamos a mostrar los Diagramas de Flujo de Datos. Estos diagramas muestran la relación entre las entradas y salidas del sistema y los flujos de datos. Se comienza describiendo el diagrama de contexto donde podemos ver la relación del sistema con el exterior y se va refinando para reflejar los módulos que lo componen y el camino seguido por los flujos de datos entre los procesos.

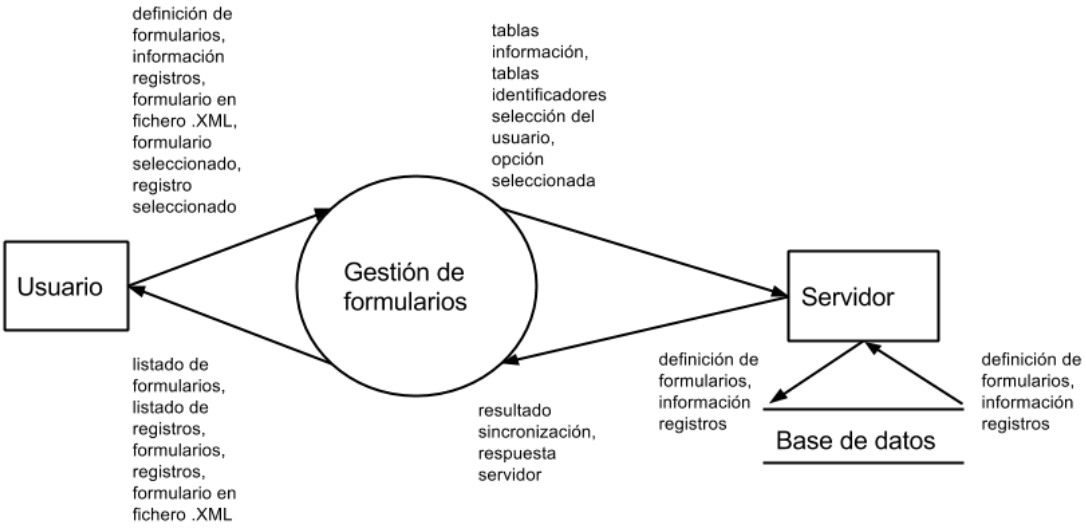
Posteriormente, se describirá el diagrama de estructura que constituye otra vista de los módulos definidos en el Diagrama de Flujo de datos.

Finalmente, se muestra un esquema entidad-relación de la base de datos alojada en el servidor, de modo que se puede observar la estructura de las dos tablas que la componen.

5.2- Diagramas de flujo de datos:

Comenzamos con el diagrama de datos de contexto que muestra como interactúa el sistema con el exterior, es decir, en nuestro caso con el usuario y con el servidor. A continuación se muestra el mencionado diagrama:

5.2.1 – Diagrama de contexto



El sistema interactúa con dos entidades externas: el usuario y el servidor. Existe un almacén de datos externo que está alojado en el servidor. Este almacén es la base de datos en la que se almacenan tanto las definiciones de formularios como las definiciones de registros.

El usuario introduce en el sistema las definiciones de formularios y los datos de registros. El sistema se encarga de gestionarlos y almacenarlos localmente. Asimismo, puede enviarlos a la otra entidad externa, el servidor, para que los almacene en el almacén de datos externo.

Se observa también la otra vía de entrada de definiciones de formularios: a través de ficheros .XML alojados en el dispositivo del usuario.

El usuario no sólo puede decidir qué ficheros .XML alimentan a la aplicación, sino que puede generar además ficheros .XML creados a partir de las definiciones JSON generadas con la aplicación constituyendo esta otra forma de salida de datos.

Por otra parte, una vez que el usuario está utilizando la aplicación, puede decidir qué formulario desea gestionar o sobre qué registro de qué formulario desea realizar una acción. Para ello, a través de la interfaz, el usuario obtiene primero un listado de formularios o, posteriormente, si ha seleccionado un formulario determinado, puede obtener un listado de los registros disponibles para el formulario seleccionado. El usuario selecciona la acción a realizar y a partir de ahí obtiene la respuesta deseada.

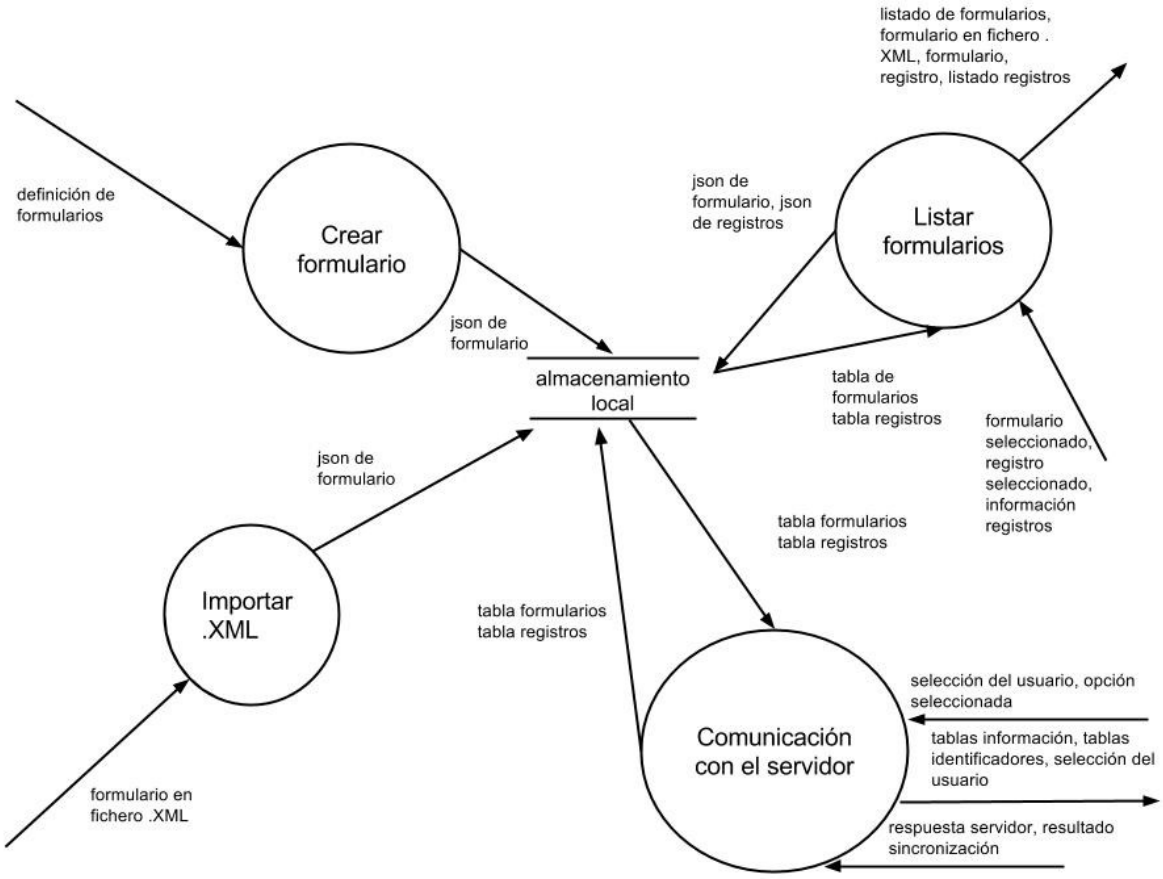
Para facilitar la sincronización entre el contenido del almacenamiento local y el almacenamiento externo, antes de enviar al servidor las definiciones de registros y formularios, se le envían los identificadores de registros y formularios. Estos identificadores vienen dados por el

código md5 de la cadena JSON que define formularios y registros, siendo calculado por la aplicación. Una vez definidos qué identificadores se encuentran en servidor y dispositivo local, el usuario puede seleccionar unos formularios o registros tanto para enviarlos al servidor, (y por tanto, ser almacenados en la base de datos) como para recibirlos (y por tanto, ser almacenados en el dispositivo local).

Se presenta a continuación el diagrama de nivel 1, donde se empiezan a desdoblar los procesos que se encargan de transformar los datos y de enviarlos de una parte a otra de la aplicación.

5.2.2 - Diagrama de nivel 1:

En este nivel se observan ya los procesos encargados de transformar y gestionar los formularios.



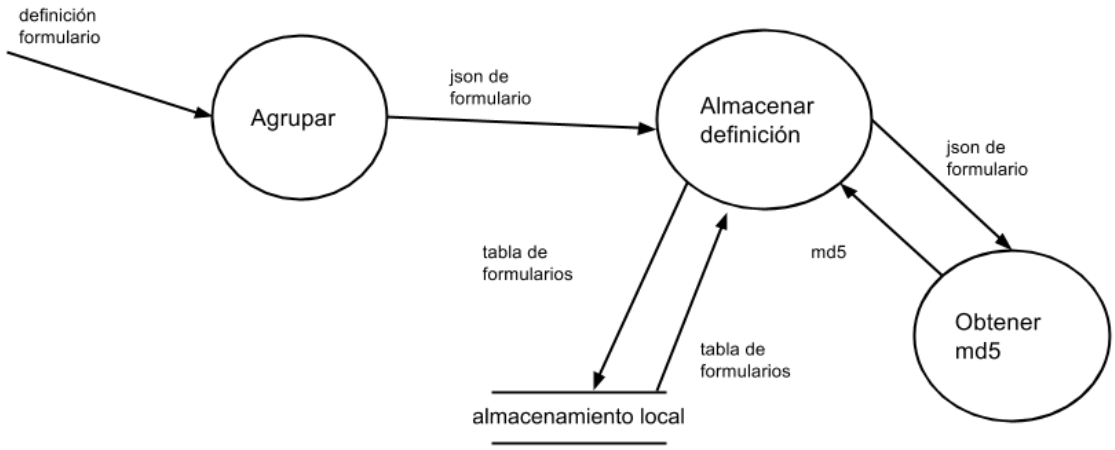
Los procesos principales que se encuentran en este nivel son los siguientes:

- Crear formulario: Este proceso se encarga de generar un formulario en formato JSON utilizando para ello los datos introducidos por el usuario. Estos datos vienen representados por “definición formulario” y se compone de los nombres de los campos que el usuario desea representar en el formulario, el tipo de estos campos, el nombre del formulario y la descripción. Aunque a este nivel no puede apreciarse, al crear un formulario se calcula su hash md5 utilizando una función. Este código, único para cada definición distinta de formularios, será el identificador de cada formulario. Este proceso, finalmente, almacena en el almacenamiento local del dispositivo en una tabla destinada específicamente para ello e identificada a través de la clave “formularios” tanto el código md5 como la definición del formulario en formato JSON.
- Importar .XML: No solo se pueden obtener formularios a través de las definiciones introducidas por el usuario, sino que se pueden importar a la aplicación ficheros .XML que almacenen la definición de un formulario generados por la aplicación Open Data Kit del proyecto Java Rosa. La aplicación toma el fichero elegido por el usuario y lo transforma en

una definición JSON que es almacenada en el dispositivo local y que puede ser tratada por los demás procesos de la aplicación. De este modo, se da a la aplicación compatibilidad con los formularios generados con la aplicación Open Data Kit. Sin embargo, existe una restricción para el uso de esta funcionalidad. Dado que la aplicación no permite anidar grupos de datos dentro de otros grupos de datos (esto significa que únicamente es posible definir un nivel de grupo de datos dentro de la aplicación), no se importarán correctamente aquellos ficheros que contengan definiciones que no cumplan esta restricción. Como se explicará más adelante, sería interesante considerar esta funcionalidad para posibles mejoras futuras.

- Gestionar formularios: En este proceso se producen todas las actividades relacionadas con la gestión de los formularios, como puede ser su edición, visualización, obtención de listado.... Para ello la aplicación es capaz de presentar al usuario un listado con los formularios disponibles, y, tomando la selección de formulario hecha por el usuario, ofrecerle las opciones de gestión del mismo.
- Comunicación con el servidor: Este proceso se encarga de la comunicación con la entidad externa Servidor. Su funcionalidad es la de facilitar la sincronización de datos entre almacenamiento local y servidor (de forma que hace posible que el usuario gestione mejor el contenido de su dispositivo local y lo alimente con los datos introducidos en la base de datos externa por otros usuarios), enviar datos al servidor para su almacenamiento en la base de datos y la recogida de datos del servidor. Este proceso sólo funcionaría si el dispositivo posee conexión a Internet. Si este no fuera el caso, no podrá obtener los datos del almacén externo, pero se pueden generar datos para enviarlos más tarde al mismo para que puedan ser compartidos con otros usuarios.

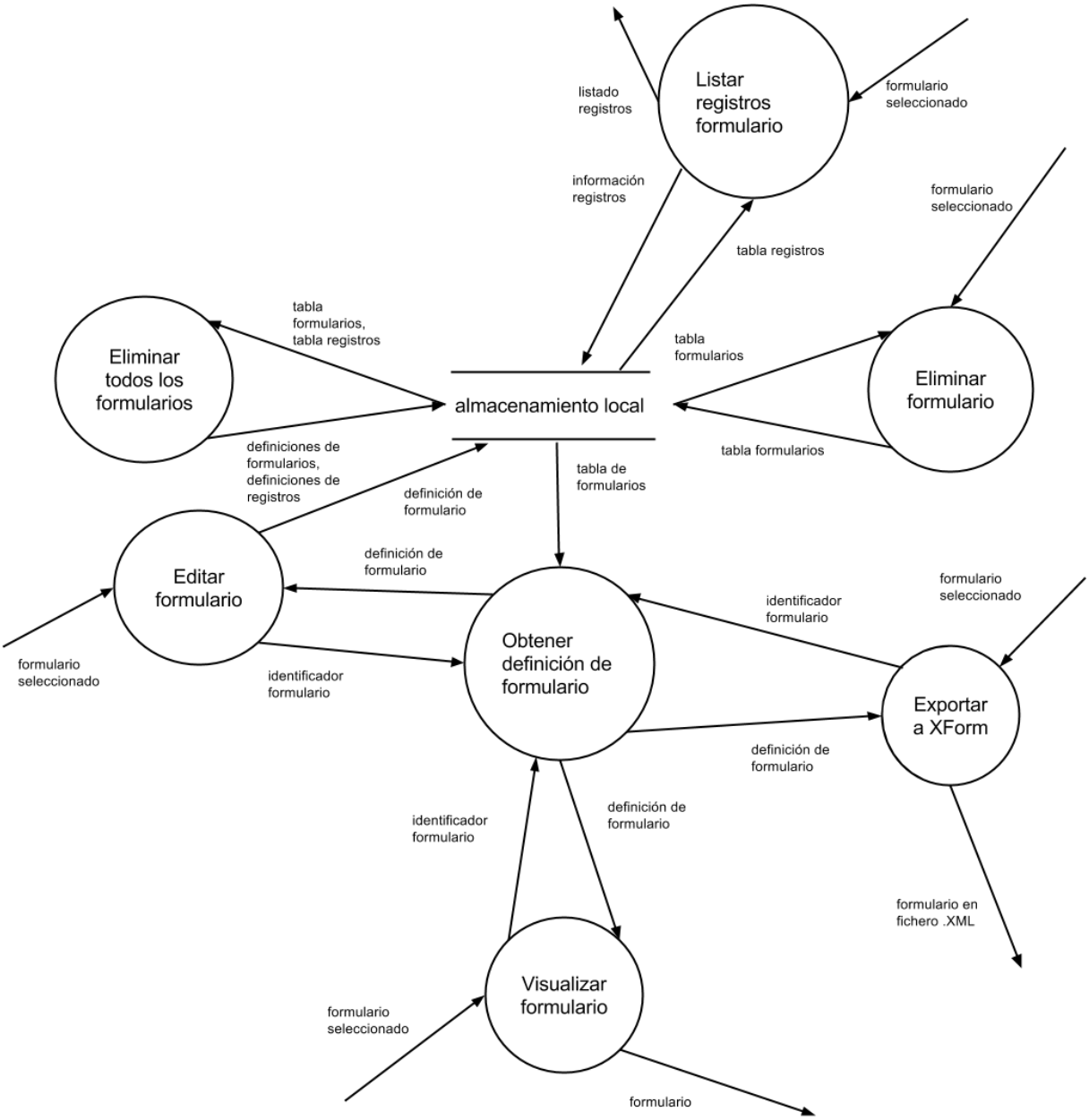
5.2.3 - Diagrama de flujo de datos 2: Crear formularios:



En este diagrama se observan los procesos implicados en la tarea de almacenar un formulario en el dispositivo. Se compone de los siguientes subprocesos:

- Agrupar: Este es el proceso que genera un JSON tomando para ello los datos introducidos por el usuario.
- Almacenar definición: Este proceso toma el JSON generado y obtiene un identificador único, el md5 correspondiente a la cadena JSON, usando la función “obtener md5”. Cuando ya dispone de ambos datos, lee del almacén local la tabla de formularios y la devuelve modificada, habiendo introducido el nuevo formulario en la misma.
- Obtener md5: Esta función recibe una cadena (en este caso el JSON que corresponde a la definición de un formulario) y devuelve su código md5 para que sea utilizado como identificador único del formulario.

5.2.4 - Diagrama de flujo de datos 2: Listar formularios:



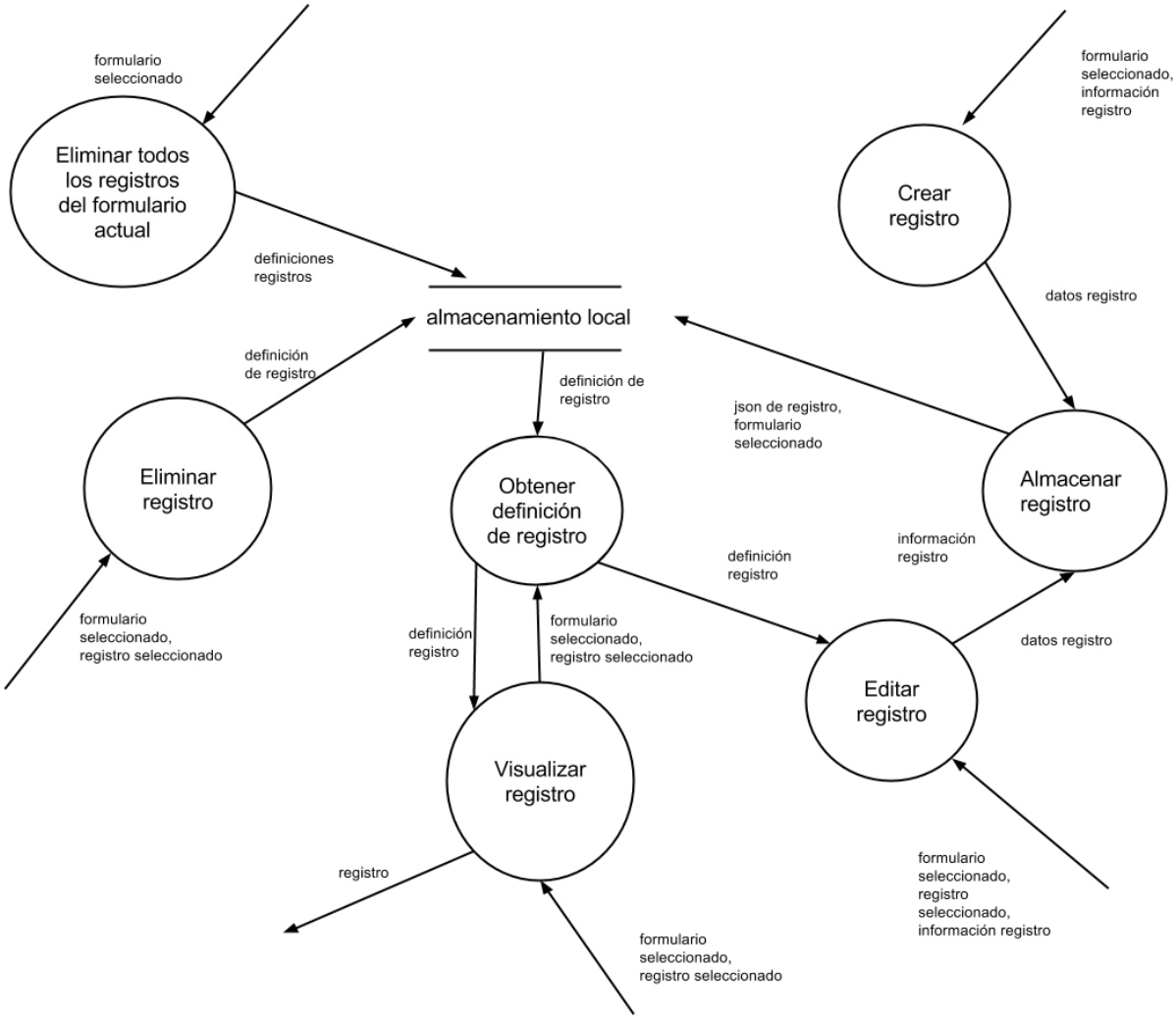
En este nivel se observan los procesos involucrados en el proceso de Gestión de los formularios. Estos procesos son accesibles desde que se ha accedido al proceso de Listar formularios. Se detalla cada uno de ellos a continuación:

- Eliminar formulario: El usuario puede seleccionar un formulario del listado de formularios obtenido al seleccionar el proceso de gestión de formularios para eliminarlo del dispositivo local. Para ello, el proceso recibe del usuario el formulario a eliminar (su identificador) y tomando la tabla de formularios la modifica eliminando el formulario seleccionado. Este

proceso además elimina los registros almacenados para ese formulario. Este proceso no elimina el formulario del almacén de datos externo en el servidor, si es que ya estuviera almacenado en él.

- Eliminar todos los formularios: Este proceso elimina del dispositivo todos los formularios almacenados así como sus registros. En definitiva, lo que este proceso hace es eliminar del almacenamiento local las claves “formularios” y “registros” con lo que se obtiene la eliminación de ambas tablas con sus correspondientes definiciones.
- Obtener definición de formulario: Este proceso obtiene la definición de un formulario concreto. Esta información es útil para otros procesos como se va a observar a continuación. Para obtener la definición del formulario, previamente este proceso habrá obtenido el identificador único del formulario del que se desea obtener la definición. Posteriormente, accede al almacenamiento local y toma la tabla “formularios”. Devuelve el JSON asociado a ese identificador.
- Editar formulario: El usuario puede editar definiciones de formularios ya creadas. A través de este proceso, se puede cargar la definición de un formulario en un editor que posee el mismo aspecto que el generador de formularios. El usuario puede eliminar campos y grupos de datos, cambiar el nombre y la descripción del formulario. El proceso carga el JSON que almacena la definición del formulario en el editor y a partir de ahí, con los datos introducidos por el usuario, se modifica el JSON generando un JSON nuevo, con un identificador único md5 diferente del original y que puede ser almacenado también en el dispositivo local. La edición del formulario no modifica el formulario original, sino que genera uno nuevo tomando como base el formulario antiguo.
- Visualizar formulario: El formulario es almacenado en el dispositivo local en forma de JSON, pero el usuario puede desear visualizar la definición del formulario de forma que se comprenda de qué elementos está formado. Para ello se obtiene el formulario en HTML presentándolo al usuario de forma amigable. Este proceso es el encargado de tomar la definición del formulario haciendo uso del proceso “Obtener definición del formulario” y de convertirla en código HTML que será mostrado al usuario de forma sencilla.
- Exportar a Xform: Esta funcionalidad es la que permite que un formulario generado a través de la aplicación sea convertido en un fichero .XML interpretable por la aplicación Open Data Kit del proyecto Java Rosa. Para ello toma la definición JSON del almacén de datos externo y lo convierte en .XML de forma parecida a como el proceso “Visualizar formulario” transforma la definición JSON en código HTML.
- Listar registros: A través de esta función se puede acceder a la lista de registros disponible para el formulario seleccionado. Para poder obtener la lista de registros, se toma el identificador del formulario seleccionado por el usuario y se accede al almacenamiento local para obtener de él la tabla con todos los registros. Una vez disponible la lista de registros para el formulario actual, se pueden seleccionar las opciones disponibles para la gestión de los registros.

5.2.5 - Diagrama de flujo de datos 2: Listar registros:



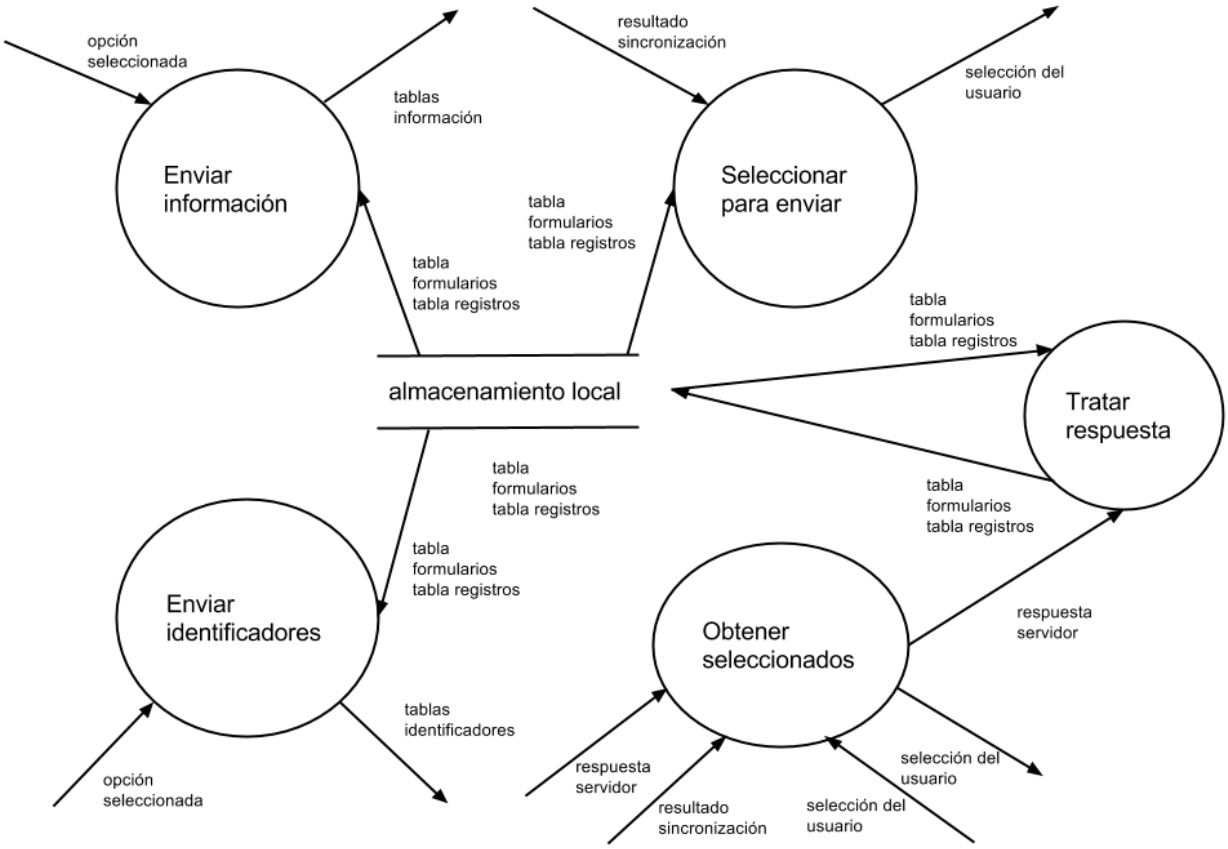
En este diagrama se observan los procesos involucrados en la gestión de los registros de cada formulario. Se detallan a continuación:

- Crear registro: Este proceso crea un registro para un formulario seleccionado por el usuario. Para ello, toma primero el identificador del formulario seleccionado y convierte la definición JSON del mismo en HTML para que el usuario pueda introducir la información de forma sencilla. Posteriormente, el usuario puede almacenar este registro en el dispositivo local. Para almacenarlo, primero se calcula el código md5 correspondiente a la cadena JSON de ese registro, de forma que en el almacén los registros se indexan por el md5 del formulario del que proceden y el md5 del formulario al que están asociados. Esto se verá en el siguiente nivel de DFD, al dividir esta burbuja.
- Eliminar registro: Este proceso elimina un registro seleccionado previamente por el usuario.

Para ello toma el formulario origen del registro y el identificador del registro y los elimina de la tabla con clave “registros” almacenada en el dispositivo:

- Eliminar todos los registros del formulario actual: Este proceso elimina todos los registros almacenados para un formulario seleccionado por el usuario. Para ello toma del mismo el identificador del formulario y, tomando del almacén la tabla “registros”, elimina el índice definido por el identificador del formulario.
- Obtener definición de registro: Este proceso obtiene la definición JSON de un registro. Para ello obtiene del usuario el identificador del registro y del formulario del que procede y accede al almacenamiento local para obtener la tabla de registros y de ahí seleccionar el JSON del registro correspondiente.
- Visualizar registro: A través de este proceso se pueden observar los datos almacenados para un determinado registro. El usuario selecciona el formulario y el registro que desea visualizar y el proceso obtiene la definición JSON del registro usando para ello el proceso “Obtener definición de registro”. Una vez obtenida la definición, la transforma en HTML mostrable en la interfaz de usuario, añadiendo a cada campo el atributo “disabled” de modo que se obtiene un visualizador de registros en modo “sólo lectura”.
- Editar registro: Un registro se puede editar seleccionando esta opción. Tal y como hacía el proceso anterior, el usuario selecciona el formulario y el registro que desea editar y el proceso obtiene la definición JSON del registro usando el proceso “Obtener definición del registro”. Después, transforma esta definición en código HTML pero a diferencia de como sucedía en el proceso anterior, no incluye el atributo “disabled”, sino que los campos aparecen rellenos con la información contenida en el registro pero pueden ser editados. Mas tarde, el usuario puede seleccionar la opción “Guardar registro como nuevo registro” o “Guardar registro sustituyendo al anterior”. La primera opción genera un nuevo registro calculando el nuevo md5 y guardándolo en el almacén de datos local, pero sin modificar el registro del cual procede. Si por el contrario se selecciona la segunda opción, se genera el nuevo md5 y se almacena el JSON en el almacén de datos local pero después se elimina el registro del que procede.

5.2.6- Diagrama de flujo de datos 2: Comunicación con el servidor:



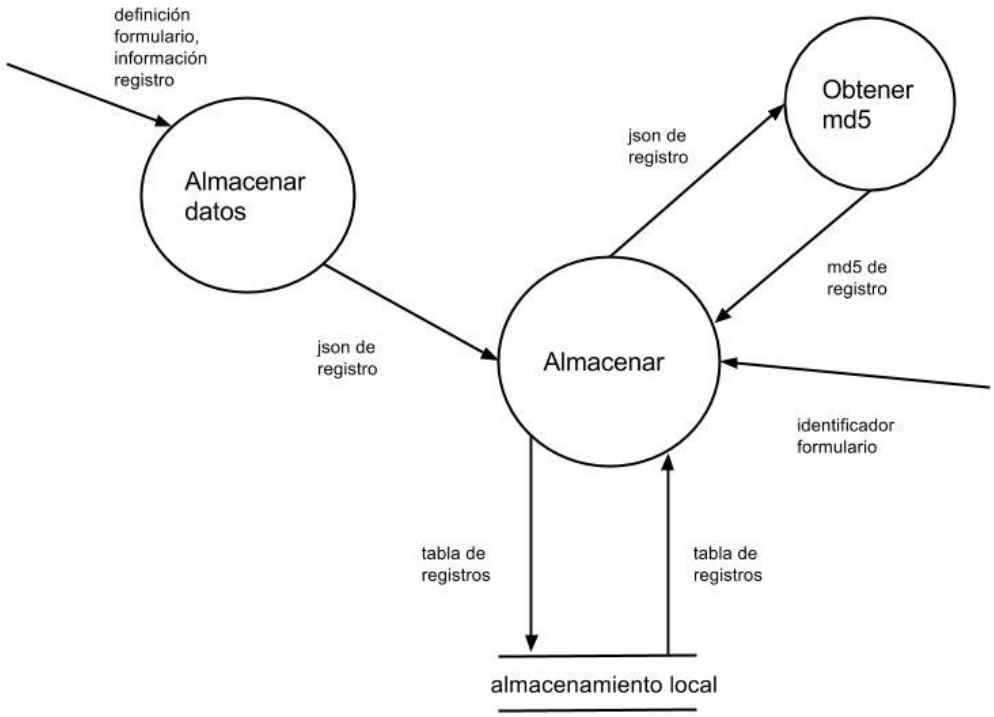
Aquí podemos ver los procesos implicados en la comunicación con el servidor. No se pueden observar sin embargo los procesos que siguen los datos una vez han llegado al servidor, ya que esos procesos dependen de la entidad externa. Se describen a continuación:

- Enviar información: Este proceso es el que se ejecuta si el usuario ha seleccionado la opción de enviar formularios y registros al servidor. Con este proceso se envían al servidor los identificadores de registros y formularios contenidos en el almacenamiento local así como los nombres, descripciones y campos primarios de los formularios.
  La finalidad de este envío previo es que el servidor compruebe qué identificadores posee almacenados en la base de datos para devolver una lista en la que aparecen coloreados en color verde aquellos formularios que se encuentran tanto en el almacenamiento local como en el servidor y en rojo aquellos que sólo se encuentran en el dispositivo. Así se puede ofrecer al usuario la posibilidad de seleccionar aquellos formularios que tan sólo se encuentran en el dispositivo, deshabilitando esta opción para aquellos formularios que ya se encuentran en el servidor. Ésta es una forma de evitar que se intente enviar varias veces la misma información.
- Enviar Identificadores: Este proceso se ejecuta si el usuario ha seleccionado la opción de obtener un listado de la información contenida en el servidor para posteriormente descargar aquellos elementos en los que esté interesado. Con este proceso se recogen la tabla de

formularios y registros del almacenamiento local para extraer los identificadores de todos los elementos, enviando esta lista al servidor. Una vez el servidor obtenga estos listados, comparará los identificadores recibidos con el contenido de la base de datos, devolviendo una lista de todos los elementos contenidos en ella en la que aparecerán coloreados en verde aquellos elementos que se encuentren en el dispositivo y en el servidor y en rojo aquellos que tan sólo se encuentren en el servidor, de forma que deshabilita la opción de seleccionar los elementos que ya se encuentran en ambos lados. Ésta es una forma de evitar que el usuario seleccione para descargar al dispositivo elementos que ya se encuentran en el almacenamiento local.

- Seleccionar para enviar: Este proceso recibe del servidor el listado de elementos comentado en el apartado “Enviar información”. Aparecen coloreados en verde aquellos elementos que están disponibles para su envío a la base de datos del servidor. El usuario selecciona aquellos que le interesa enviar y el proceso prepara dos tablas, una para los formularios y otra para los registros, con las definiciones de los elementos seleccionados, para enviarlos por AJAX al servidor. Allí se recibirán y el servidor ejecutará las sentencias SQL correspondientes para introducir la información en la base de datos.
- Obtener seleccionados: Este proceso recibe del servidor el listado comentado en el apartado “Enviar identificadores” en el que aparecen coloreados en verde aquellos elementos que se encuentran disponibles para su descarga desde el servidor y en rojo aquellos que se encuentran en servidor y dispositivo en el momento. El usuario selecciona aquellos que le interesa recibir y almacenar en su dispositivo y el proceso prepara dos tablas, una para los formularios y otra para los registros, con los identificadores de los elementos que desea recibir. Como respuesta recibe del servidor la información seleccionada por el usuario.
- Tratar respuesta: Este proceso recibe la respuesta del servidor obtenida por la petición del proceso anterior. Esta respuesta consiste en las tablas de formularios y de registros que contienen las definiciones para la selección hecha por el usuario. El proceso se encarga de almacenar esta información en el almacén de datos local, de forma que se encuentra ya disponible para su uso por el resto de la aplicación.

5.2.7 - Diagrama de flujo de datos 3: Almacenar registro:



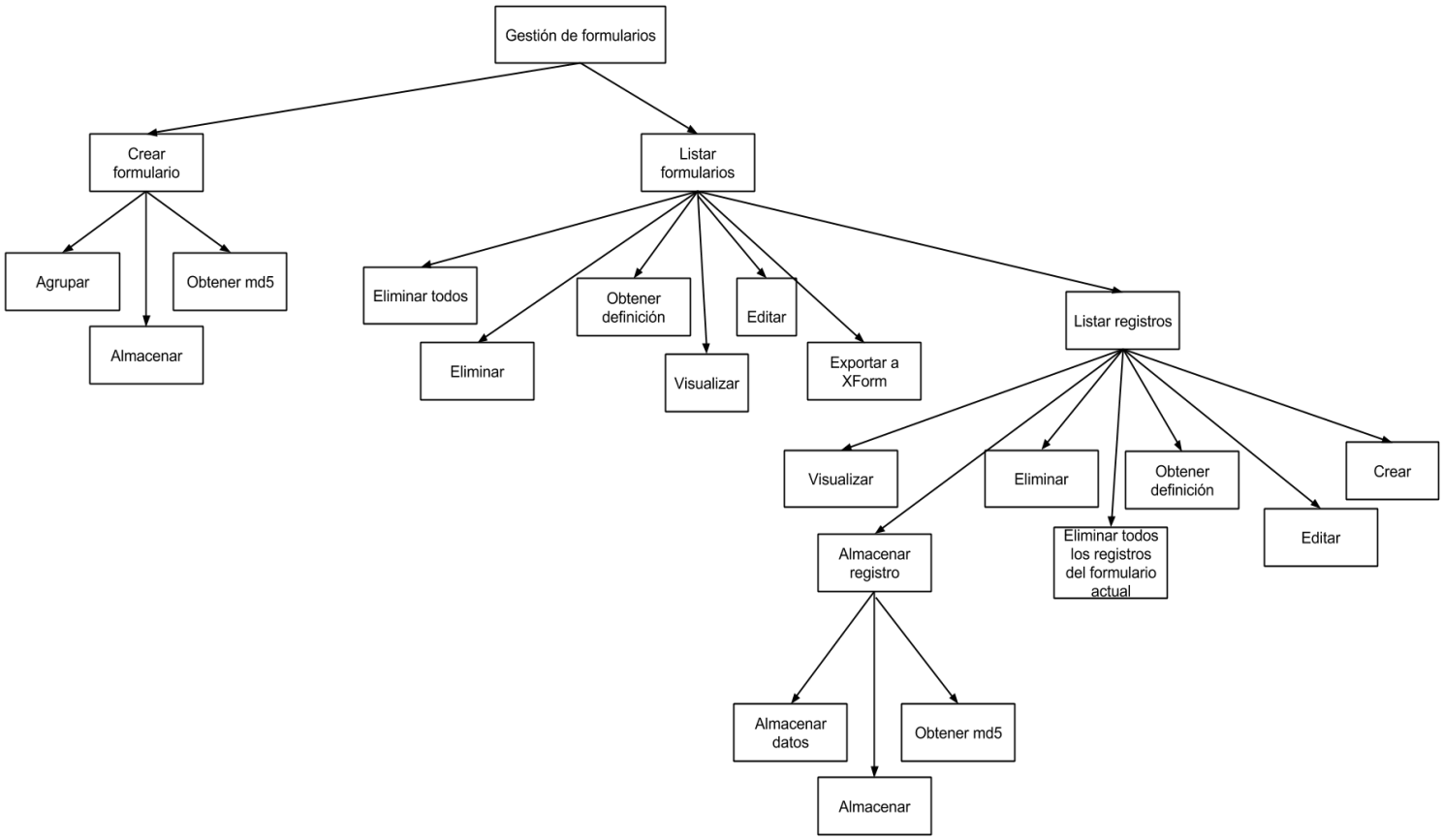
Aquí encontramos los procesos encargados de almacenar un registro de un formulario en el dispositivo. Están implicados los siguientes subprocesos:

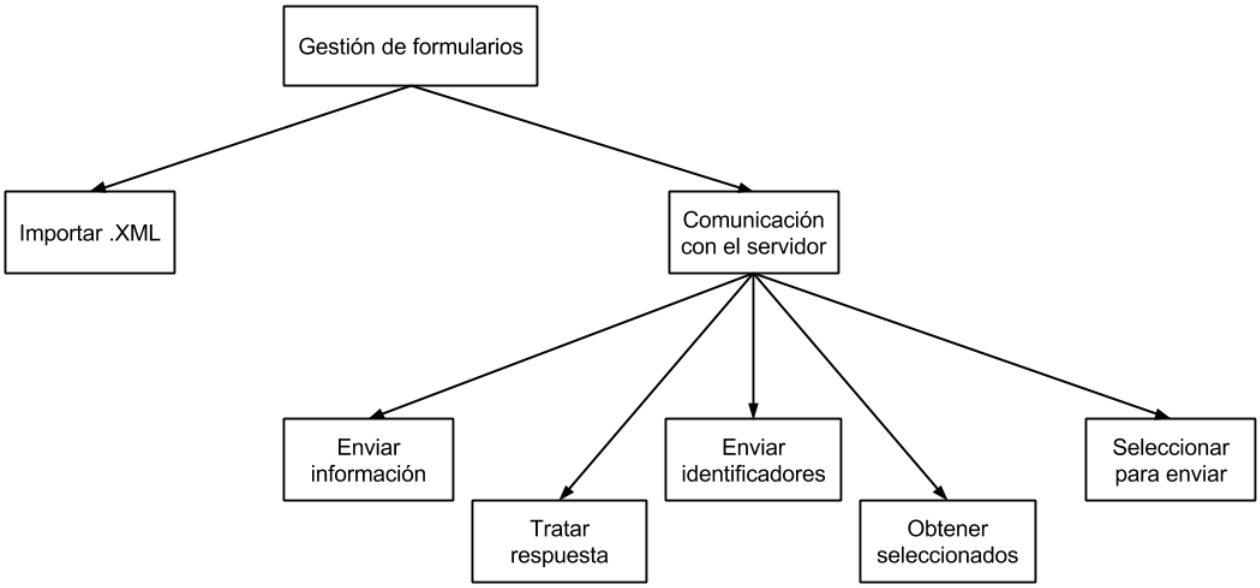
- Almacenar datos: Este proceso convierte la definición JSON de un formulario en código HTML para que el usuario pueda introducir los datos en el formulario. Cuando ha terminado, se obtiene la cadena JSON que contiene la información del registro.
- Almacenar: Este proceso es el que se va a encargar de almacenar la información del registro que se tiene en formato de cadena JSON en el almacenamiento local. Primero obtendrá el hash md5 de la cadena JSON para poder utilizarla como identificador único del registro. Lee del almacén de datos la tabla de registros y la devuelve modificada habiendo introducido en la misma el JSON del registro y su identificador único.
- Obtener md5: Esta función recibe una cadena JSON, en este caso la correspondiente al registro, y devuelve el código md5 que identifica a esa cadena.

Una vez que ya se han definido los diagramas de flujo de datos, vamos a ver los diagramas de estructura que permitirán una mayor comprensión de la estructura modular del sistema:

5.3- Diagrama de estructura:

El diagrama de estructura que corresponde al DFD de las páginas anteriores es el que sigue (se divide en dos partes para poder mostrarlo con claridad):

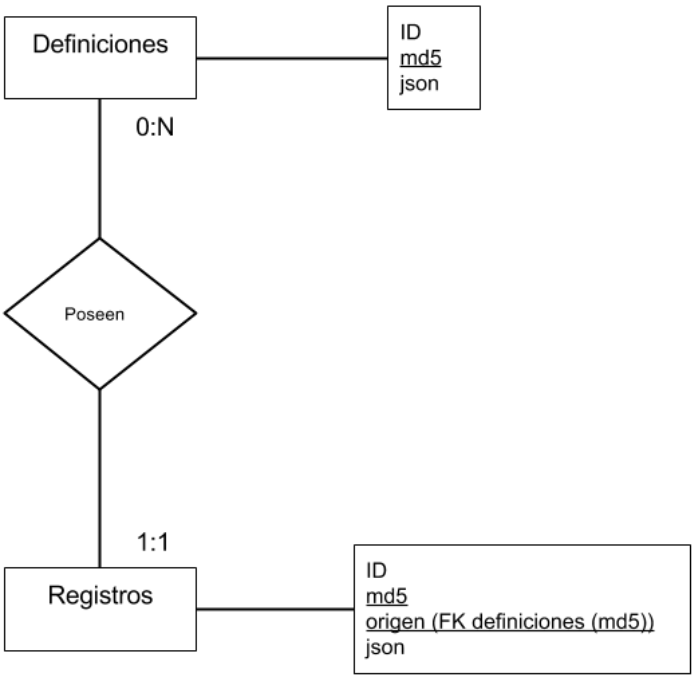




Como podemos ver, en el diagrama de estructura aparecen reflejados los módulos que habíamos definido en el DFD. El diagrama de estructura nos facilita la visión global del conjunto de módulos que componen la aplicación en el cliente, pudiendo apreciar más claramente las dependencias entre los procesos y su jerarquía.

5.4 – Base de datos.

Para poder ilustrar la base de datos almacenada en el servidor, se muestra a continuación un diagrama Entidad-Relación:



La base de datos alojada en el servidor es sencilla. Posee dos tablas, una para almacenar las definiciones de los formularios (definiciones) y otra para almacenar las definiciones de los registros (registros).

- Definiciones: Su clave primaria es el md5 del JSON que almacena. El JSON se almacena como una cadena de texto. Al ejecutar una sentencia SQL para obtener el JSON que corresponde a un md5, se obtiene la cadena de texto con el JSON completo, que posteriormente habrá que decodificar para convertir en objeto y por tanto para poder tratarlo. La tabla posee asimismo un campo id autoincremental.
- Registros: Su clave primaria contiene dos atributos: md5 y origen. En el primero se almacena el código md5 de la cadena que contiene el JSON con la información del registro. Origen es clave extranjera de la tabla definiciones y almacena el md5 del formulario origen del registro. Por tanto no se puede eliminar un formulario sin eliminar sus correspondientes registros (tal y como sucede, aunque realizado de forma distinta, en el dispositivo). El atributo JSON almacena, como en la tabla definiciones, una cadena de texto que posteriormente habrá que decodificar. En este caso este JSON almacena la definición del registro con la información que contiene.

6- Implantación:

6.1– Introducción:

Tal y como se detalló al tratar el modelo de la aplicación, este sistema responde a un caso típico de cliente-servidor.

La consecuencia de estar trabajando con este tipo de configuración es que para implantar el sistema desde cero vamos a tener que realizar dos trabajos por separado: la implantación en el cliente y la implantación en el servidor.

Estos son dos procesos que se realizan por separado siguiendo cada uno sus propios pasos.

Recordemos que es posible el funcionamiento del cliente independientemente de la conexión a un servidor, ya que se dispone de un sistema de almacenamiento local en el dispositivo que brinda una mayor independencia del escenario en el que se encuentra el usuario, sin embargo, si se desea acceder a toda la funcionalidad de la aplicación, será necesario tener configuradas las dos partes del sistema.

También es conveniente recordar que se puede disponer de más de un servidor en funcionamiento ya que el usuario puede seleccionar la dirección de aquel al que desea realizar la conexión. Esto hace que se pueda dividir la información en servidores diferentes según parámetros como por ejemplo la región o la clase de información que se almacena en ellos.

6.2– Necesidades para la implantación en el cliente:

Para la implantación del sistema en los dispositivos clientes actualmente es necesario un dispositivo Android 4.0 o posterior, ya que en versiones anteriores de la plataforma algunas de las funcionalidades del sistema no están disponibles (como el tratamiento de ficheros externos .XML o la generación de dichos ficheros).

Sin embargo, y dado que se ha utilizado el framework Phonegap de desarrollo multiplataforma, se podría adaptar, en un periodo relativamente corto de tiempo, la misma aplicación para otras plataformas como iOS, si bien actualmente no se encuentra disponible para plataformas distintas de la referida al comienzo.

El dispositivo cliente debe disponer de conexión a Internet para poder acceder a todas las funcionalidades de la aplicación (acceso a la información contenida en el servidor y envío de información al mismo), si bien esto no es normativo para la instalación de la aplicación ni para su uso al poder realizar la recogida de datos y posteriormente el intercambio de datos con el servidor una vez se ha accedido a una red de datos.

La aplicación se puede encontrar en un repositorio GitHub en la dirección siguiente: https://github.com/baile/dinamico_phonegap

Desde esta dirección se puede descargar un fichero .ZIP que contiene todo el código de la aplicación. Una vez se ha descargado el código fuente se puede compilar con Eclipse para obtener el instalador. El proceso de instalación de Eclipse se puede consultar en los anexos de esta memoria. Tras la compilación del código obtenemos el instalador en un fichero de formato APK.

Para instalar la aplicación conectamos el dispositivo al equipo donde hemos obtenido el fichero APK a través de un cable USB. Ejecutamos el fichero en el dispositivo, aceptamos los permisos necesarios para el funcionamiento de la aplicación y la instalamos.

Una vez instalada la aplicación accedemos a ella pulsando sobre el icono correspondiente en el menú de aplicaciones.

Para poder acceder al servidor donde se encuentra la base de datos externa, tendremos que configurar en la aplicación la dirección del servidor. Para ello accedemos a la opción “Opciones del servidor” y en el cuadro de texto que aparece en pantalla introducimos la dirección completa del servidor. Pulsamos “Guardar” y ya estaríamos listos para acceder a la información contenida en el servidor.

En caso de que no se pueda realizar la conexión (por que no exista conexión a Internet o por haber introducido una dirección incorrecta) la aplicación devolverá un mensaje de error indicando la imposibilidad de conectar.

6.3– Necesidades para la implantación en el servidor:

Respecto a la parte del servidor, debemos disponer de una máquina que posea un servidor Apache instalado y en funcionamiento. Además la máquina deberá tener instalados los módulos necesarios para la ejecución de PHP y MySQL server. El proceso de instalación de estos módulos y del servidor Apache está descrito en los anexos de esta memoria.

Como modo de facilitar la creación de la base de datos y su gestión se recomienda la instalación del módulo phpmyadmin. Igualmente, la descripción de los pasos necesarios para realizar esta instalación se encuentra en los anexos.

Para la creación de la base de datos accederíamos a la aplicación phpmyadmin. Una vez creada la base de datos se seguiría la especificación de tablas y campos de las tablas que se indica en la parte de diseño de la aplicación.

Una vez se dispone en la máquina servidor de todo el conjunto de módulos necesarios, encontramos los ficheros .php encargados de dar respuesta a las peticiones de los dispositivos clientes en el repositorio de GitHub en https://github.com/baile/dinamico_phonegap.

Los ficheros .php necesarios se encuentran en el directorio /dinamico_phonegap/assets/www/php. Estos ficheros se copiarían en el servidor en la carpeta www para hacerlos accesibles desde el exterior.

Tendremos que modificar uno de estos ficheros para configurar la conexión del servidor a la base de datos que hemos creado en pasos anteriores. El fichero a modificar es connect.php
Las variables a modificar serán las del comienzo del fichero en donde se indica el nombre de usuario de la base de datos, la contraseña y el nombre de la base de datos a la que se va a realizar la conexión. Guardamos los cambios realizados en este fichero.

Con este último paso ya tendríamos configurada la máquina servidor para empezar a atender las peticiones de los dispositivos clientes.

Como los usuarios pueden seleccionar desde sus dispositivos el servidor al cual quieren realizar la conexión, se pueden tener varios servidores donde cada uno trabaja con su propia base de datos. El usuario escribirá la dirección del servidor al que desea conectar desde la aplicación, por lo que si en algún momento el servidor cambia de dirección sólo tendrá que darse a conocer al usuario la nueva dirección de conexión con el mismo.

7- Pruebas:

7.1– Introducción:

El proceso de pruebas es fundamental durante el desarrollo de una aplicación. Su finalidad es la detección y posterior reparación de errores antes de que la aplicación llegue al usuario final.

Durante el desarrollo de cada módulo de la aplicación se realiza una fase de pruebas con el objetivo de verificar su comportamiento. Cuando ya se disponen de varios módulos, se realizan pruebas para verificar su funcionamiento de forma conjunta y detectar errores en el paso de parámetros y en las llamadas a otros módulos.

Al finalizar la aplicación, se comprueba su funcionamiento en conjunto. En este punto, se pretende comprobar si la aplicación hace todo lo que debe hacer, esto es, si cumple con todos los casos de uso que se habían planteado.

Las pruebas que tratan de verificar el cumplimiento de los casos de uso son las pruebas de caja negra. Estas pruebas sólo tienen en cuenta las entradas y salidas del sistema. Tratan de comprobar que las entradas no generan salidas inesperadas. Se dice que cuando ya se han cubierto la mayor parte de casos de uso con éxito, la cobertura funcional es cercana al 100%.

Cuando ya se ha obtenido una cobertura funcional cercana al 100%, se realizan las pruebas de caja blanca. Estas pruebas tratan de comprobar todos los caminos lógicos que sigue la ejecución de la aplicación. Su función es comprobar que se ejecutan el mayor número de sentencias del código. En ocasiones, en las pruebas de caja negra no se tratan todas las bifurcaciones del código. Las pruebas de caja blanca se encargan de ejecutar todas las posibles. Si existen partes del código que no se llegan a ejecutar nunca, hay que replantearse la utilidad de esa parte del código. Por eso, para poder llevar a cabo este tipo de pruebas, es necesario poseer un profundo conocimiento del código de la aplicación.

Se considera que el proceso de pruebas ha sido exitoso si se han localizado el mayor número de errores existentes tras su aplicación. Si la batería de pruebas es de poca calidad, no se encontrarán gran parte de los errores existentes, aumentando las probabilidades de que lleguen al usuario final.

La participación de otras personas en el proceso de pruebas es de gran importancia. En ocasiones, los usuarios realizan acciones que el desarrollador no había planificado. Los usuarios ejecutan acciones inesperadas. Por ejemplo, pueden pulsar varias teclas a la vez si se impacientan, tratar de realizar acciones en un orden distinto del planteado e introducir datos inesperados. Es por ello que la participación de otras personas permite comprobar el comportamiento de la aplicación frente a distintos usos de los usuarios finales.

A continuación, se describe el proceso de pruebas seguido durante el desarrollo de esta aplicación.

7.2– Proceso de pruebas:

Durante el proceso de desarrollo de la aplicación se han realizado varias pruebas tanto de los módulos por separado como del funcionamiento conjunto.

En las pruebas de módulos unitarios se prueba un módulo aislado. Para ello, se generaban casos de prueba tratando de cubrir todas las posibilidades del caso de uso. Al generar estos casos de prueba se tenía especial atención a los valores límite por ser aquellos más propensos a los errores.

Tras probar cada módulo por separado y obtener resultados más o menos satisfactorios, se

procedía a la integración entre unos y otros. Al probar subconjuntos de la aplicación se intentan encontrar errores en el paso de parámetros entre módulos y en el paso de datos de unos a otros.

Finalmente se realiza la integración del conjunto del sistema y se comprueba el funcionamiento.

En nuestro caso, tras el desarrollo de la aplicación y tras haber realizado suficientes pruebas, se propuso a varias personas que probaran la aplicación.

La finalidad de estas pruebas es obtener el feedback de los usuarios en cuanto a interfaces de usuario, usabilidad, posibles errores derivados de entradas anómalas o no habituales, comportamientos anómalos de la aplicación y valoración del rendimiento de la aplicación.

Para realizar estas pruebas se instaló la aplicación en un teléfono smartphone y en una tableta, con objetivo de comprobar cómo percibían los usuarios la aplicación en distintos tipos de dispositivo móvil. Posteriormente, se les dio a los usuarios el dispositivo y tras explicar brevemente el propósito de la aplicación se les propuso que la utilizaran, tratando de descubrir las distintas opciones, de modo que se puede valorar qué tanto de intuitiva resulta para un usuario en el primer contacto.

Al finalizar un corto espacio de tiempo, se les pasó a los usuarios un cuestionario con varias preguntas que valoraban interfaz, rendimiento y aparición de errores o resultados inesperados.

Los usuarios poseían distintos niveles de conocimientos informáticos, desde usuarios poco familiarizados con la tecnología a usuarios expertos.

Se presenta a continuación el cuestionario que rellenaron los usuarios tras realizar las pruebas:

Cuestionario de usabilidad y rendimiento de la aplicación

Pregunta 1 – La aplicación es fácil de utilizar:

Muy en desacuerdo	1	2	3	4	5	6	Muy de acuerdo

Pregunta 2 – El menú está bien organizado y es sencillo encontrar las funciones:

Muy en desacuerdo	1	2	3	4	5	6	Muy de acuerdo

Pregunta 3 – Se comprende rápidamente la función de cada botón.

Muy en desacuerdo	1	2	3	4	5	6	Muy de acuerdo

Pregunta 4 – Los botones están bien organizados y su función es la esperada.

Muy en desacuerdo	1	2	3	4	5	6	Muy de acuerdo

Pregunta 5 - La información se presenta de forma adecuada en cada pantalla:

Muy en desacuerdo	1	2	3	4	5	6	Muy de acuerdo

Pregunta 6– El tamaño de los elementos en pantalla es adecuado:

Muy en desacuerdo	1	2	3	4	5	6	Muy de acuerdo

Pregunta 7– Los colores de la aplicación son los adecuados.

Muy en desacuerdo	1	2	3	4	5	6	Muy de acuerdo

Pregunta 8– Valora la rapidez de ejecución de la aplicación.

Muy lenta	1	2	3	4	5	6	Muy rápida

Pregunta 9– La aplicación se bloquea o devuelve un error.

○ Si ○ No

Pregunta 10– Si la respuesta a la pregunta anterior es afirmativa, realiza una descripción de las acciones que llevaron al bloqueo o al error.

Pregunta 11- La impresión general de la aplicación es

Muy negativa	1	2	3	4	5	6	Muy positiva

Nombre:

Fecha:

Estas pruebas pueden considerarse un subconjunto de las pruebas de caja negra dado que no se tiene en cuenta el código, sino las entradas y las salidas del sistema. Decimos que son un subconjunto dado que no se puede asegurar que se probaran todas las posibles entradas al sistema y todo el conjunto de funcionalidades.

En cualquier caso, los resultados obtenidos son de gran interés dadas las observaciones hechas por los usuarios.

7.3– Resultados según el feedback de los usuarios:

Se presenta a continuación una relación de mejoras sugeridas por los usuarios a tener en cuenta en líneas futuras y una relación de errores encontrados:

Con respecto a la interfaz de usuario:

- Tamaño más grande de los elementos (sobre dispositivos de pantalla más grande)
- Posibilidad de girar el dispositivo.
- Cambiar el color de los elementos que actúan como botones al pulsarlos.
- Cambio en las fuentes.
- Ayuda inmediata acerca de cada función o leyenda que explique su funcionalidad.
- En el menú en la versión adaptada a pantallas más pequeñas la sensibilidad debería estar en la totalidad del menú, no sólo al pulsar sobre la palabra.
- Explicar de modo distinto algunas opciones ya que no se comprende completamente su función o significado.
- Al guardar un registro o un formulario, además de sacar el mensaje de confirmación llevar directamente a la pantalla de gestión de formularios o de registros ya que el permanecer en la pantalla de creación de registros/formularios crea confusión al usuario.
- Agrandar el espacio dedicado a añadir opciones a las listas.
- Establecer una ruta específica para almacenar los ficheros .XML que se generan con la aplicación.
- Hacer que al crear un elemento del tipo “lista” se sitúe el puntero directamente sobre él en lugar de preparar la creación de un nuevo elemento.
- El estilo al visualizar un formulario o un registro se asemeja a una lista y no se distinguen del todo los elementos que lo forman. Podría darse estilo al formulario que se visualiza mediante sombreados y líneas suaves que separen los elementos que lo conforman.
- La pantalla “Crear Formulario” tiene una disposición de elementos que crea confusión a los usuarios.
- Añadir un mensaje durante la carga del contenido del servidor al acceder al mismo ya que se genera confusión al usuario durante la espera.
- El tamaño de los campos en la funcionalidad “Crear formulario” es demasiado pequeño como para leer las instrucciones que contienen antes de editarlo.

Con respecto a la funcionalidad:

- Gestión de errores al rellenar un registro.
- Añadir la funcionalidad de crear campos obligatorios para no permitir que se queden vacíos. (Añadir la obligatoriedad como propiedad del elemento)
- Evitar que en las listas se generen opciones con el mismo nombre.
- Crear un campo de búsqueda para la ayuda.

- Crear un campo de búsqueda para filtrar los formularios por alguno de sus campos.
- Botón para volver a la página inicial que posee los iconos con las opciones de la aplicación.
- Almacenar el formulario si durante su creación se pulsa otro botón involuntariamente o mostrar un mensaje de advertencia antes de abandonar la pantalla.

Con respecto al rendimiento:

- En todos los casos los usuarios apuntaron una ejecución rápida de la aplicación, sin mostrar fallos o deficiencias en el rendimiento.

Errores:

- Al girar el dispositivo se cierra la aplicación sin dar aviso.
- Al editar un formulario no se exportan los registros creados antes de la edición, sino que se crea uno nuevo que no contiene registros, generando confusión al usuario.

7.4– Resultados según la funcionalidad:

Se muestra a continuación un listado de las funcionalidades de la aplicación y de los errores encontrados, varios de los cuales han sido subsanados al finalizar el desarrollo de la aplicación. De este modo desdoblamos los errores según el módulo en el que se encuentran.

Funcionalidad: **Crear formulario**
Descripción: Se genera un formulario utilizando distintos casos de prueba (un grupo de datos/varios, campos de tipo simple como texto, texto largo, número, fecha y campos complejos como listas).
Acceso a la funcionalidad: El usuario accede a la opción “Crear formulario” e introduce los datos requeridos (nombre del formulario, descripción) y los datos deseados. Finalmente pulsa la opción “Almacenar”.
Resultado: El resultado es el almacenamiento del formulario en el almacenamiento local del dispositivo. Aparece un mensaje confirmando la acción. Si no se introduce el nombre del formulario se lanza una alerta y se aborta la operación de almacenamiento. Lo mismo ocurre si no se introduce la descripción.

Relación de errores encontrados:

- Corregidos: No funciona el tipo de dato decimal (No sale al visualizar o editar formulario como si no se hubiera creado). La fecha se observa como un campo de texto ordinario debido a la falta de soporte del elemento HTML5 input date de Android 4.x.

Funcionalidad: **Exportar a Xform.**
Descripción: A partir de un formulario ya creado y almacenado en el dispositivo local, se genera un fichero .XML que contiene una definición Xform válida para el formulario.
Acceso a la funcionalidad: El usuario accede a la opción “Gestionar formularios” y selecciona la opción “Exportar Xform” en el formulario que desea exportar.
Resultado: Se obtiene un fichero .XML con una definición Xform válida para el formulario.

Relación de errores encontrados:

- Corregidos: No aparece ningún mensaje que indique que se ha almacenado el formulario

correctamente, aunque si que se genera el fichero. Al exportar un formulario en ODK no se muestran los campos que son de tipo “texto”. El fichero resultado de exportar el formulario no corresponde a una definición válida Xform.

Sugerencia de los usuarios:

- El fichero se almacena en la raíz de la tarjeta de memoria en lugar de almacenarse en una carpeta específica para este fin. Sería más cómodo si existiera una carpeta destino para los ficheros .XML.

Funcionalidad: **Importar Xform.**
Descripción: A partir de un formulario almacenado en forma de Xform en un fichero .XML que se encuentra en la memoria del dispositivo se obtiene en pantalla el formulario y la posibilidad de almacenarlo en el formato interno de la aplicación.
Acceso a la funcionalidad: El usuario accede a la opción “Importar XForm” y selecciona el formulario a importar en el cuadro de selección que aparece en pantalla.
Resultado: Se visualiza el formulario en pantalla y se da la posibilidad de almacenarlo en la aplicación.

Relación de errores encontrados:

- Sin corregir: En versiones más antiguas de Android no se ejecuta correctamente esta opción debido a un problema de compatibilidad. Esto genera confusión. Debería eliminarse esta opción si la aplicación se está ejecutando en versiones de Android que no sean compatibles con esta funcionalidad.

Funcionalidad: **Listar formularios.**
Descripción: El usuario obtiene un listado de los formularios almacenados localmente en el dispositivo y el conjunto de opciones que permite realizar distintas acciones sobre los formularios. Además se le da la opción de eliminar todos los formularios almacenados en el dispositivo.
Acceso a la funcionalidad: El usuario accede a la opción “Gestionar formularios” en el menú de la aplicación.
Resultado: Se obtiene un listado con los formularios almacenados y las opciones disponibles. Si no existen formularios almacenados se muestra un mensaje al usuario avisando de esta situación.

Funcionalidad: **Visualizar formulario.**
Descripción: Se muestra en pantalla el formulario generado de forma que se pueden ver los campos creados.
Acceso a la funcionalidad: El usuario accede a la opción “Gestionar formularios” en el menú de la aplicación y selecciona la opción “Visualizar formulario” en uno de ellos.
Resultado: El usuario recibe el formulario por pantalla en un modo visual.

Funcionalidad: **Editar formulario.**
Descripción: Esta función permite editar un formulario ya creado y almacenado en el dispositivo local para añadir o eliminar campos, así como cambiar el nombre o la descripción.
Acceso a la funcionalidad: El usuario accede a la opción “Gestionar formularios” en el menú de la aplicación y selecciona la opción “Editar formulario” en uno de ellos.
Resultado: Se muestra la pantalla de creación de formularios donde se han cargado el nombre y la

descripción del formulario y sus elementos, de forma que el usuario puede añadir o eliminar campos.

Relación de errores encontrados:

- Corregidos: No se cargan en la pantalla las opciones de los elementos listas de selección única o listas de selección múltiple. No se puede guardar el formulario y falla el "campo primario" ya que se muestra en blanco sin mostrar el valor anterior y aunque se rellene no permite guardar obteniendo el mensaje "Introduce un nombre para el campo primario".
- Sin corregir: Al editar un formulario se almacena como uno nuevo, sin copiar los registros que contenía el anterior en el formulario editado.

Funcionalidad: **Crear registro.**
Descripción: Se pueden rellenar los campos de un formulario ya creado, constituyendo una forma de recogida de información. Después se almacenan localmente en el dispositivo.
Acceso a la funcionalidad: El usuario selecciona la opción "Gestionar formularios" en el menú de la aplicación y selecciona la opción "Gestionar registros". Desde ahí puede seleccionar "Crear registro".
Resultado: Se carga en pantalla el formulario mostrando todos sus campos y permitiendo que el usuario introduzca datos en ellos. Se le da la opción de almacenar el registro en el dispositivo local.

Funcionalidad: **Listar registros.**
Descripción: Ofrece un listado de los registros almacenados para un formulario determinado y muestra todas las opciones disponibles para cada uno de ellos.
Acceso a la funcionalidad: El usuario selecciona la opción "Gestionar formularios" en el menú de la aplicación y selecciona "Gestionar registros" en uno de los formularios.
Resultado: Se obtiene un listado de los registros disponibles para ese formulario con sus opciones, además de dar la opción de crear uno nuevo. Si no existen registros almacenados para ese formulario se avisa al usuario y se le da la opción de crear uno nuevo.

Relación de errores encontrados:

- Corregidos: Si no existen formularios almacenados en lugar de mostrar un mensaje avisando de la no existencia de registros almacenados, la aplicación no hace nada, ya que se obtiene un error al intentar acceder a elementos de una tabla que aún no ha sido definida.

Funcionalidad: **Visualizar registro.**
Descripción: Se muestra un registro de un formulario, pero no se permite su edición (sólo lectura).
Acceso a la funcionalidad: El usuario selecciona la opción "Gestionar formularios" en el menú de la aplicación y selecciona "Gestionar registros" para uno de los formularios. A continuación selecciona "Visualizar registro" en un registro de la lista de formularios.
Resultado: Se visualiza en pantalla el registro con toda la información que contiene pero los campos aparecen deshabilitados, de forma que el usuario no puede cambiar los datos contenidos en el registro.

Funcionalidad: **Editar registro.**
Descripción: Permite que un usuario edite el contenido de un registro que estaba almacenado en el dispositivo.

Acceso a la funcionalidad: El usuario selecciona la opción “Gestionar formularios” en el menú de la aplicación y selecciona “Gestionar registros” para uno de los formularios. A continuación selecciona “Editar registro” en un registro de la lista de formularios.
Resultado: Se visualiza en pantalla el registro mostrando toda la información que contiene, y permitiendo que el usuario cambie esa información. Se le da la opción de almacenar el registro como uno nuevo o sustituyendo al anterior.

Funcionalidad: **Almacenar registro como nuevo registro.**
Descripción: Almacena un registro editado como si fuera uno nuevo.
Acceso a la funcionalidad: Se accede desde la funcionalidad “Editar registro”, seleccionando la opción “Almacenar como registro nuevo”.
Resultado: Se crea un nuevo registro que se almacena localmente con la información que el usuario ha introducido al editar un registro ya existente.

Funcionalidad: **Almacenar registro sustituyendo al anterior.**
Descripción: Al editar un registro, lo almacena sustituyendo al registro del que proviene.
Acceso a la funcionalidad: Se accede desde la funcionalidad “Editar registro” seleccionando la opción “Almacenar como registro nuevo”.
Resultado: Se elimina el registro del que proviene el registro editado y se almacena éste.

Funcionalidad: **Eliminar registro.**
Descripción: Elimina un registro del almacenamiento local.
Acceso a la funcionalidad: El usuario selecciona la opción “Gestionar formularios” en el menú de la aplicación y selecciona “Gestionar registros” para uno de los formularios. A continuación selecciona “Eliminar” en un registro de la lista de formularios.
Resultado: Se elimina el registro seleccionado de la lista de registros. Se obtiene un mensaje que confirma la eliminación del registro seleccionado.

Funcionalidad: **Eliminar todos los registros.**
Descripción: Elimina todos los registros asociados a un formulario determinado.
Acceso a la funcionalidad: El usuario selecciona la opción “Gestionar formularios” en el menú de la aplicación y selecciona “Gestionar registros” para uno de los formularios. A continuación selecciona “Eliminar todos los registros” en la parte inferior de la lista.
Resultado: Se eliminan todos los registros asociados al formulario, por lo que desaparece la lista de registros apareciendo un mensaje advirtiendo de la inexistencia de registros asociados y la opción de crear un registro nuevo.

Funcionalidad: **Eliminar formulario.**
Descripción: Elimina un formulario del almacenamiento local y sus registros asociados.
Acceso a la funcionalidad: El usuario selecciona la opción “Gestionar formularios” en el menú de la aplicación y selecciona “Eliminar” para uno de los formularios.
Resultado: Aparece un mensaje de advertencia para que el usuario confirme o cancele la acción. Si la cancela, no ocurre nada. Si la confirma, se eliminan los registros asociados al formulario y después se elimina el formulario, apareciendo un mensaje de confirmación de la acción realizada.

Relación de errores encontrados:

- Corregidos: Al intentar eliminar un formulario que no contiene registros la aplicación no hace nada, ya que se obtiene un error (intenta eliminar registros de la tabla de registros pero no existe ninguno).

Funcionalidad: **Eliminar todos los formularios.**
Descripción: Elimina todos los formularios y todos los registros del almacenamiento local.
Acceso a la funcionalidad: El usuario selecciona la opción “Gestionar formularios” en el menú de la aplicación y selecciona “Eliminar todos los formularios” en la parte inferior del listado de formularios.
Resultado: Aparece un mensaje de advertencia para que el usuario confirme o cancele la acción. Si la cancela no ocurre nada. Si la confirma se elimina toda la tabla de registros y toda la tabla de formularios, apareciendo un mensaje de confirmación de la acción realizada. En la pantalla del listado de formularios aparece un mensaje informando de la inexistencia de formularios.

Funcionalidad: **Enviar formularios y/o registros al servidor.**
Descripción: Envía al servidor los formularios y/o registros seleccionados por el usuario.
Acceso a la funcionalidad: El usuario selecciona la opción “Opciones del servidor” en el menú de la aplicación y selecciona “Enviar datos al servidor”. Antes de realizar esta acción puede definir la dirección del servidor a la que se va a realizar el envío rellenando el cuadro de texto dispuesto para este fin y pulsando “Guardar”.
Resultado: Se obtiene por pantalla un error si no es posible establecer la conexión con el servidor. Si se ha conectado, se obtiene un listado de los formularios y registros almacenados localmente en el dispositivo donde aparecen en color verde los que ya se encuentran también en el servidor y en rojo aquellos que están sin enviar. El usuario puede seleccionar los que desea enviar de entre los coloreados en rojo. Al pulsar “Enviar” los envía al servidor y se colorean en verde para distinguir aquellos que ya se han enviado. Además aparece un mensaje de confirmación de la acción realizada.

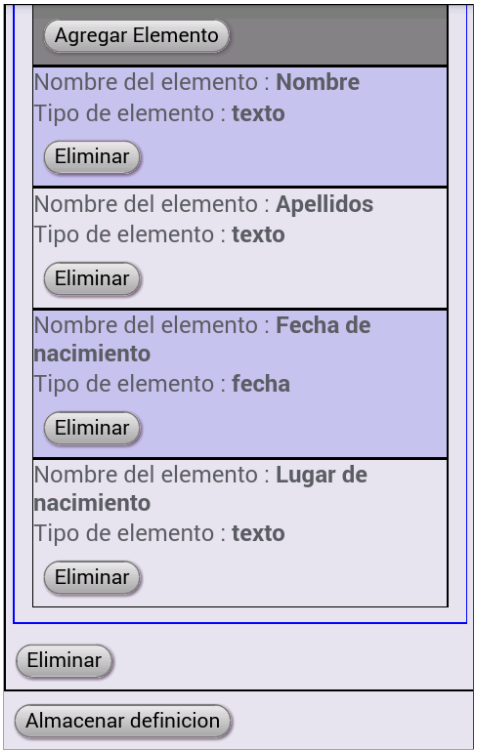
Funcionalidad: **Descargar formularios y/o registros desde el servidor.**
Descripción: Descarga del servidor los formularios y/o registros seleccionados por el usuario.
Acceso a la funcionalidad: El usuario selecciona la opción “Opciones del servidor” en el menú de la aplicación y selecciona “Obtener datos del servidor”. Antes de realizar esta acción puede definir la dirección del servidor a la que se va a realizar la conexión rellenando el cuadro de texto dispuesto para este fin y pulsando “Guardar”.
Resultado: Se obtiene por pantalla un error si no es posible establecer la conexión con el servidor. Si se ha realizado la conexión se obtiene un listado de los formularios y registros almacenados en la base de datos alojada en el servidor donde aparecen en color verde los que ya se encuentran también en el dispositivo y en color rojo los que se encuentran tan sólo en el servidor. El usuario puede seleccionar los que desea descargar al dispositivo de entre los coloreados en rojo. Al pulsar “Descargar” los recibe en el dispositivo y se colorean en verde para distinguirlos de aquellos que aún no están almacenados localmente. Además aparece un mensaje de confirmación de la acción realizada.

8- Conclusiones:

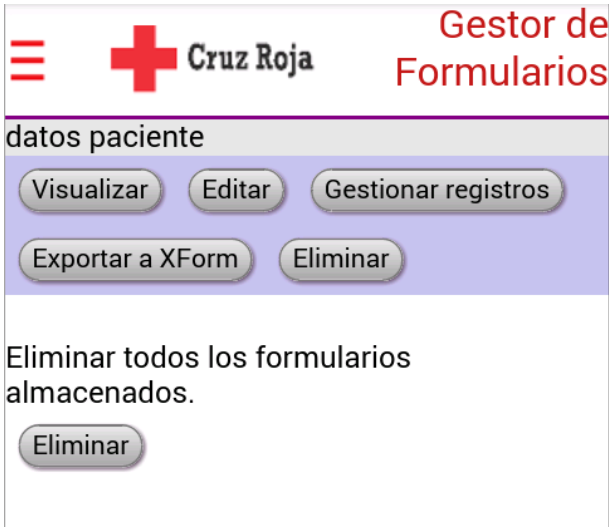
Una vez finalizado el proceso de desarrollo del proyecto, se puede concluir que llegados a este punto se ha obtenido:

- Una aplicación adaptada a dispositivos móviles 4.0 en adelante con toda la funcionalidad necesaria para la creación de herramientas de recogida de información y capaz de realizar el proceso mismo de recogida. Entre las funcionalidades implementadas se encuentran la gestión de los formularios creados, permitiendo su visualización, edición, borrado y exportación a .XML según el estándar de JavaRosa. Asimismo, se han implementado funcionalidades para la gestión de la información recogida, como la visualización, edición y borrado de registros.
  Se muestran a continuación unas imágenes tomadas de la aplicación en las que se puede visualizar su funcionamiento:

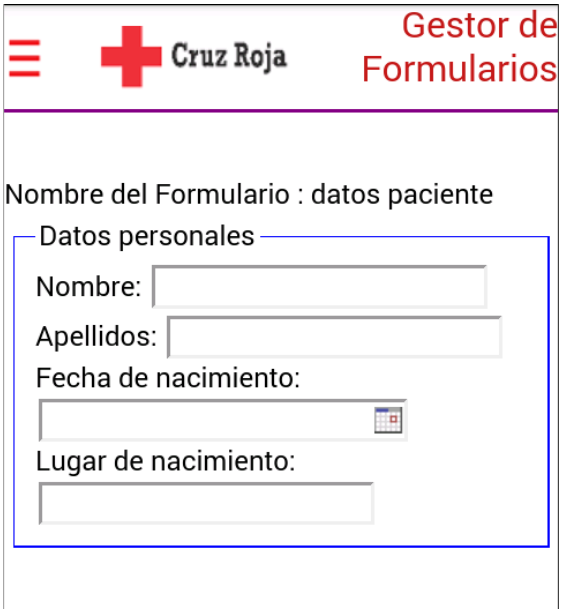
1- Crear formulario:



2- Gestionar formularios:

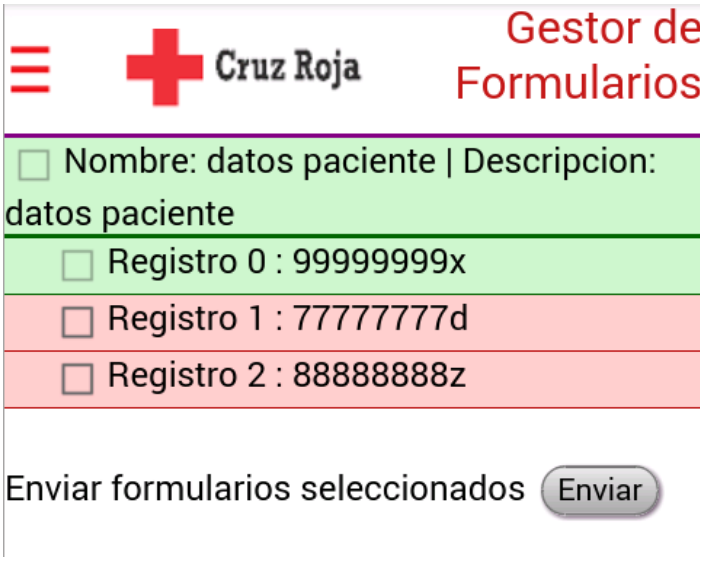


3- Visualizar formulario:



4- Gestionar registros:

5- Acceso al servidor:



- Se ha obtenido una configuración sencilla para el servidor, siendo este un punto clave para la posterior implementación del sistema. Esta sencillez asociada al servidor reduce el conocimiento técnico necesario para la puesta en marcha del sistema. Dado que la utilidad de la aplicación residirá en la gestión de datos obtenidos en situaciones de muy diverso tipo sobre el terreno en las que el personal implicado en la obtención de datos es muy

heterogéneo, se valora la sencillez como un punto muy favorable.

- Disponemos de un código fuente fácil de adaptar a otras plataformas. Como el código que corresponde al núcleo de la aplicación está basado en tecnologías web, a través del uso de Phonegap será muy sencillo adaptar la aplicación para su uso en otras plataformas como iOS. Además este código es fácil de mantener dado que en su mayor parte se encuentran separadas aquellas parte del código dedicadas a la funcionalidad, a la apariencia y a la interfaz.
- Se dispone de una interfaz de usuario capaz de adaptarse automáticamente al tamaño de la pantalla del dispositivo, de modo que se optimiza para su uso tanto en teléfonos (que poseen un tamaño de pantalla más reducido) como en tabletas (donde el tamaño de la pantalla es más amplio).
- Se cuenta además con una evaluación realizada por participantes en el proceso de pruebas. Esta evaluación nos sirve para fijar las líneas futuras de desarrollo de la aplicación con la finalidad de mejorar tanto las interfaces como la funcionalidad para adaptar la aplicación lo máximo posible a las necesidades y preferencias de los usuarios. En fases posteriores se prevee una continuación del proceso de pruebas por los usuarios finales, la realización de mejoras y solución de errores, y su posterior implementación en el entorno real de uso de la aplicación.

En el siguiente apartado vamos a describir las líneas futuras de la aplicación, proponiendo las mejoras que se podrían realizar para continuar el proceso de desarrollo.

9- Líneas futuras:

9.1 - Introducción

La aplicación puede ser desarrollada en el futuro en varias direcciones. Estos cambios pueden clasificarse según la parte a la que afectan. Recordando que nuestra aplicación sigue el modelo cliente-servidor se pueden reconocer funcionalidades que mejorarían el desempeño de cada una de las partes.

Para establecer las líneas futuras de la aplicación es imprescindible el feedback de los usuarios, ya que ellos son quienes definirán las necesidades que quedan sin cubrir o aquellas que surjan en el tiempo. Además, establecen sus impresiones acerca de la interfaz de usuario, dando lugar a posibles mejoras en materia de usabilidad.

Para recoger las impresiones de los usuarios en nuestro caso se ha utilizado un modelo de cuestionario. Esta herramienta aparece asociada al proceso de pruebas de la aplicación, y una vez analizados los resultados pueden inferirse ideas para mejoras futuras.

A continuación, se muestran parte de las mejoras a realizar, habiendo tenido en cuenta tanto los resultados del feedback de los usuarios como otras ideas que podrían completar o mejorar la aplicación construida.

9.2 - En la parte servidor

La aplicación tal y como está desarrollada tiene mucho más peso en la parte del cliente que en la del servidor. En el futuro, podría desarrollarse más ampliamente esta parte, pudiendo establecer un control de usuarios con acceso a la base de datos alojada en el servidor. Asimismo, al crear este control de usuarios, podrían establecerse varios niveles de acceso de los mismos a la información alojada en la base de datos. Por ejemplo, podrían existir usuarios administradores capaces de administrar otros usuarios y de eliminar o modificar contenido de la base de datos. Los usuarios base podrían solamente acceder al contenido de la base de datos e insertar nuevas filas en las tablas de la base de datos.

Se podría establecer una mayor complejidad en los datos, para introducirlos de una forma más ordenada. Por ejemplo, se podría hacer que cada usuario pertenezca a un “departamento “ y sólo pudiera acceder a los formularios o registros almacenados para ese departamento. Ello supondría que existirían vistas adaptadas a cada usuario. Esto tiene sentido puesto que actualmente se está mostrando mucha más información de la necesaria a los usuarios. Puede que un usuario de un departamento no esté interesado en información correspondiente a un departamento distinto del suyo. Además el hecho de que todos los usuarios puedan acceder a toda la información recogida por otros usuarios puede suponer un problema de privacidad.

Otra forma de mejorar la parte del servidor sería la modificación de la base de datos alojada en el mismo. Actualmente se está utilizando una base de datos MySQL, que es sencilla de utilizar. Sin embargo, se están almacenando en ella cadenas de texto JSON. Ello supone que no se pueden realizar consultas sobre el contenido de los formularios o de los registros, al haberse almacenado todos ellos en una única cadena de texto. Existen algunas bases de datos no relacionales que podrían solventar este problema. Este es el caso por ejemplo de MongoDB. Estas bases de datos funcionan con una filosofía distinta de la seguida por las bases de datos relacionales y actualmente se están considerando de gran utilidad para la resolución del problema de almacenamiento de datos de tipo JSON. En el futuro, podría establecerse una serie de pruebas de funcionamiento de la aplicación con este tipo de bases de datos, que probablemente abrirían un camino de nuevas posibilidades para

actualizaciones futuras.

9.3 - En la parte cliente

Esta parte también podría desarrollarse en el futuro con el objetivo de obtener una aplicación más completa.

Se pueden plantear varias mejoras relacionadas con la generación de formularios. Estas mejoras estarían encaminadas a la obtención de formularios más complejos y más específicos.

En este sentido observamos que actualmente no es posible la introducción anidada de grupos de datos, es decir, existe un grupo de datos y en él se introducen distintos campos con distintos tipos posibles de datos, pero no es posible introducir un grupo de datos dentro de un grupo de datos. Ello realmente limita el funcionamiento de la aplicación tanto en la generación de formularios con la misma como en el paso de ficheros .XML a la aplicación (ya que la aplicación Open Data Kit si que permite el uso de grupos de datos anidados). Solventar este problema dotaría a la aplicación de mayores posibilidades de generación de formularios y una mayor compatibilidad con la aplicación existente.

Continuando con la idea de creación de formularios más complejos podría establecerse la posibilidad de creación de campos obligatorios al definir el formulario. De esta forma, al crear un registro sería obligatorio rellenar los campos que posean esa propiedad, no pudiendo el usuario dejarlos en blanco.

También podría mejorarse la capacidad de creación de registros, es decir, la capacidad de recogida de información.

En este sentido, una funcionalidad muy interesante a añadir en el cliente sería la posibilidad de tomar imágenes a través de la cámara de fotos del dispositivo para documentar un registro de un formulario. Esta funcionalidad completaría los datos almacenados en un registro, al complementar la información escrita con la información visual.

La información contenida en los registros podría complementarse también añadiendo metadatos asociados como puede ser la fecha en que fue creado, el lugar (a través del uso de la geolocalización) o el usuario que lo creó (si el sistema de usuarios ya hubiera sido implementado). El uso de metadatos complementaría a la información contenida en el registro, facilitando su orden y su clasificación.

Otra posible mejora sería el desarrollo de la aplicación para otras plataformas de dispositivos móviles. Dadas las características del framework utilizado, esto no debería consumir gran cantidad de tiempo ni debería suponer una gran dificultad, puesto que el núcleo de la aplicación está hecho realmente de HTML5, CSS3 y JavaScript. Al no hacerse uso de un gran número de capacidades nativas del dispositivo, la adaptación de la aplicación a otras plataformas no parece que fuera a ser demasiado complicada. Esta adaptación ampliaría el número de dispositivos donde podría utilizarse, enfatizando aún más la idea de que pueda utilizarse en cualquier lugar, y por cualquier dispositivo.

Teniendo en cuenta la mejora comentada al hablar de las líneas futuras basadas en la parte servidor relacionada con la base de datos, se podrían añadir filtros de búsqueda de formularios o de registros. Cuando la cantidad de registros o de formularios es pequeña es sencillo navegar por la información y encontrar la buscada, pero si comienza a crecer la cantidad de información su manejo se complica. Para facilitar la navegación en este sentido es de gran utilidad el uso de herramientas de búsqueda.

Actualmente no se pueden realizar consultas acerca de campos del formulario porque los estamos guardando en la base de datos como si fueran cadenas de texto. Ello limita la capacidad de implementación de este tipo de filtros basados en el contenido de campos que actualmente están almacenados en el interior del JSON. Pero si se valorara la posibilidad de implementación de otros

tipos de base de datos que estén enfocadas al uso del tipo de datos JSON quizá en consecuencia podrían ampliarse las funcionalidades en este sentido.

La interfaz de usuario y navegabilidad general a través de la aplicación es susceptible de mejoras varias. Para establecer las líneas futuras en materia de interfaz y usabilidad se recoge el feedback de los usuarios de la aplicación y de las personas que la prueban. Los resultados se vieron al tratar la fase de pruebas de la aplicación detectando varios puntos mejorables en la interfaz que facilitarían a los usuarios la adaptación a la aplicación haciéndola más intuitiva y sencilla de usar.

10- Anexos:

10.1- Instalación de Phonegap:

A continuación, se describen los pasos a seguir para la instalación de Phonegap para crear aplicaciones móviles.

Para poder instalar Phonegap, primero es necesario tener en el ordenador el programa Eclipse. Este programa podemos encontrarlo en la web www.ecplipse.org/downloads. No es necesaria ninguna versión específica de Eclipse, se puede instalar simplemente la versión Standard.

Para desarrollar aplicaciones en Android (y, por tanto, poder probarlas en el Emulador Android) es necesario instalar en Eclipse el Android SDK que provee de las herramientas necesarias para instalar y probar las aplicaciones Android.

También es necesario instalar ADT (Android Developer Tools), que es una herramienta para Eclipse que da interfaz a varios comandos de la herramienta SDK. Esta herramienta está integrada en Eclipse.

Lo más cómodo es instalar las 3 herramientas a la vez. Para ello se dispone del ADT Bundle, que provee de la herramienta Eclipse con el plugin ADT, las herramientas SDK y las herramientas para la plataforma Android. Esta es la herramienta utilizada en el proyecto. Se pueden encontrar links para su descarga en la web http://developer.android.com/sdk/index.html. En este caso, dado que se está utilizando Linux Mint, la descarga fue la correspondiente a este sistema operativo.

Una vez que ya se dispone de esta herramienta, se puede proceder a instalar Phonegap. La forma más rápida y sencilla de llevar a cabo esta operación es la siguiente:

1- Iniciar Eclipse.
2- Pulsar Help → Install New Software
3- En el apartado “location” se escribe la siguiente ruta:
http://svn.codespot.com/a/eclipselabs.org/mobile-web-development-with-phonegap/tags/r1.2/download
4- Aparecen las herramientas a instalar, se seleccionan y se pulsa “Next”. Tras aceptar las condiciones comenzará la instalación. Al finalizar, se debe reiniciar la aplicación.

Una vez instalada, aparece un icono en la parte superior con el logo de Phonegap. Si se pulsa sobre este icono, se podrá crear un nuevo proyecto en Phonegap.

La versión de Phonegap instalada mediante este método es la 1.4.1.

10.2- Instalación de Firefox y complemento Firebug:

Es muy útil el uso del navegador Firefox por la posibilidad de utilizar Firebug. Este conjunto de herramientas son una ayuda útil durante el desarrollo web, ya que facilitan la localización y seguimiento de errores en el código CSS, HTML y JavaScript que van a ser el núcleo del proyecto Phonegap. La interfaz de la aplicación se puede ir probando en el navegador, aunque para probar el funcionamiento de funciones nativas habrá que hacer uso de un dispositivo Android emulado, tal y como se explica en el apartado siguiente.

Para poder hacer uso de Firebug, es necesario tener instalado previamente el navegador Firefox. Este navegador puede descargarse desde la URL *http://www.mozilla.org/es-ES/firefox/new/* para descargarlo en castellano.

El complemento Firebug podemos encontrarlo en la URL

https://addons.mozilla.org/es/firefox/addon/firebug/

Una vez instalados el navegador y el complemento, podemos acceder al mismo desde el menú Herramientas → Desarrollador Web → Firebug → Abrir Firebug

10.3- Creación de un dispositivo Android emulado:

Aunque una gran parte de la aplicación se puede probar en el navegador web, será muy útil disponer de un dispositivo Android emulado en el que probar el funcionamiento de características nativas y también de interfaz. Siguiendo los pasos del comienzo hemos instalado en el sistema operativo Eclipse con sus herramientas SDK y ADT. Ahora vamos a crear un dispositivo emulado.

Para ello, tenemos que pulsar el icono de la barra de herramientas "Android Virtual Device Manager". Al pulsar sobre este icono aparece una ventana con los dispositivos emulados si existieran. Para crear uno nuevo, pulsamos sobre "New".

A continuación aparece una nueva ventana en la que podremos definir las características de nuestro dispositivo emulado. Le asignamos un nombre, escogemos el dispositivo a emular (esto será importante para definir el tamaño de la pantalla), definimos su capacidad de memoria, el tamaño de su tarjeta SD... Una vez definidas las características de nuestro dispositivo, pulsamos OK.

Cuando durante el desarrollo de la aplicación queramos cargarla sobre uno de nuestros dispositivos emulados, tendremos que pulsar sobre el icono "Android Virtual Device Manager" y seleccionar el dispositivo que queremos inicializar. Para inicializarlo pulsamos "Start" y en la ventana que aparece seleccionamos "Launch". Dependiendo de las capacidades de nuestro ordenador el proceso de cargar el dispositivo emulado puede ser más o menos costoso, es normal si toma varios minutos en cargar.

Para cargar la aplicación en el dispositivo emulado pulsaríamos sobre "Main Activity" en la pestaña "Package" y pulsamos sobre el icono "Run". Si observamos la ventana del emulador veremos cómo se instala nuestra aplicación y cómo se inicializa en el dispositivo.

Esta herramienta es muy útil para comprobar el funcionamiento de la aplicación, pero, si la máquina no permite una ejecución rápida del dispositivo emulado, será especialmente interesante el uso del navegador web para ir probando la aplicación sobre todo en estadios iniciales o intermedios del desarrollo de la misma.

10.4- Actualización de la versión de Phonegap:

En un determinado momento del proyecto, al comenzar a desarrollar los procesos encargados tanto de leer ficheros .XML como de generarlos, se hizo necesario el uso de las funciones File de Phonegap. Tras algunos problemas en el uso de las funciones File, se optó por actualizar la versión de Phonegap a una más nueva. La versión escogida para su instalación fue la 2.8.1 por su forma de actualización. A continuación, se describen los pasos necesarios para realizar este procedimiento. Esta forma de proceder es válida hasta esta versión de Phonegap.

Este procedimiento sirve si disponemos de un proyecto creado con una versión más antigua de Phonegap. Recordemos que el núcleo de la aplicación es HTML, CSS y JavaScript por lo que tras algunas modificaciones en algunos ficheros generados por Eclipse al crear un proyecto nuevo, realmente se copiarán estos ficheros núcleo de la aplicación a su lugar correspondiente, con lo que habremos exportado la aplicación a una versión más nueva de Phonegap.

1- Accedemos a la dirección *www.phonegap.com/install*

2- Seleccionamos la versión de Phonegap a descargar, en este caso, 2.8.1

3- Nos habremos descargado un fichero comprimido dentro del cual nos encontramos con dos

archivos: cordova.js y cordova.jar
4- Creamos un proyecto Android con Eclipse, desde cero.
5- Creamos una carpeta a la que llamaremos “www” en el interior de la carpeta “assets” e incluimos dentro el fichero cordova.js
6- En la carpeta “libs” copiamos el fichero cordova.jar
7- En la carpeta raíz del proyecto, pulsando el botón derecho, seleccionamos “Properties” y seleccionamos “Java Build Path”.
8- En la pestaña “Libraries” pulsamos “Add Jars” y seleccionamos el fichero cordova.jar que hemos copiado en la carpeta “libs” en el paso 6.
9- Abrimos la carpeta “src” contenida en la carpeta raíz del proyecto y abrimos el archivo “Main Activity.java” en modo texto.
10- En las declaraciones import que aparecen en la parte superior, incluimos el siguiente import:
import org.apache.cordova.*
11- En el lugar donde se declara la clase “public class Main Activity extends Activity se elimina este último Acivity y se escribe DroidGap, quedando así:
public class MainActivity extends DroidGap
12- En el primer overide void onCreate (Bundle savedInsanceState) se cambia el protected por public, quedando así:
public void onCreate(Bundle savedInstanceState)
13- En ese mismo método, donde dice setContentView (R.layout.activity.main) se elimina dicha línea sustituyéndola por la siguiente:

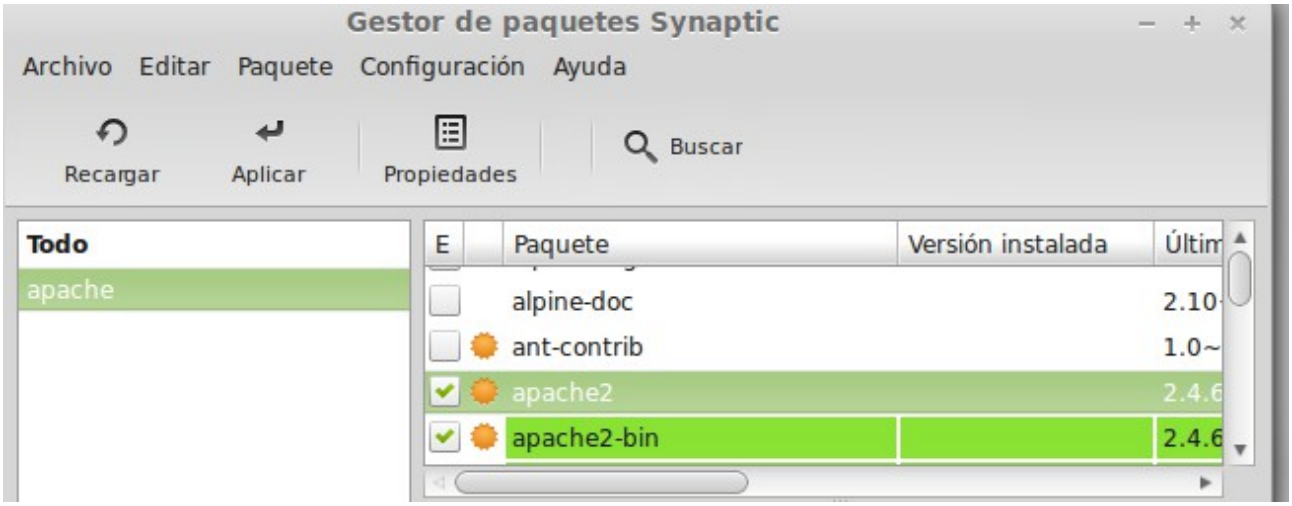
super.loadUrl("*file:///android_asset/*www*/index.html*");

En la carpeta www del proyecto antiguo, se borra el fichero phonegap 1.4.1 y se elimina también la carpeta “apis”. Esta carpeta www modificada se copia en el proyecto nuevo dentro de “assets”.
14- En el fichero que hemos descargado al comienzo, buscamos la carpeta phonegap/lib/android/xml. Copiamos esta carpeta en el proyecto nuevo que estamos modificando en Eclipse, en la carpeta “res”.
15- Por último, modificamos el Manifiest añadiendo los permisos asociados a la aplicación Android. Es probable que haya que modificarlos más adelante.

Al finalizar este proceso, habremos exportado el proyecto en Phonegap en versión más antigua al Phonegap actualizado. Hay que recordar que en aquellos HTML que vayan a hacer uso de funciones Phonegap habrá que incluir el nuevo fichero JavaScript del modo siguiente:
<script type="text/javascript" src="cordova.js"></script>

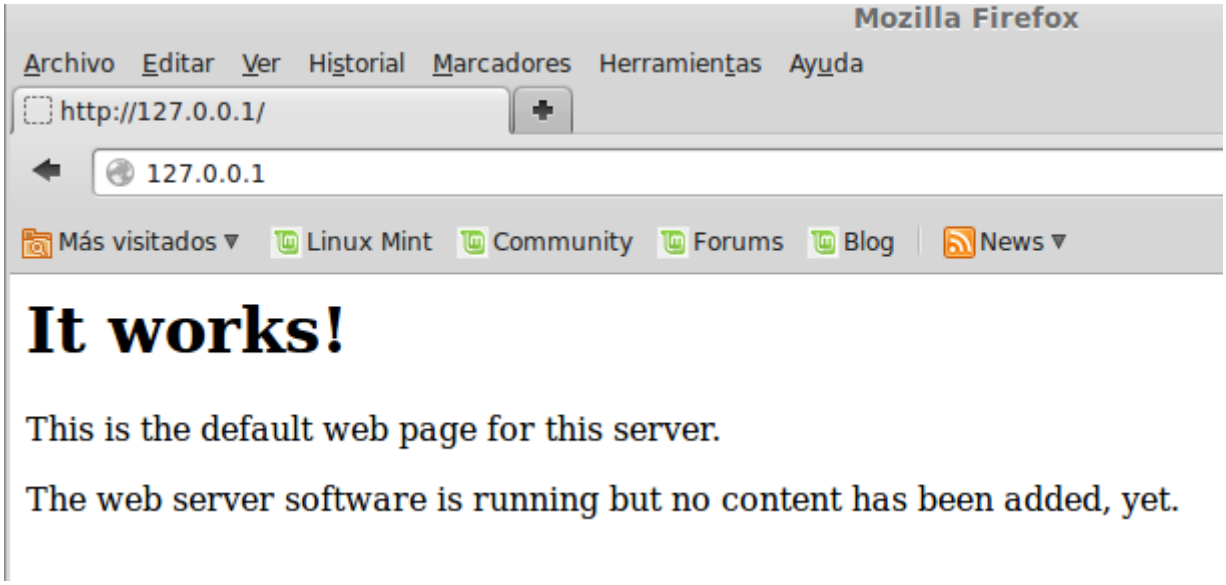
10.5- Instalación de servidor Apache:

Para la parte servidor de la aplicación instalaremos Apache. A continuación se detallan los pasos a seguir necesarios para la correcta instalación del servidor.
1- Abrimos el gestor de paquetes Synaptic en modo root.



2- Buscamos “Apache” y seleccionamos en los resultados “apache2”.
3- Seleccionamos “aplicar” y aceptamos los cambios.

Una vez se ha completado la instalación, vamos a verificar si se ha realizado correctamente. Abrimos el navegador y en la barra de direcciones tecleamos la IP del localhost, 127.0.0.1. En caso de estar funcionando correctamente, aparecerá en el navegador el mensaje “It works!”.



La carpeta en la que se encuentran los ficheros accesibles desde el servidor por defecto podemos encontrarla en /var/www. Esta es la carpeta de assets y todo lo que se coloque en ella será accesible desde el servidor.

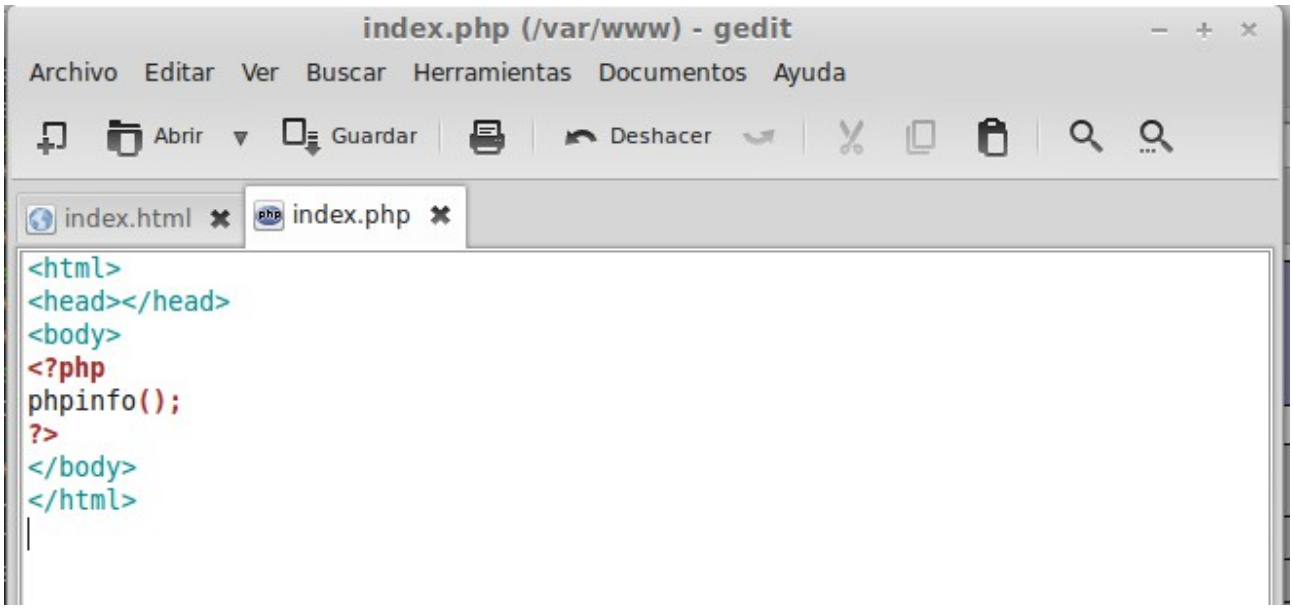
10.6- Instalación de PHP:

Será necesario instalar también PHP para poder procesar datos en el lado del servidor. A continuación se muestran los pasos seguidos para la instalación de PHP.
1- Abrimos el gestor de paquetes Synaptic en modo root.
2- Seleccionamos el paquete PHP5-mysql. Al seleccionar este paquete se seleccionarán también las

dependencias necesarias, con lo que estaremos instalando también los paquetes php5-common y php5. Si por cualquier motivo no se seleccionaran automáticamente estos paquetes, los seleccionamos.
3- Aplicamos y aceptamos los cambios.

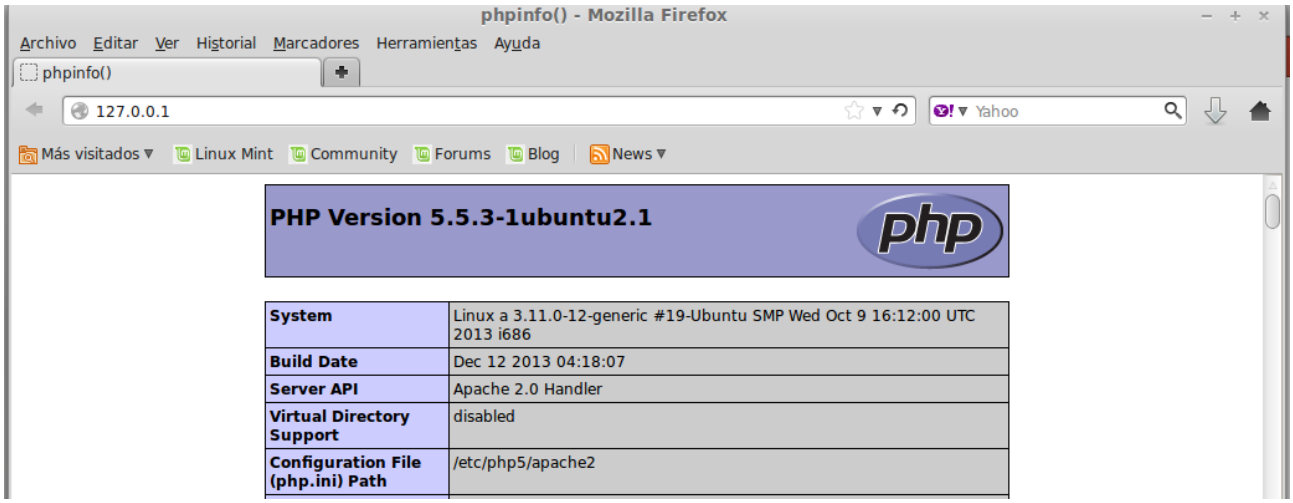
Una vez concluida la instalación podemos comprobar el correcto funcionamiento del siguiente modo.
1- Generamos un fichero .php en la carpeta de assets del servidor con el siguiente contenido (también podemos editar el fichero index.html que existe después de la instalación del servidor, sólo habrá que tener en cuenta que se debe cambiar la extensión del mismo por .php):

```
<html>
<head></head>
<body>
        <?php
                phpinfo();
        ?>
</body>
</html>
```



2- Abrimos el navegador y accedemos al fichero de la siguiente forma:
127.0.0.1/<nombre_del_fichero>.php

Si la instalación se ha realizado correctamente se mostrará en el navegador toda la información de los módulos php instalados.



Podemos comprobar a su vez que se ha instalado el módulo de mysql, necesario para poder utilizar la base de datos (en nuestro caso para realizar las consultas e inserciones en las tablas).

10.7- Instalación de phpmyadmin:

Para facilitar la gestión de la base de datos (creación de las tablas, visualización del contenido de las mismas), vamos a instalar la herramienta phpmyadmin. Esta herramienta nos ofrecerá una interfaz gráfica intuitiva para el manejo de la base de datos, evitando que tengamos que gestionarla directamente desde sentencias SQL.

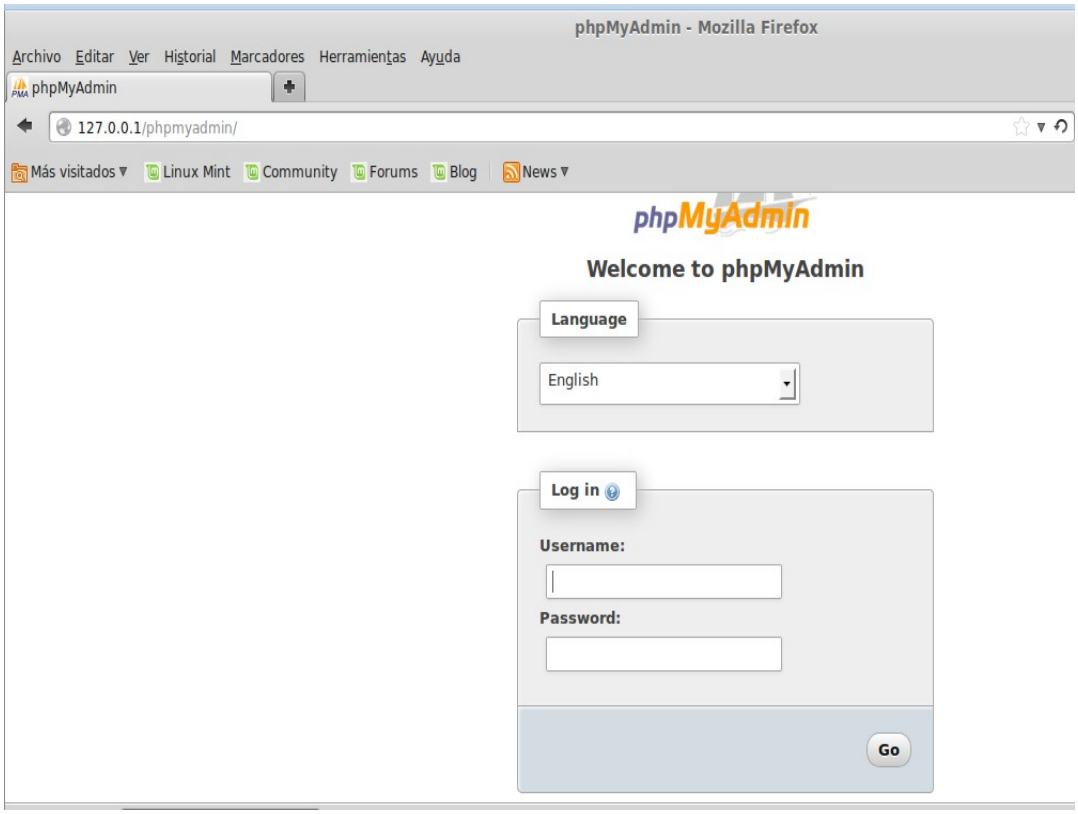
Antes de instalar phpmyadmin propiamente vamos a instalar desde Synaptic el paquete mysql-server necesario para el funcionamiento de phpmyadmin.

1- Abrimos el gestor de paquetes Synaptic en modo root.
2- Buscamos el paquete mysql-server, lo seleccionamos, aplicamos y aceptamos los cambios.
3- Nos pedirá una contraseña para la base de datos. Introducimos una, la confirmamos y continuamos la instalación.

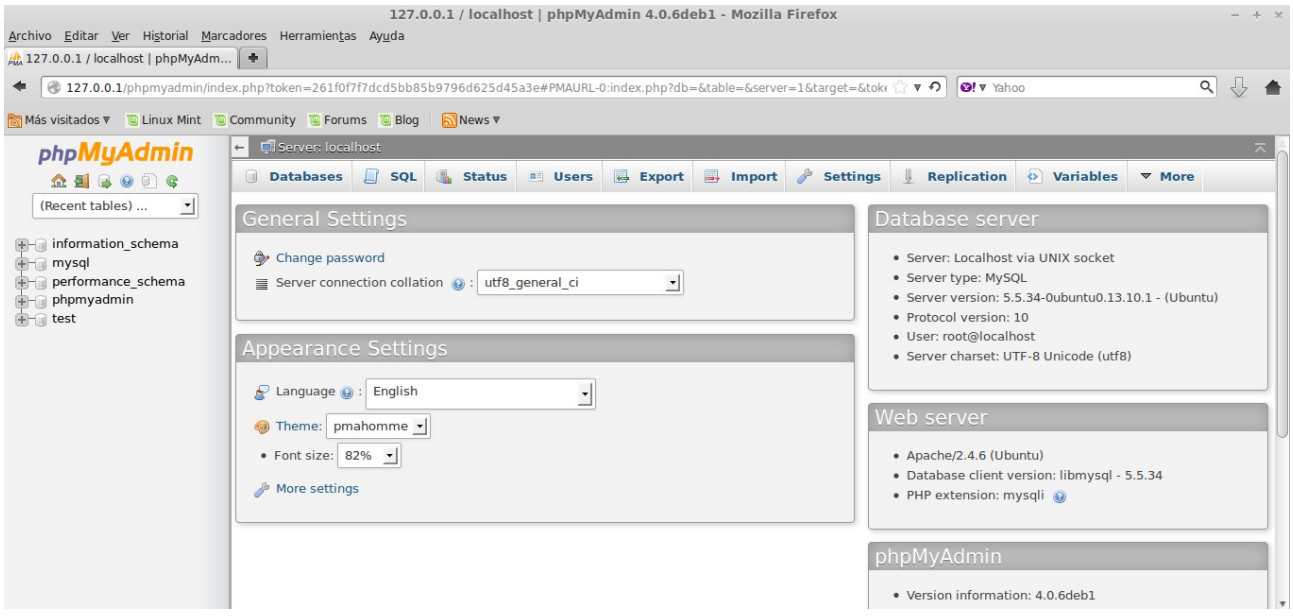
Una vez finalizada la instalación del paquete mysql-server ya podemos iniciar la instalación de phpmyadmin.

1- Abrimos el gestor de paquetes Synaptic.
2- Buscamos phpmyadmin, seleccionamos, aplicamos y aceptamos los cambios.
3- Aparece una ventana de configuración para seleccionar el servidor a actualizar. Seleccionamos el servidor Apache2 que habíamos instalado anteriormente y continuamos.
4- Nos pedirá la contraseña de la base de datos. Esta contraseña es la que definimos al instalar el paquete mysql-server. La escribimos y aceptamos.

Una vez finalizada la instalación podemos comprobar si se ha realizado correctamente abriendo el navegador y escribiendo en la barra de direcciones 127.0.0.1/phpmyadmin. Si la instalación ha sido correcta aparecerá una ventana en la que podemos introducir la contraseña.



Al introducirla aparecerá la ventana de administración de las bases de datos desde la que podremos crear la base de datos, tablas y ejecutar sentencias SQL.



10.8- Creación de repositorios en GitHub:

Para facilitar la gestión de cambios durante el desarrollo de la aplicación se ha utilizado la herramienta GitHub. Este sistema permite realizar un control de las versiones de los ficheros que componen la aplicación, además de dar alojamiento a los proyectos que resultan de su uso.

El contenido de los proyectos que se alojan en GitHub es público, por lo que cualquier persona puede visualizar y descargar los ficheros con el código fuente del proyecto, además de consultar el trabajo de los desarrolladores sobre los repositorios.

El repositorio que corresponde a este proyecto se encuentra en la dirección *https://github.com/baile/dinamico_phonegap* desde la que podemos descargar todos los ficheros que la componen y consultar los cambios que ha ido experimentando la aplicación durante el proceso.

A continuación se presenta una descripción de los pasos a seguir para la creación de un repositorio en GitHub:

1- Accedemos a la página *https://github.com/*
2- Seleccionamos la opción “Sign up for GitHub”
3- Seguimos los pasos necesarios para el registro (selección de nombre de usuario, contraseña, etc.)
4- Una vez hemos creado una nueva cuenta de usuario, accedemos a nuestra página y seleccionamos la opción “Repositories”.
5- Seleccionamos la opción “New”
6- Rellenamos la información de nuestro nuevo repositorio y finalizamos pulsando “Create repository”
7- Cuando hemos finalizado la creación de nuestro repositorio vemos una descripción de los comandos que podemos utilizar para añadir nuevos cambios al repositorio.

Varias personas pueden trabajar sobre un mismo repositorio. Esto ocurre si estamos trabajando en un equipo de desarrolladores. Para añadir desarrolladores al proyecto seleccionamos el repositorio y pulsamos “Collaborators”. Desde ahí gestionamos las personas que van a aportar contribuciones en el repositorio.