



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

 etsinf

Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación en Android para la gestión de inspecciones de calidad en entornos laborales.

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jose Collado San Pedro

Tutor: Vicente Pelechano Ferragud

Curso 2018-2019

Resum

L'arribada de les noves tecnologies ha condicionat notablement la vida de les persones i la majoria d'accions que realitzem en el nostre dia a dia. Un exemple perfecte són els telèfons mòbils intel·ligents o *smartphones*, l'objectiu dels quals persegueix ajudar-nos i facilitar-nos en moltes de les tasques més quotidianes.

En aquest treball es pretén oferir una solució de programari per a empreses i institucions involucrades en processos de registres de qualitat i auditòries. Amb aquest gestor de qualitat pretenem reduir temps i costos dels actuals processos existents a la indústria així com millorar la persistència i visualització de les dades obtingudes.

Per a això, implementarem una aplicació Android que farà ús de geolocalització mitjançant GPS i escaneig de codis QR. Emmagatzemarem les dades en una base de dades perquè puguen ser consultades amb posterioritat tant pels usuaris com pels administradors. A més, posarem a disposició de les empreses la capacitat de personalitzar certs aspectes de l'aplicació perquè s'adapte a les seues necessitats i requisits.

Durant aquest TFG mostrarem com hem dut a terme cadascun dels processos involucrats en la creació d'aquesta aplicació mòbil.

Paraules clau: Android, gestor qualitat, codis QR, geolocalització, inspecció entorns laborals, base de dades, API

Resumen

La llegada de las nuevas tecnologías ha condicionado notablemente la vida de las personas y la mayoría de acciones que realizamos en nuestro día a día. Un ejemplo perfecto son los teléfonos móviles inteligentes o *smartphones*, cuyo objetivo persigue ayudarnos y facilitarnos en muchas de las tareas más cotidianas.

En este trabajo se pretende ofrecer una solución de software para empresas y instituciones involucradas en procesos de registros de calidad y auditorías. Con dicho gestor de calidad pretendemos reducir tiempos y costes de los actuales procesos existentes en la industria así como mejorar la persistencia y visualización de los datos obtenidos.

Para ello, implementaremos una aplicación Android que hará uso de geolocalización mediante GPS y escaneo de códigos QR. Almacenaremos los datos en una base de datos para que puedan ser consultados con posterioridad tanto por los usuarios como por los administradores. Además, pondremos a disposición de las empresas la capacidad de personalizar ciertos aspectos de la aplicación para que se adapte a sus necesidades y requisitos.

Durante este TFG mostraremos como hemos llevado a cabo cada uno de los procesos involucrados en la creación de esta aplicación móvil.

Palabras clave: Android, gestor calidad, códigos QR, geolocalización, inspección entornos laborales, bases de datos, API

Abstract

The arrival of new technologies has strongly conditioned the lives of people and the majority of actions we perform in our day to day. A perfect example are smartphones, which aim to help and facilitate us in many of our daily tasks.

This work aims to offer a software solution for companies and institutions involved in quality registration processes and audits. With this quality manager we intend to reduce times and costs of the current processes in the industry as well as improve the persistence and visualization of the data obtained.

In order to do this, we will implement an Android application that will use GPS geolocation and QR code scanning. We will store the data in a database so that they can be consulted later by both users and administrators. In addition, we will give to the companies the possibility to customize certain aspects of the application to suit their needs and requirements.

During this TFG we will show how we have carried out each of the processes involved in the creation of this mobile application.

Key words: Android, quality manager, QR code, geolocation, work environments inspection, databases, API

Índice general

Índice general	v
Índice de figuras	vii
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Impacto esperado	2
1.4 Estructura de la memoria	2
1.5 Metodología	3
2 Estado del arte	7
2.1 iAuditoría	7
2.2 Calidad Cloud	7
2.3 CTI gCalidad	8
2.4 Crítica al estado del arte	10
2.5 Solución propuesta	10
3 Análisis	13
3.1 Casos de uso	13
3.1.1 Iniciar sesión	14
3.1.2 Realizar registro calidad	17
3.1.3 Consultar registros	17
3.1.4 Cambiar contrata	17
3.1.5 Cambiar idioma	18
3.1.6 Mostrar documento adjunto	18
3.1.7 Mostrar instrucciones	18
3.1.8 Mostrar términos	19
3.1.9 Contactar con administrador	19
3.1.10 Enviar sugerencia	19
3.1.11 Contactar servicio técnico	20
3.2 Entidades	20
3.2.1 Operario	20
3.2.2 Contrata	20
3.2.3 Registro	21
4 Diseño	23
4.1 Arquitectura de software	23
4.1.1 MVP	23
4.2 Diseño de interfaces	24
4.2.1 Inicio de sesión o <i>login</i>	24
4.2.2 Menú principal	24
4.2.3 Inspección	25
4.2.4 Consulta	27
4.2.5 Perfil de usuario	27
4.2.6 Ayuda	28

5 Desarrollo e implementación	29
5.1 Servidor	29
5.1.1 Amazon EC2	29
5.1.2 Ubuntu Server	30
5.1.3 LAMP	30
5.2 API REST	34
5.3 Aplicación de Android	38
5.3.1 Android	38
5.3.2 Java	39
5.3.3 Tecnologías utilizadas	40
5.3.4 Estructura del proyecto Android	43
5.3.5 Desarrollo de funcionalidad: Enviar email.	44
5.3.6 Desarrollo de funcionalidad: Lectura de código QR.	46
6 Resultado final y conclusiones	51
6.1 Cumplimiento de los objetivos	51
6.2 Relación del trabajo desarrollado con los estudios cursados	53
Glosario	55
Bibliografía	57

Índice de figuras

1.1 Ciclo de vida durante el desarrollo ágil de software	3
1.2 Tareas incluidas en cada sprint	5
1.3 Modelo de iteraciones usando agilismo	5
2.1 Capturas de pantalla durante el uso de la app iAuditoria	8
2.2 Portal web de Calidad Cloud	9
2.3 Ventanas principales de la app CTI gCalidad	9
3.1 Cuota de mercado de sistemas operativos móviles a principios de 2019 en España	14
3.2 Diagrama de tipos de usuario y plataformas de acceso a GdQ	15
3.3 Diagramas de casos de uso del usuario y administrador	16
3.4 Diagrama de clases de las entidades	21
4.1 Esquema de componentes del patrón MVP	24
4.2 Pantalla de inicio de sesión	25
4.3 Pantalla principal o <i>home</i>	25
4.4 Pantalla de escaneo de QR	26
4.5 Pantalla de registro de inspección	26
4.6 Pantalla de consulta de registros	27
4.7 Pantalla de listado de registros	27
4.8 Pantalla de perfil de usuario	28
4.9 Pantalla de ayuda	28
5.1 Configuración de instancia EC2: Sistema operativo	31
5.2 Configuración de instancia EC2: Hardware	31
5.3 Configuración de instancia EC2: Almacenamiento	31
5.4 Configuración de instancia EC2: Generación clave privada	31
5.5 Panel de control de EC2	31
5.6 Grupos de seguridad de la instancia de EC2	32
5.7 Página HTML predeterminada del servidor web Apache	33
5.8 Página de información de PHP	33
5.9 Menú principal de phpMyAdmin	34
5.10 Diagrama básico de flujo al usar una API	35
5.11 Documentación generada de nuestra API	36
5.12 Versiones de Android y porcentaje de instalaciones en junio de 2019	39
5.13 El IDE Android Studio.	40
5.14 Diagrama de capas de software o <i>stack</i> que componen Android.	41
5.15 Estructura del proyecto GdQ en Android Studio.	44
5.16 Interfaz que implementa la Vista	45
5.17 Vista o Activity	48
5.18 Presentador	49
5.19 Modelo de la entidad Operario	50

6.1	Panel web para gestión de terminales.	51
6.2	Pop-up de error en login.	52
6.3	Realizar registro desde la app.	52
6.4	Consulta de registros realizados.	52
6.5	Menú seleccionable para elegir contrata.	53

CAPÍTULO 1

Introducción

En la actualidad el uso de la tecnología y en concreto de los dispositivos móviles se ha convertido en una parte indispensable de nuestras vidas. El uso de estos últimos ha supuesto grandes beneficios y ventajas en muchas acciones que realizamos en el día a día, incluidas en nuestros entornos laborales, ámbito en el que se centra este trabajo.

La amplia presencia de teléfonos móviles *Android* facilita que las empresas puedan desarrollar aplicaciones que cumplan con las necesidades que requieren sin tener que realizar grandes inversiones en equipamiento electrónico.

El siguiente TFG detalla una solución de software para empresas que necesiten registrar estados de calidad de diferentes localizaciones o elementos. Consistirá en una *app* para Android con la que los trabajadores de cierto sector o empresa (p. ej. sector limpieza) podrán crear registros de calidad de diferentes localizaciones marcadas con códigos *QR* que deberán escanear. La aplicación permitirá informar del estado de cada localización y sus respectivos elementos además de enviar en cada registro las coordenadas *GPS* del teléfono. Contará con la funcionalidad de consultar registros previos y aplicar filtros de búsqueda. Además, tendrá en cuenta 2 tipos de usuario (Inspector y Operario) y la interfaz mostrará o ocultará ciertas funciones dependiendo de su grado de permisos. Para el almacenamiento y consulta de toda la información relacionada con los registros, usuarios, etc., se utilizará una *API* que conectará con una base de datos *MySQL*.

Aunque esta solución está pensada para ciertos sectores empresariales concretos, está previsto adaptarla en un futuro para registro de fichajes en colegios o universidades (entrada y salida de clases).

1.1 Motivación

Uno de los motivos principales por los que se planteó el desarrollo de esta solución fue la necesidad de informatizar y optimizar el proceso del control de calidad para empresas de limpieza.

Varias empresas del sector nos reportaron que en ese momento dicho proceso no era efectivo, se solía perder información y además al ser un proceso analógico, se perdía también bastante tiempo.

Además, existen planes de implantación de nuevas tecnologías llevados a cabo por Administraciones públicas. Debido a esto, en los concursos públicos para adjudicaciones de limpieza de edificios públicos se valora positivamente la informatización y automatización de dichos procesos.

Por todo esto decidimos iniciar el desarrollo de la app *GdC* (Gestor de Calidad). Nuestra intención es implantarlo en empresas privadas y públicas del sector de limpieza y similares que estén interesadas.

1.2 Objetivos

El objetivo principal del proyecto es el desarrollo de una *app* que sea capaz de llevar a cabo las siguientes acciones:

- Permitir que los usuarios se identifiquen en el sistema donde previamente han sido dados de alta por el administrador.
- Realizar registros de calidad escaneando el *código QR* o seleccionando la localización de una lista.
- Consultar registros realizados anteriormente aplicando filtros de búsqueda.
- Seleccionar sobre qué contrata se va a realizar el registro, pudiendo el operario o usuario estar dado de alta en varias al mismo tiempo.
- Contactar con el administrador de la contrata y mostrar el horario laboral del empleado.

Además de estas funcionalidades, que se explican más detalladamente en la sección **4** dedicada al diseño de la solución, tenemos como objetivo la utilización de metodologías ágiles, buenas prácticas de escritura de código y implantación de arquitecturas de software.

1.3 Impacto esperado

Esperamos que la herramienta permita a empresas y trabajadores ahorrar tiempo y esfuerzo en una tarea hasta ahora costosa. De la misma forma, se espera que elimine la posible pérdida de datos derivada de realizar los registros manualmente en papel y que los administradores puedan consultar y realizar estadísticas desde cualquier lugar mediante los teléfonos móviles.

Finalmente, esperamos poder introducir la aplicación en empresas y concursos públicos donde sea de utilidad informatizar y mejorar el proceso existente.

1.4 Estructura de la memoria

La memoria seguirá una estructura basada en las guías y recomendaciones proporcionadas por la *ETSINF*. En el capítulo **2** empezaremos comentando el estado del arte y diferentes soluciones similares a la nuestra existentes en el mercado. El capítulo **3** lo dedicaremos al análisis de la solución que proponemos, incluyendo casos de uso, diagramas de entidades, etc. Seguidamente, en la sección **4** hablaremos del diseño de la arquitectura de software que hemos elegido así como el diseño de interfaces. En el capítulo **5** detallaremos la implementación que hemos realizado en los 3 componentes que forman parte de la solución final.

1.5 Metodología

A continuación se explica la metodología que se ha llevado a cabo para el desarrollo del trabajo. Inicialmente se declararon una serie de requisitos que creíamos que la aplicación debía incorporar. Además, se realizó una búsqueda de aplicaciones similares ya existentes en *Google Play Store* y se hizo un estudio de funcionalidades que no encontramos en dichas apps y que pensábamos que eran interesantes. Esto nos permite diferenciarnos del resto de aplicaciones competidoras.

Como hemos mencionado anteriormente, el proyecto se ha centrado en torno al desarrollo ágil o *Agile Software Development*. El agilismo consiste en un proceso iterativo e incremental que permite adaptarse fácilmente a los cambios de requisitos que surgen durante el desarrollo involucrando de forma activa al cliente.

El objetivo de cada iteración es incrementar el valor del software añadiendo funcionalidades y entregar un software eficaz y sin errores. Dichas iteraciones o *sprints* incluyen todas las siguientes etapas: planificación, análisis de requisitos, diseño, desarrollo, pruebas y documentación. En 2001 un grupo de 17 desarrolladores de software publicaron lo que conocemos como *El manifiesto para el desarrollo de software ágil*. Dicho manifiesto se puede resumir en estos cuatro puntos:

1. Individuos e interacciones sobre procesos y herramientas.
2. Software funcionando sobre documentación extensiva.
3. Colaboración con el cliente sobre negociación contractual.
4. Respuesta ante el cambio sobre seguir un plan.

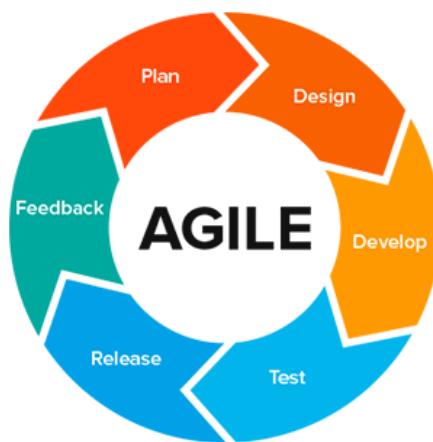


Figura 1.1: Ciclo de vida durante el desarrollo ágil de software

En la actualidad existen diferentes métodos ágiles cada uno con características, roles y métodos de trabajo diferentes. Aunque podemos encontrar similitudes entre ellas y todas ellas comparten bastante la filosofía del agilismo, no es recomendable seguir una metodología al 100 % sino utilizar varias de ellas combinándolas de forma que satisfagan nuestras necesidades, ya que cada una hace énfasis en uno u otro aspecto del proceso de desarrollo: el factor humano, las tareas, los artefactos, etc. Las más conocidas son: Scrum, eXtreme Programming, Crystal Clear o Kanban.

Así pues, algunas de las ventajas que nos aportan las metodologías ágiles son:

1. Cumplir con los requisitos que cambian durante el desarrollo sin alterar costes ni tiempos de entrega.
2. Mejorar la calidad del producto final.
3. Simplificar el proceso de forma que permita reducir el volumen de trabajo y tiempo de gestión.
4. Hacer más fácil el trabajo a los desarrolladores.
5. Aumentar productividad y hacer más trabajo en la misma cantidad de tiempo.

Nuestro proyecto de software se ha decidido planificar en 8 sprints de 1 semana cada uno. Esta decisión ha sido tomada entre los desarrolladores y el cliente final. Aun así, se planificarán más sprint una vez terminado el desarrollo para dar soporte a incidencias y errores que puedan surgir.

En la figura 1.2 podemos ver como hemos realizado la división del trabajo en sprints y qué tareas hemos incluido en cada uno.

Respecto a la metodología comunicativa con el tutor, se han realizado consultas y revisiones mediante correo electrónico y reuniones en persona con el fin de aportar *feedback* y mejoras al trabajo.

<p>▼ Tablero Sprint 1 2 incidencias 1/Dec/18 09:19 AM • 2/Dec/18 09:19 AM</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-3 Especificación requisitos <input checked="" type="checkbox"/> TFG-4 Reunión Cliente <p>+ Crear incidencia</p>	<p>▼ Tablero Sprint 4 3 incidencias</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-10 Añadir opción elegir contrata <input checked="" type="checkbox"/> TFG-11 Ventana Registros <input checked="" type="checkbox"/> TFG-20 Reunión Cliente <p>+ Crear incidencia</p>
<p>▼ Tablero Sprint 2 3 incidencias</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-5 Pantalla Login <input checked="" type="checkbox"/> TFG-6 Menu Principal <input checked="" type="checkbox"/> TFG-8 Reunión Cliente <p>+ Crear incidencia</p>	<p>▼ Tablero Sprint 5 3 incidencias</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-12 Ventana Consulta <input checked="" type="checkbox"/> TFG-13 Ventana Ayuda <input checked="" type="checkbox"/> TFG-21 Reunión cliente <p>+ Crear incidencia</p>
<p>▼ Tablero Sprint 3 2 incidencias</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-7 Ayuda despliegue servidor <input checked="" type="checkbox"/> TFG-9 Testeo Webservice <p>+ Crear incidencia</p>	<p>▼ Tablero Sprint 6 3 incidencias</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-14 Ventana Perfil <input checked="" type="checkbox"/> TFG-15 Integración con Google Maps <input checked="" type="checkbox"/> TFG-16 Corrección errores
<p>▼ Tablero Sprint 7 3 incidencias</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-17 Actualización permisos app <input checked="" type="checkbox"/> TFG-18 Añadir subir imágenes al registro <input checked="" type="checkbox"/> TFG-22 Reunión Cliente <p>+ Crear incidencia</p>	
<p>▼ Tablero Sprint 8 2 incidencias</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> TFG-19 Subir Google Play <input checked="" type="checkbox"/> TFG-23 Corrección errores usuarios <p>+ Crear incidencia</p>	

Figura 1.2: Tareas incluidas en cada sprint**Figura 1.3:** Modelo de iteraciones usando agilismo

CAPÍTULO 2

Estado del arte

En la actualidad, la amplia presencia de dispositivos móviles en todos los estratos de la sociedad ha causado que las empresas hayan cambiado el centro de sus desarrollos de aplicaciones de escritorio a aplicaciones móviles y web.

Además, gracias a la gran cantidad de documentación, librerías de código abierto, cursos online y herramientas de programación, se ha facilitado bastante el acceso y aprendizaje de las tecnologías móviles a los desarrolladores.

En el mercado de aplicaciones de Google (*Google Play Store*) existen diversas apps destinadas al control de calidad y auditoría de espacios y lugares. A continuación mencionaremos algunas de ellas y detallaremos sus ventajas e inconvenientes.

2.1 iAuditoria

iAuditoria¹ es una herramienta que permite la realización de auditorías y/o inspecciones a partir de una lista de chequeo o *Check List* previamente establecida. La app funciona sobre un entorno desplegado en la nube que puede ser utilizado en cualquier sistema operativo.

Los objetivos que la app pretende conseguir son similares a los que nos hemos propuesto en GdQ: realizar inspecciones en un establecimiento, obtener resultados fiables y efectivos, de manera rápida, optimizando los recursos y reduciendo el coste en materiales y tiempos de trabajo.

Al contrario que GdQ, iAuditoria es una solución centrada especialmente en procesos de auditoría y incluye diversos modelos como BRC v7, OHSAS 18001 o ISO 9001. Su modelo de negocio consiste en planes de pago que incluyen más funcionalidades a medida que aumentan los precios.

Dos características que se han detectado que esta app no incluye respecto a GdQ son la posibilidad de tomar fotos del lugar o elemento y el uso de GPS y QR para la geolocalización.

2.2 Calidad Cloud

La app Calidad Cloud² sirve para realizar controles de calidad en linea usando listas de chequeo o planificaciones de obra. Permite obtener métricas y comparativas en

¹<https://www.iauditoria.com>

²<https://www.calidadcloud.com/>

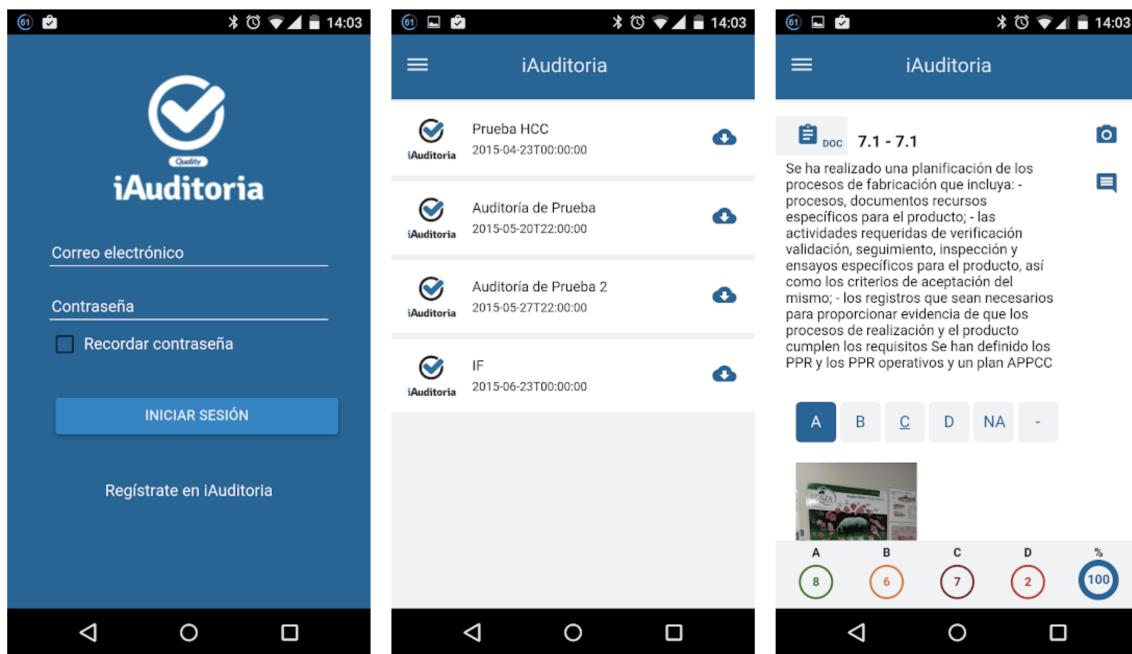


Figura 2.1: Capturas de pantalla durante el uso de la app iAuditoria

tiempo real de lo que sucede en obra, sin tener que esperar que se traspasen los datos a papel, evitando las pérdidas de datos y ampliando los puntos de control. Además se registran fotografías de los sucesos o hallazgos encontrados generando un proceso de mejora continua.

Los administradores de la organización pueden visualizar el contenido generado por los trabajadores usando un portal web que permite generar gráficas y reportes, además de gestionar los usuarios.

Una característica interesante que no se incluye en la primera versión de GdQ es la capacidad de trabajar sin conexión a internet y subir todo el contenido registrado una vez el teléfono disponga de conexión. Además, Calidad Cloud está orientada a la medición en tiempo real de los KPI de la organización y el desempeño de los trabajadores y no tanto a realizar registros sobre localizaciones.

2.3 CTI gCalidad

CTI gCalidad³ es una solución tecnológica que tiene como principal objetivo controlar y asegurar que los servicios que presta cierta empresa cumplan con unos criterios de calidad y eficiencia determinados.

Esta app permite mejorar el control sobre los diferentes servicios que se prestan en los edificios mediante la utilización de códigos QR y etiquetas NFC. Además, implementa un sistema para la gestión de incidencias y órdenes de trabajo y incorpora herramientas para la medición de los criterios de calidad.

Así pues, con CTI gCalidad se dota a la empresa de herramientas que permiten mejorar su capacidad de gestión y establecer estrategias para mejorar los parámetros de eficiencia durante los proyectos.

³<http://www.ctisl.net>

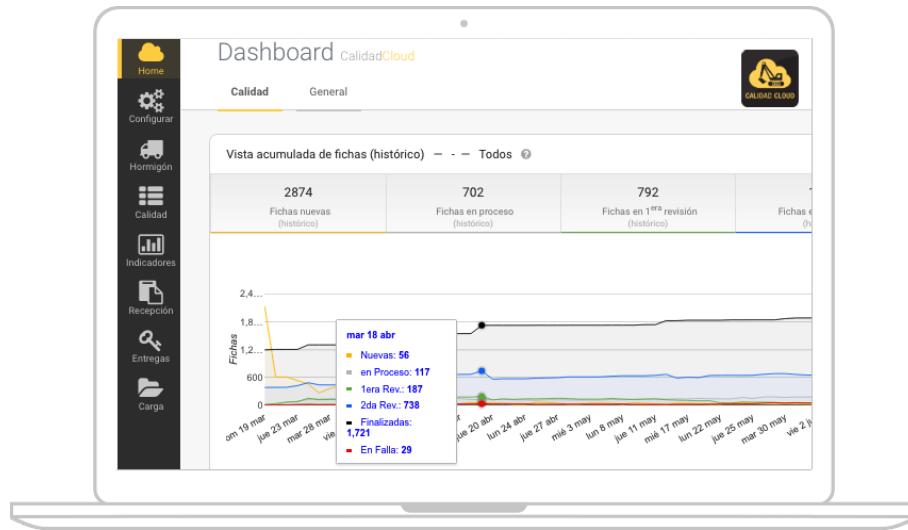


Figura 2.2: Portal web de Calidad Cloud



Powered by CTI

Figura 2.3: Ventanas principales de la app CTI gCalidad

2.4 Crítica al estado del arte

Como hemos visto en el apartado anterior, existen diversas apps destinadas al control de calidad actualmente en el mercado. Cada una de estas apps implementa diferentes funcionalidades dependiendo de que tipo de controles va a realizar y del sector empresarial donde se destina la aplicación.

De las 3 apps que hemos expuesto, la más similar a la que vamos a desarrollar en este trabajo sería CTI gCalidad, ya que iAuditoria está enfocada a la realización de procesos y estándares de auditoría y Calidad Cloud a la medición de rendimientos y KPI.

Al trabajar con códigos QR o etiquetas NFC para la detección del lugar o localización donde se está realizando el control surge la problemática de que tanto el código QR como la etiqueta se pueden duplicar y realizar controles sin estar realmente en la ubicación real.

Por este motivo, en GdQ decidimos implementar el uso de GPS para verificar que la ubicación del terminal se encuentra dentro de un rango cercano a la localización de la cual se está realizando el control. Esta característica no la hemos encontrado en las aplicaciones que se encuentran en el mercado y nos sirve para diferenciarnos del resto de competidores.

Además, otras de las funcionalidades que no hemos encontrado en apps existentes son: la capacidad de poder realizar registros para varias empresas distintas, la existencia de un rol de administrador que puede visualizar registros de los empleados usando la aplicación y la posibilidad de realizar un registro manual sin necesidad de leer el código QR en caso de deterioro o perdida.

También nos pareció interesante incorporar herramientas destinadas a mejorar la comunicación entre operarios y administradores. Con este objetivo desarrollamos la opción de contacto con el administrador de la contrata a través de la app y la posibilidad de que el administrador adjunte ficheros como horarios o instrucciones a cada uno de los operarios.

En resumen, las apps existentes en el mercado nos han permitido obtener ideas y características interesantes que merecían la pena incorporar a nuestra aplicación así como funcionalidades que no hemos encontrado en dichas apps y que veíamos necesarias para dar mas valor a nuestro producto y diferenciarnos de la competencia.

2.5 Solución propuesta

En el punto anterior dedicado a la crítica al estado del arte hemos enumerado funcionalidades y características que pensamos que son clave para la solución que a continuación proponemos. Por lo tanto, en esta sección reunimos todas ellas teniendo en cuenta que ésta será una primera versión de la aplicación y que en un futuro esperamos ampliar y mejorar.

1. **Uso de GPS:** Añadir geolocalización como una medida extra de seguridad para verificar la veracidad de los registros realizados. Además, se podrá comprobar mediante un mapa que la ubicación es correcta así como su precisión.
2. **Multi-empresa:** Posibilidad de estar dado de alta en varias empresas o contratas y poder seleccionar sobre cual de estas se van a realizar los registros y las consultas.
3. **Rol de administrador:** Existencia de un rol de administrador que puede realizar gestiones sobre cada usuario de la contrata.

4. **Registros manuales:** Modo de registros manual sin necesidad de escanear código QR para lugares que por distintas razones no pueden ser provistos de códigos QR impresos.
5. **Ficheros adjuntos:** Posibilidad de adjuntar a cada usuario diferentes ficheros que puedan ser de utilidad para la realización de sus labores.
6. **Contacto directo:** Método de contacto vía email tanto con el servicio técnico de la app como con el administrador de la empresa o contrata.

CAPÍTULO 3

Análisis

A continuación realizaremos un análisis de una parte importante que formarán la arquitectura de este producto de software: la capa de negocio. Consistirá en la explicación de las entidades necesarias involucradas en la aplicación y los casos de uso.

En nuestra propuesta inicial teníamos como idea principal que nuestra solución fuese una app móvil ya que esto permite que los trabajadores o técnicos instalen la aplicación en dispositivos que probablemente ya tengan en su poder y no sea necesario invertir dinero en equipamiento.

Este TFG se centra principalmente en el desarrollo de la app de Android. Priorizamos la app de Android ante la de iOS ya que el primero tiene mucha mayor cuota de mercado en España (figura 3.1), lo cual nos permite una captación de usuarios y empresas que quieran utilizar nuestro sistema mucho mayor que si nos centrásemos en el sistema operativo de Apple.

Además, cuenta con gran de documentación online, ya sea oficial o a través de foros, librerías de código abierto que nos pueden facilitar el trabajo y un entorno de desarrollo de gran calidad y constantemente actualizado llamado Android Studio.

Pensamos que era también importante que la solución contase con dos tipos de usuarios: operario y administrador. Estos dos tipos de usuarios se diferenciarán en el nivel de permisos que se le otorga a cada uno y la complejidad de uso de la apps. El operario utilizará únicamente la app móvil, que será lo más simple y intuitiva posible para que pueda ser utilizada y comprendida rápidamente. Sin embargo, el administrador podrá realizar más acciones relacionadas con la gestión de la contrata y además podrá acceder al panel web y trabajar de manera más cómoda y eficiente. Podemos ver un diagrama relacionando los diferentes tipos de usuarios y plataformas en la figura 3.2.

Respecto al plan inicial donde intentar introducir la app en el mercado, pensamos en centrarnos y darnos a conocer en empresas que trabajen y realicen contratos con el sector público, concretamente contratos de limpieza y auditorías. Como hemos comentado, en la actualidad, las instituciones públicas conceden más puntos a favor en contratas a las empresas que introduzcan elementos para la informatización y control tecnológico del proceso.

3.1 Casos de uso

En esta sección se detallan usando diagramas y tablas los diferentes casos de uso de la aplicación, tanto para el usuario como para el administrador. Al haberse priorizado en esta primera versión que la app solo incluya las acciones más básicas y necesarias que

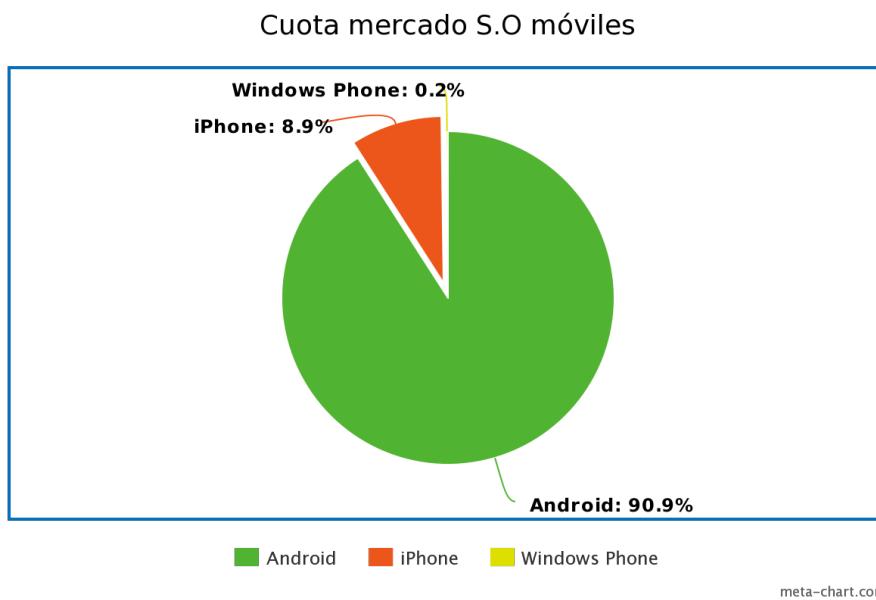


Figura 3.1: Cuota de mercado de sistemas operativos móviles a principios de 2019 en España

los usuarios deben de realizar, observamos que el diagrama no incluye gran cantidad de acciones.

3.1.1. Iniciar sesión

El primer caso de uso consiste en que el usuario o el administrador inicien sesión en la aplicación. Estos verán en pantalla el IMEI del dispositivo y deberán introducir su número de teléfono. Si el administrador ha dado de alta previamente al usuario desde el panel web, este podrá acceder. De lo contrario, mostrará una ventana emergente indicando que el administrador no ha asociado el IMEI con el número de teléfono que el usuario ha introducido.

Descripción	Inicia sesión en la app usando el número de teléfono y el IMEI como credenciales.
Actores	Usuario y administrador.
Precondición	El usuario no ha utilizado la app previamente o nunca ha llegado a iniciar sesión.
Postcondición	El usuario accede correctamente a la aplicación o se le muestra un aviso indicando que no está dado de alta.

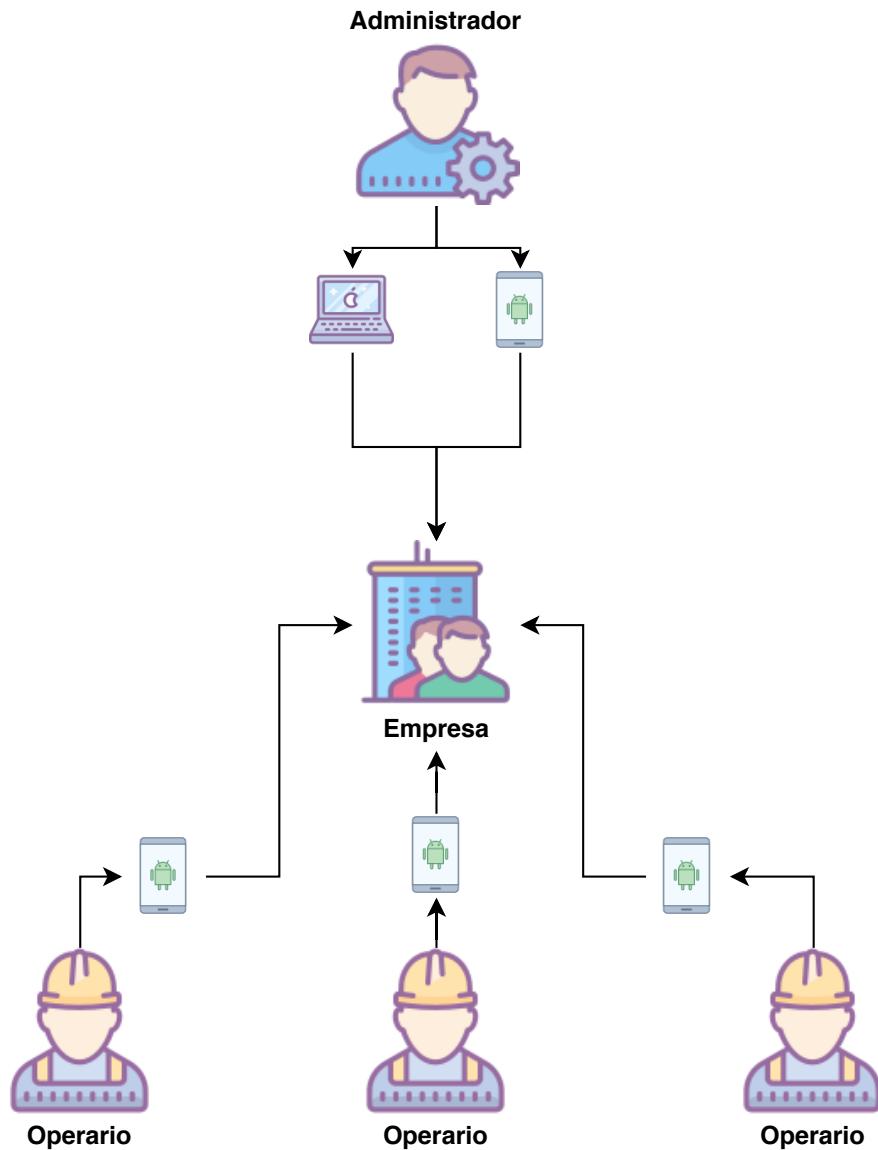


Figura 3.2: Diagrama de tipos de usuario y plataformas de acceso a GdQ

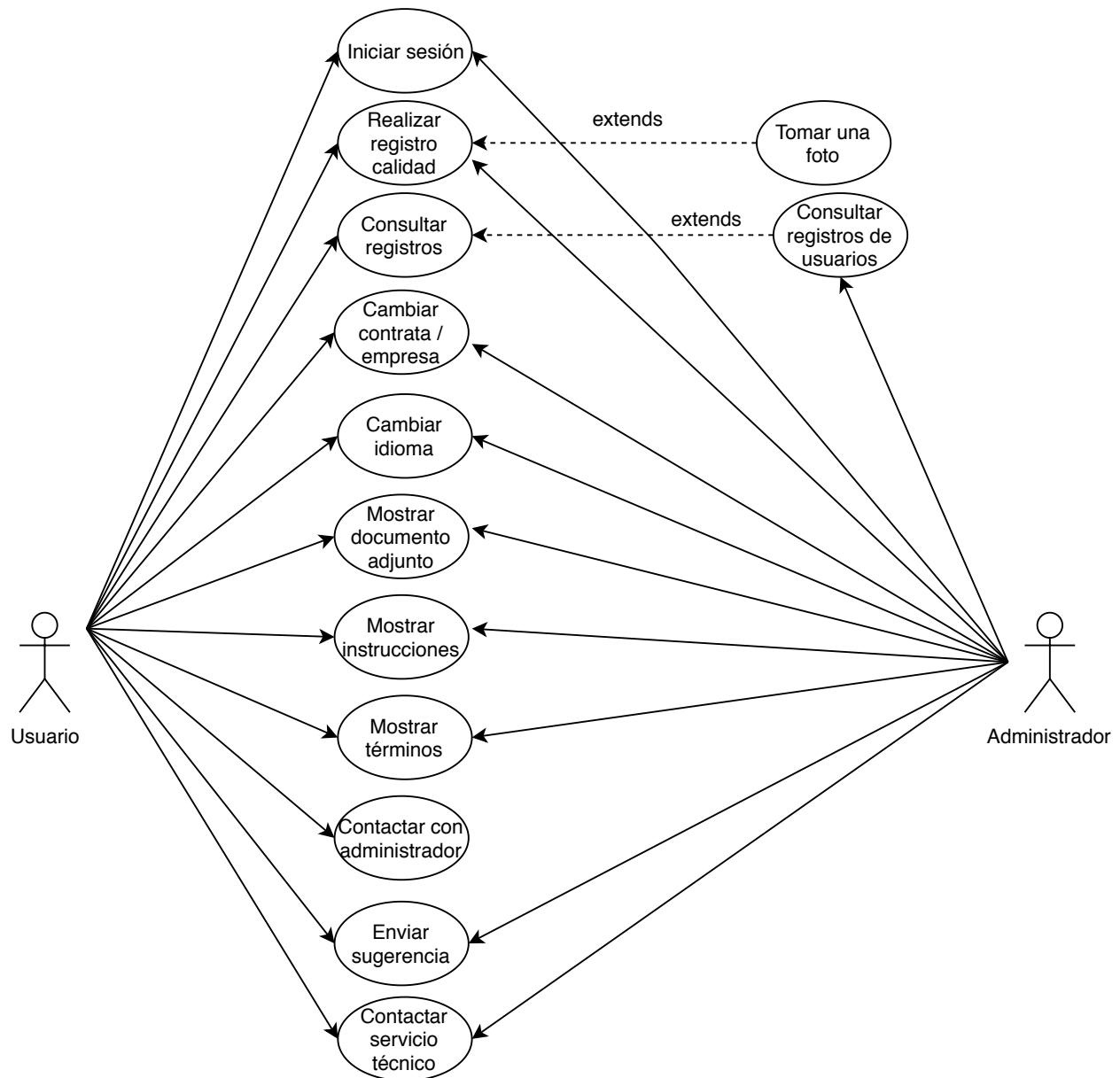


Figura 3.3: Diagramas de casos de uso del usuario y administrador

3.1.2. Realizar registro calidad

En este caso de uso el usuario o el administrador realizará un registro de calidad sobre una localización y un elemento mediante el escaneo de un código QR o introduciéndolo manualmente. Además, podrán indicar el estado en que se encuentra la instalación o el elemento y añadir fotos al registro.

Descripción	Realiza un registro de calidad sobre una localización y un elemento.
Actores	Usuario y administrador.
Precondición	El usuario deberá tener activado la localización por GPS.
Postcondición	Mediante una llamada al servidor se introduce el nuevo registro. Se muestra una ventana emergente informando si el registro se realizó correctamente o no. La app vuelve a la ventana principal.

3.1.3. Consultar registros

El usuario podrá consultar sus registros realizados anteriormente aplicando filtros de rangos de fechas, localizaciones, elementos o estados. El administrador podrá realizar estas mismas acciones pero siendo capaz también de consultar los registros de los empleados incluidos en la empresa o contrata.

Descripción	Consulta registros realizados previamente con la posibilidad de aplicar filtros de búsqueda.
Actores	Usuario y administrador.
Precondición	El usuario deberá tener conexión a Internet.
Postcondición	Se mostrará en una lista los registros que cumplan con las condiciones que el usuario o el administrador ha especificado en el filtro.

3.1.4. Cambiar contrata

El usuario o el administrador que este dado de alta en varias contratas dentro de una misma empresa podrá elegir sobre cual de ellas va a trabajar. Esto condicionará sobre qué contrata se realizarán los registros o las consultas.

Descripción	El operario cambia de contrata sobre la que va a trabajar.
Actores	Usuario y administrador.
Precondición	El usuario necesitará conexión a internet y haber iniciado sesión en la app.
Postcondición	Cualquier registro o consulta que realice el usuario se hará sobre la contrata seleccionada.

3.1.5. Cambiar idioma

En este caso de uso el usuario o el administrador cambiarán el idioma en el que se muestra la aplicación.

Descripción	El usuario cambia el idioma de la aplicación.
Actores	Usuario y administrador.
Precondición	El usuario deberá haber iniciado sesión en la app y acceder al menú de perfil.
Postcondición	La aplicación se muestra en el idioma que el usuario a seleccionado.

3.1.6. Mostrar documento adjunto

En el siguiente caso de uso el usuario o el administrador podrá descargar y previsualizar el documento adjunto que el administrador de la contrata configure para ese usuario. Este documento podrá ser un horario, un documento de normas, etc.

Descripción	El usuario visualizará el documento adjunto especificado por el administrador.
Actores	Usuario y administrador.
Precondición	El usuario deberá tener conexión a internet activa y haber iniciado sesión en la app.
Postcondición	Se descargará el documento adjunto y se mostrará usando la app que sea necesaria según el formato del archivo.

3.1.7. Mostrar instrucciones

Mediante este caso de uso se le mostrará al usuario las instrucciones de uso de la aplicación, así como preguntas y casuísticas frecuentes.

Descripción	El usuario visualizará las instrucciones de uso de la aplicación.
Actores	Usuario y administrador.
Precondición	El usuario deberá tener conexión a internet activa y haber iniciado sesión en la app.
Postcondición	Se abrirá un enlace con el navegador web donde se mostrarán las instrucciones.

3.1.8. Mostrar términos

En el siguiente caso de uso el usuario podrá consultar los términos y condiciones de uso de la app.

Descripción	El usuario visualizará los términos y condiciones de uso de la aplicación.
Actores	Usuario y administrador.
Precondición	El usuario deberá tener conexión a internet activa y haber iniciado sesión en la app.
Postcondición	Se abrirá un enlace con el navegador web donde se mostrarán los términos y condiciones de uso.

3.1.9. Contactar con administrador

El usuario contactará contactará con el administrador de la contrata para cualquier tipo de duda o problema que surja con la gestión de la contrata.

Descripción	El usuario contacta con el administrador de la contrata (su supervisor).
Actores	Usuario.
Precondición	El usuario deberá tener conexión a internet activa y haber iniciado sesión en la app.
Postcondición	Se ejecutará la aplicación de correo electrónico con la dirección de email del administrador ya cumplimentada y el usuario podrá comunicarse con el administrador.

3.1.10. Enviar sugerencia

El usuario envía una sugerencia a los desarrolladores de la app para proponer una mejora o corrección de algún error que ha detectado.

Descripción	Envío de sugerencia a través de un formulario web.
Actores	Usuario y administrador.
Precondición	El usuario deberá tener conexión a internet activa y haber iniciado sesión en la app.
Postcondición	El navegador web mostrará un formulario en la pagina web de la empresa donde el usuario podrá informar mejoras o anomalías detectadas en la app.

3.1.11. Contactar servicio técnico

El usuario contactará con el departamento de soporte de la app para temas técnicos como contratar servicios o modificaciones en sus productos contratados.

Descripción	Contacto directo con el departamento de soporte de la aplicación.
Actores	Usuario y administrador.
Precondición	El usuario deberá tener conexión a internet activa y haber iniciado sesión en la app.
Postcondición	Se ejecutará la aplicación de correo electrónico con la dirección de email del departamento de soporte ya cumplimentada y el usuario podrá abrir un ticket con su petición de soporte.

3.2 Entidades

En la siguiente sección describiremos cada una de las entidades que formarán parte de la aplicación. Como podemos ver, no hay un gran número de entidades ya que nuestro objetivo es minimizar las responsabilidades de la app y que la complejidad resida en el servidor y la base de datos.

La figura 3.4 consiste en un diagrama ER donde podemos ver las diferentes entidades, atributos y relaciones entre ellas.

3.2.1. Operario

La entidad operario almacenará la información relacionada con el operario que utiliza la aplicación, ya sea usuario o administrador. Guardará los siguientes atributos:

1. **imei**: IMEI del teléfono. Al ser único, actúa como clave primaria.
2. **nombre**: Nombre completo del usuario.
3. **esAdmin**: boolean que servirá para distinguir si el operario es administrador o usuario.

Un operario podrá realizar de 0 a N registros de calidad y deberá pertenecer al menos a una contrata.

3.2.2. Contrata

La contrata representa la obra o servicio que realiza una empresa sobre un lugar concreto. Es decir, una empresa puede ofrecer sus servicios a otras en diferentes lugares y cada uno de estos sería una contrata diferente.

1. **id**: ID único que identifica la contrata.
2. **nombre**: Nombre descriptivo de la contrata.

Un contrata tendrá N registros de calidad y N operarios asociados a ella.

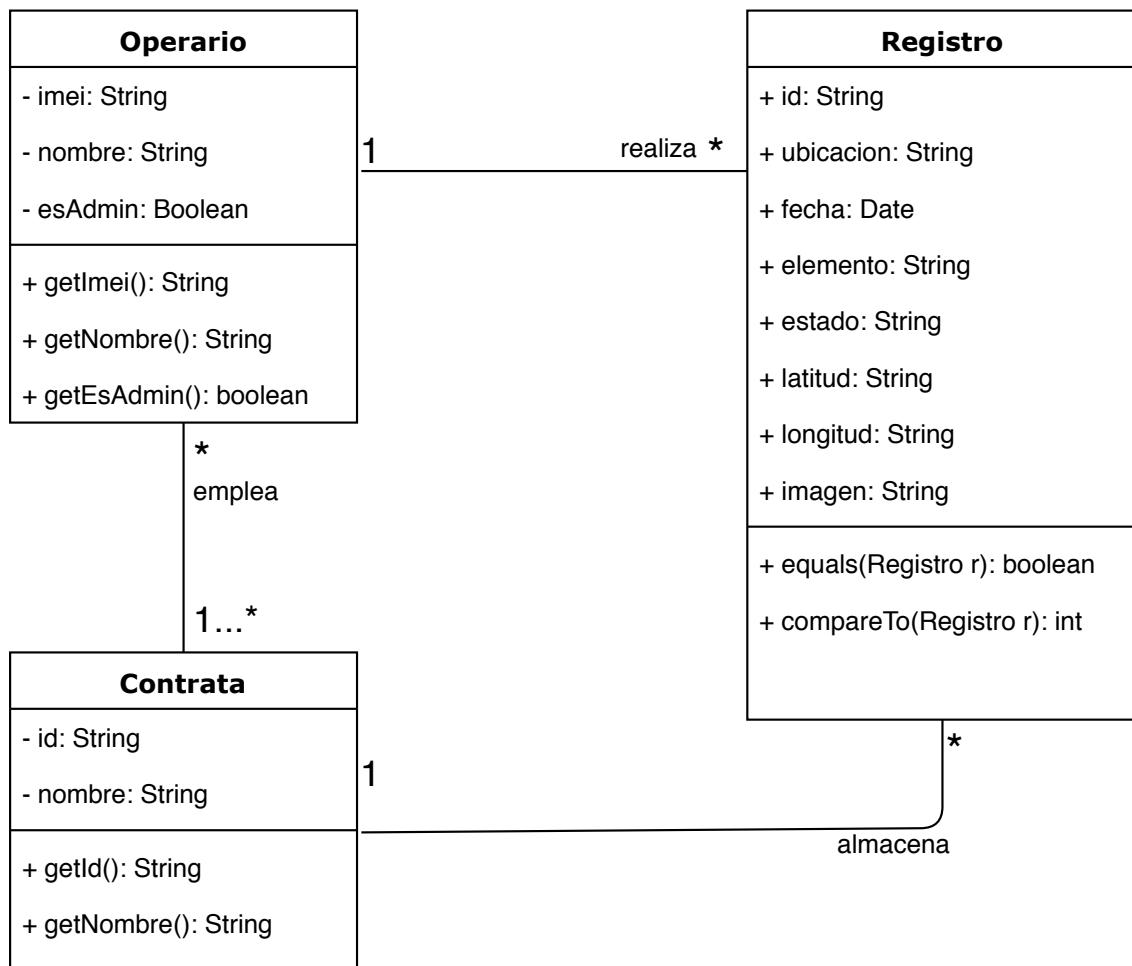


Figura 3.4: Diagrama de clases de las entidades

3.2.3. Registro

El objeto Registro guardará la información de cada uno de los registros de calidad que se realicen.

1. **id:** ID único del registro.
2. **ubicacion:** Nombre de la ubicación del registro.
3. **fecha:** Fecha de realización del registro.
4. **elemento:** Nombre del elemento que se esta inspeccionando.
5. **estado:** Texto que describe el estado del elemento.
6. **latitud:** Latitud de la ubicación del teléfono.
7. **longitud:** Longitud de la ubicación del teléfono.
8. **imagen:** Imagen codificada en Base64.

Cada registro pertenece a un operario y a una contrata.

CAPÍTULO 4

Diseño

4.1 Arquitectura de software

En esta sección detallaremos la arquitectura de software que hemos utilizado para llevar a cabo la aplicación. En concreto, la arquitectura que hemos elegido es MVP (*Modelo Vista Presentador*).

4.1.1. MVP

Los patrones de arquitectura de software permiten mantener un proyecto limpio, escalable, fácil de mantener y de testear. El MVP (*Model View Presenter*) es un patrón derivado del MVC (*Model View Controller*) que permite ampliar la cobertura del testeo casi al 100 % ya que separa toda lógica de la aplicación de la vista. Además, esta modularidad mejora bastante la mantenibilidad del producto si en un futuro deseamos añadir cambios o mejoras.

Estos son los componentes que forman parte del patrón MVP:

1. **Modelo:** Consiste en la interfaz que se encarga de gestionar los datos que la aplicación utilizará. Manejará tanto los accesos como las actualizaciones de nuestros datos de la base de datos. Es independiente de la vista y del presentador.
2. **Presentador:** Trabajará como intermediario entre el modelo y la vista. Pedirá los datos que la vista necesite al modelo y se encargará de formatearlo antes de devolver el resultado de la petición a la vista. Tendrá por lo tanto, acceso a las interfaces de la vista y el modelo.
3. **Vista:** Se define como la interfaz encargada de mostrar los datos en la aplicación y gestionar los diferentes eventos que el usuario realiza en la app. Esta no conoce la definición del modelo y se comunicará con el presentador cada vez que ocurra un evento, siendo éste quien se encargue de la lógica.

Como hemos comentado, a diferencia del MVC del cual este patrón deriva, aquí tenemos la vista y el modelo completamente separados, actuando el presentador como mediador. Esto permite que la vista y el presentador sean perfectamente *testeables* y que en caso de cambios en cualquiera de los 3 componentes, los demás no se ven afectados.

En la actualidad, el patrón MVP es bastante utilizado por los desarrolladores de aplicaciones Android e incluso podemos encontrar otros patrones derivados de éste.

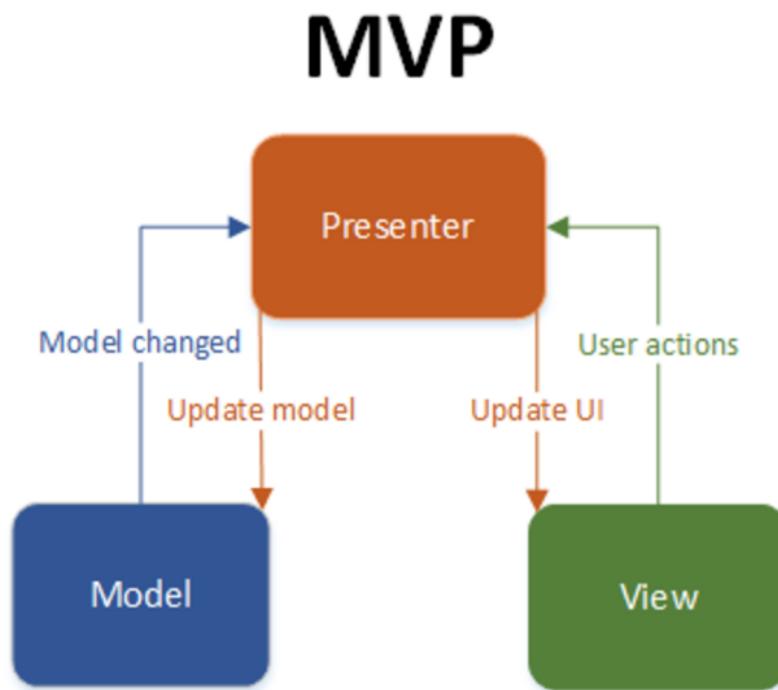


Figura 4.1: Esquema de componentes del patrón MVP

4.2 Diseño de interfaces

En el siguiente apartado dedicado al diseño mostraremos los diferentes diseños que hemos realizado para cada una de las pantallas que forman la aplicación. Han sido rea- lizando utilizando el editor de interfaces de Android Studio. Con este editor, es posible compilar diseños rápidamente arrastrando elementos de IU a un editor de diseño visual en lugar de escribir el XML de diseño manualmente. Además, permite revisar el diseño en diferentes dispositivos y versiones de Android.

Para cada uno de los diseños se describirán las funcionalidades que la ventana es capaz de realizar.

4.2.1. Inicio de sesión o *login*

Funcionalidades de la pantalla:

1. El usuario deberá poder iniciar sesión si previamente el administrador ha dado de alta el IMEI del dispositivo y el número de teléfono en el sistema.

4.2.2. Menú principal

Funcionalidades de la pantalla:

1. El usuario podrá acceder a la ventana de inspección de calidad.
2. El usuario podrá acceder a la ventana de consulta de registros.
3. El usuario será capaz de cambiar de contrato usando una lista desplegable.

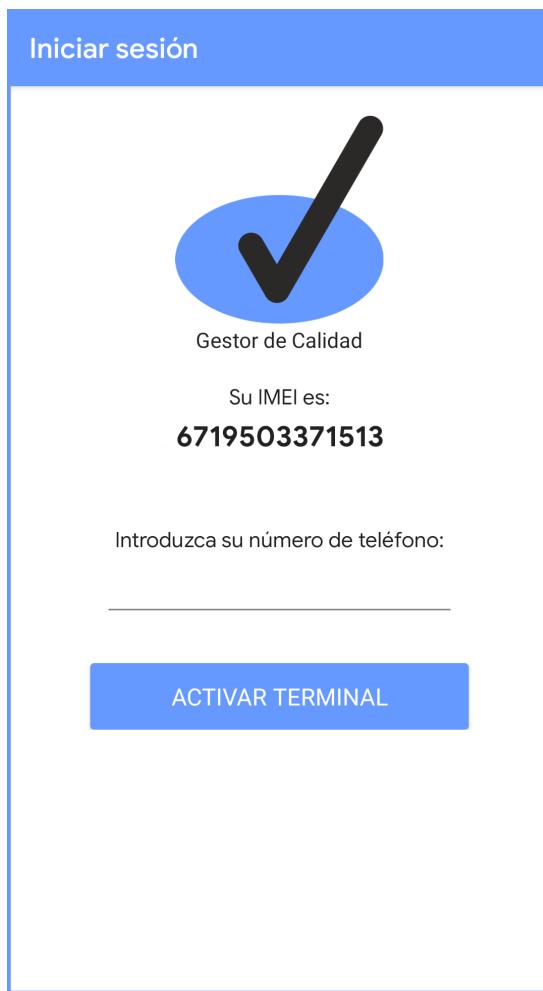


Figura 4.2: Pantalla de inicio de sesión



Figura 4.3: Pantalla principal o *home*

4. El usuario podrá acceder a la ventana de inspección de calidad.

4.2.3. Inspección

Funcionalidades de la pantalla:

1. El usuario podrá realizar un registro escaneando un código QR.
2. El usuario podrá realizar un registro seleccionando la localización en una lista desplegable.
3. El usuario elegirá el elemento que quiere inspeccionar y el estado que quiere reportar.
4. El usuario podrá seleccionar una foto de su galería o tomar una en el momento.
5. El usuario podrá consultar su posición GPS actual.



Figura 4.4: Pantalla de escaneo de QR

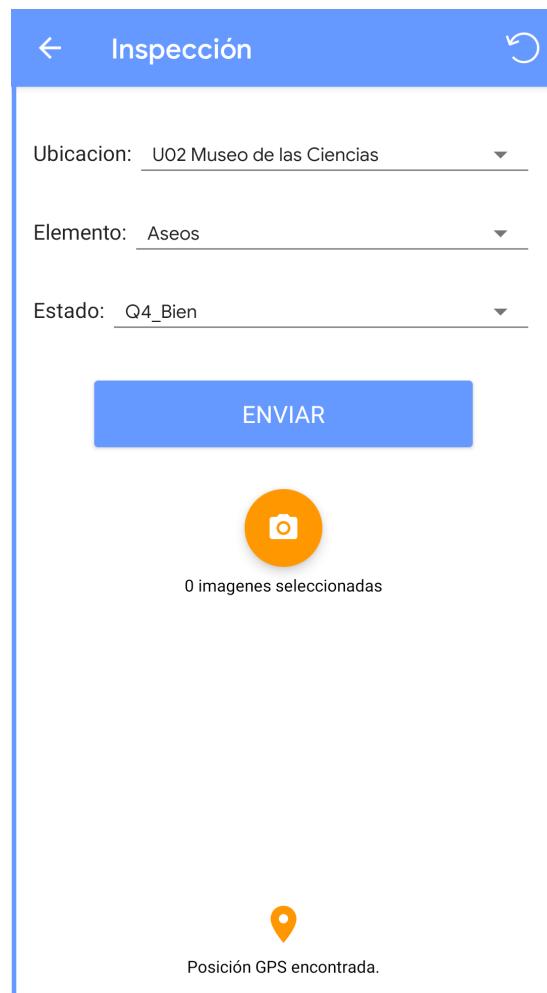


Figura 4.5: Pantalla de registro de inspección

Figura 4.6: Pantalla de consulta de registros**Figura 4.7:** Pantalla de listado de registros

4.2.4. Consulta

Funcionalidades de la pantalla:

1. El usuario especificará el rango de fechas del cual desea obtener los registros.
2. El usuario podrá elegir si filtrar la búsqueda por un objeto, elemento, estado o usuario concreto.
3. El usuario podrá visualizar en una lista los registros que cumplan con el filtro.
4. El usuario podrá restablecer la búsqueda y realizar una nueva.

4.2.5. Perfil de usuario

Funcionalidades de la pantalla:

1. El usuario podrá consultar sus datos personales.
2. El usuario podrá cambiar el idioma de la aplicación.
3. El usuario podrá activar o desactivar el GPS.
4. El usuario podrá mostrar el documento adjunto asignado.

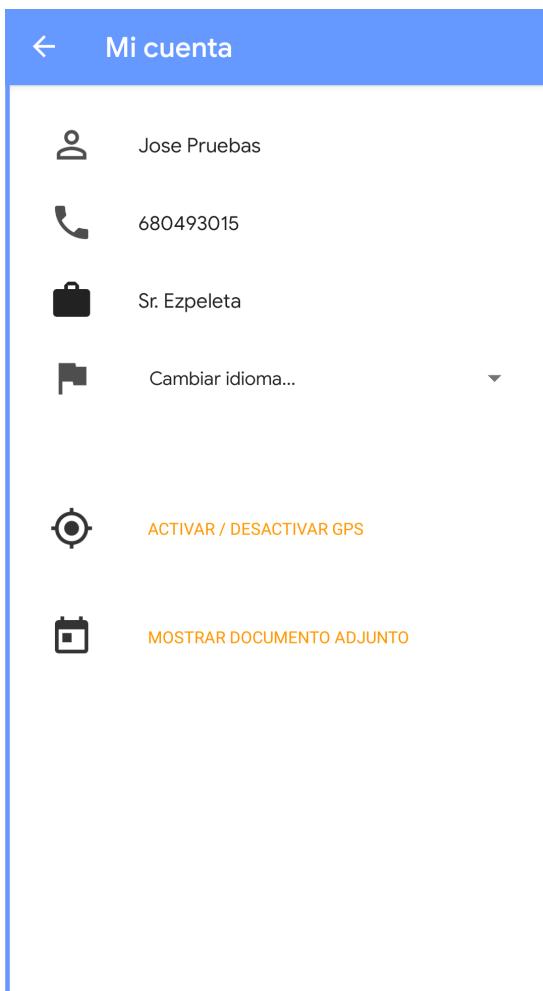


Figura 4.8: Pantalla de perfil de usuario



Figura 4.9: Pantalla de ayuda

4.2.6. Ayuda

Funcionalidades de la pantalla:

1. El usuario podrá consultar la web de instrucciones de uso.
2. El usuario podrá consultar los términos y condiciones de la app.
3. El usuario podrá contactar con el administrador de la contrata.
4. El usuario podrá enviar una queja o sugerencia a los desarrolladores.
5. El usuario podrá contactar con el servicio técnico.

CAPÍTULO 5

Desarrollo e implementación

Llegados a este punto, dedicaremos este capítulo para explicar cómo hemos realizado el desarrollo de los diferentes componentes que forman parte de la solución de software completa. Empezaremos detallando cómo hemos diseñado el servidor, seguidamente hablaremos de cómo desplegamos la API en él y finalmente el punto más importante para este TFG: el desarrollo de la aplicación Android.

5.1 Servidor

A continuación describiremos la plataforma utilizada para desplegar el servidor y los diferentes componentes y software utilizados. Con estos ha sido posible la creación de la base de datos y la implementación de la *API*.

5.1.1. Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web de alquiler de capacidad informática en la nube en forma de máquinas virtuales llamadas *instancias*. Es el producto más destacado de la plataforma de cómputo en la nube Amazon Web Services (AWS).

Está basado en las tecnologías de virtualización, permitiendo elegir entre gran variedad de sistemas operativos y con la capacidad de ser rápidamente escalable según las necesidades. De este modo, los usuarios alquilan máquinas virtuales pagando solo la capacidad utilizada, en lugar del tradicional modelo de compra o alquiler de máquinas con una capacidad concreta.

En las instancias el usuario podrá ejecutar el software o código que deseé, siendo capaz de ajustar la potencia y capacidad de las instancias, así como crear nuevas instancias o eliminarlas cuando deseé. Además, es posible elegir la posición geográfica de estas para reducir latencia y visualizar el uso de recursos mediante el panel de control web.

Los precios de este servicio son bastante menores en comparación al coste que tendría comprar o alquilar una máquina dedicada para GdQ. En nuestro proyecto hemos decidido empezar con una instancia *Micro* bajo demanda (se paga por la capacidad informática utilizada por horas) ubicada en Alemania y corriendo el sistema operativo Linux/GNU. Además, Amazon ofrece las primeras 750 horas de uso de la instancia de forma gratuita.

En las imágenes siguientes detallamos los diferentes pasos para dar de alta una nueva instancia desde el portal de Amazon EC2:

1. Seleccionamos el sistema operativo que queremos utilizar. (Figura 5.1)

2. Seleccionamos el tipo de instancia, de la cual dependerá el numero de CPUs y de memoria RAM. (Figura 5.2)
3. Seleccionamos la cantidad de almacenamiento del que dispondrá la máquina virtual. (Figura 5.3)
4. Generamos y descargamos la clave privada que nos servirá para conectarnos remotamente a la máquina. (Figura 5.4)
5. La máquina virtual que hemos creado a sido añadida a nuestro panel de control de EC2. (Figura 5.5)

Para conectar a nuestra recién creada máquina virtual en Amazon EC2, usaremos el comando ssh incluido en la mayoría de versiones de Linux, Mac o en el caso de Windows, usando un cliente SSH. Para ello, ejecutaremos el siguiente comando:

```
1 ssh -i tfg.pem ubuntu@ec2-18-195-103-59.eu-central-1.compute.amazonaws.com
```

1. *tfg.pem* es el archivo de clave privada que hemos generado desde el panel de control de EC2 para autenticarnos en la maquina virutal.
2. *ubuntu* es el usuario de la maquina al cual vamos a conectarnos.
3. *ec2-18-195-103-59.eu-central-1.compute.amazonaws.com* es el DNS público de la maquina. También es posible usar su IPv6

Una vez realizados estos pasos, ya tenemos acceso y control sobre la instancia que hemos iniciado. A continuación podemos proceder a instalar el software necesario para la creación de la base de datos y de la API.

5.1.2. Ubuntu Server

Ubuntu es una *distribución* de GNU/Linux gratuita y de código abierto basada en la arquitectura *Debian*. En la actualidad es capaz de funcionar tanto en ordenadores de sobremesa, servidores y teléfonos móviles y en arquitecturas Intel, AMD y ARM.

Se trata de un sistema operativo actualizado y estable para el usuario estándar, enfocado en la facilidad de uso y de rápida instalación. Al igual que otras distribuciones se compone de múltiples paquetes de aplicaciones normalmente distribuidos bajo una licencia libre o de código abierto.

La versión de servidor es similar a la de escritorio pero no incluye interfaz gráfica y tiene cambios en el sistema de instalación. Como hemos visto en el apartado anterior, para la implementación del servidor instalaremos Ubuntu Server 18.04 LTS en la máquina virtual eligiéndolo en la lista de sistemas operativos.

5.1.3. LAMP

LAMP es un modelo de software para servicios web formado a partir del acrónimo de los siguientes componentes *open-source*:

1. *Linux*. Sistema operativo.
2. *Apache*. Servidor HTTP.

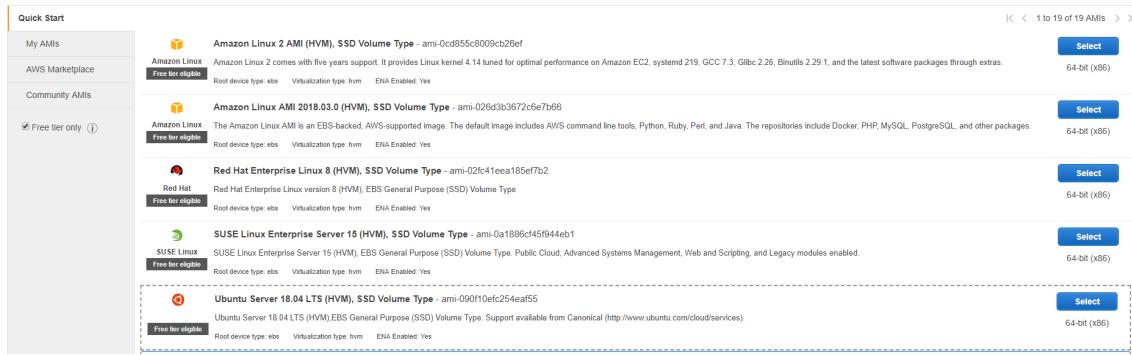


Figura 5.1: Configuración de instancia EC2: Sistema operativo

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)								
Family	Type	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance	IPv6 Support	
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes

Figura 5.2: Configuración de instancia EC2: Hardware

Step 4: Add Storage
Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encryption
Root	/dev/sda1	snap-0f6d0839bcb2e22c4	8	General Purpose SSD (gp2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

Figura 5.3: Configuración de instancia EC2: Almacenamiento

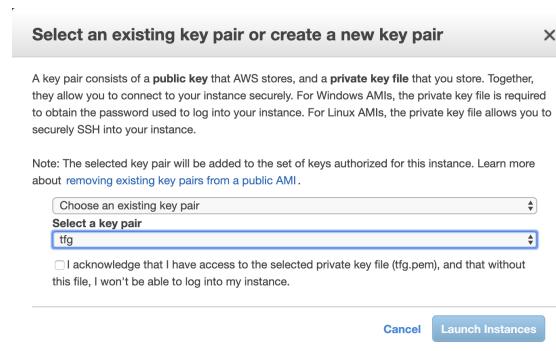


Figura 5.4: Configuración de instancia EC2: Generación clave privada

Launch Instance	Connect	Actions
search: i-00a25cf3bdea01395 <input type="button" value="Add filter"/>		
Name	Instance ID	Instance Type
i-00a25cf3bdea01395	t2.micro	eu-central-1b

Figura 5.5: Panel de control de EC2



Figura 5.6: Grupos de seguridad de la instancia de EC2

3. *MySQL*. Gestor de base de datos relacional.
4. *PHP*. Lenguaje de programación.

La combinación de estas tecnologías es usada principalmente para definir la infraestructura de un servidor, dirigido tanto a páginas web dinámicas como aplicaciones web. Una de las ventajas de LAMP es que sus componentes son ampliamente intercambiables y no existe la obligación de ceñirse a los mencionados anteriormente.

A pesar de que el origen de estos programas de código abierto no fue específicamente diseñado para trabajar entre sí, la combinación se popularizó debido a ser componentes gratuitos y a su facilidad de instalación (tanto en Linux como en Windows.)

A continuación explicamos como instalar y configurar cada uno de los componentes LAMP en Ubuntu:

Apache

Ejecutaremos los siguientes comandos en la máquina virtual:

```
1 $ sudo apt update
2 $ sudo apt install apache2
```

El primer comando actualizará la lista de repositorios de software del sistema y el segundo instalará el servidor web Apache.

Una vez termine la instalación de Apache, desde el panel de control de AWS EC2 accedemos al grupo de seguridad asignado a nuestra instancia y editamos las reglas de conexiones entrantes, de forma que permitamos el acceso desde cualquier IP al puerto 80, como vemos en la imagen 5.6

Una vez realizado esto, podemos acceder la IP de la instancia y si todo ha funcionado correctamente, veremos el html predeterminado de Apache, ubicado por defecto en */var/www/html*: figura 5.7

PHP

Proseguiremos con la instalación de LAMP instalando el lenguaje de programación PHP, en concreto la versión 7.2. Para ello, ejecutaremos el siguiente comando:

```
1 $ sudo apt-get install php libapache2-mod-php
```

Una vez realizado esto, haremos una prueba similar a la de Apache para comprobar que la instalación fue correcta y que podemos ejecutar código PHP en nuestro servidor.

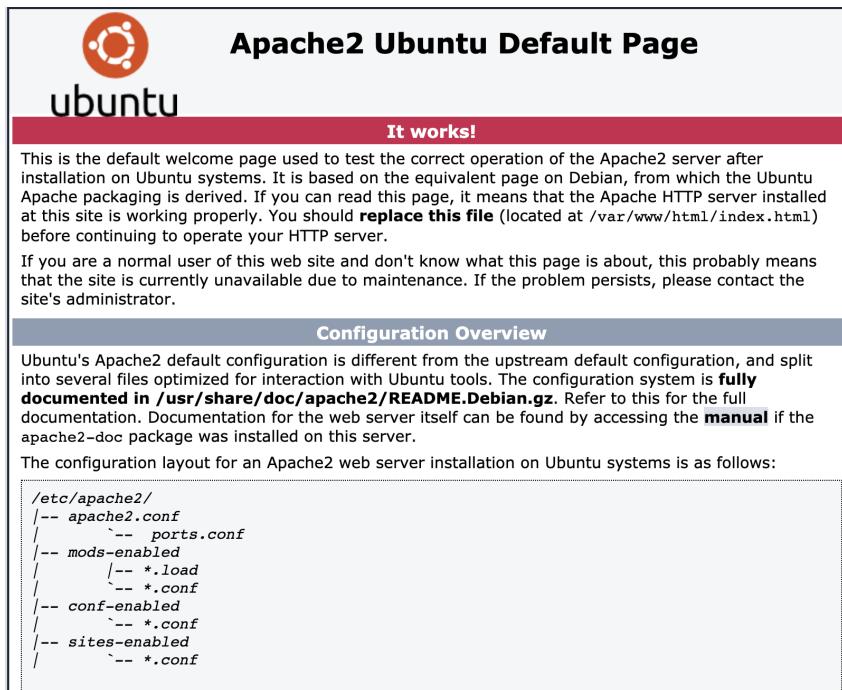


Figura 5.7: Página HTML predeterminada del servidor web Apache

PHP Version 7.2.19-0ubuntu0.18.04.1	
System	Linux ip-172-31-23-104 4.15.0-1032-aws #34-Ubuntu SMP Thu Jan 17 15:18:09 UTC 2019 x86_64
Build Date	Jun 4 2019 14:48:12
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-filinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718

Figura 5.8: Página de información de PHP

Depositaremos en la ruta por defecto de Apache (`/var/www/html`) un fichero .php con el siguiente código:

```
1 <?php phpinfo(); ?>
```

y accederemos a él a través del navegador. Si vemos una pantalla similar a [5.8](#), podemos proseguir con la instalación de LAMP

MySQL

A continuación, instalaremos el gestor de base de datos relacional MySQL. Desde la consola, ejecutaremos:

```
1 $ sudo apt install mysql-server
```

Una vez termine el proceso de instalación, ejecutaremos los siguientes comandos para crear un usuario y darle permisos de administrador.

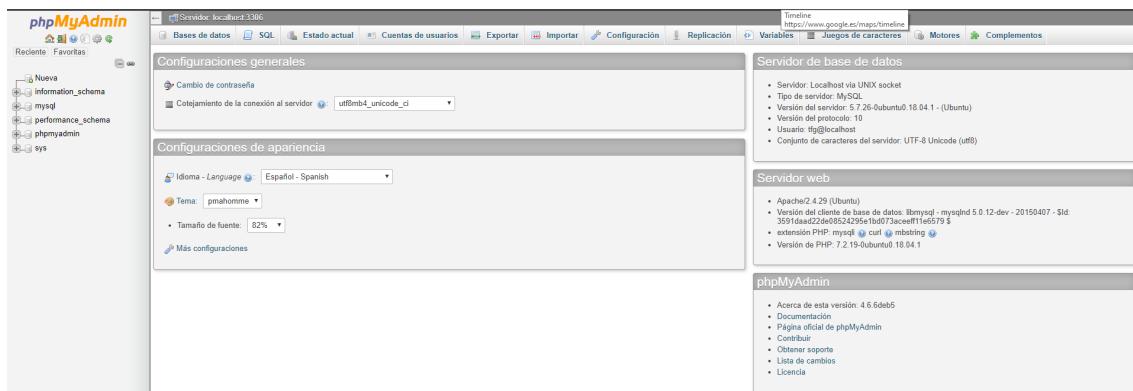


Figura 5.9: Menú principal de phpMyAdmin

```

1 $  sudo mysql
2 > CREATE USER 'nombre_usuario'@'localhost' IDENTIFIED BY 'contraseña';
3 > GRANT ALL PRIVILEGES ON * . * TO 'nombre_usuario'@'localhost';
4 > FLUSH PRIVILEGES;

```

De esta forma, en nuestra maquina virtual, hemos creado un usuario llamado *tfg*. Además, para facilitar la gestión de nuestra base de datos, instalaremos la herramienta phpMyAdmin.

Con este software, accesible a través del navegador, podemos realizar gran cantidad de gestiones en una base de datos MySQL, tales como: crear y eliminar bases de datos, crear y eliminar tablas, añadir y alterar campos, ejecutar sentencias SQL o administrar privilegios.

Su instalación es sencilla y debemos seguir los siguientes pasos:

1. Ejecutar en la consola: *sudo apt-get install phpmyadmin php-mbstring php-gettext*.
2. En la elección de nuestro servidor, seleccionar *apache2*.
3. Permitir la instalación de *dbconfig-common*.
4. Usar el socket de Unix para la conexión con la base de datos.
5. Establecer un nombre para la base de datos que phpMyAdmin usará de forma interna.
6. Indicar el usuario y la contraseña de MySQL que hemos creado anteriormente.

Si hemos ejecutado este proceso correctamente, accediendo a la IP de nuestra instancia y añadiendo */phpmyadmin*, podremos ver en nuestro navegador la web de login de phpMyAdmin. Una vez logueados, accederemos a la herramienta (figura 5.9).

5.2 API REST

El objetivo principal del servidor que acabamos de desarrollar es que en él se pueda alojar nuestra base de datos y desplegar en el una API.

Una API (*Application Programming Interface*) es un conjunto de reglas, funciones y métodos que las aplicaciones utilizan para comunicarse entre ellas a modo de capa de abstracción. Existen APIs que sirven para comunicarse con el sistema operativo, con bases de datos o con protocolos de comunicaciones.

Una de las claves del funcionamiento de las API es la facilidad de integración. Estas herramientas tienen que resultar simples de integrar en software para que las comunicaciones se desarrollen correctamente.

Además, al ser REST (*Representational State Transfer*), tiene las siguientes características:

1. **Protocolo cliente/servidor:** cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, el cliente no necesita conocer los detalles de implementación del servidor y el servidor no sabe cómo son usados los datos que envía al cliente.
2. **Conjunto de operaciones:** Existen operaciones que nos permiten manipular los recursos. Las más importantes son las siguientes: GET para consultar, POST para crear, PUT para editar y DELETE para eliminar.
3. **Interfaz uniforme:** Tiene definida una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de la misma forma. De esta manera, cada recurso del servicio REST debe tener una única dirección o *URI*.
4. **Uso de hipermedios:** Permite al usuario navegar por los distintos recursos de una API REST a través de enlaces HTML.

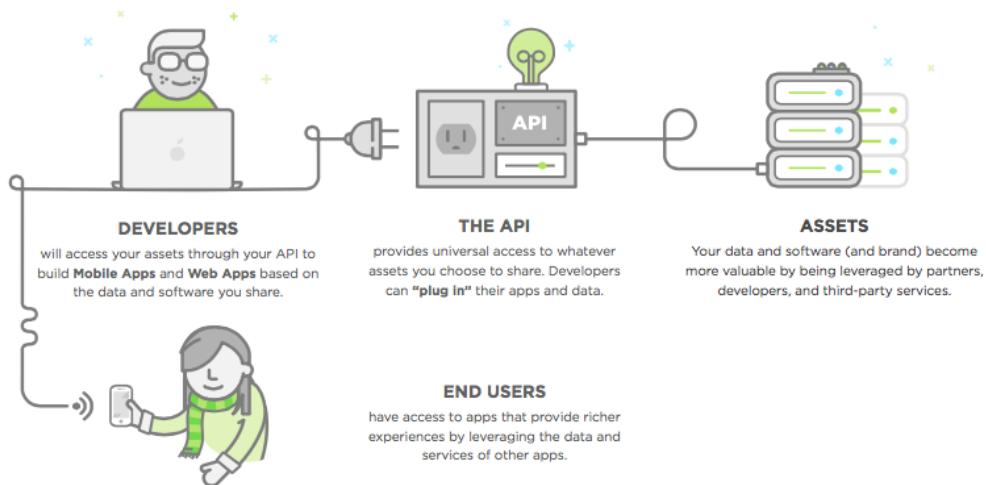


Figura 5.10: Diagrama básico de flujo al usar una API

Debido a que el objetivo de este TFG no es el desarrollo de la API si no de la aplicación de Android, utilizaremos una API desarrollada en PHP, de código libre disponible en GitHub¹. Si en el futuro vemos necesario el despliegue de una API más avanzada y compleja con otro sistema más robusto sería posible su desarrollo, pero para conseguir una primera versión funcional de la app y poder mostrarla a los clientes, con este software nos sirve.

Algunas de sus ventajas son:

1. Fácil de desplegar (solo es un fichero).

¹<https://github.com/mevdschee/php-crud-api>

2. Soporta objetos y arrays JSON.
3. Compatible con sistema de permisos sobre bases de datos, tablas y columnas.
4. Permite paginar, filtrar y ordenar los resultados.
5. Incorpora un módulo de autenticación mediante tokens.
6. Genera documentación acorde a nuestra base de datos.

Consiste en un único fichero PHP que tras configurar los datos de acceso a nuestra base de datos MySQL, nos proporciona acceso CRUD a cada una de las tablas y vistas de la BBDD. Por lo tanto, abriendo el fichero PHP y configurando las siguientes variables, seremos capaces de conectar con la base de datos:

```

1  'dbengine'=>'MySQL',
2  'hostname'=>'127.0.0.1',
3  'username'=>'gdq',
4  'password'=>'XXX',
5  'database'=>'gdq',
6  'port'=>'3306',
7  'charset'=>'utf8'
```

Si la configuración ha sido correcta y accedemos al fichero *api.php* desde nuestro navegador, nos devolverá la documentación de la API para nuestra base de datos en formato JSON. Formateándola con algún interprete JSON como *swagger.io* podemos verla de forma visual incluso testear con llamadas HTTP y obtener una respuesta.

Elementos	
GET	/Elementos List
POST	/Elementos Create
GET	/Elementos/{id} Read
PUT	/Elementos/{id} Update
DELETE	/Elementos/{id} Delete
PATCH	/Elementos/{id} Increment

Empresas	
GET	/Empresas List
POST	/Empresas Create
GET	/Empresas/{id} Read
PUT	/Empresas/{id} Update
DELETE	/Empresas/{id} Delete
PATCH	/Empresas/{id} Increment

Figura 5.11: Documentación generada de nuestra API

Así pues, a modo de ejemplo, si queremos obtener el registro de calidad con id '4', podemos realizar la siguiente petición GET:

```
1 18.185.132.114 / api .php/ Registro /4
```

La respuesta será un objeto JSON como este:

```
{  
    "Id": 4,  
    "Ubicacion": "Polideportivo Municipal",  
    "Fecha": "2019-06-02 00:08:32",  
    "Elemento": "Aseos",  
    "Estado": "Correcto",  
    "Operario": "867195033715137",  
    "lat": "39.5672354",  
    "lon": "-0.621983",  
    "IdImagen": ""  
}
```

Una petición un poco más compleja sería filtrar los registros de calidad de un operario entre dos fechas:

```
1 18.185.132.114 / api .php/ RegistrosContratas ? filter %5B %5D=IdContrata %2Ceq %2C10&  
filter %5B %5D=Operario %2Ceq %2C865485040553626&filter %5B %5D=Fecha %2Cbt %2C2019  
-07-24 %2C2019-07-26
```

Esta sería la respuesta:

```
{  
    [  
        {  
            "Id": 125,  
            "Ubicacion": "Oficina",  
            "Fecha": "2019-07-25 21:33:32",  
            "Elemento": "Cristales",  
            "Estado": "Regular",  
            "Operario": "865485040553626",  
            "lat": "39.5672354",  
            "lon": "-0.621983",  
            "IdImagen": ""  
        },  
        {  
            "Id": 126,  
            "Ubicacion": "Aseos",  
            "Fecha": "2019-07-24 21:33:32",  
            "Elemento": "Puerta",  
            "Estado": "Bien",  
            "Operario": "865485040553626",  
            "lat": "39.5672354",  
            "lon": "-0.621983",  
            "IdImagen": ""  
        }  
    ]  
}
```

Con la configuración mostrada y los tests para comprobar que funciona correctamente, ya tendríamos la API lista para integrarla en la app y así poder insertar y consultar datos en la base de datos.

5.3 Aplicación de Android

A continuación, para finalizar con la sección del desarrollo, proseguiremos con la aplicación móvil. Presentaremos el sistema operativo Android, el lenguaje de programación Java y más tecnologías utilizadas.

5.3.1. Android

En los últimos años los teléfonos móviles han experimentado una gran evolución, desde los primeros terminales, grandes y pesados, pensados sólo para hablar por teléfono en cualquier parte, a los últimos modelos pequeños y táctiles. En la actualidad, los móviles han perdido la idea inicial para lo que fueron creados ya que con ellos es posible realizar casi cualquier acción que antes hacíamos con un ordenador.

Android nace como un sistema operativo y plataforma software basado en Linux dirigido a teléfonos móviles, principalmente con pantalla táctil. Fue desarrollado inicialmente por Android Inc. y comprado por Google en el año 2005. A partir de este año, con el respaldo de Google, fue cuando empezó su rápido crecimiento y popularidad que se mantiene hoy en día con 2.000 millones de usuarios, siendo así el sistema operativo móvil más utilizado.

Al ser de código abierto (excepto librerías y aplicaciones propietarias de Google), existen gran cantidad de desarrolladores que han creado grandes comunidades donde compartir conocimiento, documentación y ayuda.

Algunas de sus características más destacadas son:

1. Núcleo basado en el Kernel de Linux.
2. Posibilidad de programar en Java y Kotlin.
3. Tienda de aplicaciones gratuitas y de pago llamada [Google Play Store](#).
4. Multitarea.
5. Adaptable a diferentes pantallas y resoluciones.
6. Ampliamente establecido en la mayoría de países.

Al desarrollar una app en Android hay que tener en cuenta un concepto llamado *fragmentación*. Esto consiste en que en el mercado conviven diferentes versiones del sistema operativo y que son los fabricantes de teléfonos los encargados de proporcionar actualizaciones. Por este motivo, antes del desarrollo hay que tomar la decisión de a partir de qué versión de Android va a ser compatible nuestra aplicación y realizar el desarrollo teniendo esto en cuenta, ya que el número de versiones con las que queremos ser compatibles influirá en nuestro código.

Nuestra aplicación será compatible a partir de la versión de la API 17 y la versión *target* a la que está orientada la app es la API 27.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

Figura 5.12: Versiones de Android y porcentaje de instalaciones en junio de 2019

El **IDE** que utilizaremos será **Android Studio**. Es la plataforma oficial para desarrollo en Android y está basado en el software **IntelliJ IDEA** de **JetBrains**. Incorpora herramientas enfocadas a facilitar la programación en esta plataforma como: soporte para **Gradle**, refactorización, creación de interfaces con arrastrar y soltar, plantillas de código comunes y soporte para emulador de diferentes versiones de Android.

Android Studio soporta la programación en Java y en Kotlin (desde 2017).

5.3.2. Java

Java es un lenguaje de programación creado en 1995 por Sun Microsystems. Nació con el objetivo de ser un lenguaje de programación de estructura sencilla que pudiera ser ejecutado en diversos sistemas operativos. Se basa en programación orientada a objetos, es concurrente y permite ejecutar un mismo programa en diversos sistemas operativos y ejecutar el código en sistemas remotos de manera segura.

Java empezó siendo el único lenguaje aceptado para programar aplicaciones Android aunque en la actualidad Google está apostando por **Kotlin**, un lenguaje de programación de tipado estático que corre sobre la máquina virtual de Java.

El entorno de ejecución sobre el que corre el código Java en Android se llama **ART** (**Android Runtime**) y desde la versión 5.0 sustituye a **Dalvik**. ART realiza la traducción del **bytecode** de la aplicación en instrucciones nativas que luego son ejecutadas por el entorno ejecución del dispositivo.

A diferencia de Dalvik, ART introduce el uso de *ahead-of-time* (AOT), que crea un archivo de compilación después a la instalación de la aplicación, evitando así que la app se compile continuamente cada vez que se inicia. En la figura 5.14, ART y Dalvik representarían el cuadrado amarillo.

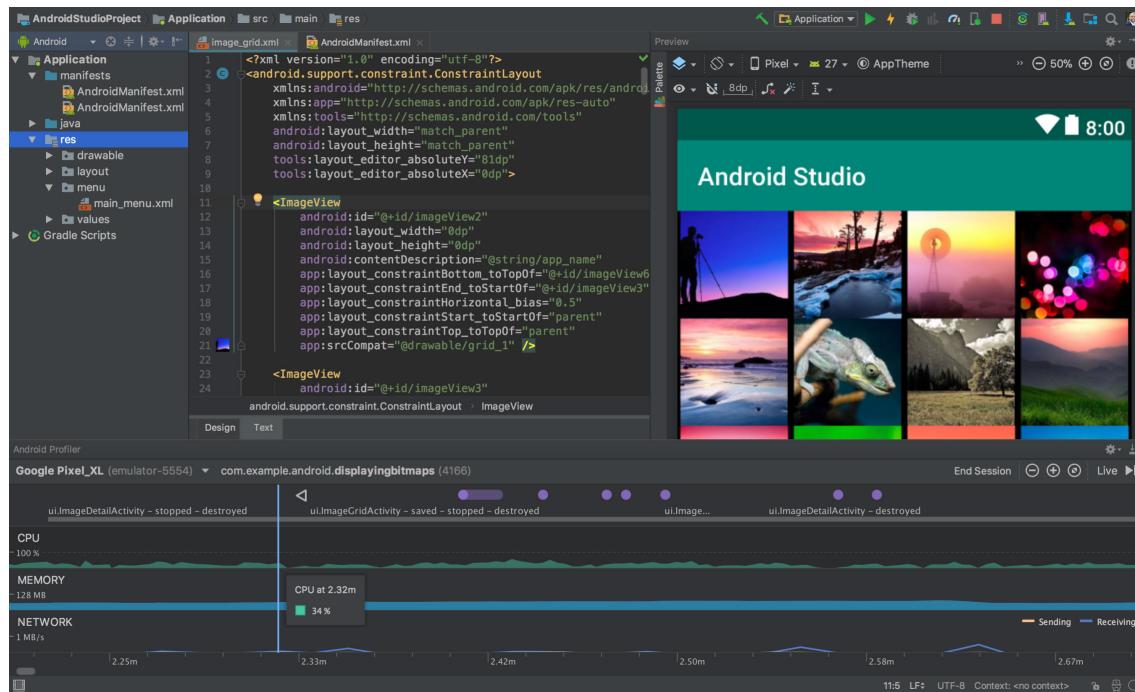


Figura 5.13: El IDE Android Studio.

5.3.3. Tecnologías utilizadas

El desarrollo de una app de Android puede realizarse utilizando solo las librerías incluidas en el sistema, pero es bastante común la inclusión de librerías externas desarrolladas por Google y otras compañías que nos pueden facilitar bastante nuestro trabajo. Además, estas librerías nos ayudan a cumplir un objetivo clave en el desarrollo de software, que consiste en reutilizar componentes ya creados y no “reinventar la rueda”.

Dexter

Dexter es una librería de Android que simplifica el proceso de petición de permisos al usuario en tiempo de ejecución.

A partir de la versión 6.0 de Android (Marshmallow) se añade la funcionalidad de que el usuario tiene la opción de permitir o rechazar cada permiso sobre el teléfono que utiliza una app en vez de aceptarlos todos cuando se instala. De ésta forma, el usuario tiene más control sobre las aplicaciones y sobre el acceso que estas tienen a su teléfono, pero requiere de más trabajo a los desarrolladores.

A continuación mostramos un ejemplo para ilustrar la facilidad de pedir el permiso de la cámara:

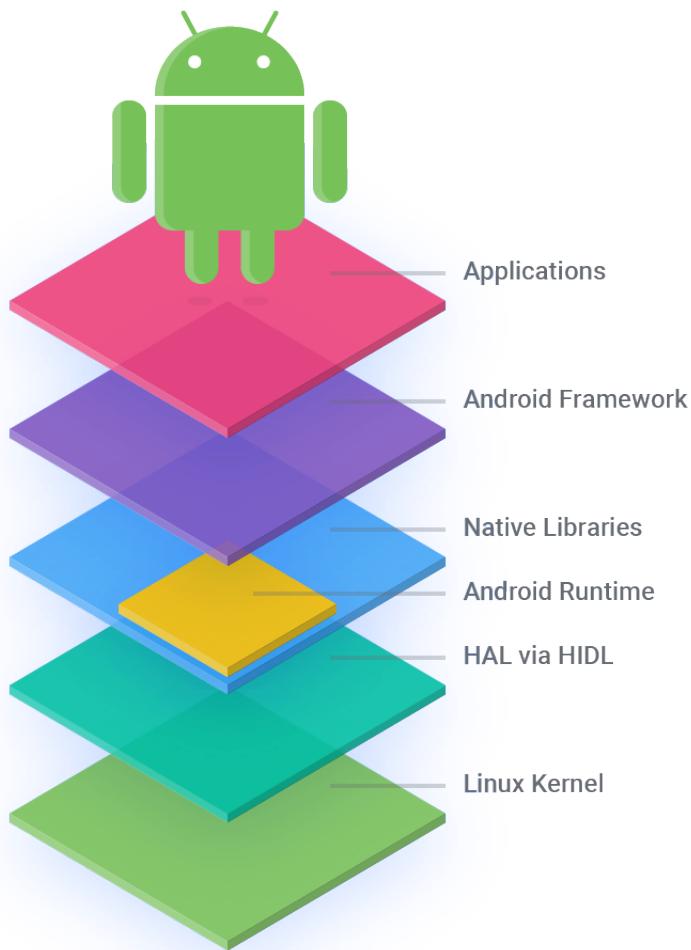


Figura 5.14: Diagrama de capas de software o *stack* que componen Android.

```
Dexter.withActivity(this)
    .withPermission(Manifest.permission.CAMERA)
    .withListener(new PermissionListener() {
        @Override public void
            → onPermissionGranted(PermissionGrantedResponse response)
            → /* Acciones a realizar si se acepta el permiso */
        @Override public void
            → onPermissionDenied(PermissionDeniedResponse response)
            → /* Acciones a realizar si se deniega el permiso */
    }).check();
```

Esta disponible en su repositorio de GitHub: <https://github.com/Karumi/Dexter>

QReader

Librería de Android que utiliza la API Mobile Vision de Google para permitir la lectura de códigos QR.

La librería tiene como objetivo la simplicidad y facilidad de uso. Esto lo consigue eliminando la mayoría del código repetitivo para trabajar con la lectura del códigos QR y

también proporciona una API fácil y simple para recuperar rápidamente la información almacenada dentro del código QR.

Disponible en: <https://github.com/nisrulz/qreader>

Butter Knife

Butter Knife es una librería que nos facilita la tarea de relacionar los elementos de las vistas, *listeners* o diferentes recursos con el código de nuestras aplicaciones Android. Por lo tanto, conseguiremos simplificar el código y nos ahorrará mucho tiempo a la hora de realizar nuestros proyectos.

En el siguiente fragmento de código podemos ver cómo métodos repetitivos típicos de una app de Android como `findViewById()`:

```
TextView textView1 = (TextView) findViewById(R.id.text_view_1);
```

lo transformamos en:

```
@Bind(R.id.tex_view_1)
TextView textView1;
```

Esta disponible en: <https://jakewharton.github.io/butterknife/>

Volley

Volley es una librería desarrollada por Google para optimizar el envío de peticiones HTTP desde las aplicaciones Android.

Funciona como una interfaz de alto nivel, liberando al programador de la administración de hilos, evitando la creación de código repetitivo para manejar tareas asíncronas por cada petición o incluso para parsear los datos que se reciben desde los servidores.

Esto hace que el trabajo de peticiones a servidores sea más fácil y rápido para el programador.

Algunas de sus características más destacadas son:

- Procesamiento concurrente de peticiones.
- Asignación de prioridad a las peticiones.
- Cancelación de peticiones, evitando la presentación de resultados no deseados en el hilo principal.
- Gestión automática de trabajos en segundo plano, dejando de lado la implementación manual de hilos.
- Implementación de caché en disco y memoria.
- Herramientas para *debug* y rastreo.

A modo de ejemplo, mostramos un fragmento de código obtenido de la documentación de Volley para realizar una petición GET de forma sencilla:

```
// Inicializamos el objeto RequestQueue.  
RequestQueue queue = Volley.newRequestQueue(this);  
String url ="http://www.upv.es/";  
  
// Realizamos la petición a la URL que hemos especificado..  
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,  
        new Response.Listener<String>() {  
    @Override  
    public void onResponse(String response) {  
        // Mostramos en el log los primeros 500 caracteres de la respuesta.  
        Log.i("TFG", "Respuesta: "+ response.substring(0,500));  
    }  
, new Response.ErrorListener() {  
    @Override  
    public void onErrorResponse(VolleyError error) {  
        // Mostramos log de error  
        Log.i("TFG", "Hubo un error");  
    }  
});  
  
// Añadimos la petición a la cola.  
queue.add(stringRequest);
```

5.3.4. Estructura del proyecto Android

En esta sección comentaremos brevemente la estructura de carpetas que forman parte del proyecto de la app en Android Studio.

- **app:** Paquete raíz donde se encuentra el proyecto.
 - **manifests/AndroidManifest.xml:** Fichero esencial que está presente en todas las aplicaciones Android. Describe la naturaleza de la aplicación y cada uno de sus componentes y contiene información como el nombre y el icono de la app. Incluye también cada una de las actividades y permisos y librerías necesarias para ejecutar la aplicación.
 - **java:** Contiene el código fuente de al app así como los tests.
 - **base:** Guardaremos aquí la clase base para las *Activity*. Está actuará como superclase de la cual heredaran las demás para ahorrar código y ganar en mantenibilidad.
 - **model:** Contendrá las clases que actúan como modelos de datos: contrata, operario y registro.
 - **utils:** Estas clases contendrán código que ayudará a realizar funciones en otras clases y que es muy probable que sea reutilizado varias veces.
 - **res:** Contiene recursos de aplicación, como layouts de interfaces, imágenes, iconos, strings de las interfaces en diferentes idiomas, estilos y colores.
- **Gradle scripts:** Este paquete almacena todos los scripts relacionados con Gradle, una herramienta que permite la automatización de compilación de código y que nos ayuda bastante con la gestión de dependencias.

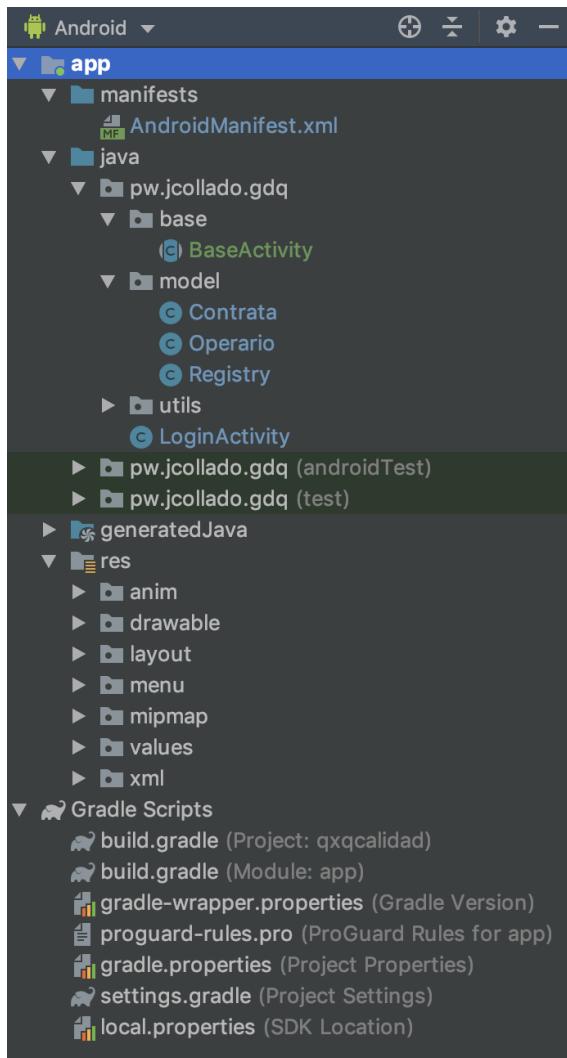


Figura 5.15: Estructura del proyecto GdQ en Android Studio.

5.3.5. Desarrollo de funcionalidad: Enviar email.

A continuación mostraremos de forma sencilla cómo hemos desarrollado una de las funcionalidades de nuestra aplicación, en concreto la de enviar un email al administrador de la contrata. Como hemos comentado en secciones anteriores, el desarrollo lo realizamos en torno al patrón de arquitectura de software *Model-View-Presenter*.

En primer lugar, la Vista la encontraremos en la Activity. Una Activity es un componente de la aplicación que contiene una pantalla con la que los usuarios pueden interactuar para realizar una acción. Por lo tanto, cada una de las pantallas o vistas de la app corresponderá con una Activity.

NuestraActivity consistirá en el código que vemos en la figura 5.17. Como ya hemos explicado en la sección 4.1, la única función de la vista será la gestionar cada una de las interacciones del usuario llamando al Presentador y mostrar en pantalla los datos que recibe de éste.

Vamos a explicar a continuación cómo quedaría el flujo desde que el usuario pulsa el botón para enviar el mail hasta que se le abre en su teléfono la aplicación gestora de emails.

En primer lugar, podemos observar que la Activity implementa la interfaz HelpView (figura 5.16). Esta interfaz incluye son los 3 métodos que la Vista deberá implementar y que serán accesibles desde el Presentador.

Seguidamente, declaramos 2 variables, una de tipo HelpActivityPresenter que será la referencia a nuestro Presentador, al que le pasamos la instancia de la Activity y otra de tipo ProgressBar, que la utilizaremos como barra de progreso para los procesos que dependan de respuestas que no son inmediatas, como llamadas a la API REST.

```
public interface HelpView{  
  
    void sendMailAdminContrata(Intent emailIntent);  
    void showProgressBar();  
    void hideProgressBar();  
  
}
```

Figura 5.16: Interfaz que implementa la Vista

Estas dos variables las inicializamos más tarde dentro del método onCreate(). Este método es ejecutado por el sistema cuando se crea la Activity.

El siguiente método que comentaremos es onClickSendMailAdmin(). Podemos ver que al método le precede la anotación @OnClick. La utilización de esta anotación es proporcionada por la librería Butter Knife y es equivalente a crear un OnClickListener() sobre el botón R.id.adminContactButton, pero mejoramos en legibilidad del código.

Este método que como hemos comentado se ejecuta al hacer click sobre el botón correspondiente en la interfaz, llama al método sendMailToAdmin() que está ubicado en el Presentador. Este es el que gestionara toda la lógica relacionada con la acción que queremos realizar.

Proseguiremos explicando el código del Presentador, en la figura 5.18. Podemos observar que declaramos y inicializamos dos variables, una que hace referencia a la Vista y otra de tipo Operario que corresponde al Modelo. Esto es debido a que el presentador actuará de intermediario entre ambos, obteniendo la información necesaria del modelo y devolviéndosela a la Vista.

A continuación encontramos el método sendMailToAdmin(), el cual hemos visto que es llamado por la Vista cuando hacemos click en el botón. Este método ejecutara en la vista el método showProgressBar() para mostrar la barra de carga, obtendrá el email del administrador de la contrata con operario.getMailAdminContrata() y creará un Intent que intentará abrir la app de email instalada en el teléfono y tendrá el email de administrador ya llenado en el *Para*.

Dicho Intent es devuelto a la Activity llamando al método view.sendMailAdminContrata, ya que al ser la Activity quien tiene el contexto de la aplicación, es quien puede lanzar el Intent con el método startActivity

Para finalizar, comentaremos la clase Operario (figura 5.19), que corresponde al Modelo dentro del patrón MVP. Podemos ver que la clase contiene 2 constructores, uno con 2 parámetros que utilizamos cuando iniciamos la app y otro sin ningún parámetro. A la variable mailAdminContrata le damos valor cuando el usuario abre la ventana de su perfil mediante una llamada al *webservice*.

5.3.6. Desarrollo de funcionalidad: Lectura de código QR.

Ahora nos centraremos en el desarrollo que hemos realizado para la lectura de códigos QR pero centrándonos en la funcionalidad en si y no tanto en el patrón arquitectónico.

Como ya hemos explicado, para obtener la información que contiene un QR utilizamos la librería QReader (sección 5.3.3)

Para empezar, en nuestra interfaz XML añadiremos un elemento de tipo SurfaceView. Aquí es donde irá la imagen de nuestra cámara. Podemos dimensionarla con el tamaño que necesitemos.

```
<SurfaceView
    android:id="@+id/camera_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_above="@+id/skipQRbt"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
```

Asociamos la recién creada SurfaceView de nuestra interfaz a una variable en nuestro código y definimos el objeto de tipo QReader. Lo inicializamos y le pasamos la variable de tipo SurfaceView y un Listener que se activará cuando sea posible leer un QR valido. En este caso, cuando leemos satisfactoriamente el código, introducimos su contenido en una etiqueta de texto de la interfaz.

Especificamos también que vamos a usar la cámara trasera, con el autofocus activado y el tamaño que tendrá será el mismo que la SurfaceView.

```
@BindView(R.id.camera_view)
SurfaceView surfaceView;

private QREader qrEader;

//Resto de código

public void startQRReader() {

    qrEader = new QREader.Builder(this, surfaceView, new
        ↪ QRDataListener() {
            @Override
            public void onDetected(final String data) {
                locationTX.post(new Runnable() {
                    @Override
                    public void run() {
                        ↪ locationTX.setText(Functions.getQRShowFormat(data));
                }
            }
            qrFound();
        }
    });
}
```

```
        });
    }
}) .facing(QREader.BACK_CAM)
    .enableAutofocus(true)
    .height(surfaceView.getHeight())
    .width(surfaceView.getWidth())
    .build();

}
```

Finalmente, en el método `onResume()`, método que se ejecuta dentro del ciclo de vida de una Activity en Android y que se lanzará después del `onStart()` o del `onPause()`, indicamos al objeto QReader que inicie la lectura de QR.

```
@Override
protected void onResume() {
    super.onResume();
    qrEader.initAndStart(surfaceView);
}
```

```
public class HelpActivity extends AppCompatActivity implements HelpView {

    private HelpActivityPresenter presenter;
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.bind(this);

        presenter = new HelpActivityPresenter(this);

        initProgressBar();
        initActionBar();
    }

    @OnClick(R.id.adminContactButton)
    public void onClickSendMailAdmin(){
        presenter.sendMailToAdmin();
    }

    @Override
    public void sendMailAdminContrata(Intent emailIntent) {
        startActivity(emailIntent);
    }

    @Override
    public void showProgressBar() {
        progressBar.setVisibility(View.VISIBLE);
    }

    @Override
    public void hideProgressBar() {
        progressBar.setVisibility(View.INVISIBLE);
    }

    private void initActionBar(){
        //Inicializamos la ActionBar
    }

    private void initProgressBar() {
        //Inicializamos la ProgressBar
    }
}
```

Figura 5.17: Vista o Activity

```
public class HelpActivityPresenter {  
  
    private Operario operario;  
    private HelpView view;  
  
    public HelpActivityPresenter(HelpView view) {  
        this.operario = new Operario();  
        this.view = view;  
    }  
  
    public void sendMailToAdmin(){  
        view.showProgressBar();  
  
        String email = operario.getMailAdminContrata();  
        view.sendMailAdminContrata(createIntentSendEmail(email));  
  
        view.hideProgressBar();  
    }  
  
    public static Intent createIntentSendEmail(String to) {  
  
        Intent emailIntent = new Intent(Intent.ACTION_SEND);  
        emailIntent.setData(Uri.parse("mailto:"));  
        emailIntent.setType("text/plain");  
  
        emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]{to});  
  
        return emailIntent;  
    }  
}
```

Figura 5.18: Presentador

```
public class Operario {  
    private String imei;  
    private String nombre;  
    private String mailAdminContrata;  
  
    public Operario(){  
    }  
  
    public Operario(String imei,String nombre){  
        this.imei = imei;  
        this.nombre = nombre;  
    }  
  
    public String getMailAdminContrata() {  
        return mailAdminContrata;  
    }  
  
    public void setMailAdminContrata(String mailAdminContrata) {  
        this.mailAdminContrata = mailAdminContrata;  
    }  
  
    //Resto de código (getters, setters, llamadas API, etc)  
}
```

Figura 5.19: Modelo de la entidad Operario

CAPÍTULO 6

Resultado final y conclusiones

A modo de conclusión de esta memoria, recuperaremos los objetivos que propusimos en la sección de objetivos (1.2) y veremos hasta qué punto han sido alcanzados o si pueden ser mejorados tomando como referencia para ello el resultado final de la aplicación.

Es necesario destacar la importancia que ha tenido para el desarrollo de este TFG y para cualquier desarrollo de software actual la aplicación de metodologías de trabajo ágiles y la elección de una arquitectura de software adecuada a nuestras necesidades.

Actualmente, debido a la naturaleza tan cambiante y dinámica de los proyectos de software, casi la totalidad de equipos de desarrollo han reemplazado las metodologías antiguas por metodologías ágiles como Scrum o Kanban.

6.1 Cumplimiento de los objetivos

A continuación, comprobaremos si los objetivos propuestos al inicio de la memoria han sido alcanzados satisfactoriamente y si es posible realizar alguna mejora de cara a futuras actualizaciones.

- Permitir que los usuarios se identifiquen en el sistema donde previamente han sido dados de alta por el administrador.

Como vemos en la figura 6.1, para que un operario pueda acceder desde la app, el administrador debe registrar primero desde el panel web el IMEI de su teléfono y asociarlo a una o varias contratas. De lo contrario, la app muestra un error indicando que el usuario no está dado de alta en el sistema (figura 6.2).

Mantenimiento de Terminales			
Menú Principal.		Menú de Mantenimiento	
Teléfono	Nombre	IMEI	Perfil
600600600	Demo-GdP	3547301300862	Operario

Figura 6.1: Panel web para gestión de terminales.

Una posible mejora sería que la app permitiese añadir nuevos operarios y gestionar a qué contratas están asignados. Actualmente esto es posible solo desde el panel web.

GdQ

No hemos encontrado ningun usuario registrado con este IMEI y número de teléfono. Contacta con el administrador del servicio para darte de alta en el sistema.

[CONTINUAR](#)

Figura 6.2: Pop-up de error en login.

- Realizar registros de calidad escaneando el *código QR* o seleccionando la localización de una lista.

Este objetivo lo hemos cumplido. Podemos verlo en el resultado final de la app en la figura 6.3. También podemos comprobar como el registro se añade a la base de datos correctamente.



Figura 6.3: Realizar registro desde la app.

- Consultar registros realizados anteriormente aplicando filtros de búsqueda.
- Este objetivo también ha sido implementando, como podemos ver en la figura 6.4, aunque en futuros trabajos pensamos añadir más tipos de filtros, como poder realizar consultas en varias contratas al mismo tiempo.



Figura 6.4: Consulta de registros realizados.

- Seleccionar sobre qué contrata se va a realizar el registro, pudiendo el operario o usuario estar dado de alta en varias al mismo tiempo.

El usuario puede elegir la contrata utilizando un menú emergente que muestra las contratas donde el operario está dado de alta (figura 6.5).

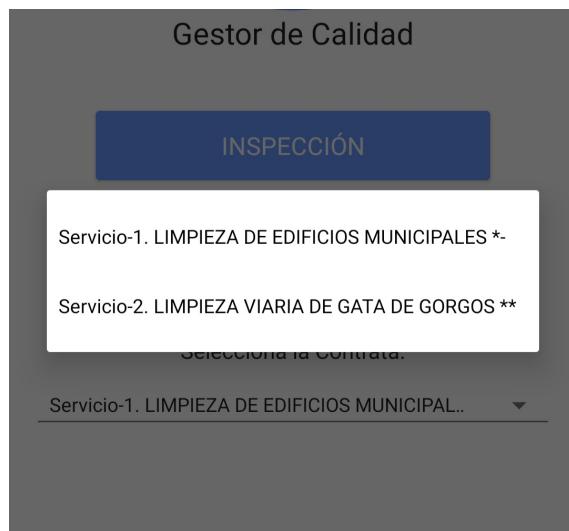


Figura 6.5: Menú seleccionable para elegir contrata.

- Contactar con el administrador de la contrata y mostrar el horario laboral del empleado.

Como ha sido detallado en la sección 5.3.5, esta funcionalidad ha sido desarrollada. Una posible mejora sería sustituir el método de contacto, que ahora es email, por un formulario propio dentro de la app y que el mensaje pueda ser visto por el administrador desde el panel web o la app.

En conclusión, podemos afirmar que los objetivos que nos propusimos antes de iniciar el desarrollo han sido implementados satisfactoriamente. Asimismo, hemos aplicado metodologías ágiles y patrones de diseño y de arquitectura de software consiguiendo un muy buen resultado durante el proceso.

6.2 Relación del trabajo desarrollado con los estudios cursados

Los estudios cursados durante el grado de Ingeniería Informática han tenido una gran importancia para el desarrollo de este proyecto de software, en él se puede ver la implicación de muchas de las asignaturas cursadas.

El grado me ha formado a un nivel básico en la gran mayoría de áreas de la informática, y ha profundizado en algunas de ellas, más concretamente en los campos relacionados con la rama que cursé, la de Ingeniería de Software.

Además, este trabajo me ha servido para adentrarme en tecnologías que no conocía o que había visto de forma superficial, como Android o MySQL.

De forma general, las asignaturas que me han proporcionado conceptos que he podido aplicar en este trabajo son:

- **Programación:** Este proyecto habría sido imposible de llevar a cabo sin los conocimientos sobre programación que aprendí. La programación es un campo muy amplio, y aunque en la universidad no he visto Android específicamente, si que aprendí Java y conceptos como clases y objetos, herencia o las estructuras de datos han sido esenciales.

- **Bases de datos:** Los conocimientos adquiridos durante la carrera sobre bases de datos han sido bastante útiles para la creación de las tablas y sus relaciones a partir del modelo visual que creamos en la sección 3.2, así como para realizar consultas SQL.
- **Desarrollo de software:** Los conocimientos que aprendí sobre el desarrollo de software han sido de gran importancia al diseñar la arquitectura del sistema, los componentes y sus interacciones. Es esencial esquematizar y planificar cómo se va a afrontar el problema antes de empezar a implementar la solución. Además, he podido aplicar varios de los patrones de diseño de software que vimos durante las clases.
- **Redes:** Ha sido de gran ayuda el conocimiento sobre redes aprendido en el grado, ya que durante este TFG he tenido que implementar un *Webservice* y consumir nuestra *API* desde el cliente móvil.

Para finalizar, aparte de todos estos aspectos que acabo de mencionar, cabe destacar la importancia que tiene el hecho de trabajar la resolución de problemas que van surgiendo en el transcurso del desarrollo. Durante el grado, es común y normal que los alumnos se enfrenten a problemas de manera independiente, así pues, cuando fuera de la universidad tenemos que enfrentarnos a un problema real, tenemos los conocimientos y la habilidad necesaria para intentar resolverlo y llegar a conclusiones y resultados por uno mismo.

Glosario

[A](#) | [B](#) | [D](#) | [G](#) | [I](#) | [M](#) | [N](#) | [Q](#)

A

Android

Sistema operativo para teléfonos móviles principalmente táctiles desarrollado por Google. Esta basado en una versión modificada del núcleo Linux y otros software de código libre. En la actualidad también cuenta con código propietario de Google.

[1](#)

API

Una interfaz de programación de aplicaciones, conocida por sus siglas en inglés, *Application Programming Interface*, hace referencia a un conjunto de procesos, funciones y métodos que ofrece una determinada biblioteca de programación a modo de capa de abstracción para que sea empleada por otro *software*. [1](#)

B

bytecode

El bytecode es el código o lenguaje intermedio entre el lenguaje máquina y el código fuente. Durante la etapa de ejecución, estos bytecodes son interpretados y ejecutados instrucción a instrucción por una máquina virtual, en el caso de Java, la JVM.

[39](#)

D

distribución

Una distribución Linux (llamada coloquialmente distro) es una colección de software basada en el núcleo Linux que incluye determinados paquetes de software para satisfacer las necesidades de un grupo específico de usuarios. Así pues, existen distribuciones dirigidas a diferentes ámbitos, como el empresarial, doméstico o para servidores. Suelen estar compuestas mayoritariamente de software libre, aunque a menudo incorporan aplicaciones o controladores propietarios. [30](#)

G

Google Play Store

Google Play Store (anteriormente Android Market) es una plataforma de distribución digital de aplicaciones (apps) mantenida por Google y destinada a los dispositivos con sistema operativo Android. Cuenta con más de 3.5 millones de aplicaciones y también se pueden encontrar libros, películas o música. [3](#), [7](#), [38](#)

GPS

El GPS, de las siglas en inglés *Global Positioning System*, es un sistema que permite determinar la posición sobre la Tierra de cualquier objeto con una precisión de pocos metros o incluso centímetros. Para esto, se sirve de 2 a 4 satélites y utiliza trilateración.¹

I**IDE**

Un IDE o Entorno de Desarrollo Integrado consiste en un software informático que proporciona diferentes servicios integrales para facilitarle al programador el desarrollo de software. Suelen incorporar un editor de código fuente, herramientas de construcción automáticas, un depurador, auto-completado inteligente de código y un compilador.³⁹

M**MySQL**

MySQL es un sistema de gestión de base de datos relacional de código abierto, basado en el lenguaje de consulta estructurado SQL. Es una de las bases de datos más populares que existen y es usada en webs como Wikipedia, Twitter o YouTube.

¹

N**NFC**

NFC (*Near field communication*) consiste en una tecnología inalámbrica de alta frecuencia pero de bajo radio de acción (entre 10 y 15 cm) que permite el intercambio de datos entre dispositivos. Su funcionamiento se basa en la creación de un campo electromagnético y soporta dos modos de funcionamiento: modo activo y pasivo.⁸

Q**QR**

Un código QR (del inglés *Quick Response*) es matriz de puntos o un código de barras bidimensional que almacena información codificada. Fue creado por la compañía japonesa Denso-Wave en 1994 liberando la patente para permitir el libre uso. La intención de los creadores fue que el código pudiese ser leído a alta velocidad, de ahí sus siglas en inglés "rápida respuesta". Los códigos QR son muy comunes en Asia, donde son el código bidimensional más popular.¹

Bibliografía

- [1] David Griffiths, Dawn Griffiths. *Head First Android Development*. O'Reilly Media; 2015.
- [2] Las enormes cifras de Google: 2.000 millones de usuarios Android y más. Consultado el 20/02/2019 en <https://elandroidelibre.elespanol.com/2017/05/cifras-de-google-2000-millones-usuarios-android.html>
- [3] Manifiesto por el Desarrollo Ágil de Software. Consultado el 03/03/2019 en <https://agilemanifesto.org/>

