UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

## Escuela Técnica Superior de Ingeniería Industrial

# Desarrollo de una aplicación móvil en Android Studio para la gestión del inventario y maquinaria de una empresa

Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

AUTOR/A: Le Metayer Rodríguez, Arturo

Tutor/a: Gil Gómez, José Antonio

Cotutor/a: García García, Inmaculada

CURSO ACADÉMICO: 2023/2024

# Resumen

El trabajo consistirá en el diseño y desarrollo de una aplicación para dispositivos móviles que facilitará el día a día de una empresa al permitir una gestión eficiente y sencilla de su inventario de productos y materiales. Con esta aplicación se podrá controlar el inventario de la organización, con acciones como agregar y editar objetos del inventario de la empresa. Además, la aplicación tendrá una funcionalidad añadida que permitirá al usuario manejar la gestión de la maquinaria de la empresa de forma similar al inventario. Tanto a cada objeto del inventario como a cada máquina, se le podrán asignar datos relevantes, como fecha de adquisición, resultados de la última inspección, fecha de la próxima inspección o fecha estimada del límite de vida útil. La aplicación dará la posibilidad de notificar al usuario de la proximidad de alguno de estos eventos de forma automática y automatizar cambios en el inventario. El trabajo se realizará en Android Studio usando el lenguaje de programación Java.

Palabras clave: Aplicación; móvil; Android; Java; programación; empresa; inventario; máquina; pieza; notificación

# Summary

The work will consist of the design and development of an application for mobile devices. With this application it will be possible to control the inventory of the organization, with actions such as adding and editing objects to the current inventory of the company. In addition, the application will have an added functionality that will allow the user to take care of the management of the company's machinery, in a similar way to the inventory. Each item in the inventory, as well as each machine, can be assigned different relevant data, such as date of acquisition, results of the last inspection, date of the next inspection or date of the estimated useful life limit. The application will give the possibility to notify the user of the proximity of any of these events automatically and automatize changes in the stock. The work will be done in Android Studio using the Java programming language.

Key words: Application; mobile; Android; Java; programming; enterprise; inventory; machine; part; notification.

# Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores del trabajo final de grado, José Antonio Gil Gómez e Inmaculada García García, por permitirme realizar un proyecto relacionado con la asignatura que me impartieron bajo su tutela. También me gustaría agradecer a mi madre y mi hermano por quitarme todas las cargas que les ha sido posible de encima para permitirme realizar este trabajo.

# Índice

# Capítulo 1: Introducción

## 1.1. Objetivo y justificación del trabajo

El objetivo de este trabajo es el desarrollo de una aplicación para dispositivos móviles con un sistema operativo Android versión 9 o superior que sea de ayuda para la organización interna de empresas pequeñas o medianas. La aplicación ha de ser intuitiva, es decir, fácil de usar para usuarios que tengan nociones de la función que pretende desempeñar la aplicación. Ha de ayudar al usuario con su gestión, siendo capaz de automatizar cambios en el inventario de la empresa y siendo centro de información para todo lo pertinente a la maquinaria y el inventario de la empresa.

La idea de este trabajo surgió gracias a la asignatura de desarrollo de aplicaciones para dispositivos móviles de la Universidad Politécnica de Valencia, que proporcionó experiencia con el desarrollo de aplicaciones móviles en Android, y a la experiencia personal del autor de este trabajo formando parte de una empresa pequeña. El desarrollo de este proyecto en forma de aplicación móvil es una forma de crear un producto accesible para todo tipo de personas que puedan plantearse formar su propia empresa facilitando esta importante decisión y ayudándoles a lo largo del camino.

Se decidió desarrollar el producto en Android debido a su facilidad de uso, transparencia del funcionamiento del sistema operativo y por la experiencia ofrecida por la *Universidad Politécnica de Valencia*. Sin embargo, cabe destacar que Google Play Store es estadísticamente la mejor plataforma para publicar una aplicación si se desea que esta sea lo más accesible posible y Android el sistema operativo para móviles más extendido en España. Según la *Comisión Nacional de los Mercados y la Competencia*, “El 78,8 % de los ciudadanos con smartphone prefirieron Android frente al 16,3 % que escogió iOS”(Comisión Nacional de los Mercados y la competencia, 2023)[1] .

## 1.2. Metodología de trabajo

Desarrollar cualquier tipo de proyecto de programación supone un reto de organización para el desarrollador debido a la enorme cantidad de ficheros distintos con centenares o miles de líneas de código que se pueden generar, dependiendo del tamaño del proyecto. No mantener un entorno digital organizado puede suponer un retraso

enorme en el tiempo de desarrollo y un amento en la fatiga del programador, debido a tener que tratar con problemas fácilmente evitables.

Por este motivo, se ha decido trabajar con un sistema de control de versiones que permite, con facilidad, guardar el progreso del desarrollo en la nube y poder acceder a instancias anteriores del código que hayan sido subidas. También permite crear una ramificación en alguna instancia de código subida anteriormente para experimentar con soluciones alternativas sin tener que borrar las soluciones iniciales y otras partes del código que entrarían en conflicto. Por último, su uso más útil a la hora de experimentar con el código e intentar hallar la mejor solución a un problema es que, al conectar Android Studio [2] con el sistema de control de versiones, se marcan todos los cambios realizados en el proyecto. De esta forma, es difícil perder la línea de código editada entre las miles de líneas totales.

En este caso, se ha decidido usar *github.com* como servidor para colocar el repositorio del proyecto, ya que este servicio sirve para llevar el control de versiones de proyectos de programación de forma gratuita.

### 1.3. Estructura de la memoria

Inmediatamente después de este capítulo, en el capítulo 2, se realizará un análisis del estado del arte de productos similares y se señalará la necesidad del público que se pretende cumplir. Así se justificará el desarrollo de la aplicación.

A continuación, en el capítulo 3, se procederá a explicar cuál es la visión del producto final y todas las funciones que debe ser capaz de desempeñar. Estas especificaciones son las que han sido usadas a modo de guía en el desarrollo del producto.

Seguidamente, en el capítulo 4, se procederá a exponer las decisiones tomadas en cuanto al diseño visual y la colocación de los distintos elementos en la pantalla. En esta explicación se justificará el motivo por el que cada decisión ha sido tomada.

En el último de los capítulos de desarrollo del producto, el capítulo 4, se comenzará con una breve explicación de términos simples que podrían ser mencionados frecuentemente a lo largo del resto del capítulo, para que los lectores menos familiarizados con conceptos relacionados con el ámbito de la programación puedan ser capaces de comprender el capítulo. El resto del capítulo consistirá en una explicación de los distintos elementos que han sido claves para desarrollar la aplicación de forma deseada y como han sido aplicados o desarrollados.

El capítulo 6 tratará sobre en análisis del cumplimiento de los requisitos que debe tener la aplicación para cumplir las especificaciones y funcionalidad expuestas en el capítulo 2. Esto se hará justificando que características implementadas cumplen cada requisito concreto.

En el capítulo 7 se llevará a cabo un presupuesto simbólico para demostrar cuanto costaría monetariamente realizar una aplicación como esta.

En el capítulo 8 se expondrán las conclusiones obtenidas de la realización de este trabajo, las posibles mejoras futuras que se podrían implementar en la aplicación y como se relaciona la aplicación con los objetivos de desarrollo sostenible de las Naciones Unidas que forman parte de la agenda 2030.

# Capítulo 2: Estado del arte

En este capítulo se pretende las herramientas disponibles en el mercado que permiten a las empresas llevar a cabo un manejo eficiente de sus activos. Con esto se pretende justificar el desarrollo de una aplicación que busca ofrecer una gestión gratuita y simple a la vez que se diferencia de otras aplicaciones similares.

## 2.1. Contexto histórico

Antes de la creación e integración en masa de los sistemas digitales en la sociedad la gestión del inventario debía haberse manejado de forma manual, ya que la dicha gestión se remonta a la época de las primeras civilizaciones, destacando en la civilización egipcia, como afirma Javier Jara, año desconocido [7]. Posteriormente, a medida que ha avanzado la tecnología, se han formado métodos más eficientes de gestión, como lo son las hojas de cálculo que existen de forma pública desde el año 1972, como afirma la editorial Etecé, 2023, [8] y presentaron un gran avance en la eficiencia de todos los aspectos empresariales, no solo en la gestión del inventario.
Actualmente, con la extensión masiva de tecnologías modernas, se dispone de herramientas de gestión capaces de automatizar diversos procesos del manejo del inventario, desde predecir la fluctuación diaria este, hasta la contabilización real de todos los activos de una empresa.

## 2.2. Revisión de soluciones existentes

### 2.2.1. Soluciones de pago

Al investigar aplicaciones para la gestión empresarial, uno puede encontrarse con una gran cantidad de productos, que intentan llamar la atención del usuario con videos y animaciones destacando la gran capacidad de gestión de su producto. Seguidamente suelen presentar algunas opciones al usuario como comprar el producto, solicitar una demo, probar una versión gratuita temporal o solicitar una reunión personal. Algunas empresas que hacen esto son Fracttal [9], Fiix [10] y Timly [11], que han sido contratadas por grandes empresas como KFC, Toyota y Siemens.

Esta clase de soluciones tienen una gran eficiencia, pero están enfocadas a empresas medianas y grandes. Una empresa pequeña no solo no necesita un nivel de gestión tan extremo, sino que, además, es muy posible que no sea capaz de permitírselo siquiera.

### 2.2.2. Soluciones gratuitas

Existen diversas soluciones gratuitas que también son usadas por empresas de gran tamaño. Algunas de estas soluciones son ofrecidas por empresas como Pipedrive [12] y HubSpot [13]. Estas soluciones son aptas para cualquier tipo de empresa ya que no requieren de ningún coste, sin embargo, podrían no ser del agrado de todo el mundo. Estas soluciones son completas y abarcan muchos más aspectos de la gestión de una empresa que el manejo del inventario o de la máquina. Para una empresa pequeña que intenta comenzar en el sector pueden resultar abrumadoras y gran cantidad de las características que implementan podrían ser irrelevantes hasta que la empresa crezca.

## 2.3. Conclusiones

Todas las soluciones mencionadas anteriormente operan principalmente en web con vinculación con la nube, aunque algunas disponen de versión de móvil. Ninguna de ellas permite una gestión simple y totalmente offline. Existe una vacío en el sector. Una solución enfocada a las empresas que están comenzando y que pueda seguir siendo útil si la empresa crece. Una solución que esté siempre al alcance del usuario en cualquier momento. Ese hueco será el que intente llenar el producto de este trabajo.

# Capítulo 3: Requisitos de la aplicación

En este capítulo se enumerarán todos los requisitos que debe de cumplir la aplicación móvil sin entrar en detalle sobre cómo se deben de cumplir, en concreto. Estos requisitos son los que se han seguido a la hora de tomar todas las decisiones en cuanto al desarrollo de la aplicación.

## 3.1. Requisitos generales

Al desarrollar la aplicación para intentar cumplir, en la medida de lo posible, con todos estos requisitos uno de los factores más importantes es que sea de uso intuitivo para el usuario, ya que una aplicación que requiera de mucho esfuerzo mental para usarse contrapesaría la comodidad de tener un gestor para tu empresa en el bolsillo.

La aplicación debe de funcionar sin ningún error que provoque el cierre automático de la aplicación, ya que si eso ocurriese no se podría considerar que la aplicación está lista para lanzarse al público.

## 3.2. Requisitos de la vista principal

La aplicación debe presentar, de forma inmediatamente accesible al usuario, un registro visualmente agradable y poco abarrotado de todas las máquinas y todos los objetos que formen parte de su inventario de los que se dispone. Esto se debe hacer de forma dividida sin que haya ningún tipo de confusión entre que elemento pertenece a cada lista.

Todos los elementos mostrados deben permitir al usuario diferenciarlos y asociarlos a su contraparte real en un vistazo rápido. De esta manera el usuario no tiene que gastar su tiempo en buscar y distinguir los elementos, lo cual podría provocar que la aplicación sea incomoda de usar.

En caso de que la cantidad de elementos a mostrar sea demasiado grande la aplicación deberá tener una o varias formas para que el usuario busque el elemento al que desea acceder sin tener que realizar una búsqueda manual uno a uno.

El usuario debe poder añadir un nuevo elemento, ya sea una máquina o un objeto de inventario, y que el cambio se muestre inmediatamente.

Todos los elementos tienen que poder eliminarse desde la vista principal para evitar que el usuario tarde mucho en eliminar varios de ellos a la vez, pero la opción de eliminarlos debe de ser lo suficientemente inaccesible como para que no sea pulsada por accidente.

La cantidad de cada objeto de inventario de la que el usuario dispone en el momento debe de ser mostrada de forma accesible y debe de poderse editar cómodamente en caso de que el usuario gaste, venda o adquiera cualquier cantidad.

### 3.3. Requisitos comunes para máquinas y objetos de inventario

Todos los elementos deberán permitir al usuario acceder a una vista con más detalles sobre el elemento. En esta vista también ha de haber una opción para eliminar el elemento, pero más accesible.

La aplicación deberá poder notificar de alguna forma al usuario cuando algún evento importante programado por él mismo vaya a ocurrir ese día. El usuario tiene que poder programar y manejar estos eventos de forma cómoda. Los eventos deberán de poder programarse tanto para fechas puntuales como para intervalos recurrentes de tiempo.

El usuario tiene que poder subir todo tipo de información pertinente sobre cada objeto de inventario o sobre cada maquinaria para poder usar la aplicación como un centro de información centralizado y organizado para los elementos de su empresa.

### 3.4. Requisitos para objetos de inventario

Debe de poderse automatizar la alteración de la cantidad restante de cada objeto de forma puntual en una fecha concreta o de forma recurrente cada cierto tiempo. Por ejemplo, si el usuario recibe un cargamento de cien sillas de madera cada 15 días la aplicación deberá sumar cien a la cantidad de sillas de madera sin que el usuario tenga que abrir siquiera la aplicación de tal forma que cuando este decida abrirla se encuentre con una cantidad de sillas actualizada y correcta.

# Capítulo 4: Diseño de la aplicación

En este capítulo se hablará sobre el desarrollo del diseño de la aplicación y como ha cambiado durante la duración del proyecto. También se justificará porque se ha decidido quedarse con el diseño final. Todas las imágenes e iconos usados en el diseño final de la aplicación son o imágenes ya disponibles por defecto en Android Studio o imágenes generadas el generador de imágenes de Microsoft Bing y posteriormente editadas manualmente.

Se ha tomado esta decisión ya que según los términos de uso de esta inteligencia artificial "Debe usar el Generador de imágenes y las Creaciones generadas únicamente (i) de manera legal y en cumplimiento de toda la legislación aplicable."(Bing, 2023)[3] Aún si las imágenes fueran sujetas a derechos de autor al nombre de Microsoft podría seguir usándolas, pero según el artículo 5 de la ley de propiedad intelectual española "Se considera autor a la persona natural que crea alguna obra literaria, artística o científica."(Estado Español, 1996)[4] Esto no contempla a la inteligencia artificial actualmente ya que no es una persona natural.

## 4.1. Primeros bocetos

Antes de programar nada es importante tener una guía de que se intenta programar. Por ello es importante empezar por el diseño de la aplicación. Para poder crear un diseño inicial se debe tener una buena idea de que función busca desempeñar cada elemento que quieras introducir en la aplicación.

El diseño inicial se hizo con algunas de los requisitos establecidos en el capítulo dos. De esta forma se esperaba que el diseño fuera lo más parecido posible al diseño final y así minimizar el tiempo gastado en cambios que pudieran afectar a gran parte del contenido de la aplicación.

Se usó una herramienta de código abierto llamada *Pencil Project* [5] para hacer el primer diseño ya que disponía de elementos visuales típicos de aplicaciones móviles para arrastrar a un lienzo en blanco y crear un diseño conceptual rápido.

### 4.1.1. Vista Principal

Primero se planteó como sería la vista principal. Para poder visualizar las máquinas y los objetos de inventario sin confundirlos los unos con los otros era necesario separarlos de forma muy obvia. Se optó por dividir la lista de máquinas y la de objetos en dos pestañas diferentes de tal forma que solo una de las listas estuviera presente en un momento dado, pero que fuera cómodo cambiar entre ambas listas. En la misma vista principal se decidió implementar una barra de búsqueda que ordenara ambas listas dependiendo de que se escribiera para facilitar al usuario la búsqueda de un elemento concreto. En la esquina inferior derecha se puso un botón con una cruz para añadir un nuevo elemento, ya que es la posición más cercana al pulgar al coger el móvil con la mano derecha. Debido a que la mayoría de la población es diestra la posición del botón a la derecha será cómoda para la mayoría de los usuarios.

Para poder cumplir con el requisito de distinguir los elementos rápidamente y asociarlos a su contraparte real se decidió que se pudiera visualizar una imagen en cada elemento de cada lista. Se tenía pensado que el usuario pudiera usar cualquier imagen deseada de su galería para cada elemento y que así pudiera distinguir los elementos rápidamente al estar familiarizado con la imagen de antemano. Al hacer el boceto inicial para facilitar su diseño y debido a su naturaleza orientativa se decidió que en lugar de imágenes se pondrían simplemente cuadrados azules o iconos circulares.

Se decidió que la lista de objetos de inventario sería una lista de una sola columna debido a que así se asemeja a una lista escrita a mano. Como las máquinas no tienen una cantidad asociada se le dio más peso visual a la imagen y eso hacía que ocupará demasiado espacio de la pantalla. Por ello, al contrario que con la lista del inventario, la lista de máquinas se decidió hacer en dos columnas.

Por último, para poder editar fácilmente de forma manual la cantidad de cada objeto del inventario se diseñó un desplegable simple con dos botones, uno para sumar y otro para restar, y con un cuadrado de texto en el que se podía escribir la cantidad a sumar y a restar.

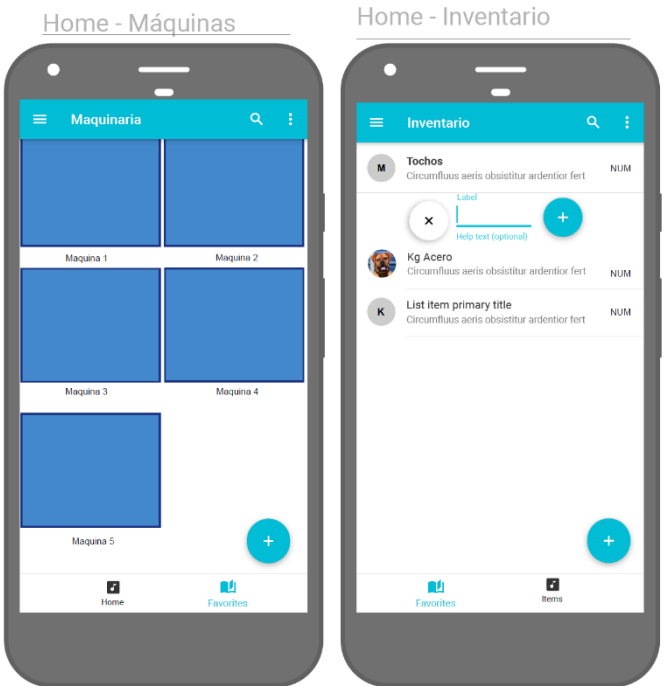


*Ilustración 1 - Boceto original de la vista principal.*

### 4.1.2. Vista de detalles

A continuación, se diseñó la vista de detalles. El objetivo de esta vista es tener una sección de la aplicación única dedicada a cada elemento que el usuario introduzca en ella. En esta vista se tenía planeado que se mostrara una versión más grande de la imagen del elemento, el nombre, una breve descripción, la lista de eventos propios de la máquina o de el objeto de inventario y un botón para ir a otra vista que permitiera editar los datos. El botón de editar se colocó en la parte inferior izquierda para que el usuario no lo relacionara subconscientemente con el botón de añadir nuevo elemento que estaba situado en la sección inferior derecha de la vista principal.

En la lista de eventos se colocaron casillas para indicar si esos eventos estaban activos o no y que el usuario pudiera activar y desactivar eventos fácilmente. Al igual que con la vista principal se ha usado un cuadrado azul como sustituto de una imagen real. Esta vista es exactamente igual para los objetos de inventario como para las máquinas.


*Ilustración 2 - Boceto original de la vista de detalle.*

### 4.1.3. Vista de elemento nuevo

Esta vista es posiblemente la más compleja de todas. En esta vista se tiene que poder declarar toda la información que vaya a estar visible en cualquier otra vista, por lo que es la vista que más abarrotada está. Para permitir que el usuario suba su propia imagen se pensó en poner una imagen con texto que incitara al usuario a pincharla, lo cual abriría un menú para subir la imagen. Esto en el boceto inicial se señalizó con un recuadro azul y un símbolo de suma.

A su derecha se decidió poner un campo para rellenar el nombre del elemento a añadir. Para la descripción se hizo lo mismo justo debajo de la imagen y en la parte inferior de la vista se añadió un recuadro para añadir etiquetas. Estas etiquetas tenian pensado usarse para la búsqueda de la vista principal y así poder agrupar varios elementos en una búsqueda común (por ejemplo, al buscar “metal” se muestran todas las máquinas que se usan para trabajar con metal).

Esta vista sí que es distinta para máquinas y objetos de inventario. Esto se debe a que se pensó que los eventos de objetos de inventario deberían especificar la cantidad

del objeto que se va a sumar o a restar cuando el evento ocurra. Las máquinas, al no tener una cantidad asociada, no requerirían de ese apartado.

El planteamiento detrás de los eventos en esta vista era que hubiera un botón que al pulsarlo añadiera a la vista un nuevo evento en blanco listo para rellenar. Los datos a rellenar en este evento a parte de los mencionados anteriormente serían el día en el que tendría lugar el evento, una casilla para marcar si es recurrente o no, un recuadro para marcar la magnitud de cada cuanto tiempo se repite y un selector para marcar la unidad de tiempo asociada a la magnitud (días, meses o años) y un recuadro para escribir una breve descripción del evento. Por último, un botón para confirmar el elemento nuevo.

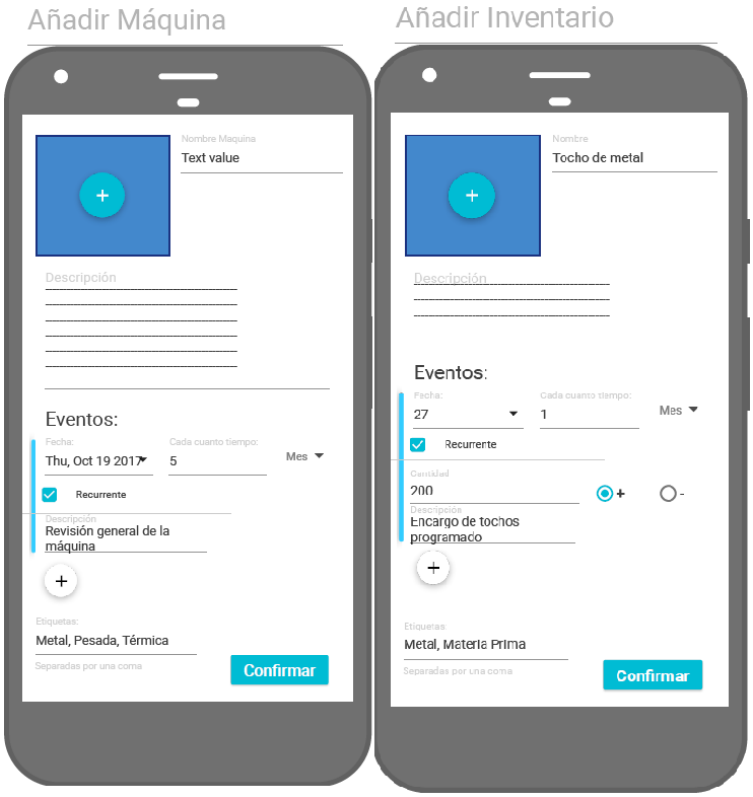


*Ilustración 3 - Boceto original de la vista de nuevo elemento.*

## 4.2. Desarrollo del diseño en Android Studio

Después de crear el boceto inicial ya existía una idea lo suficientemente específica como para intentar recrearla en Android Studio, lo cual no resulta tan sencillo como el diseño simplemente visual. Aunque a veces también se puede simplemente arrastrar un elemento, como un botón, a la vista y decidir donde debería ir en Android Studio hay interacciones más complejas que, aunque no condicionan mucho el diseño, lo acaban dificultando y provocando que el proceso de diseño se alargue.

### 4.2.1. Planteamiento de la vista principal

Al igual que el proceso de diseño del boceto, el diseño del producto comenzó por la vista principal. Debido a que algunos elementos de diseño tienen un comportamiento complejo que requería de estudio previo a intentar usarlos se tuvieron que priorizar los elementos más imprescindibles para el funcionamiento de la aplicación. Se comenzó con la implementación de las pestañas.

Al desconocer como podría funcionar la sección superior de la vista principal que se tenía planeado que mostrara “Maquinaria” o “Inventario” dependiendo de la pestaña actual se decidió simplemente escribir estas palabras en el divisor de cada pestaña, prescindiendo así de iconos para diferenciar las pestañas.

Una vez creadas las pestañas había que encontrar una forma de relacionar la selección de pestaña con lo que le fuera mostrado en pantalla al usuario. Para hacer esto se decidió usar un elemento llamado *viewPager2* que se podría usar para este propósito. Encima del viewPager2 se dejó un hueco para dejar espacio a la barra de búsqueda para un futuro. Dentro del viewPager2 se mostrarían otras vistas que corresponden con las listas de máquinas e inventario y se puede cambiar entre ambas simplemente arrastrando la pantalla de lado a lado.

Con el objeto de que mover todos los elementos en pantalla hacia fuera de la pantalla no desorientara al usuario se decidió poner un único botón para añadir un nuevo elemento a cada lista. Este botón formaría parte de la vista principal y no de las vistas interiores del viewPager2, de tal forma que al desplazarse entre las dos listas el botón se mantendría estático en la pantalla. Como funcionalidad añadida al desplazarse verticalmente por los elementos de una misma lista el botón mantendrá su posición permitiéndole al usuario añadir cómodamente un elemento nuevo en cualquier momento.

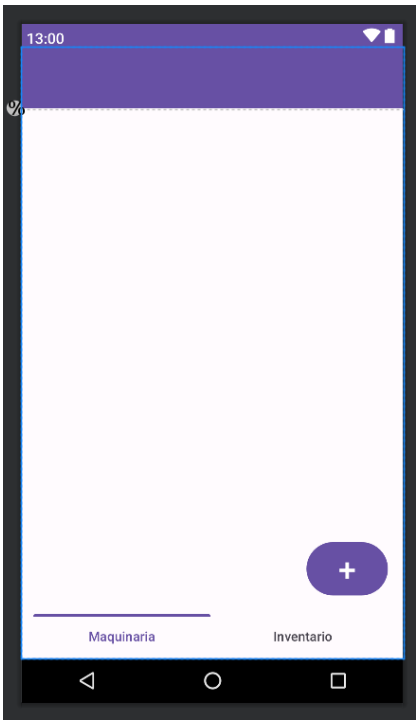


*Ilustración 4 - Primer diseño de la vista principal.*

A lo largo de la realización del proyecto se completó el diseño de la vista principal mediante la implementación de la barra superior. Por defecto al crearla en android studio el texto que se muestra es el título de la aplicación. Ya que ya no se requería de indicar que lista se estaba presenciando en cada momento se decidió mantener así. Además, en esta barra se acabó implementando el sistema de búsqueda. En vez de ser una simple barra de búsqueda se decidió por implementar un botón con forma de lupa el cual permite al usuario abrir la barra de búsqueda real. Esto se hizo así por dos motivos principales. El primero es la barra superior queda menos abarrotada y, por ende, más limpia. El segundo motivo es que si la barra de búsqueda estuviera abierta siempre no habría espacio para el texto que indica el nombre de la aplicación. Muchas aplicaciones y páginas web mundialmente conocidas como Youtube, Google y Spotify tienen algún indicativo de su marca en la parte superior izquierda de la aplicación, así que siguiendo estos ejemplos se decidió que era importante no tapar el nombre de la aplicación.

Como último detalle al diseño general de la aplicación los colores por defecto se personalizaron tanto para el tema oscuro como para el tema claro para así darle un poco de personalidad a la aplicación.

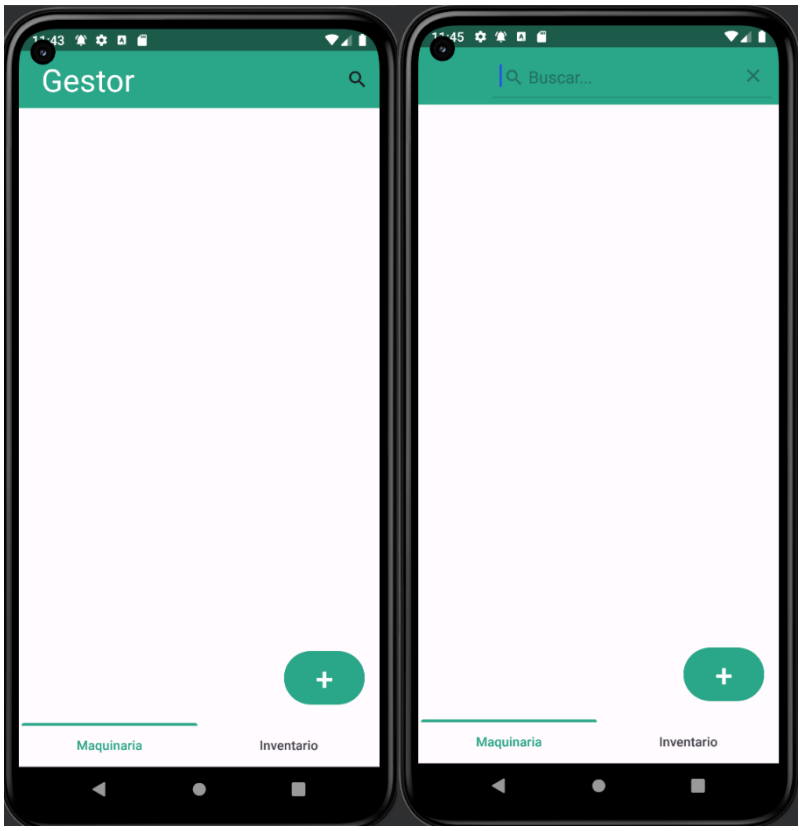


*Ilustración 5 - Diseño final de la vista principal.*

### 4.2.2. Elementos de la lista de máquina

Para poder mostrar los elementos de la vista de máquinas de forma dinámica es necesario hacer uso de un elemento llamado *RecyclerView*. Este elemento es un tipo de vista que permite mostrar un diseño gráfico por cada elemento existente en una lista interna. Colocando esta herramienta dentro del viewPager2 mencionado anteriormente podemos visualizar las listas internas de máquinas e inventario de forma similar a la idea del boceto inicial. Las únicas piezas restantes para que la visión inicial sea alcanzada son los diseños de los elementos individuales de las listas.

Para hacer el diseño de los elementos de la lista de máquinas se ha seguido la idea del boceto de tal forma que los únicos elementos en el diseño sean la imagen y el nombre de la máquina. Para hacer esto se usó un tipo de disposición llamada *LinearLayout (vertical)* y que permite que automáticamente todos los elementos añadidos se coloquen uno encima del otro. Tan solo se tuvo que editar las posiciones dentro de la misma vertical. Ambos elementos se colocaron en el centro del diseño.

Como este diseño será mostrado junto con la vista principal se implementó una característica que permite al usuario borrar la máquina rápidamente. La característica está lo suficientemente escondida como para que no sea usada por accidente pero al ser común en otras aplicaciones cabe de esperar que el usuario pueda hallarla. Esta característica provoca que se abra un recuadro con la opción de eliminar la máquina si se mantiene pulsada la imagen de esta.

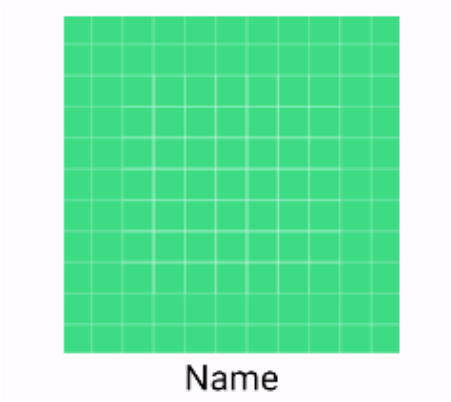


*Ilustración 6 – Diseño final de la visualización de una máquina.*

### 4.2.3. Elementos de la lista de inventario

Al igual que con el diseño de las máquinas se intentó seguir el boceto original para crear el diseño de los objetos de inventario. Originalmente había cuatro elementos principales que incluir en el diseño. Estos son la imagen del objeto, el nombre, una breve descripción y la cantidad del objeto de la que dispone el usuario.

Para crear este diseño se usó una disposición distinta a la mencionada anteriormente. Esta disposición se llama *RelativeLayout* y permite colocar todos los elementos con una posición relativa el uno del otro y relativa a los extremos de la pantalla o de la vista. Debido a limitaciones de esta disposición al ser usada con la RecyclerView mencionada anteriormente si se declara la posición de algún elemento a la parte inferior de la disposición cada elemento de la lista del inventario ocupará una pantalla de móvil entera, por lo que todos los elementos están colocados de forma relativa a los extremos derecho, izquierdo y superior de la disposición. Por último, la altura de la disposición está ajustada para que sea la justa para contener a todos los elementos en su interior. Si este ajuste no se realizara el mismo problema mencionado recientemente también ocurriría.

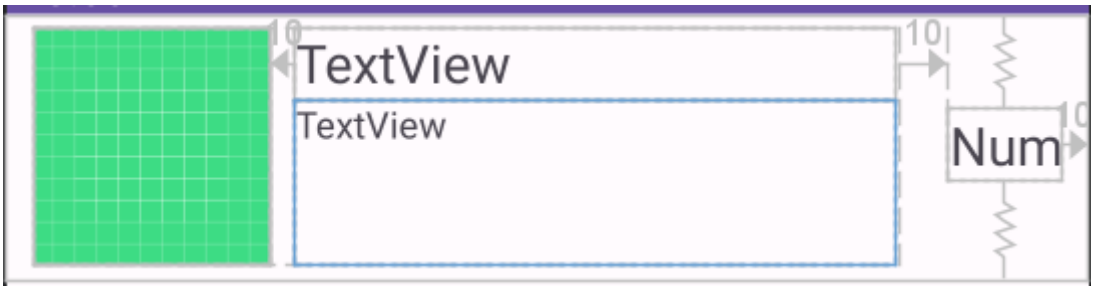


*Ilustración 7 - Primer diseño de la visualización de un objeto de inventario.*

A lo largo del proceso de desarrollo del proyecto completo este fue uno de los diseños que más se tuvo que retocar. Cuando se desarrolló el primer boceto se pretendía que para ir a la vista que enseña los detalles de un elemento concreto el usuario debía de pinchar en el objeto desde la vista principal. Esto entra en conflicto con la idea de que para editar el número que indica la cantidad del objeto de la que dispone el usuario se pueda abrir un desplegable que permita editar el número, ya que también habría que presionar el objeto desde la vista principal.

La primera solución que se planteó fue darle prioridad a la característica de poder editar la cantidad disponible de un objeto. Para hacer esto se añadieron tres elementos al diseño ya creado. Estos tres elementos eran los mismos que estaban presentes en el desplegable planteado en el boceto inicial; dos botones y un recuadro para escribir un número. Los elementos del desplegable se programaron para que por defecto fueran invisibles y no ocuparan espacios.

La idea era que al seleccionar un objeto de la lista de inventario se abriera automáticamente el desplegable por debajo, haciendo los elementos visibles, y que al seleccionar cualquier otro objeto se cerrara el primer desplegable y se abriera el nuevo. Esto planteaba un problema debido a que se estaba haciendo uso de una variable que representaba todos los elementos del diseño a la vez para abrir el desplegable. Esto incluía al propio desplegable, lo que hacía imposible pulsar los botones. La forma de solventar este problema sin afectar al diseño actual fue poner una imagen sin ningún recurso cargado enfrente de la sección no invisible por defecto y hacer que al pulsar esta imagen se abriera el desplegable.

Esta solución era viable para el propósito de editar la cantidad de cada objeto de inventario, creaba un nuevo problema para poder abrir la vista de detalles de forma cómoda. Se planteó hacer que se pudiera abrir de la misma forma que el desplegable, pero manteniendo pulsado en vez de pulsar una única vez. Esta idea se desechó rápidamente debido a que en la lista de máquinas el sistema para abrir la vista de detalles seguía siendo una simple pulsación en cada elemento, además de que no era intuitivo de probar para el usuario.

El reto era implementar ambas características de forma intuitiva y cómoda para el usuario sin que se diferenciara mucho de cómo funciona la apertura de los detalles en la lista de máquinas. Finalmente, se planteó una solución final. Esta sería hacer que dependiendo de donde pulsara el usuario en un objeto de inventario se abriera la vista de detalles o el desplegable para editar la cantidad. La imagen invisible que abría el desplegable se movió para que solo ocupara una sección alrededor del número que representa la cantidad del objeto y se creó otra imagen invisible encima del resto de los elementos que al pulsarla abriera la vista de detalles. Además, al mantenerla pulsada también abriría un recuadro con la opción de eliminar el objeto de inventario, de la misma manera que ocurre con las máquinas.

Este planteamiento servía como solución cómoda, pero faltaba hacer que fuera intuitivo para el usuario que dependiendo de donde presionara la aplicación respondería de formas distintas. Para asegurarse de que así fuera se creó una división visual entre las dos imágenes invisibles con una barra azul. Aún no parecía suficiente, por lo que se cambió el texto de la cantidad del objeto para que fuera un botón con un color sólido en forma de cuadrado rodeando el número.

Como último detalle se hizo que el desplegable tuviera con color gris oscuro para que no se camuflara con el resto de los elementos de la lista.

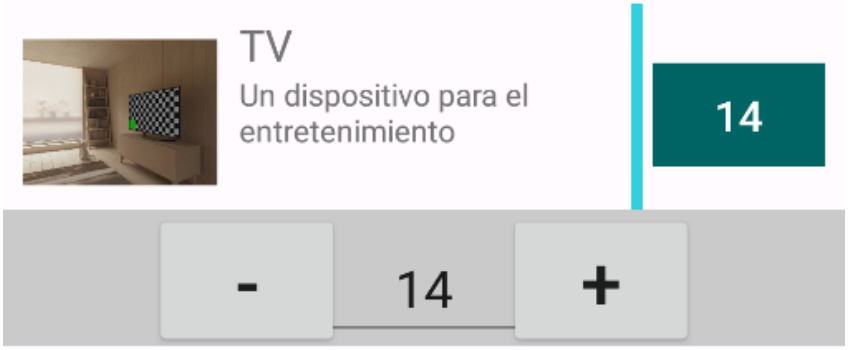


*Ilustración 8 - Diseño final de un objeto de inventario.*

### 4.2.4. Diseño de la vista de detalle

La vista de diseño se empezó a crear ya con un conocimiento previo de cómo tratar con una RecyclerView, la cual esta vista necesitaría para asemejarse al boceto inicial. La RecyclerView sería usada para mostrar una lista de los eventos de la máquina y permitir al usuario activarlos y desactivarlos a voluntad. El problema que nos plantea usar una RecyclerView en este caso es que, aunque la RecyclerView permite poder desplazarse a lo largo de los elementos que tenga dentro, no se desplaza junto con

elementos externos a la RecyclerView. El efecto resultante es tener una gran cantidad de elementos estáticos en la pantalla mientras que la lista de eventos se puede mover a voluntad.

Esto podría no ser considerado un problema para algunas aplicaciones, pero si toda la pantalla se desplaza junto con el RecyclerView el usuario puede emplear toda la pantalla en buscar el evento que quiera en caso de tener una cantidad muy elevada de ellos. Para provocar que el desplazamiento de toda la vista de detalles funcione en conjunto se pueden meter todos los elementos dentro de una *NestedScrollView*, cuyo propósito principal es precisamente cambiar este comportamiento.

Por lo demás la vista se diseñó dentro del NestedScrollView de forma normal. Se usó una disposición de LinearLayout (vertical) para poner todos los elementos rápidamente uno debajo del otro. Siguiendo la guía del diseño original los elementos en orden de arriba a abajo eran: una versión aumentada de la imagen de objeto cuyos detalles se están leyendo, el nombre del objeto, un texto indicando que lo siguiente era la descripción, la descripción en sí, un texto indicando que lo siguiente eran los eventos y la RecyclerView para enseñar los eventos.

Aunque el botón para editar los datos se colocó originalmente en la sección inferior izquierda de la pantalla para diferenciarse del botón de añadir elemento en la vista principal, después de usar la aplicación resultó ser más cómodo tener el botón también en la sección inferior derecha, ya que ya estaba preparado para pulsar un botón en esa posición. El botón de editar se colocó fuera de la NestedScrollView para que en vez de encontrarse al fondo del desplazamiento de la vista siempre se hallara en la esquina inferior derecha de la pantalla de forma estática.


*Ilustración 9 - Primer diseño de la vista de detalle.*

A medida que se desarrollaba el producto final esta vista ha sufrido ligeros cambios. En concreto se añadieron tres elementos nuevos. Uno de ellos es una imagen de una flecha apuntando hacia la izquierda para volver a la vista principal. Aunque este elemento es totalmente prescindible debido al botón de Android que ya se encarga de cumplir dicha función puede haber usuarios que no tengan costumbre de usar esta función o tengan los controles del móvil cambiados de alguna forma que dificulte usar el botón de Android. La flecha está colocada en la parte superior izquierda de la pantalla debido a que en las aplicaciones móviles suele ser un estándar para indicar la acción de volver atrás.

Otro elemento añadido ha sido una imagen de una papelera roja cuyo propósito es borrar el objeto perteneciente a la lista de máquinas o la lista del inventario por completo. Para dificultar al usuario el realizar la eliminación por accidente al pulsarse se muestra un recuadro de alerta en pantalla pidiendo al usuario que confirme la eliminación.

El último elemento es un icono de un documento siendo escrito en la parte izquierda de la imagen principal. El objetivo de este icono es llevar al usuario a una vista que no había sido contemplada en el diseño del boceto original, la cual será descrita más adelante.


*Ilustración 10 - Diseño final de la vista de detalle.*

### 4.2.5. Diseño de los eventos en la vista de detalles

Este es el uno de los diseños más sencillos de todo el proyecto. La idea original consistía en que cada elemento de la RecyclerView de la vista de detalles consistiera en el título o nombre del evento y una casilla para activar o desactivar el evento, así es como se planteó en el boceto original.


*Ilustración 11 - Primer diseño de la visualización de un evento*

Al final del proyecto resultó que la descripción asignada a cada evento solo era visible en la pestaña que permitía editarla, por lo que se consideró alterar este diseño para que incluyera también la descripción del evento. Además, de esta forma al usuario le sería más fácil distinguir unos eventos de otros cuando quisiera activar o desactivar alguno.


*Ilustración 12 - Diseño final de la visualización de un evento.*

### 4.2.6. Diseño de la vista dedicada a añadir una máquina y objeto de inventario nuevo

A pesar de que esta vista fue ideada para simplemente añadir nuevos elementos a las listas de máquinas o inventario también ha sido usada desde el principio del desarrollo del proyecto para editar elementos ya existentes en ambas listas, ya que la información que se le tiene que permitir al usuario editar es la misma que se le permite declarar en la creación de un elemento. Esta vista es la más similar al boceto original. Desde que se desarrolló por primera vez la vista no ha sufrido ningún cambio mayor. Sin embargo, el sistema de creación de eventos del que se hablará a continuación sí que es bastante distinto a la idea original. Este sistema es lo que se mostrará dentro del RecyclerView que hay en la vista tratada en este apartado.

Para realizar esta vista se han tomado decisiones similares a las de la vista de detalle, haciendo uso de un NestedScrollView para provocar que el desplazamiento de la RecyclerView esté asociado al desplazamiento del resto de la vista. Los elementos más importantes de esta vista son la imagen que al pulsarla permite al usuario subir su propia imagen a la aplicación, el recuadro para rellenar el nombre del objeto que está siendo creado, un recuadro similar para la descripción, la RecyclerView, el botón que permite añadir eventos nuevos a la RecyclerView y el recuadro para añadirle etiquetas al objeto siendo creado.

Se decidió que esta vista no dispondría de ninguna flecha para volver a la vista anterior ya que ocuparía demasiado espacio y el usuario ya tiene otras formas de salir de aquí. Siempre puede usar el botón para volver atrás de Android o confirmar la creación del objeto y luego borrarlo si no lo quiere conservar.

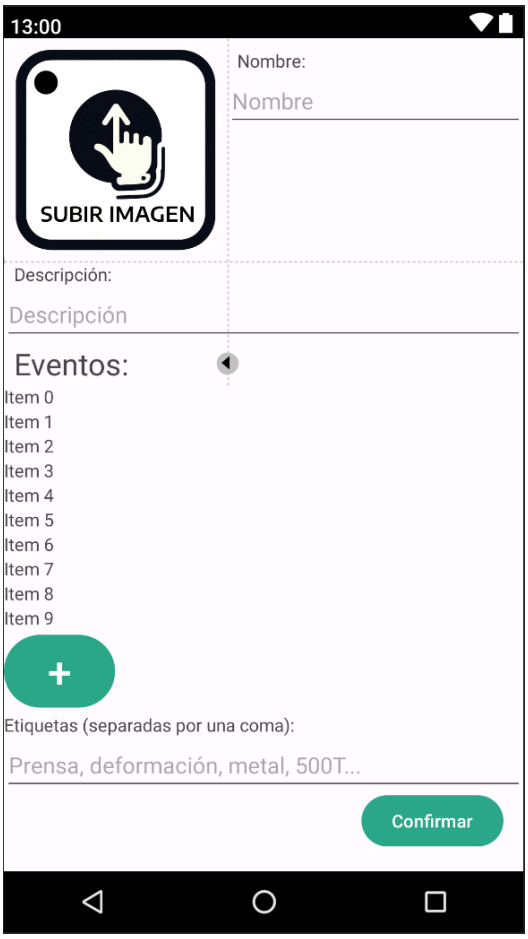


*Ilustración 13 - Diseño final de la vista de nuevo elemento/editar elemento.*

### 4.2.7. Diseño de los eventos nuevos

El objetivo de este diseño es permitir al usuario ajustar todos los parámetros importantes que debe de tener un evento tanto si es para avisar de una fecha de importancia en relación con alguna máquina como si es para avisar de un nuevo cargamento de piezas. Inicialmente se hizo un único modelo de eventos nuevos tanto para las máquinas como para los objetos de inventario. Los elementos incluidos en este diseño inicial eran el nombre del evento, la fecha en la que se llevaría a cabo, una casilla para marcar si el evento es recurrente o no, la magnitud de cada cuanto se repetiría, un selector para decidir la unidad de cada cuanto se repetiría y la descripción del evento.

La idea era que, para ahorrar espacio, evitar que el usuario rellenara información inútil y minimizar la confusión que pudiera causar tener tanta información que rellenar en pantalla, los datos a rellenar sobre cada cuanto se repite el evento no fueran visibles si la casilla de recurrencia no estaba marcada. Para seleccionar la fecha en la que se

llevaría a cabo el evento sería muy complejo pedirle al usuario que introdujera la fecha en un formato concreto, por lo que en vez de eso se hace uso de los calendarios de Android y al intentar introducir la fecha se muestra un calendario en el que el usuario puede marcar una fecha concreta.

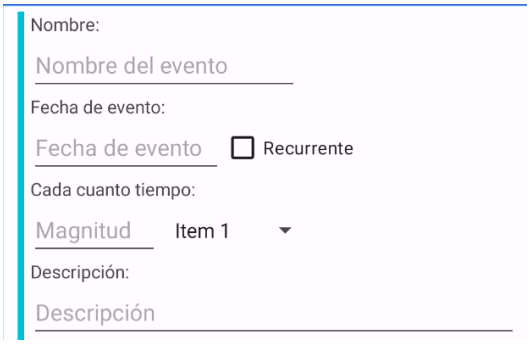


*Ilustración 14 - Diseño inicial de la visualización de relleno de datos de un evento.*

El único cambio significativo por el que pasó este diseño a lo largo del proyecto fue la introducción de una cualidad que ya había sido planteada en el boceto original. Esto era la habilidad de escribir una cantidad en una casilla y marcar con un interruptor si el evento debía de sumar esa cantidad o de restarla. Ese cambio solo se debería de aplicar a los eventos de objetos de inventario y no a las máquinas, sin embargo, el diseño anterior ya estaba siendo usado para ambas listas. Por ello se decidió simplemente añadirlo y ocultar esos apartados si el objeto a tratar es de la lista de máquinas.

Se implementó otro cambio para mejorar la calidad de vida de la aplicación. Este es un botón en forma de equis de color rojo que permite al usuario borrar eventos por completo de forma rápida y cómoda. Como los eventos son rápidos de añadir y no contienen mucha información se tomó la decisión de no dificultar la eliminación de cada evento preguntando por confirmación, agilizando así el proceso.

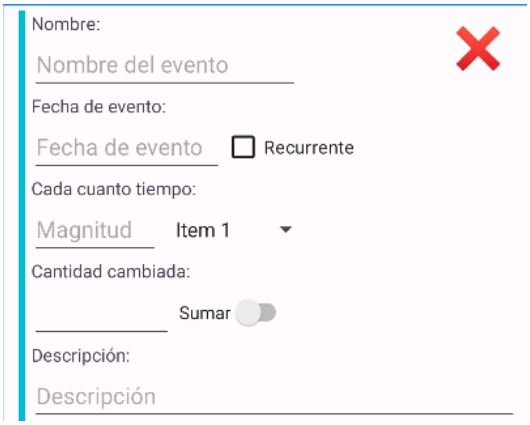


*Ilustración 15 - Diseño final de la visualización de relleno de datos de un evento para un objeto de inventario.*

#### 4.2.8. Diseño de la vista de documentos

Para que la aplicación desempeñara su objetivo de servir como centro de información y gestión para todo lo relacionado con la maquinaria y el inventario de una empresa hay una gran cantidad de datos que la aplicación debería dar al usuario la capacidad de rellenar. Si se intentara tener en cuenta cada posible detalle que le pudiera interesar tener en la aplicación al usuario habría decenas de vistas distintas con una cantidad enorme de parámetros que rellenar, lo cual solo dificultaría el uso de la aplicación.

Con el objetivo de poder tener todos estos datos en la aplicación de forma cómoda para el usuario se ha decidido implementar una vista donde el usuario pueda subir documentos en formato ".pdf" relacionados con máquinas concretas u objetos de inventario. De esta forma los manuales de instrucciones, resultados de inspecciones, listados de piezas, datos de la adquisición y otros documentos de suma importancia se hallarán en un entorno centralizado y bien organizado.

Esta vista dispone de una flecha apuntando hacia la izquierda en la parte superior izquierda de la pantalla para facilitar salir de ella. También dispone de un RecyclerView en el cual por cada documento que se suba a la aplicación aparecerá una entrada con la fecha de en la que se subió y el documento en sí representado por su nombre y el icono de ".pdf" de Windows. Por último, en la sección inferior derecha, al igual que la mayoría de los botones de la aplicación, hay un botón que abre los archivos del móvil y permite seleccionar el ".pdf" a subir. Todos los elementos de esta vista han sido colocados usando una disposición de tipo LinearLayout (vertical) para ordenarlos fácilmente.

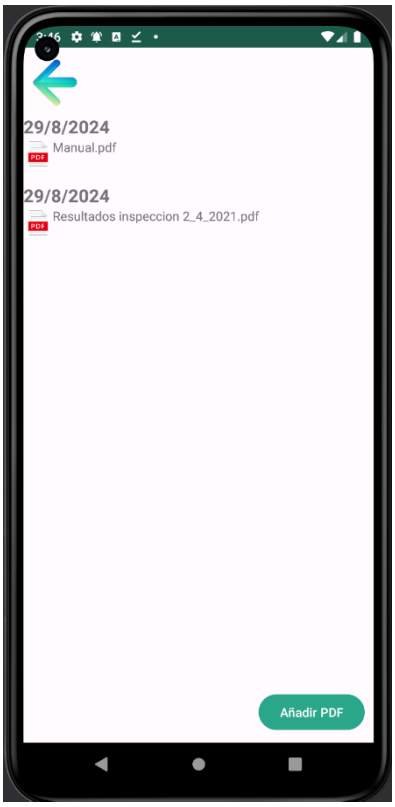


*Ilustración 16 - Diseño final de la vista de documentos.*

## 4.3. Resultados del diseño

A continuación, se incluirán ilustraciones del aspecto final que ha tomado la aplicación uniendo todos los diseños individuales mencionados previamente.

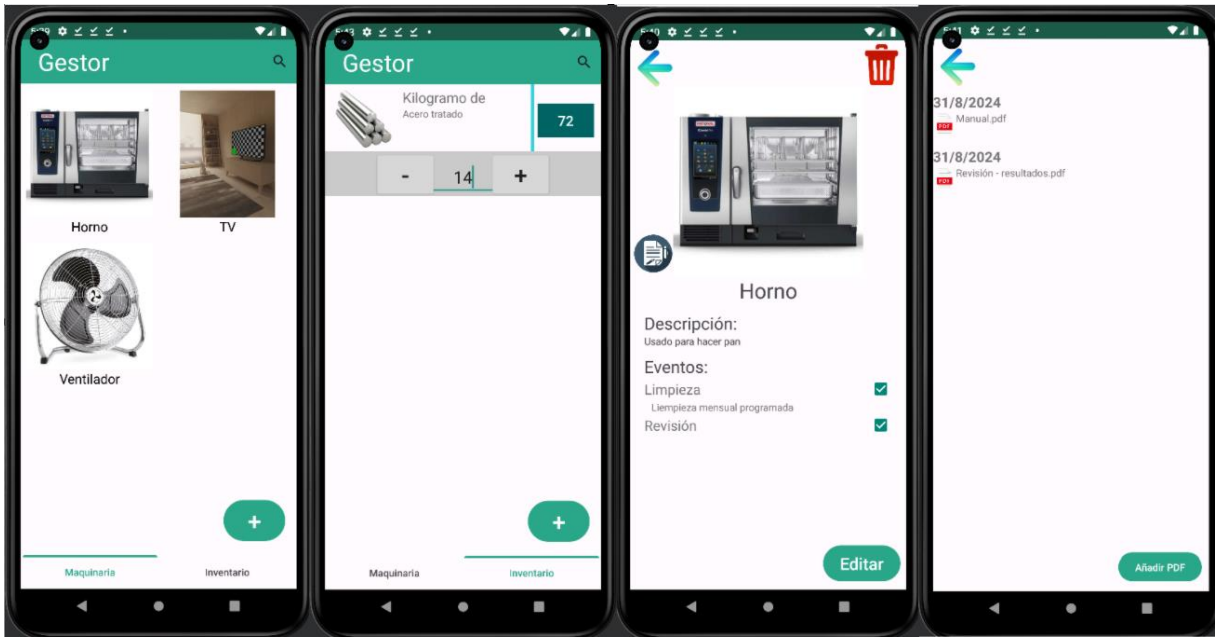


*Ilustración 17 - Visualización de la aplicación al completo (1).*

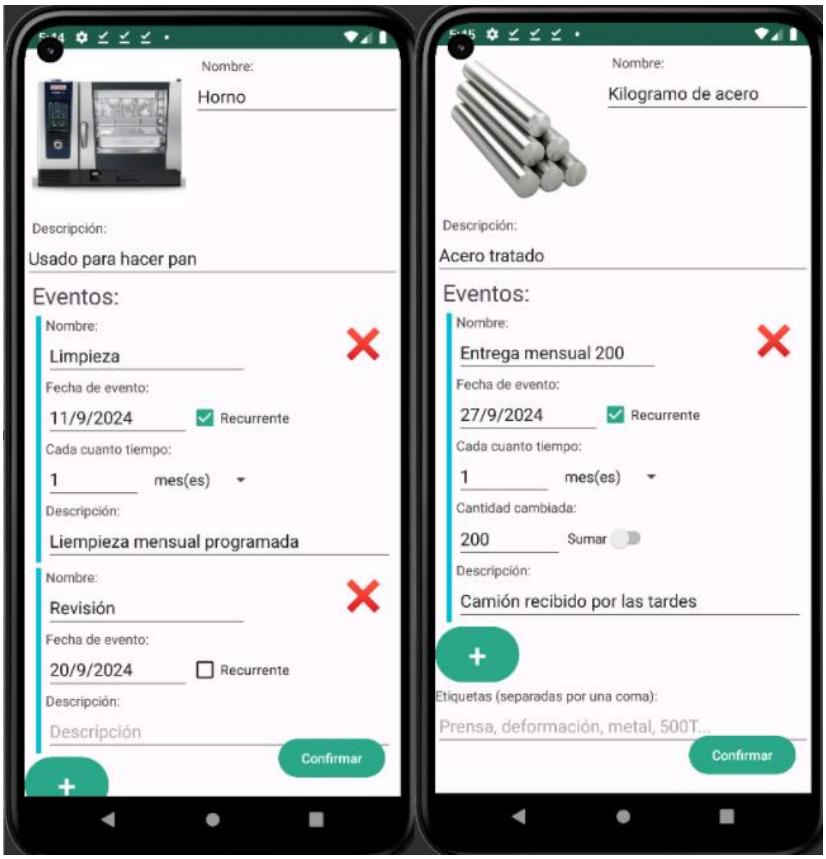


*Ilustración 18 - Visualización de la aplicación al completo (2).*

# Capítulo 5: Desarrollo técnico

## 5.1. Introducción

En este capítulo se presentarán los distintos elementos técnicos y herramientas utilizadas para el desarrollo de la aplicación. Se destacará como han sido necesarios para cumplimentar con la visión de la aplicación móvil tanto desde un aspecto de diseño y ergonomía como desde un aspecto técnico.

El objetivo es proporcionar una comprensión de porque se han usado estas herramientas por encima de otras alternativas y como han contribuido al funcionamiento de la aplicación.

Previo a este contenido el capítulo está dotado de un apartado cuyo objetivo se centra en explicar el funcionamiento de conceptos básicos de *android studio* y de la programación para facilitar el resto de la lectura a aquellos no familiarizados con el tema.

## 5.2. Explicaciones contextuales

### 5.2.1. Actividades

Al hacer una aplicación móvil en A*ndroid Studio* y en otros entornos dedicados al desarrollo de programas y aplicaciones es común la existencia de las actividades o algún equivalente. Podemos considerar a las actividades como las distintas pantallas dentro de una misma aplicación. Las actividades se vinculan con archivos con la extensión ".xml" los cuales se pueden programar a mano o de forma visual arrastrando elementos como botones y texto a una previsualización de cómo se vería la aplicación para el usuario cuando este se encontrará en esa actividad.

La actividad en sí actúa como la parte trasera de la aplicación que se encargar de todas las funciones cuando está activada y el archivo ".xml" (en adelante layout) actúa como la parte con la que el usuario puede interactuar directamente.

### 5.2.2. Listeners

En A*ndroid Studio* se tiene la posibilidad de ponerle métodos de escucha a todos los elementos con los que puede interactuar el usuario llamados “listeners” que permiten ejecutar código en respuesta a una acción del usuario. Estos métodos son útiles para todo tipo de situaciones ya que hay una gran variedad de tipos de listeners. El ejemplo más común sería “onClickListener()” el cual suele usarse con botones y permite realizar una acción cuando el usuario lo pulsa. Algunas otras situaciones en las que pueden ser útiles estos métodos son: saber cuándo el usuario empieza a escribir, está escribiendo o ha terminado de escribir en algún cuadro de texto, saber cuándo el usuario ha marcado o desmarcado una casilla y saber cuándo el usuario mantiene pulsado un elemento por unos instantes.

### 5.2.3. Métodos

Lo métodos son funciones que puede ejecutar un objeto. Pueden usarse para alterar los atributos del objeto, devolver un valor o usar el estado actual de sus atributos para desempeñar otras acciones entre muchas otras cosas.

### 5.2.4. Clases

En programación orientada a objetos, como es el caso del lenguaje de programación Java, las clases son herramientas que podemos usar para crear objetos y dotarlos de comportamiento. Mientras que una clase contiene atributos y métodos únicos de esta, un objeto es una instancia de una clase. Se puede considerar a una clase como un plano y a un objeto como la obra creada a partir del plano. Al igual que se pueden tener varios motores de coche construidos a partir del mismo plano se pueden tener varios objetos de una misma clase, cada uno con el valor que se le quiera asignar a sus atributos, pero con los mismos métodos.

### 5.3.5. Fragmento

Un fragmento es diseño que ocupa una sección de la pantalla, pero no forma parte del resto de la vista, sino que se diseña en un archivo “.xml” distinto. Algunos tipos de vistas especiales requieren de su uso, ya que estas vistas simplemente ocupan un espacio de la vista principal y el diseño de que será mostrado en estas vistas se hace por separado.

#### 5.3.6. Adaptador

Un adaptador se encarga de relacionar datos con elementos de una lista. Por ejemplo, relacionar una lista de cinco números con cinco cuadrados de texto en la pantalla. Es necesario para pasarle información a cierto tipo de vistas especiales que tienen funcionalidades únicas.

### 5.3. ViewPager2

Este elemento es la fundación de toda la aplicación, ya que es el primer elemento interactivo con el que se encuentra el usuario y a través de él se puede llegar a cualquier actividad de la aplicación. Su función es permitir el desplazamiento fluido entre distintas vistas o páginas. Haciendo uso del ViewPager2 se pueden mostrar las listas de máquinas e inventario en un mismo entorno permitiendo cambiar el foco fácilmente entre una y otra.

Por cada página distintas que se vaya a mostrar se necesita crear un fragmento distinto que mostrar. En el caso de esta aplicación los fragmentos serías solo dos. Cada fragmento solo contiene un único RecyclerView, del cual se hablará posteriormente, y en su código asociado se inicializará.

Se decidió usar ViewPager2 en vez de otras alternativas plausibles debido al gran número de comodidades que proporciona. En caso de que a lo largo del desarrollo se hubiera decido tener tres páginas a mostrar en vez de solo dos hubiera sido muy fácil implementar este cambio. Si se hubiera deseado que esa tercera página tuviera un diseño distinto a los dos anteriores no habría habido ningún impedimento para que así fuera. Además, ViewPager2 maneja el tiempo de vida de los fragmentos de forma automática para mejorar el rendimiento de la aplicación y es una de las adiciones más recientes a las librearías de Android, por lo que se puede asegurar que será compatible con futuras versiones de Android próximas.

ViewPager2 también permite implementar fácilmente un sistema de pestañas para navegar entre las páginas mostradas, de lo cual se hablará a continuación.

### 5.4. TabLayout

El TabLayout es un elemento dentro del cual se pueden introducir distintas pestañas. Al contrario que un ViewPager2, dentro del TabLayout no hay ninguna vista ni ningún diseño complejo, es solo la sección donde se muestran las diferentes pestañas y en cual se encuentra el usuario actualmente. Dentro de la actividad principal de la aplicación donde se muestran las pestañas se relaciona la posición de la página del ViewPager2 con la posición de la pestaña seleccionada. De esta forma al seleccionar una pestaña el ViewPager2 se moverá a la página seleccionada y al mover la página del ViewPager2 la pestaña seleccionada cambiará.

### 5.5. RecyclerView

Este es el elemento más importante en cuanto a la comodidad de uso de la aplicación. Si se deseara escribir en la pantalla un texto que el usuario hubiera declarado previamente el procedimiento sería simplemente incluir una caja de texto en el diseño de la actividad y hacer que al visualizar esa actividad el programa escriba el dato guardado en memoria. Este tipo de procedimiento presenta un problema. Si en vez de tener un solo cuadrado de texto se deseara que hubiese tantos como datos haya declarado el usuario no se puede simplemente incluir en el diseño de la actividad veinte recuadros de texto y esperar que el usuario no declarara más de veinte datos.

¿Cómo se puede solventar este problema? Ahí es donde entra la RecyclerView. Haciendo uso de este tipo de vista especial se puede hacer que aparezca una distribución de componentes de diseño repetida por cada elemento que forme parte de una lista. Esto quiere decir que se puede crear un diseño para cada elemento de la lista de máquinas o de la lista del inventario y hacer que ese diseño solo se muestre por pantalla tantas veces como elementos haya en esas listas.

Para crear una RecyclerView funcional hace falta el diseño que va a ser repetido en su interior, la lista a partir de la cual se van a mostrar los datos (la lista de máquinas o la lista del inventario) y el adaptador que se encargará de relacionar todos los componentes del diseño repetido con los datos de la lista. El adaptador tiene cuenta la posición de cada elemento de la lista y la posición de cada diseño mostrado en la RecyclerView, permitiendo hacer que cada entrada de la lista siendo mostrada este personalizada con los datos únicos de dicha entrada. De esta forma si se dispusiera de tres máquinas llamadas máquina uno, máquina dos y máquina 3 los cuadrados de textos destinados a mostrar el nombre de las máquinas tendrían cada uno el nombre de la máquina a la que deberían estar asociados, máquina uno, máquina dos y máquina tres.

Hay varias formas distintas de realizar un efecto igual o similar al descrito anteriormente con otras herramientas distintas a la RecyclerView, sin embargo, se

decidió usar esta porque presentaba unas ventajas por encima del resto. Otros dos elementos comúnmente utilizados para este propósito son ListView y GridView, que son componentes más antiguos y reciben menos actualizaciones a medida que pasa el tiempo.

ListView permite repetir los diseños en una única fila mientras que GridView permite repetirles usando un patrón de cuadrícula. RecyclerView puede hacer la función de ambos, ya que permite customizar la distribución en la que se muestran los elementos de la lista, además de que permite albergar una cantidad ilimitada de elementos en su interior (dentro de las limitaciones de memoria del tamaño de la lista naturales) mientras que ListView y GridView pueden dar problemas al tener que cargar listas muy grandes. Además, usando una RecyclerView se puede hacer uso de métodos ya integrados que permiten una mayor personalización de cómo se muestran los datos.

La RecyclerView no solo se utiliza en este proyecto para mostrar la lista del inventario y de las máquinas. También se usa para mostrar los eventos activos e inactivos que afectan a cada máquina u objeto de inventario en la actividad de detalles, para mostrar los eventos en su versión editable en la actividad para añadir o editar una máquina u objeto y en la actividad de documentos donde se muestran todos los archivos “pdf” subidos por el usuario. Debido a su repetido uso por toda la aplicación este ha sido una herramienta clave para el correcto desarrollo de la aplicación de forma cómoda para el usuario

### 5.6. Listas

Para organizar todos los elementos de la aplicación se ha hecho uso de listas. Las listas, como su nombre indica, son capaces de contener cualquier cantidad de ciertos valores. Las listas pueden ser de un tipo de objeto concreto, por ejemplo, una lista de valores enteros o de cadenas de letras. En la aplicación se han creado cuatro clases distintas cuyo único propósito es ser usadas para crear objetos que guardar en listas orientadas a guardar ese tipo de objetos. Estas cuatro clases representan las máquinas, los objetos de inventario, los eventos y los documentos. Cada clase permite crear uno objeto correspondiente que contenga los datos específicos asignados a ese objeto. Cada uno de esos objetos se guarda en una lista de tal manera que todas las máquinas están en una misma lista y todos los objetos de inventario están en otra. De forma opuesta cada elemento de cualquiera de esas dos listas dispone de su propia lista de objetos de evento y su propia lista de objetos de documentos. De esta forma las máquinas están separadas del inventario y cada elemento de ambas de estas listas contienen toda la información pertinente solo a ellos mismos.

### 5.7. Notificaciones y AlarmManager

El sistema de notificaciones es una de las características más importantes para permitir una gestión eficiente de la aplicación. Hacer que una aplicación envíe una notificación no es una tarea dificil ya que hay una librería de android dedicada para esto que permite realizarlo en pocas líneas de código. Primero hay que crear un canal de notificaciones en el que se declara la importancia que tendrán todas las notificaciones enviadas a través de él. Esto afectará al comportamiento que las notificaciones presentarán en el centro de notificaciones del móvil. Para esta aplicación la importancia será la máxima posible ya que se asume que el usuario no quiere arriesgarse a ignorar un evento que él mismo haya programado. En consecuencia, las notificaciones no desaparezcan ni siquiera si el usuario abre la aplicación, solo se irán si el usuario las borra expresamente. Después solo hay que construir la notificación con los datos que se desea enseñar y publicarla.

El problema es hacer que la notificación le pueda llegar al usuario cuando no esté usando la aplicación, ya que en eso radica la verdadera utilidad de las notificaciones. Por suerte Android dispone de un sistema de alarmas al que se puede acceder desde la aplicación mediante un AlarmManager. Concretamente con el AlarmManager podemos publicar alarmas ya sea para que suenen a una hora exacta una única vez o que se repitan cada cierto tiempo.

Haciendo pruebas con varios dispositivos se descubrió que esta función daba errores en versiones más actuales del sistema operativo de Android debido a que anteriormente no era necesario solicitar permisos al usuario para publicar alarmas, pero en las versiones más actuales sí que es necesario. En caso de no hacerlo la aplicación fallará. Para evitar esto ha sido necesario comprobar la versión del sistema operativo actual y si es moderno solicitar permiso.

La mayoría de los permisos se solicitan con un método simple que le muestra al usuario una pestaña para permitir o denegar el permiso. Por desgracia, la habilidad de publicar alarmas no es un permiso que se pueda solicitar de esa forma. El sistema operativo decide automáticamente si una aplicación debe de poder publicar alarmas o no por defecto. Para poder publicar alarmas el usuario tiene que activar esa opción en los ajustes del dispositivo para la aplicación. Como método para evitar que el usuario tenga que saber esto por sí solo la aplicación abrirá automáticamente los ajustes del dispositivo en la pestaña para activar el permiso de alarmas y mostrará un mensaje temporal por pantalla indicándole al usuario que active las alarmas para poder recibir

notificaciones. Claro está, justo después se solicitará el permiso de enviar notificaciones de forma normal.

Estas alarmas no son como las alarmas que uno podría usar para despertarse. Son alarmas que se disparan internamente de forma silenciosa y se pueden usar para ejecutar código de la aplicación creando una clase que extienda de BroadcastReciever. Al publicar la alarma se le puede pasar cierta información que se puede recoger de nuevo cuando se dispara, como el texto que se quiere que muestre la notificación, por ejemplo. Inicialmente la idea para el sistema de notificaciones consistía en que cada evento programado enviara una alarma programada para soltar en la fecha de cumplimiento del evento. En caso de ser un evento recurrente en vez de programar una alarma exacta se programaría una alarma repetible.

Esta solución causaba dos fallos que impedían que el sistema de notificaciones fuera fiable. El primero es que al tener varias alarmas programadas para un mismo día solo saltaban algunas de ellas y en consecuencia solo se visualizaban uno o dos eventos. El segundo problema tenía que ver con el funcionamiento de las alarmas repetibles. Estas se repetían pasado un tiempo previamente declarado, por lo que al hacer una alarma diaria o semanal no había problema. No obstante, al programar una alarma con una repetición mensual o anual la alarma dejaba de ser fiable ya que la cantidad de tiempo que dura un mes o que dura un año puede cambiar.

En consecuencia, de estos dos fallos que no parecían tener ninguna solución se decidió afrontar las notificaciones con otra perspectiva. En vez de provocar que cada evento disparara una alarma ahora la aplicación dispara una alarma todos los días. En consecuencia, la aplicación busca entre todos los eventos cuales tienen que sonar ese día y monta y pública las notificaciones correspondientes. Para hacer esto ya no se usa la característica de pasarle información a la alarma que se pueda recoger cuando suena, sino que se accede directamente a la memoria de la aplicación.

Para no saturar el centro de notificaciones del usuario en vez de hacer una notificación por cada evento se ha hecho una notificación por máquina u objeto de inventario la cual es desplegable y dentro de la cual se mencionan los eventos del día. Este tipo de notificaciones se llama BigText y la cantidad de texto que permite mostrar en su interior es dependiente del móvil usado, pero suele rondar entorno a las catorce líneas de texto en los dispositivos usados para el desarrollo de este proyecto. Se usa mínimamente una línea de texto por evento, por lo que se ha considerado que mostrar un máximo de catorce eventos por máquina u objeto de inventario al día sería más que suficiente para no perder información.

En cuanto a los eventos recurrentes, cada día cuando suena la alarma si uno de los eventos es recurrente se le añade a la fecha del evento la cantidad de días, meses o

años restantes para que vuelva a sonar y se guarda en memoria con la nueva fecha. De esta manera cuando llegase el día de la repetición del evento la alarma lo trataría simplemente como uno evento cualquiera que estaba programado para ese día. El problema con este método es bastante obvio. En caso de tener una alarma que se repita cada mes cuando esta suena en diciembre, mes doce, le intentará provocar que la nueva fecha sea el mes trece, que no existe. Afortunadamente el sistema de calendarios de Android está preparado para este tipo de cosas, por lo que se hizo uso de él. Al sumar un mes por encima de diciembre simplemente hace la conversión lógica y pone la nueva fecha en enero del año siguiente. En caso de tener un evento en el treinta y uno de enero con una repetición mensual el calendario lo ajustaría automáticamente al veintiocho de febrero o veintinueve en años bisiestos.

Esta solución parece ideal, pero cuando una fecha se ajusta como en el último ejemplo expuesto al sumar otro mes no se vuelve a posicionar en el día treinta y uno. Esto tiene fácil arreglo simplemente guardando la fecha del evento en memoria con el mismo día siempre y cambiando solo el mes. Es decir, el calendario solo se usa a la hora de comprobar si el evento debería de dispararse hoy o no, pero en la memoria de la aplicación la fecha del evento podría estar guardada como día treinta y uno del mes veintisiete del año dos mil veinticuatro.

Este método alternativo para manejar las alarmas y notificaciones parece adecuado, aunque poco elegante, ya que para buscar los eventos hace falta hacer uso de un triple bucle “for,” lo cual no es ideal. No obstante, este proceso se lleva a cabo internamente en segundo plano y no ha dado ningún problema durante las pruebas realizadas. Se ha considerado que es irrelevante si las notificaciones tardan unos segundos más en llegar debido a la ineficiencia de este sistema.

Aprovechando que se puede acceder al código y a la memoria de la aplicación cuando se disparan las alarmas es posible hacer que cuando se notifique de un evento correspondiente a algún objeto del inventario se sume o se reste la cantidad que estaba programada en el evento y se guarde la nueva cantidad en memoria, teniendo en cuenta que no debe de bajar de cero.

El sistema de alarmas tiene un gran problema que podría inutilizar toda la característica de notificaciones. Ese es que cuando el teléfono se reinicia las alarmas programadas desaparecen. La aplicación programa la alarma del día siguiente en consecuencia que se dispare la alarma que se programó el día anterior, por lo que un reinicio del dispositivo rompería la cadena de alarmas y evitaría que el usuario recibiera ninguna notificación. Por suerte la aplicación puede solicitar un permiso, el cual se le concede por defecto sin intervención del usuario, que permite recibir una alarma de notificación de reinicio cuando el dispositivo se desbloquea después reiniciarse. Haciendo uso de esto se escribió en el código de las notificaciones un comportamiento

que provoca que si la alarma es una notificación de reinicio simplemente se programe una alarma para dispararse en ese mismo instante, empezando así el bucle de alarmas de nuevo.

### 5.8. Singleton

El Singleton es la espina dorsal de la aplicación. Es una clase que se usa como punto de acceso central a los datos de la aplicación. Cada vez que en alguna clase se tiene que acceder a la lista de máquinas o de inventario se llama al Singleton que la tiene guardada. Solo se crear una única instancia del Singleton, por lo que siempre que se saca una lista o cualquier otro dato del Singleton la dirección de memoria a la que acceden es la misma. Esto quiere decir que en caso de igualar una variable local a una sacada del Singleton, si se alterara la variable local de alguna forma también se vería afectado el dato del Singleton. De esta forma la aplicación no contiene varias listas iguales en cada actividad. Además, en caso de necesitar un dato de otra actividad es muy sencillo crear una variable para asignar ese dato en el Singleton y usarlo de puente entre actividades.

El problema principal que afecta al Singleton es la habilidad de mantener los datos guardados después de cerrar la aplicación. El código de la aplicación mantiene su funcionalidad original cada vez que se reinicia la aplicación, pero los datos asignados a cada variable se pierden. La solución a esta es usar un sistema de base de datos, sin embargo, ese sistema puede llegar a ser complejo y crea una dependencia a la red de internet. Para poder asegurar que el usuario pudiera siempre acceder a la información desde cualquier punto de acceso se decidió usar un fichero de preferencias para guardar los datos de la aplicación en la propia memoria del teléfono. El fichero de preferencias tiene algunas limitaciones importantes, como por ejemplo que el tipo de datos que se puede guardar con él incluye solo los tipos más básicos: números enteros, números tipo “float,” números tipo “long,” valores binarios, cadenas de texto simples y una colección de cadenas de texto. Otro de los problemas que presenta el uso del archivo de preferencias es la cantidad de memoria que pueden ocupar los datos, ya que, aunque no hay un límite estricto se recomienda no guardar más de un "megabyte” de datos.

El primer problema se ha solucionado convirtiendo todos los datos de la aplicación en dos cadenas de “json” que se pueden escribir como cadenas de texto normal, una para la información de las máquinas y otra para la información del inventario. De esta forma no habrá ningún problema al guardar la información en el fichero de preferencias ya que como se ha mencionado uno de los tipos de datos permitidos para su uso son las cadenas de texto. A la hora de volver a cargar los datos de

memoria sencillamente se tiene que realizar una desconversión de los datos para volver a convertirlos en información con la que poder trabajar en el resto del proyecto.

El segundo problema no es tan limitante como podría sonar. La información guardada está escrita en el formato "UTF-8," lo que significa que los caracteres simples como letras, signos de puntuación y números ocupan un solo "byte" de memoria. Eso significa que podemos guardar más de un millón de caracteres en memoria antes de que existiera la posibilidad de afectar a la velocidad de carga de datos. Incluso si el usuario excediera esa cantidad no tienen poque provocar cambios notables en la aplicación, así que se considera un método de almacenamiento seguro.

### 5.9. Selector de fecha

Para permitir que el usuario seleccione la fecha en la que cada evento se disparará se pensó en hacer que dicha fecha se pudiera seleccionar directamente de un calendario. Esto se debe a que si el usuario introduce la fecha manualmente se le tendría que informar de que formato usar para escribirla y en caso de que el formato sea incorrecto la aplicación podría fallar.

Para seleccionar la fecha se hace uso del sistema de calendarios de Android y el sistema de datePickerDialog, que permite ensañar por pantalla un calendario. El sistema de calendarios de Android se usa para organizar la fecha en un formato estándar y poder introducir la fecha actual por defecto en datePickerDialog. De esta forma al abrirlo por primera vez la fecha mostrada será la actual y no el primer día del año 1970.

Al seleccionar una fecha en el datePickerDialog el único efecto en respuesta es escribir dicha fecha en el recuadro de texto destinado para ello en la edición del evento. Ese mismo recuadro dispone de un listener que automáticamente actualiza la fecha del evento cuando se edita el valor escrito en él.

### 5.10. Selector de imágenes

La característica visual más importante para poder diferenciar las máquinas y los objetos del inventario son las imágenes que el usuario le puede asignar. Para hacer esto android nos permite directamente abrir dentro de la aplicación los datos guardados del móvil y recibir la dirección del archivo seleccionado por el usuario. Esta dirección la recibe la ampliación en formato "Uri" que es esencialmente una cadena de texto

especial que índica la posición de un archivo dentro del almacenamiento del teléfono móvil.

Haciendo uso de esta dirección la aplicación puede dibujar imágenes que no forman parte de sus archivos internos, sino de los archivos del usuario, sin necesidad de solicitar permisos para acceder a las imágenes. El único inconveniente de esto viene dado por el hecho de que si el usuario decide borrar la imagen a la que apunta la dirección en el almacenamiento de su teléfono entonces también desaparecerá de la aplicación.

La solución para evitar eso es conseguir el "BitMap" de la imagen que quiere seleccionar el usuario, que es una estructura digital en forma de matriz en la que cada entrada de la matriz representa un píxel de la imagen. Esto, aunque permitiría guardar los datos en el almacenamiento de la aplicación, provocaría que dicho almacenamiento excediera el límite de información que se puede guardar con un archivo de preferencias. Una sola imagen podría sobrepasar dicho límite, por lo que, aunque ligeramente inconveniente la solución de usar la dirección interna de la imagen es preferible.

Para evitar ningún error al intentar cargar una imagen que el usuario pueda haber borrado simplemente se ha introducido una imagen por defecto señalando que ha habido algún error, la cual es mostrada si no se puede mostrar la imagen deseada originalmente.

### 5.11. Selector de documentos

De forma similar que con las imágenes, el selector de documentos funciona usando la dirección del archivo para mostrarlo. La diferencia es que con los documentos es necesario obtener su nombre para escribirlo en la aplicación. Esto es un problema debido a que a veces en vez de usar en la dirección de guardado el nombre del documento se utiliza un indicador. Por ejemplo, en la carpeta de descargas del dispositivo la dirección pude terminar como "descargas/5" en vez de "descargas/<nombre del documento>."

Para arreglar esto se hace uso de un método llamado "getContentResolver" de la clase "Context" para obtener una serie de datos complejos del documento con los que se pueda trabajar para eventualmente obtener el nombre del documento.

Como al igual que con las imágenes la dirección del documento no apuntaría a ningún sitio si el usuario borra el documento se ha tenido que contemplar este caso. Si la dirección es invalida en lugar de mostrar el nombre del documento se mostrará el texto "Documento no disponible" y en lugar de abrir un documento para su visualización si se

intenta pulsar el documento simplemente se mostrará un mensaje por pantalla avisando al usuario del error.

### 5.12. NestedScrollView

La NestedScrollView es un elemento de diseño que se usa en la aplicación para que las vistas creadas que incluyan un RecyclerView entre otras cosas no tengan un comportamiento confuso. Como se ha mencionado anteriormente la RecyclerView permite el desplazamiento de los elementos de la misma forma que en una página web normal el usuario se puede desplazar verticalmente. Aunque esto es útil puede dar algunos problemas. En caso de que la RecyclerView no ocupe la vista al completo y tenga que compartirla con otros elementos el resultado sería que solo la parte de la pantalla que ocupa la RecyclerView es desplazable.

Para una RecyclerView que muestre elementos que ocupen un gran tamaño esto puede ser extremadamente molesto ya que reduce la visibilidad, como es el caso de la RecyclerView presente en la actividad dedicada a añadir una nueva máquina u objeto de inventario. La NestedScrollView no solo permite que toda la actividad sea desplazable como lo haría una ScrollView nomal, sino que además unifica el contenido de todos los elementos en un único desplazamiento general. Esto significa que en vez de tener una RecyclerView que ocupe una sección de la pantalla y sea desplazable dentro de esa sección, la RecyclerView será estirada hasta mostrar todos los elementos que contenga y toda la actividad se convertirá en un desplazable grande.

NestedScrollView fue elegido porque es extremadamente simple de usar. No es necesario programar nada, simplemente con usarlo el NestedScrollView se encarga automáticamente de gestionar el desplazamiento.

### 5.13. Sistema de búsqueda

El sistema de búsqueda es necesario en caso de que el usuario tenga un gran número de entradas en cada lista. Para crearlo se ha hecho de tal forma que la barra de búsqueda esté introducida dentro de la barra de herramientas en la sección superior de la actividad principal.

El objetivo es que al buscar un texto se muestren todas las máquinas y objetos de inventario en sus secciones correspondientes que tengan un nombre o una etiqueta igual

al texto siendo buscado. Para esto cada vez que se altera la búsqueda en cuestión se crean dos nuevas listas vacías, una para cada lista original.

Se analizan todos los elementos de las listas originales a través del Singleton y usando el método “contains” de las cadenas de texto para comprobar si el texto buscado aparece en el nombre o en las etiquetas de cada elemento de cada lista. En la búsqueda se ha hecho irrelevante la capitalización de las letras para evitar errores humanos puntuales en la búsqueda. En el caso de que así sea se copia el elemento de la lista en las listas vacías.

Después de analizar las listas al completo se realiza una llamada a un método creado en el adaptador de la RecyclerView que muestra las listas. El propósito de este método es cambiar la lista que contiene el adaptador por la lista filtrada con los parámetros de la búsqueda y notificarle al propio adaptador de que ha habido un cambio en los datos para que actualice lo que está siendo mostrado.

Si el código se dejara así en caso de realizar una búsqueda, luego borrarla y luego intentar eliminar uno de los elementos mostrados en pantalla no ocurriría aparentemente nada debido a que la lista siendo mostrada sería una copia de la original y al eliminar un elemento se borra de la lista original en el Singleton. Para evitar que el usuario empiece a borrar elementos sin saberlo intentando borrar uno solo se ha introducido una condición en el código de la búsqueda. El comportamiento descrito anteriormente solo se realiza en caso de que la búsqueda no esté vacía, en cuyo caso la lista del adaptador se vuelve a cambiar por la original.

Debido a como estaba ya programado de antemano si se intenta eliminar un elemento en medio de una búsqueda no solo no se eliminará visualmente el elemento, sino que además se podría estar eliminando un elemento distinto internamente. Para navegar alrededor de este problema se ha hecho que la opción de eliminar objetos desde la actividad principal no esté activa durante la búsqueda, pero el usuario podrá seguir accediendo a los detalles del elemento y eliminándolo sin problemas.

### 5.14. ContextMenu

ContextMenu es un menú que aparece en el centro de la pantalla en respuesta a una acción del usuario. Muchas aplicaciones lo usan en respuesta a la pulsación prolongada de algo, al igual que en esta aplicación. Al construir el menú se puede seleccionar el número de opciones que aparecen desplegadas y el texto que tendrán escrito. Este menú se usa en la aplicación para eliminar las máquinas u objetos de inventario desde la vista principal, al igual que para eliminar los documentos en la

actividad de documentos. Para hacerlo solo hay que mantener pulsado uno de estos elementos, lo cual se ha podido detectar usando un listener.

Se puede crear una función llamada OnContextItemSelected que recibirá automáticamente la opción seleccionada en un menú y será ejecutada. Dentro de esta función es donde se lleva a cabo el código de eliminación de lo que sea que el usuario este intentado borrar. Al borrar un elemento de alguna de las listas el cambio no se refleja automáticamente en vista que está siendo presenciada. Para poder presenciar el cambio el usuario tendría que recargar la actividad.

Esto es un gran problema ya que si los cambios no se reflejan el usuario no solo podría pensar que no se ha borrado realmente nada, sino que si intenta volver a borrar el elemento podría estar borrando un elemento completamente distinto sin saberlo o incluso provocar que la aplicación se cierre debido a un fallo. Para arreglar esto después de realizar el proceso de eliminación la función OnContextItemSelected tiene que notificar al adaptador asociado a la RecyclerView correspondiente con la que se está interactuando de que ha habido un cambio en los datos. De esta manera todos los cambios se muestran instantáneamente.

# Capítulo 6: Análisis del cumplimiento de los requisitos

En este capítulo se comentará el cumplimiento de los requisitos autoimpuestos en el capítulo dos y como se ha logrado cada uno en el mismo orden en el que se mencionaron anteriormente con excepción de los requisitos generales, que serán evaluados en último lugar.

## 6.1. Requisitos de la vista principal

Al abrir la aplicación el entorno que se presenta está poco abarrotado de información ya que contiene una mínima cantidad de datos para diferenciar cada máquina y cada objeto de inventario. Además, las máquinas y el inventario se presentan de forma claramente separada debido a que se encuentran en pestañas distintas.

Para poder diferenciar los todos los elementos los unos de los otros con facilidad y asociarlos a su contraparte real en un vistazo rápido es para lo que sirven las imágenes que puede seleccionar el usuario.

Para poder buscar una máquina, un objeto o una colección de máquinas u objetos concretos se introdujo la barra de búsqueda. Con ella no solo se le permite al usuario buscar cualquier elemento por el nombre, sino también por las etiquetas asignadas permitiendo así un mayor nivel de organización.

En la actividad principal se le permite al usuario crear una nueva máquina u objeto del inventario y en la actividad que permite rellenar los datos de un objeto nuevo o existente se le permite añadir eventos nuevos. Al llevar a cabo cualquiera de estas dos acciones el listado que muestra todos los datos es actualizado al instante.

Para que todos los elementos se puedan eliminar desde la vista principal de forma lo suficientemente escondida para que no se eliminen por accidente se hace uso del ContextMenu, que presenta la opción de eliminar un elemento después de mantenerlo pulsado. Mantener pulsado algo para poder desplegar un menú de opciones no es una práctica poco común en las aplicaciones móviles, por lo que no es lo suficientemente enrevesado como para que el usuario no lo encuentre nunca, pero si está lo suficientemente escondido para no pulsarlo por accidente. Como esta no es la única forma de eliminar un elemento no se ha considerado importante señalar explícitamente

esta mecánica al usuario. Al eliminar uno de los elementos se ha hecho que el cambio se muestre instantáneamente para evitar errores.

Con esto se pueden considerar cumplidos todos los requisitos expuestos en el capítulo dos relacionados a la vista principal.

### 6.2. Requisitos comunes para máquinas y objetos de inventario

La vista de detalles es accesible desde la vista principal haciendo una única pulsación en cualquier elemento y dentro de esta vista se le muestra al usuario un icono de una papelera indicándole que puede pulsarlo para borrar el elemento.

El sistema de notificaciones se encarga de notificar todos los eventos en las fechas en las que deberían de sonar si están activos en ese momento. También está contemplada la repetición automática de los eventos cada cierto tiempo. Para asegurar la comodidad de uso del usuario para el sistema de notificaciones la fecha de los eventos recurrentes se seguirá cambiando a la próxima repetición incluso si el evento está inactivo para que el usuario pueda simplemente reactivar el evento cuando sea y que este funcione como se esperaría.

Con la implementación del sistema de documentos se le permite al usuario tener toda la información relevante de cada elemento organizada dentro de la aplicación sin la necesidad de copiar y rellenar lo que podrían de ser cientos de datos que tendrían que estar contemplados en el diseño de las vistas. De esta forma el usuario puede disponer de información útil como el listado de piezas de una máquina dentro de la información de la propia máquina.

Con estas características la aplicación ya cumple con los requisitos mínimos establecidos inicialmente.

### 6.3. Requisitos para objetos de inventario

El único requisito específico para los objetos de inventario era la automatización de los cambios en la cantidad del objeto de la que dispone el usuario de forma puntual o recurrente. Para hacer esto se ha usado el sistema de notificaciones y eventos ya que los eventos ya notificaban de cuando tienen que ocurrir estas alteraciones de la cantidad y el sistema de notificaciones ya permitía ejecutar código de la aplicación sin tenerla abierta. Simplemente se ha hecho que cuando se dispare una notificación si está tenía

que alterar la cantidad de un objeto se carguen los datos del fichero de preferencias, se edite la cantidad y se vuelva a guardar en memoria.

## 6.4. Requisitos generales

A la hora de implementar todas las características que componen la aplicación se ha intentado que la aplicación no sea incomoda de usar o demasiado complicada. Sin embargo, no hay una medida exacta para comprobar como de complicado puede ser entender cómo funciona la aplicación. Es posible que algunos usuarios tengan más problemas que otros, no obstante, en la medida de lo posible se ha intentado facilitar el uso de todas las características relevantes e imprescindibles para minimizar la cantidad de gente que pueda tener dificultades usando la aplicación.

Todos los errores encontrados en la aplicación han sido corregidos y se ha evitado que ningún error sea capaz de causar el cierre de la aplicación. No obstante, es muy difícil que cuando se pública una aplicación, página web, videojuego o cualquier otro tipo de tecnología digital no exista algún error escondido. Se puede garantizar que la aplicación desarrollada en este proyecto no contiene ningún error que el usuario se pueda encontrar mientras se utilice del modo esperado y también se puede garantizar que en el caso de existir algún error por el un mal uso de la aplicación no será sencillo de encontrar.

## 6.5. Conclusión

Todos los requisitos establecidos al comienzo de este trabajo se han cumplido durante la realización de este, a excepción posiblemente del requisito que requería que no hubiese errores ya que no es posible asegurar eso con absoluta certeza. La cuestión real es si cumplir con estos requisitos ha ayudado realmente a que la aplicación desempeñe su función satisfactoriamente.

Aunque algunos requisitos no han causado necesariamente una mejora en la funcionalidad de la aplicación, como la característica de poder eliminar los elementos de cada lista desde la vista principal, hay otros que, si lo han sido, como poder automatizar la alteración de la cantidad de cada objeto. Lo que es seguro es que cumplir con todos los requisitos no ha tenido ningún efecto negativo en la aplicación ya que todos eran mejoras.

# Capítulo 7: Presupuesto

Como se ha mencionado en otros capítulos, la aplicación será gratis, ya que parte del objetivo a cumplir implica hacer la aplicación lo más accesoria posible. Aun así, en este apartado se realizará un presupuesto basándose en cuanto podría haber costado el desarrollo de una aplicación de este calibre. Este presupuesto se expone con fines orientativos.

## 7.1. Mano de obra

| Código | Descripción de la mano de obra | Precio (€/h) |
|---|---|---|
| **MO1** | Alumno cursando un Grado de Ingeniería en Tecnologías Industriales | 4,60 |

*Tabla 1 – Precio de la mano de obra.*

## 7.2. Materiales

| Código | Unidad | Descripción del material | Precio (€) |
|---|---|---|---|
| **MA1** | h | Coste energético de un ordenador portátil de gama media operando a máximo rendimiento | 0,018 |

*Tabla 2 – Precio del material.*

## 7.3. Precios descompuestos

| Código | Descripción del material | Unidad | Cantidad | Precio (€) | Importe (€) |
|---|---|---|---|---|---|
| **UO1** | Investigación previa y planificación | h | | | 6,15 |
| **MO1** | Alumno cursando un Grado de Ingeniería en Tecnologías Industriales | h | 1 | 4,60 | 4,60 |
| **MA1** | Coste energético de un ordenador portátil de gama media operando a máximo rendimiento | h | 1 | 0,018 | 0,018 |
| | Costes directos complementarios | | 0,03 | 4,618 | 0,14 |
| | Costes directos | | | | 4,92 |
| | Costes indirectos | | 0,25 | 4,92 | 1,23 |

*Tabla 3 – Unidad de obra de la investigación previa.*

| Código | Descripción del material | Unidad | Cantidad | Precio (€) | Importe (€) |
|---|---|---|---|---|---|
| **UO2** | Desarrollo de la interfaz de usuario | h | | | 6,15 |
| **MO1** | Alumno cursando un Grado de Ingeniería en | h | 1 | 4,60 | 4,60 |

| | | | | | |
|---|---|---|---|---|---|
| | Tecnologías Industriales | | | | |
| **MA1** | Coste energético de un ordenador portátil de gama media operando a máximo rendimiento | h | 1 | 0,018 | 0,018 |
| | Costes directos complementarios | | 0,03 | 4,618 | 0,14 |
| | Costes directos | | | | 4,92 |
| | Costes indirectos | | 0,25 | 4,92 | 1,23 |

*Tabla 4 – Unidad de obra del desarrollo de la interfaz de usuario.*

| Código | Descripción del material | Unidad | Cantidad | Precio (€) | Importe (€) |
|---|---|---|---|---|---|
| **UO3** | Desarrollo informático de la aplicación | h | | | 6,15 |
| **MO1** | Alumno cursando un Grado de Ingeniería en Tecnologías Industriales | h | 1 | 4,60 | 4,60 |
| **MA1** | Coste energético de un ordenador portátil de gama media operando a máximo rendimiento | h | 1 | 0,018 | 0,018 |

| | | | | | |
|---|---|---|---|---|---|
| | Costes directos complementarios | | 0,03 | 4,618 | 0,14 |
| | Costes directos | | | | 4,92 |
| | Costes indirectos | | 0,25 | 4,92 | 1,23 |

*Tabla 5 – Unidad de obra del desarrollo informático de la aplicación.*

| Código | Descripción del material | Unidad | Cantidad | Precio (€) | Importe (€) |
|---|---|---|---|---|---|
| **UO4** | Desarrollo del trabajo escrito | h | | | 6,15 |
| **MO1** | Alumno cursando un Grado de Ingeniería en Tecnologías Industriales | h | 1 | 4,60 | 4,60 |
| **MA1** | Coste energético de un ordenador portátil de gama media operando a máximo rendimiento | h | 1 | 0,018 | 0,018 |
| | Costes directos complementarios | | 0,03 | 4,618 | 0,14 |
| | Costes directos | | | | 4,92 |
| | Costes indirectos | | 0,25 | 4,92 | 1,23 |

*Tabla 6 – Unidad de obra del desarrollo del trabajo escrito.*

### 7.4. Precios unitarios

| Número de orden | Descripción de la unidad de obra | Unidad | Precio |
|---|---|---|---|
| **UO1** | Investigación previa y planificación | h | 6,15 |
| **UO2** | Desarrollo de la interfaz de usuario | h | 6,15 |
| **UO3** | Desarrollo informático de la aplicación | h | 6,15 |
| **UO4** | Desarrollo del trabajo escrito | h | 6,15 |

*Tabla 7 – Precios unitarios del presupuesto.*

### 7.5. Estado de mediciones

| Número de orden | Descripción de la unidad de obra | Unidad | Medición |
|---|---|---|---|
| **UO1** | Investigación previa y planificación | h | 20 |
| **UO2** | Desarrollo de la interfaz de usuario | h | 100 |
| **UO3** | Desarrollo informático de la aplicación | h | 150 |
| **UO4** | Desarrollo del trabajo escrito | h | 60 |

*Tabla 8 – Estado de mediciones del presupuesto.*

| Número de orden | Descripción de la unidad de obra | Unidad | Precio (€) | Medición | Importe (€) |
|---|---|---|---|---|---|
| **UO1** | Investigación previa y planificación | h | 6,15 | 20 | 123 |
| **UO2** | Desarrollo de la interfaz de usuario | h | 6,15 | 100 | 615 |
| **UO3** | Desarrollo informático de la aplicación | h | 6,15 | 150 | 922,5 |
| **UO4** | Desarrollo del trabajo escrito | h | 6,15 | 60 | 369 |
| PEM | | | | | 2029,5 |
| Gastos generales (12%) | | | | | 243,54 |
| Beneficio industrial (6%) | | | | | 121,77 |
| PEC | | | | | 2394,81 |
| IVA (21%) | | | | | 502,91 |
| Presupuesto base de licitación | | | | | 2897,72 |

*Tabla 9 – Presupuesto final.*

# Capítulo 8: Conclusiones

En este capítulo se comentarán las conclusiones que se han obtenido en base a la realización de este trabajo, las futuras implementaciones o cambios que se podrían realizar en la aplicación para mejorarla y la relación del trabajo con los objetivos de desarrollo sostenible de las naciones unidas.

## 8.1. Conclusiones extraídas de la realización del trabajo

### 8.1.1. Conclusión del proceso

Durante la realización de este trabajo se ha alcanzado una mejor comprensión de la cantidad de trabajo que hay que destinar a los proyectos de programación. Horas de trabajo pueden traducirse en miles de líneas de código o en una sola línea, debido a la naturaleza de la programación.

Al tener un producto tangible en el que al cual se le ha tenido que destinar tanto tiempo de trabajo (propio de este tipo de proyectos), ha resultado tener menor carácter de investigación del que se había considerado inicialmente. Gran parte del desarrollo de la aplicación ha consistido en la experimentación por prueba y error haciendo uso de los elementos existentes. En ocasiones, se ha consultado la documentación oficial de Android para descubrir que herramienta se puede ajustar a las necesidades en cada caso, intentando ampliar lo que en la asignatura se había utilizado.

Este ha sido el primer proyecto de programación de este calibre realizado por el autor, lo cual ha supuesto una experiencia enriquecedora sobre cómo abordar proyectos similares en el futuro. También ha ayudado a descubrir un gran número de herramientas y soluciones para problemas aparentemente comunes que se pueden presentar en el desarrollo de aplicaciones móviles, por lo que la realización de un proyecto de características similares, en el futuro, será más fluida y permitirá incluir más innovaciones.

### 8.1.2. Conclusión del producto

La aplicación desarrollada en el trabajo para gestionar la maquinaria e inventario en una pequeña empresa, ofrece una solución práctica que mejora la eficiencia operativa y facilita el control de recursos. Mediante de la digitalización de los procesos, la herramienta permite un seguimiento más preciso del inventario y un mantenimiento

más organizado de la maquinaria, lo que ayuda a reducir los errores y la carga de trabajo humano.

Si bien esta aplicación no pretende ser la solución definitiva para la gestión empresarial de inventarios, su implementación demuestra cómo las herramientas tecnológicas pueden simplificar tareas complejas y mejorar la toma de decisiones en entornos empresariales. El proyecto destaca por su enfoque en la resolución de problemas específicos, proporcionando una base sobre la cual se podrían desarrollar futuras mejoras y personalizaciones.

En resumen, esta aplicación representa un esfuerzo significativo por parte del autor para abordar desafíos reales en la gestión de pequeñas empresas, aportando una herramienta que, aunque básica, puede tener un impacto positivo en la organización y eficiencia de las operaciones diarias.

### 8.1.3. Posibles mejoras futuras

Hay una gran cantidad de mejoras que se podrían implementar en la aplicación, tantas cómo para aumentar el calibre del proyecto hasta hacer que la aplicación sea viable hasta para empresas de gran tamaño. No obstante, en este apartado se hablará de las implementaciones más plausibles a partir del estado actual de la aplicación. Cada una de las características a implementar podrían generar una gran cantidad de problemas nuevos a resolver, pero se considerará que esos problemas son todos solucionables.

La aplicación se ha dejado preparada para facilitar la implementación de una base de datos en línea que permita almacenar los documentos y las imágenes, para evitar que el usuario tenga dichos datos en su dispositivo móvil, lo que consumiría gran cantidad de espacio, lo que puede suponer un problema en empresas grandes.

Además, con una base de datos en línea se permitiría que varias personas puedan tener acceso a la misma aplicación y con distintos permisos de edición o visualización para compartir la carga del manejo de la aplicación entre varios departamentos.

La organización que ofrece la aplicación se podría mejorar permitiendo que el usuario pueda crear distintas pestañas de inventario o de máquinas, para tener en cuenta que la empresa puede tener múltiples negocios distintos o vender productos en distintas localizaciones.

## 8.2. Relación con los objetivos de desarrollo sostenible

Los Objetivos de Desarrollo Sostenible (en adelante ODS) son un conjunto de 17 metas globales establecidas por las Naciones Unidas en 2015 [6]. Estos objetivos buscan trabajar hacia la solución de los desafíos que enfrenta la humanidad, incluyendo la erradicación de la pobreza, la protección del planeta y la garantía de paz y prosperidad para todas las personas del planeta.

En trabajo desarrollado, diseñada para optimizar la gestión de maquinaria e inventario en una empresa pequeña, puede considerarse como relacionada de manera significativa con varios ODS, contribuyendo desde una perspectiva empresarial. A continuación, se detalla cómo el trabajo puede estar relacionado con estos objetivos globales.

### 8.2.1. ODS 3: Salud y bienestar

La aplicación ayuda a asegurar el correcto mantenimiento de la maquinaria de la empresa, reduciendo la probabilidad de accidentes laborales. Además, usar esta aplicación como gestor puede ayudar a reducir el estrés mental causado por el trabajo. Esto es gracias a la automatización del inventario, al sistema de notificaciones y al punto de información centralizada que ofrece la aplicación

### 8.2.2 ODS 8: Trabajo decente y crecimiento económico

El trabajo decente se define como un trabajo productivo que provea de ingresos dignos. La aplicación creada sirve como herramienta para ayudar a individuos a gestionar su propia empresa, lo que puedo bajar ligeramente la barrera de entrada en los sectores empresariales ya que aligera la carga del usuario. Esto ayuda a promover el trabajo decente ya que, en consecuencia, puede llegar a provocar que alguien que sin introducirse en el mundo empresarial tenga ahora un trabajo considerado decente.

No solo eso, sino que además al ayudar a la formación de empresas pequeñas esto puede crear nuevos puestos de trabajo lo que también implica más trabajo decente disponible.

### 8.2.3. ODS 9: Industria, innovación e infraestructura

Este trabajo fomenta la modernización de la infraestructura de la industria ayudando a la digitalización de los procesos de gestión, lo que mejora la eficiencia de la industria.

#### 8.2.4. ODS 12: Producción y consumo responsables

Con una gestión precisa del inventario de una empresa se pueden reducir los desperdicios de materiales y recursos. La aplicación creada, al tener un sistema de gestión de inventario que incluye la automatización de las alteraciones programadas en este, se alinea con el ODS 12.

# Bibliografía


- [1]: *Android fue el sistema operativo más utilizado por los españoles | CNMC*. (s. f.). https://www.cnmc.es/prensa/panel-hogares-usos-internet-20231103#:~:text=Un%2078%2C8%20%25%20de%20los,los%20servicios%20OTT%20en%20Espa%C3%B1a
- [2]: *Cómo descargar Android Studio y App Tools - Android Developers*. (s. f.). Android Developers. https://developer.android.com/studio?hl=es-419
- [3]: Introducing the new Bing. (2023, 14 noviembre). https://www.bing.com/new/termsofuseimagecreator
- [4]: *BOE-A-1996-8930 Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia*. (s. f.). https://www.boe.es/buscar/act.php?id=BOE-A-1996-8930#a5
- [5]: *Home - Pencil project. (s. f.)*. https://pencil.evolus.vn/
- [6]: Gamez, M. J. (2022, 24 mayo). *Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible*. *Desarrollo Sostenible*. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/
- [7]: *El origen del inventario – Servicios de Inventarios*. (s. f.). https://serviciosdeinventarios.com/el-origen-del-inventario/
- [8]: Equipo editorial, Etecé. (2023, 19 noviembre). *Hoja de Cálculo - Concepto, historia y para qué sirve*. *Concepto*. https://concepto.de/hoja-de-calculo/#:~:text=La%20primera%20hoja%20electr%C3%B3nica%20de,las%20conocemos%20es%20Dan%20Bricklin.
- [9]: C, H. (s. f.). *Fracttal, software líder en gestión de mantenimiento*. https://www.fracttal.com/es/comenzar-ahora?utm_channel=GetApp&gdmcid=9bb7a3c5-b8b1-46ff-8d88-143e20a2ff9b
- [10]: CMMS Software | *Fiix is maintenance management software*. (s. f.). https://lp.fiixsoftware.com/maintenance-management-software-product-tour.html?utm_medium=cpc&utm_campaign=maintenance_management&p1=maintenance+management+software&utm_source=GetApp&gdmcid=32d0f483-514f-4999-befe-b8fb307f6769
- [11]: *Your digital maintenance Planner • Timly software*. (s. f.). Timly Software. https://timly.com/en/your-digital-maintenance-planner/?utm_medium=paid&utm_source=gartner&utm_campaign=GetApp&gdmcid=59c5d23e-0998-4c32-91b3-8a8e8d155b4d

- [12]: Oü, P. I. /. P. (s. f.). *Pipedrive organiza tus ventas*. Pipedrive. https://www.pipedrive.com/es/gettingstarted?utm_source=naturalInt&utm_medium=directories&utm_campaign=top10crm_es&utm_content=visit_website&utm_term=dPWma46NFY
- [13]: HubSpot. (s. f.). *HubSpot CRM*. https://www.hubspot.es/crm/am-ni?irclickid=0EOQzH3oLxyKTFF0PESCKXRGUkC1Kz3JExnXU80&irgwc=1&mpid=34020&utm_id=am34020&utm_medium=am&utm_source=am34020&utm_campaign=amcid_0EOQzH3oLxyKTFF0PESCKXRGUkC1Kz3JExnXU80_irpid_34020
- *Cómo crear vistas deslizantes con pestañas a través de ViewPager2*. (s. f.). Android Developers. https://developer.android.com/guide/navigation/navigation-swipe-view-2?hl=es-419
- *Cómo crear listas dinámicas con RecyclerView*. (s. f.). Android Developers. https://developer.android.com/develop/ui/views/layout/recyclerview?hl=es-419
- CodingSTUFF. (2021, 15 octubre). *Custom Tab Layout using View Pager 2 | Android Studio 2022* [Vídeo]. YouTube. https://www.youtube.com/watch?v=EukutTtf9tQ
- Easy Tuto. (2021, 27 julio). *RecyclerView Android Studio Tutorial | 2024* [Vídeo]. YouTube. https://www.youtube.com/watch?v=TAEbP_ccjsk
- *How to start Activity in adapter?* (s. f.). Stack Overflow. https://stackoverflow.com/questions/4197135/how-to-start-activity-in-adapter
- *How to scroll whole page, Recyclerview inside of Scrollview*. (s. f.). Stack Overflow. https://stackoverflow.com/questions/72715746/how-to-scroll-whole-page-recyclerview-inside-of-scrollview
- Foxandroid. (2021, 3 mayo). *How to Add Search View in Toolbar in Android Studio | SearchView on Toolbar | Actionbar* [Vídeo]. YouTube. https://www.youtube.com/watch?v=M3UDh9mwBd8
- CodingSTUFF. (2022, 26 enero). *SearchView with RecyclerView - Android Studio Tutorial (2022)* [Vídeo]. YouTube. https://www.youtube.com/watch?v=tQ7V7iBg5zE
- Puig, I. C. (2020). *Desarrollo de la aplicación Android: Guía turística para el municipio de Cullera*. *Universitat de València*.

# Anexos

## Anexo 1: Diagrama de interconexión de las actividades

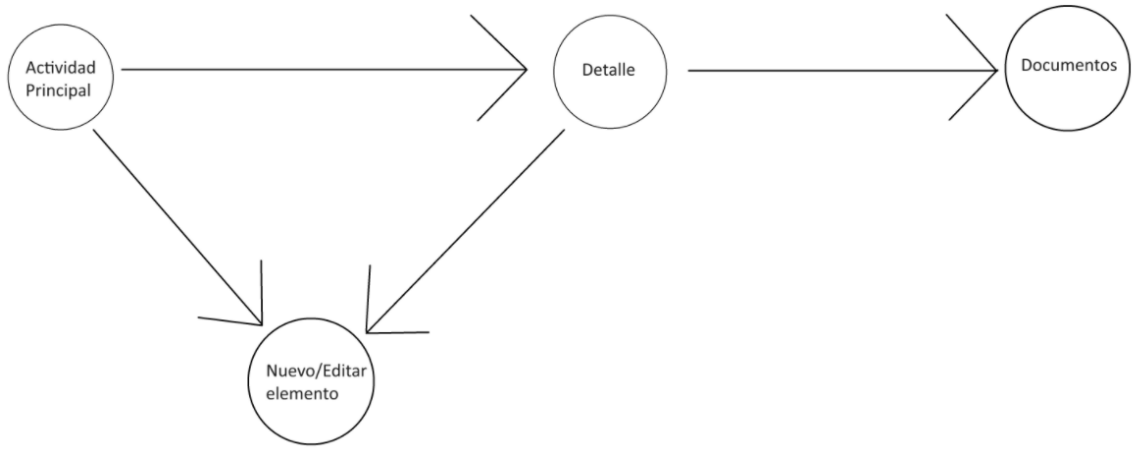


*Ilustración 19 - Diagrama de interconexión de las actividades.*

## Anexo 2: Código de las clases de la aplicación

En este anexo se comentará la función general que desempezan algunas de las clases más influyentes y se mostrará el trozo de código más importante de cada una.

### Actividad Detalle

La actividad Detalle es la actividad que se ejecuta cuando el usuario intenta acceder tanto a una máquinacomo a un objeto de inventario. Tiene que manejar en RecyclerView dentro de un NestedScrollView.

```
@Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detalle);
    Intent intent = getIntent();
```

```
    //Asignación de IDs
    recyclerViewEventos = findViewById(R.id.recyclerEventos);
    nombreDetalle = findViewById(R.id.nombreDetalle);
    descripcionDetalle = findViewById(R.id.descripcionDetalle);
    imagenDetalle = findViewById(R.id.imagenDetalle);
    editarItem = findViewById(R.id.EditarItem);
    borrarItem = findViewById(R.id.borrarItem);
    abrirDocumentos = findViewById(R.id.abrirDocumentos);
    detallesVolver = findViewById(R.id.detallesVolver);

    //Setup del recycler de eventos
    int posicion_lista = Singleton.getInstance().ArrayPositionInUse;
    tabPosition = intent.getIntExtra("Origen", 0);
    if(tabPosition == 0) {
      eventos = maquinas.get(posicion_lista).getEventoMaquina();
    } else if (tabPosition == 1) {
      eventos = inventario.get(posicion_lista).getEventoObjeto();
    }
    recyclerViewEventos.setLayoutManager(new LinearLayoutManager(this));
    recyclerViewEventos.setAdapter(new
MyRecyclerViewAdapterEventos(getApplicationContext(), eventos));


    //Escritura de datos
    if (tabPosition == 0){
      nombreDetalle.setText(maquinas.get(posicion_lista).nombreMaquina);
      descripcionDetalle.setText(maquinas.get(posicion_lista).descripcionMaquina);
      try {

imagenDetalle.setImageURI(maquinas.get(posicion_lista).getImagenMaquina());
        if (imagenDetalle.getDrawable() == null){
          imagenDetalle.setImageResource(R.drawable.image_not_available);
        }
      } catch (Exception e)
{imagenDetalle.setImageResource(R.drawable.image_not_available);}
    } else if (tabPosition == 1) {
      nombreDetalle.setText(inventario.get(posicion_lista).nombreObjeto);
      descripcionDetalle.setText(inventario.get(posicion_lista).descripcionObjeto);
      try {
```

```
            imagenDetalle.setImageURI(inventario.get(posicion_lista).getImagenObjeto());
            if (imagenDetalle.getDrawable() == null){
              imagenDetalle.setImageResource(R.drawable.image_not_available);
            }
          } catch (Exception e)
{imagenDetalle.setImageResource(R.drawable.image_not_available);}
        }

        borrarItem.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View view) {
            AlertDialog.Builder builder = new AlertDialog.Builder(Detalle.this);
            builder.setTitle("¿Borrar " + nombreDetalle.getText().toString() + "?")
                .setMessage("Esta acción es permanente")
                .setPositiveButton("Confirmar", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialogInterface, int i) {
                    if (tabPosition == 0){
                      Singleton.getInstance().maquinariaList.remove(posicion_lista);
                    } else if (tabPosition == 1) {
                      Singleton.getInstance().inventarioList.remove(posicion_lista);
                    }
                    finish();
                  }
                })
                .setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialogInterface, int i) {
                    dialogInterface.dismiss();
                  }
                });
            builder.create().show();
          }
        });

        //El botón Edit lanza la actividad de crear nuevo item pero con los datos del
        //Item detallado ya escritos para poder cambiarlos a voluntad
        editarItem.setOnClickListener(new View.OnClickListener() {
          @Override
```

```
        public void onClick(View view) {
            Intent intent = new Intent(Detalle.this, NuevoItem.class);
            intent.putExtra("Nuevo elemento", false);
            intent.putExtra("Origen", tabPosition);
            startActivity(intent);
        }
    });

    abrirDocumentos.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent1 = new Intent(Detalle.this, Documentos.class);
            startActivity(intent1);
        }
    });

    detallesVolver.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            finish();
        }
    });

}
```

### Actividad Documentos

La actividad de documentos se compone únicamente de un RecyclerView, dos imágenes y un botón. Su propósito es mostrar al usuario y permitirle acceder documentos que él haya decidido guardar ahí previamente para tener toda la información pertinente a una máquina u objeto de inventario en un solo sitio.

```
    //Lector de PDFs
    selectorPDF = registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(), result -> {
        //Obtencion de lista
```

```
        if (Singleton.getInstance().ArrayInUse == 0){
          documentos =
Singleton.getInstance().maquinariaList.get(Singleton.getInstance().ArrayPositionInUse)
.getDocumentos();
        } else// if (Singleton.getInstance().ArrayInUse == 1)
        {
          documentos =
Singleton.getInstance().inventarioList.get(Singleton.getInstance().ArrayPositionInUse).g
etDocumentos();
        }

        documentosAdapter.documentos=documentos;

        if (result.getResultCode() == Activity.RESULT_OK){
          Intent intent = result.getData();
          if (intent != null) {
            try {

            Uri uriPDF = intent.getData();
            Calendar calendar = Calendar.getInstance();
              documentos.add(new ItemDocumento(uriPDF.toString(),
calendar.get(Calendar.DAY_OF_MONTH), calendar.get(Calendar.MONTH)+1,
calendar.get(Calendar.YEAR)));

            Singleton.getInstance().save();
            //recyclerViewDocumentos.setAdapter(new
MyRecyclerViewAdapterDocumentos(Documentos.this, documentos));
            documentosAdapter.notifyDataSetChanged();
            }catch (Exception e){
              Log.e("error",e.getMessage());

            }
          }
        }else{
          Log.e("error",result.toString());
        }
      });
```

```
addPDF.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
        intent.setType("application/pdf");
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        selectorPDF.launch(intent);
    }
});
```

### Fragmento Inventario

El fragmento Inventario realiza la misma función que una actividad normal, solo que se ejecuta dentro de una pestaña del ViewPager2.

```
//Setup del recycler de inventario
recyclerViewInventario.setLayoutManager(new LinearLayoutManager(getActivity()));
inventarioAdapter = new MyRecyclerViewAdapterInventario(getActivity().getApplicationContext(), itemInventario);
recyclerViewInventario.setAdapter(inventarioAdapter);
```

### Clase ItemDocumento

Esta clase es un modelo de lista para guardar un conjunto de datos específicos que abarcan la información necesaria para cargar cada documento guardado por el usuario. Todas las clases que comienzan su nombre por “Item” en este proyecto desempeñan la misma función que esta con distintos datos guardados.

```
public ItemDocumento(String direccionPDF, int dia, int mes, int anyo) {
    this.direccionPDF = direccionPDF;
    this.dia = dia;
    this.mes = mes;
```

```
    this.anyo = anyo;
  }
```

### Actividad MainActivity

En esta actividad se ejecuta el ccódigo relacionado a las pestañas y al ViewPager2 que permite cabiar entre la vista de máquinas y la de inventario de forma fluida.

```
    //Montaje de las pestañas
    FragmentManager fragmentManager = getSupportFragmentManager();
    adapter = new MyFragmentAdapter(fragmentManager, getLifecycle());
    viewPager.setAdapter(adapter);

    //Primero haremos que las dos pestañas se carguen un instante para que no
    //Haya problemas al hacer una busqueda nada mas despues de iniciar la busqueda
    viewPager.setCurrentItem(1);
    viewPager.setCurrentItem(0);

    //Listener de las pestañas
    tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
      @Override
      public void onTabSelected(TabLayout.Tab tab) {
        tabPosition = tab.getPosition();
        viewPager.setCurrentItem(tab.getPosition());
      }

      @Override
      public void onTabUnselected(TabLayout.Tab tab) {

      }

      @Override
      public void onTabReselected(TabLayout.Tab tab) {

      }
    });
```

```
    //Esto hace que al cambiar de pestaña arrastrando la pantalla se actualice tambien
el
    //Indicador inferior que señala la pestaña actual
    viewPager.registerOnPageChangeCallback(new
ViewPager2.OnPageChangeCallback() {
      @Override
      public void onPageSelected(int position) {
        tabLayout.selectTab(tabLayout.getTabAt(position));
      }
    });
```

### Fragmento Maquinaria

El fragment Maquinaria se comporta como una actividad con la diferencia de que se está ejecutando en el interior de una de las vistas de el ViewPager2 de la actividad principal. Además, dispone de una característica repetida en los fragmentos de Inventario y Documentos, la cual es la función que recibe los datos introducidos por el usuario al abrir un menú de contexto.

```
  @Override
  public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    recyclerViewMaquinaria = view.findViewById(R.id.recyclerMaquinaria);
    registerForContextMenu(recyclerViewMaquinaria);

    //Setup del recycler de maquinaria
    recyclerViewMaquinaria.setLayoutManager(new GridLayoutManager(getActivity(),
2));
    maquinariaAdapter = new
MyRecyclerViewAdapterMaquinaria(getActivity().getApplicationContext(),
itemMaquinarias);
    recyclerViewMaquinaria.setAdapter(maquinariaAdapter);
```

### Adaptador MyRecyclerViewAdapterDocumentos

Los adaptadores se usan en este proyecto para ejecutar el código pertinente a cada elemento de una lista recurrente con datos individuales como la RecyclerView. Esto se cumple para todos los adaptadores excepto MyFragmentAdapter, que simplemente se encarga asignar los valores de las pestañas de la actividad principal.

```
  public String getNombrePDF(Uri uri){
    String nombrePDF = null;
    if (uri.getScheme().equals("content")){
      try (Cursor cursor = context.getContentResolver().query(uri, null, null, null, null)){
        if (cursor != null && cursor.moveToFirst()){
          int nameIndex = cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME);
          if (nameIndex >= 0){
            nombrePDF = cursor.getString(nameIndex);
          }
        }
      }
    }

    //En caso de que el esquema no sea content seguramente sea file, en cuyo caso esto debería bastar
    if (nombrePDF == null){
      nombrePDF = uri.getLastPathSegment();
    }
    return nombrePDF;
  }
}
```

### ViewHolder MyViewHolderDocumentos

Todos los ViewHolders de este trabajo solo se usan para relacionar elementos de diseño visibles con asginaciones para las listas recurrentes de RecyclerView.

```
public class MyViewHolderDocumentos extends RecyclerView.ViewHolder{
  TextView documentoFecha, documentoTitulo;
```

```
  ImageView documentoIcono;

  public MyViewHolderDocumentos(@NonNull View itemView) {
    super(itemView);
    documentoFecha = itemView.findViewById(R.id.documentoFecha);
    documentoTitulo = itemView.findViewById(R.id.documentoTitulo);
    documentoIcono = itemView.findViewById(R.id.documentoIcono);
  }
}
```

### BroadcastReciever Notificacion

Esta es posiblemente la clase más compleja del trabajo ya que es la única que interactua de alguna forma con elementos del Sistema operative externos a la aplicación. La clase se encarga de recibir alarmas que la avisan de eneviar notificaciones de los eventos cada día y programar una alarma para el día siguiente. Esta clase será mostrada al completo.

```
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

import androidx.core.app.NotificationCompat;
import androidx.core.app.NotificationManagerCompat;

import java.util.Calendar;

public class Notification extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    if (!Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {
      //Setup de el canal de notificacón
      NotificationChannel channel = new NotificationChannel("evento", "evento",
NotificationManager.IMPORTANCE_HIGH);
```

```
        NotificationManager manager =
context.getSystemService(NotificationManager.class);
        manager.createNotificationChannel(channel);

        Calendar calendar = Calendar.getInstance();
        Calendar calendarAux = Calendar.getInstance();

        //Setup notificación
        //Este método es necesario en vez de programar alarmas individuales por cada
evento porque android parece evitar que algunas alarmas se disparen si hay varias
puestas para el mismo timepo
        Singleton.getInstance(context).load();
        for (int i = 0; i < 2; i++) {
          if (i == 0) {
            for (int j = 0; j < Singleton.getInstance().maquinariaList.size(); j++) {
              int size =
Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().size();
              NotificationCompat.BigTextStyle bigTextStyle = new
NotificationCompat.BigTextStyle();
              StringBuilder bigTextBuilder = new StringBuilder();
              for (int g = 0; g < size; g++) {
                int dia, mes, anyo;
                dia =
Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).getDia();
                mes =
Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).getMes();
                anyo =
Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).getAnyo();
                //Se mete en el calendario auxiliar para que en el caso de tener una fecha
invalida (como el 32 de enero) se ajuste automáticamente)
                calendarAux.set(anyo, mes - 1, dia);
                if ((calendarAux.get(Calendar.DAY_OF_MONTH) ==
calendar.get(Calendar.DAY_OF_MONTH)
                    && (calendarAux.get(Calendar.MONTH) ==
calendar.get(Calendar.MONTH))
                    && (calendarAux.get(Calendar.YEAR) ==
calendar.get(Calendar.YEAR)))) {
                  if
(Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).isEstadoActua
```

```
l()) {

bigTextBuilder.append(Singleton.getInstance().maquinariaList.get(j).getEventoMaquina
().get(g).getNombreEvento() + "\n");
                    if
(!Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).isRecurrente(
)){

Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).setEstadoActu
al(false);
                    }
                }

                //Incluso si no está activo el evento recurrente actualizará su fecha para
que sea facilmente reactivable para el usuario
                if
(Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).isRecurrente()
) {
                    int magnitudRepeticion, unidadRepeticion;
                    magnitudRepeticion =
Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).getMagnitudRe
peticion();
                    unidadRepeticion =
Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).getUnidadRep
eticion();

                    //Da igual si la fecha que se crea es una no existente ya que luego al
meterla en un calendario se ajusta automaticamente
                    if (unidadRepeticion == 0) {
                      dia += magnitudRepeticion;
                    } else if (unidadRepeticion == 1) {
                      mes += magnitudRepeticion;
                    } else if (unidadRepeticion == 2) {
                      anyo += magnitudRepeticion;
                    }
                    calendarAux.set(anyo, mes - 1, dia);

Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).setDia(calenda
rAux.get(Calendar.DAY_OF_MONTH));
```

```
Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).setMes(calend
arAux.get(Calendar.MONTH) + 1);

Singleton.getInstance().maquinariaList.get(j).getEventoMaquina().get(g).setAnyo(calen
darAux.get(Calendar.YEAR));
                    Singleton.getInstance().save();
                  }
                }
              }
              if (!bigTextBuilder.toString().isEmpty()) {
                bigTextStyle.bigText(bigTextBuilder.toString());
                android.app.Notification summary = new
NotificationCompat.Builder(context, "evento")

.setContentTitle(Singleton.getInstance().maquinariaList.get(j).getNombreMaquina())
                    .setSmallIcon(R.drawable.ic_notification)

.setContentText(Singleton.getInstance().maquinariaList.get(j).getNombreMaquina())
                    .setStyle(bigTextStyle)
                    .setPriority(NotificationCompat.PRIORITY_HIGH)

.setGroup(Singleton.getInstance().maquinariaList.get(j).getNombreMaquina())
                    .build();
                //Enseñar notifiación
                NotificationManagerCompat notificationManager =
NotificationManagerCompat.from(context);
                //Los permisos se comprueban en el botón confirmar de NuevoItem
                notificationManager.notify(((i << 16) + (j)), summary);
              }
            }
          } else {
            for (int j = 0; j < Singleton.getInstance().inventarioList.size(); j++) {
              NotificationCompat.BigTextStyle bigTextStyle = new
NotificationCompat.BigTextStyle();
              StringBuilder bigTextBuilder = new StringBuilder();
              for (int g = 0; g <
Singleton.getInstance().inventarioList.get(j).getEventoObjeto().size(); g++) {
                int dia, mes, anyo;
```

```
                dia =
Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).getDia();
                mes =
Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).getMes();
                anyo =
Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).getAnyo();
                if ((dia == calendar.get(Calendar.DAY_OF_MONTH)
                    && (mes == (calendar.get(Calendar.MONTH) + 1))
                    && (anyo == calendar.get(Calendar.YEAR)))) {
                  if
(Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).isEstadoActual())
{
                    //La lógica que edita la cantidad de cada objeto de inventario afectada
por el evento está aquí.
                    int magnitudDiferencial =
Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).getSumaRestaMag
nitud();
                    boolean signoDiferencial =
Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).sumaRestaSigno;
                    int cantidadActual =
Singleton.getInstance().inventarioList.get(j).getCantidadObjeto();
                    if (signoDiferencial){
                      cantidadActual -= magnitudDiferencial;
                    } else {
                      cantidadActual += magnitudDiferencial;
                    }
                    if (cantidadActual < 0) {
                      Singleton.getInstance().inventarioList.get(j).setCantidadObjeto(0);

bigTextBuilder.append(Singleton.getInstance().inventarioList.get(j).getEventoObjeto().g
et(g).getNombreEvento() + " - existencias insuficientes"+ "\n");
                    } else {

Singleton.getInstance().inventarioList.get(j).setCantidadObjeto(cantidadActual);

bigTextBuilder.append(Singleton.getInstance().inventarioList.get(j).getEventoObjeto().g
et(g).getNombreEvento() + "\n");
                    }
                    if
```

```
(!Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).isRecurrente()){
                        //Desactiva el evento si no es recurrente para que no suene el mismo día si el usuario decir editar los eventos
                        //Así se evita que el mismo día se reste o sume dos veces la cantidad de un solo evento

Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).setEstadoActual(false);
                    }
                }

                //Incluso si no está activo el evento recurrente actualizará su fecha para que sea facilmente reactivable para el usuario
                if (Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).isRecurrente()) {
                    int magnitudRepeticion, unidadRepeticion;
                    magnitudRepeticion = Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).getMagnitudRepeticion();
                    unidadRepeticion = Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).getUnidadRepeticion();

                    //Da igual si la fecha que se crea es una no existente ya que luego al meterla en un calendario se ajusta automáticamente
                    if (unidadRepeticion == 0) {
                        dia += magnitudRepeticion;
                    } else if (unidadRepeticion == 1) {
                        mes += magnitudRepeticion;
                    } else if (unidadRepeticion == 2) {
                        anyo += magnitudRepeticion;
                    }

Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).setDia(dia);

Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).setMes(mes);

Singleton.getInstance().inventarioList.get(j).getEventoObjeto().get(g).setAnyo(anyo);
                    Singleton.getInstance().save();
```

```
                    }
                  }
                }
                if (!bigTextBuilder.toString().isEmpty()) {
                  bigTextStyle.bigText(bigTextBuilder.toString());
                  android.app.Notification summary = new
NotificationCompat.Builder(context, "evento")

.setContentTitle(Singleton.getInstance().inventarioList.get(j).getNombreObjeto())
                      .setSmallIcon(R.drawable.ic_notification)

.setContentText(Singleton.getInstance().inventarioList.get(j).getNombreObjeto())
                      .setStyle(bigTextStyle)
                      .setPriority(NotificationCompat.PRIORITY_HIGH)

.setGroup(Singleton.getInstance().inventarioList.get(j).getNombreObjeto())
                      .build();
                  //Enseñar notifiación
                  NotificationManagerCompat notificationManager =
NotificationManagerCompat.from(context);
                  //Los permisos se comprueban en el botón confirmar de NuevoItem
                  notificationManager.notify(((i << 16) + (j)), summary);
                }
              }
            }
          }
          calendar = Calendar.getInstance();
          calendar.add(Calendar.DAY_OF_MONTH, 1);
          calendar.set(Calendar.HOUR_OF_DAY, 6);
          calendar.set(Calendar.MINUTE, 0);
          calendar.set(Calendar.SECOND, 0);
          NuevoItem.programarNotificacion(context, calendar.getTimeInMillis());
      } else {
          //En caso de que se reinicie el dispositivo se lanza una alarma para volver a iniciar
el bucle diario
          NuevoItem.programarNotificacion(context, 0);
      }
    }
}
```

### Clase Singleton

La clase Singleton sirve como centro de información compartido entre todas las clases en una única instancia, ya que usando el lenguaje de programación JAVA no es sencillo acceder a los datos de otra clase y puede ser confuse averiguar donde está cada pieza de información. Esta clase es la principal responsable de haber podido desarrollar un Proyecto organizado.

```
public class Singleton{
  private static Singleton instance;

  public ArrayList<ItemMaquinaria> maquinariaList;
  public ArrayList<ItemEvento> eventoList;
  public ArrayList<ItemInventario> inventarioList;
  SharedPreferences preferences;
  Gson gson;
  Context context;
  public int ArrayPositionInUse, ArrayInUse, ArrayPositionDocumentos;
  EditText openedCantidadDiferencial;
  Button openedBotonSumar, openedBotonRestar;
  MyRecyclerViewAdapterNuevoEventoInventario myAdapterInventario;
  MyRecyclerViewAdapterNuevoEventoMaquina myAdapterMaquinas;
  RecyclerView recyclerViewEventosNuevos;
  Boolean permitirContexto;
  private Singleton() {
    maquinariaList=new ArrayList<ItemMaquinaria>();
    eventoList=new ArrayList<ItemEvento>();
    inventarioList=new ArrayList<ItemInventario>();
    context = MainActivity.context;
    preferences = context.getSharedPreferences("SavedData",
Context.MODE_PRIVATE);
    gson = new Gson();
    openedCantidadDiferencial = null;
    openedBotonSumar = null;
    openedBotonRestar = null;
    permitirContexto = true;
```

```
}

public static Singleton getInstance() {
   if (instance == null) {
      instance = new Singleton();
      instance.load();
   }
   return instance;
}

   public static Singleton getInstance(Context context) {
   if (instance == null) {
      instance = new Singleton(context);
      instance.load();
   } else {
      instance.context = context;
   }
   return instance;
}

public void load(){
   String jsonMaquinaria = preferences.getString("Maquinas", null);
   if (jsonMaquinaria == null){
      maquinariaList = new ArrayList<>();
   } else {
      Type type = new TypeToken<List<ItemMaquinaria>>() {}.getType();
      maquinariaList = gson.fromJson(jsonMaquinaria, type);
   }

   String jsonInventario = preferences.getString("Inventario", null);
   if (jsonInventario == null){
      inventarioList = new ArrayList<>();
   } else {
      Type type = new TypeToken<List<ItemInventario>>() {}.getType();
      inventarioList = gson.fromJson(jsonInventario, type);
   }
}

public void save(){
```

```
        //Save máquinas
        String jsonMaquinas = gson.toJson(maquinariaList);
        preferences.edit().putString("Maquinas", jsonMaquinas).apply();

        //Save inventario
        String jsonInventario = gson.toJson(inventarioList);
        preferences.edit().putString("Inventario", jsonInventario).apply();
    }
}
```