ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

App de apoyo a la recepción de trabajos de reprografía
(Supporting app for reprographics task admission)

Para acceder al Título de

INGENIERO TÉCNICO DE TELECOMUNICACIÓN

Autor: Roberto Díaz Fernández
Septiembre-2016

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Roberto Díaz Fernández
Director del PFC: José Luis Crespo Fidalgo
Título: "App de apoyo a la recepción de trabajos de reprografía"
Title: "Supporting app for reprographics task admission**"**

Presentado a examen el día: 27/09/2016

Para acceder al Título de

INGENIERO TECNICO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Pedro Corcuera Miró Quesada
Secretario (Apellidos, Nombre): José Luis Crespo Fidalgo
Vocal (Apellidos, Nombre): Pilar Bernardos Llorente

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(Sólo si es distinto del Secretario)

Vº Bº del Subdirector

Proyecto Fin de Carrera Nº
(a asignar por Secretaría)

Agradecimientos

A Nico por no tener en cuenta el tiempo que no te he podido dedicar mientras realizaba este proyecto, esperando que hayas podido aprender que con tesón y ganas todo lo que te propones lo puedes conseguir.

A mis padres y mi mujer en primer lugar, por la paciencia que han tenido en el transcurso del tiempo hasta terminar este proyecto y poder dar por terminada la carrera. Ha sido un placer poder contar con vuestro apoyo en cada una de mis decisiones aunque fuesen descabelladas siempre he tenido un comentario positivo para continuar hacia delante.

A APTACAN, por dejarme poner un granito de arena en su comunidad pudiendo ayudar a que sus miembros se puedan adaptar a la comunidad lo mejor posible y gracias por darme la posibilidad de ver lo fácil que es ayudar a los demás.

A mi tutor José Luis, por su ayuda, dirección y sobre todo por sus valiosos consejos y enseñanzas en este trabajo.

A todos, ¡Muchas Gracias!

Contenido

Resumen

En este documento del proyecto fin de carrera tratamos el desarrollo de una aplicación para dispositivos móviles Android para poner a disposición de los alumnos y tutores de APTACAN una herramienta desarrollada para facilitar las tareas en la reprografía del centro.

Estas tareas se pueden desglosar en tres grandes grupos: administración y creación de cualquier trabajo a realizar en la reprografía del centro, consulta y gestión de la caja de cobros utilizada en la reprografía del centro, y por último la parte a través de la cual se interactúa con el cliente y se encuentra el trabajo a realizar.

Teniendo en cuenta que la aplicación va a ser usada en un ámbito real, este proyecto abarca las prácticas habituales de una aplicación de este carácter, desde la fase de análisis, diseño e implementación dentro del ciclo de vida de software y teniendo en cuenta tras estudiar ciertos elementos o cambios en los requisitos del mismo.

Palabras Clave

- Trastorno de espectro autista.
- Autista.
- Reprografía.
- Recepción
- Cobro
- App.
- Android.
- SQLite.
- Programación.

Capítulo 1 Introducción

1.1. Contexto

Se va a implementar una aplicación para uso de autistas que se realiza dentro del marco del autismo, donde estas personas padecen un trastorno psicológico que se caracteriza por la intensa concentración de una persona en su propio mundo interior y la progresiva pérdida de contacto con el mundo exterior.

El término Trastorno del Espectro Autista (TEA) hace referencia a un conjunto amplio de condiciones que afectan al neurodesarrollo y al funcionamiento cerebral. [1]

Las personas con autismo presentan en mayor o menor medida dificultades en la comunicación e interacción social, y patrones repetitivos y restringidos de conductas, actividades e intereses.

Presentando las siguientes dificultades:

- COMUNICACIÓN:
    - Ausencia parcial o total del lenguaje y/o comunicación no verbal funcional.
    - Lenguaje y/o temas repetitivos de conversación.
    - Dificultades en el contacto ocular.
    - Dificultades en el control y comprensión de las distintas posturas corporales y gestos faciales.
    - Alteración del tono, ritmo y entonación del discurso.
- EN EL ENTORNO LABORAL PUEDE MOSTRAR DIFICULTAD:
    - En el inicio y mantenimiento de conversaciones.
    - En la comprensión de instrucciones.
    - Para solicitar ayuda.
    - Saludar.
    - Relacionarse con los compañeros durante los periodos de trabajo y de descanso.
    - Comprender normas socio laborales y de comportamiento.

    - Solucionar imprevistos.
    - Adaptarse a cambios de rutina sin previo aviso.
- INTERACCIÓN SOCIAL:
    - Aislamiento y/o pasividad social, mostrando más interés por determinados gustos, preocupaciones u objetos que por las personas de su entorno.
    - Interés por la interacción con los demás pero presenta una interacción desadaptada
    - Dificultades en la capacidad empática e incomprensión de ciertas reglas sociales y de comportamiento.

SIN EMBARGO, existen apoyos, recursos y programas específicos que permiten compensar dichas dificultades y permitir que las personas con TEA puedan desempeñar un trabajo de calidad, eficiente y competitivo.

1.2. Objetivos

Por las características descritas anteriormente se ve que los autistas van a tener problemas para desarrollar trabajos de reprografía, donde tienen que atender al cliente comunicándose con él. Una posibilidad de superar estas dificultades es utilizar aplicaciones de apoyo.

El objetivo de este proyecto es desarrollar una aplicación para una Tablet con el sistema operativo Android que permita realizar las tareas que se desarrollan en el puesto de reprografía. Tanto en el centro de educación como en los entornos laborales donde realizan sus prácticas.

Los servicios que se ofrecen en la aplicación para el servicio de tareas de APTACAN se engloban en cuatro grandes grupos:

- Creación y configuración de las plantillas de los trabajos que se pueden ofrecer en cualquier centro.
- Manejo de la caja del servicio donde se esté desarrollando la actividad.
- Presentación de los trabajos que se encuentran ya configuradas en la aplicación y mostrar dichas plantillas.
- Manejo de las imágenes que se van a poder usar en la aplicación para los distintos trabajos.

En conclusión el objetivo principal del proyecto es facilitar la interacción social por medio de esta herramienta.

1.3. Motivación

La principal motivación que me ha llevado a elegir este proyecto antes que otras opciones que tenía disponibles. Fue poder colaborar en una causa social a la vez que realizaba mi proyecto fin de carrera, posibilitando que estas personas se puedan integrar en un trabajo. Todo esto mediante un app que solvente el déficit de comunicación por medio de un dispositivo móvil, en este caso una Tablet.

A parte de esto influyen las siguientes motivaciones para realizar este proyecto:

- Ganar soltura en la realización de aplicaciones Android partiendo de los requisitos del cliente o colaborador.

- Desarrollar una aplicación que vaya a tener un uso final real, lo que la lleva a que tenga que ser estable. Incluyendo también un código que sea mantenible de forma sencilla y se pueda adaptar a cambios en el transcurso del tiempo.

- Desarrollar la aplicación utilizando las herramientas que ofrecen las nuevas versiones de Android para el desarrollo dinámico en una misma aplicación para Tablet.

- Desarrollar una interfaz fluida, que mantenga informado al usuario en todo momento de las operaciones que se están realizando.

- Establecer los pasos necesarios para desarrollar aplicaciones abarcando desde la creación del proyecto hasta la publicación del mismo.

- Conocer los detalles de la arquitectura de Android, así como obtener los conocimientos necesarios para desarrollar aplicaciones para este sistema operativo.

- Integrarse con las base de datos para dispositivos móviles, pudiendo aprender a realizar las principales funciones que lleva sqlite3 y conocer su estructura y forma de trabajar.

- Aprender a utilizar las librerías de compatibilidad para poder implementar las opciones que permiten las nuevas versiones de Android en dispositivos móviles con versiones más antiguas.

1.4. Contenido

La presente memoria se estructura en una serie de seis capítulos distribuidos de la siguiente manera:

- En el capítulo 1 se ha realizado una introducción del proyecto, hablando sobre el contexto, los objetivos y las motivaciones que se quieren lograr durante el desarrollo del mismo.

- En el capítulo 2 se hace un repaso de todas las tecnologías utilizadas durante el desarrollo de este proyecto.

- En el capítulo 3 se describen los requisitos que tenemos a la hora de programar la app.

- En el capítulo 4 se abordan los distintos casos de usos que tiene la aplicación.

- En el capítulo 5 mostramos las distintas ideas de diseño que se han utilizado a lo largo de todas las fases del desarrollo.

- En el capítulo 6 las grandes cuestiones sobre la arquitectura que contiene la app.

- En el capítulo 7 hablamos sobre las herramientas que han sido utilizadas en el desarrollo del proyecto.

- En el capítulo 8 se abordan las distintas pruebas que se tienen en cuenta en la app.

- En el capítulo 9 hablamos sobre cómo debe usarse la app acompañada de unas pinceladas sobre las cosas más relevantes de los manuales.

- En el capítulo 10 se muestran las conclusiones tras realizar el proyecto, y las líneas futuras previsibles para mejorar la aplicación APTACAN.

Además se incluye un glosario de términos con el significado de todas las siglas utilizadas en la presente memoria, así como la bibliografía utilizada durante su elaboración.

Capítulo 2 Estado del Arte

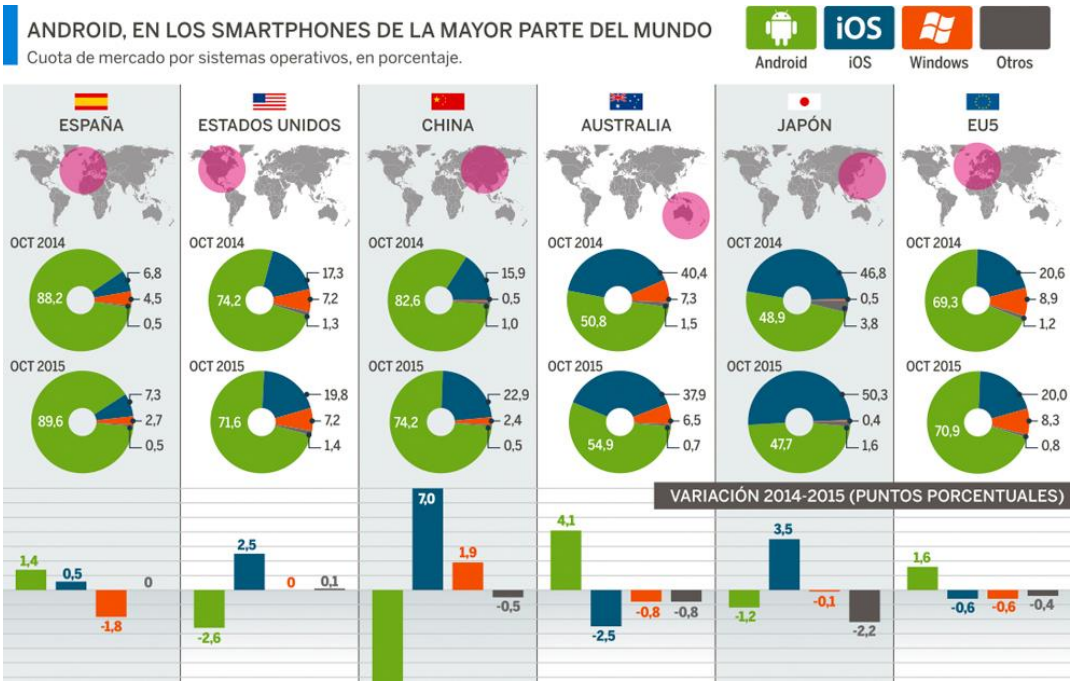
2.1. Introducción

Los teléfonos inteligentes, o más conocidos como Smartphones, están muy extendidos en todo el mundo y han experimentado un gran crecimiento. El Smartphone es el teléfono móvil capaz de gestionar el correo, manejar contenido multimedia, conectarse a internet, etc., en conclusión, almacenar información y manejarla como un pequeño ordenador de bolsillo.

Estos dispositivos inteligentes han ido aumentando en calidad y potencia de hardware llegando a superar, e incluso empezando a sustituir en el caso de las Tabletas a los PC en algunos hogares en los que sólo utilizaban éste para navegar por Internet, entrar en las redes sociales y poco más.

La gran aceptación de estos dispositivos por la mayor parte de la gente es debida a su gran variedad, en el precio, la variedad de dimensiones que nos ofrecen y los distintos sistemas operativos, la gran variedad de configuraciones y personalizaciones. Esto hace que cualquiera pueda encontrar un dispositivo adecuado para sus necesidades.

En cuanto a sistemas operativos móviles hay diversas opciones, aunque destacan como más utilizados sobre todo Android y iOS. Otros como BlackBerry o Windows Phone siguen manteniéndose. Sin embargo, el más consolidado a nivel mundial es Android, además de ser el que mayor crecimiento está teniendo como puede verse en la siguiente imagen. [2]



Queda clara la gran importancia que está cobrando la movilidad hoy en día, y parece que seguirá aumentando a lo largo de los años, de ahí la necesidad de dar a los usuarios una forma de ver la información que desean de forma adecuada para el dispositivo que estén utilizando.

Dado que el sistema operativo de Tabletas con las que cuentan en APTACAN es Android, se decidió crear esta aplicación de gestión de trabajos para dicho sistema operativo. Con esta aplicación, los miembros de APTACAN podrán entrenar a los alumnos de una forma cómoda, rápida y en cualquier lugar.

2.2. Android

2.2.1. Introducción

Android es un sistema operativo creado para ser independiente de cualquier tipo de arquitectura de hardware en los dispositivos móviles. Esta característica hace que sea tan atractivo ante los fabricantes y desarrolladores.

Adicionalmente su portabilidad, flexibilidad y seguridad les da ese toque de simpatía a las personas interesadas en los sistemas de código abierto.

La arquitectura de Android debe ser estudiada antes de comenzar a programar. Por tal motivo, en este capítulo veremos cómo está constituido el interior de Android. Estudiaremos sus características de operación en tiempo real y en qué formato de archivo se encapsula una aplicación Android.

Finalmente comprenderemos qué beneficios nos aporta el sistema automatizado de construcción Gradle, dentro de los proyectos de Android Studio.

2.2.2. Versiones

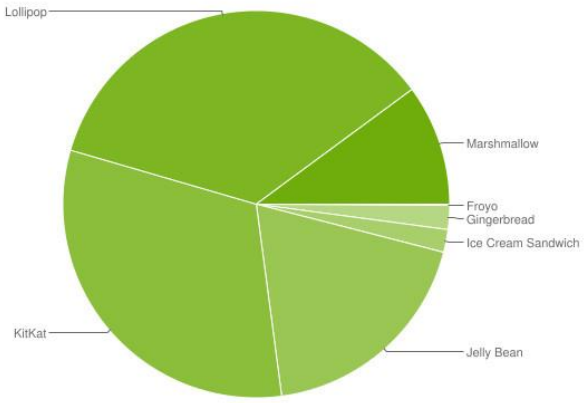
Android como todos los sistemas operativos lanza actualizaciones con frecuencia. A lo largo del año hay varias, una de bastante tamaño que suele ser el lanzamiento de una nueva versión con un nuevo nombre comercial (ej. Kit Kat o Lollipop), y actualizaciones más pequeñas de mejoras limitándose a cambiar el número de la versión.

Nos encontramos con un gran problema en Android al tener un montón de fabricantes distintos. El problema principal es que particularmente cada uno de los fabricantes debe enviarte la actualización adecuándola al hardware de cada dispositivo.

Esto nos repercute de forma que cuando Google lanza una nueva actualización para Android, esta tarde en llegar al usuario. Esto provoca que aunque mejoren el sistema operativo, no todos los usuarios o la gran mayoría se encuentren en la última versión. Como se muestra en la siguiente imagen.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.9%
4.1.x	Jelly Bean	16	6.8%
4.2.x		17	9.4%
4.3		18	2.7%
4.4	KitKat	19	31.6%
5.0	Lollipop	21	15.4%
5.1		22	20.0%
6.0	Marshmallow	23	10.1%

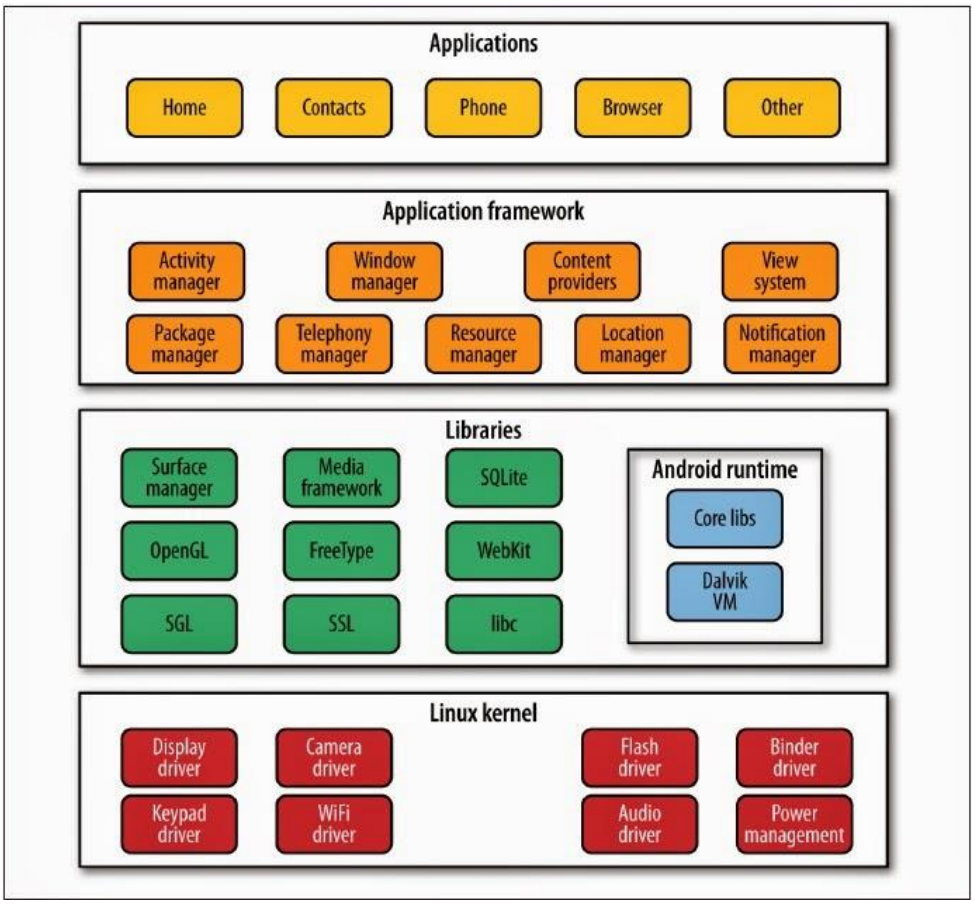
Data collected during a 7-day period ending on June 6, 2016.
Any versions with less than 0.1% distribution are not shown.



Teniendo en cuenta esto, lo que sucede es que te lleva a tomar una decisión en cuanto a la plataforma para la cual desarrollar.

2.2.3. Arquitectura

Android está construido con una arquitectura de 4 capas o niveles relacionados entre sí. A continuación veremos un diagrama ilustrativo: [4]



El diagrama indica que la estructura de Android se encuentra construida sobre el Kernel de Linux. Luego hay una capa de Librerías relacionadas con una estructura administradora en Tiempo de ejecución. En el siguiente nivel encontramos un Framework de apoyo para construcción de aplicaciones y posteriormente vemos a la capa de Aplicaciones.

Kernel De Linux

Android está construido sobre el núcleo de Linux, pero se ha modificado dramáticamente para adaptarse a dispositivos móviles. Esta elección está basada en la excelente potabilidad, flexibilidad y seguridad que Linux presenta. Recuerda que el Kernel de Linux está bajo la licencia GPL, así que en consecuencia Android también.

Capa De Librerías O Capa Nativa

En esta capa se encuentran partes como la HAL, librerías nativas, demonios, las herramientas de consola y manejadores en tiempo de ejecución. Veamos un poco el propósito de estos conceptos:

- **Hardware Abstraction Layer (HAL)**: Este componente es aquél que permite la independencia del hardware. Quiere decir que Android está construido para ejecutarse en cualquier dispositivo móvil sin importar su arquitectura física. El HAL actúa como una **arquitectura genérica** que representa a todos los posibles tipos de hardware existentes en el mercado. Aunque por el momento no hay estándares de construcción en el hardware de dispositivos móviles, el HAL permite que cada fabricante ajuste sus preferencias para que Android sea funcional sobre su tecnología.

- **Librerías nativas**: Aquí encontramos interfaces de código abierto como **OpenGL** para el renderizado de gráficos 3D, **SQLite** para la gestión de bases de datos, **WebKit** para el renderizado de los browsers, etc. También librerías para soportar los servicios del sistema como Wifi, posicionamiento, telefonía, y muchos más.

- **Demonios (Daemons)**: Los demonios son códigos que se ejecutan para ayudar a un servicio del sistema. Por ejemplo cuando se requiere instalar o actualizar una aplicación, el demonio de instalación "**installd**" es ejecutado para administrar todo el proceso. O cuando los desarrolladores vamos a ejecutar en modo de depuración nuestro teléfono desde un PC, se ejecuta un demonio llamado **adbd** (**Android Debug Bridge Daemon**) para auxiliar a dicho proceso.

- **Consola**: Al igual que otros sistemas operativos, Android permite que empleemos **comandos de línea** para la ejecución de procesos del sistema o explorar el sistema operativo.

- **Manejadores en tiempo de ejecución**: Si bien las aplicaciones Android están escritas en lenguaje **Java** y son traducidas a **bytecodes**, estas no son interpretadas por la **Máquina virtual de Java**. Android tiene su propia máquina virtual interpretadora de bytecodes llamada **Dalvik**. Esta herramienta fue diseñada para ser flexible ante el diseño de hardware de un dispositivo móvil. Además JVM no es de licencia GPL, así que Google decidió generar su propia herramienta.

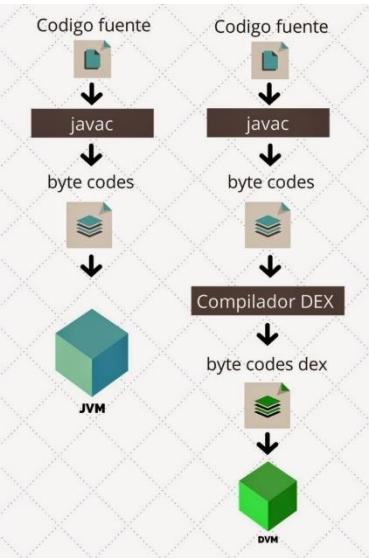
¿Cómo funciona Dalvik?

Dalvik no cambia nada en el proceso de compilación, sencillamente interviene al final como receptor de un archivo ejecutable producto de una recopilación de los **archivos .class de java**.

Las fases de la construcción de una aplicación Java. El primer paso es generar el código fuente (**archivos .java**), luego este es traducido por el **Java Compiler** (javac) y obtenemos un fichero tipo byte code (**archivos .class**). Finalmente la máquina virtual de Java (**JVM**) interpreta en tiempo real este archivo y la aplicación es ejecutada.

La ejecución de **Dalvik** es ingeniosa, simplemente espera que **javac** traduzca la aplicación a byte codes, cuando están listos los archivos, estos son compilados por el **compilador Dex.** Esta herramienta traduce los byte codes de java a un estilo de byte-codes nativos que serán convertidos a un ejecutable **.dex**. Finalmente este archivo es ejecutado por una instancia de **Dalvik VM**.

A continuación se muestra un diagrama comparativo de ambos procesos: [5]



Aunque el proceso añade unos cuantos pasos más, no debes preocuparte por ello, ya que esta tarea se le delega a la herramienta **Gradle**.

Google ha anunciado que **Dalvik VM** será reemplazada por una nueva máquina virtual llamada **ART** (**Android Runtime**) en su nueva versión **Android L**.

Por el momento no nos preocuparemos por esta situación, pero es un dato muy importante a tener en cuenta.

Framework Para Aplicaciones

Esta es la capa que nos interesa a los desarrolladores, ya que en ella encontramos todas las librerías Java que necesitamos para programar nuestras aplicaciones. Los paquetes con más preponderancia son los android.*, en ellos se alojan todas las características necesarias para construir una aplicación Android.

No obstante es posible acceder a clases como java.utils.*, java.net.*, etc. Aunque hay librerías Java excluidas como la java.awt.* y java.swing.*.

En esta capa también encontraremos manejadores, servicios y proveedores de contenido que soportaran la comunicación de nuestra aplicación con el ecosistema de Android.

Capa De Aplicaciones

Es la última instancia de funcionamiento de Android. Se centra en la ejecución, comunicación y estabilidad de las aplicaciones preinstaladas por el fabricante o las que nosotros vamos a construir. A ella acceden todos los usuarios Android debido a su alto nivel de compresión y simplicidad.

¿Qué Tipo De Archivo Tienen Las Aplicaciones Para Android?

El resultado del proceso de construcción es un archivo **comprimido** con formato **.APK** (**Android Applicacion Package**). Y dentro encontraremos los siguientes componentes:

- **Archivo Android Manifest**: Este archivo es la definición de todas las características principales que tendrá nuestra aplicación al ejecutarse en un dispositivo móvil. Con características me refiero a los bloques que posee la aplicación, los permisos, su versión, las versiones previas soportadas, las dimensiones de la pantalla, etc.

- **Archivo classes.dex**: Este será el fichero compilado preparado para ejecutarse en la **Máquina Virtual Dalvik**.

- **La carpeta Resources**: Aquí encontramos todos los archivos externos que usamos para construir nuestro proyecto, como por ejemplo nuestros iconos, audio, archivos planos de texto, los archivos .xml de diseño, etc.

- **Librerías nativas**: El archivo **.APK** también contiene aquellas librerías de las cuales depende la aplicación.

- **Carpeta META-INF**: En ella se guardan archivos que corresponden a las **Firmas Digitales** de tu aplicación. Con esta especificación puedes indicar que tú eres el creador y dueño de la aplicación, además debes indicar tu ID de desarrollador para ser reconocido y autenticado en procesos de comercialización (Google es muy riguroso en este tema).

Gradle, Sistema Automatizado de construcción

Gradle es una herramienta para automatizar el compilado, empaquetado, testeo y liberación de aplicaciones que se basen en la JVM. Como se comentó antes, ha sido creado para expandir el uso de javac.

Cuando me refiero a "automatizar" significa que podemos programar nuestras propias condiciones de construcción. Esto es posible a través de Scripting mediante DSL (Domain-Specific-Language), un lenguaje claro y especializado, con énfasis en configurar y construir aplicaciones con Gradle.

DSL permite generar una compilación basada en tareas programadas y relacionadas entre sí, reduciendo la complejidad de dependencias y automatizando labores frecuentes. Su sencillez permite dar instrucciones de manera declarativa e intuitiva al programador.

¿Qué Ventajas Tiene Utilizar Gradle?

- Una de las mayores ventajas es que le **entrega** el poder del flujo de construcción al programador. Esto quiero decir que decidimos el orden de ejecución de tareas. Asimismo podríamos elegir que archivos compilar primero, cuando detener la compilación, establecer condiciones para que se recompile o no el código y muchas situaciones más.

- **Ejecuciones incrementales**: El poder de esta característica le ahorra al programador gran cantidad de tiempo de espera. Al haber construcciones incrementales podemos decidir hasta qué punto queremos que se compile nuestra aplicación, es decir, si no es necesario compilar una parte del código debido a la ausencia de errores, entonces se procede a compilar la sección que aún no ha sido probada.

- **Múltiples versiones**: Gradle permite que construyamos varias versiones de nuestra aplicación. Por ejemplo, si deseas construir tu aplicación para **Jelly Beans** y para **Kitkat** entonces solo debes

especificar que el proyecto tendrá dos variantes de empaquetado, configurando la versión del **SDK** usada para cada versión.

- La ejecución y las pruebas se pueden realizar en un mismo proyecto.

- **Ejecución en paralelo de tareas:** Puedes ejecutar tareas en hilos diferentes para optimizar el proceso de construcción.

Esas son las características más importantes, sin embargo hay muchas más.

Conclusión

En este capítulo explicamos la forma en que está construido Android como sistema operativo.

Vimos cómo se construyen las aplicaciones Android a través de un modelo de máquina virtual y como interviene el framework.

Incluso vimos qué ventajas tiene usar Gradle para automatizar la compilación de las aplicaciones.

Capítulo 3 Requisitos

3.1. Introducción

Una vez ya nos hemos adentrado en conocer cómo funciona el sistema operativo Android y le hemos conocido internamente, es posible afrontar el comienzo del desarrollo de la App.

La aplicación de apoyo a la recepción de trabajos para APTACAN ha sido pensada para que los alumnos de esta asociación puedan realizar el trabajo de forma fácil, cómoda y en cualquier lugar de forma autónoma pudiendo atender los trabajos del área de reprografía donde se encuentren.

Desde el punto de vista de la aplicación, en cada trabajo hay los siguientes componentes:

- Definición del trabajo, es decir, crear una plantilla de los datos necesarios para llevarlo a cabo.
- Realización del trabajo, cuando el alumno utiliza la aplicación para pedir los datos al cliente y realizar lo pedido.
- Cobro final del trabajo realizado.

Entonces nos encontramos que la aplicación cuenta con tres servicios principales que son los siguientes, la configuración de los trabajos y sus respectivos elementos creando las plantillas, la consulta y gestión (Añadir y eliminar) de las imágenes a utilizar en las plantillas y la gestión del dinero de la caja para los cobros de los trabajos realizados y el uso del cliente para solicitar los trabajos que desea realizar en la reprografía.

En este capítulo se profundiza en los requisitos que se usan a la hora de planificar el desarrollo de la aplicación.

3.2. Requisitos del Proyecto

Los requisitos de la aplicación, como sucede en los proyectos reales casi siempre, se van modificando a lo largo del desarrollo del proyecto. Esto nos lleva a añadir nuevos o eliminar otros con el paso del tiempo.

El modo en el que hemos trabajado con APTACAN ha sido a través de reuniones y demostraciones del trabajo ya realizado, donde se les enseñaban los avances y ellos corroboraban el trabajo con su visto bueno o cambiaban algunos requisitos.

En un primer lugar, tras aceptar este proyecto, y con las ideas iniciales que me comentó mi tutor, se celebró una primera reunión de donde se obtuvieron unas primeras ideas de la aplicación, que fueron las siguientes:

- Hablar, donde la aplicación debería responder con sonido en algunas ocasiones, como por ejemplo diciendo la cantidad de euros que cuesta el trabajo.

- Escuchar, en momentos determinados la aplicación tenía que ser capaz de recoger lo que el cliente decía.

- Pantalla de bienvenida, saludo al cliente.
- Menú con los trabajos que se ofrecían en ese momento en reprografía, como por ejemplo Comprar o Fotocopiar.

- Plantillas determinadas con los datos necesarios para realizar los trabajos que se encontraban en ese momento en el catálogo.

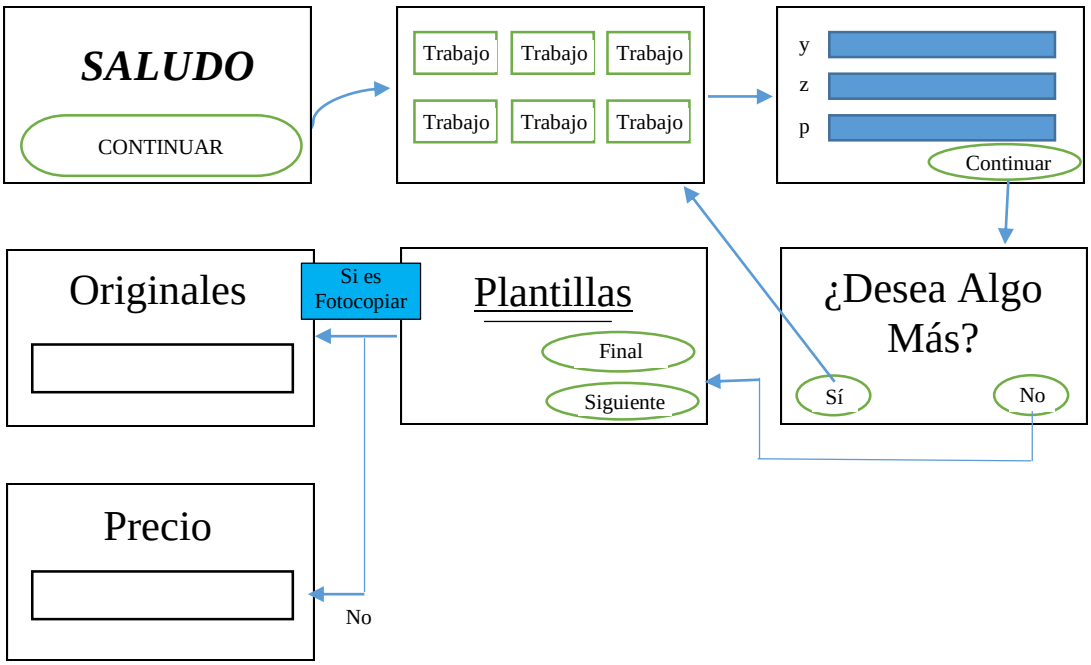
- Lugar o pantallas donde se mostrasen las tareas a realizar mediante imágenes y, por ejemplo, si es fotocopiar un lugar donde meter el número de originales.

- Una guía para el cobro del precio final por todos los trabajos realizados, resolviendo qué cambio hay que devolver.

Con estos requisitos ya definidos, pasamos a estudiar la manera de integrarlos todos juntos en la App.

Primero tratamos el tema del reconocimiento de voz y la forma de hablar, tras discutir las posibilidades que había para afrontar esto, nos dimos cuenta que era imprescindible estar conectado a internet para añadir estas funcionalidades. Además de que había que tener en cuenta el ruido que hubiese en el lugar de trabajo no interfiriese en la apreciación. Estas limitaciones al comentárselas nos llevó a tomar la decisión de desechar estos dos requisitos de la aplicación.

Por último diseñe la siguiente secuencia de pantallas para hacernos una idea de cómo sería la App, como se observa en la siguientes figuras.



Una vez analizados estos requisitos, se celebró un segundo encuentro en el lugar de trabajo de APTACAN, donde vi en primera persona la forma de trabajo en reprografía. Se les enseño la secuencia de pantallas que mostramos en las figuras anteriores y cambiamos impresiones sobre todas las funcionalidades.

A partir de aquí se decidió que una parte de la App fuese dinámica, es decir, que los responsables pudiesen configurar gran parte de los trabajos. Se propuso que hubiese un apartado donde se pudiesen gestionar las imágenes, introducir, ver y borrar. Un lugar en el cual se creasen las plantillas de los trabajos y a su vez dentro de cada trabajo las partes de las que consta cada uno con sus respectivas imágenes asociadas. Y por último un lugar donde se pueda cargar la caja de reprografía con el dinero disponible en ese momento.

En la parte del cliente, habría un menú mayormente de botones donde se cargarían los almacenados en la parte anterior por los responsables de la plantilla asociada a ese trabajo. Una vez elegidos los trabajos se mostrarían secuencialmente las imágenes asociadas, de una en una. Estas imágenes representan los pasos a dar para completar el trabajo, que el alumno irá siguiendo. Y por último una parte donde se pueda trabajar el cobro del trabajo o trabajos realizados.

3.3. Requisitos finales

Una vez que se recopiló esta nueva información, ya podemos obtener los requisitos definitivos para comenzar con el desarrollo de la App. Esta cuenta fundamentalmente con tres grandes cuestiones una parte de administración, una parte para trabajar el cliente y por último la parte de información para el trabajador.

A continuación detallamos los requisitos en función de cada uno de las distintas cuestiones que aborda la App.

- Administración, lugar donde se configuran los distintos elementos para los distintos trabajos.
    - Imágenes, lugar donde se puedan incorporar imágenes nuevas y otro desde el cual se puedan ver o eliminar las existentes.
    - Trabajos, en esta parte constará de varias pantallas donde se puedan ver los trabajos existentes, a su vez eliminar o editar y puedan crear nuevos. En cada trabajo se podrá ver cada uno de los elementos que hay para su respectiva plantilla que también se podrá ver, eliminar o editar y crear nuevos.
    - Caja, pantalla donde se pueda eliminar el dinero que hay en la caja o bien se puedan ingresar las monedas con las que se cuentan.
- Cliente, lugar donde se interactúa con el cliente, pudiendo este seleccionar los trabajos que desea y rellenar las plantillas.
    - Pantalla de bienvenida para el cliente.
    - Menú con los distintos trabajos a realizar.
    - Posibilidad de realizar más de un trabajo en una misma sesión.
    - Plantillas para recoger los datos necesarios de cada trabajo.
    - En el caso que lo requiera, pantalla donde introducir los originales aportados por el cliente.
    - Precio Final del trabajo o los trabajos realizados.
- Trabajador, lugar donde se muestran las pautas a seguir para desarrollar el trabajo.
    - Pantalla donde se muestran los trabajos secuencialmente de uno en uno.

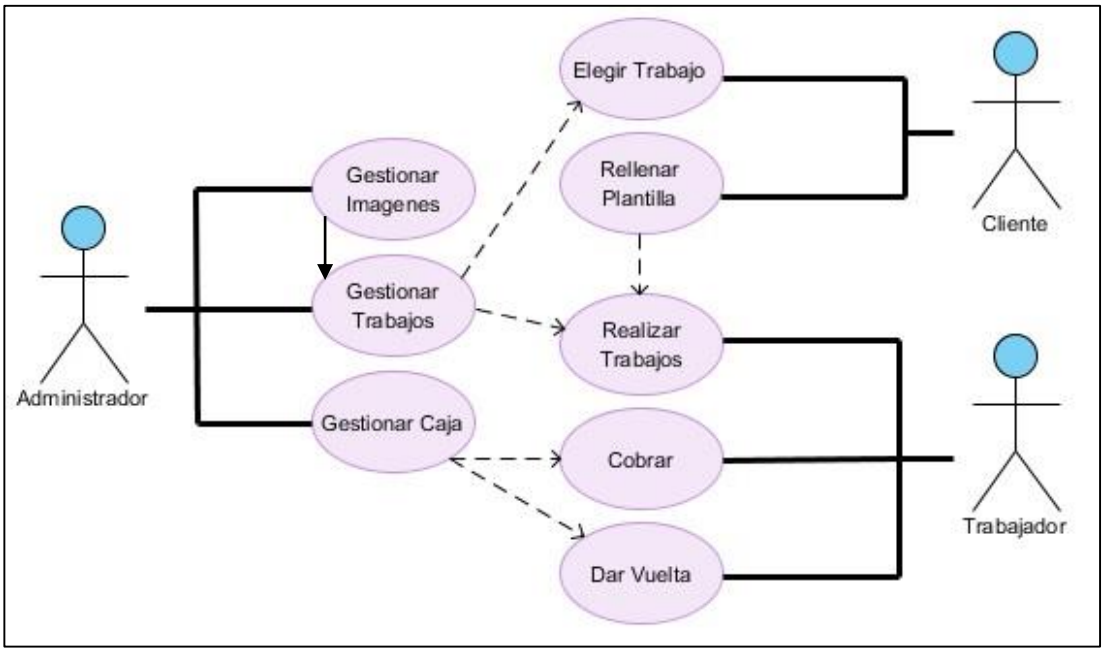
- Pantalla de cobro con las monedas y billetes aceptados para introducirles uno a uno.
- Pantalla si el dinero es exacto.
- Pantalla con el cambio de forma secuencial.
- Avisar si ya no es posible dar más cambio.
- Pantalla de despedida.

Capítulo 4 Diseño básico

4.1. Diagrama de casos de usos

Un caso de uso representa el uso típico que da o puede dar un usuario a una aplicación. Muestra de una forma sencilla y resumida la funcionalidad que tendrá asociada cada tipo de usuario de la aplicación para facilitar dicha información al cliente.

En el caso de la aplicación de APTACAN tan sólo hay tres tipos de usuarios como se puede ver en el diagrama de casos de uso de la figura. El primer tipo, que se ha denominado “administrador” se trata de cualquier miembro de APTACAN que trabaje con los alumnos. El segundo tipo de usuario, “cliente”, se trata de las personas que vienen para que se les realice un trabajo. El tercer tipo son los “trabajadores”, que realizan los trabajos y cobran el valor de estos.

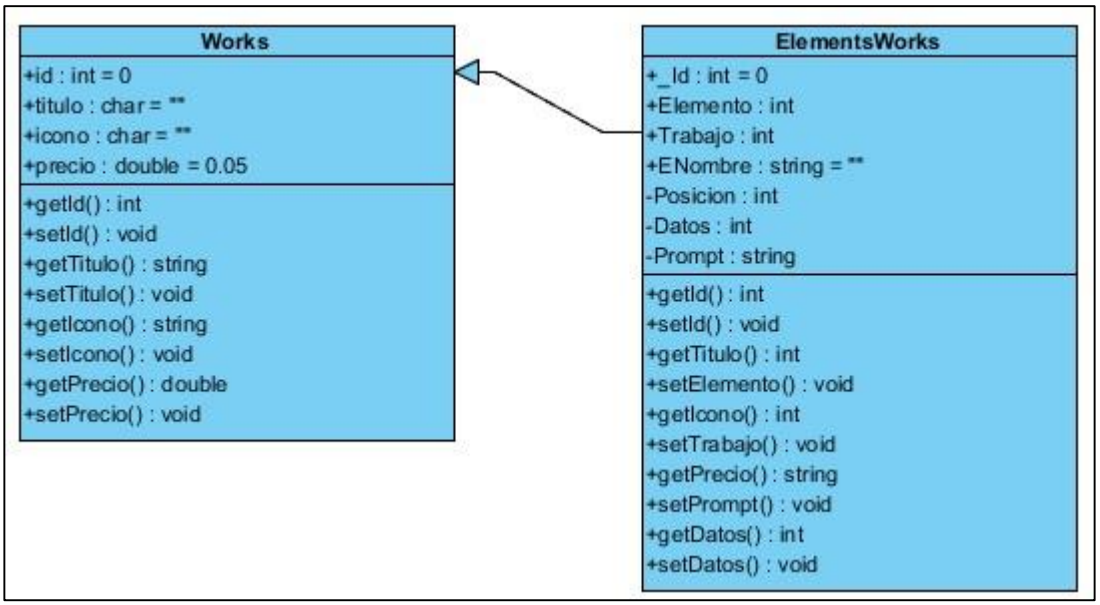


4.2. Diagrama de clases

El diagrama de clases es aquél donde se describen los atributos y operaciones de una clase y también las limitaciones impuestas al sistema. Los diagramas de clases son ampliamente utilizados en el modelado de sistemas orientados a objetos, ya que son los únicos diagramas UML, que se pueden asignar directamente con lenguajes orientados a objetos.

La aplicación cuenta con dos clases, los trabajos y los elementos de dichos trabajos cómo se observa en el diagrama de clases de la figura. Se componen de distintos atributos y poseen métodos sencillos de get() y set(), a la vez que

están relacionados entre sí por medio del id del trabajo. Les une la relación de que un trabajo puede tener varios elementos pero cada elemento sólo puede pertenecer a un trabajo.



4.3. Diagrama de actividad

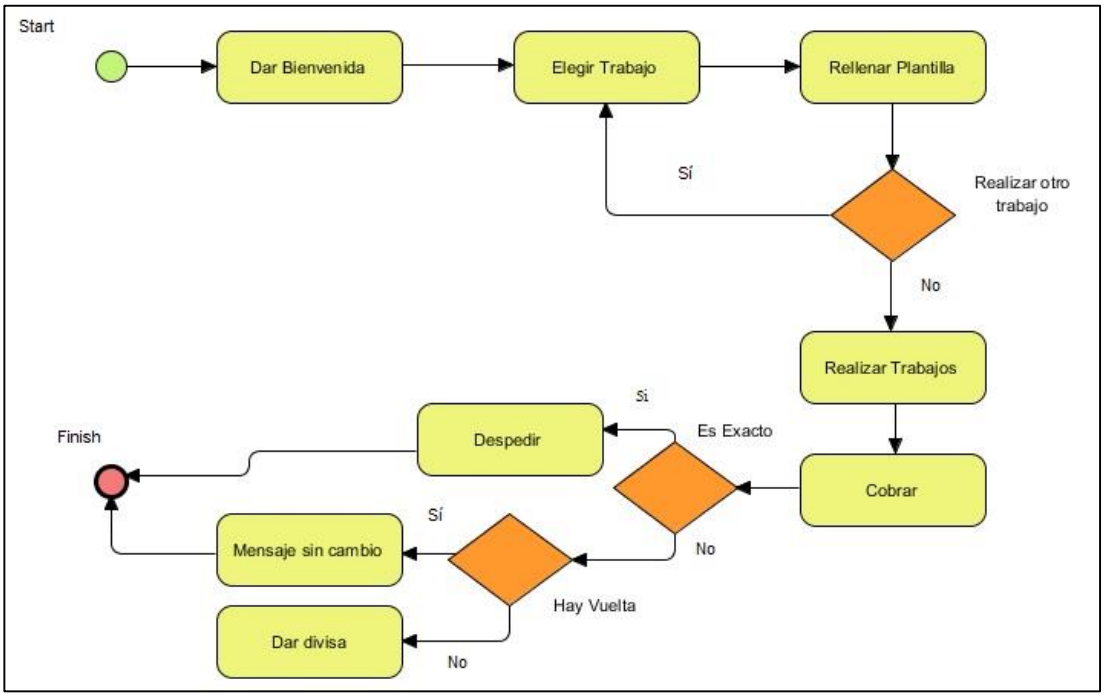
Los diagramas de actividades sirven para representar el comportamiento dinámico de un sistema haciendo hincapié en la secuencia de actividades que se llevan a cabo y las condiciones que guardan o disparan esas actividades.

El diagrama de actividad es aquél donde se muestra el flujo de control de una actividad a otra, sin incluir los mensajes. Este diagrama es adecuado para modelar el flujo de actividad del sistema.

La aplicación cuenta sobre todo con una actividad principal, donde se realiza todo el proceso para llevar a cabo todo el trabajo. En la figura se puede observar cómo llega el cliente, recibe un saludo y elige un trabajo. Una vez elegido el trabajo se le carga dinámicamente la plantilla que tiene que rellenar para ese trabajo. Una vez relleno tiene que decidir si quiere realizar más trabajos.

En esta decisión se vuelve a repetir el proceso anterior o el trabajador empieza a realizar el trabajo o los trabajos. Una vez terminados se pasa a cobrar.

En la acción de cobrar si el pago está hecho con la cantidad justa se despide al cliente, sino se procede a darle la vuelta, si hay dinero para dar la vuelta se va dando elemento a elemento y se despide. Si no, sale un mensaje para que uno de los responsables pueda solucionarlo.


Start
Dar Bienvenida
Elegir Trabajo
Rellenar Plantilla
Sí
Realizar otro trabajo
No
Realizar Trabajos
Sí
Despedir
Es Exacto
Finish
Cobrar
Sí
Mensaje sin cambio
No
Hay Vuelta
Dar divisa
No

Capítulo 5 Diseño general

5.1. Introducción

Una vez recogidos todos los requisitos, se dio comienzo al diseño de las funcionalidades necesarias en la App, a la vez que se inicia a trabajar en Android Studio.

Como ya se ha comentado con anterioridad, la aplicación se va a llevar a cabo en tres partes distintas, pero a su vez estas se van a dividir para solventar cada cuestión de la forma más sencilla posible.

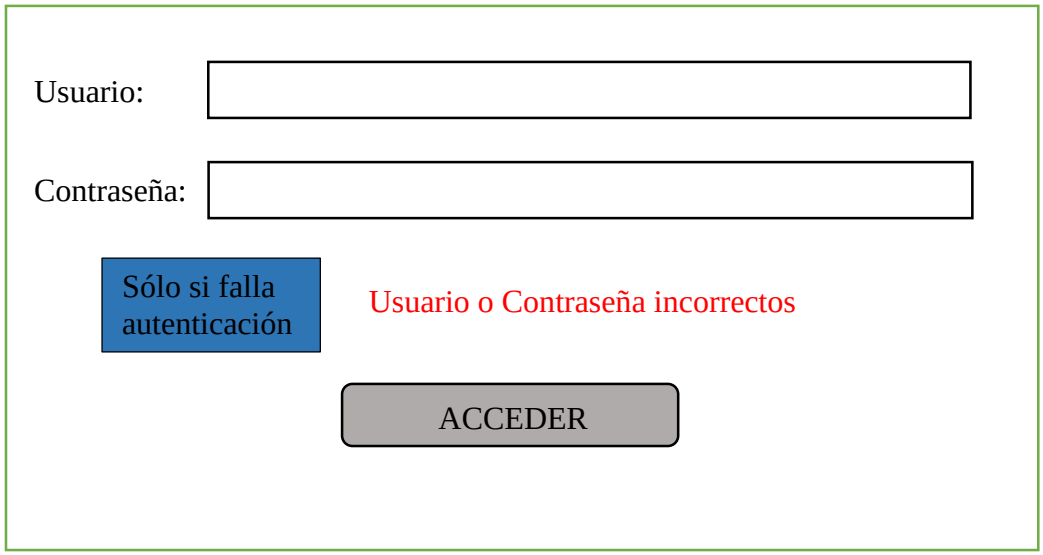
Por lo tanto en cada uno de las siguientes secciones de este capítulo van a ser los siguientes:

- Administración.
- Cliente.
- Trabajador.

5.2. Administración

Acceso

En esta parte de la App lo primero que nos encontramos es una pantalla para poder acceder a esta zona con un usuario y una contraseña. Desde aquí si introducimos correctamente los datos de acceso pasamos al menú de esta sección de la App, que definiremos en el siguiente apartado. Observa la siguiente figura:



Administración

Una vez accedidos llegamos al menú de administración, donde nos encontramos tres grandes grupos y es aquí donde otra vez vamos a volver a dividir en distintos problemas. Estos son Trabajos, Imágenes y Monedero.

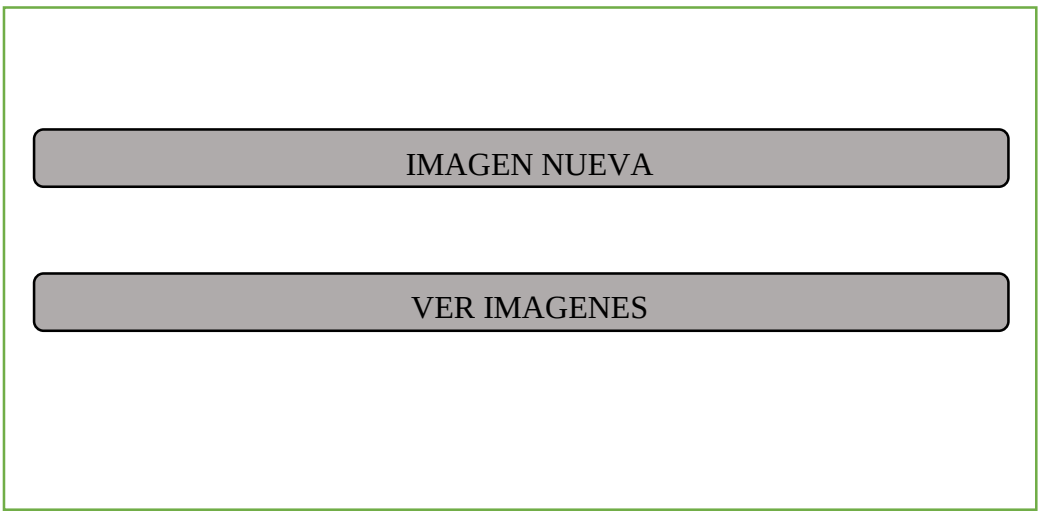
Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

De los tres problemas que se nos plantean ahora vamos a comenzar resolviendo primero el de las imágenes, posteriormente los trabajos y por último el monedero.

Imágenes

Menú Imágenes

Desde el menú de administración al pulsar sobre imágenes llegamos a otro menú. En este caso vamos a gestionar las imágenes que se usaran en la app. Aquí vamos a llevar a cabo las dos tareas que nos piden en los requisitos: añadir imágenes y ver o eliminar las existentes.
Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

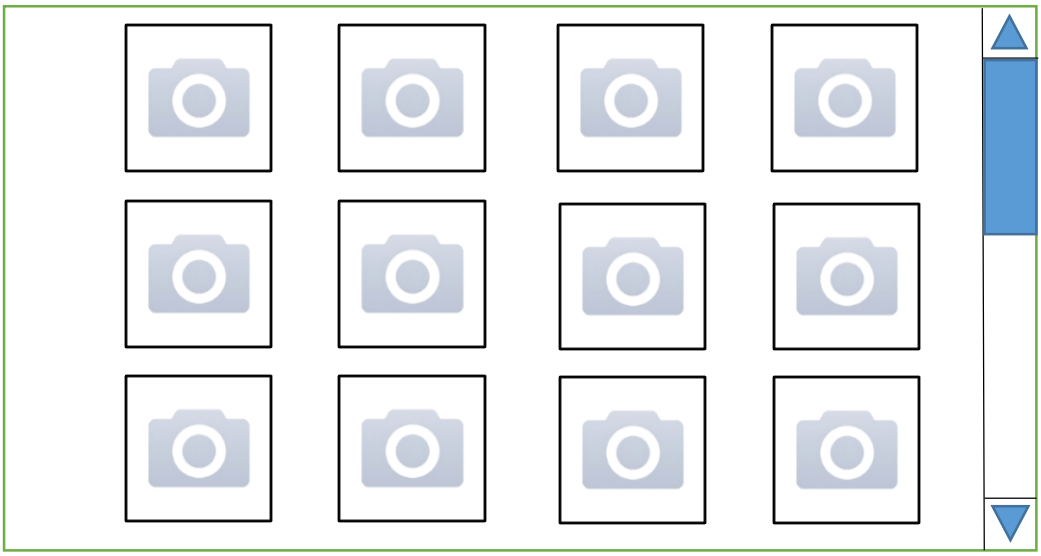


Pulsando sobre el primer botón primer botón se nos mostrará la gallería por defecto del dispositivo usado, pudiendo en algunos casos tomar directamente fotos, para su posterior uso en la App. El segundo botón nos lleva a la siguiente pantalla de la App, que mostraremos en el siguiente apartado.

GridViewActivity

Una vez que pulsamos sobre el segundo botón del menú anterior, podemos ver cómo nos aparecen las imágenes que hemos añadido anteriormente con el primer botón del menú. Éstas se pueden eliminar, haciendo clic sobre la imagen.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:



Con estas pantallas ya hemos definido y resuelto toda la parte referente a las imágenes. Con lo cual a partir de ahora empezaremos a definir todo lo referente al apartado de los trabajos.

Trabajos

En este apartado vamos a desarrollar la sección referente a los trabajos, dicha parte es complicada especialmente debido a que uno de los requisitos es poder crear, modificar y eliminar todas las partes que lo componen.

Para comenzar este desarrollo es importante tener claro cómo se van a dividir los trabajos, esto nos lleva a tener claro que la App tiene que tener un trabajo o varios. A su vez cada trabajo puede tener uno, varios o ningún elemento, y estos están complementados con los datos y cada uno de ellos tendrá asociada una imagen.

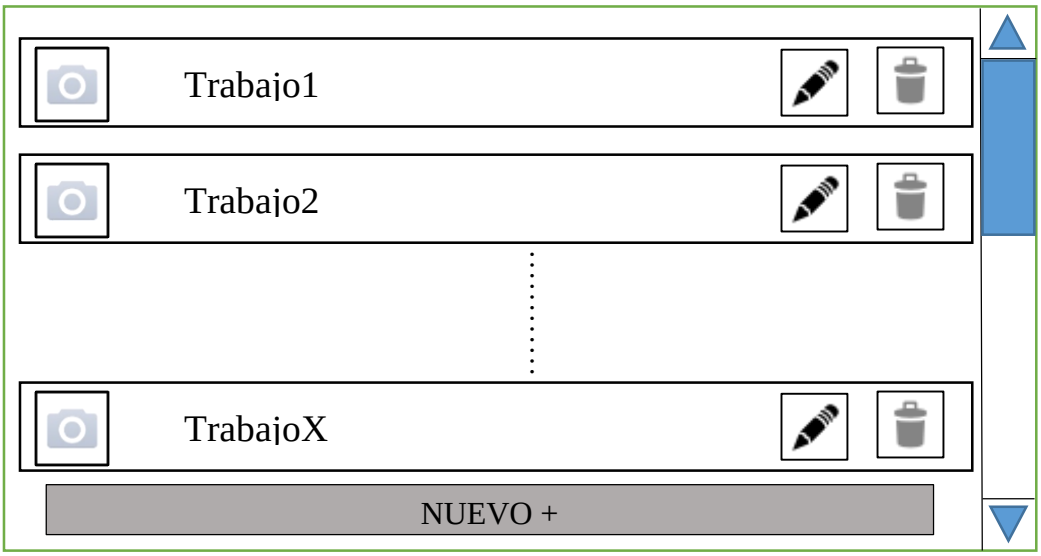
Todo esto se almacenara en el dispositivo de forma permanente. Esto será explicado en uno de los capítulos posteriores, por ahora seguimos mostrando el diseño de las distintas pantallas de este apartado de la App.

ListViewWorks

Al pulsar en el menú de administración sobre el botón de trabajos, nos lleva a esta pantalla, la cual nos muestra en una lista todos los trabajos que tenemos.

La lista está dividida en filas, una por trabajo existente y está compuesta por la imagen asociada al trabajo, el nombre de este y dos botones con imágenes una para editar y otra para eliminar. Debajo de la pantalla encontraremos un botón para crear un trabajo nuevo.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

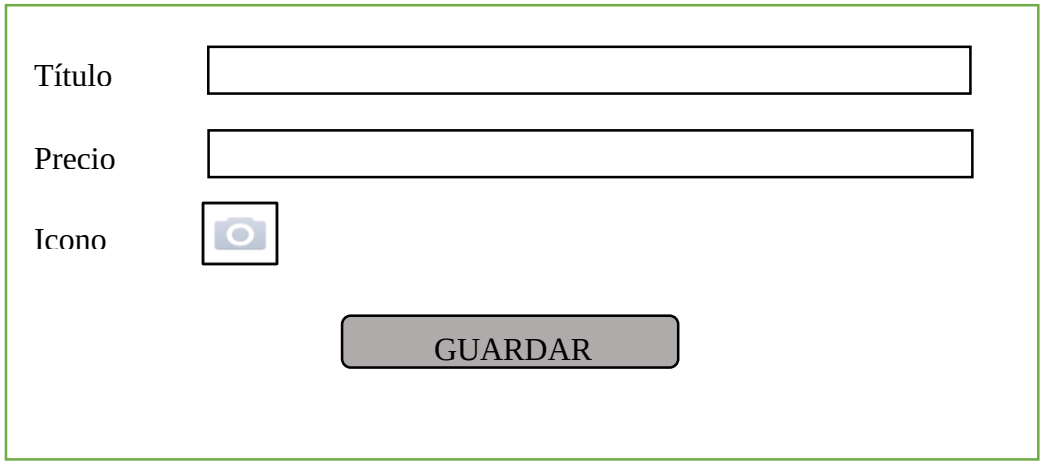


Pulsando sobre el botón de la papelera eliminamos el trabajo y sobre el lápiz o el botón nuevo + pasamos a la pantalla para introducir los datos sobre el trabajo. El otro caso posible es pulsar sobre uno de los trabajos de la lista que nos llevara a la pantalla de los elementos de ese trabajo.

Trabajo

A esta pantalla se llega cuando se pulsa sobre el lápiz o el botón nuevo +.
Aquí van a poder introducir los datos necesarios para definir un trabajo, con diferencias dependiendo desde donde se llegue a esta pantalla. Cuando llegamos desde nuevo + los lugares donde introducir los datos aparecerán vacíos, en cambio desde el lápiz obtendremos los datos que ya tiene el trabajo para editarlos en el caso que así se estime.

La pantalla sería, como se muestra en la siguiente figura:



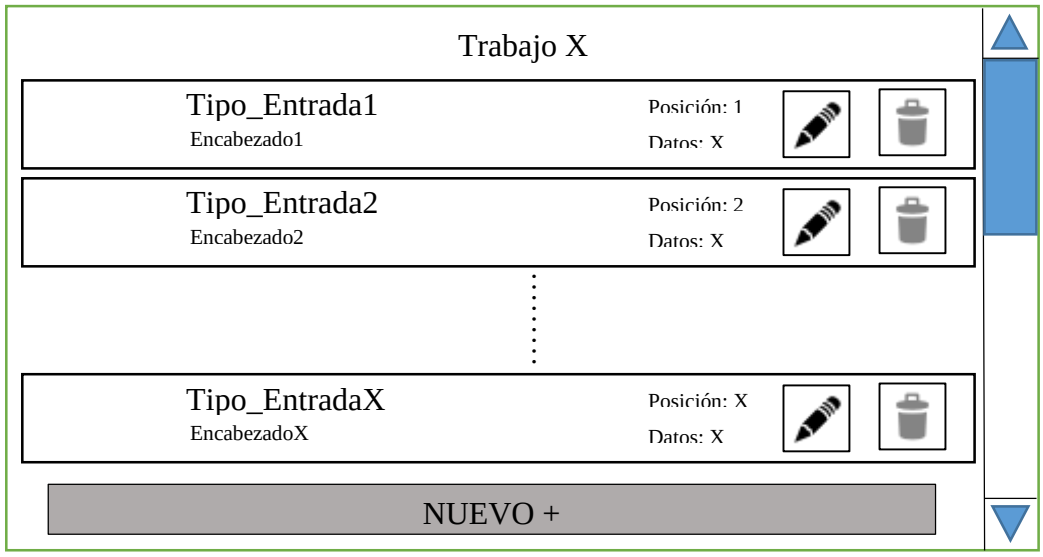
Una vez introducidos los datos mediante el botón guardar, donde se guardaran los cambios introducidos o los nuevos datos. Posteriormente regresaremos a la pantalla anterior.

ListViewWorksElements

Al pulsar en la lista de trabajos sobre uno ellos, nos lleva a esta pantalla, la cual nos muestra en una lista todos los elementos del trabajo seleccionado en la pulsación.

La lista está dividida en filas, una por cada elemento existente y esta compuesta por el tipo de elemento, el encabezado de este, la posición que ocupa en la plantilla, los datos por los que está compuesto y dos botones con imágenes una para editar y otra para eliminar. Debajo de la pantalla encontraremos un botón para crear un elemento nuevo.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:



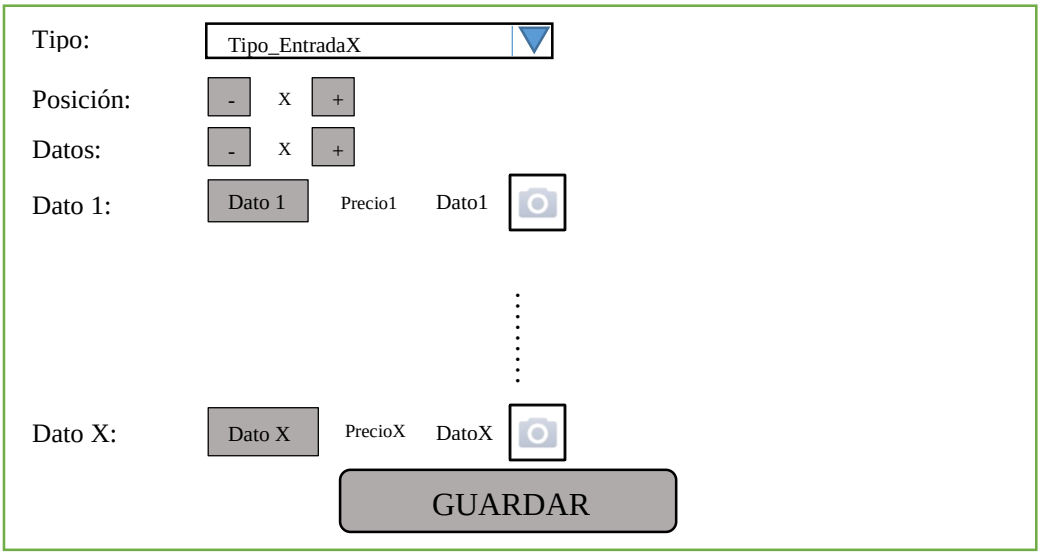
Pulsando sobre el botón de la papelera eliminamos el trabajo y sobre el lápiz o el botón nuevo + pasamos a la pantalla para introducir los datos sobre el elemento.

Elemento

A está pantalla se llega al pulsar sobre el lápiz o el botón nuevo +.

Aquí van a poder introducir los datos necesarios para definir un elemento, con diferencias dependiendo desde donde se llegue. Cuando llegamos desde nuevo + los lugares donde introducir los datos aparecerán vacíos, en cambio desde el lápiz obtendremos los datos que ya tiene el elemento para editarlos en el caso que así se estime.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:



Desde los botones del dato se puede acceder a otra pantalla, comentada en el siguiente apartado.

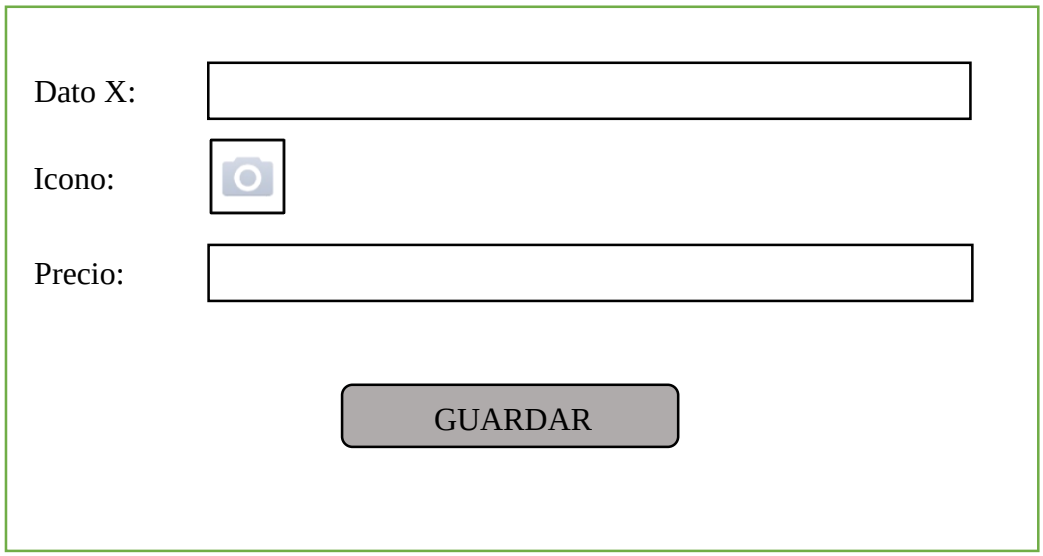
Una vez introducidos los datos mediante el botón guardar, se guardarán los cambios introducidos o los nuevos datos. Posteriormente regresaremos a la pantalla anterior.

DatoElemento

A esta pantalla se llega cuando se pulsa sobre el botón de un dato.

Aquí se van a poder introducir los datos necesarios que hay dentro de un elemento seleccionando la imagen asociada a éste, menos en el caso del primer dato, que hará referencia al encabezado de los datos cuando el elemento pueda tener más de un dato.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:



Una vez introducidos los datos mediante el botón guardar, se almacenarán los cambios introducidos o los nuevos datos. Posteriormente regresaremos a la pantalla anterior.

Con estas pantallas ya hemos definido y resuelto toda la parte referente a los trabajos. Con lo cual a partir de ahora empezaremos a definir todo lo referente al apartado del monedero.

Monedero

En este apartado vamos a desarrollar la sección referente al monedero, dicha parte es la encargada de controlar el dinero que existe en la caja en el lugar de trabajo.

Para comenzar este desarrollo es importante tener claro las monedas y billetes que se almacenan en la caja. Vamos a controlar la cantidad de cada una de ellas para tener conocimiento de todo el dinero que tenemos en la caja.

Todo esto se almacenará en el dispositivo de forma permanente. El almacenamiento será explicado en uno de los capítulos posteriores; por ahora seguimos mostrando el diseño de las distintas pantallas de este apartado de la App.

Monedero

Al pulsar en el menú de administración sobre el botón de monedero, nos lleva a esta pantalla, la cual nos muestra en una tabla las distintas monedas y billetes que se pueden tener en la caja.

La pantalla está compuesta, por una tabla con las opciones de dinero, un lugar donde nos muestra el efectivo y un botón para vaciar la caja y dejar todas las opciones a cero.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

Al pulsar sobre una de las imágenes del dinero se añade una unidad de esa moneda o billete y el contador irá mostrando el total. La parte de arriba a la izquierda nos muestra el total de dinero que se tiene en la caja. Por último el botón de limpiar caja nos vacía el almacenamiento y pone todos los contadores a cero.

Con esta pantalla ya hemos definido y resuelto toda la parte referente al monedero y por lo tanto hemos terminado la parte de Administración.

5.3. Cliente

Parte Cliente

Esta es la parte de la App encargada de interactuar con el cliente, es decir, el conjunto de pantallas desde las cuales va a tomar las decisiones sobre los trabajos que desea realizar e introducir los datos con sus preferencias para cada uno de los trabajos elegidos.

MainActivity

En esta parte de la App lo primero que nos encontramos es una pantalla para dar la bienvenida al cliente. Desde aquí si pulsamos en el botón continuar pasamos a las pantallas de pedir un trabajo. Los administradores desde el menú de arriba podrán acceder a la parte de administración.

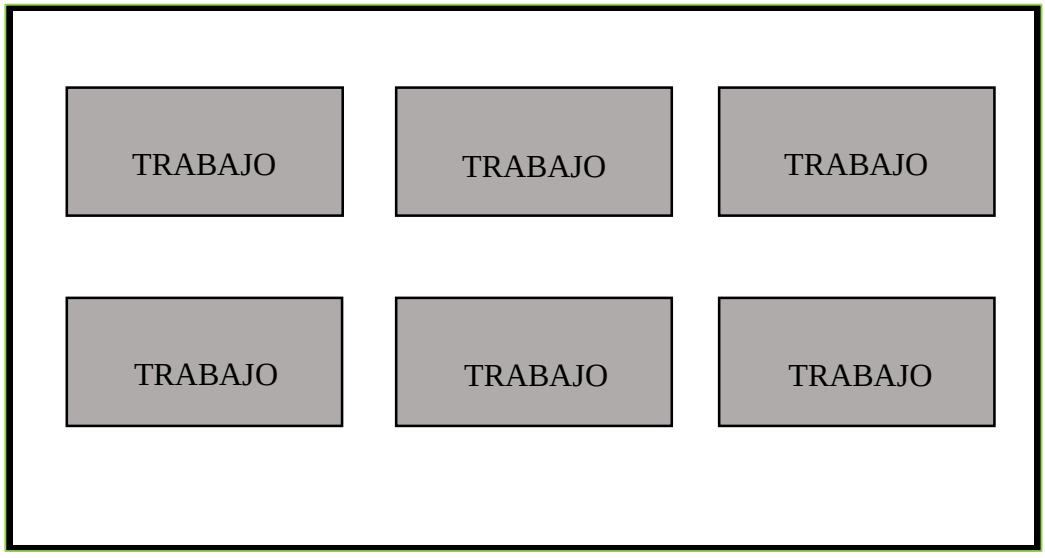
Será como la siguiente figura:

Al pulsar sobre los puntos blancos aparece el menú que se encuentra debajo, desde donde se puede acceder a la parte de administración o salir de la aplicación. Para el cliente se encuentra en la parte inferior el botón de continuar desde donde se accede a la siguiente pantalla.

Menu_Trabajos

Desde el botón continuar de la pantalla anterior accedemos a ésta, la cual se va a formar dinámicamente en función de los trabajos que tengamos almacenados en la App. Se va a construir una tabla con un botón en cada una de sus celdas, estableciendo un máximo de tres botones por fila.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

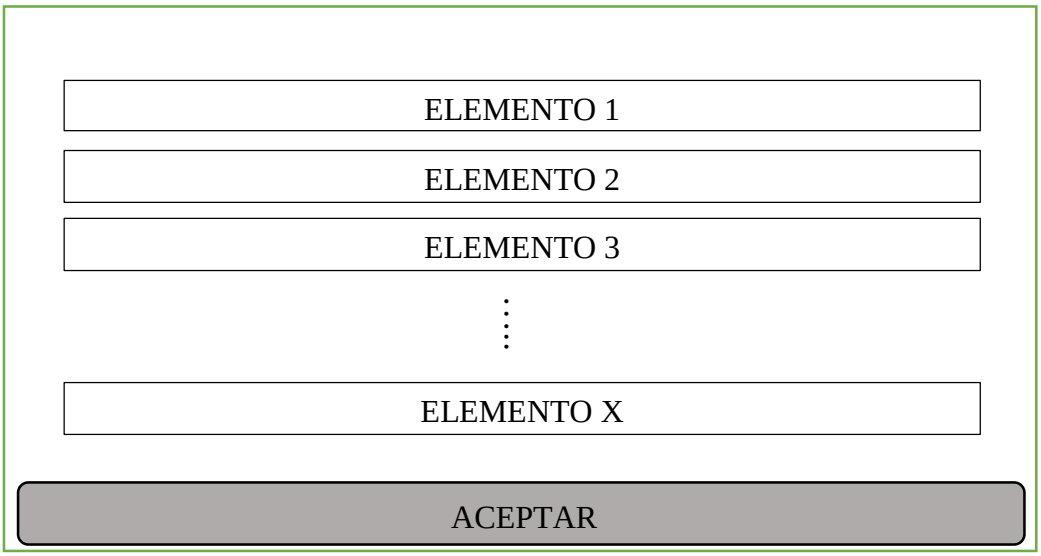


Al pulsar sobre cualquiera de los trabajos que se nos muestren en la pantalla accederemos a la siguiente pantalla.

Layout_Trabajo

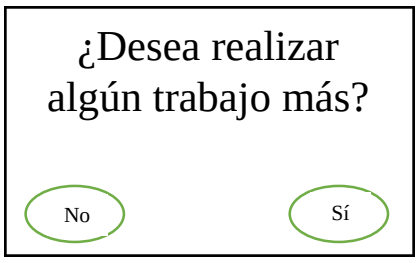
Una vez seleccionado un trabajo en la pantalla anterior, aquí se mostraran todos los elementos con los que se determina el trabajo. Se trata de una pantalla con una dificultad más grande, debido a que se construye dinámicamente. Estos elementos se corresponden con los pasos que se crearon en la parte de administración al definir el trabajo y les definiremos posteriormente, ya que se trata de un conjunto de elementos cerrado para poder configurar la plantilla.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:



Una vez rellenados todos los datos en cada uno de los elementos, pulsamos el botón aceptar donde nos mostrara una pregunta para ver si queremos realizar más trabajos. La respuesta positiva nos devolverá al menú de los trabajos y en caso negativo ya pasaremos a la parte del trabajador.

En la siguiente figura, puede verse el caso afirmativo.



Éstas son todas las pantallas que interactúan con el cliente, siendo el mismo cliente quien elige los trabajos que quiere realizar y las opciones dentro de cada trabajo adecuando éste a todas sus preferencias.

Por lo tanto hemos terminado la parte de Cliente.

5.4. Trabajador

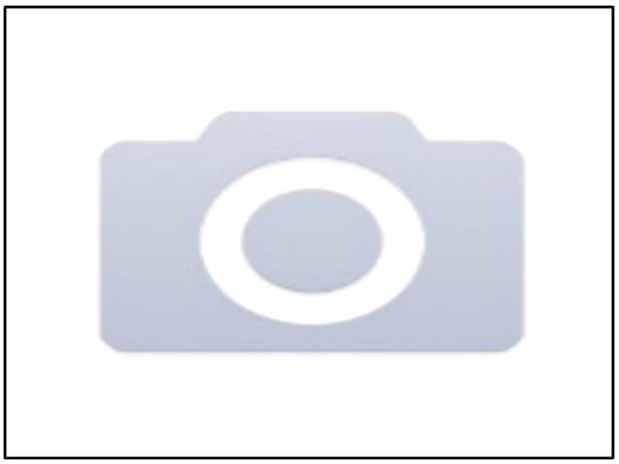
Parte Trabajador

Esta es la parte de la App encargada de interactuar con el trabajador, es decir, el conjunto de pantallas desde las cuales va a ver los trabajos que tiene que realizar y la parte donde le va a cobrar.

Ver_resultados

En esta parte de la App lo primero que nos encontramos es una pantalla con la imagen asociada al primer trabajo a realizar. Cada vez que pulse sobre la imagen va a ir cambiando la imagen al siguiente paso en la realización del trabajo, con las opciones elegidas por el cliente. Tras acabarlo, pasará a los otros trabajos que han podido ser seleccionados por el cliente.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:



Dato

Se va pulsando sobre la imagen todas las veces que sea necesario hasta que se llega a la última imagen cuando se han terminado los pasos de todos los trabajos. Ya desde aquí vamos a pasar a la siguiente pantalla.

Pagar

Una vez pasadas todas las imágenes asociadas a los pasos de todos los trabajos, aparece la pantalla que nos indica la cantidad total que nos han valido todos los trabajos realizados teniendo en cuenta las opciones escogidas.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

Al pulsar sobre el botón continuar pasamos a cobrar al cliente en la siguiente pantalla. Se le mostrara al cliente el valor total de los trabajos.

Pago

Una vez enseñado al cliente, aparece la pantalla que nos muestra las posibilidades de dinero que nos pueden entregar hasta llegar al valor.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

Según el cliente le va entregando el dinero al trabajador este pulsa en el botón asociado a la moneda o billete correspondiente, hasta que en la parte quedan es cero o inferior a cero. Si el dinero entregado es exacto se pasa a una pantalla dando las gracias, en caso de que no sea así se pasa a la entrega de las vueltas.

Vueltas

Cuando el dinero entregado es superior al valor de los trabajos realizados accedemos a está pantalla para darle la vuelta al cliente del dinero entregado. Nos muestra el valor que tenemos que ir devolviendo al cliente mediante una imagen.

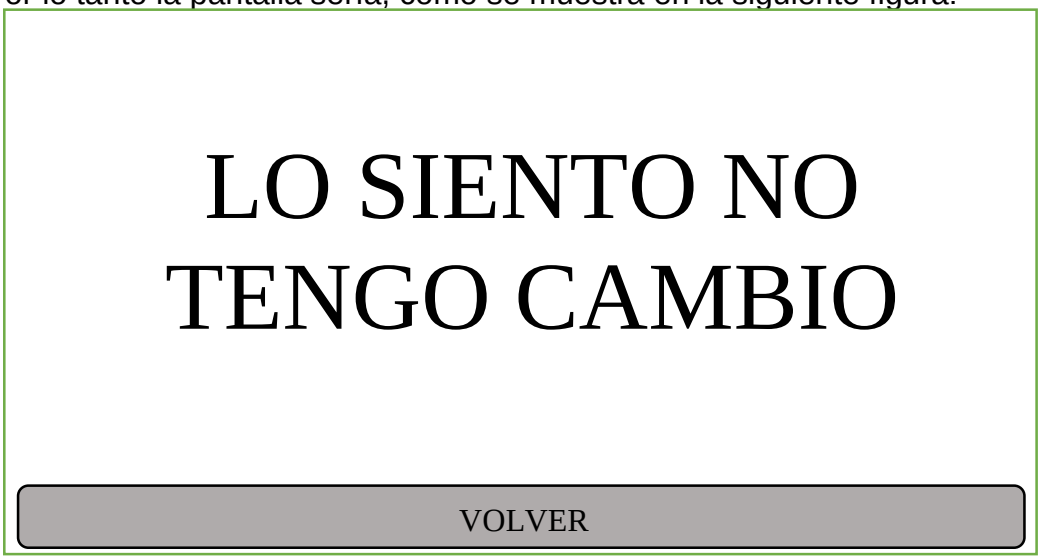
Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

Al mostrar la imagen se le da la moneda o billete mostrado al cliente y se avanza. Si todavía queda dinero que devolver se mostrara la siguiente imagen a devolver; si ya se han dado todas las vueltas se accede a la pantalla de despedida; y si la caja no tiene dinero exacto para las vueltas nos mostrara otra pantalla.

SinCambio

Si no se tiene la cantidad exacta para dar la vuelta al cliente aparece un mensaje comunicándoselo al cliente y avisando a las personas que están a su cargo.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:



Al pulsar sobre el botón de volver regresamos a la pantalla de bienvenida inicial.

Despedir

Al entregar el dinero el cliente si este es exacto o al terminar de dar las vueltas correctamente accedemos a esta pantalla. Aquí se le da un mensaje al cliente después de finalizar todo el proceso de los trabajos.

Por lo tanto la pantalla sería, como se muestra en la siguiente figura:

Al pulsar sobre el botón de volver regresamos a la pantalla de bienvenida inicial.

5.5. Elementos

Los elementos son las partes que pueden formar la plantilla de un trabajo, son una parte importante dentro de la aplicación. Se cuenta con un determinado grupo de ellos y sólo estos es posible que formen parte de la plantilla del trabajo a mostrar a los posibles clientes.

Tras varias conversaciones sobre este tema, en las reuniones de los requisitos, llegamos en consenso con los miembros de APTACAN se definieron cinco tipos distintos para ser los posibles elementos que puedan ser utilizados por los administradores para crear los trabajos. Estos son:

- Entrada numérica.
- Entrada normal.
- Opciones únicas.
- Elegibles.
- Desplegables.

Descripción de cada elemento

A continuación vamos a describir uno a uno los elementos antes mencionados, definiendo sus funciones para crear la plantilla de un trabajo.

Entrada numérica

Este elemento es el encargado de las entradas de los datos que se proporcionan con el teclado numérico solamente. Los datos asociados al elemento sólo puede ser uno, el cual va indicar la descripción del campo que estamos introduciendo. Por defecto este campo va a tener un 1 que puede ser cambiado por el cliente.

Entrada normal

Este elemento es para datos alfanuméricos. Sólo puede haber un dato asociado, el cual va indicar la descripción del campo que estamos introduciendo. Por defecto este campo va a estar vacío pero puede ser cambiado por el cliente.

Opciones únicas

Este elemento es el encargado de las entradas de los datos que tienen varias opciones para elegir una solamente. Los datos asociados al elemento sólo pueden ser tres como mínimo o un máximo de seis, el primero indica la descripción del grupo y los demás la descripción de cada una de las opciones. Por defecto el primer campo es el que se selecciona por defecto.

Elegibles

Este elemento es el encargado de las entradas de los datos que pueden ser seleccionados o no. Solo habrá un dato asociado al elemento, el cual va indicar la descripción del campo que estamos introduciendo. Por defecto este campo no se encuentra seleccionado, pero puede ser cambiado por el cliente.

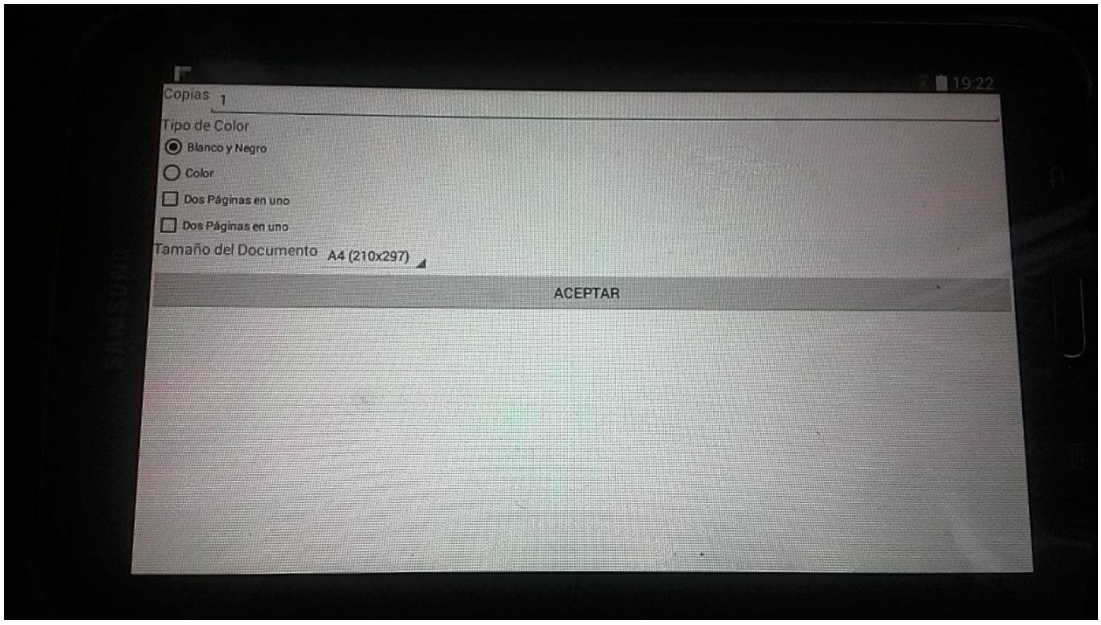
Desplegables

Este elemento es el encargado de las entradas de los datos que se eligen entre varias en un desplegable. Los datos asociados al elemento pueden ser tres como mínimo o seis como máximo, el primero indica la descripción del grupo y los demás la descripción de cada una de las opciones. Por defecto este campo va a tener seleccionado el primer elemento de la lista desplegable.

Ejemplo

Una vez explicados los distintos elementos que conforman las plantillas, estas pueden tener varios elementos del mismo, sin ser obligatorio que se encuentren todos en la misma plantilla e incluso trabajos pueden no tener ningún elemento.

A continuación mostraremos una plantilla en la que se puede observar cada uno de los elementos antes descritos en orden correlativo:



Capítulo 6 Aspectos principales de interfaz con el sistema

6.1. Introducción

En este capítulo vamos a explicar aspectos de la interfaz gráfica, y los aspectos importantes de la aplicación, exclusivamente los temas con dificultad, la persistencia de los datos y por último las bibliotecas especiales usadas para complementar las funcionalidades internas de Android.

6.2. Aspectos interfaz gráfica

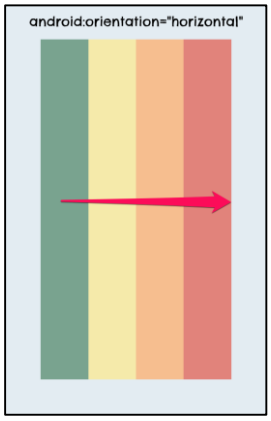
Layouts

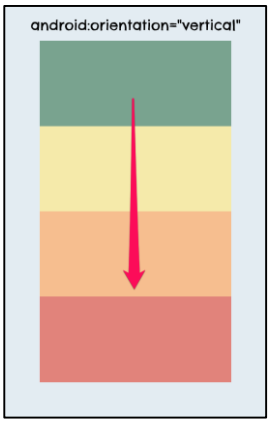
Un Layout es un elemento que representa el diseño de la interfaz de usuario de componentes gráficos como una actividad, fragmento o widget. [6]

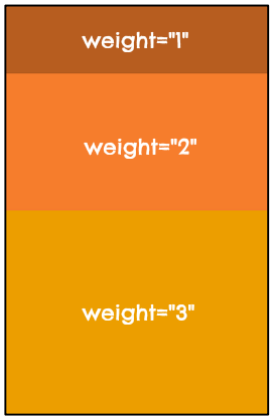
Ellos se encargan de actuar como contenedores de views (componente que permite controlar la interacción) para establecer un orden visual, que facilite la comunicación del usuario con la interfaz de la aplicación.

Vamos a explicar los Layouts utilizados en las activity (cada una de las pantallas que forman una aplicación) de la App que son los siguientes:

Un **LinearLayout** es un view group que distribuye sus hijos en una sola dimensión establecida. Es decir, o todos organizados en una sola columna (vertical) o en una sola fila (horizontal).



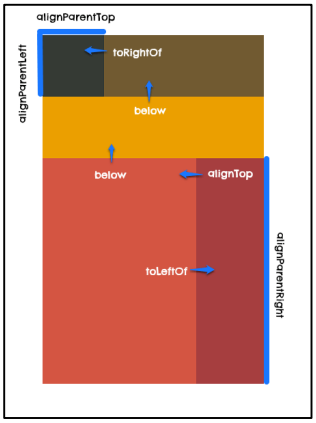




Este Layout es utilizado en nuestra aplicación sobre todo para formar partes de algunos RelativeLayout donde los componentes van seguidos en orientación horizontal. Otro de los casos es cuando los componentes van ocupando una parcela de la pantalla en orientación vertical.

El **RelativeLayout** permite alinear cada hijo con referencias cualitativas de cada hermano. Con el pensaremos en como alinear los bordes de cada view

con otros. Es equivalente a frases como “el botón estará por debajo del texto” o “la imagen se encuentra a la derecha de la descripción”. En ninguno de los casos nos referimos a una posición absoluta o un espacio determinado. Simplemente describimos la ubicación y el framework de Android computará el resultado final.

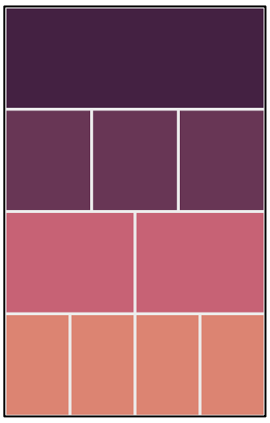


Este Layout es utilizado en la mayoría de las activity que se utiliza en la aplicación debido que podemos integrar los objetos junto al elemento y la posición que se desee. Esto nos vale para que las vistas se puedan adaptar a las pantallas respecto a su tamaño.

El **TableLayout** organiza views en filas y columnas de forma tabular. Para crear las filas se usa el componente TableRow dentro del TableLayout. Cada celda es declarada como un view de cualquier tipo (imagen, texto, otro group view, etc.) dentro de la fila. Sin embargo, puedes crear una celda con otro tipo de view. Esto hará que todo el espacio de la fila sea ocupado por el objeto.

El TableRow trae consigo un parámetro llamado android:layout_column para asignar la columna a la que pertenece cada celda en su interior. Incluso puedes usar el parámetro weight para declarar pesos de las celdas.

El ancho de cada columna es definido tomando como referencia la celda más ancha.



Este Layout se usa únicamente en las pantallas del menú de trabajos de manera que cada celda pueda representar cada uno de los trabajos y a la

hora de visualizar las opciones monetarias. En el caso de los trabajos tiene la peculiaridad de que se crea programáticamente, ya que no se sabe el número de trabajos exactos que posee la aplicación en cada momento.

Elementos

Los Elementos o Views son las partes que conforman un Layout o ViewGroup para conseguir las pantallas que deseamos. Hay una gran variedad de Views para usar, a continuación explicaremos los que se utilizan en la aplicación.

TextView, permite visualizar texto.

ImageView, permite visualizar una imagen.

EditText, es la expansión de un TextView con la capacidad de editar su contenido para recibir texto por parte del usuario.

Button e ImageButton, es un control con texto o imagen que realiza una acción cuando el usuario lo presiona.

Checkbox, es un botón de dos estados (marcado, no marcado) que actúa como control de selección ante los usuarios. Lo que permite elegir una o varias opciones de un conjunto.

RadioGroup, es un conjunto de RadioButton, el cuál es uno control de selección que proporciona la interfaz de Android para permitir a los usuarios seleccionar una opción de un conjunto. Son ideales para elegir uno de varios elementos con exclusión mutua, es decir, la selección de un radio Button obliga a descartar la de otro, permitiendo solo a un ítem estar activo.

Spinner, son menús desplegables en Android que permiten al usuario decidirse por un camino entre varias opciones.

GridView, es un view que contiene otros views en forma de grilla o cuadricula, cuya orientación puede ser vertical u horizontal.

Este elemento es utilizado para mostrar las fotos que se usan en la aplicación como una galería y cada uno de los elementos es un Layout creado para ello.

ListView, son contenedores supremamente útiles para organizar información en forma vertical y con la capacidad de usar scrolling (desplazamiento) para simplificar su visualización.

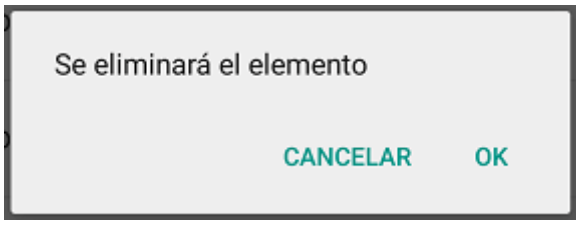
Cuando se muestran en la parte de administración los trabajos y los elementos asociados a cada trabajo se usa este elemento. Al igual que en el elemento anterior en cada una de las listas sus elementos son definidos en un Layout.

Otros

Diálogos, son elementos en la interfaz de Android que se superponen a las actividades para exigir la confirmación de una acción o el ingreso de datos.

Se caracterizan por interrumpir una tarea del usuario, alterando el foco de la aplicación e invitándolo de forma intrusiva a que vea información, decida ante una circunstancia crítica o especifique datos.

Por ejemplo, se usa para recordar la importancia de borrar un elemento de la interfaz antes de continuar con la tarea, como se puede observar en la siguiente figura.



En la aplicación se usan diálogos de alerta como el mostrado en la figura anterior. Estos diálogos son aquellos que están diseñados para mostrar un título, un mensaje o cuerpo y hasta 3 botones de confirmación en su zona de acción. Aparecen a la hora de intentar borrar un trabajo o elemento y después de rellenar un formulario de un trabajo.

6.3. Aspectos Importantes

AndroidManifest

- **Dar permisos a la aplicación**

La aplicación necesita de los siguientes permisos.

- Escribir en el directorio de la aplicación: Se necesita este permiso para almacenar en una carpeta las fotografías que se usan en la aplicación que son guardadas por los administradores.

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Los permisos se muestran al usuario a la hora de instalar la aplicación y decide si aceptar e instalar la aplicación o no.

- **Orientación de las activitys**

- Establecer todas las pantallas horizontales: Se estableció que las pantallas tenían orientación horizontal y no rotaban en el dispositivo con el movimiento.

```
android:screenOrientation="landscape"
```

- **Aplicar temas**

Un tema (theme) es un estilo (style) pero en lugar de aplicarse a un objeto View se aplica a toda la aplicación por defecto. En el fichero AndroidManifest se puede definir el tema que llevará la aplicación como un atributo de la etiqueta <application>. Este tema puede ser uno de los existentes en Android como el @android:style/Theme.Light.NoTitleBar que dota a la aplicación de un interfaz de usuario clara (pantalla en blanco principalmente y letras oscuras) y sin la barra del título de la aplicación, o el tema android:Theme.Black que dota a la aplicación de una interfaz oscura (pantalla en negro principalmente y letras claras) y con la barra del título de la aplicación. Además de los múltiples temas aplicables disponibles en Android, se pueden definir temas propios en el fichero de estilos, creándolos bien desde cero o bien partiendo de alguno de los existentes (tanto de Android como de otros creados por el desarrollador).

En el caso de esta aplicación se han creado dos temas propios para la aplicación utilizando como base los siguientes `Theme.AppCompat.Light.DarkActionBar` y `AppTheme.NoActionBar` que proveen las librerías android, es decir, tema claro con la barra oscura definiendo nosotros los colores y el otro sin barra de acción respectivamente. Para mantener el mismo tipo de botones, ediciones de texto y demás elementos gráficos que, si no, variarían.

- **Guardar en la tarjeta de memoria externa SD la aplicación**

Aunque la aplicación ocupa algo menos de 13 MB, se ha considerado que era importante dejar al usuario que pudiera almacenar la aplicación en la tarjeta de memoria externa SD. Hay que tener en cuenta si la aplicación necesita estar siempre activa como es el caso de aplicaciones de mensajería tipo Hangouts o Whatsapp o bien si no es una problemática mayor el que no funcione si se extrae la tarjeta SD o se desmonta la tarjeta SD como ocurre cuando se conecta el dispositivo móvil al ordenador para intercambio de datos. Para permitir al usuario esta opción se define el atributo android:installLocation de la etiqueta <manifest> a "auto". Otros valores son “preferExternal” e “internalOnly”. Este último es el que se utiliza por defecto si no se le define uno en concreto para la aplicación en el fichero AndroidManifest.

- **Definir el icono de la aplicación**

Se ha definido un icono propio para mostrar en la lista de las aplicaciones disponibles en el dispositivo móvil. Dicho icono se guarda en las carpetas de recursos alternativos de mipmap como las otras imágenes con sus distintos tamaños, y se referencia en el atributo android:icon dentro de la etiqueta <application> del fichero AndroidManifest.

Aspectos relativos al código

Una gran parte de las pantallas que se usan a lo largo de la aplicación son comunes y usan aspectos de códigos sencillos. Normalmente se controla las pulsaciones sobre los botones que van pasando de una pantalla a otra y se recogen algunos datos de los elementos gráficos de la pantalla. Relativo a estos casos sólo vamos a entrar en detalle en cómo interactúan las pantallas entre sí.

- **Comunicación entre actividades**

La comunicación entre actividades se realiza a través de objetos Intents. Existen dos tipos de intents: implícitos y explícitos. Los intents implícitos son aquellos en los que se busca una funcionalidad y no una actividad de una aplicación en concreto. Por ejemplo cuando se desea abrir una página web en concreto se utiliza un intent para “abrir un navegador” pero no se especifica qué navegador en concreto. El sistema será el encargado de utilizar el navegador por defecto o mostrar una lista al usuario para que elija el navegador que quiere utilizar, en caso de tener varios instalados en el dispositivo móvil. Los intents explícitos son aquellos en los que se llama a una actividad en concreto, generalmente de la propia aplicación para pasar a otra pantalla de la aplicación.

En esta aplicación se ha hecho uso de intents tanto implícitos como explícitos. En el caso de los implícitos se ha utilizado por ejemplo a la hora de seleccionar una imagen del dispositivo para utilizarla en la aplicación, guardando ésta en la carpeta habilitada para ello. Los intents explícitos se han utilizado a la hora de pasar de una actividad a otra de la propia aplicación.

En algunos casos se necesita enviar información de una actividad a otra para realizar su función correctamente. Esta información se envía añadiendo unos extras al intent. En el caso de la galería del teléfono se añadía la dirección donde se encuentran las imágenes que se quieren ver. Cuando se trata de información entre actividades de la misma aplicación, se pueden añadir además algunos tipos básicos de datos como strings (cadenas de caracteres), int (enteros), list (listas), etc. en formato clave-valor para recuperarlos en la actividad a la que se le envían. En el caso de necesitar enviar datos más complejos como pueden ser objetos (instancias) de alguna clase creada por el desarrollador en la aplicación, se pueden crear clases serializables o parcelables cuyos objetos se pueden enviar como un extra en el intent.

Los datos que se envían entre activitys siguen el patrón de una clave y un valor para poder recogerlos en la siguiente pantalla que se muestra y utilizar estos datos en el código.

Ejemplo en la siguiente figura:

```
@Override
public void onClick(View v) {

    switch (v.getId()) {
        case R.id.newimage: {
            Intent galleryIntent = new Intent(Intent.ACTION_PICK, android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
            startActivityForResult(galleryIntent, ACTIVITY_SELECT_IMAGE);
            break;
        }
        case R.id.seeimages: {
            Intent i = new Intent(this, GridViewActivity.class);
            startActivity(i);
            break;
        }
        default: {
            break;
        }
    }
}
```

- **Tabla de Trabajos**

La tabla creada nos muestra un botón por cada uno de los trabajos que tenemos creados en la aplicación por los administradores, teniendo en cuenta que el máximo de trabajos que mostraremos en una fila es de tres.

```
private void drawScreen()
{

    TableRow tableRow;
    ArrayList<Works> mStrings;
    mStrings=GetJobs();
    tableRow = null;
    for(int i=0;i<mStrings.size();i++){
        if ((i%3)==0){
            tableRow = Rowadd();
            Log.i(LOGTAG, "Crear fila");
        }
        Buttonadd(tableRow,mStrings.get(i).titulo);
    }
    Log.i(LOGTAG, "" + myTableLayout.getChildCount());
    setContentView(myTableLayout);
}
```

En el código podemos observar cómo mediante la función GetJobs obtenemos de la base de datos los trabajos existentes, cada tres trabajos como podemos observar se añade una fila nueva a la tabla y en cualquier caso se añade un botón por cada uno de los trabajos. Estos botones toman su texto del array obtenido anteriormente. Cada botón añadido se implementa para que responda al evento cuando se realiza click.

```
private void Buttonadd(TableRow tr,String titulo){
    Button buttonnew=new Button(this);
    buttonnew.setText(titulo);
    TableRow.LayoutParams pButton = new TableRow.LayoutParams(
            TableRow.LayoutParams.MATCH_PARENT,
            TableRow.LayoutParams.WRAP_CONTENT);
    pButton.weight=1;
    //buttonnew.setOnClickListener(this);
    Button.OnClickListener listener;
    listener = (OnClickListener) (v) -> { go(v); };

    buttonnew.setOnClickListener(listener);
    tr.addView(buttonnew, pButton);
}
```

- **Listas de la aplicación**

La aplicación en su conjunto cuenta con dos listas, las cuales muestran los trabajos y los elementos de un trabajo. Las listas son contenedores supremamente útiles para organizar información en forma vertical y con la capacidad de usar scrolling (desplazamiento) para simplificar su visualización. Este elemento permite mostrarle al usuario un conjunto de datos de forma práctica y accesible.

Esta clase que representa una lista vertical en la API de Android se llama ListView. Esta clase viene preparada para recibir los ítems que desplegará en la interfaz, facilitando al programador la implementación de sus características y comportamientos.

Estructuralmente un ListView contiene un View específico por cada fila. También se compone de un ScrollView, el cual permite generar el desplazamiento vertical por si se agota la pantalla para nuestros elementos.

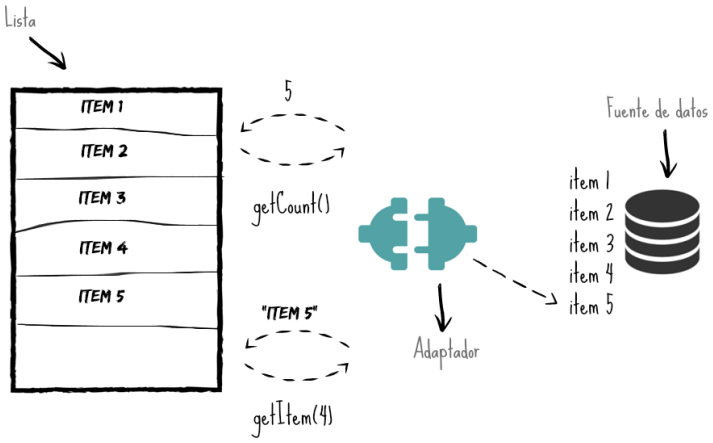
Para poder poblar con datos necesitamos usar otro objeto, el adaptador, que comunica a un ListView los datos necesarios para crear las filas de la lista. Es decir, conecta la lista con una fuente de información como si se tratase de un adaptador de corriente que alimenta un televisor. Además de proveer la información, también genera las Views para cada elemento de la lista.

Cuando relacionas un adaptador a una lista, inmediatamente comienza un proceso de comunicación interno para poblarla con una fuente de datos. Dicha comunicación se basa principalmente en los siguientes métodos del adaptador:

- **getCount ()**: Retorna la cantidad de elementos de la fuente de datos. Con este valor la lista ya puede establecer un límite para añadir items.
- **getItem ():** Obtiene un elemento de la fuente de datos asignada al adaptador en una posición establecida. Normalmente la fuente de datos es una lista de objetos o un Cursor.
- **getView ():** Retorna en el View inflado y ligado a los datos según su posición.

Aunque estos tres métodos no son los únicos que existen para establecer la relación, a mi parecer son los más significativos para entender el concepto de un adaptador.

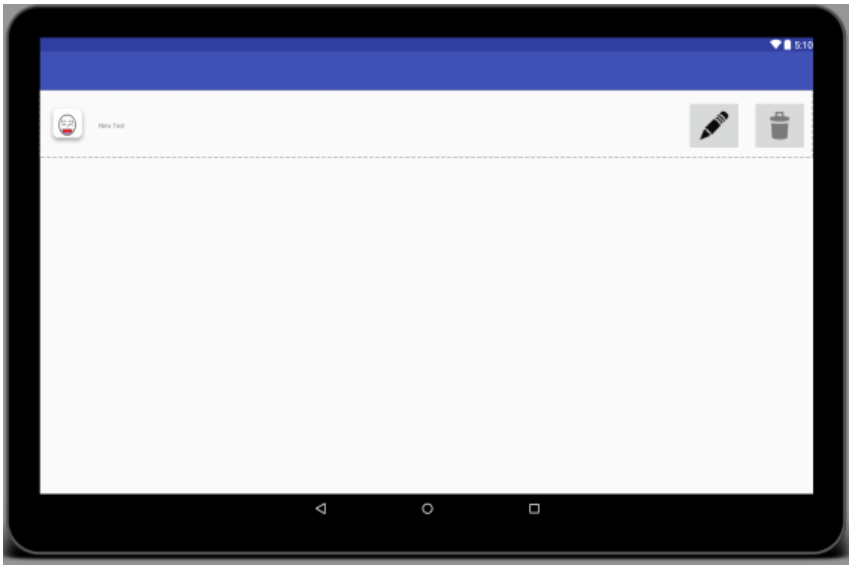
La siguiente figura muestra cómo sería esta relación:



Una vez explicado los conceptos más importantes sobre las listas se sigue con la explicación específica de las usadas en la aplicación. Las imágenes mostradas harán referencia a la lista de los trabajos teniendo en cuenta que son similares, sólo cambian en dos aspectos.

El primer paso que se dio fue la creación de sus propias clases, que veremos con más detenimiento en el siguiente capítulo, estas clases son Works y ElementsWorks. Estas clases nos facilitarán los datos que tenemos que manejar en las listas y si necesitamos realizar alguna de las acciones.

El segundo paso es crear los layouts específicos para mostrar la información deseada en cada una de las filas de la lista, works_info.xml y works_elements_info.xml, que posteriormente es usado en la lista dando tantas filas como contenga, en la siguiente figura podemos observar cómo son cada una de las filas.



Una vez que tenemos definido el layout de cada fila, se crea el adaptador que conecta los datos con la lista. Ahora con un adaptador de los de Android no se puede rellenar los datos de las listas, lo que nos lleva a crear adaptadores personalizados. Así que la aplicación cuenta con dos adaptadores, WorksAdapter.java y ElementsWorksAdapter.java. Se extienden de la clase BaseAdapter y se sobrescriben algunos métodos. En la siguiente figura podemos ver el principal método del adaptador, donde se establecen los datos en el layout creado anteriormente.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {

    //Obtenemos la vista de la fila.
    View v = this.inflater.inflate(R.layout.works_info, null);
    //Obtenemos todos los datos necesarios
    String titulo=this.dataSource.get(position).titulo;
    String icono=this.dataSource.get(position).icono;
    Double precio = this.dataSource.get(position).precio;
    Integer _id = this.dataSource.get(position).id;
    //Insertamos los datos en la fila
    ImageView imagen=(ImageView) v.findViewById(R.id.icono);
    ContextWrapper cw = new ContextWrapper(context);
    File directory = cw.getDir("imageDir", Context.MODE_PRIVATE);
    File mypath = new File(directory,icono);
    Glide.with(imagen.getContext())
            .load(mypath.getPath())
            .override(100,100)
            .into(imagen);
    ((TextView) v.findViewById(R.id.titulo)).setText(titulo);
    ((TextView) v.findViewById(R.id.txticono)).setText(icono);
    ((TextView)v.findViewById(R.id.precio)).setText(String.format("%f", precio));
    ((TextView)v.findViewById(R.id._id)).setText(String.format("%d",_id));
    //Activamos el evento click en los botones
    v.findViewById(R.id.imageButton).setOnClickListener((OnClickListener) this.context);
    v.findViewById(R.id.imageButton2).setOnClickListener((OnClickListener) this.context);
    return v;
}}
```

Por último, utilizando el archivo creado con la Activity vamos a poblar las listas con los elementos antes mostrados, el layout especifico de cada fila y el adaptador para conectar los datos. Los archivos de este apartado son List_View_Works.java y List_View_Works_Elements.java. Se manda al sistema el layout que contiene la lista a la vez que se rellena la lista.

En la función donde se asigna el adaptador, como se ve en la siguiente figura, como en otras funciones de esta aplicación se tratan las excepciones que pueden suceder en la plataforma Android. Se trata con los bloques try y catch; este último recogería la excepción y no sería interrumpida la aplicación de una forma no deseable. En cambio, se mostrará un dialogo al usuario informándole que ha ocurrido un problema.

En el código también se puede observar cómo se le añade una cabecera a la lista con el siguiente texto "Lista de Trabajos". Se incluye el encabezamiento ya que en esta Activity no se muestra la barra de acción.

```
private void displayListView(){
    try {
        WorksAdapter adapter;
        ArrayList<Works> trabajos = this.creardatos();
        adapter = new WorksAdapter(this, trabajos);
        ListView lv = (ListView) findViewById(R.id.ListView1);
        lv.setAdapter(adapter);
        lv.setOnItemClickListener(this);
        //.........Creating a Header for ListView.........
        //Initialize a new TextView instance
        TextView tv = new TextView(this);
        //Set a text for TextView widget
        tv.setText("Lista de Trabajos");
        //Apply TextView text color to MediumSlateBlue
        tv.setTextColor(Color.BLACK);
        tv.setGravity(Gravity.CENTER_HORIZONTAL);
        tv.setPadding(15, 15, 15, 15);
        //Set the TextView as ListView Header and it is non selectable
        lv.addHeaderView(tv, "", false);
    }
    catch(Exception e){
        AlertDialog.Builder builder =
                new AlertDialog.Builder(this);

        builder.setMessage("Esto es un mensaje de alerta.")
                .setTitle("Información")
                .setPositiveButton("OK", (dialog, id) -> {
                        dialog.cancel();
                });
    }

}
```

```
private ArrayList<Works> creardatos(){
    try {
        ArrayList<Works> list = new ArrayList<>();
        // Columnas de la tabla trabajos
        String KEY_ID = "_id";
        String KEY_COL1 = "titulo";
        String KEY_COL2 = "icono";
        String KEY_COL3 = "precio";
        //Array de strings para su uso en los diferentes métodos
        String[] cols = new String[]{KEY_ID, KEY_COL1, KEY_COL2, KEY_COL3};

        Cursor cursor = dbadapter.allTrabajos(cols);
        if (cursor.moveToFirst()) {
            do {
                Works libro = new Works(cursor.getInt(0), cursor.getString(1), cursor.getString(2),
                        cursor.getDouble(3));
                list.add(libro);
            } while (cursor.moveToNext());
        }
        if (!cursor.isClosed()) {
            cursor.close();
        }
        return list;
    }
    catch (Exception e){
        AlertDialog.Builder builder =
                new AlertDialog.Builder(this);

        builder.setMessage("Esto es un mensaje de alerta.")
                .setTitle("Información")
                .setPositiveButton("OK", (dialog, id) -> {
                        dialog.cancel();
                });
        return null;
    }
}
```

- **Creación de la plantilla**

La Activity de Layout_Trabajo es la parte más complicada de la aplicación, ya que cuando se diseña nunca se sabe el contenido que va a tener. Los datos que se van a usar están guardados en la base de datos al crear los administradores las distintas plantillas de recogida de los trabajos.

Hay que tener en cuenta que se crea los trabajos que tienen asociados a su vez elementos y cada elemento también tiene asociados datos. No todos tienen los mismos datos, está explicado en el apartado **4.5 Elementos**. Se creó un layout más por si alguno de los trabajos no tiene elementos.

Sabiendo esto se crean los layouts que van a ir llenando el layout de está Activity. Son los siguientes:

- Entradanumerica.xml
- Entradanormal.xml
- Opciones únicas.xml
- Elegibles.xml
- Desplegables.xml
- Sinelementos.xml

En la siguiente figura se puede observar el código del layout de uno de estos elementos que se va a usar en está Activity y donde se puede observar cómo hay algún View oculto que vamos a necesitar posteriormente.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/rlelegibles">
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/elegiblescb"
        android:checked="false" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:id="@+id/rlelegiblesicono"
        android:visibility="gone"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:id="@+id/rlelegiblesprecio"
        android:visibility="gone"/>
</RelativeLayout>
```

En los dos últimos Views se puede observar como la visibilidad del mismo está en gone; lo que hace es que se encuentre en el layout pero no se ve.

En la parte que se crea la Activity se crean programáticamente los Layouts tanto el principal (RelativeLayout) como la primera fila de este (LinearLayout), además al principal se le añade un ScrollView debido a no conocer cuántos elementos le componen. Mediante el intent se obtiene el trabajo del que se va mostrar su plantilla.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    RelativeLayout principallayout=new RelativeLayout(this);
    layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);
    ScrollView sv=new ScrollView(this);
    sv.addView(layout);
    Intent intent = getIntent();
    workpulse = intent.getStringExtra("trabajo");
    this.setTitle(workpulse);
    setContentView(principallayout);
    principallayout.addView(sv);
    dbadapter = new DbControl(this);
    obtener_idworkread();
    Cargar_Archivos();
    AnadirBoton();
}
```

Lo siguiente es obtener los elementos que conforman está plantilla accediendo a la base de datos, para ir conformando la plantilla obteniendo el elemento que hay que ir añadiendo.

```
private void Cargar_Archivos() {
    ArrayList<ElementsWorks> elementos = new ArrayList<>();
    dbadapter.open();
    Cursor cursor = dbadapter.TrabajoporTitulo(workpulse);
    if (cursor.moveToFirst()) {
        do {
            ElementsWorks elemento = new ElementsWorks(cursor.getInt(0),
                    cursor.getInt(2), cursor.getInt(1), "", cursor.getInt(3), 1, "");

            elementos.add(elemento);
        } while (cursor.moveToNext());
    } else {
        cursor.close();
    }
    for (int i = 0; i < elementos.size(); i++) {
        switch (elementos.get(i).Elemento) {
            case 1:
                Anadir_EntradaNumerica(i,elementos.get(i));
                break;
            case 2:
                Anadir_EntradaNormal(i,elementos.get(i));
                break;
            case 4:
                Anadir_Elegibles(i,elementos.get(i));
                break;
            case 5:
                Anadir_Despegables(i,elementos.get(i));
                break;
            case 3:
                Anadir_OpcionesUnicas(i,elementos.get(i));
                break;
        }
    }
    dbadapter.close();
}
```

Una vez obtenidos todos los elementos se van recorriendo todos y añadiendo este según el tipo de elemento que sea. Esto lo que hace es obtener los datos

pertenecientes a ese elemento desde la base de datos, se le añade en el layout creado para ese elemento y por último se añade éste en el layout principal. En la siguiente figura podemos observar cómo se le añaden los datos a la View (Spinner) los datos mediante un adapter de Android (ArrayAdapter).

```
private void Anadir_Despegables(Integer i,ElementsWorks Elemento){
    Cursor cursordatos;
    View inflatedLayout = getLayoutInflater().inflate(R.layout.desplegables, null, true);
    layout.addView(inflatedLayout);
    View vista3 = layout.getChildAt(i);
    cursordatos = dbadapter.obtenerdatos(Elemento._Id.toString(), Elemento.Trabajo.toString());
    cursordatos.moveToFirst();
    TextView texto3 = (TextView) vista3.findViewById(R.id.textView3);
    texto3.setText(cursordatos.getString(0));

    Integer y = 0;
    Integer[] idicono = {R.id.rldespegablesicono,R.id.rldespegablesicono2,R.id.rldespegablesicono3,R.id.rldespegab
    Integer[] idprecio = {R.id.rldespegablesprecio,R.id.rldespegablesprecio2,R.id.rldespegablesprecio3,R.id.rldesp
    while (cursordatos.moveToNext()) {
        texto3 = (TextView) vista3.findViewById(idicono[y]);
        texto3.setText(cursordatos.getString(1));
        texto3 = (TextView) vista3.findViewById(idprecio[y]);
        texto3.setText(String.format("%f", cursordatos.getFloat(2)));
        y++;
    }
    Spinner spiner = (Spinner) findViewById(R.id.spinner);
    List<String> datos = dbadapter.obtenerdatosspinner(Elemento._Id.toString(), Elemento.Trabajo.toString());
    ArrayAdapter<String> adapter2 = new ArrayAdapter<>(this,
            android.R.layout.simple_spinner_item, datos);
    adapter2.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spiner.setAdapter(adapter2);
}
```

Tras haber añadido todos los elementos que forman la plantilla del trabajo, queda terminar de rellenar el Layout, por lo tanto se inserta un botón aceptar para enviar las opciones elegidas por el cliente.

Se usa el layout principal para comprobar que tiene elementos; en caso de que no tenga ninguno, se añade el layout que hemos explicado antes que se creaba exclusivamente para este caso. Al botón se le implementa el evento al que tiene que responder en caso de hacer click sobre él.

Al recoger el evento se va leyendo los hijos del Layout principal y, dependiendo del tipo de elemento que sea, se lee de una forma concreta cada elemento y los elementos leídos se van guardando en la base de datos para tener la información posteriormente.

```
private void AnadirBoton() {
    View inflatedLayout;
    if (layout.getChildCount() == 0) {
        inflatedLayout = getLayoutInflater().inflate(R.layout.sinelementos, null, true);
        layout.addView(inflatedLayout);
    }
    inflatedLayout = getLayoutInflater().inflate(R.layout.botonlayout, null, true);
    layout.addView(inflatedLayout);
    Button boton = (Button) findViewById(R.id.button9);
    boton.setOnClickListener((view) -> {
            Integer datain = 0;
            ArrayList<ElementsWorksRead> elementosleidos = new ArrayList<>();
            //ElementsWorksRead elemento=new ElementsWorksRead(0,0,"","",0);
            for (int i = 0; i < layout.getChildCount(); i++) {
                View vista = layout.getChildAt(i);
                Integer id = vista.getId();
                String dato;
                switch (id) {
```

Una vez leídos todos los elementos se muestra un diálogo al cliente para que elija si quiere realizar algún trabajo más o no. Mientras se sigan eligiendo trabajos se utiliza un SharedPreferences (Archivo de preferencias) para saber en qué trabajo nos encontramos (Esto lo explicaremos en el siguiente apartado del capítulo).

- **Uso de AsyncTask**

Como su propio nombre indica, la clase AsyncTask permite realizar una tarea de forma asíncrona. Es una forma sencilla de crear y ejecutar un thread secundario y, a continuación, mostrar el resultado en el thread principal. Se usa para quitar trabajo al hilo principal ya que si está más tiempo del deseado por Android, el usuario puede detener el proceso.

```
public class ComprobarUsoImagen extends AsyncTask<String, Void, Boolean>{

    private Context context;
    private String imagen;

    public ComprobarUsoImagen(Context context,String imagen) {
        this.context = context;
        this.imagen=imagen;
    }

    protected Boolean doInBackground(String... params) {
        Boolean eliminar=false;
        Integer coincidencias;
        DbControl bbdd=new DbControl(context);
        bbdd.open();
        coincidencias=bbdd.EstaImagen(imagen);
        bbdd.close();
        if (coincidencias!=0){
            eliminar=true;
        }
        return eliminar;
    }
}
```

Persistencia de los datos

Los datos persistentes de una aplicación son los datos salvaguardados antes del cierre de la aplicación de tal forma que puedan ser restaurados posteriormente. Android proporciona varios mecanismos que permiten gestionar la persistencia de los datos, en función de la naturaleza de los mismos. Existen varias formas de salvaguardar los datos pero en nuestra aplicación hemos decidido usar los archivos de preferencia y las bases de datos.

En un primer momento se había pensado en usar archivos XML, y se creó un parseador, pero no cubría los requisitos que necesitábamos. El acceso a un dato concreto y cambiar la información en un determinado lugar del archivo complicaba mucho las tareas a desarrollar. Habría que reescribir el archivo nuevamente o recorrer el archivo hasta encontrar el elemento.

Por todos estos contratiempos encontrados a la hora de usar los archivos XML, hizo que nos decidimos por usar las bases de datos.

- **Archivos de preferencias**

Android proporciona un framework simple para salvaguardar y restaurar los datos de tipos primitivos. Estos datos se salvaguardan en archivos con formato XML bajo la forma de pares clave-valor. [7]

La creación y la gestión del archivo preferencias se realiza a través de un objeto de tipo SharedPreferences. El modo de acceso que se le asigna a un archivo en su creación puede ser privado (MODE_PRIVATE), accesible para las demás aplicaciones (MODE_WORLD_READABLE) y también se puede definir el nombre del archivo. Si no le definimos, obtendrá automáticamente el nombre de la Activity.

Este tipo de archivo tiene varios métodos para implementar: lectura, escritura y borrado. Para leer los datos contenidos en el archivo se escoge un método según el tipo de dato, el primer parámetro es la clave y el segundo el valor que devuelve si no existe la clave.

La escritura permite especificar nuevos datos o modificar los existentes borrándolos de nuevo. De forma similar a los métodos de lectura, existe un método por cada dato primitivo. Estos reciben el nombre de la clave y el valor del dato. Para confirmar los cambios será necesarios hacerles efectivos con el método commit.

El borrado utiliza el método remove permite suprimir un par clave-valor, se pasa el valor de la clave como parámetro. Como ocurre con la escritura, es necesario invocar el método commit para registrar los cambios.

Los archivos de preferencias se usan en nuestra aplicación para dos funciones de salvaguardar los datos. En primer lugar cuando se están eligiendo los trabajos por parte del cliente. Se usa el archivo para ir sabiendo cuantos trabajos van solicitando. De esta forma vamos obteniendo el id del trabajo que vamos a relacionar con los elementos elegidos por el cliente.

También se usa para tener guardado los datos de la caja y saber cuántas monedas tenemos en cada momento, vaciar la caja, cambiar las monedas que vamos dando en las vueltas y añadir con las que se paga. Así siempre se puede saber el dinero que hay en cada momento en la caja por parte de los administradores.

Para estos archivos es muy interesante salvaguardar los datos así, ya que con el par clave-valor es muy fácil acceder a ellos. Les podemos manipular de una forma sencilla y si es necesario podemos eliminar todas las claves.

En la siguiente figura se ve cómo uso este tipo de archivos en la aplicación, exactamente en la parte de la caja.

```
private void Cargarmonedero(){
    Boolean existe = preferencias.getBoolean("Existe", false);
    String[] valores = new String[]{"uncent", "doscent", "cincocent", "diezcent",
            "veintecent", "cincuentacent", "uneuro", "doseuros", "cincoeuros", "diezeuros",
            "veinteeuros","cincunetaeuros"};
    if (existe) {
        for (int i = 0; i < 12; i++) {
            Integer cantidad=preferencias.getInt(valores[i], 0);
            double multiplicador=obtenervalor(i);
            caja[i] = cantidad;
            total=total+cantidad*multiplicador;
        }

    } else {
        SharedPreferences.Editor editor = preferencias.edit();
        for (int i = 0; i < 12; i++) {
            caja[i]=0;
            editor.putInt(valores[i], 0);
        }
        editor.putBoolean("Existe", true);
        editor.apply();
        total=0;
    }
    rellenartxt();
}
```

- **Base de datos**

La aplicación necesitaba una base de datos para tener permanentemente los datos necesarios para configurar las plantillas de los trabajos y poder almacenar y realizar consultas sobre ellos. Android permite crear bases de datos con formato SQlite. La base de datos es privada de la aplicación y solamente ella tendrá acceso. [8]

Sqlite es un motor ligero de bases de datos de código abierto, que se caracteriza por mantener el almacenamiento de información persistente de forma sencilla.

A diferencia de otros Sistemas gestores de bases de datos como MySQL, SQL Server y Oracle DB, SQlite tiene las siguientes ventajas:

- No requiere el soporte de un servidor: SQlite no ejecuta un proceso para administrar la información, si no que implementa un conjunto de librerías encargadas de la gestión.
- No necesita configuración: Libera al programador de todo tipo de configuraciones de puertos, tamaños, ubicaciones, etc.
- Usa un archivo para el esquema: Crea un archivo para el esquema completo de una base de datos, lo que permite ahorrarse preocupaciones de seguridad, ya que los datos de las aplicaciones Android no pueden ser accedidos por contextos externos.
- Es de Código Abierto: Esta disponible al dominio público de los desarrolladores al igual que sus archivos de compilación e instrucciones de escalabilidad.

Es por eso que Sqlite es una tecnología cómoda para los dispositivos móviles. Su simplicidad, rapidez y usabilidad permiten un desarrollo muy amigable.

La forma en que una base de datos está estructurada (cantidad de tablas, registros, índices, etc.) y el conjunto de convenciones para nombrar sus objetos recibe el nombre de Esquema. Por lo general el esquema inicial se guarda en un Script que nos permita recuperar las condiciones previas en cualquier momento.

Con SQLite no es diferente, por lo que debes crear un esquema predefinido para implementarlo a la hora de crear tu base de datos.

El Android SDK nos provee una serie de clases para administrar nuestro archivo de base de datos en Sqlite.

Normalmente cuando conectamos otro gestor de bases de datos tenemos que validar los *datos del equipo*, *el usuario* y el *esquema*, pero con SQLite no se requiere nada de eso, ya que podemos trabajar directamente sobre la base de datos.

La clase que nos permitirá comunicar nuestra aplicación con la base de datos se llama SQLiteOpenHelper. Se trata de una clase abstracta que nos provee los mecanismos básicos para la relación entre la aplicación Android y la información.

Lo podemos observar en la siguiente figura:

```
public class DbControl extends SQLiteOpenHelper {

    //Ruta por defecto de las bases de datos en el sistema Android
    private static String DB_PATH = "/data/data/com.example.roberto.editar/databases/";

    private static String DB_NAME = "works.db";

    private SQLiteDatabase myDataBase;

    private final Context myContext;

    /**
     * Constructor
     * Toma referencia hacia el contexto de la aplicación que lo invoca para poder acceder a l
     * Crea un objeto DBOpenHelper que nos permitirá controlar la apertura de la base de datos.
     *
     * @param context
     */
    public DbControl(Context context) {

        super(context, DB_NAME, null, 1);
        this.myContext = context;

    }
```

Las bases de datos nos permiten realizar varias operaciones, pero necesitamos antes abrir ésta para poder tener el control sobre ella. Para ello creamos una instancia de SQLiteDataBase que es el manejador de la base de datos y mediante el método openDatabase conseguiremos el acceso a la base de datos eligiendo si queremos escritura o solo lectura.

En la siguiente figura podemos observar como accedemos a la base de datos y tomamos el control de ella.

```
public void open() throws SQLException {

    //Abre la base de datos
    try {
        //copyDataBase();
        createDataBase();
    } catch (IOException e) {
        throw new Error("Ha sido imposible crear la Base de Datos");
    }

    String myPath = DB_PATH + DB_NAME;
    myDataBase = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.OPEN_READWRITE);

}
```

Insertar Información

El método cuya funcionalidad es añadir filas a nuestras tablas se llama SQLiteDatabase.insert ().

La receta a seguir para usarlo es:

- Crea un objeto del tipo ContentValues. Éste permite almacenar las columnas del registro en pares clave-valor
- Añade los pares con el método put()
- Invoca a insert() a través de la instancia de la base de datos

Sus parámetros funcionan así:

- String table: Nombre de la tabla donde se insertará la información.
- String nullColumnHack: Nombre de una columna que acepta valores NULL y de la cual no se proveen pares clave-valor en values.
- ContentValues values: Conjunto de pares clave-valor para las columnas.

```
public void insertWork(String titulo, Double precio,String icono){
    //Creamos el registro a insertar como objeto ContentValues
    ContentValues nuevoRegistro = new ContentValues();
    nuevoRegistro.put("titulo", titulo);
    nuevoRegistro.put("precio",precio);
    nuevoRegistro.put("icono",icono);

    //Insertamos el registro en la base de datos
    myDataBase.insert("Trabajos", null, nuevoRegistro);
}
```

Lectura de información

Para obtener los registros de nuestra tabla usaremos el método query ().

Este método te ayuda a añadir todas las partes posibles de las cuales se podría componer una consulta, además que te protege de inyecciones SQL, separando las cláusulas de los argumentos.
Los parámetros tienen los siguientes propósitos:

- String table: Nombre de la tabla a consultar.
- String[] columns: Lista de nombres de las columnas que se van a consultar. Si deseas obtener todas las columnas usas null.
- String selection: Es el cuerpo de la sentencia WHERE con las columnas a condicionar. Es posible usar el placeholder '?' para generalizar la condición.
- String[] selectionArgs: Es una lista de los valores que se usaran para reemplazar las incógnitas de selection en el WHERE.
- String groupBy: Aquí puedes establecer cómo se vería la cláusula GROUP BY, si es que la necesitas.
- String having: Establece la sentencia HAVING para condicionar a groupBy.
- String orderBy: Reordena las filas de la consulta a través de ORDER BY.

```
public Cursor ConsultaElementsWorkReads(Integer id){
    String[] args = new String[] {String.format("%d",id)};
    Cursor mcursor=myDataBase.query("ElementsWorkRead",null,"_Id_WorkRead=?",args,null,null,null);
    if (mcursor != null){
        mcursor.moveToFirst();
    }
    return mcursor;
}
```

Ahora, existe otro método alternativo para realizar consultas llamado rawQuery (). Con él pasas como parámetro un String del código SQL de la consulta.

```
public Integer EstaImagen(String imagen){
    Integer repeticiones=0;
    String consulta="SELECT SUM(num) as totalnum from " +
            "(SELECT COUNT(*) as num FROM Datos where Datos.Icono='" + imagen +"' " +
            "union " +
            "SELECT COUNT(*) as num FROM Trabajos where Trabajos.icono='" + imagen +"') as tavla";
    Cursor mcursor=myDataBase.rawQuery(consulta,null);
    mcursor.moveToFirst();
    repeticiones=mcursor.getInt(0);
    return repeticiones;
}
```

Tanto query() como rawQuery() retornan un objeto de tipo Cursor.

Este objeto es un apuntador al conjunto de valores obtenidos de la consulta. Al inicio el cursor apunta a una dirección previa a la primera fila. Por lo que debes leer cada tupla moviendo el cursor a la fila siguiente.

Se emplea el método booleano moveToNext () para avanzar al siguiente registro. Éste retorna true si fue posible o false si ya no existen más elementos.

Borrar Información

Eliminar registros es muy sencillo, solo tenemos que usar el método delete ().

Recibe como parámetros el nombre de la tabla, el estilo de la selección de la cláusula WHERE y los valores de comparación para determinar qué filas borrar.

```
public void Eliminar(String tabla, String where, String[] Args){
    myDataBase.delete(tabla,where,Args);
}
```

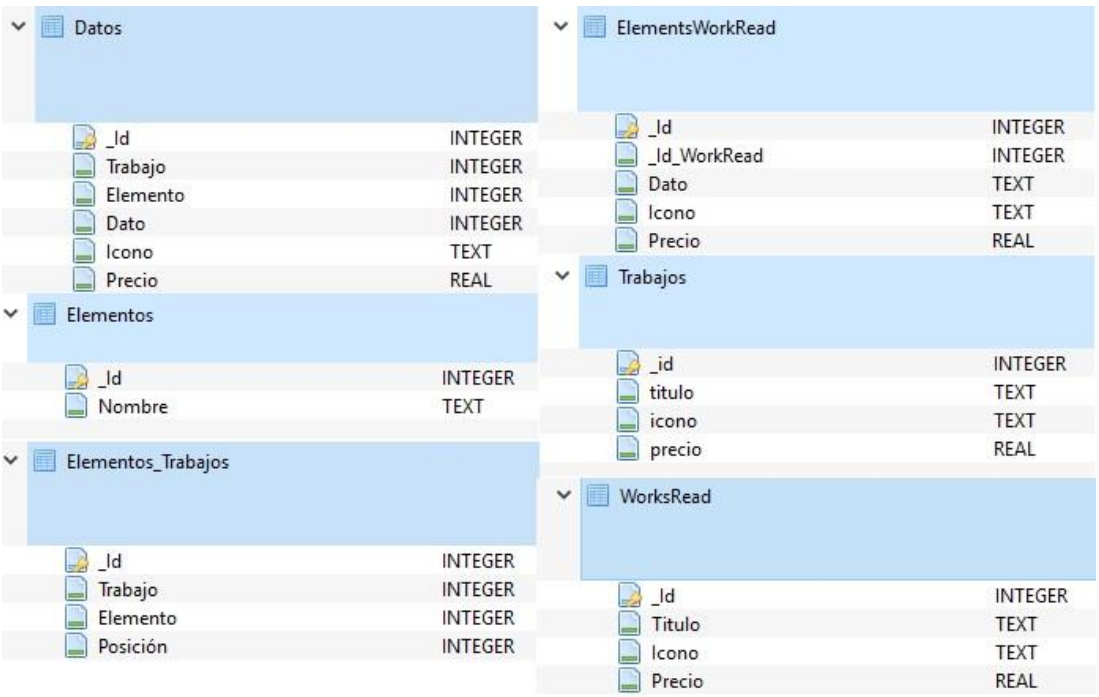
Actualizar Información

En este caso usaremos el método update (). Es exactamente el mismo estilo de uso que los métodos anteriores.

Especificaremos: Nombre de la tabla, Valores nuevos, Instrucción WHERE y Argumentos del WHERE.

```
public Integer Actualizar(String Table,ContentValues Cambios,String Condicion,String[] Args){
    Integer cambios=myDataBase.update(Table, Cambios, Condicion, Args);
    return cambios;
}
```

Tablas de la Base de Datos



En la figura anterior se observa como ésta cuenta con seis tablas para almacenar la información de la aplicación.

Trabajos, Elementos_Trabajos y Datos, son las tablas que se encargan de almacenar todos los datos referentes a las plantillas de los trabajos. Las dos últimas tienen campos que hacen referencia a la tabla trabajo y la tabla Datos otro campo que hace referencia a los elementos. De esta manera conseguimos relacionar los registros de las tablas pudiendo construir posteriormente las plantillas.

En cambio WorksRead y ElementsWorksRead, son las tablas donde se almacenan la información de los trabajos elegidos por los clientes. Al igual que las anteriores también se encuentran relacionadas.

Biblioteca de gestión de imágenes Utilizada

La librería **Glide** gestiona de forma automatizada muchos de los aspectos anteriores en la carga de imágenes. Así que al implementarla veremos como la aplicación fluye mucho mejor.

Glide es un marco rápido y eficiente de gestión de medios de código abierto útil para la carga de imágenes para Android que incluye decodificación, memoria y almacenamiento en caché de disco, y la puesta en común de recursos con una interfaz simple y fácil de usar.

El enfoque principal conseguir un desplazamiento tan suave y rápido como sea posible en una lista de imágenes, pero Glide también es eficaz para casi cualquier caso en el que haya que obtener, cambiar el tamaño o mostrar una imagen remota.

Se usa en todas las partes de la aplicación donde hay que mostrar imágenes, le da mucha agilidad a la aplicación y le quita el trabajo al hilo principal cuando está gestionando las imágenes.

```
Glide.with(holder.image.getContext())
        .load(item)
        .into(holder.image);
```

Capítulo 7 Herramientas Utilizadas

7.1. Introducción

A la hora de realizar este proyecto se ha necesitado apoyarse en distintas herramientas de software para realizar todos los pasos necesarios y poder llevar está aventura a buen puerto. Como todo mundo sabe, las herramientas son complementos que te ayudan a realizar un trabajo ahorrándote parte de él.

7.2. Herramientas

Ahora vamos a realizar una pequeña descripción de las herramientas utilizadas en todos los procesos de la realización de este proyecto.

Android Studio

Se decidió la utilización de código nativo de Android en vez de usar desarrollo web, ya que se trataba de una aplicación de uso exclusivo para el sistema operativo Android. Para la codificación del código Android se eligió la herramienta Android Studio.

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más funciones que aumentan tu productividad durante la compilación de apps para Android, como las siguientes:

- Sistema de compilación flexible basado en Gradle.
- Un emulador rápido con varias funciones.
- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android.
- Instant Run, para aplicar cambios mientras tu App se ejecuta sin la necesidad de compilar un nuevo APK.
- Integración de plantillas de código y GitHub, para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código.
- Gran cantidad de herramientas y frameworks de prueba.
- Herramientas Lint para detectar problemas de rendimiento, uso, compatibilidad de versión, etc.
- Compatibilidad con C++ y NDK
- Soporte integrado para Google Cloud Platform, que facilita la integración de Google Cloud Messaging y App Engine.

Se decidió utilizar esta herramienta frente a otras, porque pertenece a Google que también es el propietario del sistema operativo Android. [9]

Sqlite browser

Es una herramienta de código abierto de alta calidad visual, para crear, diseñar y editar la base de datos de archivos compatibles con Sqlite.

La utilizan los usuarios y desarrolladores que quieran crear bases de datos, búsquedas y editar datos. Utiliza una interfaz familiar que permite hacer las siguientes operaciones de forma más intuitiva.

- Crear archivos de base de datos.
- Crear, definir, modificar y eliminar tablas.
- Crear, definir y eliminar índices.
- Navegar, editar, añadir y eliminar registros.
- Importar y exportar ficheros texto.
- Importar y exportar tablas desde/a archivos CSV
- Importar y exportar bases de datos desde/a archivos de volcado de SQL.
- Realizar consultas SQL e inspeccionar los resultados.
- Examinar un registro de todos los comandos SQL emitidos por la aplicación.

Visual Paradigm

Es una herramienta visual que permite crear todos los diagramas UML y herramientas esenciales para el diseño de sistemas y bases de datos. Permite realizar un modelado sencillo, intuitivo y ágil del sistema que se va a desarrollar. Se ha utilizado para crear los diagramas UML de la aplicación.

Microsoft Word

Este popular procesador de textos, se ha utilizado para el proyecto en dos ocasiones. La primera mientras se diseñaba la aplicación, todas las imágenes del capítulo 4 se crearon con las formas que nos presenta está herramienta. El otro uso fue para escribir está memoria.

Capítulo 8 Pruebas

8.1. Introducción

A la hora de realizar esta aplicación se ha llevado a cabo una serie de pasos para minimizar la cantidad de posibles errores. El proceso se ha basado en crear primeramente una serie de diagramas para cada una de las funcionalidades implementadas en cada Activity. Con cada uno de estos diagramas, cuando son codificados, se prueba los posibles casos de error y se corrige para que sea más robusta y, en los casos necesarios, muestre los errores pertinentes evitando que llegue a forzar el cierre de la aplicación por un fallo. Se dan algunos casos en los que es necesario tener otras funcionalidades añadidas para realizar las pruebas, ya que, trabajan en conjunto.

Por último, se refactorizó el código y se mejoró allá donde se pudo, y se volvieron a pasar las pruebas para comprobar que siguiera funcionando todo correctamente tras la refactorización. Los fallos que fueron apareciendo era prioritario solucionarlos, o bien en el momento en el que se encontrase, o bien se anotaron y se solucionaron tras completar el trabajo que se realizaba en ese momento.

Además, en cada día de avance se ha hecho al menos un repaso general a todas las funcionalidades implementadas hasta el momento en busca de posibles errores no pensados en un primer momento y se ha comprobado que todo siguiera funcionando correctamente.

Posteriormente se ha añadido a la aplicación un conjunto de pruebas sistemáticas utilizando las librerías de pruebas Android y se ha realizado en último lugar la prueba del mono.

8.2. Pruebas manuales en el terminal

Las baterías de pruebas manuales en el terminal son las que se han estado realizando durante todo el desarrollo de la aplicación. Cada vez que se implementaba una nueva funcionalidad, se llevaba a cabo una refactorización o se corregía algún error que pudiera estar relacionado directa o indirectamente con algunas de las funcionalidades ya implementadas anteriormente se comprobaba que siguieran funcionando correctamente dichas funcionalidades. Diariamente se llevaba a cabo un repaso general de las pruebas de las funcionalidades implementadas hasta ese momento para comprobar que todo siguiera funcionando correctamente e intentar detectar posibles errores y posibles casos que pudieran inducir a futuros errores no pensados hasta el momento.

Para realizar estas pruebas nos hemos ayudado de algunas de las herramientas que posee Android Studio como son:

- Registro de Eventos: contiene los logs (trazas de ejecución de una aplicación). Existen varios registros de eventos: uno principal y varios secundarios específicos a un dominio.

- Depuración: es la fase consistente en buscar y corregir los fallos.

- DDMS: está forma de depuración contiene varias vistas: Devices, Emulator, Control, Threads, Heap, Allocation Tracker y File Explorer, que nos ayudan a ver el funcionamiento real de la aplicación y lo que sucede sobre ella.

A continuación, en la figura se muestra algunas de estas baterías de pruebas.

Nº PRUEBA	PRUEBA REALIZADA	RESULTADO	OBSERVACIONES
3.1	MainActivity		
3.1.1	Cargar Elementos de la Activity	OK	Aparecen correctamente todos los elementos en la pantalla.
3.1.2	El botón "Continuar" accede al menú de trabajos	OK	Muestra El menú de los Trabajos.
3.1.3	La barra del menú superior opción "Salir"	OK	La aplicación sale.
3.1.4	La barra del menú superior opción "Administrar"	OK	Aparece la pantalla de Login para administrar los trabajos.
3.2	Acceso		
3.2.1	Cargar Elementos de la Activity	OK	Aparecen correctamente todos los elementos en la pantalla.
3.2.2	Realiza el Login correctamente	OK	El usuario administrador accede correctamente y se observa el menú de administración.
3.2.3	Muestra el mensaje de error si el Login no es correcto	OK	Aparece un mensaje en rojo en el lugar adecuado cuando no se introducen los datos correctos.
3.3	Administración		
3.3.1	Cargar Elementos de la Activity	OK	Aparecen correctamente todos los elementos en la pantalla.
3.3.2	Los botones responden correctamente al realizar click.	OK	Dependiendo del botón pulsado aparece la pantalla seleccionada.

8.3. Pruebas unitarias

El testing, o pruebas de software, consisten en mejorar la calidad del código a través de la creación de distintas pruebas que se llevan a cabo sobre el código. Permite obtener un código más robusto y menos susceptible a que por posibles cambios futuros se rompan otras funcionalidades que aparentemente funcionaban bien previamente.

El testing sistemático consiste en crear un paquete de pruebas automáticas que exponen el código a que dé la respuesta esperada dadas unas premisas o valores de entrada. Permiten probar así que esa pieza de código funciona tal y como está previsto en todos los diferentes escenarios que al programador se le ocurran, de forma rápida y automática en lugar de tener que ir probando escenario a escenario distinto. Esto permite que los paquetes de pruebas puedan ser pasados a menudo sin la problemática de que alguno no se realice por descuido o porque sean pesados de realizar si hay que cambiar partes del código o cambiar mucho la configuración del dispositivo móvil para cada una de las pruebas. Esto conlleva un gran ahorro de tiempo en el testeo de cada funcionalidad.

Un test unitario es aquel que se encarga de probar unidades pequeñas de código. La granularidad del ámbito del test es variable, desde métodos, un conjunto de ellos, clases… Se intenta que sean probados de forma aislada de otros elementos como bases de datos o entrada de información desde Internet. Estos datos serán emulados para que los test se centren simplemente en la funcionalidad de la pieza de código que se está probando. Un test de sistema prueba la integración entre varias partes del sistema o aplicación, utilizando los mismos puntos de entrada que en la aplicación real utilizada por el usuario, incluida la interfaz gráfica. De este modo se puede probar que dada una interacción del usuario con la interfaz gráfica la aplicación reacciona de la forma esperada, pero haciéndolo de forma automatizada con todos los posibles escenarios que crea convenientes el programador.

En Android existen una serie de clases específicas para realizar test sin necesidad de otras librerías externas, basadas en JUnit 3. Las clases más importantes son ActivityInstrumentationTestCase2, ActivityUnitTestCase, ProviderTestCase2 (para Content Provider) y ServiceTestCase (para Service).

Se han realizado una serie de pruebas unitarias para probar la funcionalidad de ciertas piezas de código de la aplicación y para probar la integridad del código con la interfaz gráfica.

La primera clase utilizada para el desarrollo de este tipo de pruebas ha sido, en concreto la clase InstrumentationTestCase. No se prueba la interfaz gráfica pero sí se hace uso de la instrumentación para monitorizar. Se han hecho test para probar que la base de datos realiza las operaciones correctamente. Se ha realizado uno por tipo de operación (Lectura, Inserción, Actualización y

Borrado), y otros sobre una base de datos con errores para comprobar que realiza lo adecuado ante dichos errores.

La segunda clase se encarga de los test de sistemas, donde se ha querido probar la integración entre la funcionalidad del código con la representación gráfica en la interfaz y que interactúan del modo esperado. Para ello se ha hecho uso de la librería Robotium y de la clase ActivityInstrumentionTestCase2 de las librerías de test de Android.

Un ejemplo de este tipo de test es en el que se han probado las listas de trabajos de la pantalla de "List_View_Works". Dado que esta pantalla tiene varios componentes en las listas se han diseñado los test para que prueben que al seleccionar en los botones interiores acceda a los contenidos correctamente, que estén visibles, y además muestre la información adecuada en cada uno de los distintos elementos de la lista.

Estas pruebas podrían hacerse directamente con las clases derivadas de Instrumentation y otras clases del paquete de test de Android, pero la librería Robotium facilita la elaboración de los test. Además, Robotium permite test de caja negra, por lo que no se ha de conocer cómo está implementado el código en detalle, sino que se puede utilizar directamente instrucciones tales como "presiona el texto que ponga Aceptar" (solo.clickOnText ("Aceptar")), que facilitan en gran medida el pulsar sobre los elementos.

Para las pruebas se creó un proyecto de test en Android Studio separado del proyecto principal de la aplicación de APTACAN, pero dependiente de éste, por lo que puede hacer uso de las clases y métodos del proyecto principal para probarlos. Además, de este modo el usuario final sólo ha de instalar el .apk del proyecto principal, ocupando un menor espacio la aplicación en el dispositivo móvil del usuario.

Para crear los tests de sistema, se ha de crear un objeto Solo, perteneciente a la librería de Robotium pasándole como argumentos el objeto Instrumentation y la Activity que se quiere probar. Todos los métodos relacionados con la interfaz gráfica y la instrumentación se realizan llamándose sobre el objeto de la clase Solo, como por ejemplo Solo.assertCurrentActivity (String message, Class activityClass) o Solo.clickOnText (String text). [10]

Además, si se está probando una actividad como en el caso de los test de sistema creados, es imprescindible el incluir el constructor de la clase de test con una llamada al método super pasándole como argumento el nombre de la clase Activity a probar, como por ejemplo super (DeviceConfiguration.class).

8.4. Prueba del mono

Anteriormente hemos visto las pruebas unitarias que permiten realizar casos de prueba precisos informados por el desarrollador. Pero, si bien el propio desarrollador habrá tenido en cuenta el describir la máxima cantidad de casos de prueba, en un entorno real, el usuario podrá interactuar con la aplicación de una forma que no esté prevista por el desarrollador. Es en ese punto donde interviene el concepto de prueba del mono para ayudar al desarrollador a reducir el campo de interacciones de usuario que no se hayan probado.

La prueba del mono es una prueba aleatoria de la aplicación mediante su interfaz gráfica. Está prueba simula un uso de la interfaz gráfica de usuario mediante la pulsación de teclas, realización de gestos táctiles, clics y demás eventos que cualquier usuario pueda realizar de forma completamente aleatoria. Esto permitió probar numerosos casos de uso en los que no se pensó en el desarrollo. [11]

Los resultados fueron satisfactorios ya que no se creó ninguna acción que pusiera en jaque a la aplicación (excepción no capturada, bloqueo de la aplicación o caída), ya que durante sus ejecuciones no se detuvo en ningún momento.

Capítulo 9 Manuales

Una vez terminada la aplicación, para la entrega en APTACAN se pensó en crear un manual para ayudar a la familiarización con la aplicación y de esta manera sacarle todo el rendimiento posible. Se crearon dos manuales distintos, uno que este pensado para los administradores de la aplicación, sobre la creación de las plantillas de trabajo, y otro enfocado en cómo va el proceso en el momento de recepcionar los trabajos.

El primero de los manuales, nos guía en la administración de la aplicación ayudando a su lector en la confección de las plantillas. Se muestra cómo añadir imágenes, crear trabajos, elementos asociados ha dicho trabajo y los datos que necesita ese elemento, así como asociar una imagen en los lugares habilitados para ello.

El segundo de los manuales, muestra cómo se lleva a cabo la recepción de uno o varios trabajos por medio de la aplicación. Este proceso nos lleva desde el saludo inicial hasta el momento del cobro y la despedida. Se trata de una secuencia intuitiva y fácil de seguir; aun así se crea este documento por si surge alguna duda sobre el funcionamiento de la misma.

Se han definido todos los pasos de la forma lo más sencilla y clara posible, intentando que la persona que lo utilice en todo momento pueda comprender las acciones que se llevan a cabo para conseguir su objetivo.

Los manuales se entregan en formato electrónico a APTACAN y se añaden como anexos en la parte final de este documento.

Capítulo 10 Conclusiones

En este proyecto se ha realizado la aplicación de gestión de Reprografía de APTACAN para el sistema operativo Android. Acercando a los estudiantes y clientes de APTACAN los servicios ofrecidos de una forma que resulten más cómodos, sencillos y accesibles.

En el momento de la redacción de esta memoria aún no se utiliza el aplicativo, ya que, entre la finalización del curso pasado y el comienzo del presente, los administradores no han tenido tiempo de entrenar a los estudiantes. Pero espero que sea de gran utilidad una vez entre en funcionamiento.

Tras concluir el proyecto queda hacer un repaso de lo que ha sido este proyecto, los objetivos cumplidos, así como las líneas de trabajo futuras.

Me siento satisfecho con los objetivos cumplidos en este proyecto, a pesar de que todavía no está implementada en el centro de APTACAN, al depender de los tiempos que hay que cumplir para su implantación. La aplicación para gestionar los trabajos de reprografía aunque está terminada me hubiese gustado verla en uso en su centro para saber qué tal es la experiencia de usuario al trabajar con ella.

A continuación se hablará sobre lo realizado y diseñado en el proyecto, sobre mis objetivos personales, qué he aprendido al hacer este proyecto y por último las líneas futuras a seguir.

10.1. ¿Qué he hecho?

Para el desarrollo de esta aplicación, debido a ser para APTACAN, miembros externos de la comunidad universitaria, se ha necesitado colaborar con varias personas para llevar el proyecto hacia delante.

Principalmente me encargue de toda la programación de la aplicación en Android y la creación de pruebas sistemáticas y manuales. Otra de las tareas fue diseñar y crear las distintas pantallas que no entraban en la forma de trabajar que realizan ahora en el centro. En cuanto a los diseños creados por APTACAN tuve que adaptar algunos de ellos por la mala usabilidad del diseño actual.

A la hora de la codificación, he hecho uso de código ya implementado, siguiendo la reutilización de código inherente a cualquier API, pero adaptándolo al uso y necesidades de la aplicación. Todo ha sido fruto de la búsqueda en diversos sitios y comunidades de Internet, especialmente en la web "stackoverflow", adaptando, mezclando y mejorando las ideas propuestas y los ejemplos encontrados para esta aplicación. [12]

10.2. ¿Qué he aprendido?

Durante el desarrollo de la aplicación de recepción de trabajos de reprografía de APTACAN he podido afianzar y mejorar mis conocimientos de programación previos, al igual que introducirme en el desarrollo de Android, que nunca había realizado anteriormente.

Por ejemplo, hacer uso de las carpetas de recursos alternativos o cómo manejar los ciclos de vida y las actividades de Android. Sobre todo me ha permitido conocer elementos más complejos que manejan informaciones personalizadas, tan útiles como los ListView o GridView, que permiten una interfaz y experiencia de uso más rica y moderna.

Además, me ha ayudado a afianzar mi conocimiento sobre las base de datos para la comunicación de información, técnica útil y muy interesante para el desarrollo de aplicaciones móviles.

Por otro lado, me ha permitido aprender a realizar una aplicación para un uso real desde cero, y trabajar conjuntamente junto a otros profesionales para su consecución satisfactoria.

El utilizar una serie de buenas prácticas, tales y como los comentarios, nombres de variables y métodos descriptivos y las refactorizaciones, me ha permitido obtener código más claro, organizado y limpio, que a la hora de revisar ciertas funcionalidades pasado un tiempo ha permitido llevar a cabo estas tareas de forma mucho más rápida. Además, facilita la posibilidad de modificaciones futuras de la aplicación por si en el futuro cambiase o ampliase, no siendo necesaria mucha explicación del código de la aplicación en el proceso.

Por último, he podido comprobar la utilidad de las pruebas sistemáticas al poder realizar las mismas pruebas cambiando condiciones del escenario de forma rápida sin necesidad de tener que cambiarlas manualmente en el código o en el dispositivo móvil, pues se trata de una tarea pesada y que puede provocar errores en caso de olvidarse de dejar todo igual que al principio de las pruebas. También he podido aprender a realizar pruebas de sistema con Robotium, que en verdad hace que sea muy fácil y rápido y permite realizar test de caja negra fácilmente.

10.3. Líneas futuras

Como conclusión a esta memoria se va a hablar sobre las líneas futuras que ha de seguir la aplicación y algunas posibles mejoras futuras.

A continuación se van a enumerar posibles mejoras que no condicionan la correcta funcionalidad de la aplicación, pero pueden mejorarla sustancialmente:

- Crear el diseño para otros tipos de dispositivos.

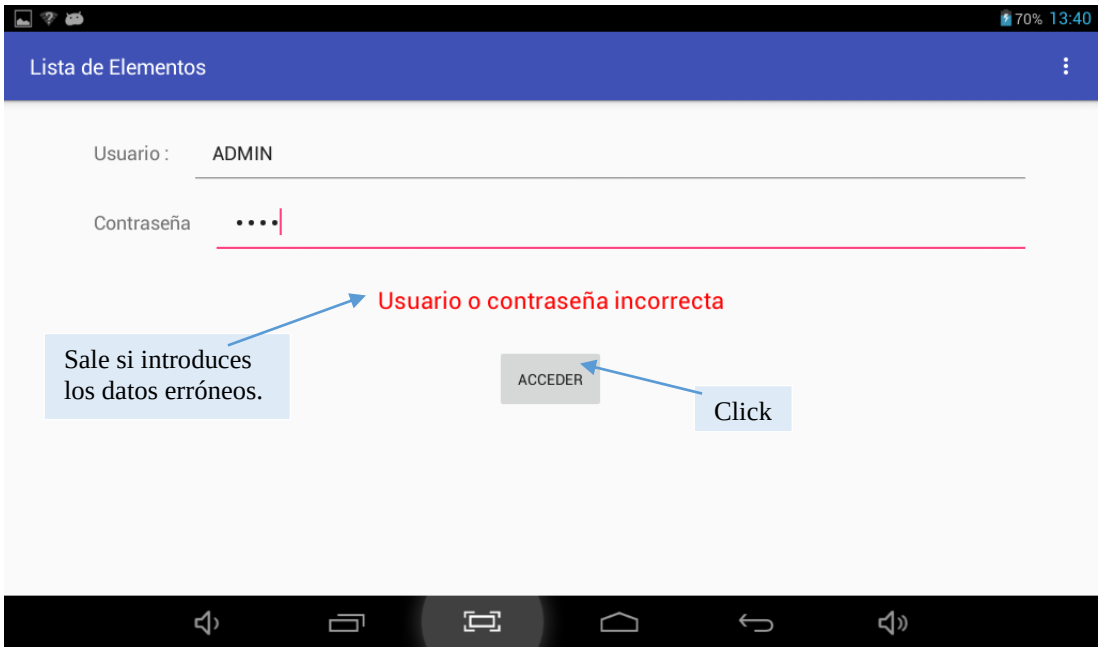
- Introducir la voz en la aplicación, para que sea capaz de enviar en forma de audio parte de la información mediante los altavoces del dispositivo.

- Creación de elementos nuevos que puedan dar más amplitud a las plantillas de los trabajos o sean buenos para ampliar su uso a otros tipos de trabajo.

Anexo I
Manual de Administración de la App.

Una vez situado en la pantalla principal, en la parte de arriba a la derecha, se encuentra el menú. Hay que hacer click sobre los puntos suspensivos, para que nos aparezcan las dos opciones disponibles (Administrar y Salir). Para poder administrar la aplicación pulsamos sobre “Administrar”.

Al pulsar sobre el botón “Administrar” llegamos a la pantalla del Login, aquí introduciremos los datos proporcionados (Usuario: “ADMIN”, Contraseña: “13579”). Una vez introducidos correctamente, estos datos, pulsamos sobre el botón “Acceder”.



Si se introducen datos erróneos aparece el mensaje de error, en cambio si los datos introducidos son correctos se avanza a la pantalla con el menú de las distintas partes configurables de la App.

En esta pantalla nos encontramos las tres posibles partes configurables (Trabajos, Imágenes y Monedero), para acceder a cualquiera de estas partes de configuración de la App, se hace click sobre el botón que se desea.

Monedero: nos va a permitir introducir los billetes y monedas que tenemos disponibles en la caja en ese momento.

Imágenes: aquí se dispone de un método de entrada de una imagen y la posibilidad de ver las imágenes almacenadas en la app, pudiéndose eliminar si así lo deseamos.

Trabajos: lugar desde donde vamos a poder crear las plantillas que se mostraran a los clientes. Está formado por trabajos, elementos y datos, con estos elementos configuras la plantilla que verá el cliente.

A continuación, se explica más detalladamente cada una de estas partes configurables, prestando una atención especial a la parte de los trabajos debido a su complejidad.

MONEDERO

Al hacer click sobre “Monedero” se muestra la pantalla mediante la cual se controla las monedas y billetes disponibles en la caja en ese momento.

Al hacer click sobre el botón “Limpiar Caja”, todos los contadores de las monedas y billetes muestran “0”. Al pulsar sobre uno de los iconos de las monedas o billetes, su contador se va incrementando en una unidad cada pulsación.

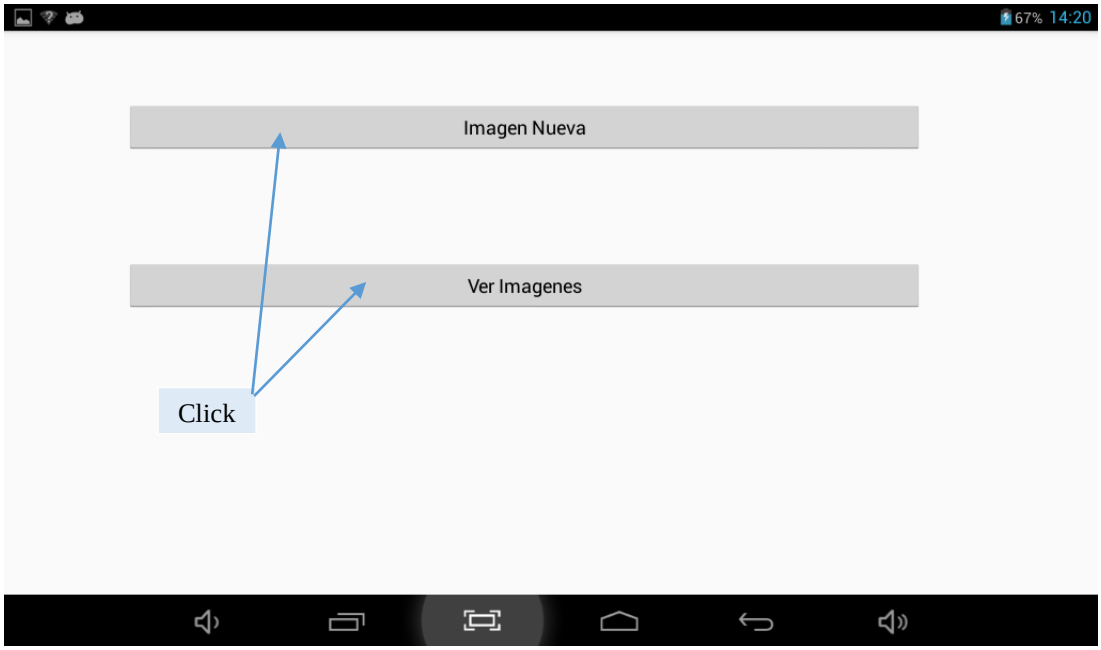
En la parte de arriba a la izquierda se muestra la cantidad total del dinero disponible en la caja, cuando se pulsa sobre una moneda o billete se incrementa la cantidad del elemento pulsado. En cambio al hacer click en “Limpiar Caja” este valor se cambia a “0”.

Todos los cambios que se van produciendo sobre la caja, ya sea añadir dinero o vaciarla, se almacenan automáticamente. Terminadas todas las acciones que haya que hacer en esta pantalla, dando hacía atrás en el dispositivo, los cambios quedan guardados y nos muestra el menú de configurables.

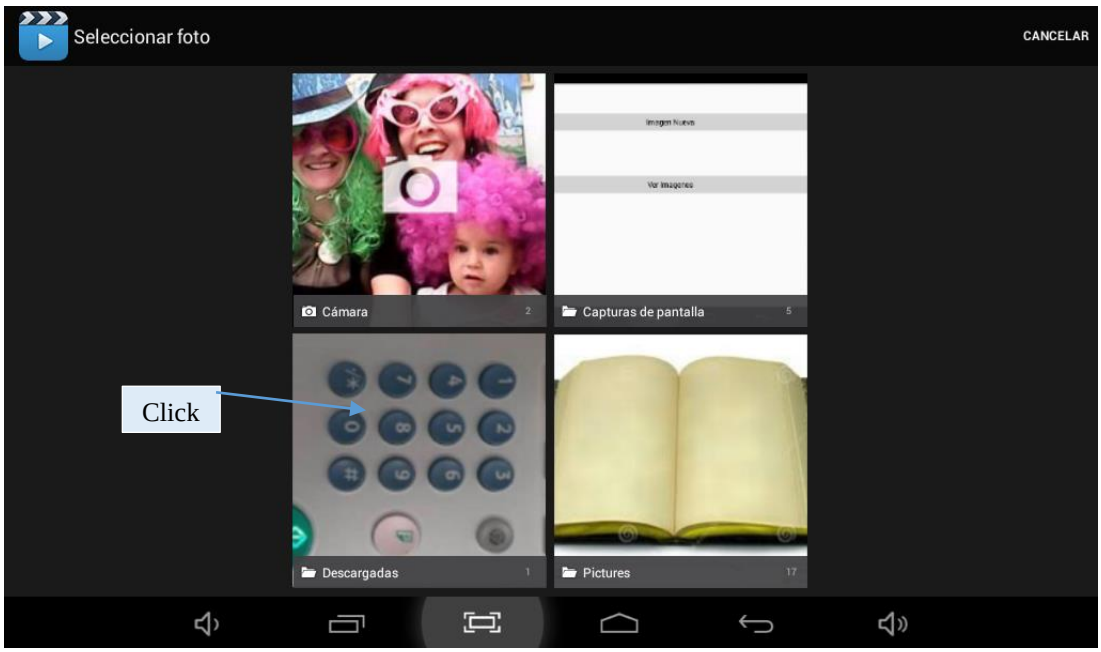
Las monedas y billetes que se han guardado serán las que sirvan en la parte principal de la aplicación para gestionar las vueltas de los trabajos realizados.

IMÁGENES

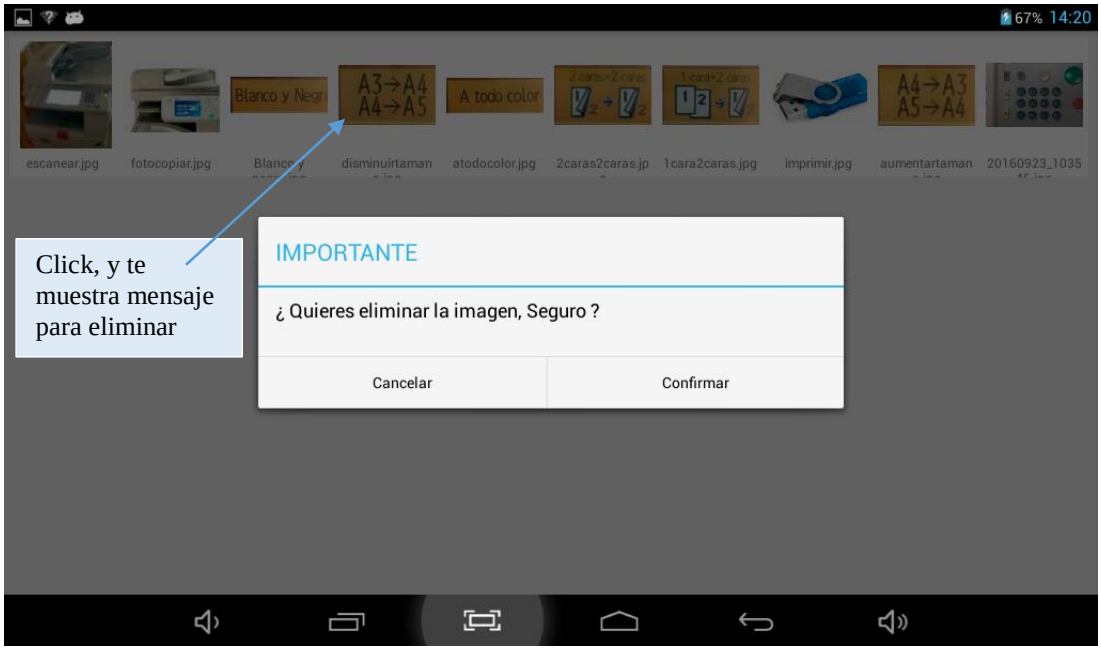
De vuelta en el menú de las partes configurables si se pulsa sobre el botón “Imágenes” nos lleva a esta pantalla, desde la cual podremos gestionar las imágenes que están asociadas a las partes del trabajo.



Si pulsas sobre el botón “Imagen Nueva” la aplicación va abrir una pantalla donde se podrá elegir una imagen entre todas las que hay en el dispositivo, cómo si abriese la galería del dispositivo. En cambio, si haces click en “Ver imágenes” te mostrara en una pantalla todas las imágenes disponibles para utilizar en la aplicación.



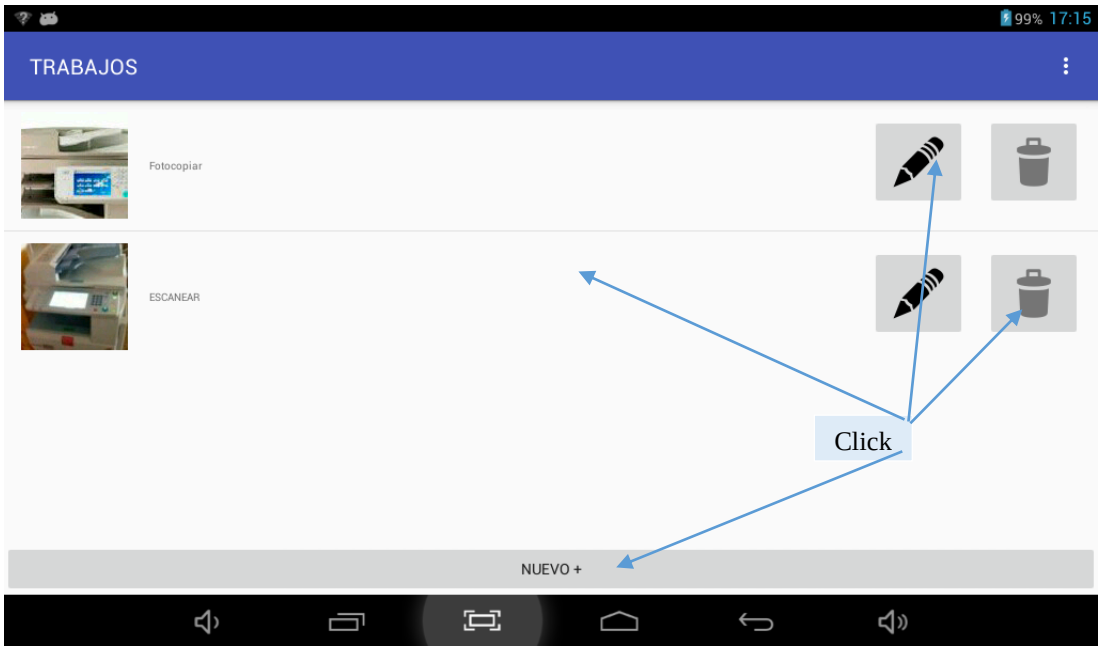
El botón “Imagen Nueva” muestra esta pantalla, donde al pulsar sobre una imagen la añade a la colección de imágenes de la aplicación y vuelve a la pantalla anterior.



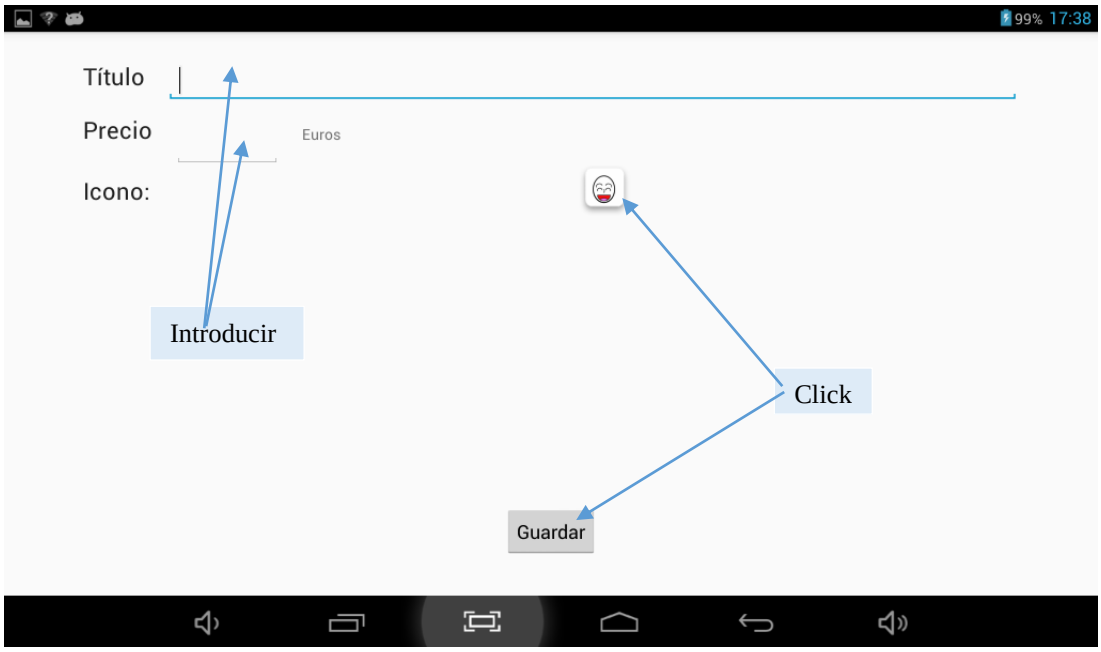
El botón “Ver Imágenes” muestra esta pantalla, donde nos muestra todas las imágenes que podemos usar en la aplicación. Al hacer click sobre una de ellas nos muestra un mensaje por si queremos eliminarla y nos devuelve a la pantalla anterior, si no pulsamos usamos la flecha hacia atrás para regresar a la pantalla anterior.

TRABAJOS

De vuelta en el menú de las partes configurables si se pulsa sobre el botón “Trabajos” nos lleva a esta pantalla, desde la cual podremos gestionar los trabajos.

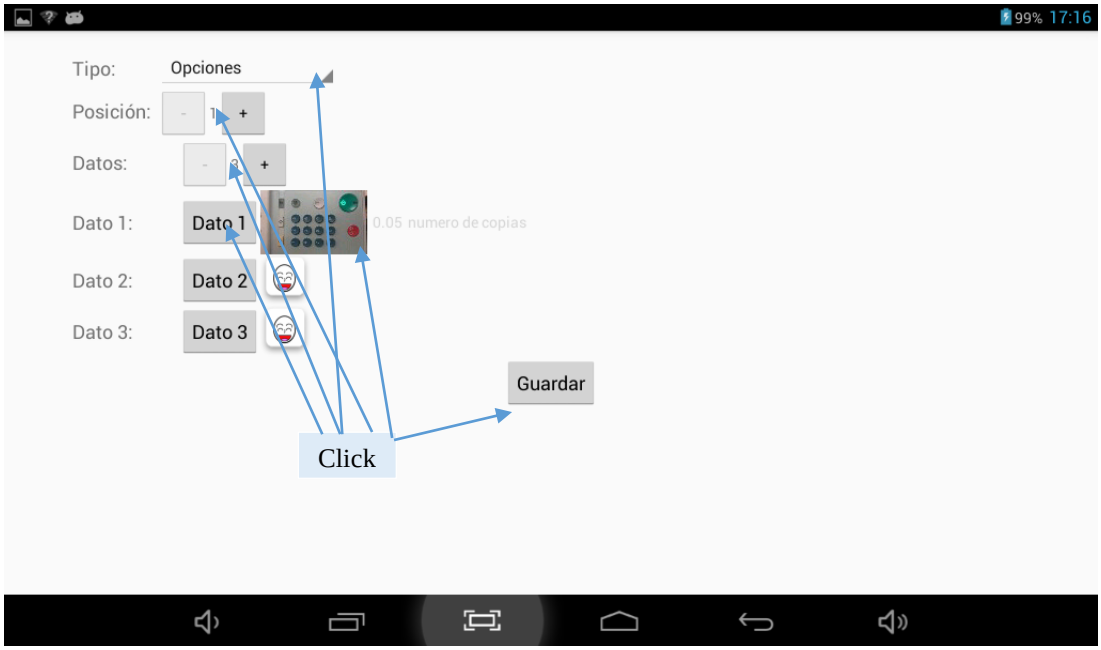


Nos muestra los distintos trabajos que se han creado para la aplicación, hay cuatro posibilidades para pulsar, en la “papelera” se eliminaría el trabajo que se encuentre en esa fila, posee un mensaje por si se da por error. En el “lápiz” o “nuevo+” muestra una pantalla para introducir los datos del trabajo o editarlos. Por último, en la fila muestra la pantalla de los elementos que tiene el trabajo.

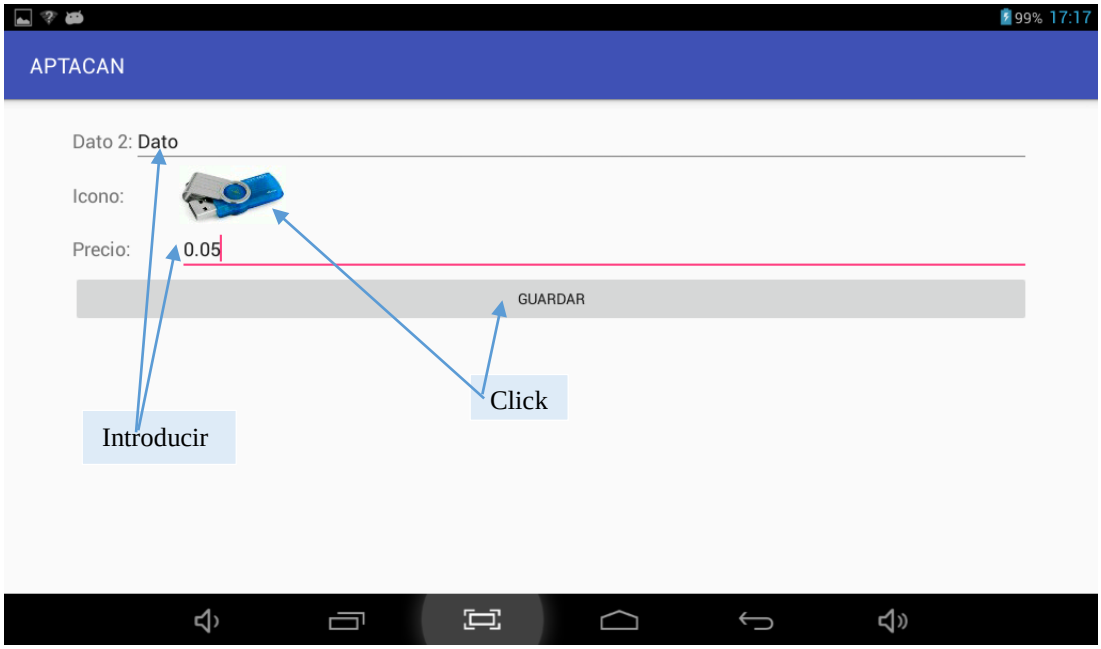


Al pulsar sobre el “lápiz” o “nuevo+” te lleva a esta pantalla donde puedes introducir los datos o cambiarlos desde el “lápiz” y haciendo click sobre la imagen se le asocia una imagen al trabajo. El botón “guardar”, almacena los datos introducidos y regresa a la lista de los trabajos.

Al pulsar sobre la fila de un trabajo muestra la lista de los elementos asociados al trabajo, hay tres posibilidades para pulsar, en la “papelera” se eliminaría el trabajo que se encuentre en esa fila, posee un mensaje por si se da por error. En el “lápiz” o “nuevo+” muestra una pantalla para introducir los datos del trabajo o editarlos.



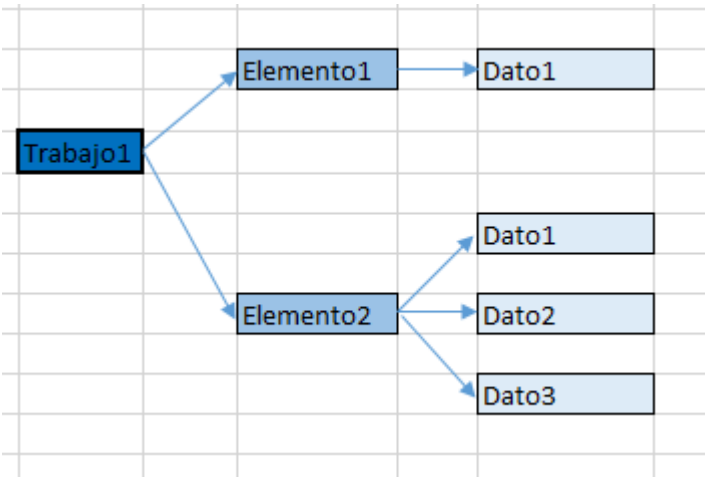
Al pulsar sobre el “lápiz” o “nuevo+” te lleva a esta pantalla donde puedes introducir los datos o cambiarlos desde el “lápiz” y haciendo click sobre la imagen se le asocia una imagen al trabajo. El botón “guardar”, almacena los datos introducidos y regresa a la lista de los trabajos. En cambio, si se pulsa sobre unos de los botones “dato X” muestra la pantalla para introducir la información de ese dato.



Al pulsar sobre el “dato X” te lleva a esta pantalla donde puedes introducir los datos o cambiarlos y haciendo click sobre la imagen se le asocia una imagen al trabajo. El botón “guardar”, almacena los datos introducidos y regresa a la pantalla del elemento.

ELEMENTOS

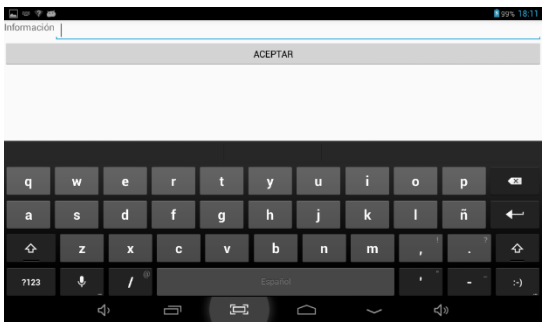
Vamos a explicar, con apoyo gráfico, como está compuesto un trabajo para comprender las siguientes pantallas. Cada trabajo es independiente de los demás, pero está compuesto por uno, varios o ningún elemento, que a su vez, pueden contener uno o varios datos, dependiendo del tipo de elemento.



Cómo se observa en este gráfico. A continuación se muestran los distintos tipos de datos que se puede asociar a un elemento.

Entrada numérica: Solicita datos al cliente, pero sólo puede ser numérico.

Entrada normal: Solicita datos al cliente, alfanuméricos.



Opciones: El cliente va a seleccionar una de todas las opciones que tiene, el primer dato es el título del grupo.

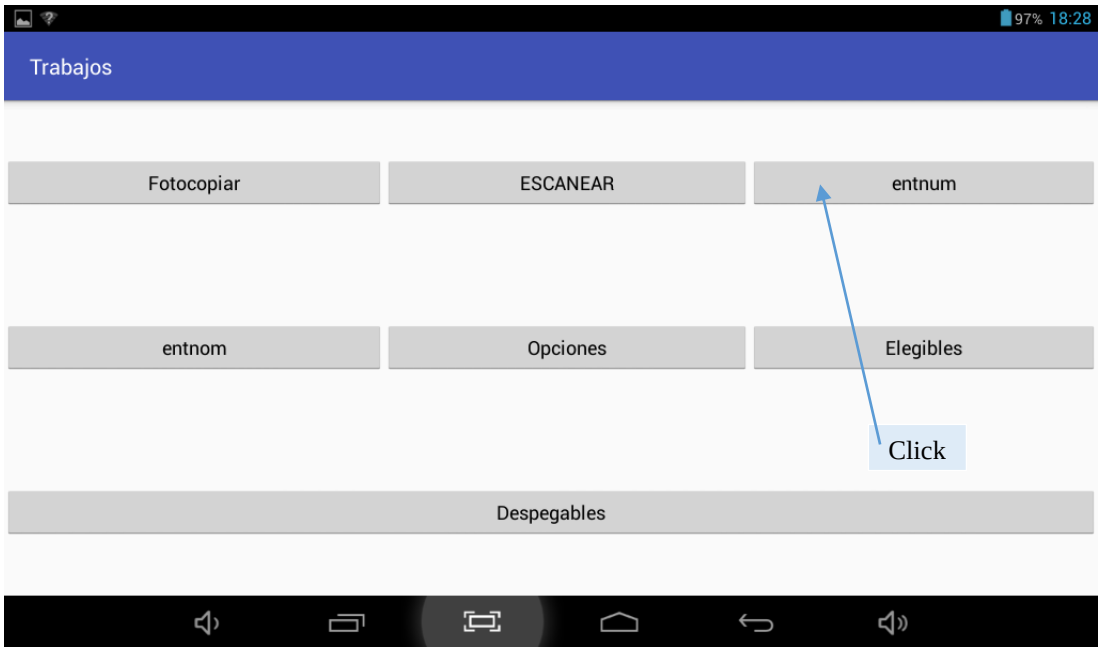
Elegible: Se da la opción para elegir o no un dato en concreto, sirve por si hay que mostrar algo que no sea obligatorio.

Desplegables: Entre varias opciones que se despliegan al pulsar sobre él se escoge una. Es útil cuando rara vez se elige una opción distinta a la mostrada para ocupar menos tamaño que con las opciones.

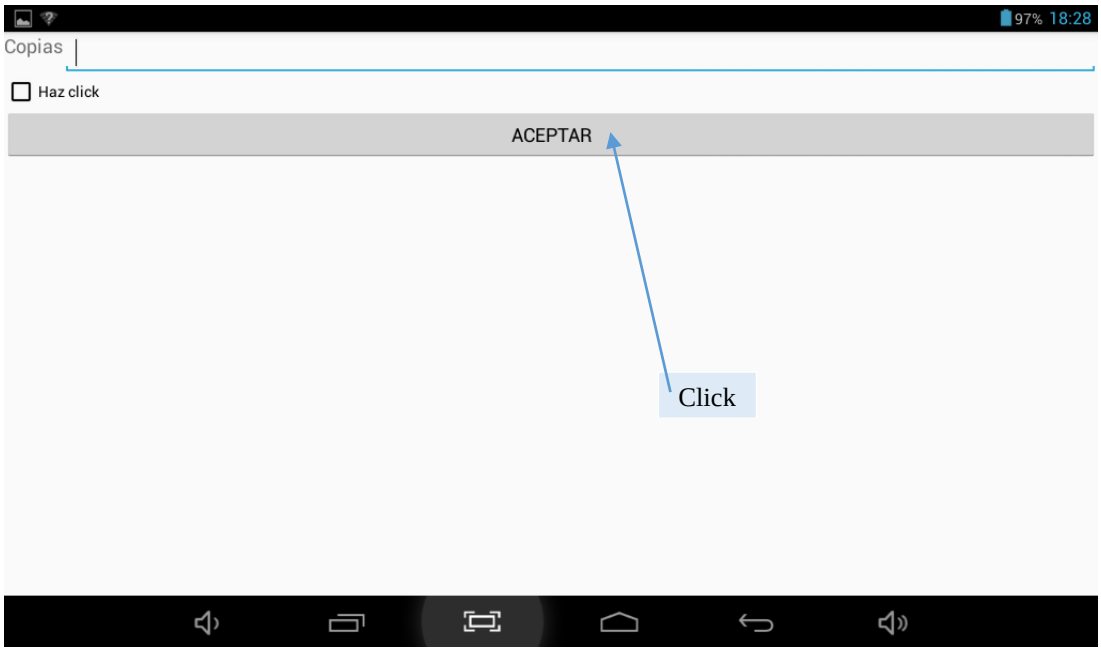
Anexo II
Manual de Uso de la App.

Una vez situado en la pantalla principal, se le entrega la Tablet al cliente. En ella se le muestra un mensaje de bienvenida y a hará click en el botón “continuar” para avanzar en la aplicación.

A continuación al cliente se le muestra todos los trabajos que se pueden atender en el dispositivo, teniendo que pulsar sobre uno de ellos, exactamente en el que desee realizar.



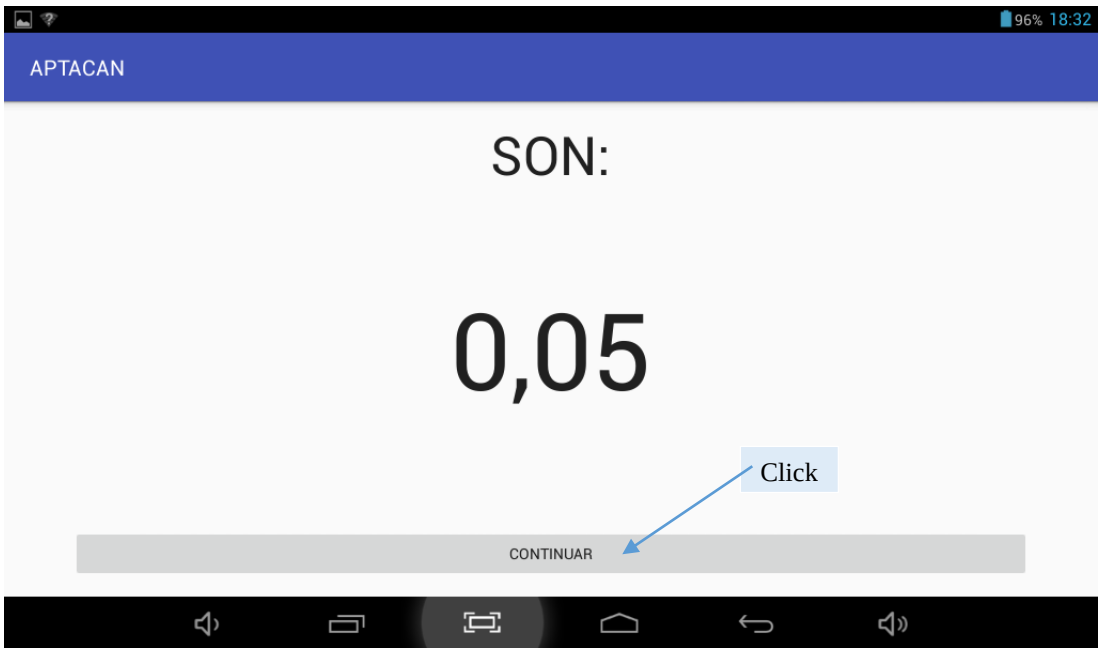
Después de seleccionar uno de los trabajos disponibles, se carga una pantalla con la plantilla creada en la parte de administración, aquí tendrá que introducir los datos según las características que tenga el trabajo.



Una vez introducidos los datos hay que pulsar sobre “aceptar”, el sistema nos muestra un mensaje con la siguiente pregunta “¿Quieres realizar algún trabajo más?”. En caso afirmativo, volverá al menú de los trabajos repitiéndose la acción tantas veces como desee el cliente. En caso negativo, el trabajador empezara a realizar los trabajos pertinentes.

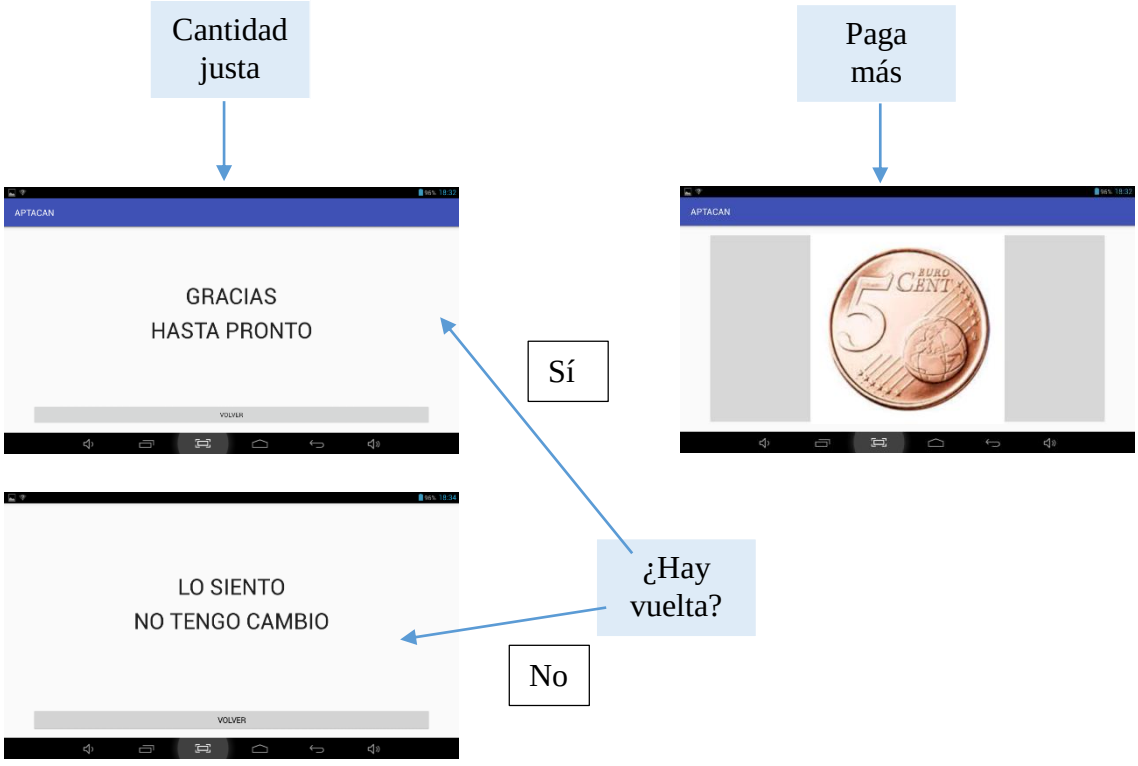
Una vez el cliente termina de elegir los trabajos y el trabajador reciba la Tablet otra vez. Empiezan a salir las imágenes relacionadas con los trabajos seleccionados y las preferencias elegidas. Para ir avanzando en las imágenes hay que pulsar sobre la imagen que aparece en la pantalla.

Al terminar de ir mostrando todas las imágenes de los trabajos con las preferencias que han sido elegidos por el cliente se muestra la siguiente pantalla con el total del precio de los trabajos.



El trabajador le enseña la Tablet al cliente para que vea el valor total de los trabajos y pulsa sobre continuar avanzando a la parte del cobro.

Una vez que el cliente ya sabe el precio y se pulsa el botón “continuar”, el trabajador irá haciendo click en la moneda o billete que le entregan hasta que le entrega la cantidad exacta o supera el precio total. El campo “Quedan” va mostrando el dinero que todavía falta por entregar y se actualiza después de cada acción. Si la cantidad es justa, se muestra una despedida y se vuelve a la pantalla inicial. En cambio, al pagar más cantidad se muestra la imagen de las monedas que hay que ir dando al cliente. Si tienes suficiente cambio una vez entregado se muestra la despedida y si no muestra un mensaje diciendo que no hay cambio. En ambos casos se vuelve a la pantalla principal. Y empieza el ciclo cuando llegue otro cliente.



Referencias

[1] Término de “autista”:
http://www.autismo.org.es/sobre-los-Tea

[2] Ventas de Smartphones en algunos lugares del mundo, por tipo de sistema operativo:
http://www.expansion.com/economia-digital/companias/2015/12/09/56684be1ca474151018b4590.html

[3] Tabla de ventas de Smartphones de los años 2012 y 2013 y el porcentaje de mercado que abarca los cinco principales sistemas operativos de móviles. http://www.idc.com/getdoc.jsp?containerId=prUS24257413

[4] Modelo de capas de la arquitectura Android:
http://www.hermosaprogramacion.com/wp-content/uploads/2014/08/android-arquitectura.jpg

[5] Comparación de los procesos de compilación, Java vs Dalvik:
http://www.hermosaprogramacion.com/wp-content/uploads/2014/08/JVM-vs-Dalvik-683x1024.jpg

[6] Información sobre los Layouts:
https://developer.android.com/guide/topics/ui/declaring-layout.html

[7] Archivos de preferencias en Android:
https://developer.android.com/reference/android/content/SharedPreferences.html

[8] Documentación oficial sobre SQlite:http://sqlite.org/

[9] Descarga e información sobre AndroidStudio:
https://developer.android.com/studio/index.html

[10] Robotium: https://code.google.com/p/robotium/

[11] Libro de Programación Android Ediciones Eni.

[12] Web stackoverflow: http://stackoverflow.com/