Universitat Oberta de Catalunya

uoc.edu

Uso de etiquetas NFC en supermercados para personas con baja visión

María Victorina Fernández San Román
Máster Universitario en Ingeniería de Telecomunicación
Área de Sistemas de Comunicación

Raul Parada Medina
Carlos Monzo Sánchez

10 de enero de 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Uso de etiquetes RFID en supermercados para personas con baja visión*
Nombre del autor:	*María Victorina Fernández San Román*
Nombre del consultor/a:	*Raul Parada Medina*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	01/2019
Titulación:	*Máster Universitario en Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Sistemas de Comunicación*
Idioma del trabajo:	*Castellano*
Palabras clave	*RFID, Baja Visión, NFC, Android, smartphone, aplicación.*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

En la actualidad, alrededor de 217 millones de personas sufren algún tipo de discapacidad visual moderada o grave, con sobre todo personas mayores de 50 años. Este trabajo pretende ayudar a la independencia de estas personas en las tareas cotidianas, concretamente a la hora de realizar la compra, mediante una aplicación para dispositivos Android que permitirá obtener, mediante tecnología NFC, la información básica de los productos disponibles en un supermercado: descripción, precio, alérgenos e información nutricional. Además se podrá ampliar dicha información y hacer uso de un asistente de voz para leerla en voz alta. También incluirá, para los empleados de los supermercados, la opción de escribir dicha información básica de los productos en sus correspondientes etiquetas NFC, completando así la funcionalidad al no ser necesario usar terceras aplicaciones.

Abstract (in English, 250 words or less):

Currently, around 217 million people are either moderate or grave visual impaired, especially among people over 50 years old. The aim of this work is to contribute to these people independency, in particular when it comes to do the shopping, by an Android application which will use NFC tecnology in order to get the products' basic information available in a supermarket. This basic information will include a short description, price, allergens and nutritional information. All this information will be zoomable, and it will be posible to use a

voice assistant as well, which will read the input out loud. Last but not least, the supermarket employees will be able to record the basic product information in each NFC label. This functionality will assure a complete product, since no extra third parties applications will be neccesary.

Índice

Lista de figuras

Lista de tablas

1. Introducción

A lo largo de este capítulo, se presentará la motivación para la realización del presente Trabajo, detallando el contexto actual, así como la carencia de soluciones que cubran los objetivos que se desean alcanzar. A continuación, se pasarán a describir dichos objetivos y se explicará el enfoque y el método seguido. En el cuatro apartado de este capítulo, se expondrá la planificación temporal del proyecto, mientras que el quinto recogerá brevemente el producto obtenido. Por último, este capítulo cerrará con una breve descripción del contenido del resto de la Memoria.

1.1 Contexto y justificación del Trabajo

Se define “baja visión” como un grado de visión que, aun siendo este parcial, permite utilizar la vista como canal primario para la obtención de información [1]. De una manera más concisa, hablamos de baja visión cuando esta está comprendida entre un rango de campo visual entre los límites máximo inferior de 20º y mínimo superior de 10º, y un límite máximo inferior 6/18 y mínima superior a 3/60 según la escala Snellen [2].

Aunque el número de afectados se está viendo reducido a lo largo de los años, se estima que en el año 2017 existían 217 millones de personas con alguna discapacidad visual moderada o grave, con un gran porcentaje asociado a la edad, incidiendo entre personas mayores de 50 años [3]. Para todas ellas, una rehabilitación visual que persiga la mayor calidad de vida posible en el paciente es imprescindible. Esta rehabilitación abarca desde una correcta evaluación y prescripción de dispositivos de ayuda, hasta una re-educación a la hora de realizar las tareas cotidianas. Por ejemplo, una persona afectada de escotoma (es decir, con un punto ciego en su visión), deberá entrenar sus habilidades oculares para aprender a encontrar su LRP (Locus Retiniano Preferente), que sustituirá al punto ciego o mácula en las funciones visuales que anteriormente desempeñaba [4].

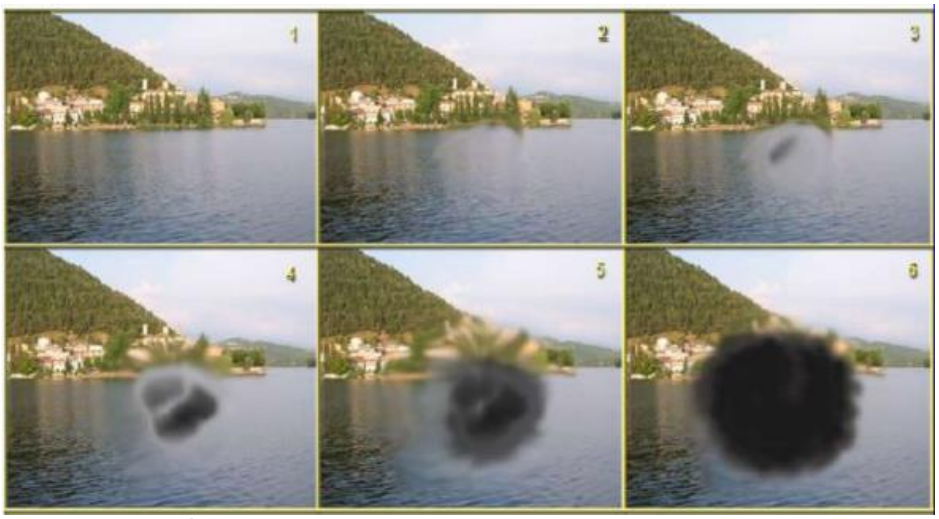


Figura 1: Evolución de un escotoma central de origen macular. Fuente: UNAH [5]

Al tratarse de una patología que por lo general se desarrolla con la edad, es frecuente que la persona que percibe una pérdida visual cada vez mayor desarrolle una falta de confianza en la información recibida, ya que la mayor parte de las percepciones de nuestras actividades habituales llegan a través de la vista.

Será por tanto necesario reajustar esta confianza en la vista y delegar parte en el resto de los sentidos. Se considerará que la integración es real y efectiva una vez que la persona afectada con esta discapacidad visual haya recuperado sus habilidades generales, haciendo uso para ello de ayudas ópticas (e incluso no ópticas) que le permitan desarrollar el potencial perceptivo que se ha visto disminuido.

Pero para que se de esta integración, la persona deberá aceptar que su pérdida visual es irreversible, lo cual no es sencillo desde un punto de vista psicológico. A menudo, la persona afectada pasará por una etapa de sufrimiento que puede llegar a convertirse en depresión.

Para poder comprender mejor la dificultad que entraña esta etapa de aceptación y superación, a continuación se enumeran los distintos componentes visuales que se pueden ver afectados, y su relación con las actividades del día a día del individuo [6]:

- Disminución de agudeza visual: mayor riesgo de caídas, movilidad limitada, y mayor dificultad en las actividades de la vida diaria, así como en el rendimiento físico.
- Disminución del campo visual. Aumenta el riesgo de accidente de coche.
- Empeoramiento de la visión subjetiva. Efecto negativo de la cognición, del afecto y del estatus funcional. Todo ello aumenta la probabilidad de ingreso en una residencia para la tercera edad.

Anteriormente se han mencionado las ayudas ópticas y no ópticas de las que un individuo con baja visión deberá hacer uso. Algunas de las ayudas ópticas son las siguientes [7]:

- Filtros selectivos. Protegen y aportan contraste tanto en visión diurna como nocturna.
- Microscopios, telemicroscopios y lupas con y sin iluminación para mejorar la lectura y la escritura.
- Telescopios. Mejoran la visión lejana en posición estática.
- Telelupas. Proporcionan hasta 60 aumentos, con diferentes tipos de pantallas y opciones.
- Sistemas biópticos para visualización lejana y próxima.

Figura 2: Telelupa o lupa electrónica. Fuente: Noticias Vigo [8]

Entre las ayudas no ópticas, se encuentran:

- Flexos de luz fría. Mejoran la iluminación y el contraste.
- Atriles. Optimizan la posición de lectura.
- Teclados para ordenador con macrotipos y diferentes fondos.
- Relojes con voz.

Las nuevas tecnologías también pueden ayudar a facilitar el día a día en las personas afectadas por la baja visión. Por ejemplo, tanto muchos libros electrónicos como *smartphones* y ordenadores permiten la lectura de texto en voz alta, el control del contraste y brillo en la pantalla, o el cambio de tamaño de las palabras. Sin embargo, algunas tareas cotidianas tales como cocinar o salir de compras no son fácilmente adaptables, y el no poder llevarlas a cabo con independencia pueden ayudar negativamente a generar esos sentimientos de frustración que ya hemos mencionado con anterioridad.

1.2 Objetivos del Trabajo

El objetivo de este proyecto es la creación de un sistema que sirva para mejorar la experiencia de las personas afectadas por baja visión en sus

visitas cotidianas a los supermercados, ayudándolas a realizar la compra de una manera autónoma.

Para ello, se contará con un sistema de dos tipos de etiquetas NFC pasivas. Las primeras servirán para indicar el pasillo o sección de supermercado, para que el usuario pueda saber qué clase de productos esperar a lo largo de dicha sección. Por otra parte, las segundas irán asociadas cada una de ellas a un producto en cuestión y recogerán los aspectos más importantes del mismo: breve descripción del artículo, precio, alérgenos e información nutricional.

Todas ellas podrán ser leídas mediante una aplicación Android instalada en un *smartphone* con tecnología NFC, que será capaz de leer la etiqueta NFC y mostrar por pantalla la información almacenada. Esta lectura se realizará sin necesidad de pulsar ningún botón o realizar algún otro tipo de interacción con la aplicación. Simplemente, desde la pantalla principal y una vez la etiqueta esté a su debida distancia, la información se mostrará automáticamente.

Para una correcta interpretación de dicha información, será posible tanto ampliar el texto como solicitar a la aplicación que lo lea en voz alta. En la primera de las opciones, se hará uso del ya conocido y extendido gesto del “pellizco” para ampliar la zona que contiene el texto. En cuanto a la lectura en voz alta, al considerarlo como una ayuda suplementaria, se optará por implementar un botón que, al pulsarlo, lea la información.

Con el fin de completar la funcionalidad de nuestra aplicación, también se incluirá un módulo de gestión, dirigido a los responsables o encargados de cada supermercado. Este módulo será accesible vía usuario y contraseña, y permitirá escribir en una etiqueta NFC la información del producto que posteriormente podrá ser leída por las personas con baja visión. El acceso a este módulo estará restringido a los usuarios normales dada la naturaleza del mismo.

1.3 Enfoque y método seguido

La solución planteada en este trabajo pretende mitigar el sentimiento de ruptura con la rutina en las personas cuya visión se va degenerando con el tiempo. Es decir, busca ser un soporte que ayude a realizar la compra con la mayor facilidad posible, y no un método alternativo. Un ejemplo de dicho método alternativo sería la compra *on-line*, que como inconveniente cuenta con la limitación al usuario en variedad (no todos los supermercados cuentan con dicha opción, o el catálogo disponible no es tan amplio como en la tienda física), y cantidad, ya que conceptos como los gastos de envío o el importe mínimo hacen que esta solución no sea factible para la mayoría de las compras del día a día.

Se han encontrado proyectos que, de una manera u otra, tratan de proporcionar una solución a la misma problemática. Por ejemplo,

RightHear [9] es una aplicación disponible para dispositivos iOS y Android que hace uso de la tecnología Bluetooth para servir de guía a lo largo de pasillos, estanterías y cajas de cobro [10]; mientras que *Third Eye*, que forma parte del proyecto Visual Cortex on Silicon [11], se sirve de un guante táctil equipado con una webcam y busca crear un sistema electrónico que funcione como el córtex visual humano [12]. Ambos enfoques han sido descartados ya que *RightHear* no proporciona un alto grado de detalle en el producto, sino que está más enfocado a dar información de carácter general a personas con ceguera para ayudarlas a moverse dentro de tiendas, hospitales, etc.; mientras que *Third Eye* es una solución altamente costosa (además de compleja), que impediría que llegara al público en general.

1.4 Planificación del Trabajo

En primer lugar, para la realización del presente trabajo será necesario acotar nuestros objetivos, así como desarrollar un plan de trabajo en base a esos objetivos y estimar los resultados que se obtendrán.

Una vez claros los puntos anteriores, necesitaremos conocer el estado del arte de nuestro proyecto. En nuestro caso, los proyectos existentes relacionados con este se utilizaron en la primera etapa para contextualizar y acotar nuestros objetivos, por lo que, en nuestro caso, el estado del arte cubrirá la evolución de la tecnología, desde la historia del RFID hasta el uso del NFC hoy en día en una gran variedad de dispositivos.

La tercera etapa se centra en el desarrollo de la aplicación. Con este fin, en primer lugar será necesario escoger un IDE (Entorno de Desarrollo Integrado) en el que desarrollar nuestra aplicación. Además, necesitaremos ejecutar continuamente el código de nuestra *app* para comprobar los cambios que se vayan implementando. Por este motivo, también tendremos que hacernos con un teléfono móvil que cuente con tecnología NFC, así como con varias etiquetas NFC que nos sirvan para nuestros *tests*.

Tanto el entorno de desarrollo, como el modelo de *smartphone* o el tipo de etiqueta deberán ser cuidadosamente seleccionados. En caso del IDE, necesitaremos estudiar todas las opciones disponibles y escoger aquella que nos permita desarrollar la aplicación de una manera eficiente. Por otra parte, las etiquetas NFC deberán tener la capacidad suficiente para almacenar toda la información requerida, así como contar con un estándar de amplia compatibilidad con distintos dispositivos. Finalmente, en cuanto al modelo de dispositivo, deberemos estudiar qué marca o modelo es el más extendido entre los consumidores.

Antes de empezar a escribir código para nuestra aplicación, deberemos realizar un diseño de la interfaz de la misma. Este punto es especialmente importante para nuestro proyecto, debido a las

necesidades especiales del público al que va dirigido la *app*. Por tanto, se deberá optar por una interfaz lo más sencilla posible para la pantalla principal, mientras que el módulo destinado a la gestión no tendrá, en principio, ninguna limitación de diseño.

Durante esta tercera etapa, se estima que la parte más costosa temporalmente sea la dedicada a desarrollar el código para implementar el *zoom* o lupa en la aplicación. Esto se debe, además de a la dificultad intrínseca de la funcionalidad, a la propia curva de aprendizaje del IDE escogido y del lenguaje de programación, que será Java en este caso.

Si bien durante las etapas anteriores se ha ido trabajando a la vez en la redacción de la memoria del TFM, se reservarán tres semanas para la revisión y mejora de la misma.

Por último, se deberá realizar una presentación y/o vídeo explicativo que suponga la defensa del trabajo realizado.

Todas estas etapas se recogen en el diagrama de Gantt que se muestra a continuación:

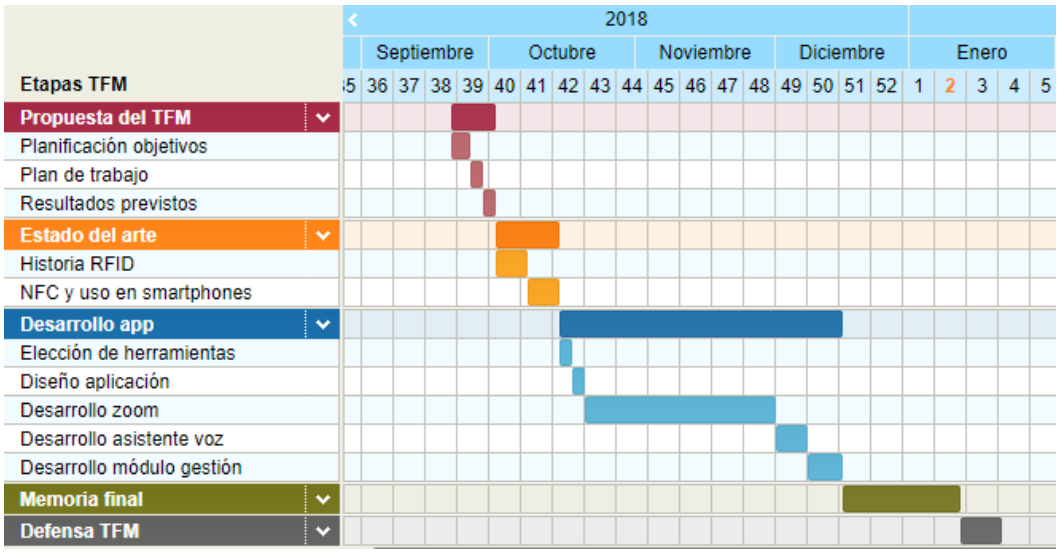


Tabla 1: Diagrama de Gantt del TFM, visualizado por semanas.

1.5 Breve sumario de productos obtenidos

El producto obtenido al final del desarrollo de este Trabajo Fin de Máster será una aplicación Android destinada a la lectura y escritura de etiquetas NFC. Estas etiquetas NFC deberán contener la información más relevante del producto del supermercado al que se encontrarán asociadas.

La lectura de las etiquetas NFC estará destinada a personas con necesidades visuales especiales, y la escritura a los responsables o

encargados del supermercado, por lo que las necesidades en cada caso serán diferentes.
Por estos motivos, nuestra solución cuenta con tres interfaces diferentes:

- Pantalla principal. Compuesta de una vista de texto donde se mostrará la información de cada etiqueta, y dos botones: el primero de ellos permitirá la reproducción de texto por el asistente de voz, y el segundo nos conducirá al módulo de gestión.
- Pantalla de acceso. Esta interfaz mostrará un campo de usuario y otro campo de contraseña, y servirá para dar acceso al módulo de gestión.
- Pantalla de gestión. Constará de un cuadro de texto en el que se introducirá la información a escribir, y dos botones: uno para realizar dicha escritura, y otro para volver a la pantalla principal.

1.6 Breve descripción de los otros capítulos de la memoria

Durante el siguiente capítulo de esta memoria, se estudiará el estado del arte de la tecnología NFC, desde el origen e historia del RFID hasta la evolución del NFC hasta nuestros días.

A continuación, durante el capítulo 3 se justificará cada una de las elecciones previas al desarrollo del trabajo: tecnología a utilizar, sistema operativo al que va orientada la aplicación, tipo de dispositivo y marca en el que se implementará e IDE en el que desarrollar el código. También se profundizará en el código desarrollado y se mostrarán los resultados del mismo.

El capítulo 4 tratará las conclusiones de este TFM, así como una reflexión crítica sobre el logro de los objetivos planteados y el seguimiento de la planificación y metodología. También se describirán aquí las posibles líneas de trabajo futuro.

Los capítulos posteriores estarán dedicados a un Glosario de términos y acrónimos más utilizados (capítulo 5), la Bibliografía de esta Memoria (capítulo 6), y Anexos (capítulo 7), en el que se incluirá el código de la aplicación.

2. Estado del arte

En este capítulo, dividido en dos apartados diferentes, se estudiará la evolución de la tecnología que se va a emplear en este TFM. En primer lugar, se detallará la historia del RFID, y a continuación se explicará el nacimiento del NFC y sus aplicaciones.

2.1. Historia de RFID

RFID (*Radio Frequency Identification*) es la tecnología de radio que se usa para comunicarse entre unas etiquetas y sus lectores. Aunque pueda parecer una tecnología sencilla, abarca teoría de circuitos, teoría de antenas, técnicas de microondas, diseño de receptores, ingeniería de redes, ingeniería de sistemas, diseño de circuitos integrados, etc. A continuación, se repasará la evolución de esta tecnología desde la década de 1960 hasta la actualidad.

La actividad comercial de la tecnología RFID empezó en los años 60. Por aquel entonces, varias compañías comenzaron a desarrollar material electrónico de vigilancia contra el robo de mercancía. Estos sistemas, en su mayoría, hacían uso de microondas, generando armónicos mediante un semiconductor; o se servían de circuitos resonantes. Además, únicamente permitían detectar la presencia o ausencia de etiqueta. En aquellos años, las etiquetas con múltiples bits se encontraban en fase experimental, y dadas las limitaciones en la circuitería, su tamaño era considerable. A pesar de esto, el bajo coste de las etiquetas básicas anteriormente descritas, unido a su alta eficacia frente a los robos, sirvió para que la tecnología RFID fuese un éxito.

En la década de los 70 aumentó notablemente la investigación en torno a RFID, y los avances que se pudieron llevar a cabo fueron notables: el uso de circuitos CMOS de bajo consumo, así como de *arrays* de diodos fusibles para la memoria de las etiquetas en lugar de interruptores y cableado ayudaron al avance de la tecnología, pero posiblemente el desarrollo más importante vendría de la mano de Koelle *et al*. [13], que supuso el nacimiento de etiquetas completamente pasivas con un rango operacional de decenas de metros.

La aparición de los ordenadores personales en los años 80 impulsó rápidamente la expansión de las aplicaciones RFID, ya que simplificaba enormemente la recolección y el manejo de la información producida por los sistemas RFID. Además, las etiquetas se empezaron a construir mediante circuitos integrados CMOS a medida, y la memoria EEPROM se convirtió en la más utilizada debido a la posibilidad de su fabricación a gran escala y posterior programación individual.

Durante la década posterior el crecimiento tampoco cesó, siendo ya imposible estimar cuántas empresas decidieron entrar en el mercado.

En estos años, los diodos Schottky fueron incluidos en un circuito integrado CMOS, lo que permitió la fabricación de etiquetas RFID con un único circuito integrado.

A principios de los 2000 las etiquetas ya podrían fabricarse como pegatinas, y el tamaño comenzaba a estar limitado por el de la antena, lo que suponía un desafío constante para la mejora del diseño, junto con la búsqueda de memorias no volátiles aún mejores. Además, la posibilidad de incluir diodos microondas en silicio en la misma etiqueta que el circuito ha permitido reducir el tamaño de la misma, así como reducir su coste a la vez que aumenta su funcionalidad y fiabilidad.

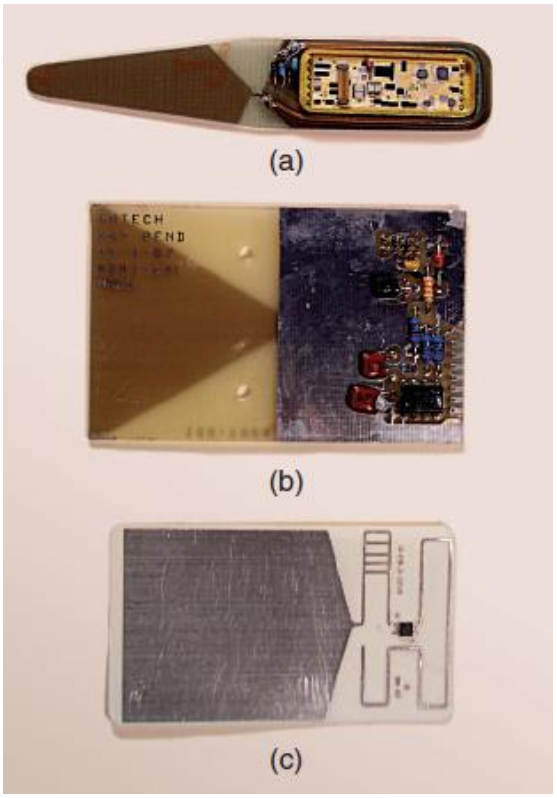


Figura 3: Evolución de las etiquetas RFID. (a) 1976. 12 bits, el circuito supone la mitad de la etiqueta. (b) 1987. 128 bits, el circuito ocupa una cuarta parte de la etiqueta. (c) 1999. 1024 bits, el circuito supone un área insignificante. Fuente: *The history of RFID* [14].

2.2. Nacimiento de NFC y uso en *smartphones*

El término NFC (*Near Field Communication*) fue acuñado en 2004 por el NFC Forum, formado por Nokia, Philips y Sony. Sin embargo, no se trata de una tecnología nueva, sino que nace de RFID.

NFC opera a una frecuencia de 13.56 MHz y es una extensión de los estándares RFID. Evidentemente, comparte muchas propiedades físicas con RFID, tales como la posibilidad de comunicación sin la existencia de una línea directa de visión, pero presenta tres diferencias clave [15]:

1. NFC permite la comunicación en ambos sentidos, por lo que puede usarse para interacciones más complejas.
2. El alcance de NFC está limitado a unos pocos centímetros debido a motivos de seguridad, aunque en teoría se podrían llegar a alcanzar distancias diez veces mayores.
3. Solo se puede escanear una etiqueta NFC al mismo tiempo.

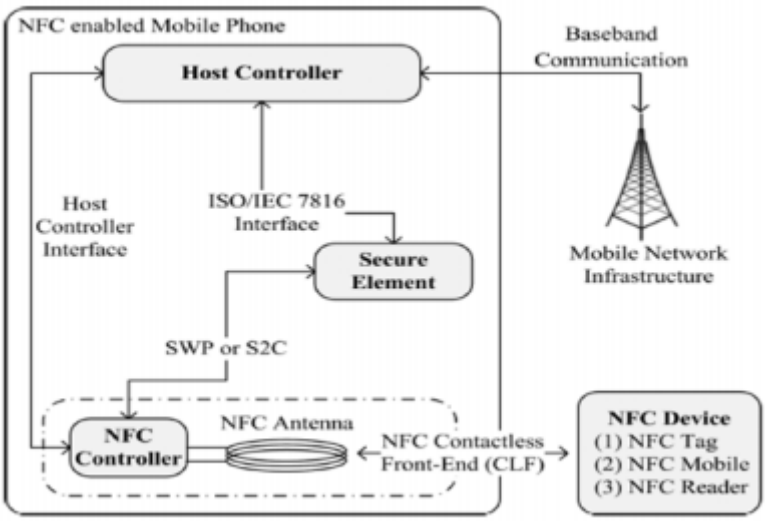


Figura 4: Arquitectura NFC integrada en un teléfono móvil. Fuente: Emerging Technologies for User-Friendly Mobile Payment Applications [16]

Durante el año 2006 comenzaron a crearse las primeras especificaciones para las etiquetas NFC, pero también para los *Smart posters y Smart tags*. Estas tecnologías permitían obtener desde información de un producto hasta música e incluso vídeos. Este mismo año, Nokia lanzó al mercado el Nokia 6131, primer teléfono móvil con chips NFC integrados.

En el año 2009, las aplicaciones P2P (*Peer to Peer*) se emplearon en la tecnología NFC, permitiendo así compartir todo tipo de archivos entre móviles a través de una conexión NFC.

En 2010 llegaría de la mano de Samsung el Nexus S, primer *smartphone* con tecnología NFC integrada. Hoy en día son muchas las marcas que optan por incluir esta característica en sus teléfonos móviles de gama media-alta, y es relativamente sencillo encontrar un modelo con NFC que se adapte a nuestras exigencias y nuestro presupuesto [17].

Algunos de los usos cotidianos que se le está dando a la tecnología NFC son los siguientes:

- Pagos. Sin duda se trata del uso con mayor desarrollo en los últimos años. La comodidad y facilidad que este método reporta al usuario ha hecho que tanto bancos como empresas de telefonía se interesen en integrar este servicio lo más rápido posible.

- Identificación. Por ejemplo, para acceder a ciertos lugares donde es preciso que el usuario se identifique.
- Recogida de datos. Gran ejemplo de esto es Google.
- Sincronización. NFC resulta más cómodo a la hora de sincronizar diferentes dispositivos, ya que al contrario que Bluetooth, no es necesario un emparejamiento previo.
- Automatización. Es posible configurar las etiquetas NFC para que realicen ajustes de manera automática al pasar el teléfono móvil cerca de ellas. Un ejemplo sería una etiqueta a la entrada de una biblioteca que desactivara el sonido y activase el WiFi.

3. Trabajo desarrollado

A lo largo de este tercer capítulo, se razonarán todas las motivaciones que llevaron a utilizar la tecnología NFC (3.1), desarrollar la aplicación para Android y escoger Samsung como la marca sobre la que probar la aplicación (3.2), y decantarnos por Android Studio como entorno de desarrollo (3.3). Además, el apartado 3.4 contendrá una descripción detallada del producto desarrollado.

3.1. Etiquetas NFC

Durante el planteamiento inicial de este proyecto, se barajaron diferentes opciones que finalmente fueron descartadas en pro de las etiquetas NFC. A continuación, se pasa a describir brevemente cada una de ellas y justificar la correspondiente selección.

3.1.1. Código de barras

La principal ventaja de los códigos de barras es evidente: vienen estampados en el embalaje de cada producto, por lo que el coste asociado a la fabricación y posterior colocación de las etiquetas es nulo. Sin embargo, la información que almacena cada uno de ellos es fija y se corresponde con el código del país, código de la empresa, código del producto y dígito de control [18].

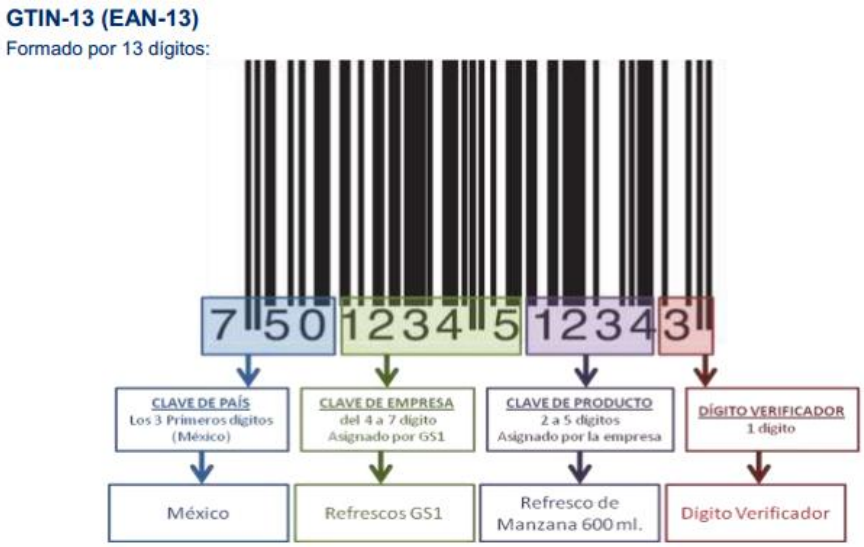


Figura 5: Estructura del código de barras EAN-13. Fuente: Softseti [19]

Esto supondría que cada supermercado debería de crear y mantener una base de datos que almacenara la información correspondiente a cada producto (más de 25.000 en el caso de grandes superficies [20]), y que dicha información fuera enviada a cada cliente que así lo solicitara mediante la *app* desarrollada.
Además, deberemos tener en cuenta el tipo de usuarios que utilizarán la aplicación. Dado que está dirigida a personas con baja visión o incluso

invidentes, el hecho de tener que utilizar la cámara del *smartphone* para escanear el código de barras no resulta una opción muy accesible.

3.1.2. Códigos QR

A diferencia de los códigos de barras, los códigos QR son capaces de almacenar hasta 4.296 caracteres alfanuméricos en su versión 40, compuesto de 177x177 módulos [21].

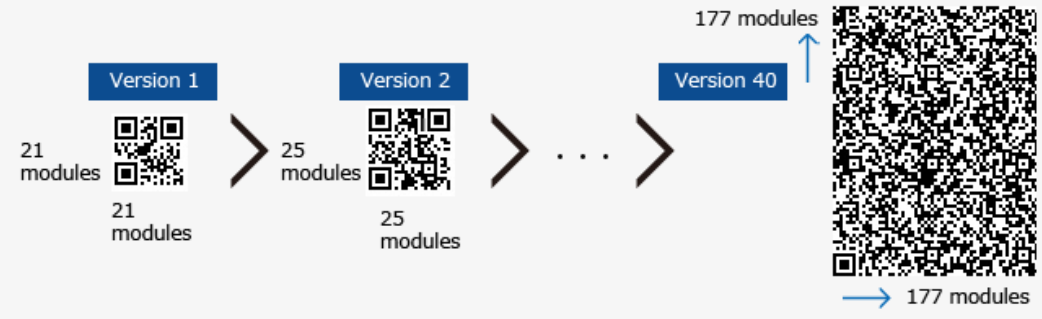


Figura 6: Evolución de los módulos en las distintas versiones de códigos QR.
Fuente: QR code.com [21]

Aunque su capacidad sería suficiente para la información que nos interesa almacenar, estos códigos no son reprogramables. Es decir, si fuera necesario actualizar la información de un determinado producto debido a una variación en su composición, no sería posible modificar la información almacenada en el código, sino que habría que generar uno nuevo que recogiera los cambios.
Pero este no es el mayor de sus inconvenientes. Al igual que los códigos de barras, la lectura de estos códigos requiere hacer uso de la cámara del teléfono móvil, lo que sería una mala experiencia de usuario como ya se ha explicado anteriormente.

3.1.3. Etiquetas NFC

Al contrario que las otras dos etiquetas descritas, el escaneo de las etiquetas NFC no se hace a través de una cámara, sino que simplemente requiere acercar el *smartphone* a la etiqueta para recoger la información almacenada en la misma, siendo este un método mucho más rápido y sencillo. Además, este tipo de etiquetas sí pueden ser sobrescritas en caso de que sea necesario. Es decir, los posibles cambios en la información que cada etiqueta almacena no supondrán un coste extra, al poder reutilizar las existentes y no tener que adquirir nuevas.
Aunque se podría pensar que el hecho de tener que contar con la tecnología NFC en el terminal es un gran inconveniente, la realidad es que gracias al auge del pago mediante NFC, se estima que en la actualidad más del 85% de los teléfonos móviles ya cuentan con esta tecnología [22].

De entre las diferentes etiquetas NFC existentes, se optó por una con chip NTAG, ya que estas son compatibles con todos los teléfonos y tabletas equipadas con tecnología NFC [23]. Dentro de las posibilidades (NTAG203, NTAG212, NTAG213, NTAG215 y NTAG216), se estableció un compromiso entre la capacidad de almacenamiento y el precio, por lo que se optó por las NTAG215, con una memoria de 504 bytes y un precio unitario, en el momento de su compra, de aproximadamente 0.64€.

Figura 7: NTAG215 escogida. Fuente: Amazon.es [24]

3.2. Aplicación Android

La elección del teléfono móvil para la obtención de la información almacenada en las etiquetas NFC resulta evidente: en la actualidad, alrededor del 72.5% de la población española posee un *smartphone*. Pero la penetración de estos dispositivos es incluso mayor en otros países europeos, tales como Reino Unido o Alemania, donde el porcentaje sube hasta el 82.2% y el 78.8%, respectivamente [25]. Además, en el caso de que el usuario no contase de antemano con un teléfono móvil con tecnología NFC, en la actualidad pueden adquirirse por menos de 150€ [26].

Por otro lado, la elección de Android como sistema operativo frente a iOS se basó en la cuota de mercado de cada uno de ellos. Según datos de Statista, en el año 2017, el 85.9% de los smartphones contaban con Android como SO [27]. En cuanto al modelo escogido, también se ha tenido en cuenta el volumen de ventas de los distintos fabricantes que hacen uso de Android. El tercer trimestre de 2018 cerró con Samsung con un 18.9% de cuota de mercado [28]. Así pues, se ha optado por un Samsung Galaxy S7 como modelo de referencia para el desarrollo de la aplicación, si bien es cierto que otros competidores como Huawei se están abriendo paso cada vez más.

Vendor	3Q18 Units	3Q18 Share	3Q17 Units	3Q17 Share
Samsung	73.360,1	18,9 %	85.605,3	22,3 %
Huawei	52.218,4	13,4 %	36.501,8	9,5 %
Apple	45.746,6	11,8 %	45.441,9	11,8 %
Xiaomi	33.219,7	8,5 %	26.853,2	7,0 %
OPPO	30.563,4	7,9 %	29.449,2	7,7 %
Others	153.960,0	39,6 %	159.742,0	41,6 %
Total	**389.068,2**	**100,0 %**	**383.593,4**	**100,0 %**

Tabla 2: Comparación ventas 3Q17 vs 3Q18. Fuente: Gartner [29]

3.3. IDE (Entorno de Desarrollo Integrado)

A continuación, se definirán los diferentes entornos de desarrollo integrado (IDE, *Integrated Development Environment*) que se barajaron para el desarrollo del software, y se justificará la elección de Android Studio frente a sus competidores.

3.3.1. IntelliJ IDEA

Diseñado por Jet Brains, incluye múltiples herramientas para el desarrollo de aplicaciones móviles y tecnologías empresariales para distintas plataformas, proporcionando soporte para Java, pero también para muchas otras tales como CSS, HTML, JavaScript, PHP, Python, Ruby, Node JS, etc. Además, cuenta con una gran variedad de plugins, desarrollados tanto por Jet Brains como por terceras partes a través de su SDK.

También cuenta con una interfaz muy intuitiva que incluso permite cierto grado de personalización; asistencia contextual que permite al desarrollador autocompletar su código, ver los objetos disponibles e importar sugerencias; una documentación clara, exhaustiva y fácil de navegar; y una estabilidad y robustez que permite no tener que reiniciar el proyecto si alguno de los archivos se corrompe.

La principal desventaja de este IDE es su precio: 499 euros durante el primer año, cuota que va disminuyendo en los años sucesivos [30].

3.3.2. Eclipse

Eclipse es un editor de código abierto y de uso gratuito que fue creado originalmente como un entorno únicamente Java. En la actualidad, sin embargo, sus capacidades han aumentado enormemente debido a la gran cantidad de complementos y extensiones disponibles. Al igual que IntelliJ IDEA, también soporta otros lenguajes además de Java: C, C++, PHP, Python, Perl, Ruby, etc.

Entre las ventajas de Eclipse, cabe destacar su alto grado de personalización, además de la gran cantidad de *plugins* disponibles gracias a la amplia comunidad de desarrolladores, si bien estos puede que no cuenten con un soporte activo y la resolución de errores sea lenta.

Sin embargo, la disponibilidad de los diferentes *plugins* y características en Eclipse es mucho más lenta que en IntelliJ IDEA o Android Studio, e incluso es posible que el plugin ADT (Android Development Tools) pierda completamente el soporte en el futuro [31].

3.3.3. Android Studio

Android Studio es actualmente el IDE oficial para el desarrollo de aplicaciones Android, como en su día lo fue Eclipse. Ya que ha sido creado por Google, este proporciona soporte para sus servicios, permitiendo que la integración de *Google Services* sea muy sencilla. Además, basa en IntelliJ IDEA, por lo que las características de este también se encuentran presentes en Android Studio, pero de manera gratuita en este caso.

Otra ventaja que presenta es la herramienta de desarrollo Gradle, que permite llevar a cabo tareas imposibles de realizar con otras herramientas, tales como la actualización de la versión del sistema sin que esto perjudique el proyecto, o la definición de manera separada de las versiones de desarrollo y de producción.

Merece la pena destacar también la velocidad de este IDE, que con su característica "*Instant Run*" permite desarrollar la aplicación una única vez y actualizar únicamente aquellas partes de código que han sido modificadas, lo que conlleva que el desarrollador es capaz de ver esos cambios en funcionamiento un par de segundos. También contribuye a la velocidad su emulador, que puede llegar a ser hasta 10 veces más rápido que si hubiera que utilizar un dispositivo físico.

El mayor inconveniente de Android Studio es precisamente que únicamente admite el desarrollo de aplicaciones Android, por lo que sería necesario hacer uso de otros IDEs en caso de querer desarrollar aplicaciones para otras plataformas en Java.

Con todo lo explicado anteriormente, se descartan IntelliJ IDEA y Eclipse como los posibles IDEs a utilizar, debido al precio del primero y las limitaciones del segundo; y se opta por Android Studio, cuyo único gran inconveniente no es una limitación dada la naturaleza de la aplicación.

3.4. Estructura y funcionalidad de la aplicación

Por el tipo de usuarios a los que va dirigida, esta aplicación permite ser funcional desde el momento en el que se abre, para así facilitar su uso.

Por este motivo se ha prescindido de pantallas de inicio que, aunque estéticamente pueden resultar atractivas, no solo no aportarían ningún valor añadido a personas con baja visión e invidentes, sino que supondrían un inconveniente.

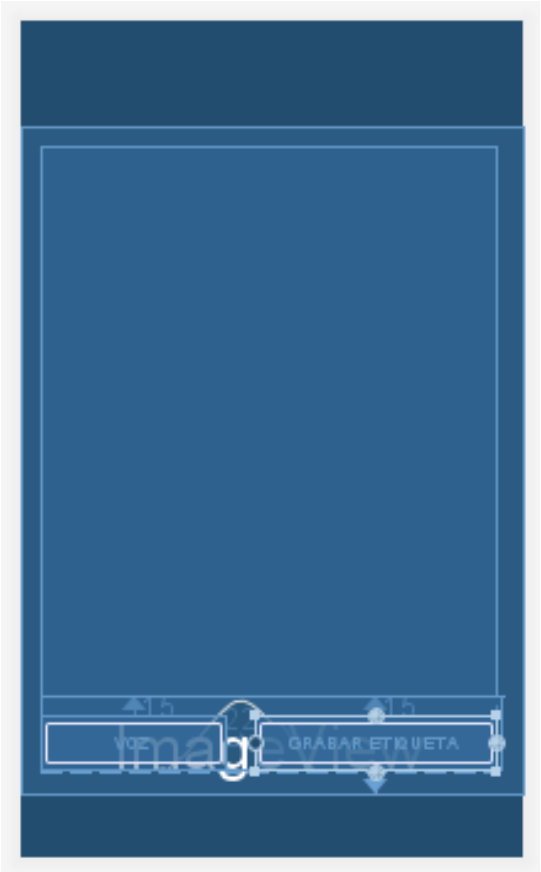
Nuestra aplicación, SuperNFCado, cuenta con tres interfaces distintas. El código más relevante asociado a la aplicación puede encontrarse en el capítulo 7, Anexo. Cabe destacar, no obstante, que para indicar a la aplicación que ha de hacer uso de la tecnología NFC, se deben incluir las siguientes líneas de código en el archivo *AndroidManifest.xml*, automáticamente generado por Android Studio:

```
<uses-permission android:name="android.permission.NFC" />
<uses-feature android:name="android.hardware.nfc"
android:required="true"/>
```

También se debe indicar que el contenido que se va a leer es texto plano:

```
<action android:name="android.nfc.action.NDEF_DISCOVERED" />
<category android:name="android.intent.category.DEFAULT"/>
<data android:mimeType="text/plain" />
```

Por otra parte, para el diseño de las distintas interfaces, Android Studio hace uso de archivos XML y nos permite dos opciones diferentes: generar el código a través del diseño, o bien crear el diseño a medida que vamos escribiendo el código.




```
<TextView
    android:id="@+id/message"
    android:layout_width="348dp"
    android:layout_height="421dp"
    android:scrollbars = "vertical"
    android:textStyle="bold"
    android:textColor="#000000"
    android:gravity="center"/>

<Button
    android:id="@+id/btn_write"
    android:layout_width="184dp"
    android:layout_height="43dp"
    android:layout_below="@+id/message"
    android:layout_alignEnd="@+id/message"

android:layout_alignRight="@+id/message"
    android:layout_alignParentBottom="true"
    android:layout_marginTop="15dp"
    android:text="GRABAR ETIQUETA" />

<Button
    android:id="@+id/btn_voice"
    android:layout_width="142dp"
    android:layout_height="43dp"
    android:layout_below="@+id/message"

android:layout_alignStart="@+id/message"
    android:layout_alignLeft="@+id/message"

android:layout_alignBottom="@+id/btn_write"
    android:layout_marginTop="15dp"
    android:layout_marginEnd="22dp"
    android:layout_marginRight="22dp"

android:layout_toStartOf="@+id/btn_write"

android:layout_toLeftOf="@+id/btn_write"
    android:text="VOZ" />
```

Tabla 3: Diseño de la pantalla principal y código asociado.

El diseño de la primera de las interfaces, la pantalla de inicio, se encuentra recogido en el archivo *activity_main.xml*, y los diferentes métodos que permiten la funcionalidad se recogen en un fichero Java llamado *MainActivity.java.* Desde esta interfaz se podrá leer una etiqueta NFC, activar el asistente de voz para que lea la información en voz alta, o pasar a la segunda interfaz.

Dado que la aplicación está enfocada a personas con necesidades visuales especiales, la lectura de una etiqueta NFC se realiza tan solo acercando el dispositivo a dicha etiqueta, sin necesidad de ningún otro paso intermedio, como por ejemplo, pulsar un botón que permita la lectura. Así pues, la pantalla principal se compone de una amplia zona destinada a mostrar el texto de las etiquetas, y dos sencillos botones: uno para que el asistente de voz lea la información contenida en las etiquetas, y un segundo botón para acceder al módulo de gestión en el que se podrán grabar las etiquetas.

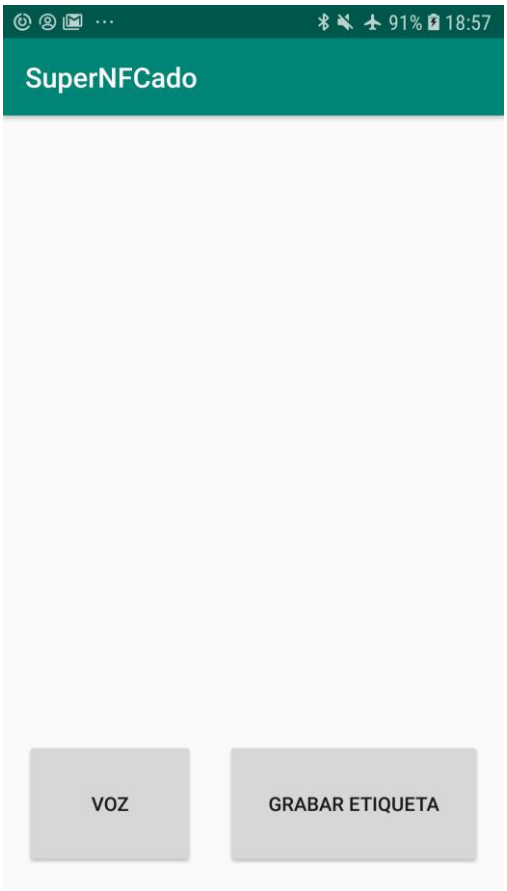


Figura 8: Pantalla principal

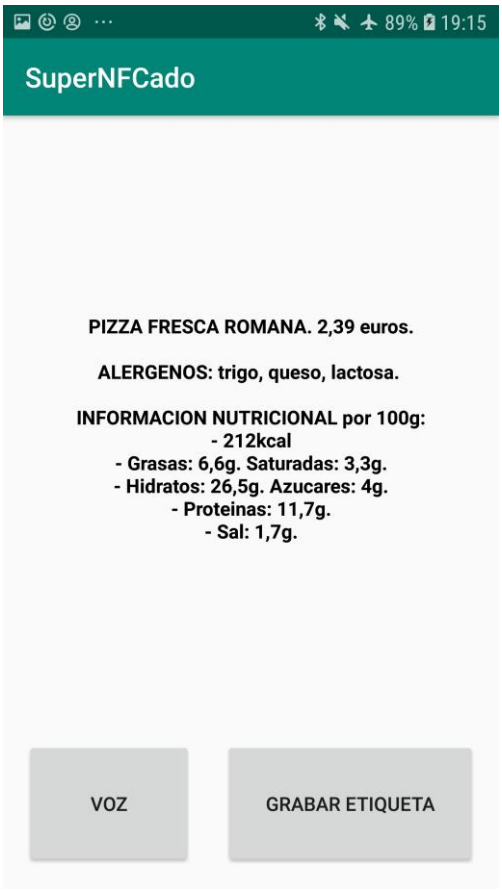


Figura 9: Lectura de etiqueta NFC

Tal y como ya se ha comentado anteriormente, la información extraída de la etiqueta NFC será ampliable para permitir su visualización a los usuarios. Para ello, deberemos implantar un *listener* que reaccione cuando la pantalla es “pellizcada” por el usuario:

```
private class ScaleListener extends
ScaleGestureDetector.SimpleOnScaleGestureListener {
    @Override
    public boolean onScale(ScaleGestureDetector
scaleGestureDetector){
        mScaleFactor *= scaleGestureDetector.getScaleFactor();
        mScaleFactor = Math.max(0.1f,
                Math.min(mScaleFactor, 10.0f));
        mEtMessage.setScaleX(mScaleFactor);
        mEtMessage.setScaleY(mScaleFactor);
        return true;
    }
}
```

A continuación, se muestran capturas de pantalla con diferente *zoom*:

Figura 10: Zoom incluyendo toda la información

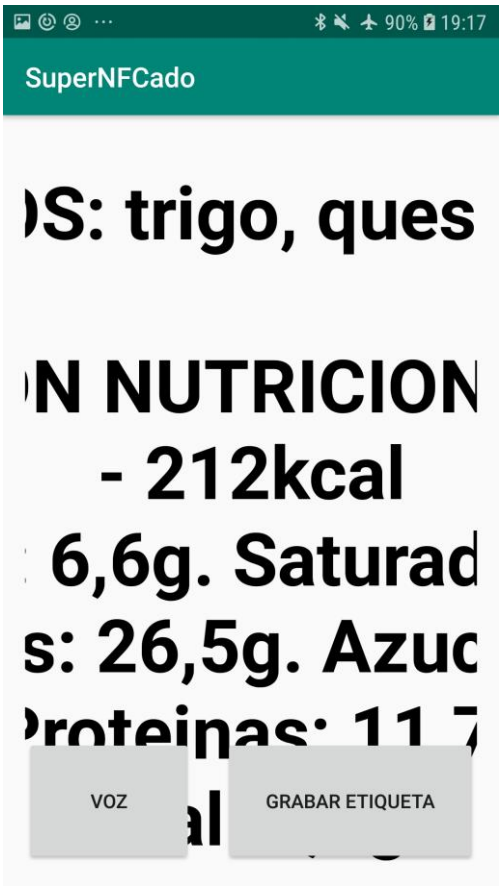


Figura 11: Zoom centrado en las kcal.

Por otro lado, cada uno de los botones presentes en esta primera pantalla implementa el método *setOnClickListener*, que nos permite definir qué tarea se lleva a cabo al pulsar cada uno de ellos. En el primero de los casos, el pulsar el botón "VOZ" llamará al método *onClick*, que se encargará de convertir el contenido de la etiqueta a un tipo *String*, y hará uso de la librería *Android.speech.tts.TextToSpeech* para lanzar el asistente de voz:

```
mBtVoice.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view){
        String text = mEtMessage.getText().toString();
        if(text.length() > 0){
            textToSpeech.speak(text, TextToSpeech.QUEUE_ADD,
null);
        }
    }
});
```

El segundo de los botones simplemente nos llevará a una nueva interfaz en la que tendremos que escribir el nombre de usuario y contraseña para acceder a la escritura de las etiquetas:

```
mBtWrite.setOnClickListener(view -> showWriteFragment());
```

Este grabado de etiquetas no debería ser accesible para cualquier usuario, sino que únicamente los empleados de los distintos supermercados son los que deberían tener acceso a esta característica. Por ello, antes de poder introducir el texto deseado y grabar una etiqueta NFC, se deberá introducir un usuario y contraseña. Concretamente, por sencillez se ha optado por un único usuario escrito en el propio código de la *app*: admin@admin.com, con credenciales “12345678”.

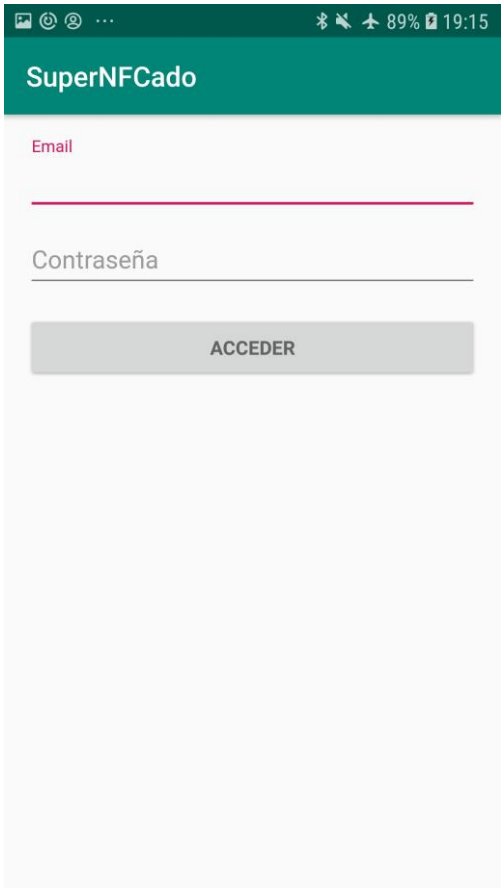


Figura 12: usuario y contraseña para acceder

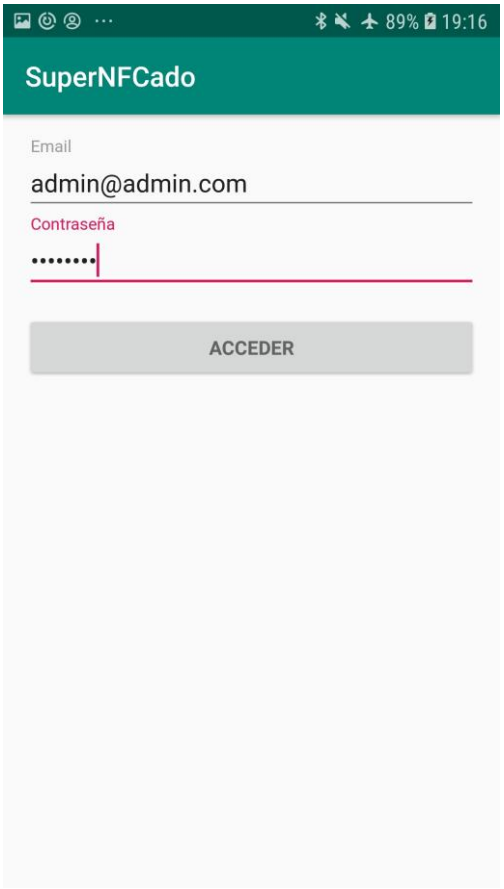


Figura 13: usuario y contraseña *hardcoded*

Android Studio cuenta con una plantilla que nos permite generar automáticamente una interfaz de este tipo, en la que hay que introducir un usuario y contraseña. Esta plantilla también incluye el método *attemptLogin*, el cual permite verificar si los datos que se están introduciendo son correctos o no. A continuación, se muestran parte de estas comprobaciones:

```
if (!TextUtils.isEmpty(password) && !isPasswordValid(password)) {

mPasswordView.setError(getString(R.string.error_invalid_password));
    focusView = mPasswordView;
    cancel = true;
}


// Check for a valid email address.
```

```
if (TextUtils.isEmpty(email)) {
    mEmailView.setError(getString(R.string.error_field_required));
    focusView = mEmailView;
    cancel = true;
} else if (!isEmailValid(email)) {
    mEmailView.setError(getString(R.string.error_invalid_email));
    focusView = mEmailView;
    cancel = true;
```

En caso de que las comprobaciones sean satisfactorias, pasaremos a la tercera y última interfaz de SuperNFCado (Figura 13). En ella, el usuario logueado podrá escribir la información de un determinado producto en el contenedor de tipo *EditText*, y grabarla en una etiqueta NFC al pulsar el correspondiente botón “GRABAR ETIQUETA”. Por comodidad, también se incluye un botón “MENU PRINCIPAL”, que permite volver a la primera de las interfaces anteriormente descritas. Al igual que en los botones anteriores, el método onClickListener nos permite actuar cuando uno de ellos sea pulsado:

```
mBtWrite.setOnClickListener(view -> showWriteFragment());
mBtBack.setOnClickListener(view -> mainMenu());
```

El método que nos permite escribir la información en la etiqueta se llama *writeToNFC*, y actúa cuando se detecta próxima una etiqueta NFC:

```
private void writeToNfc(Ndef ndef, String message){
    if (ndef != null) {

        try {
            ndef.connect();
            NdefRecord mimeRecord =
NdefRecord.createMime("text/plain",
message.getBytes(Charset.forName("US-ASCII")));
            ndef.writeNdefMessage(new NdefMessage(mimeRecord));
            ndef.close();
            //Write Successful
            Toast.makeText(this,
getString(R.string.message_write_success),
Toast.LENGTH_SHORT).show();

        } catch (IOException | FormatException e) {
            e.printStackTrace();
            Toast.makeText(this,
getString(R.string.message_write_error),
Toast.LENGTH_SHORT).show();

        } finally {
            return;
        }

    }
}
```

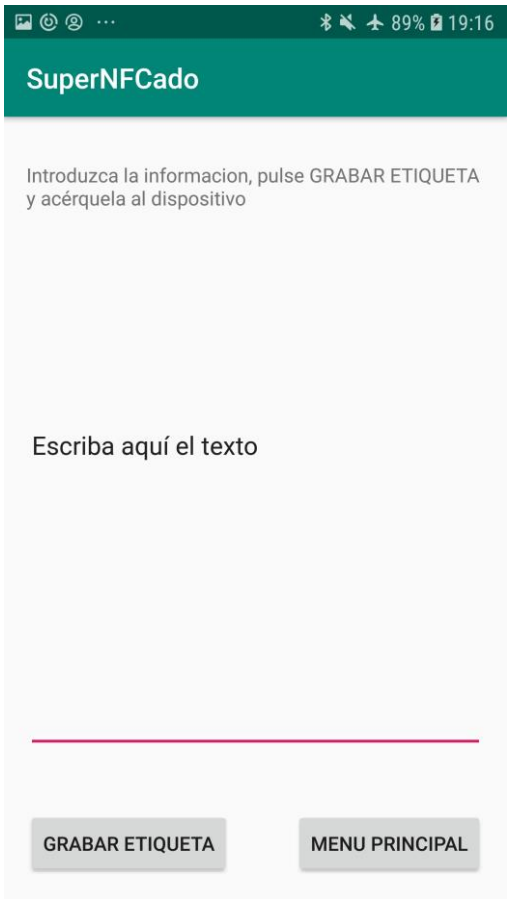


Figura 14: Interfaz para escribir etiqueta tras acceso

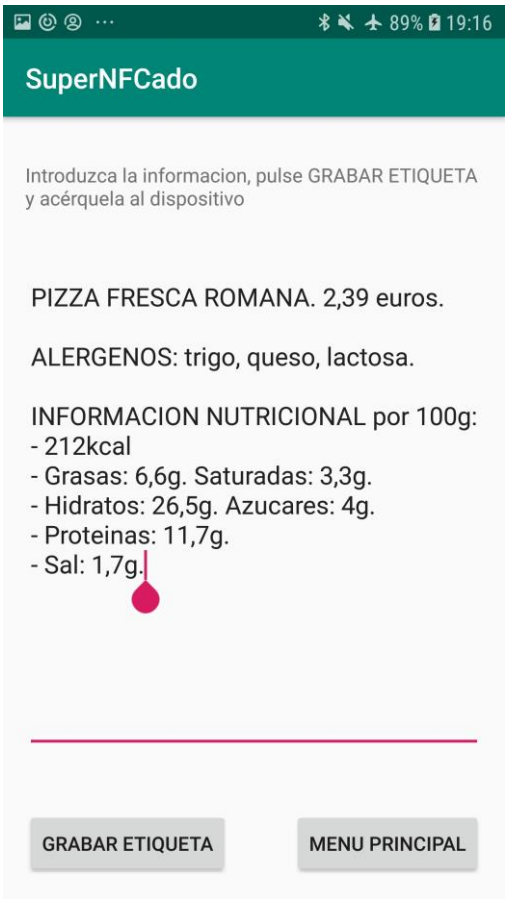


Figura 15: Interfaz para escribir etiqueta con texto a escribir

4. Conclusiones

A lo largo del desarrollo del trabajo, he tenido la oportunidad de inicarme en el uso del entorno de desarrollo Android Studio, el cual nunca había utilizado con anterioridad. Además, aunque en el pasado ya había trabajado con Java como lenguaje de programación, nunca lo había empleado para el desarrollo de aplicaciones Android. También ha servido para realizar un arduo trabajo de búsqueda de información.

En cuanto a los objetivos planteados, estoy bastante satisfecha con lo logrado en este Trabajo, si bien faltaría mejorar la parte encargada de poder aplicar *zoom* en el texto para que la aplicación fuera perfectamente funcional. Esto se ha debido, principalmente, a la imposibilidad de hallar documentación de apoyo que ayudara a la implementación del *zoom* en un TextView, ya que toda la información encontrada hacia referencia a ImageView; y a la falta de tiempo necesaria para estudiar la posibilidad de adaptar el código correspondiente de una ImageView a una TextView.

En cuanto a la metodología, personalmente hubiera preferido empezar con el desarrollo del proyecto en sí, y redactar la Memoria una vez terminado, o al menos definido prácticamente en su totalidad, el trabajo a realizar. El planteamiento utilizado, en mi caso en particular, ha provocado dificultades extra a la hora de realizar la Memoria ya que no contaba con una visión global del desarrollo de la aplicación.

Las líneas futuras de este trabajo serían:

- Mejorar el *zoom* del texto mostrado al leer la etiqueta NFC, permitiendo mayor movilidad a lo largo de la pantalla.
- Creación de una base de datos que contenga diferentes nombres de usuario y contraseñas de personal autorizado, en lugar de tener dicho credenciales directamente en el código de la aplicación.
- Creación de una base de datos que contenga la información de los diferentes productos y los asocie a un código, para así no tener que reescribir las etiquetas cada vez que hubiera un cambio, sino realizarlo una única vez. De este modo, cada etiqueta contendría el código correspondiente, y la aplicación conectaría con la base de datos para obtener la información asociada al producto.

5. Glosario

- ADT. *Android Development Tools*. Herramientas de Desarrollo de Android.
- App. Apócope de *application* (aplicación).
- CMOS. *Complementary metal-oxide-semiconductor*. Semiconductor complementario de óxido metálico.
- CSS. *Cascading Style Sheets*. Hojas de Estilo en Cascada.
- EEPROM. *Electrically Erasable Programmable Read-Only Memory*. ROM programable y borrable eléctricamente.
- HTML. *HyperText Markup Language*. Lenguaje de Marcas de Hipertexto.
- IDE. *Integrated DEvelopment Environment*. Entorno de Desarrollo Integrado.
- NFC. *Near Field Communication*. Comunicación de Campo Cercano.
- P2P*. Peer to Peer*. Red entre pares.
- PHP. *Hypertext Preprocessor*. Preprocesador de Hipertexto.
- QR. *Quick Response*. Respuesta Rápida.
- RFID. *Radio Frequency Identification*. Identificación por Radiofrecuencia.
- SDK. *Software Development Kit*. Kit de Desarrollo de Software.
- SO. Sistema Operativo.
- TFM. Trabajo Fin de Máster.
- WiFi. *Wireless Fidelity*. Fidelidad Inalámbrica.

6. Bibliografía

[1] CEBRIÁN DE MIGUEL (2003). *Glosario de discapacidad visual*. ANORMI, S.L. ISBN: 84-484-0090-9.

[2] RODRÍGUEZ FUENTES, A. (2005). *¿Cómo leen los niños con ceguera y baja visión?* Colección Escuela y necesidades educativas especiales. Aljebe. ISBN: 84-9700-231-8.

[3] RUPERT R. A. BOURNE, SETH R. FLAXMAN y otros (agosto, 2017). *Magnitude, temporal trends, and projections of the flobal prevalence of blindness and distance and near visión impairment: a systematic review and meta-analysis*. Recuperado el 30 de septiembre de 2018: https://www.thelancet.com/action/showPdf?pii=S2214-109X%2817%2930293-0

[4] LAGO ÁLVAREZ, M. (2013). *Rehabilitación Visual en degeneración macular asociada a la edad*. Trabajo Fin de Máster, Universidad de Valladolid. Recuperado el 30 de septiembre de 2018: https://uvadoc.uva.es/bitstream/10324/4409/6/TFM-%20M%2051.pdf

[5] CHAVEZ NUÑEZ, E.Y. (noviembre, 2012). *La mácula y su patología*. UNAH. Visitada el 2 de enero de 2019: https://es.slideshare.net/sweetevy/la-mcula-y-su-patoplogia

[6] ALBÀ ARBALAT, S. (enero, 2015). *Aspectos psicológicos y su relación con la calidad de vida en pacientes con baja visión*. Gaceta. Artículo Científico, nº 499.

[7] BCNBAIXAVISIÓ. *Productos de baja visión y ayudas visuales*. BCNBaixaVisión. Visitada el 2 de enero de 2019: https://www.bcnbaixavisio.com/productos-de-baja-vision/

[8] NOTICIAS VIGO (diciembre, 2017). *La baja visión y las ayudas visuales más efectivas*. Noticias Vigo. Visitada el 2 de enero de 2019: https://www.noticiasvigo.es/la-baja-vision-las-ayudas-visuales-mas-efectivas/

[9] RIGHTHEAR. Visitada el 1 de octubre de 2018; https://right-hear.com/

[10] DON, A. (septiembre, 2018). *Tech that guides the vusually impaired around supermarkets targets UK launch*. The Grocer. Visitada el 1 de octubre de 2018: https://www.thegrocer.co.uk/stores/service-and-availability/tech-that-guides-visually-impaired-around-supermarkets-targets-uk-launch/571341.article

[11] Visual Cortex On Silicon. *An NSF Expedition in Computing*. Visitada el 1 de octubre de 2018: http://www.cse.psu.edu/research/visualcortexonsilicon.expedition/

[12] WINNER-PENN STATE, C. (mayo, 2015). *Device helps visually impaired ´see´ groceries*. Visitada el 1 de octubre de 2018: https://www.futurity.org/visually-impaired-grocery-stores-technology-929382/

[13] KOELLE, A. (1975). *Short-Range Radio-Telemetry for Electronic Identification using modulated Backscatter*. Proceedings of the IEEE Volume: 63 Issue 8 (1975) ISSN: 0018-9219

[14] LANDT, J. (2005). *The history of RFID*. IEEE Potentials Potentials, IEEE. 24(4):8-11 Jan, 2005

[15] NFC.Today (2017). *The Difference Between NFC and RFID*. NFC.Today. Visitada el 2 de enero de 2019: https://nfc.today/advice/difference-nfc-rfid-explained

[16] KAW RAINA, V. (2014). Emerging *Technologies for User-Friendly Mobile Payment Applications*. Birla Institute of Technology, India. Visitada el 2 de enero de 2019: https://www.researchgate.net/profile/Vibha_Raina2/publication/260684432_Emerging_Technologies_for_User-Friendly_Mobile_Payment_Applications/links/0046353200a5851c08000000/Emerging-Technologies-for-User-Friendly-Mobile-Payment-Applications.pdf?origin=publication_detail

[17] SMITH, E. *NFC Near Field Communication. History*. NFCNearFieldCommunication.org. Visitada el 2 de enero de 2019: http://www.nfcnearfieldcommunication.org/history.html

[18] ONBARCODE (2018). *ENA-13 Introduction*. OnBacode.COM. Visitada el 2 de enero de 2019: http://www.onbarcode.com/ean_13/

[19] SOFTSETI (noviembre, 2014). *Código de barras*. Softseti Blog. Visitada el 2 de enero de 2019: https://www.softseti.net/blog/2014/11/codigo-de-barras/

[20] E.P. (2015). *Los supermercados elevan su surtido un 12% de media en el último año*. Expansión. Visitada el 2 de enero de 2019: http://www.expansion.com/empresas/distribucion/2015/11/30/565c388f268e3e127a8b4578.html

[21] DENSO WAVE INCORPORATED. *Information capacity and versions of the QR code*. Denso Wave Incorporated. Visitada el 2 de enero de 2019: https://www.qrcode.com/en/about/version.html

[22] ELBLOGDEORANGE (abril, 2013). *En el 2017 el 85% de los smartphones estarán equipados con tecnología NFC*. Blog oficial de Orange. Visitada el 2 de enero de 2019:

http://blog.orange.es/innovacion/en-el-2017-el-85-de-los-smartphones-estaran-equipados-con-tecnologia-nfc/

[23] SHOPNFC. *Cómo elegir la etiqueta NFC*. ShopNFC. Visitada el 2 de enero de 2019: https://www.shopnfc.com/es/content/11-como-elegir-la-etiqueta-nfc

[24] AMAZON. *Timeskey NFC Tags NTAG215 Sticker Chip NXP NTAG 215, 504 Bytes Memoria*. TimesKey. Visitada el 2 de enero de 2019: https://www.amazon.es/gp/product/B077SVZ9V1/

[25] NEWZOO. Top 50 *Countries/Markets by Smartphone Users and Penetration*. NewZoo. Visitada el 2 de enero de 2019: https://newzoo.com/insights/rankings/top-50-countries-by-smartphone-penetration-and-users/

[26] SANZ FERNÁNDEZ, J. (julio, 2018). *Móviles baratos con NFC para pagar sin sacar la tarjeta*. El País. Visitada el 2 de enero de 2019: https://cincodias.elpais.com/cincodias/2018/07/10/album/1531229175_838886.html

[27] PASCUAL, J.A. (julio, 2018). *Android vs iPhone: la guerra de los smartphones en cifras*. Computer Hoy. Visitada el 2 de enero de 2019: https://computerhoy.com/reportajes/industria/android-vs-iphone-guerra-smartphones-cifras-271447

[28] RAMÍREZ, I. (diciembre, 2018). *Samsung tiene su peor trimestre con Huawei cada vez más cerca, según Gartner*. Xataka Movil. Visitada el 2 de enero de 2019: https://www.xatakamovil.com/mercado/samsung-tiene-su-peor-trimestre-huawei-cada-vez-cerca-gartner

[29] GUPTA, A. ESCHERICH, M. (diciembre, 2018). *Market Share: PC, Ultramobile and Mobile Phone ASPs, 3Q18 Update*. Gartner. Visitada el 2 de enero de 2019: https://www.gartner.com/doc/3896667/market-share-pc-ultramobile-mobile

[30] JET BRAINS. *Toolbox Subscription*. Jet Brains. Visitada el 2 de enero de 2019: https://www.jetbrains.com/idea/buy/#edition=commercial

[31] SUKAJ, E. (noviembre, 2018). *What are the best IDEs for Android development in Java?* Slant. Visitada el 2 de enero de 2019: https://www.slant.co/topics/4649/~ides-for-android-development-in-java

7. Anexos

7.1. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:gravity="center"
    tools:context="com.example.app_v2.MainActivity">

    <ImageView
        android:id="@+id/imageview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:layout_below="@+id/message" />

    <TextView
        android:id="@+id/message"
        android:layout_width="348dp"
        android:layout_height="421dp"
        android:scrollbars = "vertical"
        android:textStyle="bold"
        android:textColor="#000000"
        android:gravity="center"/>

    <Button
        android:id="@+id/btn_write"
        android:layout_width="184dp"
        android:layout_height="43dp"
        android:layout_below="@+id/message"
        android:layout_alignEnd="@+id/message"
        android:layout_alignRight="@+id/message"
        android:layout_alignParentBottom="true"
        android:layout_marginTop="15dp"
        android:text="GRABAR ETIQUETA" />

    <Button
        android:id="@+id/btn_voice"
        android:layout_width="142dp"
        android:layout_height="43dp"
        android:layout_below="@+id/message"
        android:layout_alignStart="@+id/message"
        android:layout_alignLeft="@+id/message"
        android:layout_alignBottom="@+id/btn_write"
        android:layout_marginTop="15dp"
        android:layout_marginEnd="22dp"
        android:layout_marginRight="22dp"
        android:layout_toStartOf="@+id/btn_write"
```

```
        android:layout_toLeftOf="@+id/btn_write"
        android:text="VOZ" />

</RelativeLayout>
```

7.2. Login_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".LoginMenu">

    <!-- Login progress -->
    <ProgressBar
        android:id="@+id/login_progress"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:visibility="gone" />

    <ScrollView
        android:id="@+id/login_form"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:id="@+id/email_login_form"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <android.support.design.widget.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">

                <AutoCompleteTextView
                    android:id="@+id/email"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:hint="@string/prompt_email"
                    android:inputType="textEmailAddress"
                    android:maxLines="1"
                    android:singleLine="true" />

            </android.support.design.widget.TextInputLayout>

            <android.support.design.widget.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
```

```
                <EditText
                    android:id="@+id/password"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:hint="@string/prompt_password"
                    android:imeActionId="6"

android:imeActionLabel="@string/action_sign_in_short"
                    android:imeOptions="actionUnspecified"
                    android:inputType="textPassword"
                    android:maxLines="1"
                    android:singleLine="true" />

            </android.support.design.widget.TextInputLayout>

            <Button
                android:id="@+id/email_sign_in_button"
                style="?android:textAppearanceSmall"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="16dp"
                android:text="ACCEDER"
                android:textStyle="bold" />

        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

7.3. Write_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentTop="true"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:gravity="center"
    tools:context=".NFCWriteMenu">

    <Button
        android:id="@+id/btn_write"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="GRABAR ETIQUETA" />

    <EditText
        android:id="@+id/editText"
        android:layout_width="367dp"
        android:layout_height="390dp"
        android:layout_above="@+id/btn_write"
        android:layout_alignParentStart="true"
```

```
        android:layout_alignParentTop="true"
        android:layout_marginBottom="41dp"
        android:ems="10"
        android:inputType="textMultiLine"
        android:text="Escriba aquí el texto"
        android:layout_alignParentLeft="true" />

    <TextView
        android:id="@+id/tv_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"

android:layout_marginTop="@dimen/activity_horizontal_margin"
        android:text="@string/message_tap_tag"/>

    <Button
        android:id="@+id/btn_back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/btn_write"
        android:layout_alignEnd="@+id/editText"
        android:layout_alignParentBottom="true"
        android:text="MENU PRINCIPAL" />

</RelativeLayout>
```

7.4. MainActivity.java

```
package com.example.app_v2;

import android.app.PendingIntent;
import android.content.Intent;
import android.content.IntentFilter;
import android.nfc.FormatException;
import android.nfc.NdefMessage;
import android.nfc.NfcAdapter;
import android.nfc.Tag;
import android.nfc.tech.Ndef;
import android.speech.tts.TextToSpeech;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.MotionEvent;
import android.view.ScaleGestureDetector;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.io.IOException;

public class MainActivity extends AppCompatActivity implements
com.example.myapplication.Listener, TextToSpeech.OnInitListener{

    public static final String TAG =
MainActivity.class.getSimpleName();
    private TextView mEtMessage;
    private Button mBtWrite;
```

```
    private Button mBtVoice;
    TextToSpeech textToSpeech;
    int MY_DATA_CHECK_CODE = 1000;

    private ScaleGestureDetector mScaleGestureDetector;
    private float mScaleFactor = 1.0f;
    private final static float mMinZoom = 0.5f;
    private final static float mMaxZoom = 5.0f;

    private boolean isDialogDisplayed = false;
    private boolean isWrite = false;

    private NfcAdapter mNfcAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mEtMessage = (TextView) findViewById(R.id.message);
        mBtWrite = (Button) findViewById(R.id.btn_write);
        mBtVoice = (Button) findViewById(R.id.btn_voice);
        textToSpeech = new TextToSpeech(this, this);

        mBtWrite.setOnClickListener(view ->
showWriteFragment());

        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);

        mBtVoice.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View view){
                String text = mEtMessage.getText().toString();
                if(text.length() > 0){
                    textToSpeech.speak(text,
TextToSpeech.QUEUE_ADD, null);
                }
            }
        });
        Intent intent = new Intent();

intent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
        startActivityForResult(intent, MY_DATA_CHECK_CODE);

        mScaleGestureDetector = new ScaleGestureDetector(this,
new ScaleListener());
    }

@Override
    public boolean onTouchEvent(MotionEvent motionEvent) {
        mScaleGestureDetector.onTouchEvent(motionEvent);
        return true;
    }

    @Override
    protected void onActivityResult(int requestCode, int
resultCode, Intent data){
        if(resultCode == MY_DATA_CHECK_CODE){
            textToSpeech = new TextToSpeech(this,this);
        } else{
            Intent intent = new Intent();
```

```
intent.setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
        }
    }

    private void showWriteFragment() {
        isWrite = true;
        Intent mIntent = new Intent(MainActivity.this,
LoginMenu.class);
        startActivity(mIntent);
    }

    @Override
    public void onDialogDisplayed() {

        isDialogDisplayed = true;
    }

    @Override
    public void onDialogDismissed() {

        isDialogDisplayed = false;
        isWrite = false;
    }
    @Override
    protected void onResume() {
        super.onResume();
        IntentFilter tagDetected = new
IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED);
        IntentFilter ndefDetected = new
IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
        IntentFilter techDetected = new
IntentFilter(NfcAdapter.ACTION_TECH_DISCOVERED);
        IntentFilter[] nfcIntentFilter = new
IntentFilter[]{techDetected,tagDetected,ndefDetected};

        PendingIntent pendingIntent = PendingIntent.getActivity(
                this, 0, new Intent(this,
getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
        if(mNfcAdapter!= null)
            mNfcAdapter.enableForegroundDispatch(this,
pendingIntent, nfcIntentFilter, null);

    }
    @Override
    protected void onPause() {
        super.onPause();
        if(mNfcAdapter!= null)
            mNfcAdapter.disableForegroundDispatch(this);
    }
    @Override
    protected void onNewIntent(Intent intent) {
        Tag tag =
intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

        Log.d(TAG, "onNewIntent: "+intent.getAction());

        if(tag != null) {
            Toast.makeText(this,
getString(R.string.message_tag_detected),
Toast.LENGTH_SHORT).show();
            Ndef ndef = Ndef.get(tag);
```

```
            onNfcDetected(ndef);
        }
    }
    private void onNfcDetected(Ndef ndef) {
        try {
            ndef.connect();
            NdefMessage ndefMessage = ndef.getNdefMessage();
            String message = new
String(ndefMessage.getRecords()[0].getPayload());
            Log.d(TAG, "readFromNFC: "+message);
            mEtMessage.setText(message);
            ndef.close();

        } catch (IOException | FormatException e) {
            e.printStackTrace();
        }
    }

    private class ScaleListener extends
ScaleGestureDetector.SimpleOnScaleGestureListener {
        @Override
        public boolean onScale(ScaleGestureDetector
scaleGestureDetector){
            mScaleFactor *=
scaleGestureDetector.getScaleFactor();
            mScaleFactor = Math.max(0.1f,
                    Math.min(mScaleFactor, 10.0f));
            mEtMessage.setScaleX(mScaleFactor);
            mEtMessage.setScaleY(mScaleFactor);
            return true;
        }
    }

    @Override
    public void onInit(int i) {
        if(i == TextToSpeech.SUCCESS){
            Toast.makeText(this, "Success!",
Toast.LENGTH_SHORT).show();
        } else if(i == TextToSpeech.ERROR){
            Toast.makeText(this, "Error!",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

7.5. LoginMenu.java

```
package com.example.app_v2;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.annotation.TargetApi;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.app.LoaderManager.LoaderCallbacks;

import android.content.CursorLoader;
```

```
import android.content.Loader;
import android.database.Cursor;
import android.net.Uri;
import android.os.AsyncTask;

import android.os.Build;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.text.TextUtils;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

import static android.Manifest.permission.READ_CONTACTS;

/**
 * A login screen that offers login via email/password.
 */
public class LoginMenu extends AppCompatActivity implements
LoaderCallbacks<Cursor> {

    /**
     * Id to identity READ_CONTACTS permission request.
     */
    private static final int REQUEST_READ_CONTACTS = 0;

    /**
     * A dummy authentication store containing known user names
and passwords.
     * TODO: remove after connecting to a real authentication
system.
     */
    private static final String[] DUMMY_CREDENTIALS = new
String[]{
            "admin@admin.com:12345678"
    };
    /**
     * Keep track of the login task to ensure we can cancel it
if requested.
     */
    private UserLoginTask mAuthTask = null;

    // UI references.
    private AutoCompleteTextView mEmailView;
    private EditText mPasswordView;
    private View mProgressView;
    private View mLoginFormView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login_menu);
```

```
        // Set up the login form.
        mEmailView = (AutoCompleteTextView)
findViewById(R.id.email);
        populateAutoComplete();

        mPasswordView = (EditText) findViewById(R.id.password);
        mPasswordView.setOnEditorActionListener(new
TextView.OnEditorActionListener() {
            @Override
            public boolean onEditorAction(TextView textView, int
id, KeyEvent keyEvent) {
                if (id == EditorInfo.IME_ACTION_DONE || id ==
EditorInfo.IME_NULL) {
                    attemptLogin();
                    return true;
                }
                return false;
            }
        });

        Button mEmailSignInButton = (Button)
findViewById(R.id.email_sign_in_button);
        mEmailSignInButton.setOnClickListener(new
OnClickListener() {
            @Override
            public void onClick(View view) {
                attemptLogin();
            }
        });

        mLoginFormView = findViewById(R.id.login_form);
        mProgressView = findViewById(R.id.login_progress);
    }

    private void populateAutoComplete() {
        if (!mayRequestContacts()) {
            return;
        }

        getLoaderManager().initLoader(0, null, this);
    }

    private boolean mayRequestContacts() {
        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M) {
            return true;
        }
        if (checkSelfPermission(READ_CONTACTS) ==
PackageManager.PERMISSION_GRANTED) {
            return true;
        }
        if (shouldShowRequestPermissionRationale(READ_CONTACTS))
{
          /*  Snackbar.make(mEmailView,
R.string.permission_rationale, Snackbar.LENGTH_INDEFINITE)
                    .setAction(android.R.string.ok, new
View.OnClickListener() {
                        @Override
                        @TargetApi(Build.VERSION_CODES.M)
                        public void onClick(View v) {
                            requestPermissions(new
String[]{READ_CONTACTS}, REQUEST_READ_CONTACTS);
```

```
                    }
                });*/
        } else {
            requestPermissions(new String[]{READ_CONTACTS},
REQUEST_READ_CONTACTS);
        }
        return false;
    }

    /**
     * Callback received when a permissions request has been
completed.
     */
    @Override
    public void onRequestPermissionsResult(int requestCode,
@NonNull String[] permissions,
                                           @NonNull int[]
grantResults) {
        if (requestCode == REQUEST_READ_CONTACTS) {
            if (grantResults.length == 1 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                populateAutoComplete();
            }
        }
    }


    /**
     * Attempts to sign in or register the account specified by
the login form.
     * If there are form errors (invalid email, missing fields,
etc.), the
     * errors are presented and no actual login attempt is made.
     */
    private void attemptLogin() {
        if (mAuthTask != null) {
            return;
        }

        // Reset errors.
        mEmailView.setError(null);
        mPasswordView.setError(null);

        // Store values at the time of the login attempt.
        String email = mEmailView.getText().toString();
        String password = mPasswordView.getText().toString();

        boolean cancel = false;
        View focusView = null;

        // Check for a valid password, if the user entered one.
        if (!TextUtils.isEmpty(password) &&
!isPasswordValid(password)) {

mPasswordView.setError(getString(R.string.error_invalid_password
));
            focusView = mPasswordView;
            cancel = true;
        }

        // Check for a valid email address.
```

```
        if (TextUtils.isEmpty(email)) {
mEmailView.setError(getString(R.string.error_field_required));
            focusView = mEmailView;
            cancel = true;
        } else if (!isEmailValid(email)) {
mEmailView.setError(getString(R.string.error_invalid_email));
            focusView = mEmailView;
            cancel = true;
        }

        if (cancel) {
            // There was an error; don't attempt login and focus
the first
            // form field with an error.
            focusView.requestFocus();
        } else {
            // Show a progress spinner, and kick off a
background task to
            // perform the user login attempt.
            showProgress(true);
            mAuthTask = new UserLoginTask(email, password);
            mAuthTask.execute((Void) null);
        }
    }

    private boolean isEmailValid(String email) {
        //TODO: Replace this with your own logic
        return email.contains("@");
    }

    private boolean isPasswordValid(String password) {
        //TODO: Replace this with your own logic
        return password.length() > 4;
    }

    /**
     * Shows the progress UI and hides the login form.
     */
    @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
    private void showProgress(final boolean show) {
        // On Honeycomb MR2 we have the ViewPropertyAnimator
APIs, which allow
        // for very easy animations. If available, use these
APIs to fade-in
        // the progress spinner.
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.HONEYCOMB_MR2) {
            int shortAnimTime =
getResources().getInteger(android.R.integer.config_shortAnimTime
);

            mLoginFormView.setVisibility(show ? View.GONE :
View.VISIBLE);

mLoginFormView.animate().setDuration(shortAnimTime).alpha(
                    show ? 0 : 1).setListener(new
AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
```

```
                    mLoginFormView.setVisibility(show ?
View.GONE : View.VISIBLE);
                }
            });

            mProgressView.setVisibility(show ? View.VISIBLE :
View.GONE);

mProgressView.animate().setDuration(shortAnimTime).alpha(
                    show ? 1 : 0).setListener(new
AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    mProgressView.setVisibility(show ?
View.VISIBLE : View.GONE);
                }
            });
        } else {
            // The ViewPropertyAnimator APIs are not available,
so simply show
            // and hide the relevant UI components.
            mProgressView.setVisibility(show ? View.VISIBLE :
View.GONE);
            mLoginFormView.setVisibility(show ? View.GONE :
View.VISIBLE);
        }
    }

    @Override
    public Loader<Cursor> onCreateLoader(int i, Bundle bundle) {
        return new CursorLoader(this,
                // Retrieve data rows for the device user's
'profile' contact.

Uri.withAppendedPath(ContactsContract.Profile.CONTENT_URI,

ContactsContract.Contacts.Data.CONTENT_DIRECTORY),
ProfileQuery.PROJECTION,

                // Select only email addresses.
                ContactsContract.Contacts.Data.MIMETYPE +
                        " = ?", new
String[]{ContactsContract.CommonDataKinds.Email
                .CONTENT_ITEM_TYPE},

                // Show primary email addresses first. Note that
there won't be
                // a primary email address if the user hasn't
specified one.
                ContactsContract.Contacts.Data.IS_PRIMARY + "
DESC");
    }

    @Override
    public void onLoadFinished(Loader<Cursor> cursorLoader,
Cursor cursor) {
        List<String> emails = new ArrayList<>();
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            emails.add(cursor.getString(ProfileQuery.ADDRESS));
            cursor.moveToNext();
```

```
        }

        addEmailsToAutoComplete(emails);
    }

    @Override
    public void onLoaderReset(Loader<Cursor> cursorLoader) {

    }

    private void addEmailsToAutoComplete(List<String>
emailAddressCollection) {
        //Create adapter to tell the AutoCompleteTextView what
to show in its dropdown list.
        ArrayAdapter<String> adapter =
                new ArrayAdapter<>(LoginMenu.this,

android.R.layout.simple_dropdown_item_1line,
emailAddressCollection);

        mEmailView.setAdapter(adapter);
    }


    private interface ProfileQuery {
        String[] PROJECTION = {
                ContactsContract.CommonDataKinds.Email.ADDRESS,

ContactsContract.CommonDataKinds.Email.IS_PRIMARY,
        };

        int ADDRESS = 0;
        int IS_PRIMARY = 1;
    }

    /**
     * Represents an asynchronous login/registration task used
to authenticate
     * the user.
     */
    public class UserLoginTask extends AsyncTask<Void, Void,
Boolean> {

        private final String mEmail;
        private final String mPassword;

        UserLoginTask(String email, String password) {
            mEmail = email;
            mPassword = password;
        }

        @Override
        protected Boolean doInBackground(Void... params) {
            // TODO: attempt authentication against a network
service.

            try {
                // Simulate network access.
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                return false;
```

```
            }

            for (String credential : DUMMY_CREDENTIALS) {
                String[] pieces = credential.split(":");
                if (pieces[0].equals(mEmail)) {
                    // Account exists, return true if the
password matches.
                    return pieces[1].equals(mPassword);
                }
            }

            // TODO: register the new account here.
            return true;
        }

        @Override
        protected void onPostExecute(final Boolean success) {
            mAuthTask = null;
            showProgress(false);

            if (success) {
                //finish();
                Intent mIntent = new Intent(LoginMenu.this,
NFCWriteMenu.class);
                startActivity(mIntent);
            } else {

mPasswordView.setError(getString(R.string.error_incorrect_passwo
rd));
                mPasswordView.requestFocus();
            }
        }

        @Override
        protected void onCancelled() {
            mAuthTask = null;
            showProgress(false);
        }
    }
}
```

7.6. NFCWriteMenu.java

```
package com.example.app_v2;

import android.app.PendingIntent;
import android.content.Intent;
import android.content.IntentFilter;
import android.nfc.FormatException;
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.nfc.Tag;
import android.nfc.tech.Ndef;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.widget.Button;
import android.widget.TextView;
```

```
import android.widget.Toast;

import java.io.IOException;
import java.nio.charset.Charset;

public class NFCWriteMenu extends AppCompatActivity implements
com.example.myapplication.Listener {

    public static final String TAG =
MainActivity.class.getSimpleName();

    private TextView mEtMessage;
    private Button mBtWrite;
    private Button mBtBack;

    private boolean isDialogDisplayed = false;
    private boolean isWrite = false;

    private NfcAdapter mNfcAdapter;

    private TextView mTvMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.write_menu);

        mEtMessage = (TextView) findViewById(R.id.editText);
        mBtWrite = (Button) findViewById(R.id.btn_write);
        mBtBack = (Button) findViewById(R.id.btn_back);

        mBtWrite.setOnClickListener(view ->
showWriteFragment());
        mBtBack.setOnClickListener(view -> mainMenu());

        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);

        mTvMessage = (TextView) findViewById(R.id.tv_message);
    }

    private void mainMenu() {
        Intent mIntent = new Intent(NFCWriteMenu.this,
MainActivity.class);
        startActivity(mIntent);
    }

    private void showWriteFragment() {

        isWrite = true;
        mTvMessage.setText(getString(R.string.message_tap_tag));
    }

    @Override
    public void onDialogDisplayed() {

        isDialogDisplayed = true;
    }

    @Override
    public void onDialogDismissed() {
```

```
        isDialogDisplayed = false;
        isWrite = false;
    }
    @Override
    protected void onResume() {
        super.onResume();
        IntentFilter tagDetected = new
IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED);
        IntentFilter ndefDetected = new
IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
        IntentFilter techDetected = new
IntentFilter(NfcAdapter.ACTION_TECH_DISCOVERED);
        IntentFilter[] nfcIntentFilter = new
IntentFilter[]{techDetected,tagDetected,ndefDetected};

        PendingIntent pendingIntent = PendingIntent.getActivity(
                this, 0, new Intent(this,
getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
        if(mNfcAdapter!= null)
            mNfcAdapter.enableForegroundDispatch(this,
pendingIntent, nfcIntentFilter, null);

    }
    @Override
    protected void onPause() {
        super.onPause();
        if(mNfcAdapter!= null)
            mNfcAdapter.disableForegroundDispatch(this);
    }
    @Override
    protected void onNewIntent(Intent intent) {
        Tag tag =
intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

        Log.d(TAG, "onNewIntent: "+intent.getAction());

        if(tag != null) {
            Toast.makeText(this,
getString(R.string.message_tag_detected),
Toast.LENGTH_SHORT).show();
            Ndef ndef = Ndef.get(tag);

            if (isWrite) {
                String messageToWrite =
mEtMessage.getText().toString();
                onNfcDetected(ndef,messageToWrite);

            }
        }
    }

    public void onNfcDetected(Ndef ndef, String messageToWrite){

        writeToNfc(ndef,messageToWrite);
    }

    private void writeToNfc(Ndef ndef, String message){
        if (ndef != null) {

            try {
                ndef.connect();
```

```
                NdefRecord mimeRecord =
NdefRecord.createMime("text/plain",
message.getBytes(Charset.forName("US-ASCII")));
                ndef.writeNdefMessage(new
NdefMessage(mimeRecord));
                ndef.close();
                //Write Successful
                Toast.makeText(this,
getString(R.string.message_write_success),
Toast.LENGTH_SHORT).show();

            } catch (IOException | FormatException e) {
                e.printStackTrace();
                Toast.makeText(this,
getString(R.string.message_write_error),
Toast.LENGTH_SHORT).show();

            } finally {
                return;
            }

        }
    }

}
```