Adquisición de información y semántica en Internet de las Cosas: El problema de los objetos "no conectados"

Information acquisition and semantics in the Internet of Things: The "unconnected" objects issue

Josué Delgado Martín
Grado de Ingeniería Informática
Escuela Superior de Ingeniería y Tecnología

Trabajo de Fin de Grado

La Laguna, 3 de junio de 2015

D. José Ignacio Estévez Damas, con N.I.F. 43.786.097-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna.

C E R T I F I C A

Que la presente memoria titulada:

"Adquisición de información y semántica en Internet de las Cosas: el problema de los objetos no conectados"

ha sido realizada bajo su dirección por D. Josué Delgado Martín, con N.I.F. 78.648.301-P.

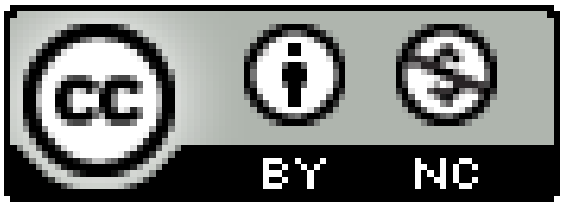
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 3 de junio de 2015.

Agradecimientos

Gracias a mi tutor José Ignacio Estévez Damas por guiarme en este proceso. Agradezco a mis compañeros y profesores de universidad los momentos compartidos, y por último, pero no menos importante, quiero agradecer el apoyo recibido por parte de mis amigos y familia, que son mi fuerza y motivación.

Licencia

Esta obra está bajo una licencia de © Creative Commons Reconocimiento-NoComercial 4.0 Internacional.



Resumen

En el presente proyecto, se pretende desarrollar una aplicación móvil dentro del marco que engloba el concepto de *"Internet de las Cosas"* (IoT). La idea principal de este concepto se basa en interconectar los objetos de uso cotidiano de manera que se puedan ampliar sus usos, así como la precisión y cantidad de información disponible sobre los mismos. Concretando un poco más, se pretender dotar de *"inteligencia"* a objetos cotidianos que carecen de ella, convirtiéndoles en *"Smart Item"*.

Se ha desarrollado una aplicación para *"Smartphone Android"* que permite al usuario actualizar fácilmente la información disponible sobre un objeto *"no inteligente"* (no dispone de recursos computacionales, más allá de una etiqueta electrónica) haciendo uso de la tecnología NFC (Near Field Communication), los sistemas de información distribuidos y el interfaz del propio dispositivo móvil. En concreto, se utilizarán etiquetas electrónicas NFC, siendo estas un subconjunto de las RFID, para dotar de *"inteligencia"* a los objetos cotidianos, de forma que un teléfono inteligente equipado con tecnología NFC es usado no solo para leer las etiquetas, sino también para aumentar la información con relaciones sobre el contexto actual del objeto mediante acciones simples realizados con el interfaz del móvil.

El resultado será la ya mencionada aplicación móvil con la que se interactuará con los objetos, captando o dotando de información a los mismos, además de un sistema de representación compacto de la información recibida. Esta aplicación móvil estará además conectada con un servidor donde se procesará la información, siendo esta susceptible de modificación mediante una sencilla interfaz web y una base de datos distribuida entre el servidor y Smartphone.

Palabras clave: Internet de las Cosas (IoT), NFC, RFID, antena NFC, Android, Smartphone, semántica de comunicación, servidor, bases de datos distribuidas.

Abstract

In this project, we want to develop a smartphone app to approach several issues related to the *"Internet of Things"* (IoT). IoT is based on connecting everyday objects to make them increase their possible usages and the amount and accuracy of available information about them. Specifying a bit more, this project approaches the problem of providing *"intelligence"* to everyday objects, even when they are not connected to a network, to make them be like *"Smart Items"*.

We developed a *"Smartphone Android"* app that allows the user to easily update information from a *"not smart"* object (it hasn't got computational resources except an electronic tag) using NFC (Near Field Communication) technology, distributed information systems and the features provided by the smartphone interface. In fact, NFC electronic tags (which are a RFID subgroup) are used to provide *"intelligence"* to everyday objects, in such a way that a smartphone equipped with NFC technology is used not only to read the tags but also to augment this information with relations about the present context of the object through simple actions carried out with the mobile.

The final result is the mentioned smartphone app, that will interact with the objects catching or providing information to them, In addition, we have developed a compact representation system to store the acquired information. This smartphone app is connected also to a server where the information is processed and modified using a simple web interface and a distributed database between the server and the smartphone.

Keywords: Internet of Things (IoT), NFC, RFID, NFC antenna, Android, Smartphone, communication's semantic, server, distributed database.

Índice general

Contenido

Índice de figuras

Figuras Apéndice A

Figuras Apéndice B

Índice de tablas

1. INTRODUCCIÓN AL PROYECTO

El mundo está cada vez más interconectado. Tener un teléfono móvil (Smartphone) con conexión a Internet es algo típico en la actualidad, cosa que hace apenas cinco años no era ni mucho menos común de ver. La progresión en este aspecto sigue al alza, y cada vez, existen nuevos objetos *"inteligentes"* que se interconectan entre sí o a internet. Un claro ejemplo sobre esto, podrían ser los objetos cotidianos, cuyo uso en principio distaba del que actualmente se le pueden dar, y que ahora están complementados con la capacidad de estar conectados. Esta línea de desarrollo está englobada dentro del concepto denominado *"Internet de las Cosas (IoT)"*.

El Internet de las Cosas se refiere a la interconexión de todos los objetos cotidianos con la red. Este concepto fue propuesto por Kevin Ashton en el Auto-ID del MIT donde se hacían estudios sobre la identificación por radio frecuencia (RFID). Actualmente, existen muchos objetos cotidianos a los cuales se les ha dotado de algún tipo de conectividad a la red. Se estima que para 2020 existirán más de 30 mil millones de dispositivos con un sistema adaptado al Internet de las Cosas. Si todas las cosas estuvieran conectadas a internet y dispusieran de algún sistema de sensado, se contaría con información en tiempo real de cada objeto, siendo esto de especial valor. A continuación se listará algunas áreas donde se puede aplicar Internet de las Cosas: **[1-3]**

- **En el hogar**: Internet de las Cosas ofrece una oportunidad inmejorable para obtener y analizar datos estadísticos sobre el comportamiento humano, siendo esto de gran interés, por ejemplo, en el marketing.

- **Monitorización ambiental**: Control del aire, agua, atmósferas, etcétera. Internet de las Cosas permite recabar más información sobre el medio y protegerlo de manera más efectiva.

- **Gestión de infraestructuras**: Control del estado de infraestructuras como puentes, carreteras, vías férreas, etcétera, permitiendo un mejor control de la conservación de los mismos.

- **Gestión de energía**: Monitorizando el consumo de energía y realizando ajustes para mejorar la eficacia.

Estos son solo algunos de los posibles campos de actuación sobre los cuales se puede trabajar con Internet de las Cosas.

Otro punto de vista diferente, y sobre el que se basa este proyecto, sería el de dotar a cada objeto cotidiano, siendo estos, aquellos objetos que no fueron diseñados para IoT, de algún tipo de identificación, además de información, de forma que el usuario pueda interactuar con cada objeto que le rodee sin que este tenga que ser necesariamente *"inteligente"*. En este aspecto, la tecnología NFC/RFID es la más adecuada debido a que mediante ella se pueden relacionar objetos físicos con dispositivos inteligentes de uso común, como los Smartphones. Normalmente, el objeto físico tendrá una etiqueta electrónica (etiquetas NFC) con información y el Smartphone tendría el papel de obtener y procesar dicha información. **[3-5]**

En cuanto a RFID y NFC tenemos:

- RFID es un sistema de identificación basado en una memoria electrónica y transmisión por radiofrecuencia ampliamente utilizado. Se le considerada el sucesor del código de barras en cuanto a logística se refiere.

- NFC es una tecnología de comunicación basado en RFID.

En el capítulo *Fundamentos Tecnológicos*, se profundizará en el funcionamiento y uso del RFID y NFC.

Resumiendo, con Internet de las Cosas, se pretende crear un mundo donde todo esté conectado a internet o sea susceptible de interactuar con el humano como Smart Item.

Existen multitud de aplicaciones para Smartphone mediante las cuales se puede leer o escribir en etiquetas NFC. Sin embargo, a la hora de leer estas etiquetas con el objetivo de alcanzar un grado de interacción mayor, el campo de aplicaciones existentes es bastante escueto. Con este proyecto se ha pretendido experimentar con nuevas aplicaciones de la tecnología NFC, orientadas a la captura de información y semántica de los objetos mediante la especificación de relaciones entre varios Smart Items.

1.1. ANTECENDENTES

1.1.1. INTERNET DE LAS COSAS

Si de por si el campo de la informática es de origen reciente en cuanto al desarrollo de conocimiento se refiere, el denominado **Internet de las Cosas** o *Internet of Things* (IoT) apenas supera la década de vida desde que se nombró por primera vez por Bill Joy en 1999. **[12]**

- En 1990 Jhon Romkey y Simon Hacket desarrollaron el primer objeto con conexión a internet conocido: una tostadora. Dicha tostadora se podía encender y apagar desde internet.

- A pesar de que Jhon Romkey y Simon Hacket crearon el primer objeto conectado a internet, no fue hasta 1999 cuando el ingeniero Bill Joy sentara las bases de Internet de las Cosas al exponer los beneficios de tener dispositivos conectados a internet.

- Diez años después, Kevin Ashton acuñó el término Internet de las Cosas con su artículo en RFID Journal el 12 de julio de 2009. En este artículo se exponía la idea de conectar todos los objetos que nos rodean a internet con el objetivo de poder obtener mayor información de ellos en cualquier momento.

- Según la empresa Cisco, en torno a los años 2008-2009 nace Internet de las Cosas debido a que el número de dispositivos electrónicos superó al número total de habitantes en la tierra.

- En 2011, se presentó el protocolo IPv6 con el que se pretende, entre otras cosas, identificar de forma única cada objeto.

- Las previsiones apuntan que para el año 2020 existan entre 30 mil y 50 mil millones de dispositivos conectados a internet.

1.1.2. IDENTIFICACIÓN POR RADIOFRECUENCIA

La **Identificación por radiofrecuencia** *o Radio Frequency Identification* (RFID) es una tecnología que genera campos magnéticos con el objetivo de enviar datos en modo inalámbrico. Esta tecnología data de 1940, y se ha ido adaptando hasta llegar a los tiempos actuales. Su usabilidad es alta y por ello está presente en gran diversidad de campos; pasando desde teléfonos móviles, emisoras de radio, instrumental quirúrgico, etcétera. **[13]**

- Existen antecedentes de uso desde 1920, pero no fue hasta 1940 que se documentó su uso para detectar aviones enemigos y aliados durante la Segunda Guerra Mundial.

- En la década de 1950 a 1960, las empresas de los países avanzados empezaron a desarrollar tecnología para distinguir si un objeto había sido pagado o no dentro de un comercio. Este fue el comienzo de los sistemas antirrobo.

- A partir de 1970 su uso se extendió y se generalizó. Pasó a usarse en variedad de ámbitos, desde identificar a ganado insertando etiquetas RFID bajo la piel de los animales, hasta en las puertas de las centrales nucleares, abriéndose o cerrándose al pasar algunos tipos de vehículos.

1.1.3. COMUNICACIÓN DE CAMPO CERCANO

La **Comunicación de campo cercano** o *Near Field Communication* (NFC) surgió a partir de la tecnología RFID, siendo por tanto una tecnología de comunicación inalámbrica. Su principal diferencia es que fue concebida principalmente para su uso en dispositivos móviles como sistema de identificación y comunicación de corto alcance (en torno a los 14milímetros). **[14]**

- NFC comenzó a desarrollarse en 2002 cuando Philips y Sony decidieron crear un protocolo compatible con las tecnologías existentes en ese momento. Concretamente surgieron los sistemas Mifare de Philips y FeliCa de Sony.

- El protocolo NFC fue aprobado en 2003 como estándar ISO 18092.

- En 2004 Philips, Sony y Nokia crearon el NFC fórum, consiguiendo que empresas como Google, Paypal o Visa participaran en ella.

- Actualmente NFC es compatible con el estándar ISO/IEC-14443(RFID) para tarjetas de proximidad sin contactos, lo que le hace compatible con todas las infraestructuras de pagos sin contactos existentes.

1.2. OBJETIVO GENERAL DEL PROYECTO

Desarrollar una aplicación para Smartphone dentro del marco de Internet de las Cosas. Concretamente, se pretende desarrollar una aplicación para el sistema operativo Android mediante la cual se capte información de objetos de uso cotidiano, creando una imagen virtual de ellos almacenada en una base de datos distribuida. Cada objeto tendrá una etiqueta NFC asociada, lo cual permitirá diferentes tipos de interacciones con dichos objeto. Dicha etiqueta servirá para facilitar la actualización de la imagen virtual del objeto mediante la aplicación desarrollada.

1.3. OBJETIVOS ESPECÍFICOS

Con el desarrollo de este proyecto se pretenden conseguir los siguientes objetivos:

- Crear una aplicación prototipo para Smartphone en Android empleando tecnología RFID/NFC.

- Analizar y seleccionar los casos de uso existentes dentro del marco ofrecido por la aplicación.

- Realizar un proceso de validación de la aplicación sobre los casos de uso propuestos.

- Creación de documentación que de soporte a la aplicación, tanto de cara al usuario final como a un posible programador que trabaje sobre futuras versiones de la misma.

Especificando un poco más, se pretende crear una aplicación prototipo para Smartphone en Android que puede leer o escribir texto en etiquetas electrónicas. Concretamente, se utilizarán etiquetas NFC que están basadas en el estándar RFID.

La aplicación tendrá que poder escribir información en etiquetas NFC asociadas a objetos. Posteriormente, y tras asociar información relevante a cada objeto, se podrán realizar diferentes tipos de interacciones con los objetos, creando relaciones entre sí, y por tanto, dando mayor información y usabilidad a los mismos. Por tanto, el núcleo de la aplicación se basará en la captura de información mediante el dispositivo móvil en base a las diferentes interacciones creadas. Este proceso se realiza con la mayor sencillez posible, debido a la simplicidad de la interfaz desarrollada

y la sencillez de los pasos a seguir por el usuario. Simplemente se necesita seleccionar el tipo de interacción a realizar, y posteriormente, tocar con el dispositivo móvil las etiquetas NFC asociadas a objetos. Si el dispositivo móvil está en posición portrait, se realizará una inserción de información en la base de datos en consonancia con el tipo de interacción seleccionada. Destacar, que al leer las diferentes etiquetas NFC, lo que se estará realizando es una inserción de la imagen virtual del objeto real, junto con información complementaria obtenida en el proceso de captura de información con el tipo de interacción seleccionada.

Por otro lado, también se podrá obtener toda la información almacenada sobre un objeto. Para ello, simplemente se acercará el dispositivo móvil en posición landscape a la etiqueta NFC del objeto, mostrando toda la información asociada al mismo.

En cuanto a la información captada, estará almacenada en una base de datos distribuida compuesta por dos nodos diferenciados: La base de datos del dispositivo móvil y la base de datos de un servidor. Desde dicho servidor se podrá visualizar, añadir y modificar la información a través de una sencilla interfaz web desarrollada para el mismo, replicando los cambios en la base de datos del dispositivo móvil.

En el capítulo de esta memoria *Casos de uso*, se detallarán diferentes casos prácticos de uso con el objetivo de que un usuario pueda comprender la utilidad de la aplicación. Además, en el *Apéndice A: Manual del usuario*, se explica la interfaz tanto del dispositivo móvil como del servidor, facilitando la comprensión de la misma.

1.4. PLANIFICACIÓN

A continuación se muestra una tabla con la planificación del proyecto.

Semana	Actividades	Puntos
Fase 1. Investigación		**2**
10/01 -14/01	Definición de casos de uso	0.5
15/12-23/01	Revisión de la bibliografía	0.5
23/12-14/02	Elección de tecnologías a usar en el desarrollo	1
Fase 2. Desarrollo del prototipo		**3.5**
15/02-20/02	Estudio de software existente de captura de información	0.5
21/02-28/02	Diseño de la aplicación	1
01/03-29/03	Desarrollo de la aplicación prototipo	2
Fase 3. Verificación del software		**3**
30/03-10/04	Verificación de los diferentes subsistemas de la aplicación	2
11/04-20/04	Verificación de la aplicación en diferentes arquitecturas	1
Fase 4. Documentación		**2.5**
21/04-01/05	Documentación del programador	0.5
02/05-10/05	Documentación del usuario	0.5
11/05-30/05	Memoria del proyecto	1
Total		**10**

Tabla 1.1 *Planificación para el desarrollo del proyecto*

2. DISEÑO Y METODOLOGÍA

2.1. METODOLOGÍA

La metodología que mejor se adapta a la llevada en el desarrollo del proyecto, es la metodología XP.

La **programación extrema** *o eXtreme Programming* (XP) es una metodología de desarrollo ágil de la ingeniería del software formulada por Kent Beck y destinada a dotar de capacidad de adaptación a los cambios en los requisitos del software a desarrollar. Algunas características de esta metodología que se han puesto en práctica en este proyecto han sido: **[15]**

- Desarrollo iterativo e incremental.
- Objetivos de realización a corto plazo y por lo tanto entregas pequeñas.
- Simplicidad en el diseño.
- Pruebas unitarias continuas.

A continuación se describirán los pasos seguidos en la ejecución del proyecto, siguiendo la línea de la metodología ágil XP.

1. Los objetivos semanales a conseguir se iban colocando en un portfolio semanal compuesto por 4 columnas. Dichas columnas marcaban las tareas hechas (done), las tareas que se estaban realizando hasta ese momento (doing), las tareas que se estaban comprobando (testing) y las tareas que aún estaban por realizar (to do). Para la realización y visualización de dicho porfolio se utilizó la herramienta Trello. En la *Figura 2.1* se muestra un ejemplo de lo anterior.

2. Cada semana se iban añadiendo un número limitado de objetivos a conseguir. Durante el desarrollo del proyecto el límite estaba en 3 objetivos por semana, ampliable a 4 si se cumplían los objetivos antes de lo previsto.

3. Las tareas van fluyendo a medida que se van realizando desde la columna *"to do"* hasta la columna *"done"*. Cada vez que una tarea se empieza a desarrollar, dicha tarea pasa a la columna *"doing"*. Tras finalizar la tarea, esta pasa a la columna *"testing"* donde se realizarán las pruebas necesarias que verifiquen el funcionamiento o consecución del objetivo desarrollado. Si, la tarea pasas las pruebas, finalmente se moverá a la columna *"done"*. En cambio, si alguna de las pruebas fallas, la tarea volverá a la columna *"doing"*.

4. Cada día se elegirá el objetivo a conseguir de las diferentes tareas existentes en la columna *"to do"* o se seguirá con el desarrollo o prueba de algunas de las tareas que puedan existir en las columnas *"doing"* o *"testing"*.

5. Al finalizar cada semana, se realizaba una evaluación del estado del proyecto.

6. Por otro lado, cada 15 ó 20 días se tenía una reunión con el director del proyecto para informar del estado del mismo desde la última reunión realizada. El director tendría el rol de cliente dentro de la metodología ágil XP, añadiendo o modificando objetivos a medida que se iba realizando el proyecto.

7. Destacar que, ante todo, se busca la simplicidad a la hora de desarrollar. Para ello, ante de comenzar a programar se realizaba unos esquemas de los pasos a seguir y la estructura básica del código a realizar. De esta manera se conseguía tener unas ideas claras y bien definidas de los pasos a seguir, ahorrando tiempo a posteriori y evitando posibles problemas por una mala gestión y consecución del objetivo.

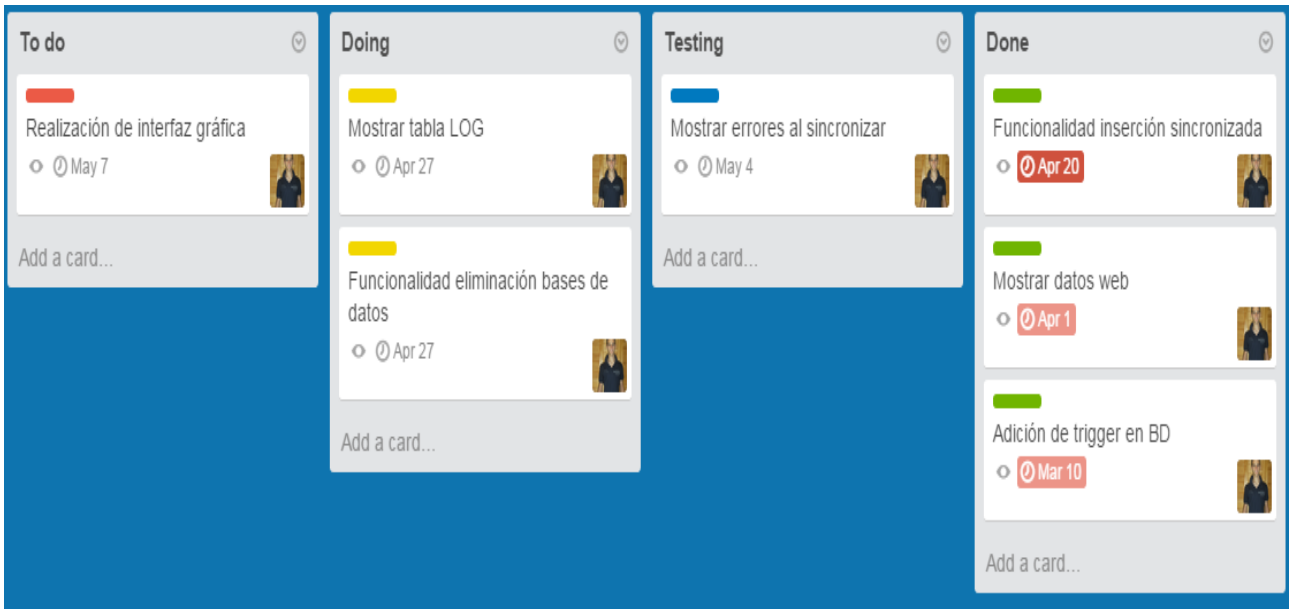


Figura 2.1 *Ejemplo de Portfolio con la herramienta Trello*

2.2. ENTORNOS DE TRABAJO

En este apartado se nombrarán las herramientas utilizadas en el desarrollo del proyecto. Es importante utilizar las últimas versiones de las mismas y que sean compatibles entre ellas. Además, se debe buscar las herramientas con las que el programador se sienta más cómodo, dado que en muchas ocasiones existen diferentes opciones de herramientas que cumplen el mismo propósito.

- Java JDK 1.8.0
- SDK Android
- MySQL
- SQLite
- Android Studio
- Eclipse ADT
- MySQL WorkBench
- SQLite Admin
- Trello
- GitHub
- XAMP
- Brackets
- Genymotion
- VirtualBox
- PHP MyAdmin

3. FUNDAMENTOS TECNOLÓGICOS

3.1. IDENTIFICACIÓN POR RADIO FRECUENCIA (RFID)

La **Identificación por Radio Frecuencia o** *Radio Frequency IDentification* **(**RFID) es una tecnología de comunicación inalámbrica de intercambio de datos entre un lector RFID y una etiqueta RFID**.** Como se puede intuir al ser una tecnología de comunicación inalámbrica, se basa en las ondas de radio, siendo capaz de intercambiar información entre el lector y la etiqueta a metros de distancia. Un sistema RFID se compone de dos partes principales, el transpondedor y el lector. **[3,6]**

El **transpondedor**, cuya función se basa en la recepción de señales y se encuentra en el objeto a ser identificado, normalmente etiquetas RFID. Estas etiquetas están compuestas por circuitos integrados mediante los cuales se trasmiten los datos almacenados en ellas. Además, tienen una gran capacidad de almacenamiento de datos. Están divididas en dos grupos: **[3,11]**

- Etiquetas pasivas. No precisan de fuente de alimentación. La propia señal que les llega de los lectores es suficiente para activar los circuitos y recuperar los datos a leer. Suelen tener menor rango de distancia de operación y pueden permanecer activas duramente menos tiempo que las etiquetas activas debido a la escasa energía captada durante la lectura. Por otro lado, el tamaño de estas etiquetas puede ser muy reducido, llegando a milímetros de superficie. **[3,11]**

- Etiquetas activas. Tienen su propia fuente de alimentación, lo que las hace ideales para trasmitir datos a mayores distancias, además de tener menor tasa de errores de lectura y por lo tanto mayor fiabilidad que las etiquetas pasivas. Por otro lado, su tamaño mínimo es mucho mayor que el ofrecido por las etiquetas pasivas, en torno a los centímetros de superficie. **[3,11]**

El **lector** está compuesto por un transceptor con un decodificador para la interpretación de datos, una unidad de control para la toma de decisiones y una antena para la recepción y envío de las ondas de radio. Adicionalmente, algunos lectores tienen algún tipo de interfaz adicional para el envío de datos a otros sistemas. **[3,11]**

A continuación mostrará una representación de un sistema RFID.

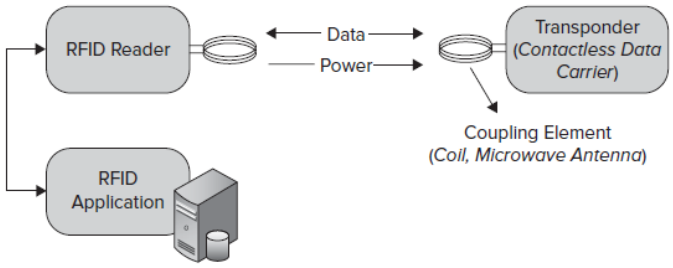


***Figura 3.1** Sistema RFID* **[3]**

3.2. COMUNICACIÓN DE CAMPO CERCANO (NFC)

La **Comunicación de Campo Cercano** o *Near Field Communication* (NFC) es una tecnología de comunicación inalámbrica a corta distancia para el intercambio de datos. Está basada en el estándar ISO 14443 (RFID) **[16]**. Cómo en el caso de los sistemas RFID, los sistemas NFC se comunican mediante ondas de radio, aunque estas son de menor alcance, en torno a los 14 centímetros de distancia como máximo. Además, trabaja en una banda de 13,56 MHz, lo que provoca que no se necesite de ningún tipo de licencia para su uso. Dependiendo de los tipos de interacciones realizadas, los sistemas NFC tienen dos modos de trabajar diferentes: **[3,4,5]**

- Modo activo. Ambos dispositivos generan sus propios campos electromagnéticos. **[4]**
- Modo pasivo. Solo uno de los elementos genera campo electromagnético, mientras el otro utiliza ese campo para suministrarse de energía y enviar o recibir los datos del elemento generador. **[4]**

En cuanto a los tipos de comunicación, existen esencialmente tres tipos:

- Modo emulación de tarjeta o *Card emulation mode.* Este modo es utilizado cuando el dispositivo NFC funciona como una tarjeta. Un ejemplo de este modo sería el pago mediante un dispositivo móvil con NFC. **[3,4,5]**
- Modo lectura/escritura o *Reader/Writer mode.* Este modo se utiliza para modificar, almacenar o leer datos de un dispositivo pasivo NFC, generalmente etiquetas NFC. En el caso de la lectura, el dispositivo que recibe la información podría estar configurado para realizar una acción u otra en consonancia con el tipo de información captada. Por ejemplo, se lee una URL y se abre un navegador web con la página web cargada. **[3,4,5]**

- Modo punto a punto o *Peer to Peer mode.* Este modo es específico de los sistemas NFC. Se establece un vínculo entre dos dispositivos activos NFC, generalmente teléfonos móviles (Smartphone), y se comienza el intercambio de información sin límite de tamaño. **[3,4,5]**

Un esquema de lo anterior se muestra a continuación en la siguiente figura.

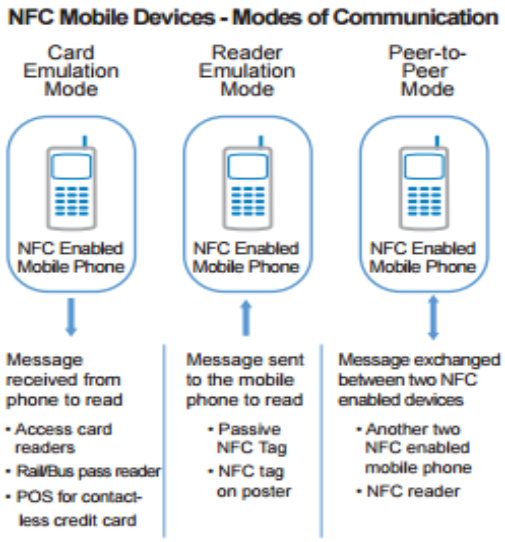


Figura 3.2 *Modos de comunicación NFC* **[5]**

Por último, los tipos de elementos posibles en un sistema NFC son:

- Teléfonos móviles/Smartphones con NFC. Son actualmente los dispositivos NFC más importantes. Mediante ellos se pueden aprovechar las diferentes opciones que ofrecen los ecosistemas NFC, además de una rápida adaptación y difusión entre usuarios debido a la facilidad de uso que ofrecen. Normalmente actúan como un elemento activo dentro del sistema NFC. **[3,4,5]**

- Lectores NFC. Son dispositivos que tienen la habilidad de captar información de otros dispositivos NFC y enviarla a otros elementos para su procesamiento. Actúan e modo activo dentro del sistema NFC. **[3,4,5]**

- Etiqueta NFC. Una etiqueta NFC no es más que una etiqueta RFID sin fuente de alimentación integrada. Actúan en modo pasivo dentro de un sistema NFC. **[3,4,5]**

Uno de las principales ventajas de estos sistemas es la facilidad de uso. Apenas realizando una acción tan intuitiva como tocar un dispositivo NFC, comienza la comunicación. El iniciador de la comunicación siempre será un elemento que funcione en modo activo, como un teléfono móvil. El receptor sería un elemento que funcione activa o pasivamente indistintamente, como otro móvil o etiqueta NFC. **[3,4,5]**

3.2.1. NFC EN MÓVILES

Un dispositivo móvil que cuenta con tecnología NFC se compone típicamente de varios circuitos integrados, **elementos de seguridad** (SE), **interfaz de control NFC** (NFC-C) e **interfaz entre el SE y NFC-C.**

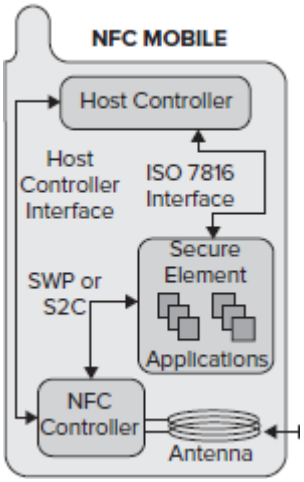


Figura 3.3 *Estructura NFC móvil* **[3]**

Los **elementos de seguridad** son importantes para proveer al usuario de un entorno seguro de uso en actividades tan importantes como el pago mediante un dispositivo NFC. La idea es que todas las aplicaciones que necesiten de un almacenamiento seguro de datos se ejecuten en la memoria del SE. Estos elementos están compuestos por protocolos, software y hardware que proveen de todo lo necesario para que el almacenamiento de los datos se haga de forma segura. Además, es necesario el uso de algún sistema operativo para funcionar (MULTOS o JAVACARD). Existen varias alternativas para estos elementos de protección. Se dividen en elementos de seguridad extraíbles y no extraíbles. Algunas de ellos son: **[3,4,5]**

- SE mediante hardware integrado. Provee de seguridad máxima. Se añade al dispositivo NFC una etiqueta inteligente que no puede ser eliminada. Dicha tarjeta es personalizada de forma que solo el usuario final pueda utilizarla. Como contrapartida, cada vez que se cambie de usuario habría que integrar una nueva tarjeta inteligente debido a que no es transferible ni reutilizable. Este SE es no extraíble. **[3,4,5]**

- SE con tarjeta de memoria segura (SMC). Este SE está compuesto por una memoria integrada y un controlador de tarjetas inteligente. Ofrece un alto grado de seguridad y gran capacidad. Por otro lado, es un SE extraíble y por lo tanto se puede usar en diferentes dispositivos NFC. **[3,4,5]**

- SE con tarjetas universales de circuitos integrados (UICC). Funciona de igual manera que el anterior SE. Se diferencia en que la memoria estará ahora en la tarjeta SIM de usuario. Este SE es extraíble. **[3,4,5]**

La **interfaz de control NFC** está compuesta por un contacto analógico/digital Front-End, una antena NFC y un circuito integrado controlador para permitir transacciones NFC. El circuito controlador se encarga de modular/demodular la señal analógica entre la antena NFC y la señal de radio frecuencia. El controlador es compatible tanto con comunicación activa como pasiva y los modos de comunicación peer-to-peer y lectura/escritura. Además, también suelen ser compatibles con los estándares RFID ISO / IEC 15693. **[3,4,5,17]**

Para la **interfaz entre el SE y NFC-C** se dispone de dos opciones en función del SE elegido. Son las siguientes:

- Si el SE es integrado, se suele utilizar una interfaz bajo el protocolo NFC-WI. NFC-WI es un interfaz de cableado digital estandarizado. En esta solución se tiene dos cables conectados al SE. Un cable de entrada y otro de salida. Ambos cables trasmiten señales binarias entre el SE y el NFC-C. **[3,4,5]**

- Si el SE es extraíble, se utiliza una interfaz bajo el protocolo SWP. Es un protocolo digital de cableado full-dúplex, y por lo tanto, solo se tiene un cable entre el SE y el NFC-C. Su funcionamiento se basa en el principio maestro esclavo, siendo el maestro el controlador NFC. **[3,4,5]**

3.2.2. TIPOS DE ETIQUETAS NFC

Las **etiquetas NFC** están divididas en cuatro tipos en función de una serie de características que las diferencia entre sí. Son los siguientes:

- Etiquetas de tipo 1. Las etiquetas de este tipo están basada en el estándar ISO 14443A. Son de lectura o escritura y opcionalmente se pueden bloquear para que solo se puedan leer. El tamaño de almacenamiento es de 96 bytes ampliable a 2 Kbyte. Velocidad de transmisión es de 106 Kbit/segundo. **[4]**

- Etiquetas de tipo 2. Este tipo de etiquetas son prácticamente iguales a las anteriores, menos en el tamaño de almacenamiento básico. Su tamaño estándar es de 48 bytes. **[4]**

- Etiquetas de tipo 3. Las etiquetas de este tipo están bajo la norma industrial Japonesa X 6319-4, comúnmente conocido como FeliCa. Estas etiquetas son pre-configuradas en su fabricación para ser de lectura - escritura o sólo de lectura. El tamaño de almacenamiento es variables, siendo el límite 1 Mbyte por servicio (operación de escritura). La velocidad de transmisión también puede variar siendo entre 212 kbit/s a 424 Kbit/segundo. **[4]**

- Etiquetas de tipo 4. Este tipo de etiquetas tiene la particularidad de ser compatibles con el estándar ISO 14443A e ISO 14443B. La diferencia entre ambos estándares radica en el método de modulación e inicialización de las transacciones. El tamaño de almacenamiento es variable hasta un límite de 32 Kbyte por servicio (operación de escritura). La velocidad de transmisión es de 424 Kbit/segundo. **[4]**

3.2.3. FORMATO DE INTERCAMBIO DE DATOS NFC (NDEF)

El Formato de Intercambio de Datos NFC o *NFC Data Exchange Format* (NDEF) es el formato de intercambio de datos entre dos dispositivos del sistema NFC. Estos intercambios de datos entre dispositivos se pueden dar entre un elemento activo (teléfono móvil) y otro pasivo (etiqueta NFC) o entre dos elementos activos. El intercambio de datos comienza al acercar dos dispositivos del sistema NFC. **[6]**

NDEF es un formato de mensaje binario que encapsula una o más cargas útiles definidas por la aplicación en un único mensaje. Cada mensaje tiene uno o más registros, los cuales albergan los datos útiles con un tamaño máximo de 2^32 -1 octetos, existiendo la opción de encadenar registros si fuera necesario por necesidades de tamaño. La estructura de un mensaje sería la siguiente: **[6]**

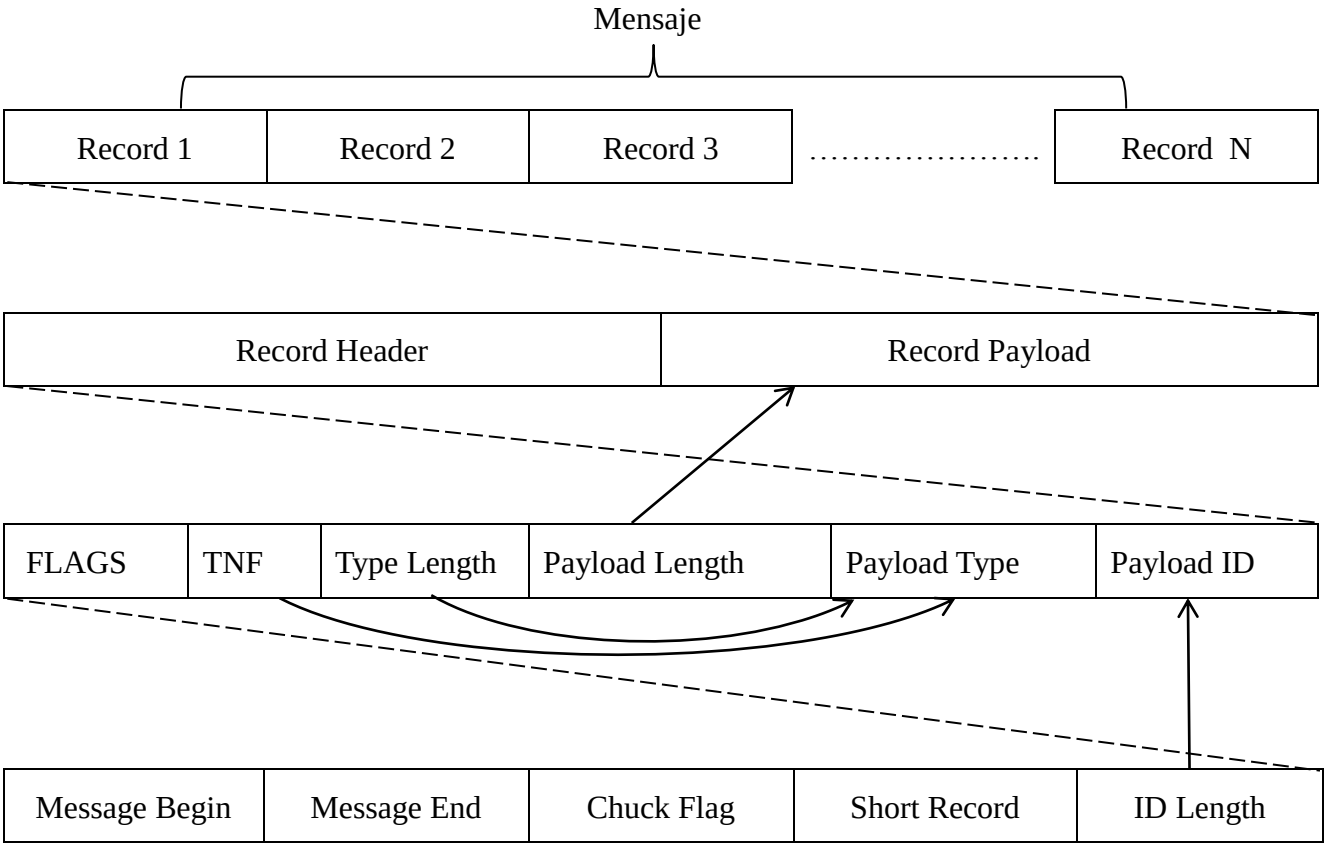


Tabla 3.1 *Estructura de un mensaje NDEF*

Se puede observar, la estructura de un mensaje NDEF es a priori compleja. A continuación se describirán cada una de las partes señaladas en la *tabla 3.1*.

Como ya se comentó con anterioridad, un mensaje está compuesto por uno o más a registros. Cada registro contendrá la carga útil de datos además de información de control necesaria. Esto es representado en la tabla anterior de forma que cada registro tiene un *"Recorder Payload"* con la carga útil y un *"Recorder Header"* con la información de control. La información de control está compuesta por:

- FLAGS. Los flags como su propio nombre indica, son avisos que se activan sobre algunos tipos de registros. Son los siguientes: **[6]**
    - Message Begin. Indica el registro inicial del mensaje NDEF. Es un campo de 1 bit de tamaño.
    - Message End. Indica el registro final del mensaje NDEF. Es un campo de 1 bit de tamaño.
    - Chuck Flag. Indica si la carga útil está fragmentada en diferentes registros. Es un campo de 1 bit de tamaño.
    - Short Record. Indica si la carga útil consta de un solo octeto. Es un campo de 1 bit de tamaño.
    - ID Length. Indica la longitud en octetos del campo *ID*. Es un campo de 8 bit sin signo de tamaño.

- TNF. El *Type Name Format* representa la estructura del valor del campo tipo definido a continuación. Este campo tiene 3 bits y puede tomar los siguientes valores: **[6]**
    - Empty, cuyo valor es el 0×00. Indica que este registro no tiene asociado ningún tipo de carga útil.
    - NFC Forum well-known type, cuyo valor es el 0×01. Es un tipo de formato de datos diseñado para uso y creación de tipos comunes de datos en etiquetas. Se compone de una URI compuesta por un campo URN (Uniform resource name) con el valor *"nfc"* seguido de una cadena que indica el espacio de nombre y el tipo del mismo (wkt:type). Los dos tipos principales son: **[7]**
        - NFC Forum Global Type: Los registro de tipo Global deben comenzar con una letra mayúscula. Los tipos existentes dentro de este grupos son Sp (Smart Poster), T (Text), U (URI) y Sig (Signature). Por ejemplo: urn:nfc:wkt:U.
        - NFC Forum Local Type: Comenzarán con una letra minúscula o en su defecto un número. Tiene la finalidad de ser usados dentro del contexto de otro registro (Registro de tipo Global).
    - Media-type, cuyo valor es el 0×02. Indica que la carga útil sigue un formato media-type BNF definido en el RFC 2046. **[8]**
    - Absolute URI, cuyo valor es el 0×03. Indica que la carga útil sigue un formato absolute URI BNF definido en el RFC 3986. **[9]**

- NFC Forum external type, cuyo valor es el 0×04.Indica que la carga útil tiene asignado por el usuario un espacio de nombres propio, siguiendo las reglas y formato RTD (Record Type Definitions). Se diferencian en que la cadena de espacio de nombres tiene la extensión *"ext"* en vez de *"wkt".* **[10]**
- Unknown, cuyo valor es el 0×05. Indica que el tipo de carga útil es desconocido.
- Unchanged, cuyo valor es el 0×06. Indica la terminación de una carga útil fragmentada en diferentes registros.
- Reserved, cuyo valor es el 0×07. Campo extra reservado para futuros tipos de carga útil. No se debe usar.

- Type Length. Indica la longitud en octetos del campo *Type*. Este campo puede tener valor cero para ciertos tipos de valores de *TNF.* Es un campo de 8 bit sin signo de tamaño. **[6]**

- Payload Length. Indica la longitud en octetos del campo *Payload*. Es un campo de tamaño igual a un entero sin singo. **[6]**

- Payload Type. Este campo describe el tipo de *Payload* contenido en el registro. **[6]**

- Payload ID. Este campo contiene un identificador URI. **[6]**

3.3. BASES DE DATOS DISTRIBUIDAS

Una **base de datos distribuida** (BDD) está compuesta por un conjunto de *"sitios"* o *"nodos"* con sus correspondientes bases de datos interconectadas entre sí. Dichas interconexión es tal, que desde cualquier punto de la red se puede acceder a cualquier dato, aparentando ser en su totalidad datos locales. Por lo tanto, un **sistema de bases de datos distribuidos** (SBDD) está compuesto por: **[18]**

- Diferentes computadores llamados sitios o nodos donde se alojarán las bases de datos.

- Redes de interconexión entre los diferentes sitios para intercambiar datos entre las diferentes bases de datos.

Existen diferentes tipos de bases distribuidas en función del tratamiento de datos que realizan. Son las siguientes:

- Centralizada. Todos los datos se encuentran en un solo nodo. Este tipo de base de datos distribuida asemeja su funcionamiento al modelo Cliente/Servidor. Los clientes se conectan al servidor central en busca de la información. Su única ventaja se encuentra en tener el procesamiento distribuido en los diferentes nodos. **[18]**

- Replicadas. Los datos están replicados en cada uno de los nodos, y por lo tanto, cada uno tiene su propia copia local y completa. Este modelo de base de datos distribuida presenta un alto coste en almacenamiento y actualización de los datos, pero como contrapartida proporciona una fiabilidad y disponibilidad máxima. **[18]**

- Particionadas. Los datos se encuentran repartidos en cada uno de los nodos disponibles. Los datos no se replican, por lo tanto, cada nodo tiene datos en exclusiva teniendo que ceder o pedir datos a el resto de nodos si fuera necesario. El costo de esta base de datos distribuida es menor que en las replicadas, pero su disponibilidad y fiabilidad no es máxima. **[18]**

- Híbridas. Presentan un esquema o modelo basado en las bases de datos distribuidas replicadas y particionadas. De esta forma, los datos están en replicados en uno o más nodos de forma que la fiabilidad y disponibilidad aumenta, aunque también su costo. **[18]**

Las bases de datos distribuidas presentan una seria de ventajas y desventajas. Algunas de ellas son:

Ventajas de las BDD [18]

- Mayor disponibilidad. Un fallo en un fragmento no tiene por qué afectar a toda la base de datos completamente. Además, si estamos en un modelo híbrido o replicado, ese fragmento dañado podría estar almacenado en otro nodo del sistema de bases de datos distribuidas.

- Economía. Si la cantidad de almacenamiento o de computo necesario para una base de datos es considerablemente grande, es más barato tener varias máquinas trabajando en esa base de datos que solamente una única máquina con todos los datos.

- Rendimiento. Este sistema asegura que los datos estarán de forma local en el nodo con más demanda. Además, estos sistemas de bases de datos pueden trabajar en paralelo para balancear la carga.

Desventajas de las BDD [18]

- Complejidad. La base de datos debe de ser transparente de cara al usuario. Además, se tiene que tener en cuenta la naturaleza distribuida de la misma a la hora del diseño.

- Seguridad. Necesita de un mayor nivel de seguridad debido que son más subsistemas a tratar que en una base de datos totalmente centralizada.

- Integridad. Mantener las diferentes bases de datos de cada nodo actualizadas genera una serie de reglas, y por consiguiente, mayor transmisión de datos entre los nodos para poder asegurar la integridad.

- Falta de estándares. No existen herramientas o metodologías que ayuden a pasar de una base de datos centralizada a una base de datos distribuida con facilidad.

- Diferentes tecnologías. Cada nodo puede utilizar diferentes tipos de sistemas gestores y modelos de bases de datos, lo que no facilita la replicación de los datos de manera directa.

3.4. ANDROID

Android, es el sistema operativo sobre el cual correrá la aplicación creada en este proyecto. Es un sistema operativo con núcleo Linux para dispositivos móviles (Smartphone) y Tablets desarrollado en un principio por Android Inc y posteriormente por Google. Este sistema operativo tiene una serie de elementos principales, los cuales son: **[19]**

- Núcleo Linux. Utilizado como una capa de abstracción entre el hardware y el software. Además, surte al sistema operativo de servicios de seguridad, gestión de memoria, controladores, etcétera. **[19]**

- Runtime de Android. El runtime de Android es un conjunto de bibliotecas que proveen de casi toda las funciones disponibles para el lenguaje de programación Java. **[19]**

- Bibliotecas. Un conjunto de bibliotecas en C/C++ que necesitan algunos componentes del sistema, como la parte de gráficos 3D o base de datos (SQLite). **[19]**

- Marco de trabajo. Las aplicaciones desarrolladas para este sistema operativo tienen el mismo acceso que las aplicaciones base a las APIs del sistema. **[19]**

- Aplicaciones Base. Aplicaciones que otorgan funcionalidades mínimas al sistema operativo. Un ejemplo de aplicaciones base son el navegador o la aplicación de correo electrónico. **[19]**

Actualmente existen multitud de versiones del sistema operativo Android. Desde la 1.0 o Apple Pie, hasta la última versión existente 5.1 o Lollipop. Cada versión aporta mejorías y avances con respecto a sus predecesoras.

En cuanto a lenguaje de programación sobre el cual se ha desarrollado la aplicación, ha sido Java junto su SDK (Software Development Kit). Por lo tanto, la aplicación ha sido desarrollada en

el lenguaje nativo del sistema operativo Android. Este hecho hace posible el acceso a las diferentes APIs del sistema, incluidas las que dan soporte a los elementos físicos del dispositivo móvil, siendo esto indispensable para poder acceder y utilizar en la aplicación el NFC móvil. La versión mínima de la API necesitada para poder acceder al NFC es la 10, disponible en dispositivos con la versión de Android 2.3 o superior. **[19]**

3.4.1. GOOGLE CLOUD MESSAGING

Google ofrece una serie de servicios para sus diferentes plataformas Android. Uno de ellos es el denominado **Google Cloud Messaging** (GCM), antiguamente conocido como *"Notificaciones Push"*.

Algunas aplicaciones necesitan tener la capacidad de poder recibir notificaciones cuando un evento externo al propio dispositivo móvil sucede. Normalmente estos eventos suceden en aplicaciones web o servidores externos. Un ejemplo de evento externo podrían ser los avisos que se despliegan en el Smartphone cuando un nuevo correo electrónico es recibido. Existen varias soluciones para hacer posible este tipo de comunicación. Las más representativas son:

- Mantener abierta una conexión permanente con el servidor. De esta forma se obtendrían en tiempo real cualquier aviso o notificación que sucediera en el servidor. Esta solución tiene los inconvenientes de consumir muchos recursos, principalmente memoria y datos de la red para mantener abierta la conexión. Por otro lado, en un dispositivo móvil la energía es un recurso valioso, y el utilizar esta técnica provocaría un gran desgaste de la misma debido al carácter permanente de la conexión. **[20]**

- Revisar de forma periódica el servidor. Esta técnica, también conocida como *"polling"*, se basa en consultar periódicamente el servidor de forma que se obtenga si ha sucedido algún evento desde la última vez que se comprobó. Con esta técnica, aunque se disminuye el uso de los recursos del dispositivo móvil, no se consigue el efecto de instantaneidad, es decir, se pueden dar eventos en el servidor en un momento concreto, coincidiendo o no con el periodo con el que se revisa la existencia de los mismos. Dependiendo del tipo de aplicación o funcionalidad buscada, puede ser necesario tener la capacidad de recibir la notificación de que un evento surgió con carácter inmediato, y por lo tanto, esta técnica no sería válida. **[20]**

Las dos técnicas anteriores tienen una serie de carencias en cuanto al uso de recursos o en la capacidad de procesar las notificaciones de manera instantánea. Google creó el servicio de **Google Cloud Messaging** con el objetivo de solventar estas carencias. La idea de este servicio radica en que será el servidor el encargado de enviar las notificaciones en cuanto estas se produzcan, y por lo tanto, el dispositivo móvil solo tendrá que esperar a que lleguen, siendo esto un ahorro máximo en cuanto a recursos se refiere, además de poder recibir las notificaciones de manera inmediata. Para que este servicio se pueda llevar a cabo, se introduce un nuevo elemento: *un servidor de mensajería*. **[20]**

El servidor de mensajería tiene la función de ser el intermediario entre la aplicación del servidor y la aplicación del dispositivo móvil. Las partes implicadas deberán estar registradas previamente en el servidor de mensajería, asociando a cada aplicación del lado del servidor una o más aplicaciones del lado del dispositivo móvil. Para poner en marcha el servidor de mensajería, y por consiguiente el servicio GCM, es necesario crear un proyecto dentro de la denominada *"Consola de desarrolladores de Google"* en el apartado dedicado a GCM. Una vez creado, tendremos a disposición un ID del proyecto o *"Project ID"* y una *"APIKey"* para registrar las aplicaciones del lado del servidor. En el *Apéndice B: Manual del programador*, se describirán en profundidad los pasos a seguir para crear un proyecto de Google y obtener el Project ID y la APIKey. **[20]**

El funcionamiento de este servicio se basa en una serie de pasos reflejados a continuación:

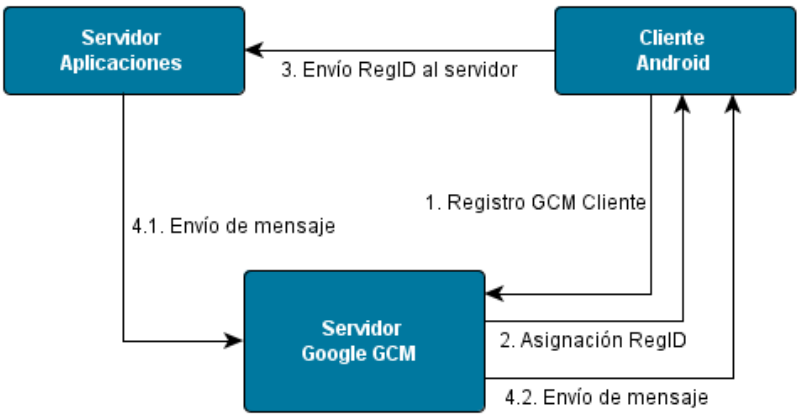


Figura3.4. *Proceso de validación GCM* **[20]**

De manera más específica y siguiente el diagrama mostrado en la figura anterior, los pasos a seguir para poner en marcha el servicio GCM son:

- Paso previo. Registro del servidor o aplicación externa al dispositivo móvil en el servidor de mensajería. Para ello, la aplicación del lado del servidor utilizará la APIKey obtenida anteriormente. Todas las notificaciones envidadas desde el servidor estarán acompañadas de esta APIKey. **[20]**

- Paso 1. Registro GCM del dispositivo móvil. Registro de la aplicación del dispositivo móvil como cliente capacitado para recibir notificaciones. Este registro se realizará asociando la aplicación con el Project ID obtenido anteriormente. Además, la aplicación deberá cumplir una seria de requisitos: **[20]**

    - La aplicación debe estar corriendo sobre Android 2.2 o superior.
    - Tener configurada una cuenta de Google en el dispositivo.
    - Tener instalado el paquete Google Play Store en el dispositivo.

- Paso 2. Asignación del RegID. Si el registro fue aceptado, se recibirá un *"código de registro"* o RegID que la aplicación deberá conservar. Además, la aplicación debe estar preparada para poder recibir actualizaciones en el RegID. **[20]**

- Paso 3. Envío del RegID al servidor. La aplicación del dispositivo móvil enviará a la aplicación del servidor el RegID. Este RegID actuará como identificador único de dispositivo, el cual utilizará la aplicación del servidor para enviar individualmente las notificaciones. La aplicación del servidor deberá almacenar el RegID de cada dispositivo que se registre en el servidor de mensajería. **[20]**

- Paso 4.1. Envío de un mensaje al servidor de mensajería. La aplicación del servidor enviará una notificación a un dispositivo en concreto utilizando la APIKey y el RegID. Dicha notificación pasará previamente por el servidor de mensajería. **[20]**

- Paso 4.2. Envío de un mensaje a la aplicación del dispositivo móvil. El mensaje recibido en el servidor de mensajería es reenviado a un cliente determinado en función del RegID especificado por la aplicación del servidor.

A priori, puede parecer un procedimiento complejo, pero es bastante intuitivo y metódico. Las ventajas de este servicio superan con amplitud la complejidad de la implementación del mismo. En el *Apéndice B: Manual del programador* se encuentra una explicación detallada sobre cómo se ha implementado este servicio.

4. CASOS DE USO

Antes de pasar a explicar la aplicación en sí, se ha creído conveniente introducir algunos casos de uso sobre los cuales se ha basado el funcionamiento de la aplicación. En los siguientes sub-apartados de este capítulo se explicarán algunos de ellos, sin ser estos los únicos casos de uso sobre los cuales se puede trabajar, ya que la versatilidad de la aplicación permite adaptarse a diferentes necesidades.

Por otro lado, destacar que todos los casos de uso que se puedan desarrollarse con esta aplicación parten de la misma base común: La captación de información mediante el dispositivo móvil. Hay que tener en cuenta que toda la información almacenada en la base de datos hace referencia a una imagen virtual de cada objeto físico leído. El usuario será el encargado de introducir la información de manera correcta en la base de datos, además de mantener la misma actualizada. De lo contrario, la información que la aplicación ofrece al usuario puede ser incorrecta, restando utilidad a la aplicación. Con el objetivo de facilitar esta tarea al usuario, se ha hecho especial hincapié en dos factores comunes a toda la aplicación. Son los siguientes:

1. **La facilidad de uso.** La facilidad de uso en toda la aplicación ha sido una constante. La razón es que se trata de actualizar o insertar nueva información de objetos no conectados, por lo que depende de la voluntad del usuario que la imagen virtual del de un objeto sea lo más correcta posible. Centrándonos en la actividad de la inserción o búsqueda de información, solo se necesita acercar el dispositivo móvil a las etiquetas NFC del objeto deseado. Dependiendo en qué posición se acerque el dispositivo móvil a las etiquetas (landscape, portrait), se iniciará una búsqueda en la base de datos del objeto leído o una inserción de información.

2. **La sencillez de la interfaz gráfica.** Se ha desarrollada una interfaz gráfica sencilla para toda la aplicación. Volviendo a la actividad correspondiente a la inserción o búsqueda de información, se ha hecho especial énfasis en este punto. Esta actividad consta de 2 funcionalidades a elegir, siendo estas intuitivas y directas al usuario. Son las siguientes

    - Botón atrás: Tiene una utilidad muy simple. Provoca el cierre de la actividad actual devolviendo al menú principal.

    - Botón de bajar volumen: Con este botón se activa o desactiva el tipo de lectura agrupado de etiquetas NFC asociado a un tipo de interacción definida previamente. Cada tipo de interacción está a su vez asociada a un tipo de relación, aportando de esta forma información complementaria sobre los objetos leídos. Por tanto, el usuario tendrá que determinar lo siguiente:

        - Modo de lectura agrupado desactivado: Con este modo de lectura, un usuario al acercar el dispositivo móvil a la etiqueta NFC del objeto a leer, solo obtendría información del mismo, sin relacionar o enlazar con otros objetos. Este modo de lectura sería el típico ofrecido por

cualquier lector de etiquetas NFC, mostrando la información contenida en la etiqueta leída.

- Modo de lectura agrupado activado: Con este modo de lectura, al leer las etiquetas NFC de diferentes objetos, entraría en juego el tipo de interacción y el tipo de relación seleccionada previamente por el usuario, quedando los objetos leídos asociados a estos tipos de interacciones y relaciones. Los casos de uso que se explicarán a continuación, trabajan con este modo de lectura activado.

Se puede observar la simplicidad de uso de las dos funcionalidades a elegir. Además, se ha dotado a la interfaz de texto e imágenes explicativas para que el usuario sepa en todo momento la acción que debe realizar.

En el *Apéndice A: Manual del usuario*, se tiene a disposición información detallada sobre el funcionamiento de la aplicación y su interfaz gráfica.

4.1. CASO DE USO: POSICIONAMIENTO

Este caso de uso nace de la necesidad existente de tener información acerca de la localización de diferentes objetos en un entorno que muchas veces no es propicio o adecuado para mantener el control sobre cada uno de ellos. Mediante el uso de la aplicación, se podrá registrar un número ilimitado de objetos, asociándolos a un objeto contenedor al que se le supone una posición conocida e invariable. De esta forma, con un simple gesto, como es el de tocar con el teléfono móvil la etiqueta NFC del objeto contenedor, podríamos averiguar qué objetos están asociados a dicho contenedor. Con el objetivo de mejorar la comprensión de este caso de uso, se desarrollarán dos ejemplos a continuación.

Posicionamientos de libros

Supongamos una estantería haciendo la función de librería y por lo tanto de objeto contenedor. En ella existen una serie medianamente extensa de libros y se pretende buscar uno en concreto. En un caso como este, puede resultar difícil saber a priori qué libros contiene dicha librería, y muchos menos si el que se busca se encuentra en esa u otra librería. Para solventar este problema, se va a utilizar una estructura de datos que denominaremos tripleta. Esta tripleta consiste en: un tipo de interacción, un tipo de relación y un objeto de referencia. Como el objetivo es construir relaciones entre objetos de forma simple, se ha jerarquizado el conjunto de relaciones en dos niveles: tipo de interacción y tipo de relación. Por ejemplo, el tipo de relación *"Esta en"* forma parte del tipo de interacción *"Espacial"*. Por otro lado, se ha establecido un objeto de referencia para terminar de particularizar la relación. Por ejemplo, si el objeto de referencia es una librería X, la relación *"Esta en"* acaba siendo *"Esta en la librería X"*. Partiendo de esto, se puede comprender mejor los siguientes pasos:

1. Asociar una etiqueta NFC a cada libro. Cada etiqueta contendrá una referencia o información de importancia para el usuario acerca de cada libro.

2. Asociar una etiqueta NFC a la librería, siendo esta nuestro objeto de referencia. Esta etiqueta contendrá información inequívoca y concreta acerca del objeto contenedor, de forma que el usuario pueda saber dónde está dicho objeto.

3. Crear la tripleta tipo de interacción-tipo de relación-objeto de referencia. Por ejemplo, se podría crear la interacción de tipo *"Espacial"*, asociada al tipo de relación *"Esta en"* con el objeto contenedor *"Librería"*.

4. Asociar libros a la tripleta. Este proceso, como ya se ha comentado con anterioridad, se realizará de manera sencilla. Simplemente se necesita acercar el dispositivo móvil a las diferentes etiquetas NFC de los objetos. Automáticamente se iniciaría el proceso de lectura y almacenamiento de información en la base de datos, siendo el primer objeto leído, el objeto de referencia en la relación.

Tras finalizar el proceso de adicción de información, si se leyera la etiqueta NFC de la librería utilizando la aplicación, se tendría un listado de todos los libros que está contiene, ayudando a un mejor posicionamiento y localización de los mismos

A continuación se encuentra una tabla a modo de visualización del caso de uso anterior.

Interacción	Relación	Objeto Contenedor	Objetos
Espacial	Está en	Librería	Libro 1
Espacial	Está en	Librería	Libro 2
Espacial	Está en	Librería	Libro 3
Espacial	Está en	Librería	Libro N

Tabla 4.1. *Ejemplo caso de uso. Posicionamiento*

Posicionamiento herramientas

Este caso de uso, es análogo al anteriormente explicado. Se pretende identificar qué herramientas se encuentran dentro de un objeto contenedor, por ejemplo un armario. De igual manera que en el caso anterior, se dotará de etiquetas NFC al objeto contenedor (Armario) con información acerca del mismo, y a cada objeto que este contiene (Herramientas). Se crea el mismo tipo de tripleta (Espacial-Está en-Armario) y posteriormente se inicia el proceso de adicción de objetos. El resultado no varía con respecto al caso anterior, exceptuando al objeto contendor y los objetos contenidos. De esta forma, tendríamos localizadas las diferentes herramientas que se encuentran en el armario.

4.2. CASO DE USO: JERARQUIZACIÓN

La idea central de este caso de uso se basa en la necesidad de clasificar objetos de alguna forma lógica, destacando por importancia, familias, usos u otros tipos de jerarquizaciones existentes y utilizadas en la vida diaria. Usando la aplicación se podrían crear diferentes tipos de agrupaciones, dando un valor añadido a cada objeto. Además, este caso de uso se podría combinar con el anterior (Posicionamiento), pudiéndose dar el caso de que un mismo objeto forma parte de diferentes tipos de relaciones.

Con el objetivo de una mejor compresión, se presentarán dos ejemplos de este caso uso.

Jerarquización de medicamentos

Para este caso de uso, nos posicionaremos en un establecimiento donde se tenga un gran número de medicamentos de diferentes tipos y para diferentes afecciones. Con la aplicación se pretender agrupar a los diferentes medicamentos por usos o familias, de forma que se pueda listar de forma rápida los diferentes tipos de medicamentos para una misma afección. Los pasos a seguir serían los siguientes:

1. Añadir una etiqueta NFC a cada medicamento con sus datos de identificación.

2. Asociar una etiqueta NFC a una zona de almacenaje con un tipo de uso o familia de medicamentos.

3. Crear la tripleta tipo de interacción-tipo de relación-objeto de referencia. En este caso, se podría crear la interacción de tipo *"Familiar"*, asociada a la relación *"Usado cómo"* con el objeto diferenciador *"Antigripal"*.

4. Asociar medicamentos a la tripleta. Este proceso se realiza de la misma manera que en el caso de uso del posicionamiento anteriormente explicado.

Una vez finalizado el proceso de inserción de medicamentos, al tocar una etiqueta NFC con el teléfono móvil, se listaría los tipos de medicamentos existentes del tipo de familia al que se hace referencia con la etiqueta.

En la siguiente tabla se puede observar información acerca de este caso de uso.

Interacción	Relación	Familia del objeto	Objetos
Familiar	Usado cómo	Antigripal	Medicamento 1
Familiar	Usado cómo	Antigripal	Medicamento 2
Familiar	Usado cómo	Antigripal	Medicamento 3
Familiar	Usado cómo	Antigripal	Medicamento N

Tabla 4.2. *Ejemplo caso de uso. Jerarquización*

Jerarquización de libros

Volviendo al caso de uso acerca del posicionamiento de los libros, lo que se pretende conseguir ahora es asociar los diferentes tipos de temáticas de libros con los propios libros. Se asocia una etiqueta NFC a cada libro con información acerca del mismo. Por otro lado, se tendrá una etiqueta NFC con la familia o género literario de los libros a asociar. Supongamos que se quiere asociar los libros por el género literario *"Poesía"*. Tendríamos una etiqueta NFC con el objeto de referencia (Poesía) y los diferentes objetos agrupados (Libros de poesía). De forma análoga al caso anterior, se crearía una tripleta (Familiar-Tipo de temática-Poesía) sobre la cual posteriormente se adicionarán los diferentes objetos. De esta forma, al tocar una etiqueta NFC correspondiente al género literario se listarían los libros que corresponden con este género.

4.3. CASO DE USO: POSTER INTELIGENTE

En este caso de uso, se tiene la intención de evitar saltarse pasos a seguir en protocolos de actuación que exijan un riguroso seguimiento del mismo. Usando la aplicación, se podría crear diferentes protocolos asociados a actividades con sus pasos o elementos. La idea es asociar una etiqueta NFC a un poster identificativo de una actividad. Con esto, un usuario al tocar la etiqueta NFC asociada a un poster que identifica dicha actividad, mostraría la lista de elementos a tener en cuenta en la realización o práctica del mismo. A continuación, se desarrollará un ejemplo de este caso de uso.

Protocolo elementos de seguridad

Para realizar este caso de uso, nos posicionaremos en un centro de trabajo donde se tengan que seguir una serie de pasos en cuanto a vestimenta de seguridad requerida se refiere. Dependiendo del tipo de tarea a realizar, se necesitará utilizar diferentes tipos de vestimenta. Mediante la aplicación, se podría crear los diferentes protocolos de vestimenta requeridos en función del trabajo a realizar. Los usuarios solo tendrían que tocar con el dispositivo móvil la etiqueta NFC asociada al poster que indica el tipo de trabajo, y se desplegaría un listado de vestimentas necesarias para realizar dichas tareas. Los pasos a seguir serían los siguientes:

1. Añadir una etiqueta NFC a cada vestimenta de seguridad o protectora con información identificativa de la misma.

2. Asociar una etiqueta NFC a cada poster que identifique cada actividad o trabajo realizado.

3. Crear la tripleta tipo de interacción-tipo de relación-objeto de referencia. En este caso, se podría crear la interacción de tipo *"Protocolizar"*, asociada a la relación *"Necesita de"* con el objeto diferenciador *"Actividad industrial"*.

4. Asociar vestimenta de seguridad a la tripleta. Esta operación se realizará de la misma manera que en los casos de uso anteriores.

Una vez finalizado el proceso de inserción de vestimenta de seguridad a cada actividad, al tocar con el teléfono móvil la etiqueta asociada a la actividad a realizar, desplegaría una lista con las diferentes vestimentas que se deberían usar.

En la siguiente tabla se puede observar información acerca de este caso de uso.

Interacción	Relación	Familia del objeto	Objetos
Protocolizar	Necesita de	Actividad Industrial	Vestimenta 1
Protocolizar	Necesita de	Actividad Industrial	Vestimenta 2
Protocolizar	Necesita de	Actividad Industrial	Vestimenta 3
Protocolizar	Necesita de	Actividad Industrial	Vestimenta N

Tabla 4.3. *Ejemplo caso de uso. Protocolizar*

5. DESCRIPCIÓN DEL PROTOTIPO

5.1. ELEMENTOS DEL PROTOTIPO

El prototipo de aplicación Android desarrollado está compuesto por varios elementos principales, jugando un papel indispensable para el correcto funcionamiento y usabilidad requeridas en el prototipo. Dichos elementos son el **dispositivo móvil**, las **etiquetas NFC**, el **servidor** y la **base de datos distribuida**. En el capítulo *Fundamentos Tecnológicos*, se describió cada una de estas tecnologías, pero realmente no se ahondó como se utilizaban o integraban en la aplicación. A continuación, se describirá cada uno de estos elementos haciendo énfasis en cómo son usados en la aplicación.

Dispositivo móvil

Para el desarrollo de la aplicación, realmente se utilizó un teléfono móvil inteligente o Smartphone con el sistema operativo Android 4.4.4 o KitKat y diversos emuladores Android en distintas versiones del sistema operativo. Este dispositivo contaba con el hardware necesario del NFC.

Dentro del sistema NFC en modo pasivo que se ha desarrollado con este prototipo, el Smartphone jugaría un papel de elemento o dispositivo activo y su modo de comunicación es de lectura o escritura. El Smartphone leerá o escribirá información en las etiquetas NFC.

En cuanto al almacenamiento de información, el Smartphone será uno de los nodos de la base de datos distribuida. Android provee por defecto un gestor de base de datos (SQLite), y por tanto, se ha utilizado este gestor de base de datos al ser el óptimo para este tipo de dispositivos.

Etiquetas NFC

El tipo de etiqueta NFC utilizada en el proyecto es de la marca SMARTRAC BullEye 320_3 y circuito integrado NXP NTAG216. Esta etiqueta NFC está basada en el estándar ISO 14443A y está clasificada dentro del grupo de etiquetas NFC de tipo 2. La capacidad de esta etiqueta es de 888 bytes de tamaño máximo.

En el sistema NFC de la aplicación, la etiqueta tiene el papel de elemento o dispositivo pasivo. El elemento activo, en este caso el Smartphone, sería el encargado de interactuar con la etiqueta, dotando o adquiriendo datos de la misma.

Servidor

El servidor no es más que una computadora que tendrá el papel de ser uno de los nodos de la base de datos distribuida. Por tanto, en el servidor se ejecutará una base de datos. En este caso, se ha optado por un gestor de base de datos MySQL. Además, se ha provisto de una sencilla interfaz web

mediante la cual el usuario pueda visualizar y modificar la base de datos distribuida de forma sencilla. Para ello, el servidor además de actuar como uno de los nodos de la base de datos distribuida, también tendrá la función de ser un servidor web, concretamente Apache. En este servidor web correrán una serie de script en PHP, mediante los cuales se consigue visualizar correctamente el contenido de la base de datos distribuida, además de dotar de elementos necesarios para modificar los datos de la misma, enviando un aviso al Smartphone en caso de que esta operación suceda.

Base de datos distribuida

La base de datos distribuida de la aplicación es de tipo replicada, y por lo tanto, en cada nodo de la misma estará una copia completa de la base de datos. En cuanto al modelo, cada nodo utilizará un modelo relacional de base de datos. La aplicación en su totalidad cuenta con dos nodos nombrados anteriormente. Son los siguientes:

- El dispositivo móvil o Smartphone con un gestor de base de datos SQLite.
- El servidor con un gestor de base de datos MySQL.

En cuanto la estructura de la base de datos, tenemos que está compuesta por una serie de tablas relacionadas entre sí. A continuación se puede observar una figura que representa las tablas y sus relaciones.

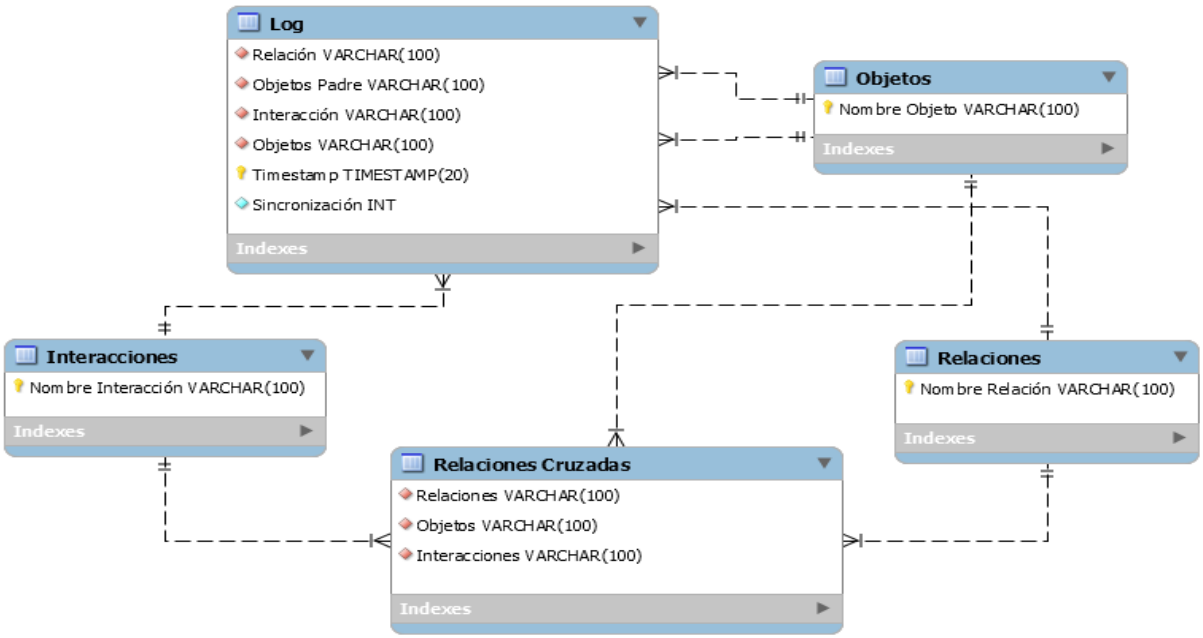


Figura 5.1. *Estructura de la base de datos distribuida*

Se observa en la figura anterior, que la estructura de la base de datos está compuesta por cinco tablas. Son las siguientes:

- Tabla Objetos. Esta tabla almacenará el nombre de cada uno de los objetos leídos.

    - Atributos:
        - Nombre objeto. Primary Key.

- Tabla Relaciones. Esta tabla almacenará el nombre de cada una de las diferentes relaciones dadas al utilizar la aplicación.

    - Atributos:
        - Tipo relación. Primary Key.

- Tabla Interacciones. Esta tabla almacenará el nombre de cada tipo de interacción que surja al utilizar la aplicación.

    - Atributos:
        - Tipo interacción. Primary Key.

- Tabla Relaciones Cruzadas. Esta tabla se formará a partir de los atributos de las tres tablas anteriores, formando la tripleta tipo de interacción, tipo de relación y objeto de referencia. Con esta tabla se pretende dejar constancia de los tipos de relaciones que se generan al realizar ciertos tipos de interacciones sobre objetos. Un ejemplo podría ser:

  Se crea el tipo de interacción *"Espacial"* asociada al tipo de relación *"Está en"*. Posteriormente se lee un objeto *"Librería"*. La tabla Relaciones Cruzadas contendría la tripleta (Esta en, Librería, Espacial). Con esta tripleta se pretende representar los diferentes tipos de relaciones asociados a tipos de interacciones. En este caso se representaría la relación *"Está en la librería"*, siendo esta creada por medio de una interacción de tipo *"Espacial"*.

    - Atributos:
        - Tipo interacción. Primary Key.
        - Nombre objeto de referencia. Primary Key.
        - Tipo relación. Primary Key.

- Tabla Log. Esta tabla se formará en parte por los atributos de las tres primeras tablas nombradas anteriormente. El resto de atributos son para otorgar información extra. Se pretende captar información acerca del tipo de interacción activa y su consiguiente relación, objetos leídos, el tiempo de su lectura y si actualmente estos datos están sincronizados o no con el resto de nodos de la base de datos distribuida. Siguiendo el ejemplo del caso anterior, se crea o selecciona el tipo de interacción *"Espacial"* asociada al tipo de relación *"Está en"*. Posteriormente se leen varios objetos *"Librería, Libro1"*.Lo que se pretende representar con esta tabla es que para un tipo de interacción *"Espacial"* se tiene que *"El libro 1está en la librería con un tiempo X y con un estado de sincronización Y"*. Destacar que el primer objeto leído sería el que conforme la particularización de la tripleta tipo de interacción, tipo de relación y objeto de referencia.

    - Atributos:
        - Tipo de relación. NOT NULL.

- Objeto Padre o de referencia. NOT NULL.
- Objeto. NOT NULL.
- Tipo de interacción. NOT NULL.
- Timestamp. Primary Key.
- Sincronización. NOT NULL. Un 0 significa no sincronizado, un 1 sincronizado y un 2 a la espera de eliminar.

En cuanto a la sincronización de los dos nodos que conforman la base de datos distribuida, se realizará teniendo en cuenta una serie de consideraciones.

- Se podrá introducir datos, y por tanto replicarlo al resto de nodos, por ambos nodos que conforman la base de datos distribuida.

- Solo se podrá eliminar datos desde el servidor. Esta operación es exclusiva a este dispositivo.

- Ambos nodos realizarán la operación de sincronización cuando se inicie esta desde el Smartphone. Desde el servidor se enviará un aviso en caso de existir información a replicar.

Desde el Smartphone se leerá información de las etiquetas NFC. Cada lectura supondrá un almacenamiento en la base de datos local del dispositivo móvil. Posteriormente se iniciará un proceso de sincronización de los datos a petición del usuario. Llegados a ese punto, el usuario podrá observar los datos de las diferentes tablas a través de la interfaz web del servidor, añadiendo o eliminando datos de las mismas si lo desea. En caso de realizarse algún cambio en la base de datos del servidor, se enviará un aviso al Smartphone con el objetivo de que este inicie nuevamente la operación de sincronización de datos.

En el *Apéndice B: Manual del programador* se encuentra una explicación detallada sobre cómo se ha implementado cada una de estas funcionalidades.

5.2. FUNCIONAMIENTO

Se puede constatar que el grueso este proyecto se basa en la aplicación en Android desarrollada. Realmente, y como se ha comentado en apartados previos, la aplicación en sí no está compuesta únicamente por el aplicativo que funciona en el Smartphone, sino que también se compone de otros elementos. Tanto desde el Smartphone como desde el servidor, se puede interactuar de manera diferente con dichos elementos que componen el total de la aplicación. A continuación se mostrará un análisis de la interfaz de ambos como forma de explicar el funcionamiento de la aplicación en su totalidad.

5.2.1. INTERFAZ SMARTPHONE

La aplicación se instalará en el teléfono móvil, y se podrá iniciar mediante el icono que está dentro del círculo rojo de la siguiente imagen:

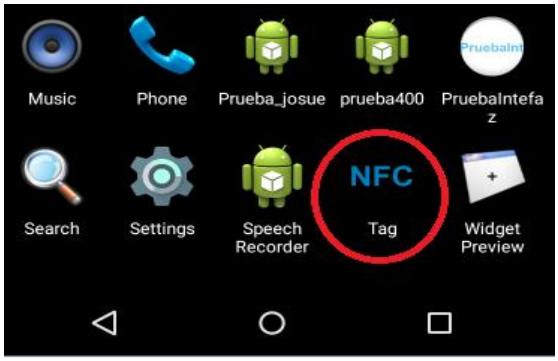


Figura 5.2. *Icono de inicio aplicación móvil*

Pulsando sobre el icono anterior, se daría comienzo a la actividad principal o main activity de la aplicación. Es la siguiente:

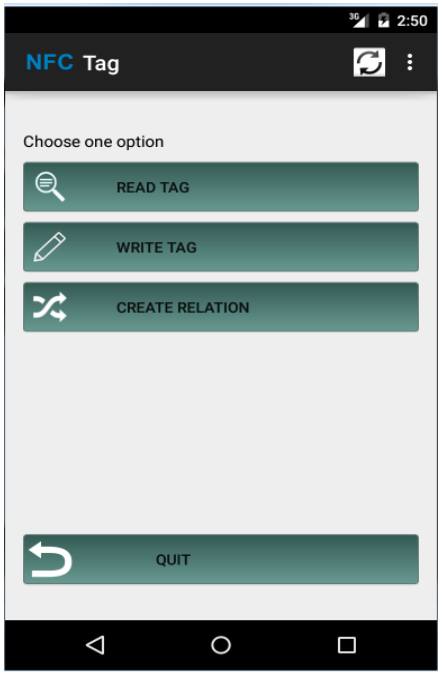


Figura 5.3. *Actividad principal aplicación móvil*

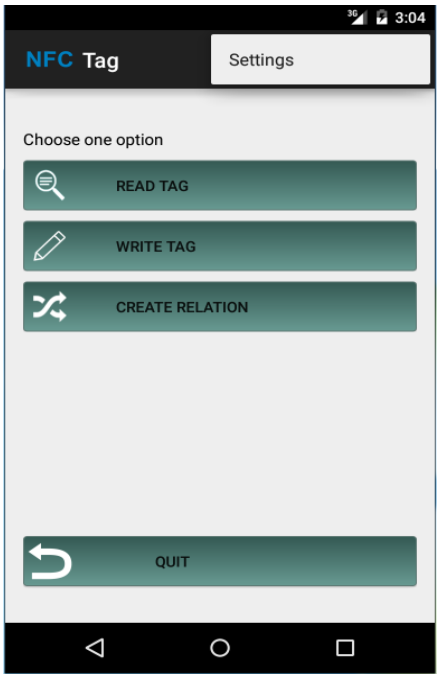


Figura 5.4. *Actividad principal aplicación móvil 2*

La pantalla principal tiene una serie de botones que se describirán a continuación:

- Botón Read Tag. Este botón selecciona el modo lectura de la aplicación. Inicia una segunda actividad donde ya se puede comenzar a leer etiquetas NFC.

- Botón Write Tag. Este botón selecciona el modo escritura de la aplicación. Inicia una segunda actividad donde elegir el tipo de escritura a realizar en la etiqueta NFC.

- Botón Create Relation. Este botón selecciona el modo crear, eliminar o seleccionar relación. Inicia una segunda actividad donde crear, eliminar o seleccionar el tipo de relación actualmente usada en la aplicación.

- Botón Quit. Este botón cierra y finaliza la aplicación.

En la barra superior de la aplicación existen dos botones o iconos más. El de sincronización y el de settings. Su función es la siguiente:

- Botón sincronización. Este botón inicia la sincronización del Smartphone con el servidor.

- Botón settings. Muestra algunas características de la aplicación.

En esta actividad, destacan las siguientes partes del código:

GCM

En esta actividad se tiene implementado el código sobre el servicio de Google, *"Google Cloud Messaging"*. Con este servicio lo que se pretende conseguir es avisar al usuario del dispositivo móvil de la existencia de datos no sincronizados, sin la necesidad de que la aplicación tenga que estar revisando el estado del servidor continuamente. El propio servidor será el encargado de enviar los avisos en caso de que fuera necesario, consiguiendo una mayor eficiencia de recursos en el dispositivo móvil al no tener que realizar este ningún tipo de operación más allá de un previo registro de la misma. El código es el siguiente:

Al iniciar la aplicación se comprueba si se tiene almacenado el RegID. En caso de no tenerlo se inicia el proceso de registro llamando a la función initGCM*()*, comprobando si los servicios de *Google Play Store* están instalados.

```
private void initGCM(){
...
       if (checkPlayServices()) {
           registerInBackground(); //Check if Google Play Store is installed
       }
...
}
```

Si está instalado el servicio, se llama a la función *registerInBackground()*, a través de la cual se obtendrá el RegID.

```
private void registerInBackground() {
...
       @Override
       protected String doInBackground(Void... params) {
              try {
                   if (gcmObj == null) { //Get RegId
                       gcmObj = GoogleCloudMessaging.getInstance(getApplicationContext);
                       regId = gcmObj.register(ApplicationConstants.GOOGLE_PROJ_ID);
                   }
```

```
                }catch (IOException ex) {
                        msg = "Error :" + ex.getMessage();
                }
        }
        @Override
        protected void onPostExecute(String msg) {
                if (!TextUtils.isEmpty(regId)) {
                     // Store RegId created by GCM Server in SharedPref
                     storeRegIdinSharedPref(applicationContext, regId);
                }
        }
...
}
```

A continuación, se almacena el registro en el dispositivo y se llama a la función encargada de enviarlo al servidor.

```
private void storeRegIdinSharedPref(Context context, String regId) {
...
        SharedPreferences.Editor editor = prefs.edit();
        editor.putString(REG_ID, regId);
        editor.commit();
        storeRegIdinServer();//Function call to store on the server
...
}
```

Esta función enviará el RegID utilizando una petición HTTP asíncrona. Además, se encargará de manejar los errores que surjan en el proceso de envío.

```
private void storeRegIdinServer() {
...
        AsyncHttpClient client = new AsyncHttpClient();
        RequestParams params = new RequestParams();
        params.put("regId", regId);
        client.post(ApplicationConstants.APP_SERVER_URL, params,new AsyncHttpResponseHandler() {
        @Override
        public void onSuccess(int arg0, Header[] arg1, byte[] arg2) {
                    Toast.makeText(applicationContext,"Reg Id shared successfully with Web App
                    ",Toast.LENGTH_LONG).show();
        }
        @Override
        public void onFailure(int statusCode, Header[] arg1, byte[] arg2,Throwable arg3) {

        ... //Get Errors
        }
...
}
```

Una vez realizado todo este proceso, se habría finalizado el registro del dispositivo móvil en el servidor de mensajes GCM.

Sincronización de bases de datos

En esta actividad también se tiene implementado el código que realiza las peticiones de sincronización de los datos.

Al pulsar el botón de sincronización, se llama a estas dos funciones:

```
syncMySQLDBSQLite(); //Sincronización web. Móvil envía datos a la web.
syncSQLiteMySQLDB(); //Sincronización móvil. Web envía datos al móvil.
```

La primera función, *syncMySQLDBSQLite(),* envía los datos no sincronizados desde el dispositivo móvil a al servidor .

```
public void syncMySQLDBSQLite() {
...
        AsyncHttpClient client = new AsyncHttpClient();
        RequestParams params = new RequestParams();
        ArrayList<HashMap<String, String>> userList = mydatabase.getAllData();
        if(userList.size()!=0) {
           if(mydatabase.dbSyncCount() != 0) {
              prgDialog.show();
              params.put("usersJSON",mydatabase.composeJSONfromSQLite());
               client.post("http://192.168.0.13:80/nfc/insertuser_.php",params ,new Asyn  cHttpRe-
               sponseHandler() {


               @Override
               public void onFailure(int statusCode, Header[] arg1, byte[] arg2, Throwable arg3){

                ... //Get Errors
                }
                @Override
                public void onSuccess(int arg0, Header[] arg1, byte[] arg2) {
                JSONArray arr = new JSONArray(cadena);
                for(int i=0; i<arr.length();i++) {
                        JSONObject obj = (JSONObject)arr.get(i);
                        mydatabase.updateSyncStatus (obj.get("relacion").toString(),
                        obj.get("objetoPadre").toString(), obj.get("objeto").toString(),
                        obj.get("interaccion").toString(),obj.get("tiempo").toString());
                        }

                   }
                });
        }
...
}
```

Esta función recoge los datos no sincronizados y los envía en formato JSON mediante peticiones HTTP asíncronas, de igual manera que en el caso del GCM. Posteriormente, espera una respuesta de los datos enviados. Si se devuelve un error, se ejecutaría la función *onFailure()*, en caso contrario, se ejecutaría la función *onSuccess()*. Esta última función recibe la misma cadena de datos enviada a modo de comprobación de que se han insertado la información en la base de datos del servidor correctamente. Por último, se actualizaría la base de datos del dispositivo móvil para señalar que los datos ya están sincronizados.

En la segunda función, *syncSQLiteMySQLDB(),* los pasos a seguir son exactamente iguales que en el caso anterior, pero a la inversa. Esta vez se recogen los datos a sincronizar de la base de datos del servidor, utilizando una vez más peticiones HTTP asíncronas. Una vez recogidos los datos a sincronizar, y desde la función *onSucces()* que denota el éxito al recoger los datos, se llama a la función *updateSQLite(cadena)* con la cadena obtenida para actualizar los datos en la base de datos del dispositivo móvil. El código es el siguiente:

```
public void updateSQLite(String response) {
...
        ArrayList<HashMap<String, String>> datasynclist;
        datasynclist = new ArrayList<HashMap<String, String>>();
        // Create GSON object
        Gson gson = new GsonBuilder().create();
        JSONArray arr = new JSONArray(response);
        if(arr.length() != 0) {
```

```
            for (int i = 0; i < arr.length(); i++) {
                  // Get JSON object
                  JSONObject obj = (JSONObject) arr.get(i);
                  queryValues = new HashMap<String, String>();
                 queryValues.put("relacion", obj.get("relacion").toString());
                  queryValues.put("objetoPadre", obj.get("objetoPadre").toString());
                  queryValues.put("objeto", obj.get("objeto").toString());
                  queryValues.put("interaccion", obj.get("interaccion").toString());
                  queryValues.put("tiempo", obj.get("tiempo").toString());
                  updateMySQLSyncSts(gson.toJson(datasynclist));
                }
        }
...
}
```

Por último, se llamaría a la función *updateMySQLSyncSts(gson.toJson(datasynclist))*. Con esta función se pretende conseguir que la base de datos del servidor marque sus datos como ya actualizados. El código sería el siguiente:

```
public void updateMySQLSyncSts(String json) {
...
        AsyncHttpClient client = new AsyncHttpClient();
        RequestParams params = new RequestParams();
        params.put("sincro", json);
        client.post("http://192.168.0.13:80/nfc/updatesyncsts.php", params, new AsyncHttpRe-
        sponseHandler() {

        @Override
        public void onFailure(int arg0, Header[] arg1, byte[] arg2, Throwable arg3) {
        ... //Get Errors
        }
        @Override
        public void onSuccess(int arg0, Header[] arg1, byte[] arg2) {
        ...
        }

    });
...
}
```

Una vez más, utilizando una petición HTTP asíncrona se envían los datos a marcar como sincronizados en la base de datos externa. Como en casos anteriores, el método *onFailure ()* procesará los errores que se puedan producir y el método *onSuccess()* gestionará el éxito de la operación del envío de la cadena con los datos a marcar como actualizados.

5.2.1.1. ACTIVITY LECTURA DE ETIQUETAS

Esta actividad es iniciada al pulsar el botón *"Read Tag"* de la actividad principal. La lectura de la etiqueta NFC comenzaría nada más acercar el teléfono a una etiqueta NFC, estando activado el NFC del propio Smartphone. Dependiendo de cómo se lea dicha etiqueta, el teléfono móvil en posición **landscape** o **portrait**, se realizará una lectura con el objetivo de realizar una búsqueda en la base de datos, o por el contrario, se realizará una inserción en la base de datos. El objetivo de elegir un tipo de acción u otra en función de la posición de la pantalla, es de dotar de simplicidad y facilidad de uso a la aplicación. El hecho de poder marca el tipo de acción a realizar con un simple movimiento físico, otorga un grado de usabilidad de carácter directo para el usuario, ayudando de esta forma a la facilidad de uso.

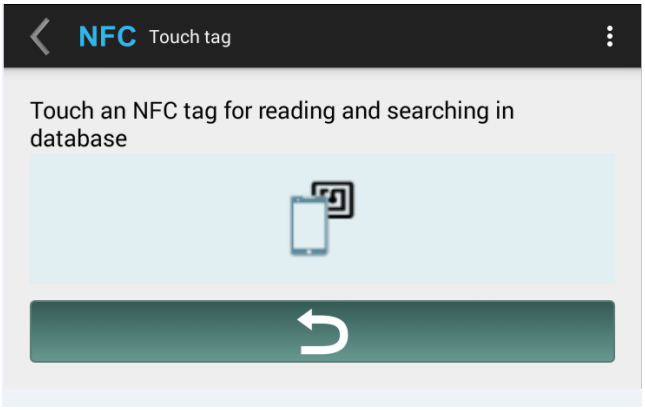


Figura 5.5. *Lectura en posición Landscape.*

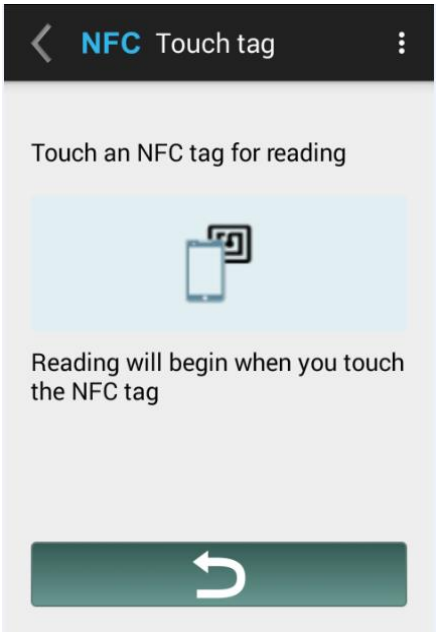


Figura 5.6. *Lectura en posición Portrait.*

Esta actividad está compuesta simplemente por un botón que volvería a mostrar la actividad principal además texto e imágenes explicativas sobre los pasos a seguir para leer una etiqueta NFC.

Al acercar el teléfono móvil a la etiqueta NFC, se dispararía un evento que provocaría el inicio de una nueva actividad. Esta actividad está asociada a la actividad de lectura de etiquetas, y dependiendo en qué posición estaba el dispositivo móvil (landscape o portrait) al leer la etiqueta, mostrará unos resultados correspondientes a una búsqueda o por el contrario, mostrará el resultado de una inserción.

Lectura en posición landscape

Al acercar el teléfono móvil a la etiqueta NFC en posición landscape, se inicia una búsqueda en la base de datos acerca del objeto leído, mostrando si existe alguna coincidencia en la tabla *"Log"*. Supongamos que en la etiqueta NFC está almacenado el nombre de un objeto, en este caso *"Rey"*. Al hacer una lectura con el Smartphone en posición landscape se realizará una búsqueda de todas las relaciones con el objeto leído. Un posible resultado de la búsqueda sería:

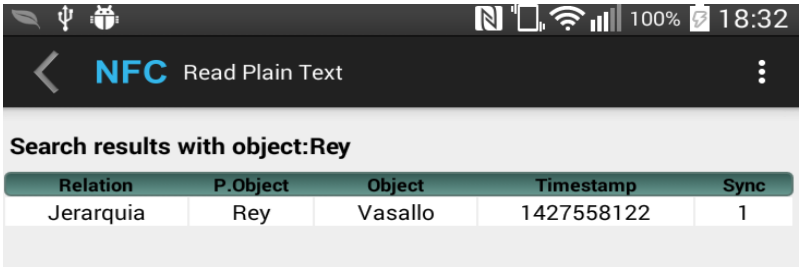


Figura 5.7. *Resultado de la búsqueda del objeto leído.*

Como resultado se ha obtenido una única entrada en la tabla *"Log"* de la base de datos acerca del objeto *"Rey"*. La relación es de tipo *"Jerarquía"* donde el objeto padre o de referencia es *"Rey"*, objeto agregado o hijo es *"Vasallo"*, timestamp de inserción en la base de datos *"1427558122"* y sincronización *"1"*, lo que da a entender que esta relación está sincronizada con la base de datos del servidor.

En esta actividad, destaca el código de lectura de datos de una etiqueta NFC, y la búsqueda de datos en la base de datos del dispositivo. El código es el siguiente:

En primer lugar, en el manifest.xml de la aplicación Android, se ha asociado a esta actividad un intent-filter. Este intent-filter proporcionará que la actividad se ejecute o se inicie al tocar con el dispositivo móvil una etiqueta NFC, aunque esta aplicación esté cerrada.

```
<activity

        android:name=".ActivityReadText"
        android:configChanges="orientation|screenSize|keyboardHidden|keyboard"
        android:label="@string/title_activity_ReadText"
        android:parentActivityName="com.example.nfc_writing.ActivityMenuRead" >
        <intent-filter>
           <action android:name="android.nfc.action.NDEF_DISCOVERED" />
                 <category android:name="android.intent.category.DEFAULT" />
                 <data android:mimeType="text/plain" />

         </intent-filter>

</activity>
```

A continuación, se muestra el código sobre la lectura de datos de una etiqueta NFC.

```
...
if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())){// Read NFC card
        NdefMessage[] message = getNdefMessages(getIntent());
        for ( int i = 0; i< message.length; i++)
              for(int j=0; j<message[0].getRecords().length;j++) {
                  NdefRecord record = message[i].getRecords()[j];
                  statusByte = record.getPayload()[0];
                  int languageCodeLength = statusByte & 0x3F;
                  String languageCode = new String( record.getPayload(), 1, languageCodeLength,
                  Charset.forName("UTF-8"));
                  int isUTF8 = statusByte-languageCodeLength;
                  if(isUTF8 == 0x00) {
                      payload = new String( record.getPayload(), 1+languageCodeLength, rec
                      ord.getPayload().length-1-languageCodeLength, Charset.forName("UTF-8"));
                  }
                  else if (isUTF8==-0x80) {
                       payload = new String( record.getPayload(), 1+languageCodeLength,
                       record.getPayload().length-1-languageCodeLength,Charset.forName("UTF-
                      16"));
                  }
                  myText.append((j+1) + "th. Record Tnf: " + record.getTnf() + "\n");
                  myText.append((j+1) + "th. Record type: "+ new String(record.getType()) + "\n");
                  myText.append((j+1) + "th. Record id: " + new String(record.getId()) + "\n");
                  myText.append((j+1) + "th. Record payload: " + payload + "\n");
               }
        }
}
...
```

En el código anterior se realizan varias cosas. Primero, se detecta si el intent entrante se corresponde con el provocado al acercar el dispositivo móvil a una etiqueta NFC. En caso de ser así, se almacenaría los datos del intent entrante en una estructura de datos NDEF. Por último, se recorrerían los diferentes registros que componen el mensaje leyendo la carga útil de los mismos. Dicha carga útil puede estar codificada en UTF-8 o UTF-16.

Finalmente, se realiza la operación de búsqueda de datos en la base de datos a partir de la carga útil leída con anterioridad. El código es el siguiente:

```
...
else { //Search the database
        SQLiteDatabase db = myDatabase.getWritableDatabase();
        String[] campos = new String[] {"*"};
        String[] args = new String[] {payload};
        Cursor c = db.query("Log", campos, "objetoPadre=?", args, null, null, null);
        if (c.moveToFirst()) {
            do {
                  String relacion_ = c.getString(0);
                  String objetoPadre = c.getString(1);
                  String objeto = c.getString(2);
                  String tiempo = c.getString(4);
                  String sincro = c.getString(5);
                  showtable(2, relacion_,objetoPadre, objeto, tiempo,sincro);
               } while(c.moveToNext());
}
...
```

Para buscar los datos que coinciden con la carga útil leída de las etiquetas NFC, se crea una instancia de la base de datos. Posteriormente se realiza una consulta sobre la misma, buscando coincidencias y mostrándolas en el caso que las hubiese mediante la función *showtable()*.

Lectura en posición portrait

Al acercar el teléfono móvil a la etiqueta NFC en posición portrait, se iniciará un almacenamiento en la base de datos del objeto leído. Este modo de lectura tiene dos casos diferenciados:

- Activado el modo de lectura agrupado. Este modo se activa o desactiva pulsando la tecla física de bajar volumen del dispositivo.

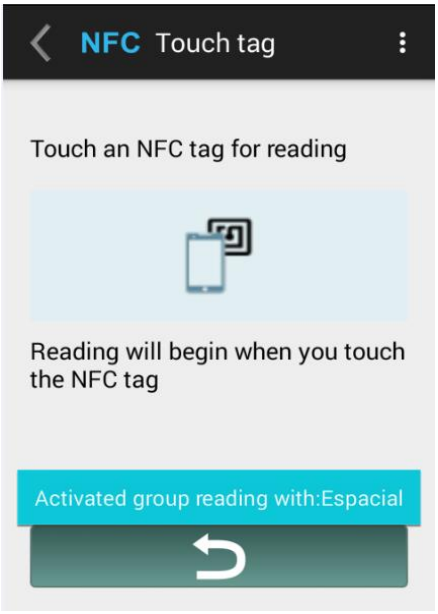


Figura 5.8. *Activado el modo lectura agrupado*

Si este modo está activado, las lectura realizadas a continuación de su activación estarán asociadas a un tipo de interacción concreta, y por tanto, a un tipo de relación. Existen dos casos:

- Primera Lectura. Si es la primera lectura, el objeto leído corresponderá al objeto padre o de referencia de la relación marcada por el tipo de interacción activada.

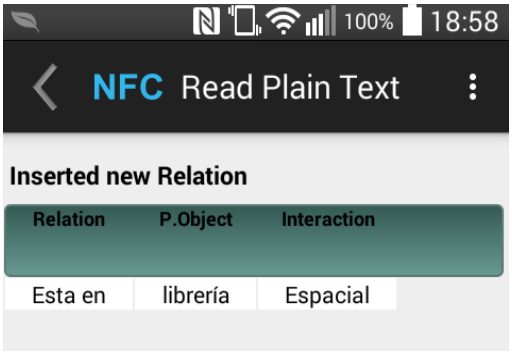


Figura 5.9. *Primera lectura agrupada*

Se puede observar que la primera lectura indica que el objeto leído es un objeto padre o de referencia para ese tipo de interacción con su correspondiente relación. En este caso el objeto padre o de referencia es *"Librería"*, el tipo de interacción es *"Espacial"* y el tipo relación es *"Esta en"*. La información se almacenará de manera adecuada en las tablas que correspondan en la base de datos.

- Lecturas siguientes. Para las siguientes lecturas, ya se tiene el tipo de relación definida con su objeto padre o de referencia, por lo cual pasarán a ser objetos hijos o asociados pertenecientes a la relación.

Figura 5.10. *Lectura agrupada sucesiva*

En este caso, el tipo de relación es *"Esta en"*, con el objeto padre o de referencia *"librería"*, se le ha añadido el objeto hijo o asociado *"libro1"*, con timestamp *"1430157517"* y sincronización *"0"*.

- Desactivado el modo de lectura agrupado. En este caso se pulsaría el botón físico de bajar el volumen de forma que se desactivara el modo de lectura agrupado.

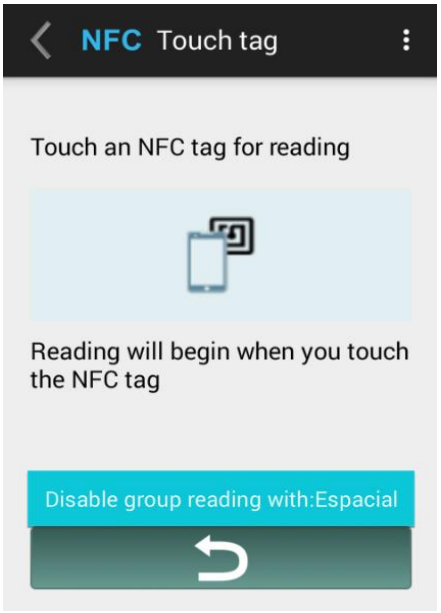


Figura 5.11. *Desactivado el modo lectura agrupado.*

De esta forma, al leer etiquetas NFC, los objetos leídos no tendrán ningún tipo de relación con otros, almacenándose directamente en la base de datos.

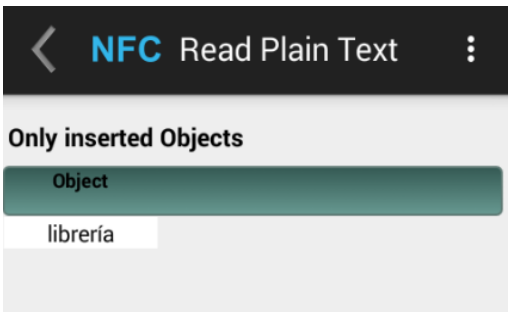


Figura 5.12. *Lectura de objeto*

En este caso se ha añadido a la base de datos el objeto *"Rey"*.

En esta actividad, destacan diferentes partes del código. Son las siguientes:

En la actividad de lectura de etiquetas, se ha sobrescrito el método de la tecla física de bajar volumen del Smartphone. Al usar esta tecla en esta actividad, se activaría y desactivaría el modo de lectura agrupado. El código es el siguiente:

```
//Select grouping mode through volume button
public boolean onKeyDown(int keyCode, KeyEvent event) {
...
       Vibrator v = (Vibrator) getSystemService(VIBRATOR_SERVICE);
       switch(keyCode) {

       case KeyEvent.KEYCODE_VOLUME_DOWN:
            sharedPreferences prefs = getSharedPrefe en
            ces("MyPreferences",Context.MODE_PRIVATE);
           Boolean bool = prefs.getBoolean("LMactive",false);
           String txt = prefs.getString("Lmultiple", "vacio");
           SharedPreferences.Editor editor = prefs.edit();
           if(bool == true) {
               editor.putBoolean("LMactive", false);
               editor.commit();
               Toast.makeText(this,"Disable group reading with:"+txt,
               Toast.LENGTH_SHORT).show();
               v.vibrate(500);
             }
           else {
              editor.putBoolean("LMactive", true);
              editor.commit();
              Toast.makeText(this,"Activated group reading with:"+txt,
              Toast.LENGTH_SHORT).show();
              v.vibrate(500);
             }
       return true;
       }
return super.onKeyDown(keyCode, event);
...
}
```

Por otro lado, y como ocurría en el caso de la lectura en posición landscape, al acercar el dispositivo móvil a una etiqueta NFC se iniciaría el proceso de captura de datos. La diferencia radica en que, en este caso, no se realizará una búsqueda en la base de datos sino una serie de inserciones de la carga útil leída en base a los casos presentados anteriormente. El código es el siguiente:

```
...
if(orientation == 1) { //Insertion in the database
   queryValues = new HashMap<String, String>();
   if(bool == true) {//If grouping is active
       tipo = prefs.getString("Lmultiple", "vacio");
       relacion = prefs.getString("Relacion","vacio");
       if(prefs.getBoolean("OPactive", false)==true) {//Inserting the parent object
          SharedPreferences.Editor editor = prefs.edit();
          editor.putBoolean("OPactive", false);
          editor.putString("OPadre",payload);
          editor.commit();
          queryValues.put("objeto",payload);
          myDatabase.insert(queryValues, "Objetos");
          queryValues.put("relacion",relacion);
          queryValues.put("interaccion",tipo);
          myDatabase.insert(queryValues,"RCruzadas");
          showtable(0,null, payload, null, null,null);
       }
       else { //Inserting the child object
          Long tsLong = System.currentTimeMillis()/1000;
          String timestamp = tsLong.toString();
          queryValues.put("objeto",payload);
          myDatabase.insert(queryValues,"Objetos");
          queryValues.put("relacion",relacion);
          queryValues.put("objetoPadre",prefs.getString("OPadre","SinPadre"));
          queryValues.put("interaccion",tipo);
          queryValues.put("tiempo",timestamp);
          queryValues.put("sincro","0");
          myDatabase.insert(queryValues,"Log");
          showtable(0, null,prefs.getString("OPadre","SinPadre"),payload,   timestamp,null);
       }
    }
    else { //If grouping is not active
      queryValues.put("objeto",payload);
      myDatabase.insert(queryValues,"Objetos");
      showtable(1, null,null, payload, null,null);
   }
}
...
```

En el código anterior, se diferencian los tipos de inserción en la base de datos en función del modo de lectura agrupado esté activado o no. Además, en cada caso, se llama a la función *showtable()* con el objetivo de mostrar los datos insertados.

5.2.1.2. ACTIVITY MENÚ ESCRITURA DE ETIQUETAS

Esta actividad es iniciada al pulsar el botón *"Write Tag"* de la actividad principal. Esta actividad tiene la función de seleccionar el tipo de escritura de datos a realizar.

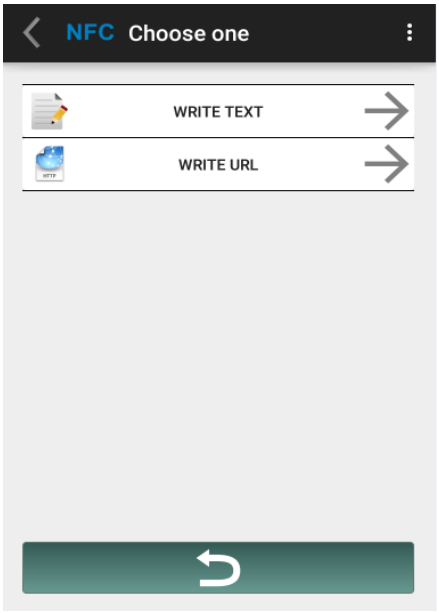


Figura 5.13. *Menú tipo de escritura*

La pantalla tiene una serie de botones mediante los cuales se puede seleccionar los tipos de escritura de datos posible.

- Botón Write Text. Este botón inicia la actividad de escritura de texto plano en la etiqueta NFC.
- Botón Write URL. Este botón inicia la actividad de escritura de URLs en la etiqueta NFC.
- Botón Back. Este botón cierra la actividad actual, devolviendo el control a la actividad principal.

Se puede observar que esta actividad tiene el único objeto de servir de selector de escritura de datos en la etiquetas NFC. No se tiene implementado código de especial relevancia más allá de la funcionalidad de los propios botones.

5.2.1.3. ACTIVITY ESCRITURA TEXTO PLANO/URL

Esta actividad se inicia al seleccionar algunas de las opciones de escritura disponibles en la actividad menú escritura. En esta actividad tiene la funcionalidad de escribir texto plano o URL, en función de lo elegido, en las etiquetas NFC.

Figura 5.14. *Escritura de texto plano*

Figura 5.15. *Escritura de texto plano 2*

En las figuras anteriores, se muestra la apariencia de la actividad de escritura de texto plano. En el campo de texto se escribe lo que se desea almacenar en la etiqueta NFC. Posteriormente se pulsa sobre el botón *"Save & Write"* y se acercaría el dispositivo móvil a la etiqueta NFC. La escritura sería inmediata en la etiqueta NFC. En el caso de haber seleccionado la escritura de URL el proceso sería exactamente igual al descrito.

En esta actividad destacan varias partes del código en referencia a la escritura. Tras el proceso de captura de texto o URL a escribir en la tarjeta NFC, es necesario sobrescribir diferentes métodos. Son los siguientes:

```
...
@Override //Set up the NDEF message
public void onNewIntent(Intent intent) {
        Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        Locale locale= new Locale("en","US");
        byte[] langBytes = locale.getLanguage().getBytes(Charset.forName("US-ASCII"));
        boolean encodeInUtf8=false;
        Charset utfEncoding = encodeInUtf8 ? Charset.forName("UTF-8") : Charset.forName("UTF-16");
        int utfBit = encodeInUtf8 ? 0 : (1 << 7);
        char status = (char) (utfBit + langBytes.length);
        byte[] textBytes = data.getBytes(utfEncoding);
        byte[] data = new byte[1 + langBytes.length + textBytes.length]; data[0] = (byte) status;
        System.arraycopy(langBytes, 0, data, 1, langBytes.length);
        System.arraycopy(textBytes, 0, data, 1 + langBytes.length, textBytes.length);
```

```
        NdefRecord textRecord = new NdefRecord(NdefRecord.TNF_WELL_KNOWN, NdefRecord.RTD_TEXT, new
        byte[0], data);
        NdefMessage newMessage= new NdefMessage(new NdefRecord[] { textRecord });
        writeNdefMessageToTag(newMessage, tag);//Write in NFC tag
}
...
```

Sobrescribiendo el método *onNewIntent()* se consigue el efecto de captar el descubrimiento de una etiqueta NFC cercana al dispositivo móvil. Este descubrimiento generará un intent que se aprovechará para enviar los datos a escribir. Como se puede observar, en el código se forma la estructura de datos a escribir y se encapsula en un registro NDEF, que a su vez este está encapsulado en un mensaje NDEF. Una vez formado el mensaje, se llama a la función *writeNdefMessageToTag()* la cual iniciará la escritura en la etiqueta NFC.

Además del método anterior, se necesita sobrescribir los métodos *onResume()* y *onPause()*.

```
@Override
public void onPause() {
        super.onPause();
        if(mNfcAdapter != null)
        mNfcAdapter.disableForegroundDispatch(this);
}
@Override
public void onResume() {//prioritize activity in the foreground
        super.onResume();
        if(mNfcAdapter != null)
                mNfcAdapter.enableForegroundDispatch(this, mPendingIntent,mFilters,mTechLists);

        }
```

Al sobrescribir el método *onResume()* activando el ForengroundDispatch lo que se consigue es dar prioridad a la actividad frente a otras con respecto a la recepción o descubrimiento de una etiqueta NFC cercana. De esta manera evitaríamos que otra aplicación se abriera y leyera el contenido de la etiqueta NFC cuando se estaba en medio del proceso de escritura de la etiqueta. Sobrescribiendo el método *onPause()* desactivando el ForengroudDispatch, se anularía la prioridad recibida en el método *onResume()*.

Por último, se necesita declarar y asignar una serie de valores a diferentes variables.

```
...
private void setupForenground() {//Set up forengroud
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        mPendingIntent = PendingIntent.getActivity(this, 0, new Intent(this,
        getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
        IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
        mFilters = new IntentFilter[] {ndef, };
        mTechLists = new String[][] { new String[] { Ndef.class.getName() }, new String[] { Ndef-
        Formatable.class.getName() }};
}
...
```

Con este método se consigue obtener acceso al adaptador NFC del dispositivo móvil, y seleccionar la lista de tipos de etiquetas que la aplicación podrá soportar.

5.2.1.4. ACTIVITY RELACIONES

Esta actividad es iniciada al pulsar sobre el botón *"Create Relation"* de la actividad principal. Su funcionalidad consiste en crear o eliminar las diferentes interacciones-relaciones utilizadas para los modos de lectura agrupados.

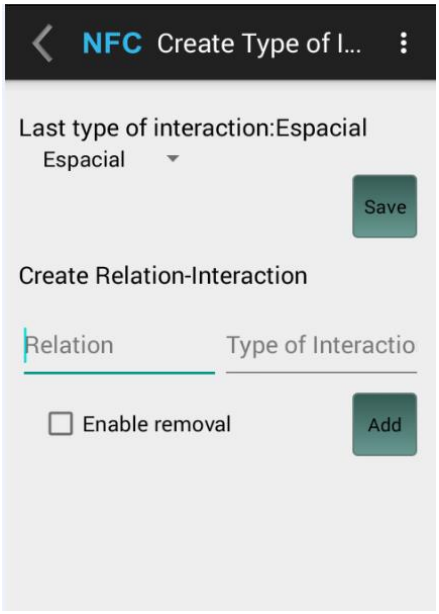


Figura 5.16. *Crear interacción-relación*

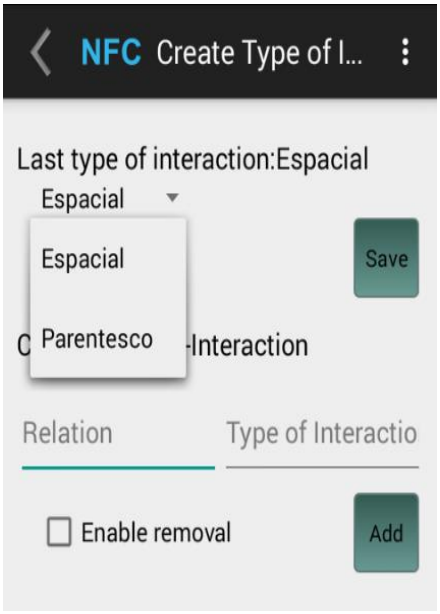


Figura 5.17. *Seleccionar interacción*

En las figuras anteriores, se observan los casos de crear nuevas relaciones – interacciones y seleccionar el tipo de interacción deseado.

En el caso de la creación de nuevas relaciones – interacciones, estas estarán almacenadas en una base de datos local a la aplicación. La inserción se realizaría al añadir la relación junto a su interacción en los campos de texto. Una vez introducida está información, al pulsar sobre el botón *"Add"*, se añadiría a la tabla local la nueva información.

En el segundo caso, correspondiente a la selección del tipo de interacción a utilizar, se elegiría la interacción buscada seleccionando de las disponibles en la lista desplegable. Al pulsar sobre el botón *"Save"*, se iniciaría la actividad de lectura de etiquetas NFC con el modo agrupado activado por defecto.

Existe un tercer caso correspondiente a la eliminación de las relaciones- interacciones de la tabla local. Esta operación se daría tras la selección del check box *"Enable removal"* como se muestra en la figura a continuación.

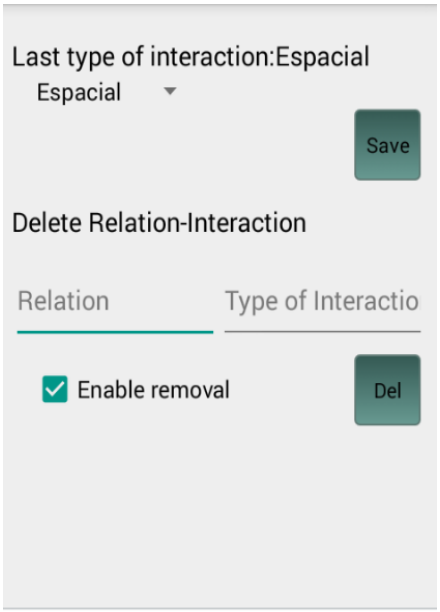


Figura 5.18. *Eliminar relación-interacción*

Si el check box está activado, solo se tendría que escribir la relación-interacción a eliminar en la tabla local y esta dejaría de aparecer en la lista desplegable. Este proceso se llevaría a cabo tras pulsar el botón *"Del"* anteriormente *"Add"* de la actividad.

En esta actividad destaca el código acerca de la inserción o eliminación de las relaciones-interacciones.

En el caso de las inserciones o eliminaciones de relaciones – interacciones el código es el siguiente:

```
...
if(box.isChecked())   {
   if(c.getCount() > 0) {
      queryValues.clear();
      queryValues.put("interaccion",interaction.getText().toString());
      myDatabase.delete(queryValues,"Conf_spinner");
      myDatabase.close();
      reiniciar();
   }
else {
   if(c.getCount() ==0) {
      queryValues.clear();
      queryValues.put("relacion",relation.getText().toString());
      queryValues.put("interaccion",interaction.getText().toString());
      myDatabase.insert(queryValues, "Conf_spinner");
      myDatabase.insert(queryValues, "Interaccion");
      myDatabase.insert(queryValues,"Relacion");
      myDatabase.close();
      reiniciar();
 }
}
...
```

Mediante una clausula if else se comprueba si el check box está activado o no y por tanto si está en modo inserción o eliminación. En el caso de la eliminación, se comprueba que la relación-interacción a eliminar existe y se reinicia la interfaz para recargar la lista desplegable sin la

interacción. En el caso opuesto, lo que se comprueba es que la relación-interacción a insertar no existe previamente antes de introducirla.

5.2.1.5. ACTIVITY SINCRONIZACIÓN

Esta actividad se inicia al pulsar sobre el botón de sincronización de la actividad principal. Al pulsar sobre este botón se inicia el proceso de sincronización bidireccional entre la base de datos del dispositivo móvil y la base de datos del servidor. Se puede observar a continuación el resultado de pulsar dicho botón.

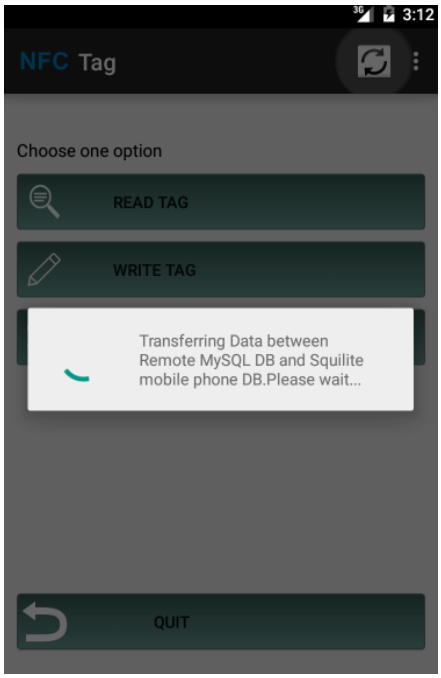


Figura 5.19. *Transferencia de datos en la sincronización*

Relation	P.Object	Object	Timestamp	Sync
Esta en	libreria	libro	1427558024	1
Esta en	libreria	libro1	1427558026	1
Familiar	libreria	libro	1427558043	1
Familiar	libreria	libro1	1427558046	1
Jerarquia	Rey	Vasallo	1427558122	1
Esta en	libreria	libreria	1427580742	1
Esta en	libreria	libreria	1427746543	1
Familiar	libro1	libro	1427991136	1
Esta en	libreria	libreria	1428756867	1
Esta en	libreria	libro1	1428756927	1
Esta en	libreria	libreria	1428756950	1
Esta en	libreria	pteee	1428756960	1
Esta en	libreria	pteee	1428757033	1
Esta en	libreria	Rey	1428757057	1
Esta en	libreria	Rey	1428757064	1
Esta en	libreria	Rey	1428757072	1

Figura 5.20. *Datos sincronizados*

Este proceso tarda un tiempo en completarse, por lo que se ha provisto de un dialogo emergente, *Figura 5.19*., mientras se realiza dicha operación. Una vez completada esta operación, se mostrará la tabla Log con los datos que contiene. Si la sincronización se ha realizado correctamente, todos los datos deberían tener un 1 en el campo *"Sync"*. En Caso contrario, el proceso de sincronización ha fallado.

En esta actividad, solo es reseñable el código acerca de obtener y mostrar los datos. El proceso de sincronización es realizado en la actividad principal mediante las funciones *syncMySQLDBSQLite()* y *syncSQLiteMySQLDB()* como ya se ha especificado en el apartado *5.2.1. Interfaz del Smartphone*.

El código acerca de mostrar y obtener los datos es el siguiente:

```
...
table_layout = (TableLayout) findViewById(R.id.tableLayout1);
Cursor c = db.query("Log", campos, null, null, null, null, null);
for (int i = 0; i < rows; i++) {
      TableRow row = new TableRow(this);
       row.setLayoutParams(new Layout Params(LayoutParams.MATCH_PARENT, Layout Params.
       WRAP_CONTENT));

       for (int j = 0; j < cols; j++) {
              if(j!=3) {
                  LayoutParams layoutParams = new Layout-
                  Params(LayoutParams.WRAP_CONTENT,LayoutParams.WRAP_CONTENT);
                  layoutParams.setMargins(1, 1, 1, 1);
                  TextView tv = new TextView(this);
                  tv.setLayoutParams(layoutParams);
                  tv.setBackgroundColor(Color.WHITE);
                  tv.setTextSize(14);
                  tv.setGravity(Gravity.CENTER);
                  tv.setText(c.getString(j));
                  tv.setTextColor(Color.BLACK);
                  row.addView(tv);
              }
       }
       c.moveToNext();
       table_layout.addView(row);
}
...
```

Con este código se pretende crear una tabla dinámica con los datos existentes en la tabla *"Log"*. Para cada dato de cada fila se le asignan una serie de propiedades como el tamaño de letra, color de fondo y demás, exceptuando la fila sobre el tipo de interacción, que está siendo obviada y por lo tanto no visualizada.

5.2.2. INTERFAZ DEL SERVIDOR

El servidor tendrá su propia interfaz con el objetivo de poder manipular y visualizar los datos de forma sencilla para el usuario. La interacción con dicha interfaz se realizará mediante el uso del navegador, teniendo a disposición del usuario tres ventanas diferenciadas: **La ventana de inserción**, **la ventana de eliminación** y **la ventana de mostrar datos**.

En los siguientes puntos de este capítulo se explicarán la funcionalidad y scripts PHP desarrollados para la interfaz web del servidor.

5.2.2.1. VENTANA INSERCIÓN

En esta ventana se insertan datos a la base de datos del servidor. Está compuesta por un formulario con cuatro campos a rellenar por el usuario. En la *Figura 5.21* que se encuentra a continuación se puede observar dicho formulario.

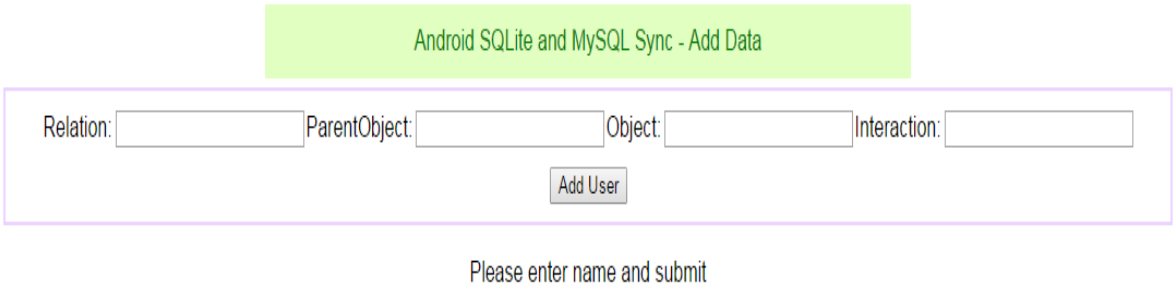


Figura 5.21. *Formulario inserción datos B.D. del servidor*

Los campos a rellenar por el usuario son:

- Relation. Este campo se debe rellenar con el tipo de relación a insertar.
- Parent Objetct. Este campo se debe rellenar con el nombre del objeto de referencia o padre de la relación insertada.
- Object. Este campo se debe rellenar con el nombre del objeto a insertar en la relación.
- Interaction. Este campo se debe rellenar con el tipo de interacción a insertar.

Se puede observar que aún faltan dos campos por dar valor que no se han añadido en el formulario. Estos campos son el timestamp y la sincronización.

En el caso del timestamp, se añadirá en la inserción automáticamente tomando como valor el instante de tiempo en el que se realizó la inserción en la base de datos.

En el caso de la sincronización, automáticamente tomará el valor 0 (no sincronizado). De esta forma nos aseguramos que se tomen estos valores en el proceso de sincronización.

En cuanto al código destaca la parte relativa a la inserción y envío de un aviso de sincronización al dispositivo móvil desde el servidor. El código es el siguiente:

```
<?php
include_once './db_functions.php';
//Create Object for DB_Functions class
if(isset($_POST["relation"]) && !empty($_POST["relation"])&& isset($_POST["parentObject"]) &&
!empty($_POST["parentObject"])&& isset($_POST["object"]) && !empty($_POST["object"])&& is-
set($_POST["interaction"]) && !empty($_POST["interaction"])){
```

En primer lugar, comprobamos que todos los campos del formulario están iniciados y no son nulo mediante las funciones isset y empty.

```
    $db = new DB_Functions();
    //Store User into MySQL DB
    $relation = $_POST["relation"];
```

```
    $pObject = $_POST["parentObject"];
    $object = $_POST["object"];
    $interaction = $_POST["interaction"];
    $fecha = new DateTime();
    $time=$fecha->getTimestamp();
    $timestamp=idate('U', $time);
    $res = $db->storeUser($relation,$pObject,$object,$interaction,$timestamp,"0");
```

En caso de haber rellenado de forma correcta el formulario, se inicia una conexión con la base de datos. Se recogen los datos del formulario y se calcula el timestamp a insertar. Seguidamente, se realiza una inserción en la base de datos con los diferentes datos obtenidos.

```
    if($res){
        //Post message to GCM when submitted
        $pushStatus = "New Data";
        $pushMessage = "New Data";
        $gcmRegID  = file_get_contents("GCMRegId.txt");
        if (isset($gcmRegID) && isset( $pushMessage)) {
            $gcmRegIds = array($gcmRegID);
            $message = array("m" => $pushMessage);
            $pushStatus = $db->sendMessageThroughGCM($gcmRegIds, $message);
        }
```

Si la inserción fue correcta en la base de datos, se envía una notificación al dispositivo móvil mediante el GCM. Para ello se llama a la función sendMessageThroughGCM() con el RegID del dispositivo móvil y el mensaje a enviar.

```
?>
<div id="msg">Insertion successful.GCM status:<?php echo $pushStatus; ?></div>

<?php }else{ ?>
<div id="msg">Insertion failed</div>
<?php }
} else{ ?>
<div id="msg">Please enter name and submit</div>
<?php }

?>
```

Por último, se muestra un mensaje en la interfaz web sobre el resultado de la inserción y envío del mensaje al dispositivo móvil.

5.2.2.2. VENTANA ELIMINACIÓN

En esta ventana, se eliminan datos de la base de datos del servidor. Esta ventana está compuesta por un formulario como en el caso de la eliminación, pero esta vez con un único campo a rellenar.

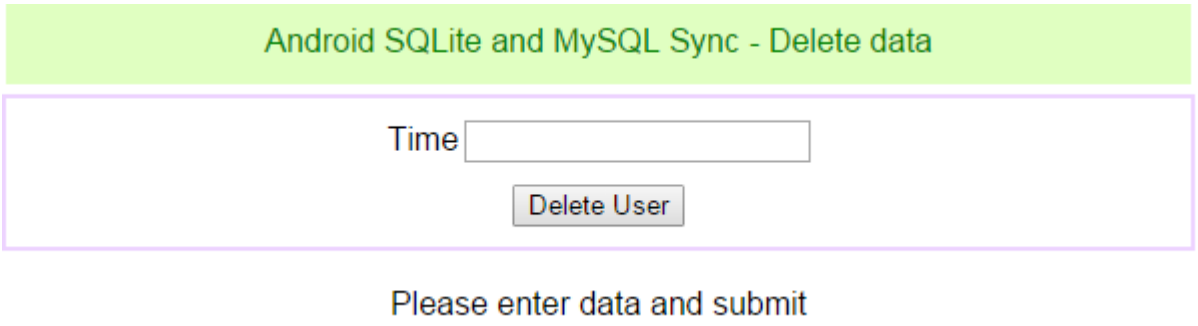


Figura 5.22. *Formulario eliminar datos B.D. del servidor*

El campo a rellenar es el del timestamp o time. Como el instante de inserción es único e irrepetible, solo se necesita seleccionar por este campo los datos a eliminar.

En cuanto al código, siguiendo la línea del caso de la inserción, destaca la eliminación y envió de mensaje al dispositivo móvil. El código es el siguiente:

```
<?php
include_once './db_functions.php';
//Create Object for DB_Functions clas
if(isset($_POST["tiempo"]) && !empty($_POST["tiempo"])){
```

En primer lugar se comprueba que el campo a rellenar no esté vacío ni sea nulo.

```
    $db = new DB_Functions();
    $tiempo = $_POST["tiempo"];
    $res = $db->delete($tiempo);
```

En segundo lugar, se inicia una conexión con la base de datos y se eliminan los datos señalados con el timestamp.

```
    if($res){

        $pushMessage = "Deleted Data";
        $gcmRegID  = file_get_contents("GCMRegId.txt");
        if (isset($gcmRegID) && isset( $pushMessage)) {
            $gcmRegIds = array($gcmRegID);
            $message = array("m" => $pushMessage);
            $pushStatus = $db->sendMessageThroughGCM($gcmRegIds, $message);
        }
```

Si el proceso de eliminación fue correcto, se envía un mensaje al dispositivo móvil de igual forma que en el caso de la inserción.

```
?>
<div id="msg">Deleted successfully.GCM status:<?php echo $pushStatus; ?></div>
<?php }else{ ?>
<div id="msg">Deleted failed</div>
<?php }
}  else{ ?>
<div id="msg">Please enter data and submit</div>
<?php }

        ?>
```

Por último, se muestra un mensaje en la interfaz web del éxito o error de la operación de eliminación y envío de mensaje.

5.2.2.3. VENTANA MOSTRAR DATOS

En esta ventana se muestran los datos contenidos en la base de datos del servidor. Estos datos pueden estar en diferentes estados de sincronización, siendo los estados los siguientes: *"sincronizado"*, *"no sincronizado"* y *"a la espera de eliminar"*. Estos estados son representados en la columna de sincronización con los colores blanco, negro y verde. La interfaz será recargada automáticamente cada dos segundos para siempre mostrar los datos actuales.

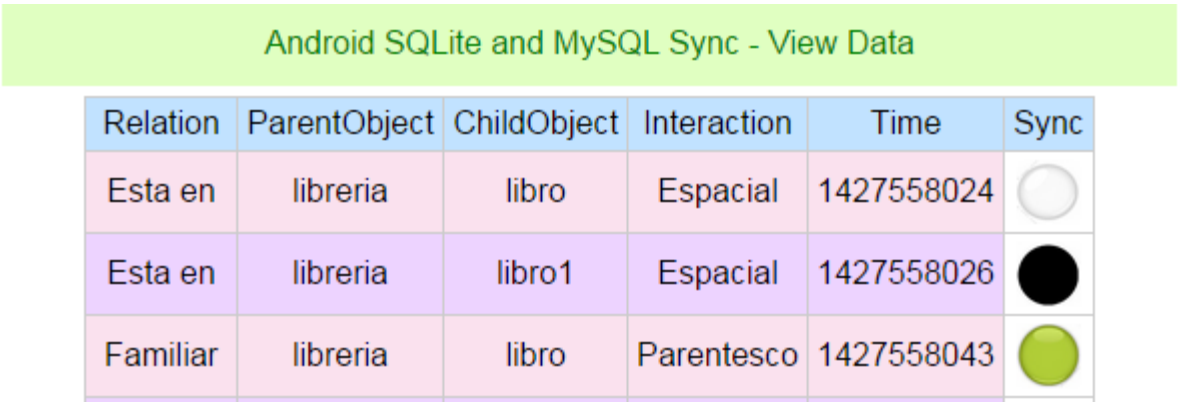


Figura 5.23. *Mostrar datos B.D. del servidor*

En la *Figura 5.22.*, se observa una tabla con los diferentes campos y estados de sincronización a mostrar. Los campos son los siguientes:

- Relation. Muestra el tipo de relación.
- Parent Object. Muestra el objeto padre o de referencia de la relación.
- ChildObject. Muestra los diferentes objetos asociados a la relación.

- Time. Muestra el timestamp o instante de inserción.
- Sync. Muestra el estado de sincronización de la relación. Los tipos de estados son:

    - No sincronizado. Representado por el color blanco. Son los datos que aún no se han sincronizado con el dispositivo móvil.
    - Sincronizado. Representado por el color verde. Son los datos que están sincronizados actualmente entre el dispositivo móvil.
    - A la espera de eliminar. Representa los datos que han sido eliminados en la base de datos y están a la espera de eliminar en el dispositivo móvil. Estos datos estuvieron sincronizados en algún momento, en caso contrario, la eliminación se realizaría directamente sin necesidad de comprobar en el dispositivo móvil su eliminación.

En cuanto al código destaca el mostrar los datos y la recarga de la interfaz. El código es el siguiente:

Para la recarga de la interfaz cada dos segundos, se ha utilizado un sencillo script en JavaScript. Es el siguiente:

```
<script>
     var val= setInterval(function(){
     location.reload();
     },2000);
</script>
```

En cuanto al código de mostrar los datos, es el siguiente:

```
<?php
include_once 'db_functions.php';
$db = new DB_Functions();
$data= $db->getAllData();
```

En primer lugar se conecta con la base de datos y se traen todos los datos a mostrar.

```
if ($data != false)
    $no_of_rows = mysql_num_rows($rows);
else
    $no_of_rows = 0;

        ?>
<?php
if ($no_of_rows > 0) {
        ?>
```

A continuación se comprueba si existen datos a mostrar, es decir, se comprueba si la consulta ha sido devuelta vacía.

```
        <table>
            <tr
id="header"><td>Relation</td><td>ParentObject</td><td>ChildObject</td><td>Interaction</td><td>Tim
e</td><td>Sync</td></tr>
            <?php
    while ($row = mysql_fetch_array($data)) {
            ?>
            <tr>
                <td><span><?php echo $row["relacion"] ?></span></td>
                <td><span><?php echo $row["objetoPadre"] ?></span></td>
                <td><span><?php echo $row["objeto"] ?></span></td>
                <td><span><?php echo $row["interaccion"] ?></span></td>
                <td><span><?php echo $row["tiempo"] ?></span></td>
                <td id="sync"><span>
                    <?php
                if($row["sincro"]=="1")
            {
                echo "<img src='img/green.png'/>";
            }else if($row["sincro"]=="2")
            {
                echo "<img src='img/black.png'/>";
            }else  {
                echo "<img src='img/white.png'/>";
            }
                    ?></span></td>
            </tr>
            <?php } ?>
        </table>
        <?php }else{ ?>
        <div id="norecord">
            No records in MySQL DB
        </div>
        <?php } ?>
    </body>
</html>
```

Finalmente, si existen datos a mostrar, se recorren los datos devueltos y se van insertando de manera ordenada en una tabla dinámicamente construida. Destacar que en el caso del campo de la sincronización se comprobará su valor para insertar un color representativo u otro. Un 0 coincidiría con el estado *"no sincronizado"* (blanco), un 1 coincidiría con el estado *"sincronizado"* (verde) y un 2 coincidiría con el estado *"a la esperara de eliminar"*.

5.3. PRUEBAS

En el desarrollo de los diferentes elementos que conforman el total del proyecto, se realizaron diferentes tipos de pruebas. Son las siguientes:

Pruebas de la aplicación móvil con diferentes pantallas

Estas pruebas tenían la intención de comprobar el buen comportamiento de los elementos gráficos de la aplicación móvil en diferentes pantallas. Se probó la aplicación en diferentes dispositivos físicos y virtuales. Concretamente:

- Se instaló la aplicación en un dispositivo físico de 4,5 pulgadas de tamaño. Este dispositivo fue el utilizado durante todo el desarrollo de la aplicación como base.

- Dispositivos emulados.

    - Dispositivo de 3,5 pulgadas de tamaño de pantalla.
    - Dispositivo de 4 pulgadas de tamaño de pantalla.
    - Dispositivo de 5 pulgadas de tamaño de pantalla.

La interfaz se adaptó en todo momento a los diferentes tipos de pantalla, sin provocar en ella deformaciones o solapamientos en ninguna de las diferentes actividades en posición landscape o portrait.

Versiones de Android soportadas

En cuanto a las diferentes versiones de Android que soportan la aplicación tenemos lo siguiente:

La aplicación tiene como limitación utilizar una versión de Android superior a la 2.3. Esta restricción viene dada debido al uso del servicio de Google, *"Google Cloud Messaging"* y la APIs necesaria para poder utilizar, principalmente, el NFC móvil y en menor medida, elementos de la interfaz de la aplicación.

Actualmente, la mayor parte de dispositivos Android existentes tienen una versión de Android comprendida entre la 2.3 y la reciente 5.1. Este hecho provoca que la aplicación pueda funcionar en la inmensa mayoría de dispositivos con Android existentes que cuenten con la tecnología de comunicación NFC.

6. CONCLUSIÓN Y TRABAJOS FUTUROS

Internet de las Cosas está cada vez más presente en la vida cotidiana de millones de personas. Hoy en día es común ver cómo casi cualquier objeto puede interactuar de una forma u otra con el usuario aportando nuevas utilidades o información sobre el mismo. Desde los teléfonos móviles inteligentes hasta las casas, absolutamente todo tiende, en un futuro próximo, a estar conectado a la red, y por lo tanto, haciendo posible la aparición de nuevas funcionalidades para el usuario.

Con este proyecto se ha demostrado cómo es posible dotar de nuevas funcionalidades e información a objetos que a priori tenían unas limitaciones en cuanto a interacción con el usuario se refiere. Asociando etiquetas NFC y utilizando un objeto de tan común uso como es un Smartphone, se ha conseguido llegar a crear nuevas formas de interacción y captación de información sobre objetos de uso cotidiano, dando lugar a una imagen virtual del mismo. La aplicación muestra resultados satisfactorios, dando mayor información y por tanto usabilidad al usuario en sus interacciones diarias con los diferentes objetos de uso cotidiano. Como un trabajo o desarrollo futuro, se podrían implementar nuevas formas de captación de información de objetos, creando nuevos posibles *"casos de uso"* y añadiendo mayor cantidad y calidad de información en los ya existentes.

Es obvio que aún este campo tiene mucho potencial por explotar. A medida que la tecnología avance y los costos asociados al hardware necesario para este tipo de tecnología sean inferiores, Internet de las Cosas ganará popularidad y por tanto tendrá un mayor desarrollo. Este proyecto muestra una de las vías de desarrollo existentes en Internet de las Cosas, además de una serie de casos de uso, que lejos de estar cerrada, se puede ampliar y adaptar a multitud de nuevas necesidades.

7. CONCLUSIONS AND FUTURE WORKS

Internet of Things is day after day more present in people's life. Nowadays is common to see how almost every object can interact in one way or another with the final user. This leads to discovering new usages and ways of getting more information about the object. From smartphones to houses, absolutely everything is going to be connected to the Internet in a close future, providing new features to the final user.

In this project we have displayed how it is possible to provide new features and information to objects that at first sight had some limitations when referring to interaction with the final users. By means of NFC tags and using such a common use object as a Smartphone, we have created new ways of interaction and information collection on everyday objects, making a virtual image of the object. The app shows good results, giving more information, therefore more usability to the final users in their daily interactions with different everyday objects. As a future development or Project, it might be implemented new ways to catch object's information, creating new potential "usage cases" and adding more quality and quantity of information in the existing ones.

It's obvious that these technologies have a lot of potential to be exploited. As long as these technologies continues improving and the costs related to the involved resources decreases, Internet of Things will earn popularity, so it will have a bigger development. This project shows one of the possible ways to make the Internet of Things more pervasive. In addition, we have presented a set of use cases that can be easily extended and adapted to a lot of new necessities.

8. PRESUPUESTO

El proyecto se ha realizado utilizando exclusivamente herramienta de software libre. Los únicos costos vendrían derivados del dispositivo móvil físico, las etiquetas NFC y la mano de obra.

Referencia	Cantidad	Coste
Software	10	0 €
Dispositivo móvil	1	150€
Etiquetas NFC	30	7€
Mano de obra	1	4050€
Total		4207€

Tabla 8.1. *Presupuesto del proyecto*

Justificación del presupuesto

El proyecto se ha realizado utilizando diferentes elementos físicos con su correspondiente coste asociado:

- El precio mínimo de un dispositivo móvil que disponga de la tecnología de comunicación NFC ronda los 150 € de media.
- Las etiquetas NFC se venden normalmente en pack de 30 unidades. Las etiquetas NFC utilizadas en este proyecto tiene un costo de 7 € de media por pack.

En cuanto a la mano de obra, se ha calculado en base a lo que cobraría un ingeniero informático por el desarrollo del mismo utilizando su propio PC para desarrollar. Suponiendo que se le pagara 18 € por hora, siendo este sueldo por debajo de lo que debería cobrar un ingeniero informático, y que ha trabajo 45 días en el proyecto, el costo asociado a la mano de obra es de 4050€.

Finalmente, el sumatorio del costo de los elementos físicos utilizados y la mano de obra da lugar a los 4207 € presupuestados en la *Tabla 8.1. Presupuesto del proyecto.*

BIBLIOGRAFÍA

[1] Internet de las Cosas
Wikipedia
http://en.wikipedia.org/wiki/Internet_of_Things

[2] Internet de las Cosas.
Página web Cisco
http://www.cisco.com/web/LA/soluciones/executive/assets/pdf/internet-of-things-iot-ibsg.pdf

[3] Libro de consulta principal
Near Field Communication y Radio Frequency Identification
Professional NFC Application Development for Android, Coskun, Vedat Ok, Kerem Ozdenizci, Busra (2014), ISBN: 978-1-118-38009-3. Págs. 3-7, 10-22,24-54

[4] Near Field Communication (NFC)

Wikipedia
http://es.wikipedia.org/wiki/Near_field_communication

Páginas webs sobre NFC
http://www.nfc.cc/technology/nfc/

Página web oficial NFC-forum
http://nfc-forum.org/

[5] Artículo sobre Near Field Communication. Págs. 67-73.
http://www.infosys.com/infosys-labs/publications/Documents/winning-it.pdf#page=69

[6] Formato NDEF
PDF de NFC-Forum
http://www.eet-china.com/ARTICLES/2006AUG/PDF/NFCForum-TS-NDEF.pdf

[7] Record Type Definition, NDEF. Pág. 6.
http://www.egov.ufsc.br/portal/sites/default/files/cdn_nfc_final.pdf

[8] Media-Type, NDEF.
https://www.ietf.org/rfc/rfc2046.txt

[9] Absolute URI, NDEF.
https://www.ietf.org/rfc/rfc3986.txt

[10] External Type, NDEF. Pág.6.
http://goo.gl/tJxpxa

[11] Identificación por radiofrecuencia (RFID).
Wikipedia. Versión inglés
http://en.wikipedia.org/wiki/Radio-frequency_identification
Wikipedia. Versión español
http://es.wikipedia.org/wiki/RFID

[12] Historia sobre Internet de las Cosas
Páginas webs
http://www.quees.info/que-es-internet-de-las-cosas.html
http://www.sorayapaniagua.com/2012/04/15/un-poco-de-historia-sobre-internet-de-las-cosas/

[13] Historia RFID
Páginas Webs
http://www.infotecarios.com/viejas-tecnologias-con-nuevos-usos-en-gestion-tecnologica-identificacion-por-radiofrecuencia-rfid/
http://www.pandaid.com/el-origen-de-la-tecnologia-rfid/

[14] Historia NFC
Páginas webs
http://histinf.blogs.upv.es/2012/11/21/nfc/
http://www.junsuft.com/2012/11/tecnologia-nfc-near-field-communication.html

[15] Documentación sobre eXtreme Programming
Wikipedia
http://en.wikipedia.org/wiki/Extreme_programming
Página web
http://programacionextrema.tripod.com/fases.htm

[16] Estándar ISO 14443
Wikipedia
http://es.wikipedia.org/wiki/ISO_14443
PDF acerca de la ISO
http://www.atmel.com/Images/doc2056.pdf

[17] Estándar ISO 15693
Wikipedia
http://en.wikipedia.org/wiki/ISO/IEC_15693

[18] Bases de datos distribuidas, BDD.
Wikipedia
http://es.wikipedia.org/wiki/Bases_de_datos_distribuidas
Páginas Webs
https://modelosbd2012t1.wordpress.com/2012/03/08/bases-de-datos-distribuidas/
https://basesdedatos.wordpress.com/7-bases-de-datos-distribuidas/

[19] Documentación sobre Android
Wikipedia
http://es.wikipedia.org/wiki/Android
Página web oficial
https://developer.Android.com/guide/index.html
Otras Páginas webs
http://www.monografias.com/trabajos101/sistema-operativo-Android/sistema-operativo-Android.shtml
http://www.xatakAndroid.com/sistema-operativo/que-es-Android
http://www.Androidcurso.com/index.php/tutoriales-Android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-actividad
http://hipertextual.com/archivo/2010/08/ios-sdk-vs-Android-sdk-i/

[20] Google Cloud Messaging
Páginas Webs
http://www.sgoliver.net/blog/notificaciones-push-android-google-cloud-messaging-gcm-introduccion/
http://balusoft.net/2013/03/17/push-notifications-en-android-a-traves-de-google-cloud-messaging/
Página Web oficial
https://developer.android.com/google/gcm/index.html

APÉNDICE A: MANUAL DEL USUARIO

Este apéndice tendrá la función de manual de usuario, explicando la interfaz gráfica tanto de la aplicación móvil como de la interfaz web del servidor.

En primer lugar, comenzaremos con la interfaz gráfica de la aplicación móvil. Dicha interfaz está dividida por actividades, cada una de ellas con una funcionalidad diferenciada.

En segundo lugar, continuaremos explicando la interfaz web del servidor. Dicha interfaz está compuesta por una serie de ventanas con las que se operará sobre la base de datos del servidor.

INTERFAZ DE LA APLICACIÓN MÓVIL

Una vez instalada la aplicación en el móvil, esta será accesible mediante el icono que está dentro de un círculo rojo en la siguiente figura.

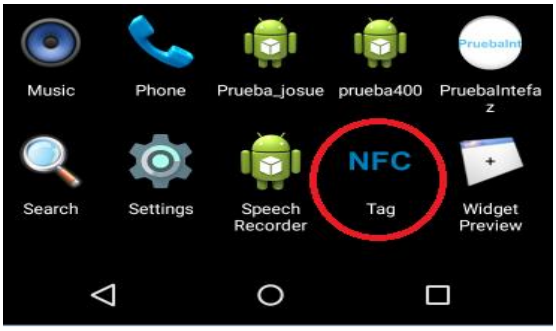


Figura A.1. *Icono de inicio aplicación móvil*

Pulsando sobre el icono anterior, se daría comienzo a la actividad principal o main activity de la aplicación. Es la siguiente:

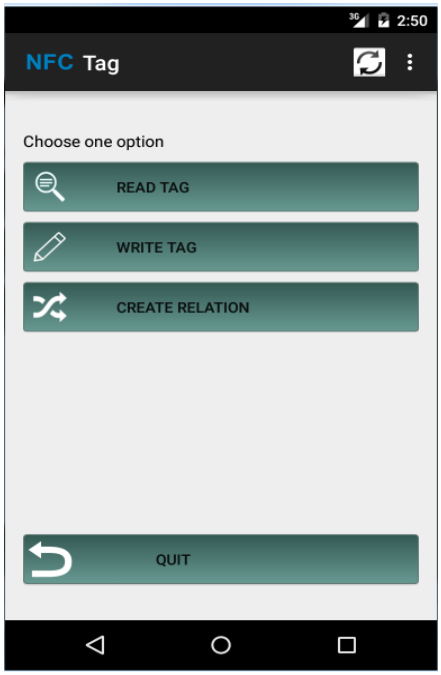


Figura A.2. *Actividad principal aplicación móvil*

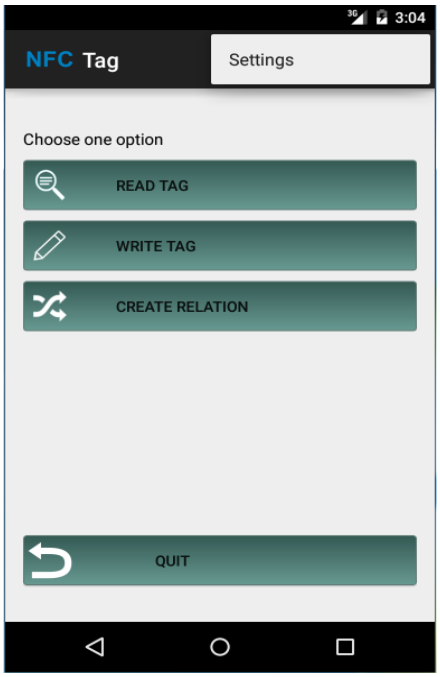


Figura A.3. *Actividad principal aplicación móvil*

La pantalla principal tiene una serie de botones que se describirán a continuación:

- Botón Read Tag. Este botón selecciona el modo lectura de la aplicación. Inicia una segunda actividad donde ya se puede comenzar a leer etiquetas NFC.

- Botón Write Tag. Este botón selecciona el modo escritura de la aplicación. Inicia una segunda actividad donde elegir el tipo de escritura a realizar en la etiqueta NFC.

- Botón Create Relation. Este botón selecciona el modo crear, eliminar o seleccionar relación. Inicia una segunda actividad donde crear, eliminar o seleccionar el tipo de relación actualmente usada en la aplicación.

- Botón Quit. Este botón cierra y finaliza la aplicación.

En la barra superior de la aplicación existen dos botones o iconos más. El de sincronización y el de settings. Su función es la siguiente:

- Botón sincronización. Este botón inicia la sincronización del Smartphone con el servidor.
- Botón settings. Muestra algunas características de la aplicación.

ACTIVITY LECTURA DE ETIQUETAS

Esta actividad es iniciada al pulsar el botón *"Read Tag"* de la actividad principal. La lectura de la etiqueta NFC comenzaría nada más acercar el teléfono a una etiqueta NFC, estando activado el NFC del propio Smartphone. Dependiendo de cómo se lea dicha etiqueta, el teléfono móvil en posición **landscape** o **portrait**, se realizará una lectura con el objetivo de realizar una búsqueda en la base de datos, o por el contrario, se realizará una inserción en la base de datos. El objetivo de elegir un tipo de acción u otra en función de la posición de la pantalla, es de dotar de simplicidad y facilidad de uso a la aplicación. El simple hecho de poder marca el tipo de acción a realizar con un simple movimiento físico, otorga un grado de usabilidad de carácter directo para el usuario, ayudando de esta forma a la simplicidad de uso.

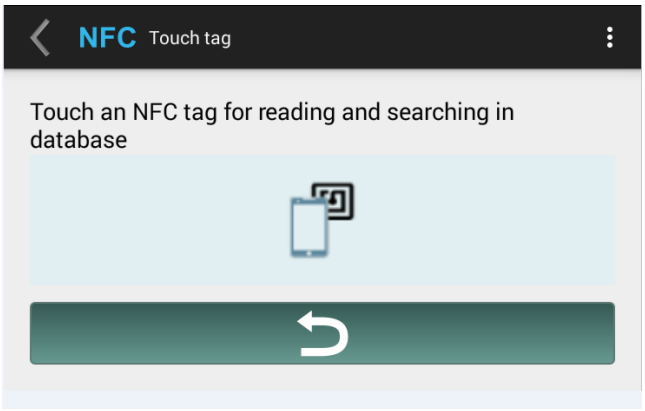


Figura A.4. *Lectura en posición Landscape.*

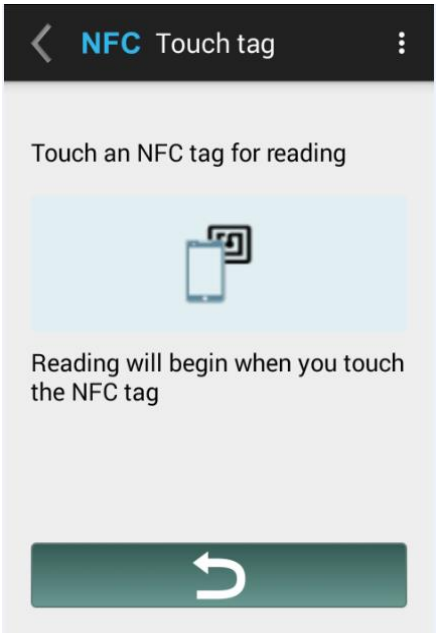


Figura A.5. *Lectura en posición Portrait.*

Esta actividad está compuesta simplemente por un botón que volvería a mostrar la actividad principal, además texto e imágenes explicativas sobre los pasos a seguir para leer una etiqueta NFC.

Al acercar el teléfono móvil a la etiqueta NFC se dispararía un evento que provocaría el inicio de una nueva actividad. Esta actividad está asociada a la actividad de lectura de etiquetas, y dependiendo en qué posición estaba el dispositivo móvil (landscape o portrait) al leer la etiqueta, mostrará unos resultados correspondientes a una búsqueda o por el contrario mostrará el resultado de una inserción.

Lectura en posición landscape

Al acercar el teléfono móvil a la etiqueta NFC en posición landscape, se inicia una búsqueda en la base de datos acerca del objeto leído, mostrando si existe alguna coincidencia en la tabla *"Log"*. Supongamos que en la etiqueta NFC está almacenado el nombre de un objeto, en este caso

"Rey". Al hacer una lectura con el Smartphone en posición landscape se realizará una búsqueda de todas las relaciones con el objeto leído. Un posible resultado de la búsqueda sería:

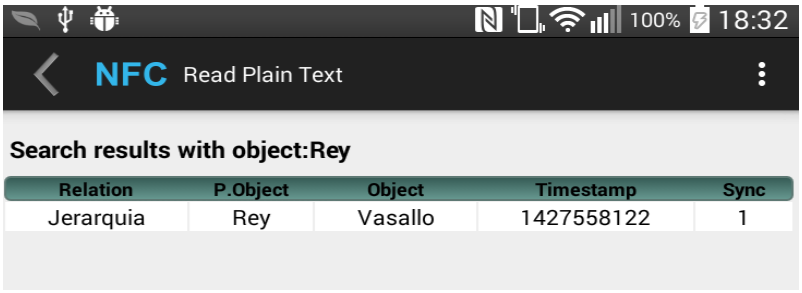


Figura A.6. *Resultado de la búsqueda del objeto leído.*

Como resultado se ha obtenido una única entrada en la tabla *"Log"* de la base de datos acerca del objeto *"Rey"*. La relación es de tipo *"Jerarquía"* donde el objeto padre o de referencia y es *"Rey"*, objeto asociado o hijo es *"Vasallo"*, timestamp de inserción en la base de datos *"1427558122"* y sincronización *"1"*, lo que da a entender que esta relación está sincronizada con el servidor.

Lectura en posición portrait

Al acercar el teléfono móvil a la etiqueta NFC en posición portrait, se iniciará un almacenamiento en la base de datos del objeto leído. Este modo de lectura tiene dos casos diferenciados:

- Activado el modo de lectura agrupado. Este modo se activa o desactiva pulsando la tecla física de bajar volumen del dispositivo.

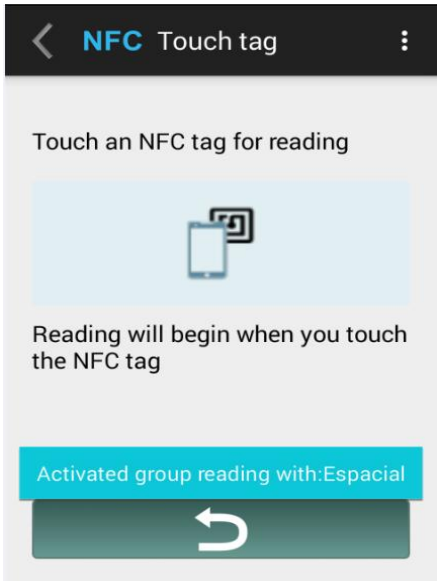


Figura A.7. *Activado el modo lectura agrupado*

Si este modo está activado, las lectura realizadas a continuación de su activación estarán asociadas a un tipo de interacción concreta, y por tanto, a un tipo de relación. Existen dos casos:

- Primera Lectura. Si es la primera lectura, el objeto leído corresponderá al objeto padre o de referencia de la relación marcada por el tipo de interacción activada.

Figura A.8. *Primera lectura agrupada*

Se puede observar que la primera lectura indica que el objeto leído es un objeto padre o de referencia para ese tipo de interacción con su correspondiente relación. En este caso, el objeto padre o de referencia es *"Librería",* la interacción es *"Espacial"* y la relación es *"Esta en"*. La información se almacenará de manera adecuada en las tablas que correspondan en la base de datos.

- Lecturas siguientes. Para las siguientes lecturas, ya se tiene el tipo de relación definida con su objeto padre o de referencia, por lo cual pasarán a ser objetos hijos pertenecientes a la relación.

Figura A.9. *Lectura agrupada sucesiva*

En este caso, a la relación *"Esta en",* con el objeto padre o de referencia *"librería"*, se le ha añadido el objeto hijo *"libro1"* con timestamp *"1430157517"* y sincronización *"0"*.

- Desactivado el modo de lectura agrupado. En este caso se pulsaría el botón físico de bajar el volumen de forma que se desactivara el modo de lectura agrupado.

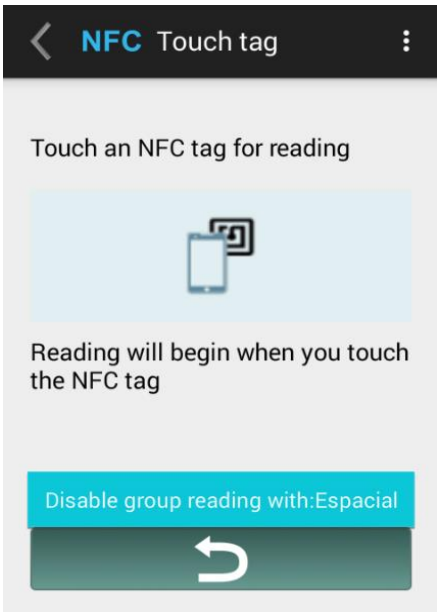


Figura A.10. *Desactivado el modo lectura agrupado.*

De esta forma, al leer etiquetas NFC, los objetos leídos no tendrán ningún tipo de relación con otros, almacenándose directamente en la base de datos.

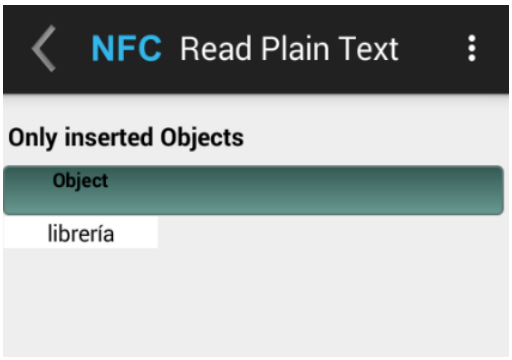


Figura A.11. *Lectura de objeto*

En este caso se ha añadido a la base de datos el objeto *"Rey"*

ACTIVITY MENÚ ESCRITURA DE ETIQUETAS

Esta actividad es iniciada al pulsar el botón *"Write Tag"* de la actividad principal. Esta actividad tiene la función de seleccionar el tipo de escritura de datos a realizar.

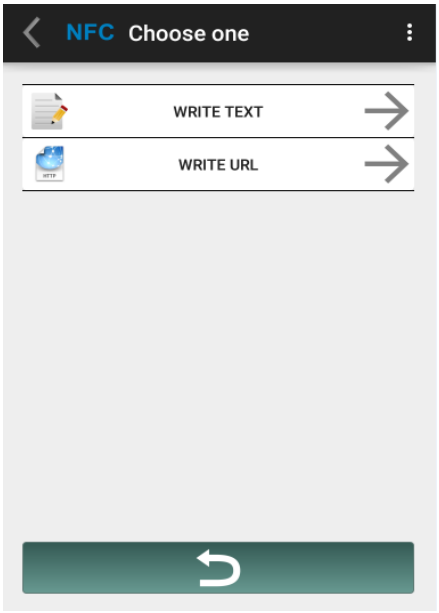


Figura A.12. *Menú tipo de escritura*

La pantalla tiene una serie de botos mediante los cuales se puede seleccionar los tipos de escritura de datos posible.

- Botón Write Text. Este botón inicia la actividad de escritura de texto plano en la etiqueta NFC.
- Botón Write URL. Este botón inicia la actividad de escritura de URLs en la etiqueta NFC.
- Botón Back. Este botón cierra la actividad actual, devolviendo el control a la actividad principal.

ACTIVITY ESCRITURA TEXTO PLANO/URL

Esta actividad se inicia al seleccionar algunas de las opciones de escritura disponibles en la actividad menú escritura. En esta actividad tiene la funcionalidad de escribir texto plano o URL, en función de lo elegido, en las etiquetas NFC.

Figura A.13. *Escritura de texto plano*

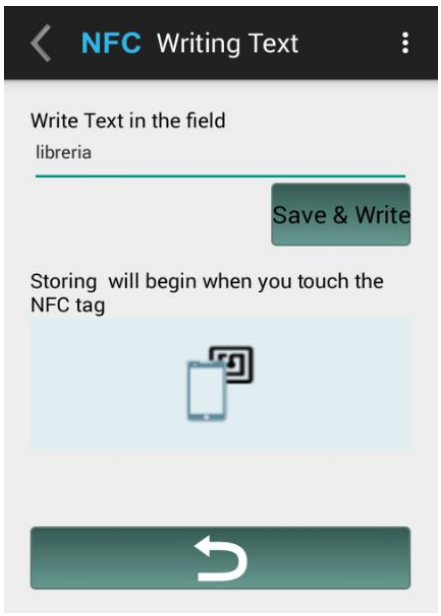


Figura A.14. *Escritura de texto plano 2*

En las figuras anteriores, se muestra la apariencia de la actividad de escritura de texto plano. En el campo de texto se escribe lo que se desea almacenar en la etiqueta NFC. Posteriormente se pulsa sobre el botón *"Save & Write"* y se acercaría el dispositivo móvil a la etiqueta NFC. La escritura sería inmediata en la etiqueta NFC. En el caso de haber seleccionado la escritura de URL el proceso sería exactamente igual al descrito.

ACTIVITY RELACIONES

Esta actividad es iniciada al pulsar sobre el botón *"Create Relation"* de la actividad principal. Su funcionalidad consiste en crear o eliminar las diferentes interacciones-relaciones utilizadas para los modos de lectura agrupados.

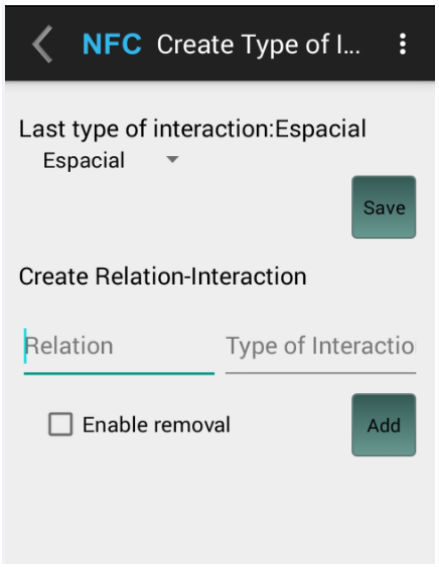


Figura A.15. *Crear interacción-relación*

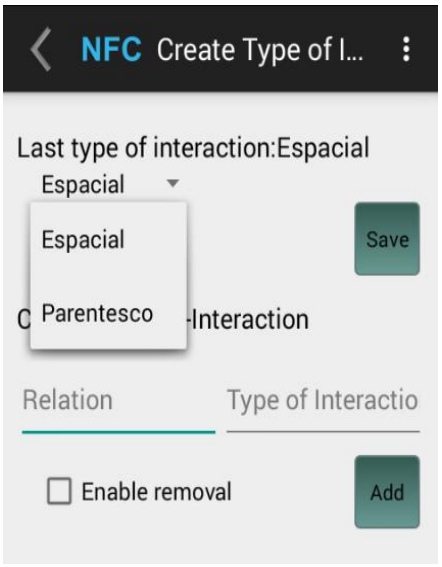


Figura A.16. *Seleccionar interacción*

En las figuras anteriores, se observan los casos de crear nuevas relaciones-interacciones y seleccionar el tipo de interacción deseado.

En el caso de la creación de nuevas relaciones-interacciones, estas estarán almacenadas en una base de datos local a la aplicación. La inserción se realizaría al añadir la relación junto a su interacción en los campos de texto. Una vez introducida está información, al pulsar sobre el botón *"Add",* se añadiría a la tabla local la nueva información.

En el segundo caso, correspondiente a la selección del tipo de interacción a utilizar, se elegiría la interacción buscada seleccionando de las disponibles en la lista desplegable. Al pulsar sobre el botón *"Save"*, se iniciaría la actividad de lectura de etiquetas NFC con el modo agrupado activado por defecto.

Existe un tercer caso correspondiente a la eliminación de las relaciones-interacciones de la tabla local. Esta operación se daría tras la selección del check box *"Enable removal"* como se muestra en la figura a continuación.

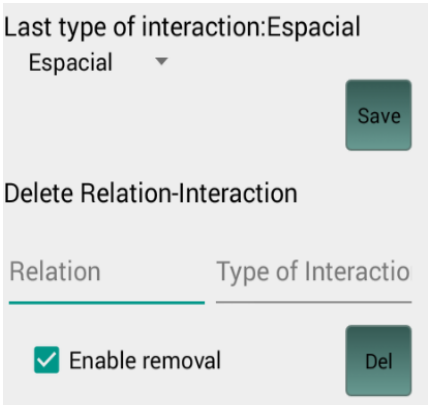


Figura A.17. *Eliminar relación-interacción*

Si el check box está activado, solo se tendría que escribir la relación-interacción a eliminar en la tabla local y esta dejaría de aparecer en la lista desplegable. Este proceso se llevaría a cabo tras pulsar el botón *"Del"* anteriormente *"Add"* de la actividad.

ACTIVITY SINCRONIZACIÓN

Esta actividad se inicia al pulsar sobre el botón de sincronización de la actividad principal. Al pulsar sobre este botón se inicia el proceso de sincronización bidireccional entre la base de datos del dispositivo móvil y la base de datos del servidor. Se puede observar a continuación el resultado de pulsar dicho botón.

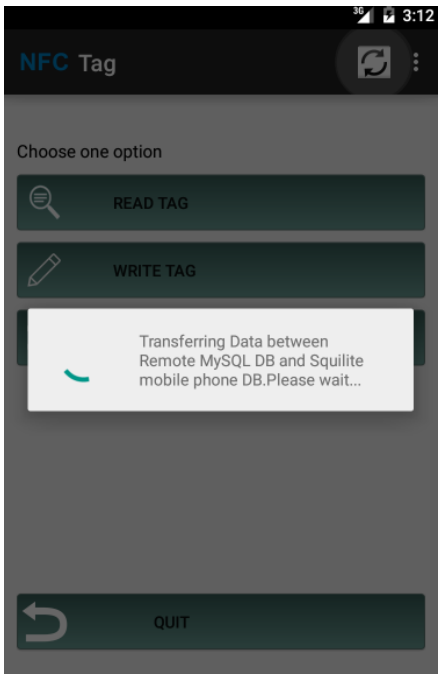


Figura A.18. *Transferencia de datos en la sincronización*

Relation	P.Object	Object	Timestamp	Sync
Esta en	libreria	libro	1427558024	1
Esta en	libreria	libro1	1427558026	1
Familiar	libreria	libro	1427558043	1
Familiar	libreria	libro1	1427558046	1
Jerarquia	Rey	Vasallo	1427558122	1
Esta en	libreria	libreria	1427580742	1
Esta en	libreria	libreria	1427746543	1
Familiar	libro1	libro	1427991136	1
Esta en	libreria	libreria	1428756867	1
Esta en	libreria	libro1	1428756927	1
Esta en	libreria	libreria	1428756950	1
Esta en	libreria	pteee	1428756960	1
Esta en	libreria	pteee	1428757033	1
Esta en	libreria	Rey	1428757057	1
Esta en	libreria	Rey	1428757064	1
Esta en	libreria	Rey	1428757072	1

Figura A.19. *Datos sincronizados*

Este proceso tarda un tiempo en completarse, por lo que se ha provisto de un dialogo emergente, *Figura A.18.*, mientras se realiza dicha operación. Una vez completada esta operación, se mostrará la tabla Log con los datos que contiene. Si la sincronización se ha realizado correctamente, todos los datos deberían tener un 1 en el campo *"Sync"*. En Caso contrario, el proceso de sincronización ha fallado.

INTERFAZ DEL SERVIDOR

El servidor tendrá su propia interfaz con el objetivo de poder manipular y visualizar los datos de forma sencilla para el usuario. La interacción con dicha interfaz se realizará mediante el uso del navegador, teniendo a disposición del usuario tres ventanas diferenciadas: **La ventana de inserción**, **la ventana de eliminación** y **la ventana de mostrar datos**.

VENTANA INSERCIÓN

En esta ventana se insertan datos a la base de datos del servidor. Está compuesta por un formulario con cuatro campos a rellenar por el usuario. En la *Figura 5.21* que se encuentra a continuación se puede observar dicho formulario.

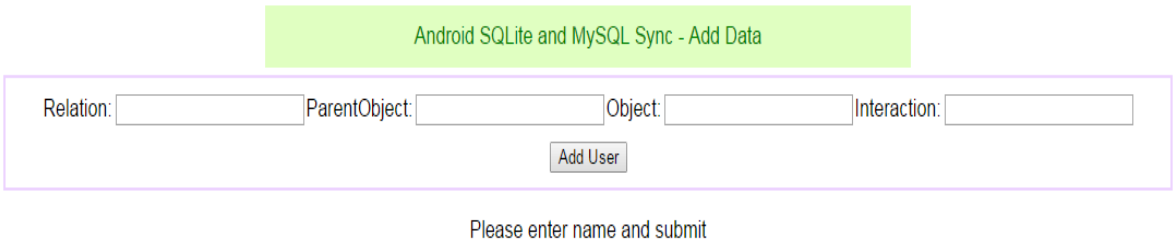


Figura A.20. *Formulario inserción datos B.D. del servidor*

Los campos a rellenar por el usuario son:

- Relation. Este campo se debe rellenar con el tipo de relación a insertar.
- Parent Objetct. Este campo se debe rellenar con el nombre del objeto diferenciador o padre de la relación insertada.
- Object. Este campo se debe rellenar con el nombre del objeto a insertar en la relación.
- Interaction. Este campo se debe rellenar con el tipo de interacción a insertar.

Se puede observar que aún faltan dos campos por dar valor que no se han añadido en el formulario. Estos campos son: El timestamp y la sincronización.

En el caso del timestamp, se añadirá en la inserción automáticamente tomando como valor el instante de tiempo en el que se realizó la inserción en la base de datos.

En el caso de la sincronización, automáticamente tomará el valor 0 (no sincronizado). De esta forma nos aseguramos que se tomen estos valores en el proceso de sincronización.

VENTANA ELIMINACIÓN

En esta ventana, se eliminan datos de la base de datos del servidor. Esta ventana está compuesta por un formulario como en el caso de la inserción, pero esta vez con un único campo a rellenar.

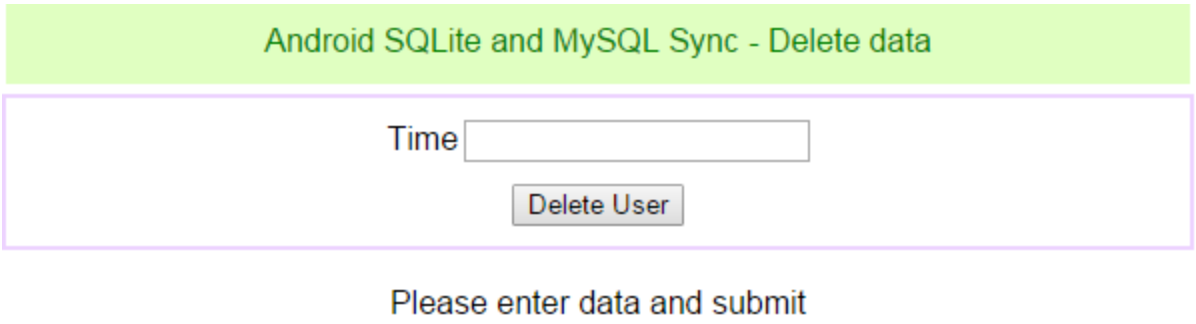


Figura A.21. *Formulario eliminar datos B.D. del servidor*

El campo a rellenar es el del timestamp o time. Como el instante de inserción es único e irrepetible, solo se necesita seleccionar por este campo los datos a eliminar.

VENTANA MOSTRAR DATOS

En esta ventana se muestran los datos contenidos en la base de datos del servidor. Estos datos pueden estar en diferentes estados de sincronización, siendo los estados los siguientes: *"sincronizado"*, *"no sincronizado"* y *"a la espera de eliminar"*. Estos estados son representados en la columna de sincronización con los colores blanco, negro y verde. La interfaz será recargada cada dos segundos para siempre mostrar los datos actuales.

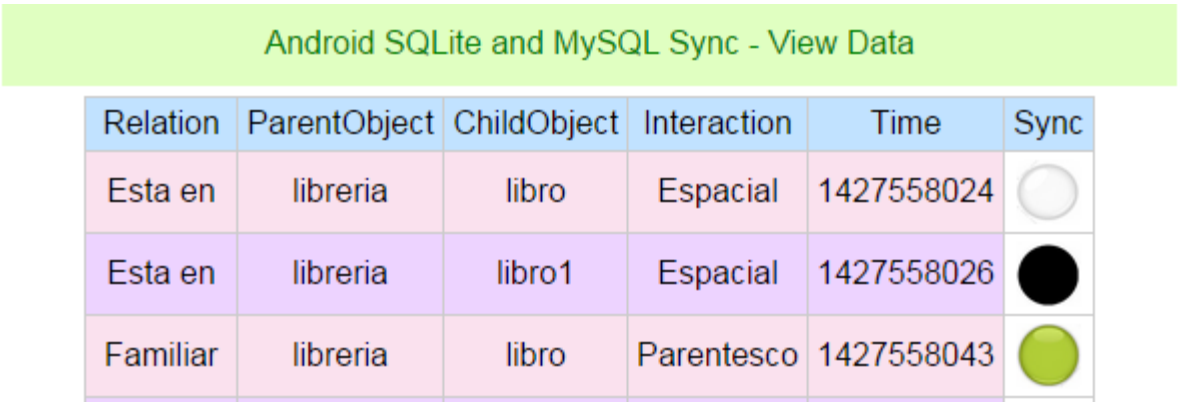


Figura A.22. *Mostrar datos B.D. del servidor*

En la *Figura 5.22.*, se observa una tabla con los diferentes campos y estados de sincronización a mostrar. Los campos son los siguientes:

- Relation. Muestra el tipo de relación.
- Parent Object. Muestra el objeto diferenciador de la relación.
- ChildObject. Muestra los diferentes objetos asociados a la relación.
- Time. Muestra el timestamp o instante de inserción.
- Sync. Muestra el estado de sincronización de la relación. Los tipos de estados son:

    - No sincronizado. Representado por el color blanco. Son los datos que aún no se han sincronizado con el dispositivo móvil.

    - Sincronizado. Representado por el color verde. Son los datos que están sincronizados actualmente entre el dispositivo móvil.

    - A la espera de eliminar. Representa los datos que han sido eliminados en la base de datos y están a la espera de eliminar en el dispositivo móvil. Estos datos estuvieron sincronizados en algún momento, en caso contrario, la eliminación se realizaría directamente sin necesidad de comprobar en el dispositivo móvil su eliminación.

APÉNDICE B: MANUAL DEL PROGRAMADOR

Este apéndice tendrá la función de manual del programador, detallando como instalar la aplicación en el dispositivo móvil y la configuración del servidor. Además, también se detallarán las diferentes partes del código de interés, con el objetivo de facilitar la comprensión del mismo a futuros desarrolladores que trabajen sobre la misma.

En primer lugar, se explicará cómo instalar la aplicación en un Smartphone, mediante el entorno de desarrollo utilizado, y la configuración del servidor y su interfaz web.

En segundo lugar, se detallarán las partes del código de importancia tanto de la aplicación móvil como del entorno web utilizado.

INSTALACIÓN

INSTALACIÓN DE LA APLICACIÓN MÓVIL

Antes de explicar cómo instalar la aplicación en un dispositivo móvil, señalar que es necesario que dicho dispositivo tenga una versión 2.3 o superior del sistema operativo Android, debido al uso de la versión 10 de la API de Android y el servicio de Google, *"Google Cloud Messagin"*. Tanto la API como el servicio están disponibles a partir de dicha versión.

Para instalar la aplicación en el dispositivo móvil, se necesita instalar y configurar previamente el entorno de trabajo utilizando, en este caso Eclipse. Los pasos a seguir son los siguientes:

1. Instalar la última versión del kit de desarrollo de java o JDK.

En primer lugar, habría que descargar e instalar el JDK. La última versión disponible es la 8-u45. Se aceptan las licencias pertinentes y se elige la versión de JDK a instalar en función del sistema operativo a utilizar.

Una vez descargado, comenzar la instalación eligiendo la ruta de instalación que se desee.

Opcional

La documentación oficial de ORACLE recomienda la adición de variables de entorno sobre el JDK al sistema operativo utilizado. Concretamente se añadirán tres variables de entorno: PATH, JAVA_HOME y CLASSPATH. Su utilidad es la siguiente:

- PATH: Con esta variable de entorno se señalaría los ficheros ejecutables del JDK, entre ellos el compilador e intérprete de Java.

- JAVA_HOME: Con esta variable de entorno el sistema operativo conocería la ruta de instalación del JDK.

- CLASSPATH: Esta variable de entorno señala la ruta de los ficheros de clases de Java.

Cada sistema operativo tiene una forma diferente de añadir las variables de entorno. En el caso de Windows se haría lo siguiente:

1) En primer lugar añadiremos la variable de entorno JAVA_HOME. Para ello, clicaremos con el botón derecho del ratón sobre *"EQUIPO"* en el menú de inicio de Windows. A continuación pulsamos en propiedades → Configuración Avanzada → Opciones Avanzadas → Variables de entorno → Editar

   En las siguientes figuras se ilustran los pasos:

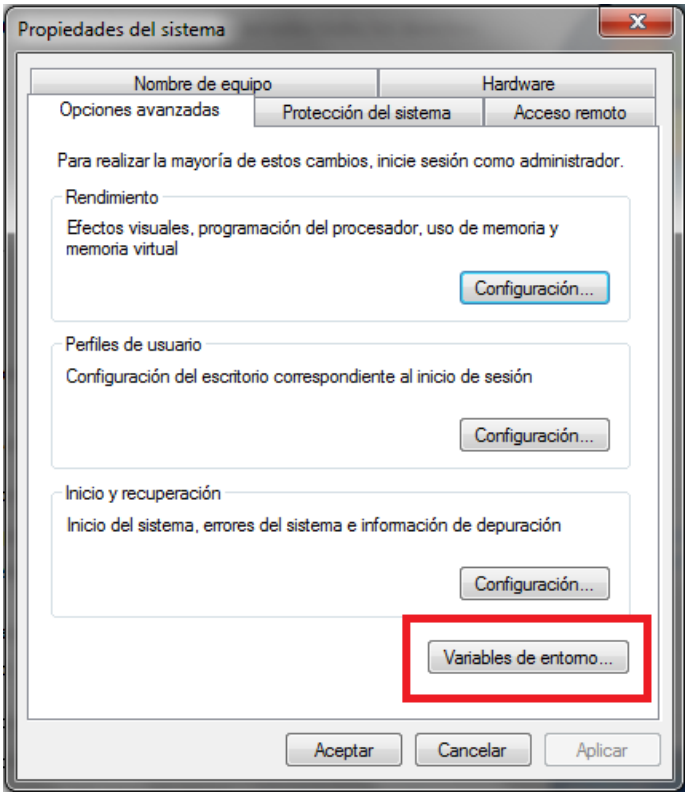


Figura B.1. *Añadir variables de entorno.*

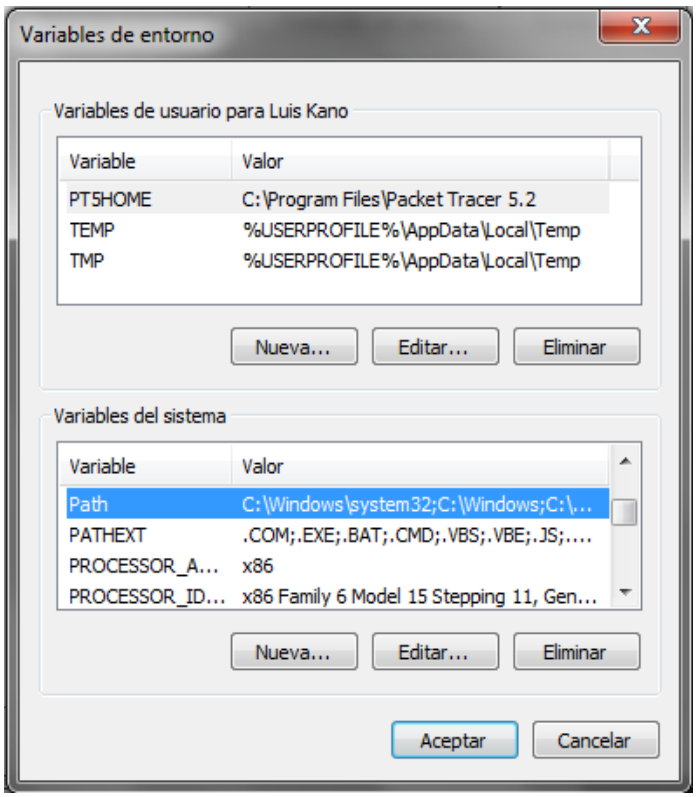


Figura B.2. *Añadir variables de entorno 2.*

2) Añadimos los siguiente valores:

Nombre de la variable: PATH

Valor de la variable: C:\WINDOWS;C:\WINDOWS\system32;%JAVA_HOME%\bin

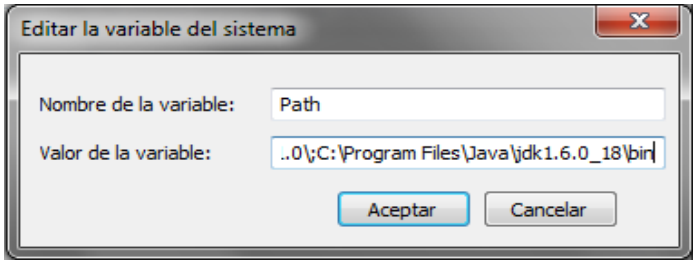


Figura B.3. *Añadir variables de entorno 3.*

3) Añadimos la otras dos variables de entorno de igual manera con los siguiente valores:

Nombre de la variable: JAVA_HOME

Valor de la variable: C:\Program Files\Java\VERSION DE TU JAVA

Nombre de la variable: CLASSPATH

Valor de la variable: C:\Program Files\Java\VERSION DE TU JAVA\src.zip

Una vez realizado, estaría finalizado el proceso de configuración del JDK, y por lo tanto listo para usarse.

2. Instalar Eclipse y el plugin ADT para Android.

En segundo lugar, descargaremos una versión de Eclipse a elección del programador. Se recomienda una versión estándar del mismo, sobre la cual se añadirá el plugin ADT con las funcionalidades necesarias para programar en Android. Una vez descargada la versión de Eclipse deseada, solo es necesario descomprimir el RAR con el IDE donde se desee y ejecutarlo para iniciarlo.

Una vez iniciado, procederemos a la instalación y configuración del plugin. Los pasos son los siguientes:

1) Con el Eclipse abierto, seleccionamos help → Install New Software. Aparecerá una pantalla como la que muestra la siguiente figura:

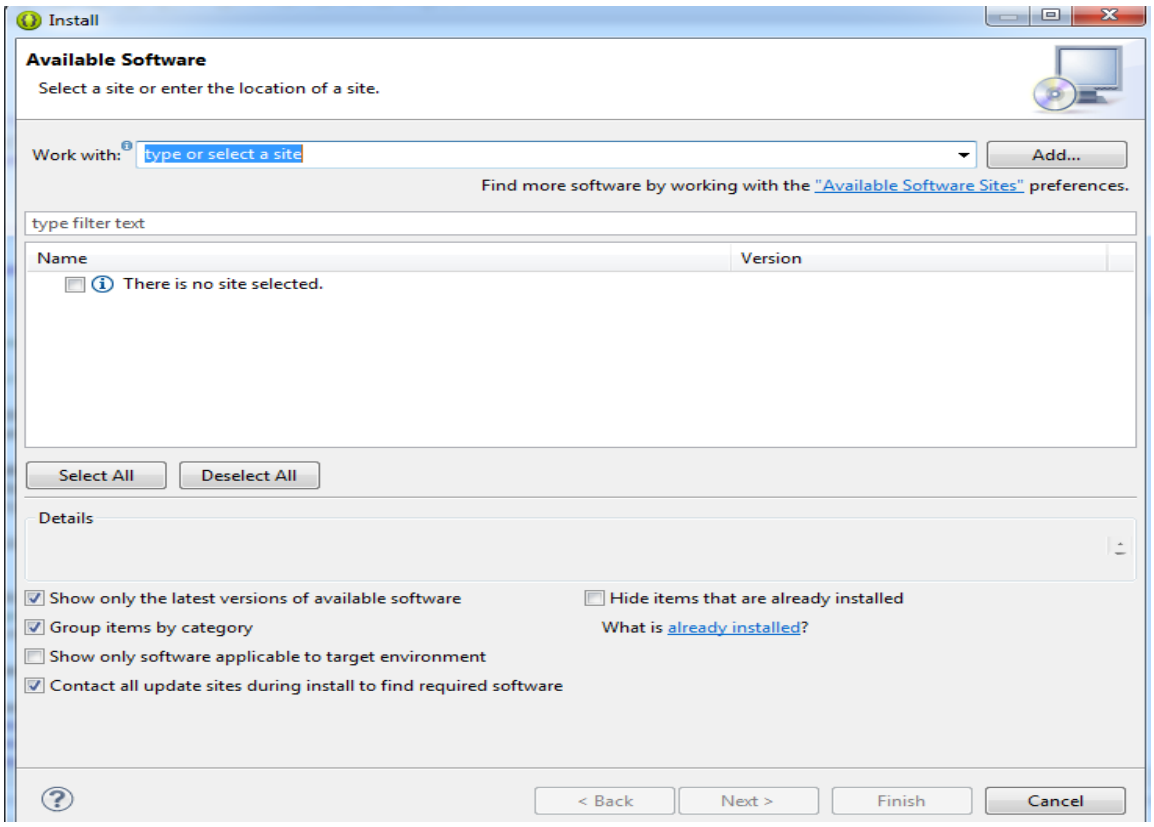


Figura B.4. *Instalación Plugin ADT*

2) Pulsamos el botón Add y se abrirá una nueva ventana. En el apartado Name pondremos un nombre identificativo al plugin a instalar, por ejemplo, ADT- PLUGIN. En el apartado Location escribimos la siguiente URL: http://dl-ssl.google.com/android/eclipse/

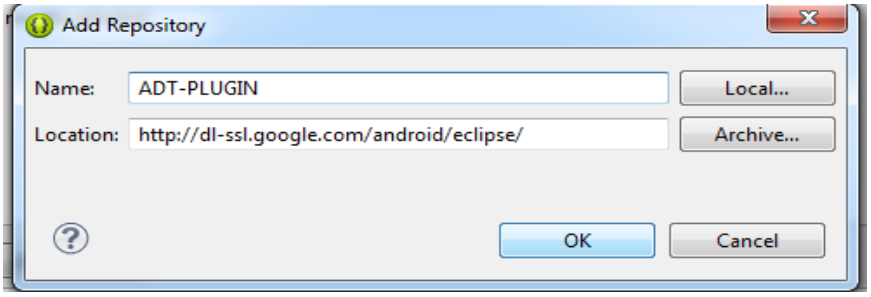


Figura B.5. *Instalación Plugin ADT 2*

Al darle a OK, en la pantalla anterior debería aparecer un conjunto de paquetes a instalar.

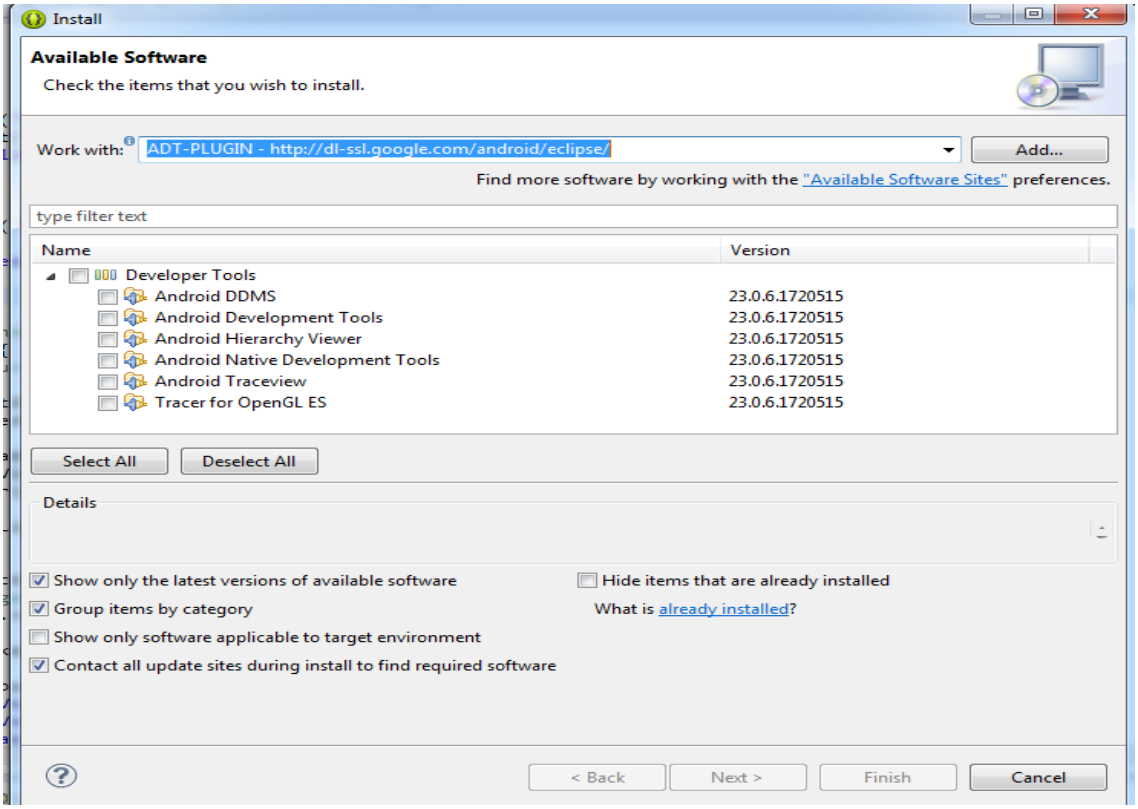


Figura B.6. *Instalación Plugin ADT 3*

3) Seleccionamos los paquetes a instalar y pulsamos sobre NEXT para comenzar la instalación. Una vez finalizada la instalación, reiniciamos el IDE Eclipse.

4) En este paso ya tendríamos instalado el plungin, pero necesitaríamos configurarlo adecuadamente. Seleccionamos Windows → Preferences. Esto abrirá una nueva ventana. Seleccionamos Android, y colocamos en el SDK Location la ruta donde se encuentra el plugin que descargamos en el paso anterior.

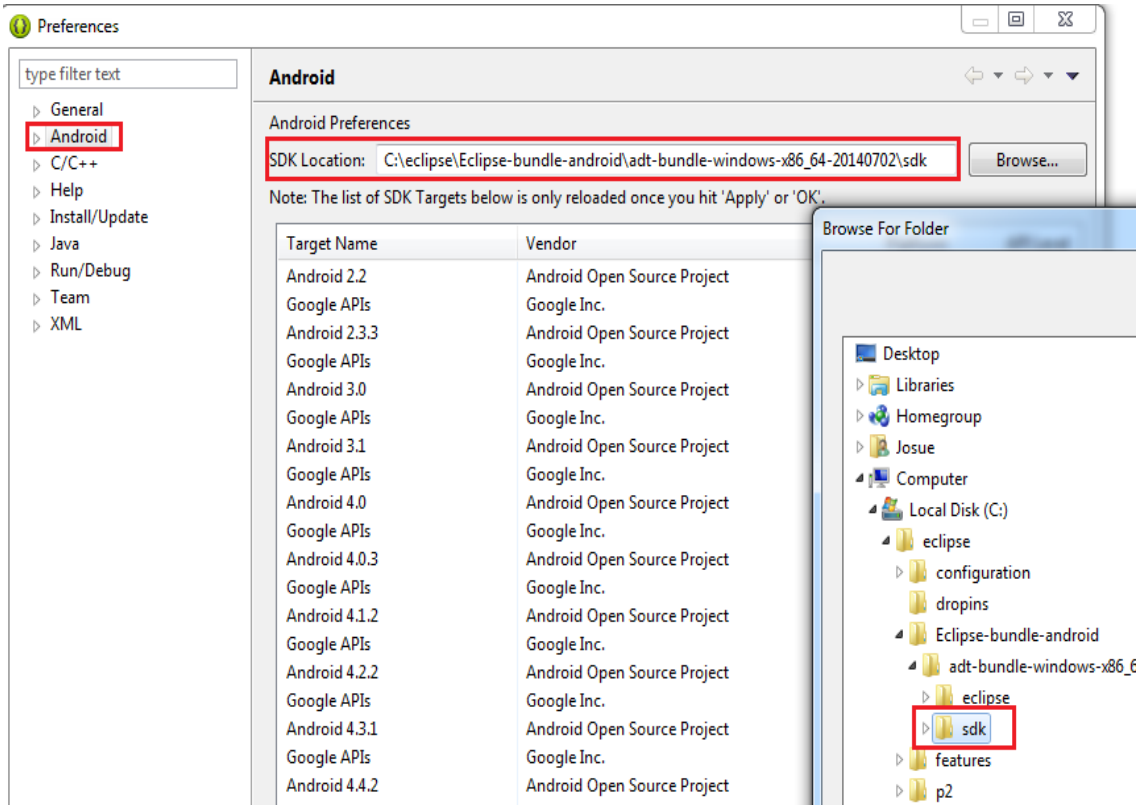


Figura B.7. *Instalación Plugin ADT 4*

5) Con esto el IDE estaría configurado el plugin para funcionar correctamente. Solo faltaría descargar los paquetes asociados al mismo para poder empezar a utilizar el IDE de programación. Para ello pulsamos en el siguiente botón que deberían aparecer en la barra de herramientas:

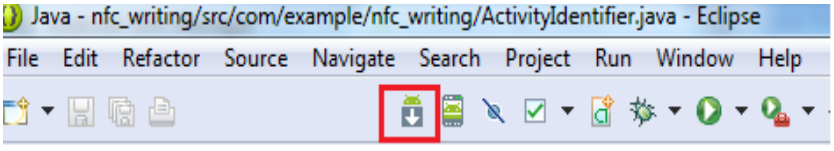


Figura B.8. *Configurando Plugin ADT*

Pulsamos sobre él y esperamos a que se abra la ventana correspondiente. Una vez abierta, observaremos que tenemos a disposición un gestor de paquetes sobre las diferentes APIs, herramientas y demás extras disponibles para programar. Se recomienda como mínimo instalar todos los paquetes correspondientes a herramientas, las APIs como mínimo desde la 4.4 hacia delante y los paquetes de extras. Seleccionamos todo lo anterior e instalamos.

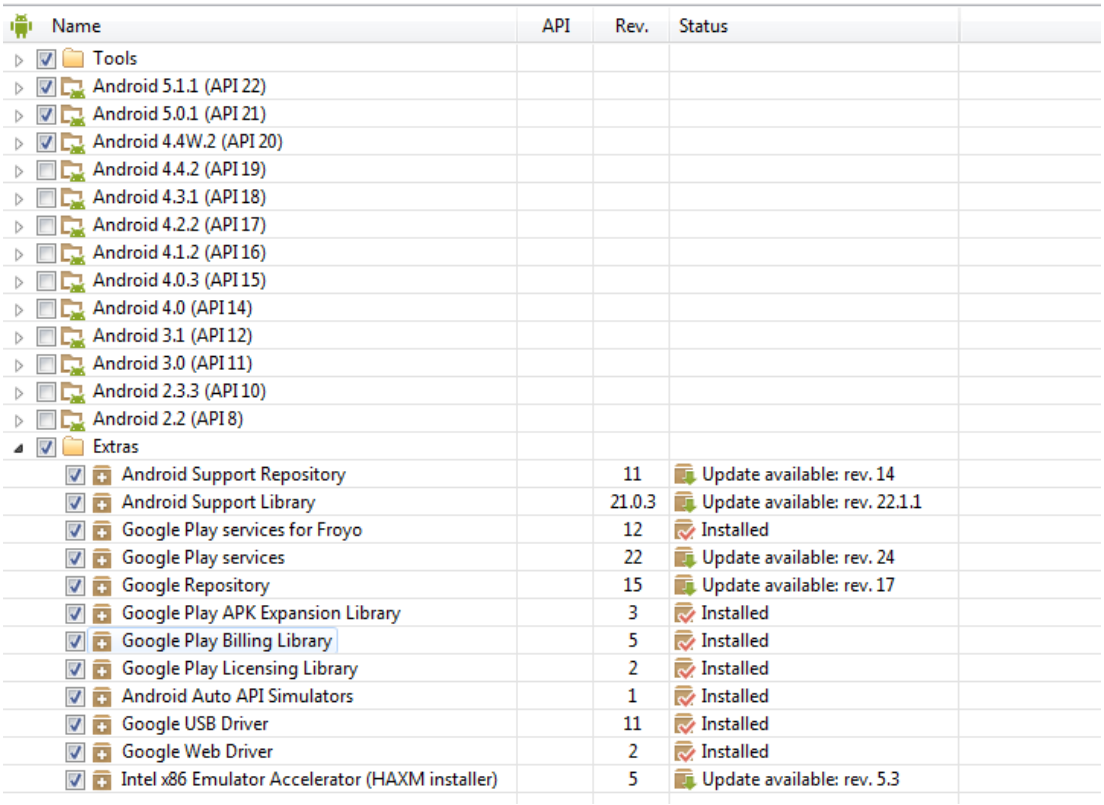


Figura B.9. *Configurando Plugin ADT 2*

Una vez finalizada la instalación, que puede llevar un tiempo medianamente largo, se recomienda reiniciar el IDE. Periódicamente se debería comprobar la existencia de nuevos paquetes a instalar, con el objetivo de mantener siempre las últimas versiones de los mismos.

Una vez hecho lo anterior, el IDE está preparado para ser usado.

3. Importar el proyecto.

Para importar el proyecto simplemente hay que seguir unos sencillos pasos, tras la previa descarga del código de GitHub. Los pasos son los siguientes.

1) Colocamos el código que se ha descargada desde GitHub en un carpeta dentro del Workspace de Eclipse.

2) Seleccionamos File → ImportProyect → Existing Android Code into your space y elegimos la carpeta que contiene el código de la aplicación y seguimos los pasos de importación.

Realizado lo anterior, el proyecto estaría importado correctamente en nuestro IDE. En caso

de contener errores, lo más probable es que se necesite la importación manual de librerías. En el siguiente punto se explica cómo realizar esto. En caso contrario, ir directamente al punto 5 de este apartado del Apéndice

4. Instanciación de librerías usadas en el proyecto.

La aplicación utiliza una serie de librerías externas a las propias APIs de cada versión del sistema operativo Android. Algunas de ellas, son accesibles descargándolas desde el propio gestor de herramientas Android, seleccionándolas en el apartado de extras (en el paso anterior, se recomendó instalar todas las librerías existentes en ese apartado). Otras, puede ser usadas importándolas como librerías externas del proyecto. Ambos casos son usados en el proyecto.

Las librerías descargadas utilizando el gestor de paquetes son:

- Appcompat v-7, descargada desde el paquete Android Support Repository.

  Appcompat v-7, no es más que una librería de compatibilidad para poder usar las últimas funcionalidades del sistema operativo Android en las versiones previas al mismo.

- Google Play Services, descarga desde el paquete Google Play Service.

  Google Play Services es un servicio de Google mediante el cual se pueden recibir mensajes generados por un evento externo en el dispositivo móvil. Con este paquete se tiene todas las librerías necesarias para implementar este servicio en el lado del Smartphone.

  Las librerías añadidas de forma externa son:

1) Gson 2.2.4 pudiéndose descargar desde aquí.

   Gson es una librería para Java para convertir o desconvertir cadenas a o desde JSON.

2) Android-async-http, pudiéndose descargar desde aquí.

   Es una librería mediante la cual la aplicación puede procesar fuera del hilo principal de ejecución peticiones HTTP asíncronas.

Todas las librerías anteriores, son utilizadas de una manera o de otra dentro del proyecto. A continuación describiremos los pasos a seguir para poder tener acceso a ellas.

En el caso de las librerías descargadas desde el gestor de paquetes se haría lo siguiente:

1) Seleccionamos File → ImportProyect → Existing Android Code into your Workspace y

elegimos la carpeta que contiene la librería AppCompat-v7. Aunque la opción de importar presupone que el código está en nuestro workspace, en este caso, no tiene por qué ser así. La AppCompat-v7 estará descargada donde se encuentre el gestor de paquetes Android.

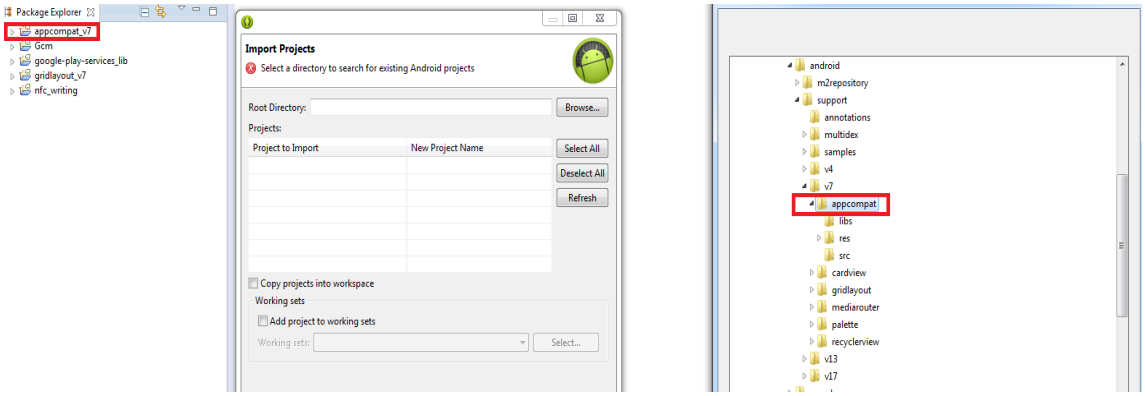


Figura B.10. *Agregando librerías al proyecto.*

2) Referenciar al proyecto la librería descargada. Hacemos click derecho sobre el proyecto → propiedades → Android → Add y seleccionamos a que librería queremos hacer referencia.

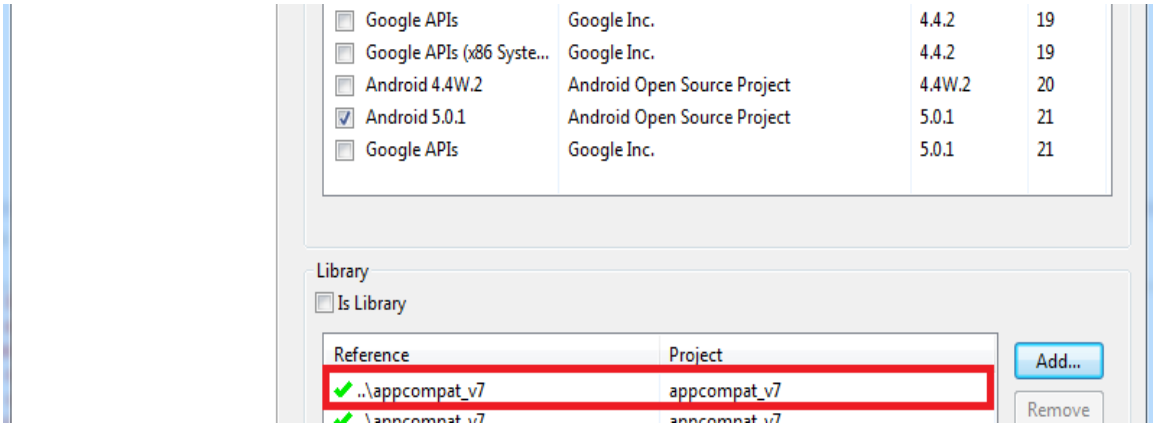


Figura B.11. *Agregando librerías al proyecto 2.*

Este proceso se realizará tanto para la librería AppCompat-v7 como para los servicios de Google Play Services. Una vez realizado, tendríamos acceso a la API de ambas en nuestro proyecto.

En el caso de las librerías añadidas externamente, se realiza lo siguiente:

1) Hacemos click derecho sobre el proyecto → propiedades → Java Build Path → Libraries → Add external JAR. Buscamos las librerías a añadir y pulsamos en OK.

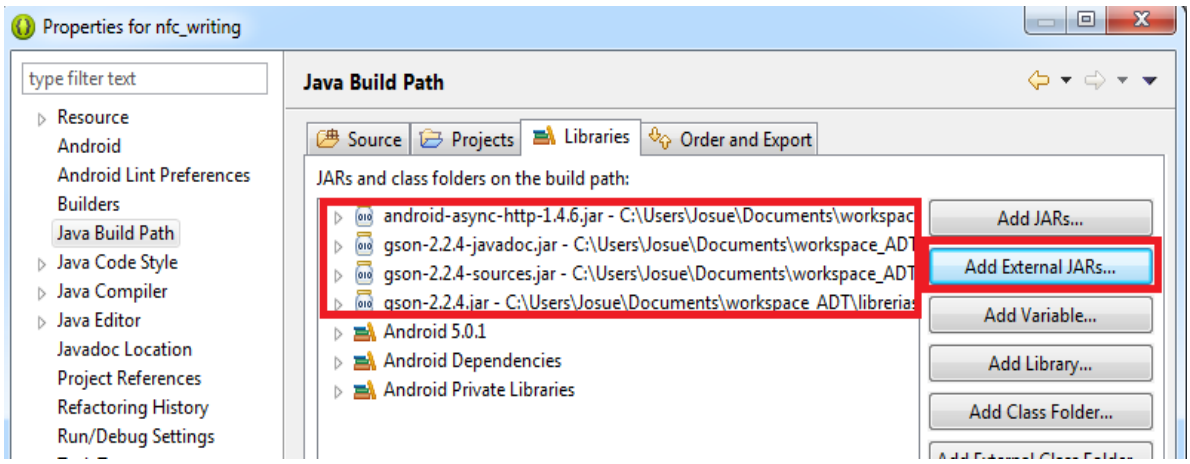


Figura B.12. *Agregando librerías al proyecto 3.*

2) Hacemos referencia a estas librerías para poder tener acceso sobre ellas en el proyecto. Hacemos click derecho sobre el proyecto → propiedades → Java Build Path → Libraries → Order and Export. Seleccionamos todas las librerías y pulsamos OK.

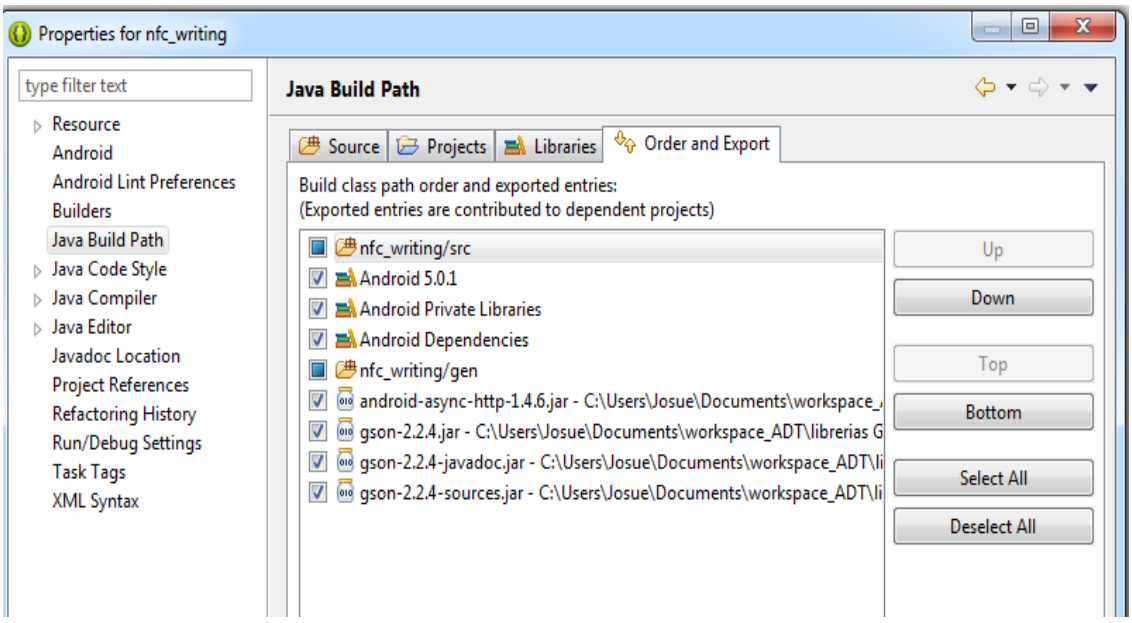


Figura B.13. *Agregando librerías al proyecto 4.*

Una vez realizado esto, tendríamos acceso a la API de estas librerías en nuestro proyecto.

5. Instalación de la aplicación en el dispositivo móvil.

Una vez llegados a este punto, el proyecto está preparado para ser compilado e instalado en el dispositivo móvil. Los pasos son los siguientes:

1) Conectar el dispositivo al PC donde se está el IDE con el código. Instalar los drivers del mismo. Dichos driver se pueden obtener en la página oficial del fabricante del mismo.

2) Clicar en el icono de ejecutar proyecto.

Figura B.14. *Instalar la aplicación*

3) Seleccionar el dispositivo físico o virtual donde instalar la aplicación

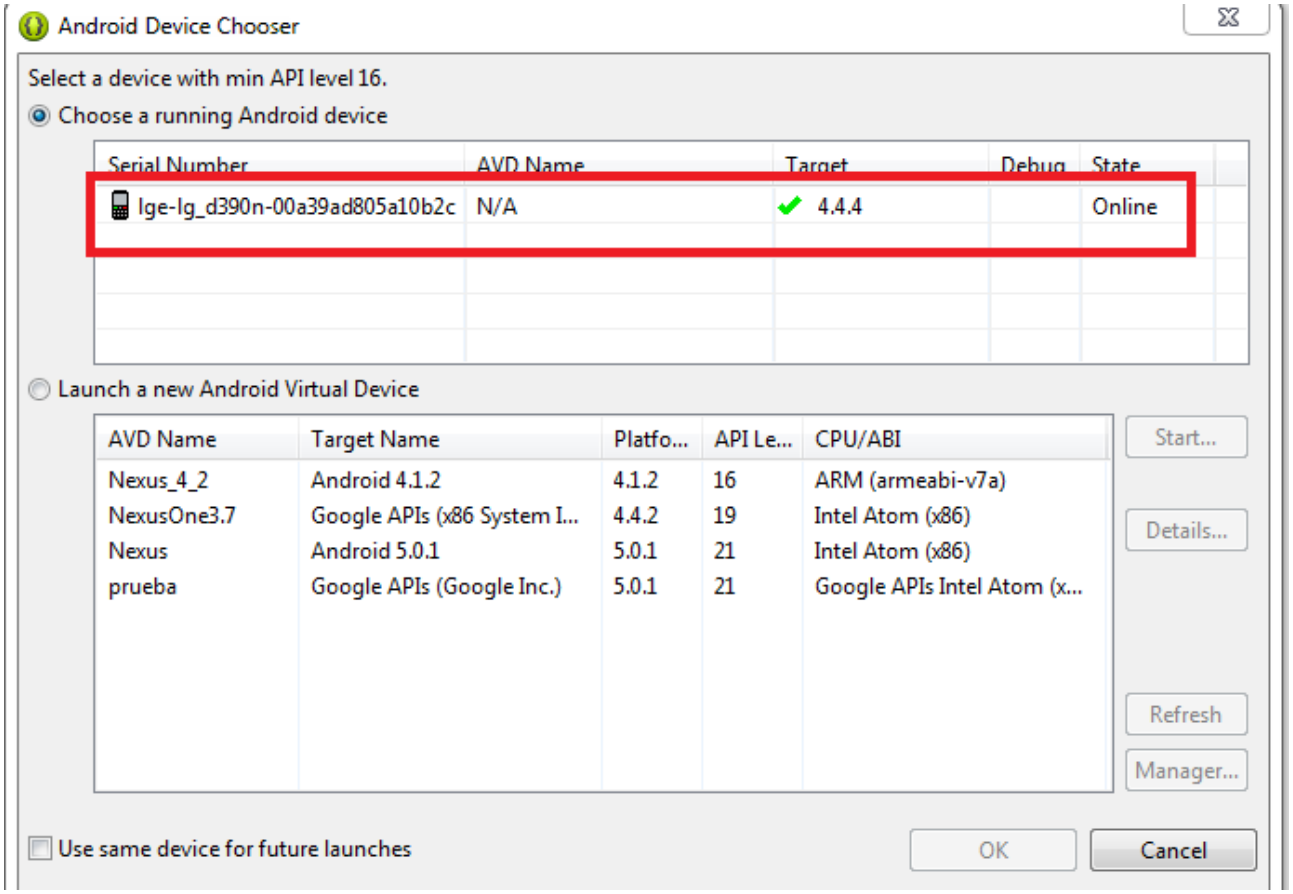


Figura B.15. *Instalar la aplicación 2*

La aplicación se instalaría y se iniciaría automáticamente al finalizar el proceso de instalación en el dispositivo elegido.

INSTALACIÓN DEL ENTORNO WEB

Para instalar el entorno de trabajo en el servidor, se necesitaba una base de datos, además de una pequeña interfaz web para manejar la misma. En el proyecto se ha utilizado XAMPP, pero no es de obligatorio uso, quedando a elección del desarrollador en cada momento.

XAMPP es un servidor independiente de plataforma de software libre compuesto por una base de datos MySQL, el servidor APACHE, y los interpretes de los lenguajes PHP y PERL. Se a elegido esta opción por su fácil configuración e instalación. Simplemente se necesita bajar el ejecutable de instalación e instalarlo y ya se tendría en funcionamiento todo lo necesario para comenzar a realizar las pruebas. XAMP se puede descargar desde aquí.

Una vez instalado, tenemos a nuestra disposición un panel de control donde iniciar y para los diferentes servicios requeridos. Iniciaremos los servicios de la base de datos y del servidor Apache.

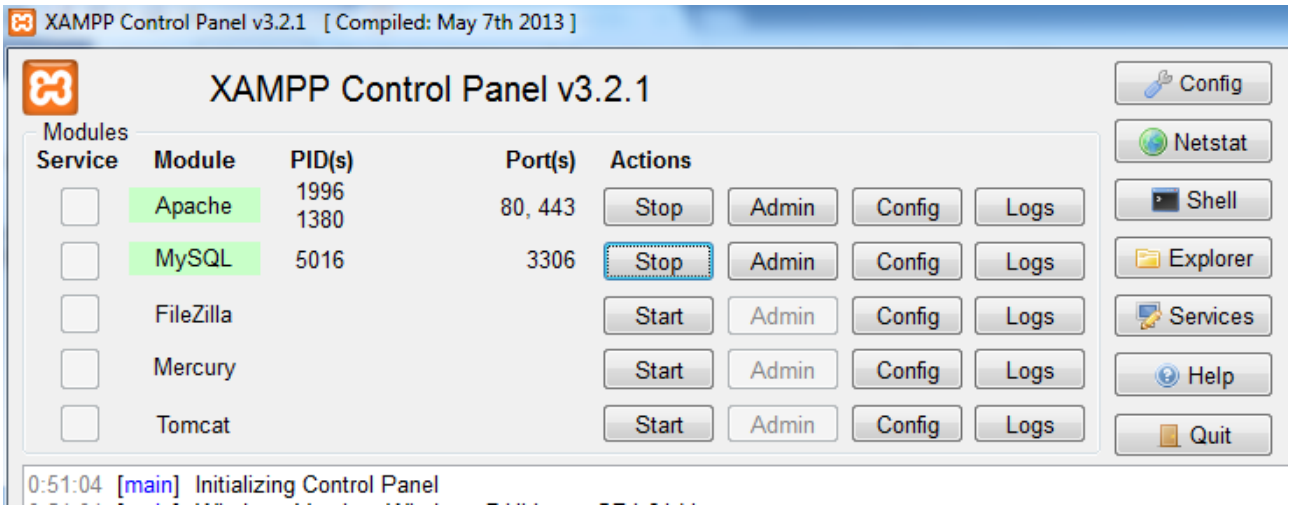


Figura B.16. *Instalación y configuración entorno Web.*

El código PHP utilizado en el proyecto se metería dentro de la carpeta *"htdocs"*. Dentro de la misma estarán accesibles desde http://localhost/mi_carpeta.

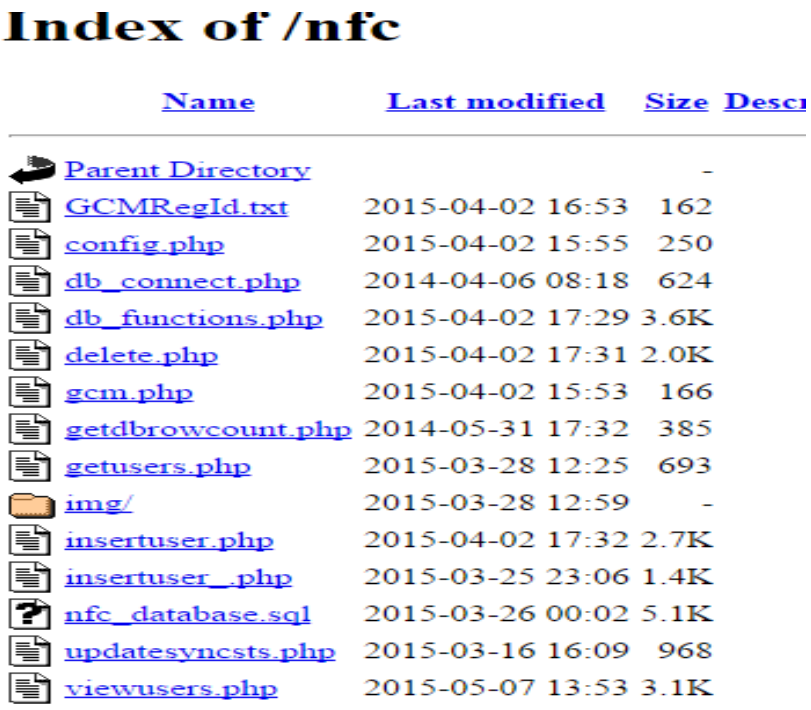


Figura B.17. *Instalación y configuración entorno Web 2.*

En caso de que sea necesario, se podría configurar las diferentes opciones de seguridad y acceso a usuarios de la base de datos desde el propio panel de configuración de XAMPP, pulsando sobre admin en el apartado de MySQL.

Finalmente, tendríamos en funcionamiento el servidor y la base de datos en nuestra red local interna, pudiendo realizar diferentes pruebas con nuestra aplicación móvil.

OBTENCIÓN CREDENCIALES GCM LADO DEL SERVIDOR

En este apartado, se explicará cómo obtener la APIKey y el ProjectID para el servicio de Google *"Google Cloud Messaging"*. Los pasos a seguir son los siguientes:

1. Entrar en la consola de desarrolladores de Google.

   Para acceder a la consola de desarrolladores se puede clicar aquí o en el siguiente link: https://code.google.com/apis/console.

2. Crear un proyecto.

   Si es la primera vez que entramos en la consola, se pedirá crear un nuevo proyecto.

Figura B.18. *Creación de un proyecto consola Google*

3. Activar el servicio de Google Cloud Messaging.

Una vez creado el proyecto, se abrirá una nueva ventana. Clicaremos en Apis y autenticación → APIs → Cloud Messaging for Android.

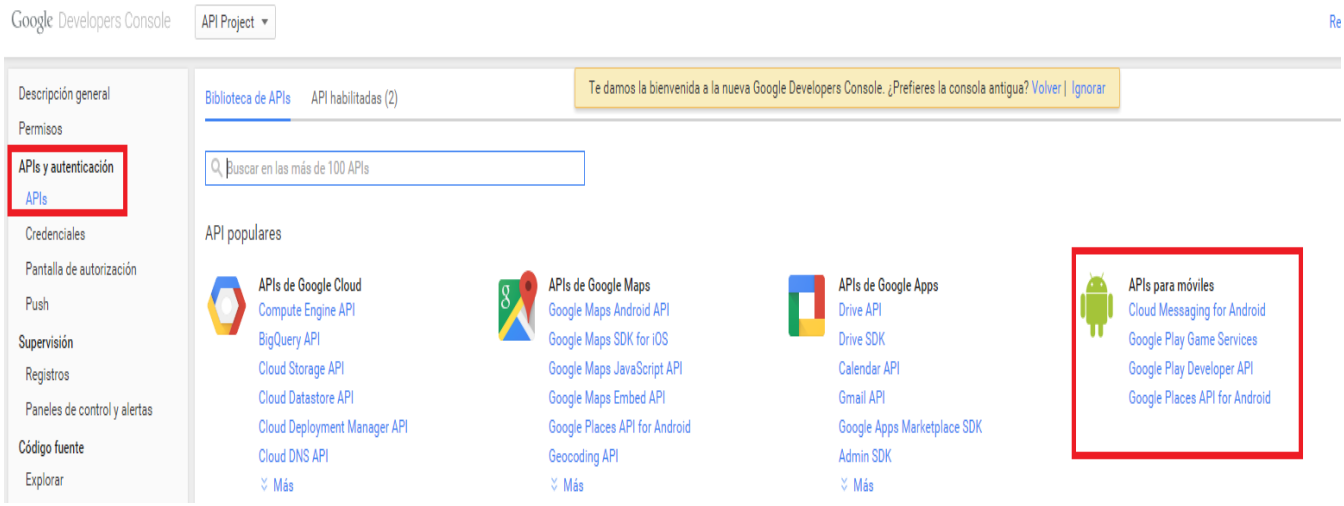


Figura B.19. *Activación de la API*

Aparecerá una nueva ventana donde elegiremos habilitar API.

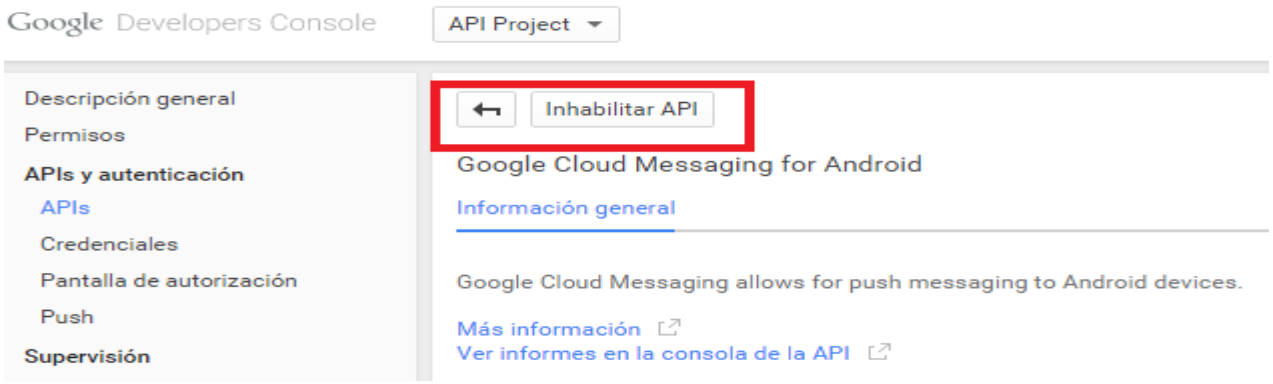


Figura B.20. *Activación de la API 2*

4. Obtener la APIKey y el ProjectID.

En la misma ventana donde nos encontramos, clicamos sobre *"Ver información en la consola de la API"*. Pulsamos en la nueva ventana que aparecerá sobre Api Access y por último sobre new Server Key para generar la APIKey.

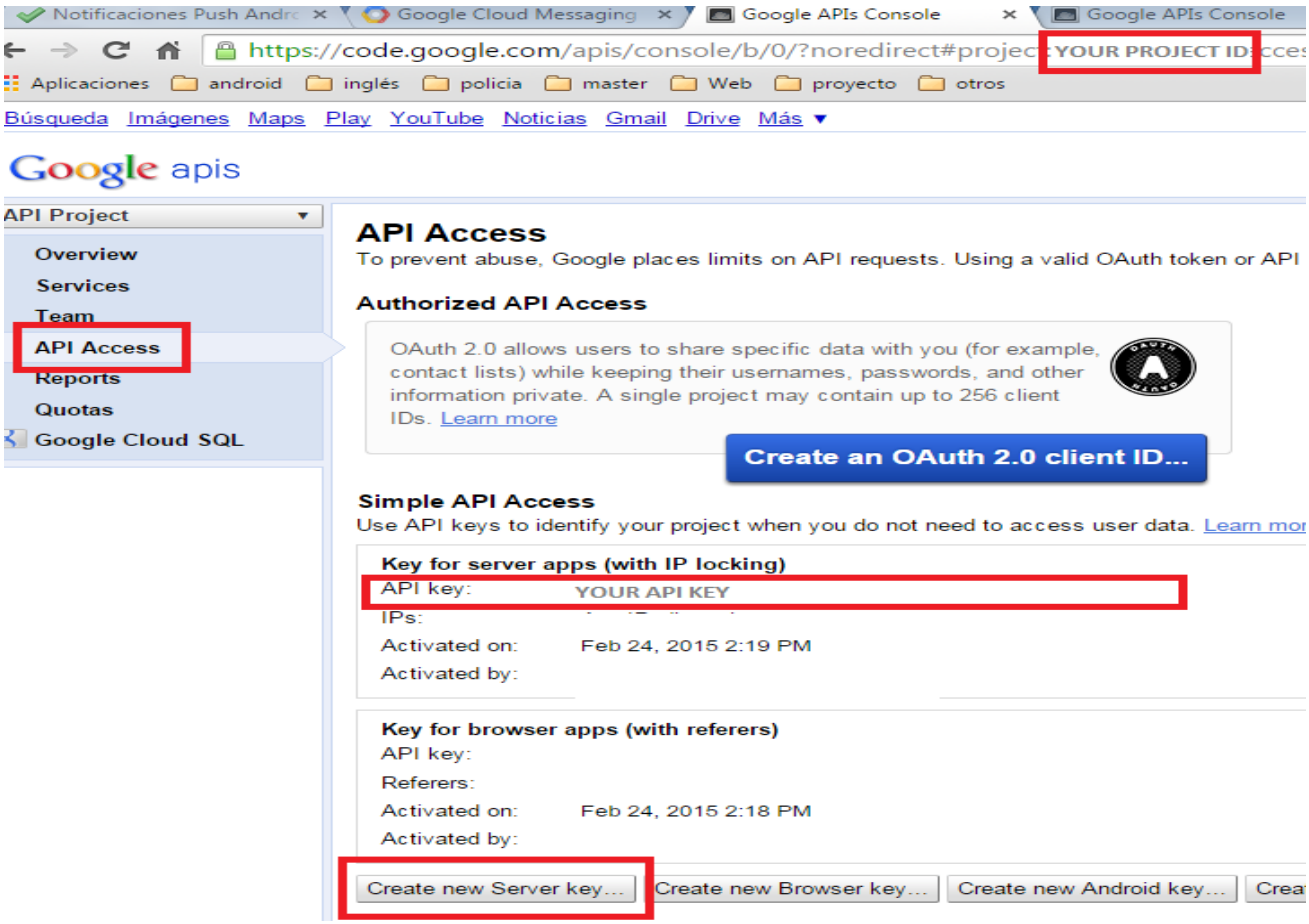


Figura B.21. *Obtención de APIKey y ProjectID*

Finalmente, en la figura anterior se puede observar como en la URL de la página está el número correspondiente al ID del proyecto, es decir, el ProyectID y en la sección de *"Key for server apps"*, se encuentra la APIKey. Por motivos de seguridad se han ocultado dichos valores en la figura de ejemplo anteriormente expuesta.

ESTRUCTURA DEL CÓDIGO

En este apartado, se comentarán las partes del código de interés para el programador, de forma que se le facilite la comprensión y asimilación del código desarrollado.

CÓDIGO APLICACIÓN MÓVIL

Comenzaremos explicando el código de importancia de la aplicación móvil. El código se dividirá por funcionalidades desarrolladas, de forma que el código se explicará en base a ello. Se puede tener acceso al código fuente del mismo en GitHub.

GCM. Lado del aplicativo móvil

A continuación se explicará el código sobre el servicio de Google, *"Google Cloud Messaging"*. Con este servicio lo que se pretende conseguir es avisar al usuario del dispositivo móvil de la existencia de datos no sincronizados, sin la necesidad de que este tenga que estar revisando el estado del servidor. El propio servidor será el encargado de enviar los avisos en caso de que fuera necesario, consiguiendo una mayor eficiencia de recursos en el dispositivo móvil al no tener que realizar este ningún tipo de operación. El código es el siguiente:

Al iniciar la aplicación se comprueba si se tiene almacenado el RegID. En caso de no tenerlo, se inicia el proceso de registro llamando a la función initGCM*()*, comprobando si los servicios de *Google Play Store* están instalados.

```
private void initGCM(){
...
       if (checkPlayServices()) {
           registerInBackground(); //Check if Google Play Store is installed
       }
...
}
```

Si está instalado el servicio, se llama a la función *registerInBackground()*, a través de la cual se obtendrá el RegID.

```
private void registerInBackground() {
...
       @Override
       protected String doInBackground(Void... params) {
              try {
                   if (gcmObj == null) { //Get RegId
                       gcmObj = GoogleCloudMessaging.getInstance(getApplicationContext);
                       regId = gcmObj.register(ApplicationConstants.GOOGLE_PROJ_ID);
```

```
                }
            }catch (IOException ex) {
                    msg = "Error :" + ex.getMessage();
            }
        }
        @Override
        protected void onPostExecute(String msg) {
            if (!TextUtils.isEmpty(regId)) {
                // Store RegId created by GCM Server in SharedPref
                storeRegIdinSharedPref(applicationContext, regId);
            }
        }
...
}
```

A continuación, se almacena el registro en el dispositivo y se llama a la función encargada de enviarlo al servidor.

```
private void storeRegIdinSharedPref(Context context, String regId) {
...
        SharedPreferences.Editor editor = prefs.edit();
        editor.putString(REG_ID, regId);
        editor.commit();
        storeRegIdinServer();//Function call to store on the server
...
}
```

Esta función enviará el RegID utilizando una petición HTTP asíncrona. Además, se encargará de manejar los errores que surjan en el proceso de envío.

```
private void storeRegIdinServer() {
...
        AsyncHttpClient client = new AsyncHttpClient();
        RequestParams params = new RequestParams();
        params.put("regId", regId);
        client.post(ApplicationConstants.APP_SERVER_URL, params,new AsyncHttpResponseHandler() {
        @Override
        public void onSuccess(int arg0, Header[] arg1, byte[] arg2) {
                Toast.makeText(applicationContext,"Reg Id shared successfully with Web App
                ",Toast.LENGTH_LONG).show();
        }
        @Override
        public void onFailure(int statusCode, Header[] arg1, byte[] arg2,Throwable arg3) {

        ... //Get Errors
        }
...
}
```

Con este código, se tendría el proceso de registro en el servidor de mensajería GCM finalizado. Ahora se necesita preparar a la aplicación para que esta pueda recibir y procesar mensajes del servidor de mensajería GCM adecuadamente.

Para recibir los mensajes crearemos un Broadcast Receiver. Es el siguiente:

```
public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                // TODO Auto-generated method stub
                ComponentName comp = new ComponentName(context.getPackageName(),
```

```
                        GCMNotificationIntentService.class.getName());
                startWakefulService(context, (intent.setComponent(comp)));
                setResultCode(Activity.RESULT_OK);
        }


}
```

El Broadcast receiver anterior recibirá los mensajes e iniciará un servicio que los procese. Destacar que se ha utilizado un tipo específico de Broadcast, concretamente el wakefulBroadcastReceiver. Con esto nos aseguramos que el dispositivo esté activo el tiempo necesario para procesar el mensaje en el servicio que lancemos desde este.

Para procesar los mensajes tenemos el siguiente servicio.

```
public class GCMNotificationIntentService extends IntentService {
...

    NotificationCompat.Builder builder;
    public GCMNotificationIntentService() {
        super("GcmIntentService");
    }
    @Override
    protected void onHandleIntent(Intent intent) {
        Bundle extras = intent.getExtras();
        GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);
        String messageType = gcm.getMessageType(intent);
        if (!extras.isEmpty()) {
            if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE
                    .equals(messageType)) {
                sendNotification("Message Received from Google GCM Server:\n\n"
                        + extras.get(ApplicationConstants.MSG_KEY));
            }
        }
        WakefulBroadcastReceiver.completeWakefulIntent(intent);
    }
...
```

Con el código anterior, lo primero que se hace es conocer el tipo de mensaje recibido. Para ellos se crea una nueva instancia del GCM y con *getMessageType* se obtiene el tipo de mensaje. Posteriormente, se obtiene los parámetros del mensaje con *getExtras*. Existen varios tipos de mensajes pero el que interesa para esta aplicación es el MESSAGE_TYPE_MESSAGE, siendo estos el tipo de mensajes genéricos del servicio. Si tenemos el tipo de mensaje que esperábamos, llamamos a la función *sendNotification()* para procesar y mostrar el mensaje. Destacar, que una vez que tengamos finalizado el procesamiento del mensaje, se debe volver a permitir que el dispositivo entre en modo *"sleep"*. Para ellos se llama a la función *WakefulBroadcastReciver.completeWakefulIntent()*. En caso de olvidar llamar a esta función, el servicio nunca se cerraría provocando un rápido consumo de la batería.

Por último se procesa el mensaje:

```
private void sendNotification(String msg) {
            Intent resultIntent = new Intent(this, MainActivity.class);
            resultIntent.putExtra("msg", msg);
            PendingIntent resultPendingIntent = PendingIntent.getActivity(this, 0,
            resultIntent, PendingIntent.FLAG_ONE_SHOT);
            NotificationCompat.Builder mNotifyBuilder;
            NotificationManager mNotificationManager;
            mNotificationManager = (NotificationManager) getSystem
            Service(Context.NOTIFICATION_SERVICE);
```

```
            mNotifyBuilder = new NotificationCompat.Builder(this)
                    .setContentTitle("Alert")
                    .setContentText("You've received new message.")
                    .setSmallIcon(R.drawable.ic_launcher);
            // Set pending intent
            mNotifyBuilder.setContentIntent(resultPendingIntent);
            // Set Vibrate, Sound and Light
            int defaults = 0;
            defaults = defaults | Notification.DEFAULT_LIGHTS;
            defaults = defaults | Notification.DEFAULT_VIBRATE;
            defaults = defaults | Notification.DEFAULT_SOUND;
            mNotifyBuilder.setDefaults(defaults);
            // Set the content for Notification
            mNotifyBuilder.setContentText("New unsync data");
            // Set autocancel
            mNotifyBuilder.setAutoCancel(true);
            // Post a notification
            mNotificationManager.notify(notifyID, mNotifyBuilder.build());
...
    }
```

Con este código, se crea un intent que se abrirá cuando pulsemos sobre la notificación entrante. Además, se ha configurado de qué forma se avisará al usuario de la aparición de un mensaje. En este caso, se ha provisto de todo tipo de aviso siempre que sea posible (sonoro, lumínico y vibración) además de mostrar un mensaje entrante en el dispositivo, que en el caso de pulsarlo, provocará el inicio de la aplicación móvil. Todo esto se ha realizado mediante el NotificationManager, que provee todo lo necesario para configurar y mostrar un mensaje emergente en el dispositivo móvil.

Realizado esto, la aplicación estaría lista para poder recibir y procesar mensajes adecuadamente.

Lectura de etiquetas NFC y almacenamiento-búsqueda en una base de datos

En primer lugar, en el manifest.xml de la aplicación Android, se ha asociado a esta actividad un intent-filter. Este intent-filter proporcionará que la actividad se ejecute o se inicie al tocar con el dispositivo móvil una etiqueta NFC, aunque esta aplicación esté cerrada.

```
<activity

       android:name=".ActivityReadText"
       android:configChanges="orientation|screenSize|keyboardHidden|keyboard"
       android:label="@string/title_activity_ReadText"
       android:parentActivityName="com.example.nfc_writing.ActivityMenuRead" >
       <intent-filter>
          <action android:name="android.nfc.action.NDEF_DISCOVERED" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="text/plain" />

        </intent-filter>

</activity>
```

A continuación, se muestra el código sobre la lectura de datos de una etiqueta NFC.

```
...
if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())){// Read NFC card
       NdefMessage[] message = getNdefMessages(getIntent());
       for ( int i = 0; i< message.length; i++)
             for(int j=0; j<message[0].getRecords().length;j++) {
                 NdefRecord record = message[i].getRecords()[j];
                 statusByte = record.getPayload()[0];
                 int languageCodeLength = statusByte & 0x3F;
                 String languageCode = new String( record.getPayload(), 1, languageCodeLength,
                 Charset.forName("UTF-8"));
                 int isUTF8 = statusByte-languageCodeLength;
                 if(isUTF8 == 0x00) {
                     payload = new String( record.getPayload(), 1+languageCodeLength, rec
                     ord.getPayload().length-1-languageCodeLength, Charset.forName("UTF-8"));
                 }
                 else if (isUTF8==-0x80) {
                      payload = new String( record.getPayload(), 1+languageCodeLength,
                      record.getPayload().length-1-languageCodeLength,Charset.forName("UTF-
                     16"));
                 }
                 myText.append((j+1) + "th. Record Tnf: " + record.getTnf() + "\n");
                 myText.append((j+1) + "th. Record type: "+ new String(record.getType()) + "\n");
                 myText.append((j+1) + "th. Record id: " + new String(record.getId()) + "\n");
                 myText.append((j+1) + "th. Record payload: " + payload + "\n");
              }
       }
}
...
```

En el código anterior se realizan varias cosas. Primero, se detecta si el intent entrante se corresponde con el provocado al acercar el dispositivo móvil a una etiqueta NFC. En caso de ser así, se almacenaría los datos del intent entrante en una estructura de datos NDEF, mediante la función *getNdefMessager()*. Esta función pertenece a una clase padre, con métodos comunes entre diferentes Actividades. Concretamente el código de este método de recuperación de datos de una etiqueta NFC realiza lo siguiente:

```
...
//Read to Tag
protected NdefMessage[] getNdefMessages(Intent intent) {
              NdefMessage[] message = null;
              if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
                 Parcelable[] rawmessage =
                 intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
                 if(rawmessage != null) {
                    message = new NdefMessage[rawmessage.length];
                    for( int i = 0; i < rawmessage.length; i++) {
                         message[i]= (NdefMessage) rawmessage[i];
                    }
                 }
                 else {
                   byte[] empty = new byte [] {};
                   NdefRecord record = new NdefRecord(NdefRecord.TNF_UNKNOWN,empty,empty,empty);
                   NdefMessage msg = new NdefMessage(new NdefRecord[]{record});
                   message = new NdefMessage[]{msg};
                 }
              }
              else {
                     Log.d("","Unknow intent.");
                     finish();
              }
              return message;
       }
...
```

En este código, se recupera el mensaje en su totalidad y se almacena en la variable *rawmessage*. A continuación, comprueba si lo leído pertenecía a una etiqueta vacía, y por lo tanto no se leyó nada o por el contario, se leyó contenido. En el caso de estar en una etiqueta con contenido, se procesa el mensaje almacenando lo leído en un mensaje NDef, compuesto por registros.

Volviendo al primer código, una vez tenida la estructura del mensaje NDef, se recorrerían los diferentes registros que componen el mensaje leyendo la carga útil de los mismos. Dicha carga útil puede estar codificada en UTF-8 o UTF-16.

Finalmente, se realiza la operación de búsqueda o inserción de datos en la base de datos a partir de la carga útil leída con anterioridad. La inserción o búsqueda vendrá marcada por la posición (landscape o portrait) en la que se encuentre el dispositivo móvil en el momento de la lectura de la etiqueta NFC. El código es el siguiente:

```
...
if(orientation == 1) { //Insertion in the database
   queryValues = new HashMap<String, String>();
   if(bool == true) {//If grouping is active
       tipo = prefs.getString("Lmultiple", "vacio");
       relacion = prefs.getString("Relacion","vacio");
       if(prefs.getBoolean("OPactive", false)==true) {//Inserting the parent object
          SharedPreferences.Editor editor = prefs.edit();
          editor.putBoolean("OPactive", false);
          editor.putString("OPadre",payload);
          editor.commit();
          queryValues.put("objeto",payload);
          myDatabase.insert(queryValues, "Objetos");
          queryValues.put("relacion",relacion);
          queryValues.put("interaccion",tipo);
          myDatabase.insert(queryValues,"RCruzadas");
          showtable(0,null, payload, null, null,null);
       }
       else { //Inserting the child object
          Long tsLong = System.currentTimeMillis()/1000;
          String timestamp = tsLong.toString();
          queryValues.put("objeto",payload);
          myDatabase.insert(queryValues,"Objetos");
          queryValues.put("relacion",relacion);
          queryValues.put("objetoPadre",prefs.getString("OPadre","SinPadre"));
          queryValues.put("interaccion",tipo);
          queryValues.put("tiempo",timestamp);
          queryValues.put("sincro","0");
          myDatabase.insert(queryValues,"Log");
          showtable(0, null,prefs.getString("OPadre","SinPadre"),payload,   timestamp,null);
       }
    }
    else { //If grouping is not active
      queryValues.put("objeto",payload);
      myDatabase.insert(queryValues,"Objetos");
      showtable(1, null,null, payload, null,null);
   }
...
```

En el código se diferencian los tipos de inserción en la base de datos en función del modo de lectura agrupado esté activado o no. Además, en cada caso, se llama a la función *showtable()* con el objetivo de mostrar los datos insertados en cada caso. Para activar o desactivar el modo agrupado se pulsa la tecla de bajar volumen en la Actividad correspondiente. El código es el siguiente:

```
//Select grouping mode through volume button
public boolean onKeyDown(int keyCode, KeyEvent event) {
...
       Vibrator v = (Vibrator) getSystemService(VIBRATOR_SERVICE);
       switch(keyCode) {

       case KeyEvent.KEYCODE_VOLUME_DOWN:
            sharedPreferences prefs = getSharedPrefe en
            ces("MyPreferences",Context.MODE_PRIVATE);
           Boolean bool = prefs.getBoolean("LMactive",false);
           String txt = prefs.getString("Lmultiple", "vacio");
           SharedPreferences.Editor editor = prefs.edit();
           if(bool == true) {
                editor.putBoolean("LMactive", false);
                editor.commit();
                Toast.makeText(this,"Disable group reading with:"+txt,
                Toast.LENGTH_SHORT).show();
                v.vibrate(500);
              }
           else {
                editor.putBoolean("LMactive", true);
                editor.commit();
                Toast.makeText(this,"Activated group reading with:"+txt,
                Toast.LENGTH_SHORT).show();
                v.vibrate(500);
              }
       return true;
       }
return super.onKeyDown(keyCode, event);
...
}
```

El efecto de activar o desactivar el modo agrupado se consigue sobrescribiendo la funcionalidad del botón de volumen en una actividad en concreto, de forma que si tocáramos ese botón en dicha actividad, se activaría o desactivaría el modo agrupado.

El siguiente caso corresponde con el la búsqueda en la base de datos.

```
else { //Search the database
       SQLiteDatabase db = myDatabase.getWritableDatabase();
       String[] campos = new String[] {"*"};
       String[] args = new String[] {payload};
       Cursor c = db.query("Log", campos, "objetoPadre=?", args, null, null, null);
       if (c.moveToFirst()) {
           do {
                String relacion_ = c.getString(0);
                String objetoPadre = c.getString(1);
                String objeto = c.getString(2);
                String tiempo = c.getString(4);
                String sincro = c.getString(5);
                showtable(2, relacion_,objetoPadre, objeto, tiempo,sincro);
          } while(c.moveToNext());
}
...
```

Para buscar los datos que coinciden con la carga útil leída de las etiquetas NFC, se crea una instancia de la base de datos. Posteriormente se realiza una consulta sobre la misma, buscando coincidencias y mostrándolas en el caso que las hubiese mediante la función *showtable()*.

Escritura de etiquetas NFC

Tras el proceso de captura de texto o URL a escribir en la tarjeta NFC, es necesario sobrescribir diferentes métodos. Son los siguientes:

```
...
@Override //Set up the NDEF message
public void onNewIntent(Intent intent) {
        Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        Locale locale= new Locale("en","US");
        byte[] langBytes = locale.getLanguage().getBytes(Charset.forName("US-ASCII"));
        boolean encodeInUtf8=false;
        Charset utfEncoding = encodeInUtf8 ? Charset.forName("UTF-8") : Charset.forName("UTF-16");
        int utfBit = encodeInUtf8 ? 0 : (1 << 7);
        char status = (char) (utfBit + langBytes.length);
        byte[] textBytes = data.getBytes(utfEncoding);
        byte[] data = new byte[1 + langBytes.length + textBytes.length]; data[0] = (byte) status;
        System.arraycopy(langBytes, 0, data, 1, langBytes.length);
        System.arraycopy(textBytes, 0, data, 1 + langBytes.length, textBytes.length);
        NdefRecord textRecord = new NdefRecord(NdefRecord.TNF_WELL_KNOWN, NdefRecord.RTD_TEXT, new
        byte[0], data);
        NdefMessage newMessage= new NdefMessage(new NdefRecord[] { textRecord });
        writeNdefMessageToTag(newMessage, tag);//Write in NFC tag
}
...
```

Sobrescribiendo el método *onNewIntent()* se consigue el efecto de captar el descubrimiento de una etiqueta NFC cercana al dispositivo móvil. Este descubrimiento generará un intent que se aprovechará para enviar los datos a escribir. Como se puede observar, en el código se forma la estructura de datos a escribir y se encapsula en un registro NDEF, que a su vez este está encapsulado en un mensaje NDEF. Una vez formado el mensaje, se llama a la función *writeNdefMessageToTag()* la cual iniciará la escritura en la etiqueta NFC. El código de este método es el siguiente:

```
...
//Write to tag
protected boolean writeNdefMessageToTag(NdefMessage message, Tag detectedTag) {
        int size = message.toByteArray().length;
        try {
                Ndef ndef = Ndef.get(detectedTag);
                if(ndef != null) {
                   ndef.connect();
                   if(!ndef.isWritable()) {
                       Toast.makeText(this,"Tag Read Only",Toast.LENGTH_SHORT).show();
                       return false;
                       }
                   if(ndef.getMaxSize()< size) {
                      Toast.makeText(this,"Data cannot writtent to tag. Tag capacity is " +
                      ndef.getMaxSize(), Toast.LENGTH_SHORT).show();
                      return false;
                   }
                   ndef.writeNdefMessage(message);
                   ndef.close();
                   Toast.makeText(this,"Message is written tag", Toast.LENGTH_SHORT).show();
                   return true;
                }
                else {
                   NdefFormatable ndefFormat = NdefFormatable.get(detectedTag);
                   if(ndefFormat != null) {
                      try {
                        ndefFormat.connect();
                        ndefFormat.format(message);
                        ndefFormat.close();
```

```
                        Toast.makeText(this,"The data is written to tag",
                        Toast.LENGTH_SHORT).show();
                        return true;
                        }catch (IOException e) {
                           Toast.makeText(this,"Fail to form tag", Toast.LENGTH_SHORT).show();
                           return false;
                       }
                     }
                   else {
                     Toast.makeText(this,"NDEF is not supported", Toast.LENGTH_SHORT).show();
                     return false;
                   }
                 }

        }
        catch (Exception e) {
          Toast.makeText(this,"Write operation is failed", Toast.LENGTH_SHORT).show();
          return false;
        }

}

...
```

Este método, recibe el mensaje NDef a escribir. Realiza una serie comprobaciones sobre la etiqueta detectada, como el tamaño de la misma, su formato, si es posible escribir en ella, etc. En caso de darse positivamente todas estas condiciones, se realizará la escritura de la etiqueta NFC correctamente, mostrando el éxito de la misma. En caso contrario, se mostrará que el proceso ha fallado.

Además de los métodos anteriores, se necesita sobrescribir los métodos *onResume()* y *onPause()*.

```
@Override
public void onPause() {
        super.onPause();
        if(mNfcAdapter != null)
        mNfcAdapter.disableForegroundDispatch(this);
}
@Override
public void onResume() {//prioritize activity in the foreground
        super.onResume();
        if(mNfcAdapter != null)
                mNfcAdapter.enableForegroundDispatch(this, mPendingIntent,mFilters,mTechLists);

        }
```

Al sobrescribir el método *onResume()* activando el ForengroundDispatch lo que se consigue es dar prioridad a la actividad frente a otras con respecto a la recepción o descubrimiento de una etiqueta NFC cercana. De esta manera evitaríamos que otra aplicación se abriera y leyera el contenido de la etiqueta NFC cuando se estaba en medio del proceso de escritura de la tarjeta. Sobreescribiendo el método *onPause()* desactivando el ForengroudDispatch, se anularía la prioridad recibida en el método *onResume()*.

Por último, se necesita declarar y asignar una serie de valores a diferentes variables.

```
...
private void setupForenground() {//Set up forengroud
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
```

```
        mPendingIntent = PendingIntent.getActivity(this, 0, new Intent(this,
        getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
        IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
        mFilters = new IntentFilter[] {ndef, };
        mTechLists = new String[][] { new String[] { Ndef.class.getName() }, new String[] { Ndef-
        Formatable.class.getName() }};
}
...
```

Con este método se consigue obtener acceso al adaptador NFC del dispositivo móvil, y seleccionar la lista de tipos de etiquetas que la aplicación podrá soportar.

Sincronización base de datos

Para la sincronización de las bases de datos, se tiene implementado el código que realiza las peticiones de sincronización de los datos. Es el siguiente:

Al pulsar el botón de sincronización, se llama a estas dos funciones:

```
syncMySQLDBSQLite(); //Sincronización web. Móvil envía datos a la web.
syncSQLiteMySQLDB(); //Sincronización móvil. Web envía datos al móvil.
```

La primera función, *syncMySQLDBSQLite(),* envía los datos no sincronizados desde el dispositivo móvil a al servidor .

```
public void syncMySQLDBSQLite() {
...
        AsyncHttpClient client = new AsyncHttpClient();
        RequestParams params = new RequestParams();
        ArrayList<HashMap<String, String>> userList = mydatabase.getAllData();
        if(userList.size()!=0) {
           if(mydatabase.dbSyncCount() != 0) {
              prgDialog.show();
              params.put("usersJSON",mydatabase.composeJSONfromSQLite());
               client.post("http://192.168.0.13:80/nfc/insertuser_.php",params ,new Asyn  cHttpRe-
               sponseHandler() {


               @Override
               public void onFailure(int statusCode, Header[] arg1, byte[] arg2, Throwable arg3){

                ... //Get Errors
                }
                @Override
                public void onSuccess(int arg0, Header[] arg1, byte[] arg2) {
                JSONArray arr = new JSONArray(cadena);
                for(int i=0; i<arr.length();i++) {
                       JSONObject obj = (JSONObject)arr.get(i);
                       mydatabase.updateSyncStatus (obj.get("relacion").toString(),
                       obj.get("objetoPadre").toString(), obj.get("objeto").toString(),
                       obj.get("interaccion").toString(),obj.get("tiempo").toString());
                       }

                   }
                });
        }
...
}
```

Esta función recoge los datos no sincronizados y los envía en formato JSON mediante peticiones HTTP asíncronas, de igual manera que en el caso del GCM. Posteriormente, espera una respuesta de los datos enviados. Si se devuelve un error, se ejecutaría la función *onFailure()*, en caso contrario, se ejecutaría la función *onSuccess()*. Esta última función recibe la misma cadena de datos enviada a modo de comprobación de que se han insertado la información en la base de datos externa correctamente. Por último, se actualizaría la base de datos del dispositivo móvil para señalar que los datos ya están sincronizados.

En la segunda función, *syncSQLiteMySQLDB()*, los pasos a seguir son exactamente iguales que en el caso anterior, pero a la inversa. Esta vez se recogen los datos a sincronizar de la base de datos del servidor, utilizando una vez más peticiones HTTP asíncronas. Una vez recogidos los datos a sincronizar, y desde la función *onSucces()* que denota el éxito al recoger los datos, se llama a la función *updateSQLite(cadena)* con la cadena obtenida para actualizar los datos en la base de datos del dispositivo móvil. El código es el siguiente:

```
public void updateSQLite(String response) {
...
        ArrayList<HashMap<String, String>> datasynclist;
        datasynclist = new ArrayList<HashMap<String, String>>();
        // Create GSON object
        Gson gson = new GsonBuilder().create();
        JSONArray arr = new JSONArray(response);
        if(arr.length() != 0) {
           for (int i = 0; i < arr.length(); i++) {
                 // Get JSON object
                 JSONObject obj = (JSONObject) arr.get(i);
                 queryValues = new HashMap<String, String>();
                queryValues.put("relacion", obj.get("relacion").toString());
                 queryValues.put("objetoPadre", obj.get("objetoPadre").toString());
                 queryValues.put("objeto", obj.get("objeto").toString());
                 queryValues.put("interaccion", obj.get("interaccion").toString());
                 queryValues.put("tiempo", obj.get("tiempo").toString());
                 updateMySQLSyncSts(gson.toJson(datasynclist));
               }
        }
...
}
```

Por último, se llamaría a la función *updateMySQLSyncSts(gson.toJson(datasynclist))*. Con esta función se pretende conseguir que la base de datos externa marque sus datos como ya actualizados. El código sería el siguiente:

```
public void updateMySQLSyncSts(String json) {
...
        AsyncHttpClient client = new AsyncHttpClient();
        RequestParams params = new RequestParams();
        params.put("sincro", json);
        client.post("http://192.168.0.13:80/nfc/updatesyncsts.php", params, new AsyncHttpRe-
        sponseHandler() {

        @Override
        public void onFailure(int arg0, Header[] arg1, byte[] arg2, Throwable arg3) {
        ... //Get Errors
        }
        @Override
        public void onSuccess(int arg0, Header[] arg1, byte[] arg2) {
        ...
        }

    });
...
}
```

Una vez más, utilizando una petición HTTP asíncrona se envían los datos a marcar como sincronizados en la base de datos externa. Como en casos anteriores, el método *onFailure ()* procesará los errores que se puedan producir y el método *onSuccess()* gestionará el éxito de la operación del envío de la cadena con los datos a marcar como actualizados.

CÓDIGO ENTORNO WEB

En este apartado, se comentarán las partes del código de interés del servidor. De igual manera que con el código del aplicativo móvil, el código está disponible en GitHub.

GCM lado del servidor

El código sobre el servicio Google, *"Google Cloud Messaging",* desarrollado en el lado del servidor es el siguiente.

En primer lugar, se recoge el RegID enviado desde el dispositivo móvil.

```
<?php

if(!empty($_GET["shareRegId"])) {
    $gcmRegID  = $_POST["regId"];
    file_put_contents("GCMRegId.txt",$gcmRegID);
    echo "Done!";
    exit;
}

?>
```

Este código comprueba que se está enviando el RegID a través de una variable puesta a true en la URL. El RegID se recoge y se almacena en un fichero para su posterior uso.

Cada vez que se introduzca o eliminen datos de la base de datos del servidor, se enviará un mensaje al dispositivo móvil. Para ello se hace lo siguiente:

```
...
if($res){
//Post message to GCM when submitted
   $pushMessage = "New Data";
   $gcmRegID  = file_get_contents("GCMRegId.txt");
   if (isset($gcmRegID) && isset( $pushMessage)) {
       $gcmRegIds = array($gcmRegID);
       $message = array("m" => $pushMessage);
       $pushStatus = $db->sendMessageThroughGCM($gcmRegIds, $message);
    }
}

...
```

Se recoge el mensaje a enviar y el RegID y se llama a la función *sendMessageThroughGCM* que enviará el mensaje al dispositivo móvil mediante el uso del servidor de mensajes GCM. El código de esta función es el siguiente:

```php
<?php
public function sendMessageThroughGCM($registration, $message) {
        //Google cloud messaging GCM-API url
        require_once 'config.php';
        $url = 'https://android.googleapis.com/gcm/send';
        $fields = array(
            'registration_ids' => $registration,
            'data' => $message,
        );
        // Update your Google Cloud Messaging APIKey
        $headers = array(
            'Authorization: key=' . GOOGLE_API_KEY,
            'Content-Type: application/json'
        );
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_POST, true);
        curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt ($ch, CURLOPT_SSL_VERIFYHOST, 0);
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));
        $result = curl_exec($ch);
        if ($result === FALSE) {
            die('Curl failed: ' . curl_error($ch));
        }
        curl_close($ch);
        return $result;
    }
}

?>
```

La función crea un array con el mensaje a enviar y el RegID recibidos por parámetros. Posteriormente se crea otro array con la APIKey del servidor y el tipo de contenido a enviar. Por último se envía al servidor de mensajes GCM una seria de parámetros como la URL de la API de Android GCM, la APIKey del servidor, opciones de envío seguro y por último, el array con el mensaje y RegID en codificación JSON. Por último, se almacena la respuesta de envío al servidor de mensajes GCM. Con estos datos, el servidor de mensajes GCM realizará las gestiones necesarias para hacer llegar el mensaje al dispositivo móvil deseado.

Inserción en la base de datos

En cuanto a la inserción de datos en la base de dartos, destaca lo siguiente:

```php
<?php
include_once './db_functions.php';
//Create Object for DB_Functions class
if(isset($_POST["relation"]) && !empty($_POST["relation"])&& isset($_POST["parentObject"]) &&
!empty($_POST["parentObject"])&& isset($_POST["object"]) && !empty($_POST["object"])&& is-
set($_POST["interaction"]) && !empty($_POST["interaction"])){
```

En primer lugar, comprobamos que todos los campos del formulario están iniciados y no son nulo mediante las funciones isset y empty.

```
    $db = new DB_Functions();
    //Store User into MySQL DB
    $relation = $_POST["relation"];
    $pObject = $_POST["parentObject"];
    $object = $_POST["object"];
    $interaction = $_POST["interaction"];
    $fecha = new DateTime();
    $time=$fecha->getTimestamp();
    $timestamp=idate('U', $time);
    $res = $db->storeUser($relation,$pObject,$object,$interaction,$timestamp,"0");
```

En caso de haber rellenado de forma correcta el formulario, se inicia una conexión con la base de datos. Se recogen los datos del formulario y se calcula el timestamp a insertar. Seguidamente, se realiza una inserción en la base de datos con los diferentes datos obtenidos.

Si la inserción fue correcta en la base de datos, se envía una notificación al dispositivo móvil mediante el GCM. Para ello se llama a la función sendMessageThroughGCM() con el RegID del dispositivo móvil y el mensaje a enviar.

```
?>
<div id="msg">Insertion successful.GCM status:<?php echo $pushStatus; ?></div>

<?php }else{ ?>
<div id="msg">Insertion failed</div>
<?php }
} else{ ?>
<div id="msg">Please enter name and submit</div>
<?php }

?>
```

Por último, se muestra un mensaje en la interfaz web sobre el resultado de la inserción y envío del mensaje al dispositivo móvil.

Eliminación de la base de datos

En cuanto a la eliminación de datos de la base de datos destaca lo siguiente:

```
<?php
include_once './db_functions.php';
//Create Object for DB_Functions clas
if(isset($_POST["tiempo"]) && !empty($_POST["tiempo"])){
```

En primer lugar se comprueba que el campo a rellenar no esté vacío ni sea nulo.

```
    $db = new DB_Functions();
    $tiempo = $_POST["tiempo"];
    $res = $db->delete($tiempo);
```

En segundo lugar, se inicia una conexión con la base de datos y se eliminan los datos señalados con el timestamp. Si el proceso de eliminación fue correcto, se envía un mensaje al dispositivo móvil de igual forma que en el caso de la inserción.

```
?>
<div id="msg">Deleted successfully.GCM status:<?php echo $pushStatus; ?></div>
<?php }else{ ?>
<div id="msg">Deleted failed</div>
<?php }
}  else{ ?>
<div id="msg">Please enter data and submit</div>
<?php }
```

```
       ?>
```

Por último, se muestra un mensaje en la interfaz web del éxito o error de la operación de eliminación y envío de mensaje.

Mostrar datos

En cuanto al código destaca el mostrar los datos y la recarga de la interfaz. El código es el siguiente:

Para la recarga de la interfaz cada dos segundos, se ha utilizado un sencillo script en JavaScript. Es el siguiente:

```
<script>
     var val= setInterval(function(){
     location.reload();
     },2000);
</script>
```

En cuanto al código de mostrar los datos, es el siguiente:

```
<?php
include_once 'db_functions.php';
$db = new DB_Functions();
$data= $db->getAllData();
```

En primer lugar se conecta con la base de datos y se traen todos los datos a mostrar.

```
if ($data != false)
    $no_of_rows = mysql_num_rows($rows);
else
    $no_of_rows = 0;

        ?>
<?php
if ($no_of_rows > 0) {
        ?>
```

A continuación se comprueba si existen datos a mostrar, es decir, se comprueba si la consulta ha sido devuelta vacía.

```
        <table>
            <tr
id="header"><td>Relation</td><td>ParentObject</td><td>ChildObject</td><td>Interaction</td><td>Tim
e</td><td>Sync</td></tr>
            <?php
    while ($row = mysql_fetch_array($data)) {
            ?>
            <tr>
                <td><span><?php echo $row["relacion"] ?></span></td>
                <td><span><?php echo $row["objetoPadre"] ?></span></td>
                <td><span><?php echo $row["objeto"] ?></span></td>
                <td><span><?php echo $row["interaccion"] ?></span></td>
                <td><span><?php echo $row["tiempo"] ?></span></td>
                <td id="sync"><span>
                    <?php
                if($row["sincro"]=="1")
            {
                echo "<img src='img/green.png'/>";
            }else if($row["sincro"]=="2")
            {
                echo "<img src='img/black.png'/>";
            }else  {
                echo "<img src='img/white.png'/>";
            }
                    ?></span></td>
            </tr>
            <?php } ?>
        </table>
        <?php }else{ ?>
        <div id="norecord">
            No records in MySQL DB
        </div>
        <?php } ?>
    </body>
</html>
```

Finalmente, si existen datos a mostrar, se recorren los datos devueltos y se van insertando de manera ordenada en una tabla dinámicamente construida. Destacar que en el caso del campo de la sincronización se comprobará su valor para insertar un color representativo u otro. Un 0 coincidiría con el estado *"no sincronizado"* (blanco), un 1 coincidiría con el estado *"sincronizado"* (verde) y un 2 coincidiría con el estado *"a la esperara de eliminar"*.