Grado en Ingeniería Informática

Trabajo Final de Grado

Aplicación móvil para reconocer imágenes (Android)

Autor:
Javier Agut Barreda

Supervisor:
David Benlloch García
Tutor académico:
Raúl Montoliu Colás

Fecha de lectura: 24 de julio de 2014
Curso académico 2013/2014

Agradecimientos

Durante la elaboración de este proyecto han participado, directa o indirectamente, una serie de personas a las que quiero dar las gracias.

En primer lugar, al tutor de este proyecto, Raúl Montoliu, por su implicación y disponibilidad durante el desarrollo del proyecto.

En segundo lugar, a David Benlloch y Joan Fabregat, integrantes de la empresa *Rubycon Information Technologies*, por su ayuda en todo momento, sus consejos y, naturalmente, por invitarnos a café todos los días.

Por supuesto, agradecer también a Sergi Estellés, por su participación en las partes conjuntas del proyecto, ya que sin él no habría sido posible este proyecto.

Por último, agradecer a mi familia y amigos su apoyo en los momentos difíciles, sobre todo todo a mis amigos y a mi novia, que me entendieron y apoyaron cuando no podía ir a la playa con ellos.

Resumen

En este documento se presenta el resultado del Trabajo Final de Grado y estancia en prácticas consistente en una aplicación para el sistema operativo móvil Android, capaz de reconocer imágenes de un conjunto acotado, desarrollada en la empresa *Rubycon Information Technologies*.

La funcionalidad ofrecida por este proyecto pasa por una aplicación que permite realizar fotografías y mostrar los resultados, un sistema que permite reconocer las imágenes y una base de datos que contenga estas junto a su información.

La aplicación es capaz de enviar la imagen al módulo de reconocimiento, alojado en un servidor, y mostrar la información que devuelve este, ya sea en forma de video, audio, web o galería de imágenes, todo esto sin salir de la aplicación.

Además, permite guardar los resultados obtenidos para consultarlos sin tener que volver a realizar el proceso de reconocimiento.

Palabras clave

Aplicación, Android, Reconocimiento, Imágenes, Fotografía.

Keywords

Application, Android, Recognition, Images, Photography.

Índice general

Índice de figuras **11**

Índice de tablas **17**

Índice de códigos **18**

1. Introducción **21**

1.1. Motivación del Proyecto 21

1.2. Necesidades del Proyecto 22

1.2.1. Antecedentes 22

1.2.2. Solución 22

1.3. Contexto 23

1.4. Objetivos del Proyecto 24

2. Estudio Previo **27**

2.1. Android 27

2.1.1. ¿Qué es Android? 27

2.1.2. Características Técnicas 28

2.1.3. Entorno de Desarrollo 29

2.2. Servidor 31

2.2.1. Flask 31

2.2.2. Python Eve 31
2.2.3. Django 31
2.2.4. NodeJS 32
2.2.5. Spring 32
2.2.6. Conclusión 32
2.3. Base de datos 33
2.3.1. SQLite 33
2.3.2. MySQL 34
2.3.3. MongoDB 34
2.3.4. Conclusión 34
2.4. Reconocimiento de imágenes 35
2.4.1. Algoritmos de detección de características 35
2.4.2. Algoritmos de comparación de características 37
2.4.3. Conclusión 38

3. Descripción del proyecto **39**

3.0.4. Aplicación Android 40
3.0.5. Base de datos 41
3.0.6. Reconocimiento de imágenes 41
3.0.7. Servidor 41

4. Planificación del proyecto **43**

4.1. Metodología 43
4.1.1. SCRUM 44
4.2. Definición de las tareas 46
4.3. Planificación temporal de las tareas 51

4.4. Estimación de Recursos del Proyecto 55
4.4.1. Recursos Físicos 56
4.4.2. Recursos de Software 57
4.5. Seguimiento del Proyecto 59
4.6. Estimación de Costes 65

5. Análisis **67**

5.1. Visión general de la Aplicación 67
5.2. Análisis de requisitos 68
5.2.1. Requisitos funcionales 69
5.2.2. Requisitos de datos 75
5.2.3. Requisitos No Funcionales 76

6. Diseño **77**

6.1. Diseño de la interfaz 77
6.1.1. Elementos comunes 78
6.1.2. Vistas Principales 79
6.1.3. Vistas de Resultados 81
6.2. Diseño del sistema 83
6.2.1. Diseño de los Procesos 83
6.2.2. Diagramas de Clase 85

7. Implementación **87**

7.1. Aplicación Android 87
7.1.1. Interfaz Gráfico 87
7.1.2. Transición animada entre pantallas 89

7.1.3. Realización de la fotografía . . . 89
7.1.4. Comunicación con el servidor REST . . . 92
7.1.5. Presentación de los resultados . . . 94
7.1.6. Gestión de favoritos . . . 95
7.2. Reconocimiento de imágenes . . . 96
7.2.1. Detección y comparación de descriptores . . . 97
7.2.2. Proceso de Reconocimiento . . . 99
7.2.3. Paralelización del proceso . . . 101
7.3. Servidor . . . 102
7.3.1. Servicios REST . . . 102
7.3.2. Modelo . . . 103
7.3.3. Base de Datos . . . 103
7.3.4. Puesta en marcha . . . 104
7.4. Parámetros por defecto . . . 107
7.5. Parámetros algoritmo SURF . . . 115
7.6. Parámetros algoritmo SIFT . . . 118
7.7. Parámetros algoritmo ORB . . . 122
7.8. Parámetros algoritmo FLANN . . . 123

8. Resultado Final **127**

8.1. Tutorial . . . 129
8.2. Pantallas Principales . . . 130
8.3. Cámara y envío . . . 132
8.4. Resultados . . . 134

9. Conclusiones **137**

9.1. Conclusiones Técnicas 137

9.2. Conclusiones Personales 137

9.3. Trabajo Futuro 139

A. Definiciones y Abreviaturas **145**

Índice de figuras

1.1. Libros etiquetados con códigos de barras [6]. 22

1.2. Código QR utilizado en un cartel publicitario [2]. 23

2.1. Cuota de distribución de versiones de Android (Abril 2014). 28

2.2. Caputura del IDE Android Studio. 30

2.3. Ejemplo del detector SURF. 36

2.4. Ejemplo del comparador FLANN. 37

3.1. Visión global del sistema. 40

4.1. Marco de trabajo SCRUM. 45

4.2. Tareas para el diagrama de Gantt. 51

4.3. Diagrama de Gantt del proyecto. 52

4.4. Pila del producto en la herramienta Jira. 62

4.5. Reporte de un *sprint* en la herramienta Jira. 63

4.6. Tablero de tareas en la herramienta Jira. 64

4.7. Resumen del proyecto en la herramienta Jira. 64

5.1. Tablero de tareas en la herramienta Jira. 68

6.1. Visión general del flujo de la aplicación, mendiante prototipos. 78

6.2. Icono de la aplicación. 79

6.3. Prototipo de la vista “inicio”. 80

6.4. Prototipo de la vista ´´favoritos”. 80

6.5. Prototipo de la vista “cámara”. 81

6.6. Prototipo de la vista “tutorial”. 81

6.7. Prototipo de la vista “resultado web”. 82

6.8. Prototipo de la vista “resultado galería”. 82

6.9. Prototipo de la vista “resultado audio”. 83

6.10. Prototipo de la vista “resultado video”. 83

6.11. Diagrama de actividad del proceso principal de la aplicación. 84

6.12. Diagrama de datos de la parte correspondiente a la base de datos. 85

6.13. Parte del diagrama de clases del módulo de reconocimiento del servidor. 86

7.1. Diagrama de clases correspondiente a la transición entre pantallas. 89

7.2. Diagrama de clases correspondiente a la implementación de la cámara. 91

7.3. Diagrama de clases correspondiente a la comunicación con el servidor REST. . . 92

7.4. Diagrama de clases correspondiente a la presentación de resultados. 94

7.5. Diagrama de clases correspondiente a la gestión de favoritos. 96

7.6. Diagrama de clases del algoritmo detector. 98

7.7. Diagrama de clases del algoritmo comparador. 99

7.8. Diagrama de clases del algoritmo reconocedor. 100

7.9. Diagrama de clases del modelo contenido en el servidor. 103

7.10. Conjunto de imágenes utilizadas para las pruebas. 106

7.11. Comparación de algoritmos SURF, SIFT y ORB con la imagen *Android4Maier-Foto8*. 107

7.12. Comparación de algoritmos SURF, SIFT y ORB con la imágenes *Por1cafealdia-Foto3*. 107

7.13. Descriptores y coincidencias resultantes de comparar la imagen *Android4Maier-Foto8* con el resto del conjunto de datos de prueba utilizando el algoritmo SURF. 108

7.14. Descriptores y coincidencias resultantes de comparar la imagen *Por1cafealdia-Foto3* con el resto del conjunto de datos de prueba utilizando el algoritmo SURF. 109

7.15. Comparación de las coincidencias obtenidas para las imágenes *Android4Maier-Foto8* y *Por1cafealdia-Foto3* con el algoritmo SURF. 109

7.16. Descriptores y coincidencias resultantes de comparar la imagen *Android4Maier-Foto8* con el resto del conjunto de datos de prueba utilizando el algoritmo SIFT. 110

7.17. Descriptores y coincidencias resultantes de comparar la imagen *Por1cafealdia-Foto3* con el resto del conjunto de datos de prueba utilizando el algoritmo SIFT. 111

7.18. Comparación de las coincidencias obtenidas para las imágenes *Android4Maier-Foto8* y *Por1cafealdia-Foto3* con el algoritmo SIFT. 112

7.19. Descriptores y coincidencias resultantes de comparar la imagen *Android4Maier-Foto8* con el resto del conjunto de datos de prueba utilizando el algoritmo ORB. 113

7.20. Descriptores y coincidencias resultantes de comparar la imagen *Por1cafealdia-Foto3* con el resto del conjunto de datos de prueba utilizando el algoritmo ORB. 114

7.21. Comparación de las coincidencias obtenidas para las imágenes *Android4Maier-Foto8* y *Por1cafealdia-Foto3* con el algoritmo ORB. 114

7.22. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *hessianThreshold*. 115

7.23. Comparación de las coincidencias obtenidas para la imagen *por1cafealdia-Foto3*, con diferentes valores de *hessianThreshold*. 116

7.24. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *nOctaves*. 116

7.25. Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *nOctaves*. 117

7.26. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *nOctaveLayer*. 117

7.27. Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *nOctaveLayer*. 118

7.28. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *contrastThresold*. 119

7.29. Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *contrastThreshold*. 119

7.30. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *edgeThreshold*. 120

7.31. Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *edgeThreshold*. 120

7.32. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *nOctaveLayer*. 121

7.33. Comparación de las coincidencias obtenidas para la imagen *Poe1cafealdia-Foto3*, con diferentes valores de *nOctaveLayer*. 121

7.34. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores para sus parámetros. 122

7.35. Comparación de las coincidencias obtenidas para la imagen *Pro1cafealdia-Foto3*, con diferentes valores para sus parámetros. 123

7.36. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *FLANN_ INDEX*. 124

7.37. Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *FLANN_ INDEX*. 124

7.38. Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *COEFICIENTE_ DISTANCIA*. 125

7.39. Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *COEFICIENTE_ DISTANCIA*. 125

8.1. Visión general del flujo de la aplicación, mendiante capturas finales. 128

8.2. Captura final del paso 1 del tutorial. 129

8.3. Captura final del paso 2 del tutorial. 129

8.4. Captura final del paso 3 del tutorial. 130

8.5. Captura final del paso 4 del tutorial. 130

8.6. Captura final de la vista “inicio”. 131

8.7. Captura final de la vista “favoritos”. 131

8.8. Captura final de la vista “acerca de”. 132

8.9. Captura final de la vista “licencias”. 132

8.10. Captura final de la vista “cámara”. 133

8.11. Captura final de la vista “animación”. 133

8.12. Captura final de la vista “error”. 133

8.13. Captura final de la vista “resultado web”. 134

8.14. Captura final de la vista “resultado galería”. 134

8.15. Captura final de la vista “audio”. 135

8.16. Captura final de la vista “video”. 135

Índice de tablas

1.1. Distribución de las partes del proyecto. 24

2.1. Tabla resumen de los *frameworks* analizados. 33

2.2. Tabla resumen de las bases de datos analizadas. 34

2.3. Resultados obtenidos de las pruebas sobre los algoritmos con el programa Find-Object. 37

4.1. Historias de usuario del proyecto. 48

4.2. División en tareas de las historias de usuario del proyecto. 50

4.3. Estimación de las historias de usuario. 54

4.4. Duración de los *sprints* del proyecto. 55

4.5. Distribución de las historias de usuario entre los *sprints* del proyecto. 59

4.6. Comparación del tiempo real y estimado de las historias de usuario. 61

5.1. Plantilla utilizada para definir los casos de uso. 69

5.2. Plantilla utilizada para definir los requisitos de datos. 69

5.3. Plantilla utilizada para definir los requisitos no funcionales. 69

5.4. Caso de uso 01, Realizar fotografía. 70

5.5. Caso de uso 02, Reconocer imagen. 70

5.6. Caso de uso 03, Proporcionar información. 71

5.7. Caso de uso 03, Ver resultado del análisis. 71

5.8. Caso de uso 05, Compartir resultados. 72

5.9. Caso de uso 06, Gestionar favoritos. 72

5.10. Caso de uso 07, Ver información sobre la aplicación. 73

5.11. Caso de uso 08, Ver tutorial. 73

5.12. Caso de uso 09, Gestionar campañas. 74

5.13. Caso de uso 10, Gestionar imágenes. 74

5.14. Caso de uso 11, Ver estadísticas. 75

5.15. Requisito de datos 01, Imágenes. 75

5.16. Requisito de datos 02, Campañas. 75

5.17. Requisito de datos 03, Accesos. 76

5.18. Requisito de datos 04, Favoritos. 76

5.19. Requisito no funcional 01, Usabilidad. 76

5.20. Requisito no funcional 02, Respuesta rápida. 76

7.1. Características de los dispositivos utilizados para las pruebas (Fuente: *GSMArena*).105

7.2. Valores por defecto del algoritmo SURF. 115

7.3. Valores por defecto del algoritmo SIFT. 118

7.4. Valores por defecto del algoritmo ORB. 122

Índice de códigos

7.1. Parte del *layout* de *InicioFragment*. 87
7.2. Método *hacerFoto*. 90
7.3. Método *pedirImagen*. 91
7.4. Parte del escuchador *JsonHttpResponseHandler* utilizado para recibir la respuesta del servidor. 93
7.5. Interfaz *FavoritoDao*. 96
7.6. Fragmento del método *reconocer*. 100
7.7. Servicio REST para obtener el thumbnail de una imagen. 102
7.8. Clase correspondiente al objeto *BaseDeDatos*, utilizado para la conexión con la base de datos. 104

Capítulo 1

Introducción

En este capítulo introductorio se presenta una breve descripción del proyecto, la motivación por la que se ha realizado y su situación en el contexto actual. También se describe las necesidades que han llevado a la elaboración de este y la solución que se busca subsanar. Por último, se describen los objetivos que se pretenden abarcar durante su elaboración.

1.1. Motivación del Proyecto

Desde que en 2007 Apple sacara al mercado el novedoso iPhone, el mundo de la telefonía móvil ha evolucionado a un ritmo asombroso. Los mismos usuarios que antiguamente enviaban SMS con sus serviciales teléfonos de bolsillo, han pasado a estar continuamente conectados con los impresionantes teléfonos inteligentes, bautizados como *Smartphones*, de los que disponemos en la actualidad.

El éxito de estos nuevos dispositivos ha sido inaudito, tal es así, que en febrero de 2013 sólo en España existían 55.740.000 millones de teléfonos móviles, lo que corresponde a un 118 % de la población, con lo que podemos afirmar que prácticamente todos los españoles disponen de un Smartphone en sus bolsillos [15].

El usuario de hoy en día ya está acostumbrado a utilizar aplicaciones que le facilitan acciones de la vida cotidiana. Son muchas las aplicaciones que han ido viendo la luz y ofrecen servicios de todo tipo para el usuario. Por ejemplo, gracias a aplicaciones como *Google Maps*, llegar a una dirección nueva se convierte en una tarea fácil. Otra labor que resulta realmente útil y fascinante es poder saber qué canción está sonando mediante la grabación de unos segundos, gracias a la aplicación *Shazam*.

La aplicación realizada en este proyecto pretende facilitar información al usuario a través de la realización de una fotografía con su *Smartphone*. Es decir, el usuario podría comprar una entrada para un concierto a raíz de realizar una fotografía al cartel de este, podría también escuchar un CD de música capturando con su móvil la portada de este, o podría obtener información sobre un cuadro al fotografiarlo y un largo etcétera.

1.2. Necesidades del Proyecto

1.2.1. Antecedentes

En la actualidad existen varias maneras de poder etiquetar un producto para su posterior identificación. Uno de los más utilizados desde hace muchos años son los códigos de barras (Figura 1.1). Gracias a estos y con la ayuda de un dispositivo adecuado para ello, es posible leer este código y saber de qué producto se trata. Son utilizados desde hace muchos años para todo tipo de rastreo y control de productos, como puede ser en un supermercado o en una biblioteca.

Figura 1.1: Libros etiquetados con códigos de barras [6].

Con la nueva era, en lo que a tecnología se refiere, la de los *Smartphones*, han adquirido gran protagonismo en este ámbito los códigos bidimensionales, los llamados códigos QR (Figura 1.2). Estos códigos se pueden detectar al instante utilizando la cámara de un *Smartphone*, por lo tanto cualquier persona poseedora de uno de estos populares dispositivos puede capturarlos. Una vez el dispositivo los ha reconocido, estos proporcionan la información almacenada, como una URL, la cual será mostrada en el mismo dispositivo.

1.2.2. Solución

Con este proyecto se propone una alternativa más visual a las comentadas en la sección anterior. Para ello se ha optado por identificar los productos de una forma diferente, utilizando su propia imagen. De esta manera se podrían etiquetar productos y posteriormente identificarlos simplemente con fotografías. Además, de esta manera un producto se representaría a sí mismo, sin necesidad de códigos o caracteres irreconocibles por el ojo humano.

Por lo tanto, la solución propuesta pasa por la implementación de los siguientes puntos:

- Una base de datos que relacione las imágenes que pueden ser capturadas con las información que se desea mostrar, pudiendo ser cualquier tipo de información requerida por la empresa.

Figura 1.2: Código QR utilizado en un cartel publicitario [2].

- Un sistema que sea capaz de buscar una imagen en la base de datos y devolver la información asociada a ella.

- Una aplicación móvil que permita realizar fotografías, conectar con el sistema de reconocimiento de imágenes, y finalmente, mostrar la información asociada a la imagen en cuestión.

1.3. Contexto

El trabajo realizado es parte de un proyecto propuesto y elaborado en la empresa *Rubycon Information Technologies*, la cual desarrolla aplicaciones para dispositivos Android e iOS.

Dicha empresa es una *Start-Up* de Base Tecnológica, que ha sido creada en los últimos años y apuesta de lleno por las tecnologías actuales con el objetivo de facilitar a la sociedad el acceso a la información, ofreciéndoles herramientas para hacerlo en cualquier lugar y momento [9].

Debido a la amplitud del proyecto, ha sido dividido en dos proyectos compartiendo el 50 %, el cual se ha realizado de manera conjunta. Por un lado, se ha elaborado para el sistema operativo Android, mientras que por otro lado, se ha realizado la misma aplicación para iOS, la cual se ha llevado a cabo por el alumno Sergi Estellés Jovaní.

En cuanto a la parte en común, desde el principio de la estancia en prácticas los responsables de la empresa dejaron claro que debería desarrollarse utilizando la técnica del *Pair Programming,* la cual consiste en el desarrollo en pareja en un mismo ordenador, favoreciendo así el resultado

final del producto y el trabajo en equipo [8]. Además, hicieron hincapié en que el objetivo de los dos proyectos era el mismo, por lo tanto, el trabajo debería realizarse como un equipo de desarrollo real y manteniendo la consistencia en todo momento entre las dos aplicaciones. Junto con todo esto, los supervisores en la empresa nos han inculcado durante la estancia en prácticas el uso de las metodologías ágiles, que tan buenos resultados están dando en la actualidad.

En lo referente a la aplicación móvil, puede ser desarrollada para diferentes sistemas operativos, entre los cuales los más destacados en la actualidad son Android (Google), iOS (Apple) y Windows Phone (Microsoft).

En este caso, la aplicación se ha desarrollado para el primero, un sistema operativo de código libre y con nombre de llegar de otro planeta. La evolución de Android desde su salida ha sido constante. Su gran potencial unido a la ventaja de ser de código libre, por lo cual diferentes fabricantes pueden crear sus versiones personalizadas de él, le han llevado a conseguir una gran acogida entre los usuarios, llegando en la actualidad a abarcar un 81.3 % de la cuota de mercado actual [1].

Por último, para hacer constancia de las partes que se han desarrollado de manera totalmente individual y las que se han realizado de manera común en los dos proyectos, en la siguiente tabla se detalla el reparto de estas entre los dos alumnos.

	Javier Agut	Sergi Estellés
Servidor REST	X	X
Base de datos MongoDB	X	X
Base de datos Android	X	
Base de datos iOS		X
Reconocimiento de imágenes	X	X
Reproductores Android	X	
Reproductores iOS		X
Cliente móvil Android	X	
Cliente movil iOS		X
Diseño interfaces	X	X
Gestión del proyecto	X	X

Tabla 1.1: Distribución de las partes del proyecto.

1.4. Objetivos del Proyecto

Los objetivos que se deben cumplir durante el desarrollo de este **Trabajo Final de Grado** se detallan en los siguientes puntos:

- **Desarrollo de una aplicación para el sistema operativo Android**, a través de la cual el usuario pueda realizar una fotografía y obtener la información correspondiente a esta.
- **Implementación de un sistema capaz de reconocer imágenes** en una base de datos,

la cual contiene la diferente información asociada a estas.

- **Obtener una experiencia real de trabajo con metodologías ágiles**, es decir, poner en práctica los conocimientos teóricos adquiridos durante la carrera, trabajando en un equipo de desarrollo real con el fin de la obtención de un producto final, pudiendo así comprobar y entender de manera práctica cómo funciona un equipo de desarrollo ágil.
- **Formar parte de un equipo real con un objetivo común**, el cual implica el desarrollo de la misma aplicación en dos sistemas diferentes, con los inconvenientes que conlleva el uso de dos plataformas distintas. Además de los diferentes estándares que hay que estipular y la constante revisión de que el rumbo que se está siguiendo es el mismo.

Capítulo 2

Estudio Previo

En el siguiente capítulo se aborda un estudio de las diferentes tecnologías utilizadas para la elaboración del proyecto, desde el sistema operativo Android, hasta el algoritmo de reconocimiento de imágenes, pasando por las tecnologías referentes a la base de datos y al servidor.

2.1. Android

En el siguiente apartado se realiza una introducción al sistema operativo Android, en la cual se comentan algunas de sus características más importantes, ya que explicar el sistema el profundidad resultaría muy extenso para este proyecto.

2.1.1. ¿Qué es Android?

Android fue inicialmente desarrollado por Android Inc., empresa adquirida por Google en 2005. Pero fue en 2008 cuando empezó a obtener popularidad gracias a la fundación del Open Handset Alliance, un consorcio formado por 48 empresas de desarrollo hardware, software y de telecomunicaciones, que decidieron promocionar el software libre [33].

Android es un sistema operativo, basado en Linux, inicialmente pensado para dispositivos móviles, pero que ha ido evolucionando hasta ser utilizado en tabletas, televisores, relojes, coches, etc. Una de las diferencias con otros sistemas operativos es que cualquier persona que sepa programar puede crear nuevas aplicaciones o, incluso, realizar una modificación de este, ya que es de código libre.

Desde que se lanzo Android Beta en noviembre de 2007, han hecho publicas catorce versiones más de este sistema operativo. Y en la actualizad, aún existen dispositivos con ocho de estas en vigor. En la Figura 2.1 se puede ver un gráfico con la cuota de distribución de versiones de Android en Abril del 2004.

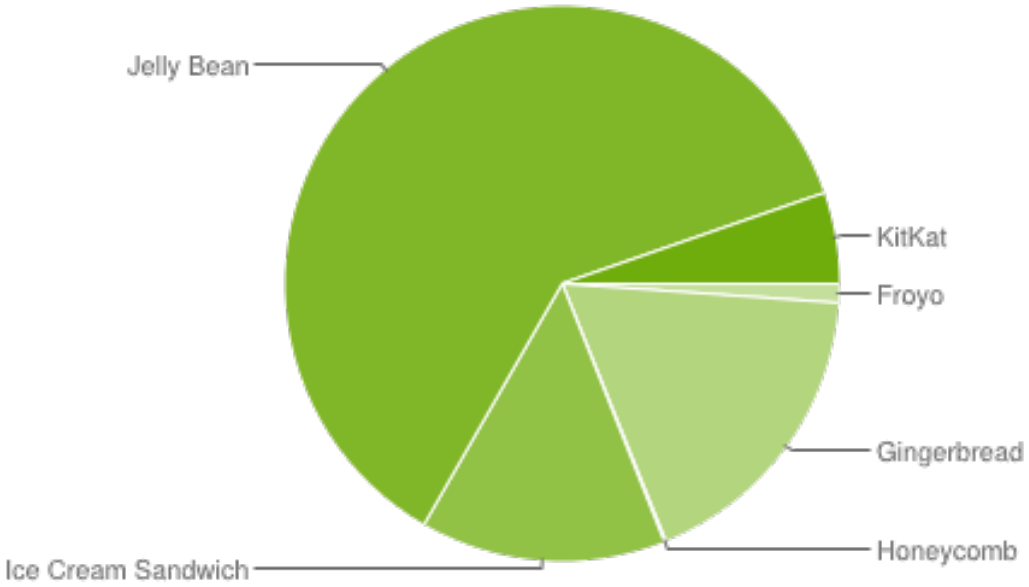


Figura 2.1: Cuota de distribución de versiones de Android (Abril 2014).

2.1.2. Características Técnicas

A continuación se presentan las características técnicas del sistema operativo Android [33]:

Diseño de dispositivo: La plataforma es adaptable a pantallas de mayor resolución, VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 2.0 y diseño de teléfonos tradicionales. Almacenamiento: SQLite, una base de datos liviana, que es usada para propósitos de almacenamiento de datos. Conectividad: Android soporta las siguientes tecnologías de conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC y WiMAX.GPRS,UMTS,HSPA+ Y HSDPA+ Mensajería: SMS y MMS son formas de mensajería, incluyendo mensajería de texto y ahora la Android Cloud to Device Messaging Framework (C2DM) es parte del servicio de Push Messaging de Android.

Navegador web: El navegador web incluido en Android está basado en el motor de renderizado de código abierto WebKit, emparejado con el motor JavaScript V8 de Google Chrome. El navegador por defecto de Ice Cream Sandwich obtiene una puntuación de 100/100 en el test Acid3. Soporte de Java: Aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable Dalvik y corre en la Máquina Virtual Dalvik. Dalvik es una máquina virtual especializada, diseñada específicamente para Android y optimizada para dispositivos móviles que funcionan con batería y que tienen memoria y procesador limitados. El soporte para J2ME puede ser agregado mediante aplicaciones de terceros como el J2ME MIDP Runner. Soporte multimedia: Android soporta los siguientes formatos multimedia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.

Soporte para streaming: Streaming RTP/RTSP (3GPP PSS, ISMA), descarga progresiva de HTML (HTML5 ¡video¿tag). Adobe Flash Streaming (RTMP) es soportado mediante el Adobe Flash Player. Se planea el soporte de Microsoft Smooth Streaming con el port de Silverlight a Android. Adobe Flash HTTP Dynamic Streaming estará disponible mediante una actualización de Adobe Flash Player.

Soporte para hardware adicional: Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.

Entorno de desarrollo: Incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software. El entorno de desarrollo integrado es Eclipse (actualmente 3.4, 3.5 o 3.6) usando el plugin de Herramientas de Desarrollo de Android. Google Play: Google Play es un catálogo de aplicaciones gratuitas o de pago en el que pueden ser descargadas e instaladas en dispositivos Android sin la necesidad de un PC.

Multi-táctil: Android tiene soporte nativo para pantallas capacitivas con soporte multi-táctil que inicialmente hicieron su aparición en dispositivos como el HTC Hero. La funcionalidad fue originalmente desactivada a nivel de kernel (posiblemente para evitar infringir patentes de otras compañías). Más tarde, Google publicó una actualización para el Nexus One y el Motorola Droid que activa el soporte multi-táctil de forma nativa.

Bluetooth: El soporte para A2DF y AVRCP fue agregado en la versión 1.5; el envío de archivos (OPP) y la exploración del directorio telefónico fueron agregados en la versión 2.0; y el marcado por voz junto con el envío de contactos entre teléfonos lo fueron en la versión 2.2.

Videollamada: Android soporta videollamada a través de Google Talk desde su versión HoneyComb.

Multitarea: Multitarea real de aplicaciones está disponible, es decir, las aplicaciones que no estén ejecutándose en primer plano reciben ciclos de reloj.

Características basadas en voz: La búsqueda en Google a través de voz está disponible como .Entrada de Búsqueda"desde la versión inicial del sistema.

Tethering: Android soporta tethering, que permite al teléfono ser usado como un punto de acceso alámbrico o inalámbrico (todos los teléfonos desde la versión 2.2, no oficial en teléfonos con versión 1.6 o inferiores mediante aplicaciones disponibles en Google Play (por ejemplo PdaNet). Para permitir a un PC usar la conexión de datos del móvil Android se podría requerir la instalación de software adicional.

2.1.3. Entorno de Desarrollo

Desde que se lanzara Android, Google ofrece un kit de desarrollo para facilitar la programación de las aplicaciones para este sistema operativo. Hasta el presente se recomendaba a los programadores utilizar el conocido IDE Eclipse y, para ello, se proporciona un plugin llamado ADT. Incluso, desde la página de desarrolladores de Android, se pude descargar una versión

con todo lo necesario para realizar esta tarea. No obstante, en la edición de Google I/O del año 2013, se lanzó la primera preview de Android Studio (Figura 2.2). Este es un entorno de desarrollo integrado para Android, basado en IntelliJ IDEA. Google recomienda su uso para la implementación de aplicaciones Android, y es por eso que ha ido potenciando este entorno, en contra de Eclipse.

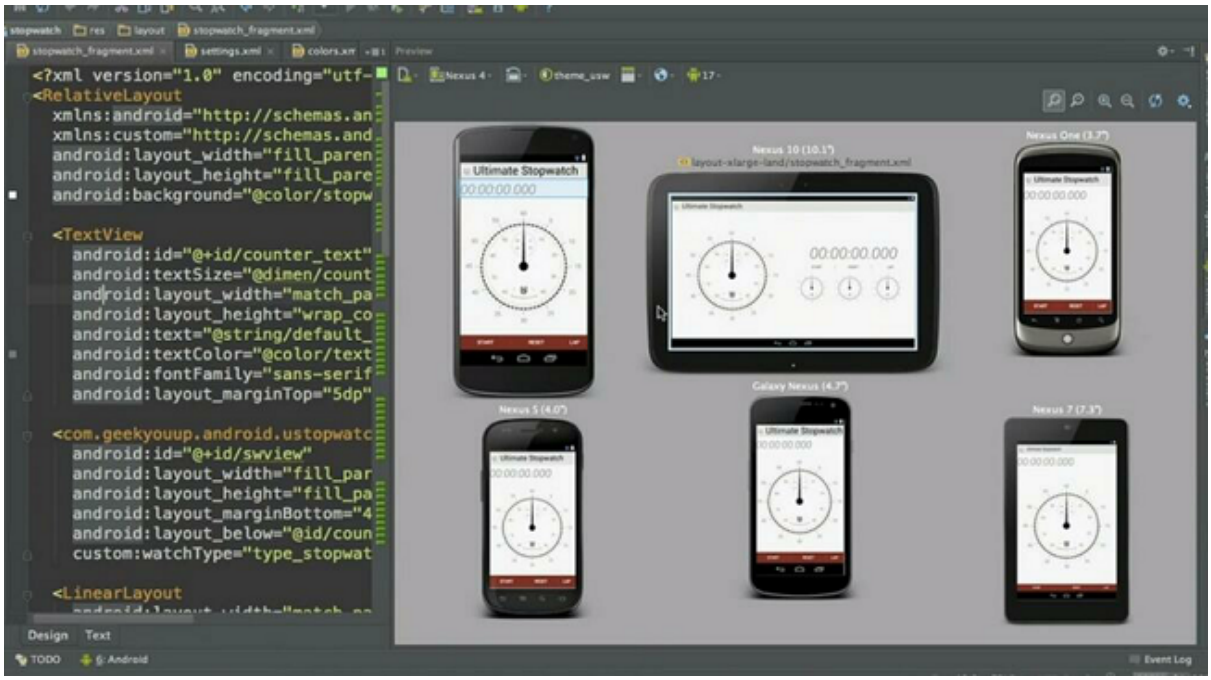


Figura 2.2: Caputura del IDE Android Studio.

Si no se opta por la utilización de estos IDEs ya preparados y configurados por Google, será necesario descargar por una parte el SDK de Android (Android Software Development Kit), y por otra, el JDK de Java (Java Development Kit) necesario para el uso del anterior.

Para el desarrollo de este proyecto se ha optado, bajo la recomendación del experto en Android de la empresa, por la utilización de Android Studio, ya que incorpora nuevas características y tiene el apoyo total de Google. Algunas de las características más importantes de esto son:

- Soporte para Gradle.
- Refactorización específica para Android y soluciones rápidas.
- Herramientas para medir el rendimiento, usabilidad, compatibilidad de versiones y otros problemas.
- ProGuard y capacidad para firmar apps
- Asistentes basados en plantillas para crear diseños y componentes en Android.

- Un editor rico en cuanto a diseño, que permite realizar drag and drop a los componentes de la UI.
- Vista previa de los layouts en múltiples configuraciones de pantalla en simultaneo y mucho más.

2.2. Servidor

A continuación se presentan las diferentes opciones valoradas para la implementación del servidor con la API Rest.

2.2.1. Flask

Este es un *microframework* para el lenguaje Python basado en las librerías Werkzeug y JInja 2. Una de sus mayores características es minimalista y sencillez, con tan solo un siete líneas de código Python se tiene una servidor "Hola Mundo"montado y en marcha [12].

A pesar de su sencillez, dispone de un gran potencial. Este está enfocado inicialmente para el desarrollo de páginas web, pero con él también se pueden implementar servicios RESTful y, como su estilo indica, de una manera muy sencilla.

Su paradigma de programación está orientado a objetos y se puede hacer uso de él teniendo en cuenta su licencia BSD-Licence. Gracias a sus numerosos plugins se pueden utilizar diferentes bases de datos, como PostreSQL, MariaDB, MySQL, MongoDB, CouchDB, Oracle y SQLite.

2.2.2. Python Eve

Eve es un framework que usa Flask, MongoDB y Redis como motor. Gracias a el se puede implementar de una manera sencilla servicios web RESTful con todas sus funciones. Si Flask de por si ya es sencillo, los creadores de este framework buscan utilizar y reducir aún más su sencillez centrándose en este tipo de servicios web. Su filosofía pasa por que cualquier pueda ofrecer una API REST sin tener grandes conocimientos sobre ello [27].

Al igual que su motor Flask, ofrece una licencia BSD-Licence y su paradigma de programación es orientado a objetos. En cuanto a las bases de datos se pueden utilizar directamente MongoDB, MySQL y otras NoSQL.

2.2.3. Django

Django es un framework web escrito en Python y de código abierto que permite al desarrollador construir aplicaciones web de manera rápida y con menos código. Sus principios pasan

por la reutilización de código, la conectividad, la extensión de componentes, el desarrollo rápido y el principio "No te repidas", que hace hincapié la reutilización de código [16].

Este también permite crear servicios RESTful, aunque no es su principal meta, es muy utilizado por los desarrolladores de sitios webs complejos. Su paradigma de programación es orientado a objetos y su uso está disponible bajo una licencia BSD-Licence.

2.2.4. NodeJS

Este es un entorno de programación basado en el conocido leguaje de programación Javascript. Fue creado para ser útil en la elaboración de programas de red y se utiliza para la programación en la capa del servidor [21].

NodeJS dispone de soporte directo para bases de datos MongoDB y MySQL. Además, a parte de Javascript, también puede ser programado con Ruby. Su paradigma de programación puede ser orientado a objetos, orientado a eventos o funcional. Y, por último, está disponible bajo una licencia MIT License.

2.2.5. Spring

Spring es un framework que permite desarrollar aplicaciones mediante la plataforma Java. Este proporciona un contenedor ligero que se puede utilizar tanto para aplicaciones web o para cualquier aplicación desarrollada con Java, aunque destaca por ser utilizado en entornos web [32].

Es de código abierto y se puede utilizar bajo la licencia Apache License GPLv2. SU paradigma de programación es orientado a aspectos y a parte de Java, también se puede utilizar con ruby, C# y Python.

2.2.6. Conclusión

Después de estudiar las diferentes opciones para implementar la API REST podemos sacar conclusiones de qué tecnología es la que más nos conviene utilizar. Para ello, se ha elaborado una tabla a modo resumen, la cual se puede ver en la Tabla 2.1.

Finalmente, siguiendo las recomendaciones ofrecidas por el experto en servidores de la empresa, la balanza se decantó hacia Flask. Este es un framework muy potente a la vez que sencillo, con una rápida curva de aprendizaje. Además, el uso de Python es una ventaja, ya que es un lenguaje sencillo y se ha utilizado a lo largo de la carrera cursada por el alumno.

	Flask	Python Eve	Django	NodeJS	Spring
Licencia	BSD-Licence	BSD-Licence	BSD-Licence	MIT Licence	Apache Licence GPLv2
Bases de Datos	PostgreSQL, MariaDB, MySQL, MongoDB, CouchDB, SQLite	MongoDB, MySQL, NoSQL	PostgreSQL, MySQL, SQLite	MongoDB, MySQL	Relacionales y no relacionales
Paradigmas de programación	Orientado a objetos	Orientado a objetos	Orientado a objetos	Orientado a objetos, orientado a eventos, funcional	Orientado a aspectos
Lenguaje de programación	Python	Python	Python	JavaScript, ruby	Java, ruby, C#, Python

Tabla 2.1: Tabla resumen de los *frameworks* analizados.

2.3. Base de datos

Actualmente, existen diferentes maneras de persistir datos. Para decantarnos por una de ellas hemos analizado diversos gestores de bases de datos recomendados por el experto de la empresa en la materia. Antes de nada, hay que tener en cuenta que en este sistema no se va a implementar una gran base de datos con infinidad de tablas, en principio, solo se dispondrá de una sola tabla donde se guardarán las imágenes, aunque estas dispongan de diferentes tipos de información.

2.3.1. SQLite

Es un sistema gestor de bases de datos relacional interna para bases de datos pequeñas, es decir, la información podría ser almacenada en el dispositivo. Además, Android ofrece facilidades para trabajar con este tipo de bases de datos [10].

Esta peculiaridad, por un lado, puede favorecer el coste de transacciones entre el dispositivo y el servidor pero, por otro lado, consumiría mucha memoria del dispositivo, limitando el tamaño de la base de datos considerablemente. Comentar también que la curva de aprendizaje es bastante rápida, siendo este tipo de bases de datos ideales pare realizar pruebas.

2.3.2. MySQL

Esta es una de las bases de datos más conocidas. Se trata de un sistema de gestión de base de datos relacional, el cual dispone de muchas opciones de personalización, como puede ser la gestión de usuarios, o la implementación de índice, cosa que puede beneficiar al rendimiento en sistemas de gran tamaño [31].

El almacenamiento se realiza a través de un servidor y su curva de aprendizaje no es nada alta, ya que se ha utilizado en varias asignaturas de la carrera .

2.3.3. MongoDB

Cuando se habla de MongoDB, estamos hablando de un sistema de gestión de bases de datos no relacionales, o noSQL ([35]). Es decir, un base de datos que no tiene tablas, tiene colecciones de datos [25].

En las colecciones se almacenan contenidos que pueden tener diferentes campos. Más concretamente, este sistema almacena los datos en documentos de tipo JSON, cosa que puede favorecer la integración con las aplicaciones que trabajen con este tipo de formatos. Si se implementa una API REST que envie y reciba datos en forma de JSON, con este tipo de da bases de datos se puede agilizar el proceso, ya que los datos ya estarían alojados en este formato, en este caso, esto podría ser un gran punto a favor de MongoDB.

2.3.4. Conclusión

Una vez estudiados los pros y los contras de estos tres sistemas de gestión de bases de datos, se ha elaborado una tabla resumen (2.2).

Finalmente, se ha considerado implementar la base de datos con MongoDB, ya que en esta se deben guardar imágenes que almacenen diferentes tipos de información, y el hecho de que los datos se guarden en colecciones facilita esta tarea.

Por otro lado, como los datos entre la aplicación y el servidor se tendrán que gestionar mediante una API REST, si estos se guardan en forma de JSON, se pueden enviar directamente sin tener que transformarlos, un punto también muy determinante.

	SQLite	**MySQL**	**MongoDB**
Almacenamiento	Dispositivo	Servidor	Servidor
Tipo	Relacional	Relacional	No relacional
Configurable	No	Si	Si

Tabla 2.2: Tabla resumen de las bases de datos analizadas.

2.4. Reconocimiento de imágenes

Existen varios métodos de reconocer imágenes. Uno de las más utilizados en la actualidad es el reconocimiento mediante el uso de las herramientas ofrecidas por OpenCV.

OpenCV es una biblioteca de visión artificial de código libre que fue desarrollada originalmente por Intel. En ella se ofrecen diferentes algoritmos para detectar y comparar imágenes, a parte de otros tipos de procesamiento de imágenes. Esta biblioteca está escrita en C/C++, pero puede ser utilizada desde otros lenguajes como Python, Java y MATLAB. Además, se puede instalar en diferentes sistemas operativos, como son Windows, Linux, Mac OS, Android e iOS [18].

Según la documentación de OpencCV, una buen manera para reconocer una imagen dentro de otra es, primero, analizando las dos imágenes y obteniendo sus descriptores con un algoritmo de detección de características (Feature Detection) y, posteriormente, comparando los descriptores de las dos imágenes mediante un algoritmo de comparación de características (Feature Matching). Los algoritmos utilizados en la documentación de OpenCV son SIFT, SURF ORB, mientras que los utilizados de comparación son FLANN y BFMatcher. En los siguientes apartados se estudiarán estos algoritmos para obtener una conclusión final sobre cuál conviene utilizar para la implementación del proyecto.

2.4.1. Algoritmos de detección de características

En primer lugar, encontramos el algoritmo SIFT (Scale-Invariant Feature Transform). Este transforma la información obtenida de la imagen en coordenadas invariantes a la escala, rotación y luminosidad de la imagen. Esto también supone un mayor coste de computación que en otros algoritmos, ya que tiene que tener en cuenta bastantes factores [20].

Este algoritmo consta de cuatro etapas: generación del espacio escala mediante funciones diferenciales gaussianas; localización de puntos invariantes; asignación de orientación a los puntos y generación del descriptor del punto.

Por otro parte, existe el algoritmo SURF (Speed Up Robust Featue), el cual es muy similar al anterior y también pretende extraer los puntos invariantes de una imagen (Figura 2.3). Pero a diferencia del anterior, este reduce el coste computacional, siendo así más rápido que el anterior. Las etapas de este algoritmo son: creación del espacio escala; localización de puntos invariantes; asignación de orientación y generación del descriptor.

Figura 2.3: Ejemplo del detector SURF.

Una alternativa a los anteriores es el algoritmo ORB (Orientated FAST and Roate BIREF). Consiste en una unión del detector FAST y descriptor BRIEF con algunas modificaciones. ORB es más rápido que SURF y SIFT, además de ser una alternativa libre, ya que estos están patentados para uso comercial [30].

Para obtener una comparación sobre los diferentes comparadores, se realizaron una serie de pruebas con un par de imágenes utilizando el programa Find-Object (corriendo en un procesador Core2Duo a 2,33GHz). Este programa permite probar diferentes algoritmos de OpenCV mediante una simple interfaz [24]. Los resultados de estas pruebas se pueden ver en la Tabla 2.3, en los cuales se puede observar para cada combinación de detector-descriptor: el número de características encontradas, el tiempo de detección, el tiempo de procesamiento y el tiempo total. Con estos resultados se puede ver que los mejores tiempos se obtienen con SURF.

Detector	Descriptor	Núm. caract. encontradas	Tiempo detección caract.(ms)	Tiempo procesamiento descriptores (ms)	Tiempo total(ms)
BRISK	SURF	70	1300	45	1345
SIFT	SURF	130	150	55	205
Star	SURF	93	51	30	81
SURF	**SURF**	**160**	**260**	**140**	**300**
SURF	SIFT	137	250	680	930
SURF	BRISK	140	280	1400	1680
SURF	FREAK	100	270	230	500

Tabla 2.3: Resultados obtenidos de las pruebas sobre los algoritmos con el programa Find-Object.

2.4.2. Algoritmos de comparación de características

En lo referente a los comparadores de características, según la documentación de OpenCV, el más destacable es FLANN (Figura 2.4). Este es una biblioteca para realizar búsquedas aproximadas en espacios dimensionales elevados, que contiene una colección de algoritmos para trabajar adecuadamente en la búsqueda de los vecinos más cercanos y un sistema que elige automáticamente el mejor algoritmo junto con los parámetros óptimos en función del conjunto de datos analizados [23].

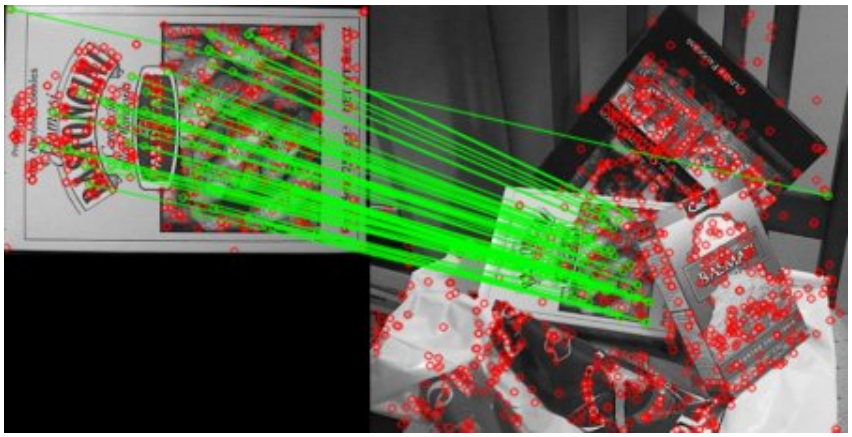


Figura 2.4: Ejemplo del comparador FLANN.

Por otro lado, existe una alternativa llamada BFMatcher (Brute-Forece Matcher), que consiste en una comparación a "fuerza bruta", es decir, compara todas las características con todas, utilizando al igual que FLANN la distancia entre los puntos [29].

2.4.3. Conclusión

Tras investigar sobre los algoritmos de reconocimientos de imágenes, se han visto las diferentes posibilidades para lograr que una imagen sea reconocida. Parece ser que los algoritmos más utilizados sean la combinación de SURF y FLANN, pero como existen diferentes algoritmos y no se puede saber cuál será el más indicado para este proyecto, se realizara una mínima implementación de todos y se someterán a distintas pruebas para llegar a una conclusión sobre los que se deben utilizar (dichas pruebas se pueden ver en el Capítulo **??**).

Capítulo 3

Descripción del proyecto

Como ya ha sido comentado en la introducción (Capítulo 1), el proyecto consiste en una aplicación para el sistema operativo Android que sea capaz de reconocer imágenes y mostrar la información que se le ha asociado anteriormente, la cual se encuentra almacena en un servidor.

Para conseguir este resultado, se requiere la implementación de diferentes componentes que permitan alcanzar los objetivos planteados para el sistema. La parte que más representará al proyecto será la aplicación, la cual ejercerá como capa de abstracción entre el usuario final y el sistema. Además, hará falta un servidor a través del cual se puedan realizar peticiones desde el dispositivo móvil, y desde donde se realizará el análisis y reconocimiento de la imagen, ya que realizarlo todo desde el dispositivo móvil supondría unas altas prestaciones en este, limitando los dispositivos desde los que se podría utilizar el sistema.

Por lo tanto, el servidor debe, por un lado, contener el algoritmo de reconocimiento de imágenes, además de estar conectado a una base de datos que contendrá las imágenes con las que tiene que comparar junto con la información que se quiera ofrecer de cada una. Y, por otro lado, deberá ofrecer un API para que se pueda acceder a este proceso desde la aplicación móvil.

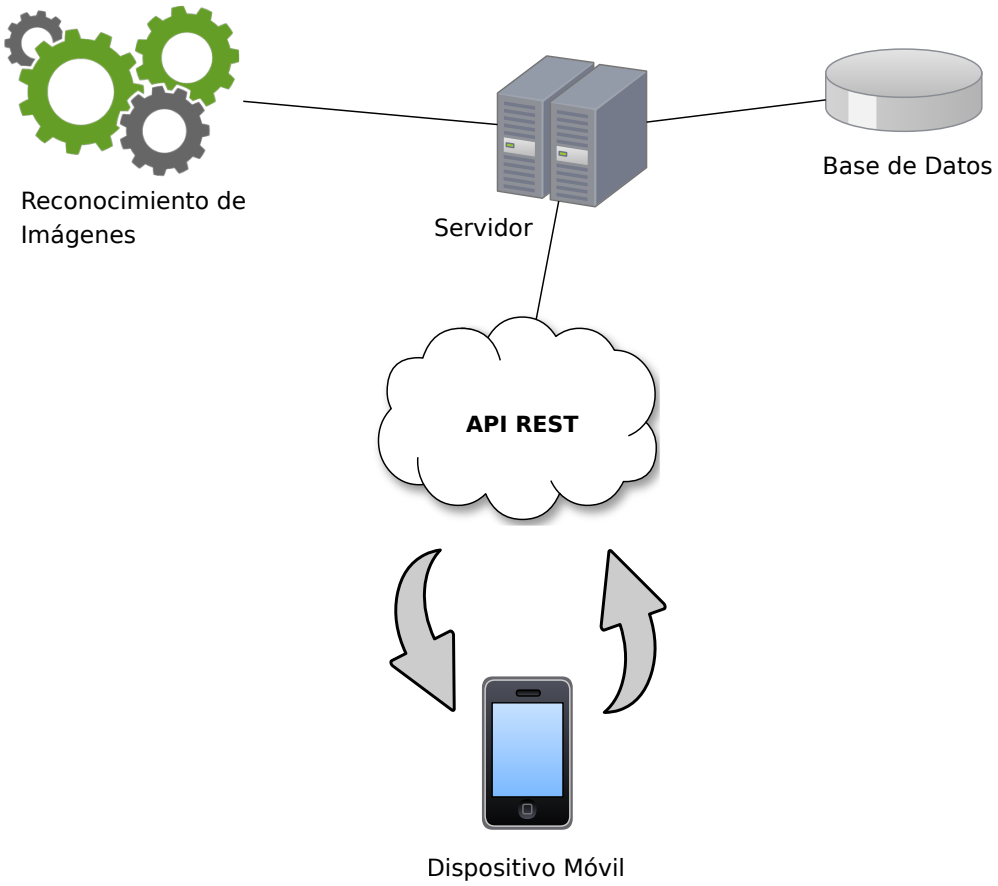


Figura 3.1: Visión global del sistema.

3.0.4. Aplicación Android

Consistirá en una aplicación nativa para el sistema operativo Android, y será la parte correspondiente al cliente. Sin duda, este es el punto más visual del proyecto y es por ello que, a la hora de su elaboración, se han tenido en consideración los estándares de usabilidad planteados por la plataforma utilizada, en este caso Android [5].

Su funcionabilidad consiste en permitir realizar fotografías desde ella y contactarlas con el sistema de reconocimiento de imágenes, a través de la API proporcionada por el servidor. Asimismo, debe mostrar los resultados del reconocimiento desde dentro de la aplicación, resultados que pueden ser de diferentes tipos: video, audio, galería de imágenes, página web o código HTML.

Para mostrarlos sin salir de la aplicación se requiere la implementación de un reproductor de video, uno de audio y una galería, específicos para la aplicación.

3.0.5. Base de datos

En lo referente a la base de datos, supone un punto importante en el sistema ya que es en ella donde se almacenan las imágenes para que puedan ser reconocidas, además de la información que se ofrecerá al obtener el resultado de este proceso.

Por otro lado, también se hace uso de la base de datos para almacenar estadísticas sobre los análisis realizados, tales como los registros de acceso, el sistema operativo desde donde se han enviado las imágenes, los resultados obtenidos, etc.

Cabe destacar que, al contener las imágenes resultados de diferentes tipos, no se ha utilizado una base de datos relacional, sino que se ha optado por una NoSQL que, en vez de guardar los datos en tablas, lo hacen en estructuras de datos, como por ejemplo en formato JSON.

3.0.6. Reconocimiento de imágenes

El resultado que se pretende alcanzar mediante esta parte del sistema es el del reconocimiento de una imagen dentro de una base de datos a través del siguiente proceso.

En primer lugar, se toma una imagen que directamente va a ser comparada con un conjunto de estas. Va a existir una base de datos en la que hay almacenadas una agrupación de imágenes que van a servir para comparar la imagen tomada principalmente. Al realizar está comparación, el sistema va a identificar si existen similitudes y alguna imagen lo suficientemente parecida a la principal como para determinar que se trata de esta.

Para llevar a cabo este proceso se va a emplear la conocida librería de visión artificial OpenCV, que ha sido utilizada en proyecto de imagen importantes, como el del vehículo no tripulado Stanley de la Universidad de Stamford.

3.0.7. Servidor

El servidor será el encargado de realizar el reconocimiento de imágenes y almacenar la base de datos. Además, ofrecerá una API RESTful para que desde la aplicación móvil se puedan realizar peticiones, para realizar el procedimiento de reconocer una imagen o para consultar información de la base de datos.

Además, se ha mejorado la velocidad del reconocimiento, utilizando hilos, para que los procesos de detección y comparación de imágenes se realicen en parelelo. En su totalidad se ha implementado con el leguaje Python, tanto el reconocimiento de imágenes como la conexión con la base e datos.

Capítulo 4

Planificación del proyecto

En el siguiente apartado se presenta en primer lugar la metodología utilizada para el desarrollo de proyecto, justificando su uso y comparándola con otras. Además, se mostrará una planificación de las tareas a realizar durante el proyecto, que serán analizadas para poder presentar una estimación sobre su duración. También se expondrán los recursos necesarios para su elaboración. Y, por último, como se ha llevado a cabo el seguimiento del proyecto.

4.1. Metodología

Como ya se ha mencionado anteriormente, uno de los puntos a tener en cuenta de este proyecto es que su realización se lleva a cabo de manera conjunta. Teniendo en cuenta este aspecto, es muy importante la elección de una buena metodología de desarrollo con tal de que los integrantes del equipo sepan en todo momento qué deben hacer y cómo avanza este .

Antiguamente las empresas que se dedicaban a la construcción de software, también llamadas fábricas de software, utilizaban metodologías para la gestión de sus proyectos, como si se tratase de un proyecto arquitectónico. Las mismas metodologías tradicionales que hicieron fracasar grandes proyectos, estudiados en la asignatura Gestión de Proyecto de Ingeniería Software (EI1040), como el Superconducting Super Collider [11], o que hicieron sobrepasar considerablemente el presupuesto establecido como el Eurotúnel [3].

Este desarrollo tradicional puede ser efectivo en proyectos de larga duración en los que se disponga de un presupuesto elevado y grandes recursos. Los proyectos de desarrollo software de las últimas décadas predominan por ser desarrollados en entornos cambiantes y con escaso tiempo, para los cuales este enfoque tradicional no es el más adecuado. Es por este motivo que en los últimos años han emergido con gran fuerza las metodologías ágiles, las cuales se adaptan en cuanto a flexibilidad y tiempo a estos proyectos [7].

Desde el principio del proyecto, tanto por parte de la empresa como por la de los alumnos hubo una clara tendencia hacia el uso de una metodología ágil, ya que estas están siendo las más utilizadas por las grandes empresas del sector en los últimos años.

Existen muchas variedades de metodologías ágiles, pero una de las características de este tipo de desarrollo es que no se tiene por qué realizar una elección y ceñirse a las pautas marcadas, sino que ofrecen una gran libertad a la hora de utilizar sus principios. Para una empresa la mejor metodología ágil es la creada por ellos mismos basándose en las existentes y adaptándola a sus necesidades. Tal es el caso, que dicha empresa ya opera de esta manera, por lo tanto, tiene un modo de trabajo estipulado. La metodología ágil adoptada por la empresa parte de *SCRUM*, aunque con una adaptación a la manera de trabajar de esta. Además, también utilizan técnicas de otras metodologías, como Extreme Programming. En este caso, la técnica que predomina es la de Pair Programming, o como su propio nombre indica programación por parejas. Consiste en que dos desarrolladores actúen desde un mismo ordenador, uno toma el mando del ordenador (controlador) y el otro le da apoyo ayudándole en las tareas realizadas (navegador), y cada cierto tiempo se cambian los roles. De esta manera se consigue un mejor producto además de un mejor ambiente de trabajo [8].

En este proyecto se hará uso de la metodología *SCRUM*, pero también de técnicas utilizadas en eXtrem Programming. Todo esto siguiendo un método de desarrollo de productos, seguido por la empresa y llamado *Lean startup*, el cual consiste en una filosofia de desarrollo cíclico. En cada uno de los ciclos realizados, se debe obtener un producto mínimo viable y proporcionar *feedback* al cliente [34].

4.1.1. SCRUM

Scrum es un marco de trabajo mediante el cual las personas pueden hacer frente a problemas complejos adaptativos, al mismo tiempo que conseguir entregar productos del valor máximo posible, de una manera productiva y adaptativa.

Este se lleva utilizando para el desarrollo de productos complejos desde los años 90, aunque es en la última década cuando más protagonismo ha adquirido. Scrum no es un técnica o un proceso, sino que es un marco de trabajo dentro del cual se pueden emplear varias técnicas y procesos [22].

Este marco de trabajo se basa en la realización de iteraciones cortas, denominadas sprints, que suelen durar entre una y cuatro semanas, en las cuales se obtiene un producto "acabado", es decir, que se pueda entregar al cliente. Por lo tanto, al finalizar cada sprint el cliente comprueba cómo avanza su producto. Scrum tiene tres características importantes: los roles, los artefactos y las reuniones.

En primer lugar, en un equipo de desarrollo que utilice Scrum, se deben abordar los siguientes roles:

- El Dueño del Producto (Product Owner)
- Scrum Master
- Equipo de Desarrollo (Development Team)
- Clientes o Usuarios

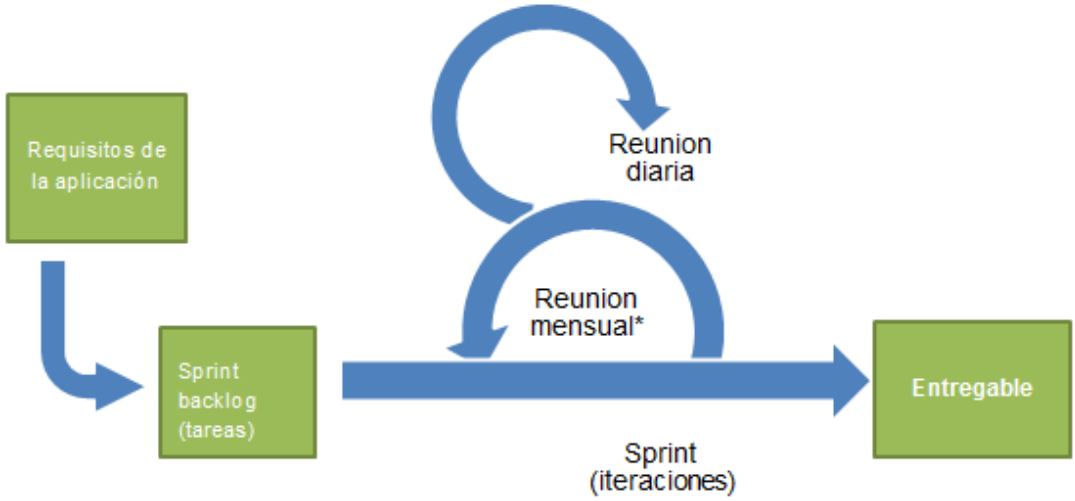


Figura 4.1: Marco de trabajo SCRUM.

El dueño del producto es el responsable de obtener el mayor valor del producto y del trabajo del equipo de desarrollo. Este es el encargado de gestionar la pila del producto y por tanto debe ordenarla y asegurarse que esta es entendible por el equipo de desarrollo.

El Scrum Master es el responsable de que la metodología se esté aplicando de manera correcta. Este adopta el papel de líder del Equipo de Desarrollo, además, es el encargado de transmitir a las personas externas los avances del proyecto y las acciones que puedan ser de ayuda o no.

Por otra parte, el Equipo de Desarrollo es el grupo de personas responsable de implementar las funcionalidad indicadas en la pila del producto. Los usuario o clientes, son las personas que se beneficiarán con el resultado final del producto y, por tanto, deben seguir el proceso y aportar las ideas que sean necesarias.

En segundo lugar, el Scrum también se caracteriza por los artefactos, que favorecen la transparencia de información, para que todos entiendan de la misma forma el contenido del proyecto. Estos son los siguientes:

- Pila del Producto (Product Backlog)
- Pila del Sprint (Sprint Backlog)

La Pila del Producto es una lista de tareas, en forma de historias de usuario, la cual abarca todas las funcionalidades o requerimientos que se deben realizar durante el proyecto. Esta debe ser realizada al inicio del proyecto por el Dueño del Producto, con ayuda del Equipo de Desarrollo.

La Pila del Sprint corresponde a la lista de tareas específica de cada sprint. Esta se debe

construir al inicio de cada sprint con las tareas de la Pila del Producto que se quieran realizar durante este .

Por último, otra característica importante en Scrum son los diferentes tipos de reuniones que se deben celebrar durante el proyecto:

- Reunión de Planificación del Sprint (Sprint Planning Meeting)
- Scrum Diario (Daily Scrum)
- Revisión del Sprint (Sprint Review)
- Retrospectiva del Sprint (Sprint Retrospective)

La reunión de Planificación del Sprint se debe realizar antes de cada sprint. Su objetivo es el de planificar el sprint, por tanto, se creará la Pila del Sprint para este y el Sprint Goal, un documento con una descripción de lo que se intentará alcanzar durante el sprint. En esta reunión deben participar prácticamente todos los integrantes del proyecto, exactamente, el Dueño del Producto, el Scrum Master y el Equipo de Desarrollo.

Durante el proyecto se deben realizar pequeñas reuniones diarias, llamas Scrum Diario, mediante las cuales se comenta entre los integrantes del Equipo de Desarrollo y el Scrum Master las tareas que se han hecho, las que se van a hacer y los obstáculos a dificultades que se puedan tener.

Otra reunión característica de Scrum es la revisión del Sprint. Esta se realiza al final de cada sprint y en ella el equipo de desarrollo muestra los avances realizados durante el sprint.

Por último, una vez se ha finalizado y presentado el resultado del sprint, se celebra una reunión llamada Retrospectiva del Sprint, en la cual el Dueño del producto revisa con el Equipo de Desarrollo los objetivos completados durante el sprint, y se analizan los aspectos positivos y negativos de este .

4.2. Definición de las tareas

Siguiendo el objetivo de mantener una experiencia real con la metodología *SCRUM*, para la definición de las tareas se realizó una reunión con los integrantes de la empresa, los cuales tomaron el papel de clientes. En dicha reunión se habló sobre todo lo que los clientes deseaban que la aplicación pudiera realizar, preguntándoles en todo momento las dudas que iban surgiendo y mostrándoles apoyo en las que les surgían a ellos.

Como resultado de este encuentro se obtuvieron una serie de requisitos que debía cumplir la aplicación, que fueron analizados y transformados en las historias de usuario que se presentan en la tabla 4.1.

Las historias de usuario son una representación de los requisitos de software breve y concisa, que aporten un valor al cliente, ya que este debe aprobar las historias de usuario y ayudar

a priorizarlas antes de empezar los *sprints*. Las historias de usuario suelen tener la siguiente estructura:

$$Como < rol > yo\, quiero < deseo > para < beneficio > \tag{4.1}$$

Con el objetivo de simplificar la longitud de estas, para facilitar la lectura posteriormente en la herramienta utilizada para el seguimiento del proyecto, siguiendo la recomendación de los integrantes de la empresa, se ha optado por utilizar la ulterior redacción :

$$< rol >< deseo > \tag{4.2}$$

Además, se ha reducido el rol a una sola inicial, siendo las siguientes:

- D: Desarrollador (en este caso el alumno).
- U: Usuario de la aplicación.
- A: Administrador del sistema.
- C: Cliente.
- E: Estudiante.

Id.	Historia
HU01	[D] Familiarizarse con el entorno de trabajo.
HU02	[D] Conceptos de programación móvil.
HU03	[D] Investigación tecnologías servidor.
HU04	[D] Investigación algoritmos reconocimiento de imágenes.
HU05	[U] Realizar una fotografía con mi dispositivo móvil.
HU06	[U] Enviar la imagen al servidor.
HU07	[U] Recibir la imagen en el servidor.
HU08	[D] Cotejar imagen con la base de datos.
HU09	[U] Ver tutorial al iniciar la aplicación por primera vez, pudiéndolo omitir.
HU10	[U] Acceder al tutorial en cualquier momento.
HU11	[U] Ver el contenido multimedia sin salir de la aplicación.
HU12	[U] En caso de error al enviar una imagen, se me permite reintentar o volver a empezar.
HU13	[U] Compartir el resultado de la búsqueda en Facebook y Twitter.
HU14	[U] Asignar una puntuación a los resultados obtenidos.
HU15	[U] Marcar como favorito un resultado obtenido.
HU16	[U] Acceder a mis favoritos.
HU17	[U] Apartado de “acerca de” donde aparezcan datos sobre la empresa.
HU18	[U] Apartado de licencias se especifiquen las licencias utilizadas para el desarrollo de la aplicación.
HU19	[U] Navegar por la aplicación mediante un “action bar”.
HU20	[A] Añadir imágenes con un contenido asociado al sistema.
HU21	[A] Añadir campañas al sistema a las que asignar imágenes.
HU22	[C] Almacenar las fotos realizadas, la imagen a la que corresponden y desde que móvil se han enviado.
HU23	[C] Integrar la aplicación con Google Analytics.
HU24	[E] Redacción del manual de usuario.
HU25	[E] Redacción de la Memoria.
HU26	[E] Preparación de la presentación.

Tabla 4.1: Historias de usuario del proyecto.

Una vez redactadas las historias de usuario, para facilitar la posterior estimación de estas, se dividieron en tareas (Tabla 4.2), especificando aquí todas las labores que se deben realizar para todo el proyecto de la empresa; las que pertenecen al documentado en este documento y

las que pertenecen al del otro alumno, junto con quien se ha elaborado parte de este .

Ident.	Historia	Tarea
TA01	**HU01**	Aprender funcionamiento del Confluence y Jira.
TA03	**HU01**	Aprender funcionamiento Android Studio.
TA05	**HU02**	Estudio fundamentos Android.
-	**HU03**	No es necesario fragmentar esta historia de usuario.
-	**HU04**	No es necesario fragmentar esta historia de usuario.
TA09	**HU05**	Implementar cámara Android.
TA11	**HU06**	Realizar petición Android.
TA12	**HU07**	Implementar servicio REST para recibir una imagen.
TA13	**HU08**	Construir base de datos.
TA14	**HU08**	Implementar algoritmo para obtener los descriptores de la imagen.
TA15	**HU08**	Implementar algoritmo para matchear los descriptores contra la base de datos.
-	**HU09**	No es necesario fragmentar esta historia de usuario.
-	**HU10**	No es necesario fragmentar esta historia de usuario.
TA19	**HU11**	Implementar reproductor de video Android.
TA21	**HU11**	Implementar reproductor de audio Android.
TA23	**HU11**	Implementar galería Android.
-	**HU12**	No es necesario fragmentar esta historia de usuario.
TA26	**HU13**	Implementar función compartir Facebook Android.
TA28	**HU13**	Implementar función compartir Twitter Android.
TA29	**HU14**	Implementar servicio puntuación y almacenar en la base de datos.
TA31	**HU15**	Crear base de datos interna Android.
TA33	**HU15**	Añadir funcionalidad favoritos a la interfaz iOS.
-	**HU16**	No es necesario fragmentar esta historia de usuario.
-	**HU17**	No es necesario fragmentar esta historia de usuario.
-	**HU18**	No es necesario fragmentar esta historia de usuario.
-	**HU19**	No es necesario fragmentar esta historia de usuario.
-	**HU20**	No es necesario fragmentar esta historia de usuario.
-	**HU21**	No es necesario fragmentar esta historia de usuario.
-	**HU22**	No es necesario fragmentar esta historia de usuario.
-	**HU23**	No es necesario fragmentar esta historia de usuario.

Tabla 4.2: División en tareas de las historias de usuario del proyecto.

4.3. Planificación temporal de las tareas

A pesar de que el desarrollo del proyecto se realice siguiendo una metodología ágil, para la redacción de la propuesta técnica inicial se realizó un diagrama de Gantt con lo que más o menos se pensaba que sería el desarrollo del proyecto, diferenciando algunas de las tareas que por parte de la empresa se habían propuesto al inicio del proyecto. En la Figura 4.2 se puede ver las tareas que se pensaron en ese momento y en la Figura 4.3 el resultado del diagrama de Gantt con ellas. De todas maneras cabe destacar que este diagrama no es representativo en cuanto a lo que será el proyecto, ya que en *SCRUM* no se puede realizar diagramas de este tiempo donde se planifique todas las tareas que se realizarán a lo largo del proyecto, sino que se va decidiendo antes de cada *sprint*.

Nombre de tarea	Duración	Comienzo	Fin
Proyecto App Reconocimiento de imágenes	**100 días**	**lun 03/03/14**	**vie 18/07/14**
1º Sprint	**10 días**	**lun 03/03/14**	**vie 14/03/14**
Estudio previo	**10 días**	**lun 03/03/14**	**vie 14/03/14**
Conceptos de programación en iOS	10 días	lun 03/03/14	vie 14/03/14
Conceptos de programación en Android	10 días	lun 03/03/14	vie 14/03/14
Comparación algoritmos reconocimiento de imágenes	10 días	lun 03/03/14	vie 14/03/14
Estudio de tecnologias para implementar servicios REST	10 días	lun 03/03/14	vie 14/03/14
Definición de la pila de producto	3 días	mié 12/03/14	vie 14/03/14
Estimación de la pila de producto	3 días	mié 12/03/14	vie 14/03/14
Entrega propuesta técnica borrador	0 días	vie 14/03/14	vie 14/03/14
2º Sprint	**15 días**	**lun 17/03/14**	**vie 04/04/14**
Desarrollo cliente beta (Android)	8 días	lun 17/03/14	mié 26/03/14
Desarrollo cliente beta (iOS)	8 días	lun 17/03/14	mié 26/03/14
Primera versión del servicio REST	7 días	jue 27/03/14	vie 04/04/14
Entrega propuesta técnica definitiva	0 días	vie 04/04/14	vie 04/04/14
Pruebas de aceptación con el cliente	0 días	vie 04/04/14	vie 04/04/14
3º Sprint	**15 días**	**lun 07/04/14**	**vie 25/04/14**
Comunicación cliente Android-servidor	15 días	lun 07/04/14	vie 25/04/14
Comunicación cliente iOS-servidor	15 días	lun 07/04/14	vie 25/04/14
Implementación base de datos	7 días	lun 07/04/14	mar 15/04/14
Rectificar indicaciones cliente anterior Sprint	4 días	lun 07/04/14	jue 10/04/14
Pruebas de aceptación con el cliente	0 días	vie 25/04/14	vie 25/04/14
4º Sprint	**15 días**	**lun 28/04/14**	**vie 16/05/14**
Algoritmo de reconocimiento de imágenes	15 días	lun 28/04/14	vie 16/05/14
Rectificar indicaciones cliente anterior Sprint	4 días	lun 28/04/14	jue 01/05/14
Pruebas de aceptación con el cliente	0 días	vie 16/05/14	vie 16/05/14
5º Sprint	**15 días**	**lun 19/05/14**	**vie 06/06/14**
Integración del sistema	15 días	lun 19/05/14	vie 06/06/14
Rectificar indicaciones cliente anterior Sprint	4 días	lun 19/05/14	jue 22/05/14
Pruebas de aceptación con el cliente	0 días	vie 06/06/14	vie 06/06/14
6º Sprint	**15 días**	**vie 06/06/14**	**vie 27/06/14**
Rectificar indicaciones cliente anterior Sprint	8 días	lun 09/06/14	mié 18/06/14
Inicialización del servidor y base de datos	8 días	lun 09/06/14	mié 18/06/14
Testeo	7 días	jue 19/06/14	vie 27/06/14
Pruebas de aceptación con el cliente	0 días	vie 06/06/14	vie 06/06/14
7º Sprint	**15 días**	**lun 30/06/14**	**vie 18/07/14**
Redacción del manual de usuario	2 días	lun 30/06/14	mar 01/07/14
Redacción del trabajo final de grado	12 días	lun 30/06/14	mar 15/07/14
Preparación de la presentación	3 días	mié 16/07/14	vie 18/07/14

Figura 4.2: Tareas para el diagrama de Gantt.

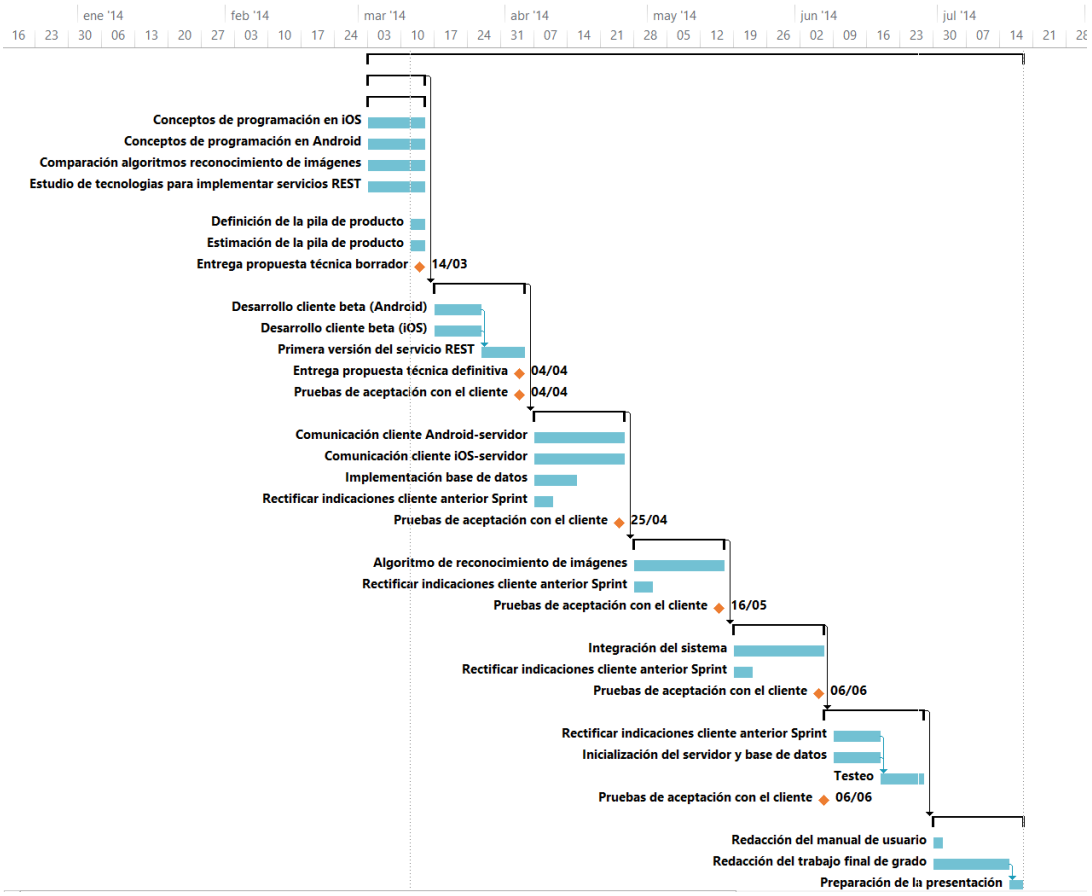


Figura 4.3: Diagrama de Gantt del proyecto.

Para llevar a cabo una planificación eficaz utilizando una metodología ágil, una vez redactadas las historias de usuario, debemos estimarlas. Antes de realizar la estimación y empezar con las interacciones debemos definir tres conceptos clave en *SCRUM*:

- La duración de los *sprints*.
- Los puntos de historia.
- La velocidad del equipo.

En primer lugar, para la duración de los *sprints* la metodología elegida recomienda realizarlos de entre 15 a 30 días. Por parte de la empresa se recomendó realizar los *sprints* de tres semanas (15 días laborables aproximadamente), para poder presentar el resultado del proyecto más a menudo y de esta manera ayudarnos más en la aplicación de la metodología. En segunda instancia, como ya se ha explicado en el apartado correspondiente, la estimación se realizó mediante puntos de historia para los cuales determinamos, siguiendo la recomendación realizada por parte de la empresa, que un punto de historia sería igual a dos horas de trabajo. La jornada laboral acordada con la empresa fue de cuatro horas diarias, por lo que un día de trabajo equivale a dos puntos de historia.

El último concepto a tener en consideración es la velocidad, la cual en *SCRUM* se puede calcular mediante la siguiente fórmula:

$$Velocidad = \frac{Trabajo}{Tiempo} \tag{4.3}$$

Como existen festivos y no todos los *sprints* tienen la misma duración vamos a establecer la velocidad ideal, esto es, la máxima velocidad. Teniendo en cuenta que la duración ideal de un *sprint* es de 15 días, lo que corresponde a 30 puntos de historia, podemos establecer que el equipo de desarrollo podrá desarrollar historias de usuario como máximo de 30 puntos de historia. Esta velocidad deberá ser replanteada en cada *sprint* dependiendo de la duración de cada uno y pudiendo observar las estadísticas de los otros.

La estimación es una de las fases más difíciles de la ingeniería software, ya que no es una labor sencilla determinar el tiempo que va a costar una tarea determinada, y es por eso que las metodologías ágiles proponen el uso de estos puntos de historia. Para realizar dicha tarea de la forma más eficaz posible se utilizó la adaptación de una técnica, propuesta por Extreme Programming, denominada Planning Poker. Esta técnica consiste en que cada miembro del equipo de desarrollo posee una baraja de cartas que contiene números siguiendo la secuencia de Fibonacci. Para cada historia de usuario los componentes del equipo deben elegir la carta con puntos de historia que creen que costará la historia. Si todos eligen el mismo número, este será el elegido, si no es así, se deberá llegar a un consenso explicando cado uno sus argumentos y repitiendo el proceso hasta que todos elijan el mismo número [26]. Está técnica ya era conocida por su aplicación en algunas asignaturas estudiadas a lo largo de la carrera, como Paradigmas de Software y Métodos Ágiles, incluso se disponía de una baraja de cartas para su aplicación.

EL resultado de esta fase de estimación se puede ver en la 4.3.

Id.	Historia	Puntos
HU01	[D] Familiarizarse con el entorno de trabajo.	8
HU02	[D] Conceptos de programación móvil.	8
HU03	[D] Investigación tecnologías servidor.	4
HU04	[D] Investigación algoritmos reconocimiento de imágenes.	8
HU05	[U] Realizar una fotografía con mi dispositivo móvil.	10
HU06	[U] Enviar la imagen al servidor.	5
HU07	[U] Recibir la imagen en el servidor.	5
HU08	[D] Cotejar imagen con la base de datos.	40
HU09	[U] Ver tutorial al iniciar la aplicación por primera vez, pudiéndolo omitir.	5
HU10	[U] Acceder al tutorial en cualquier momento.	1
HU11	[U] Ver el contenido multimedia sin salir de la aplicación.	16
HU12	[U] En caso de error al enviar una imagen, se me permite reintentar o volver a empezar.	12
HU13	[U] Compartir el resultado de la búsqueda en Facebook y Twitter.	8
HU14	[U] Asignar una puntuación a los resultados obtenidos.	8
HU15	[U] Marcar como favorito un resultado obtenido.	4
HU16	[U] Acceder a mis favoritos.	2
HU17	[U] Apartado de “acerca de” donde aparezcan datos sobre la empresa.	2
HU18	[U] Apartado de licencias se especifiquen las licencias utilizadas para el desarrollo de la aplicación.	2
HU19	[U] Navegar por la aplicación mediante un “action bar”.	4
HU20	[A] Añadir imágenes con un contenido asociado al sistema.	8
HU21	[A] Añadir campañas al sistema a las que asignar imágenes.	2
HU22	[C] Almacenar las fotos realizadas, la imagen a la que corresponden y desde que móvil se han enviado.	2
HU23	[C] Integrar la aplicación con Google Analytics.	2
HU24	[E] Redacción del manual de usuario.	4
HU25	[E] Redacción de la Memoria.	40
HU26	[E] Preparación de la presentación.	6

Tabla 4.3: Estimación de las historias de usuario.

Después de estimar todas las historias de usuario, llega el momento de empezar con los *sprints*.

Para realizar la planificación hay que tener en cuenta los festivos y los días no laborables que tendrán lugar durante la estancia. Los festivos en Castellón en los cuatro meses en la empresa son los siguientes:

- 19 de Marzo: San José.
- 24 de Marzo, Fiestas de la Magdalena.
- 28 de Marzo: Fiesta local.
- 18 de Abril: Viernes Santo.
- 21 de Abril: lunes de Pascua.
- 1 de Mayo: Fiesta del Trabajo.

Como la duración de la estancia en prácticas está estipulada en cuatro meses, exactamente 79 días descontando los días no laborables, se ha calculado que se realizarán seis *sprints* durante su duración, más un *sprint* una vez acabada la estancia, el cual se dedicará totalmente a la redacción del trabajo final de grado junto con la elaboración de la presentación. Esto hace un total de siete *sprints*, la fecha de los cuales se puede ver en la tabla 4.4.

Sprint	**Fecha**
1	Del 3 de Marzo al 14 de Marzo (10 días).
2	Del 14 de Marzo al 17 de Abril (12 días).
3	Del 7 de Abril al 28 de Abril (13 días).
4	Del 28 de Abril al 19 de Mayo (14 días).
5	Del 19 de Mayo al 9 de Junio (15 días).
6	Del 9 de Junio al 30 de Junio (15 días).
7	Del 30 de Junio al 14 de Julio (15 días).

Tabla 4.4: Duración de los *sprints* del proyecto.

4.4. Estimación de Recursos del Proyecto

En este apartado del proyecto se explicarán los diferentes recursos que han sido necesarios para la elaboración del proyecto, ya sean recursos materiales, como ordenadores o dispositivos, o los recursos de software.

4.4.1. Recursos Físicos

Ordenador portátil: desde el que se realiza una gran parte del proyecto. Las características del mismo son:

- Procesador Intel Core 2 Duo P7350 a 2.00 GHz
- 4 GB de memoria RAM
- Disco Duro de 250 GB.
- Tarjeta Gráfica NVIDIA GeForce 9600M GT de 512MB de VRAM.

Primer dispositivo móvil de la empresa: indispensable para poder probar la aplicación y todas sus partes. Sus características son:

- Marca y modelo: Samsung Galaxy Young
- Procesador Single-Core Qualcom MSM7227A Snapdragon 1GHz Cortex-A5
- Gráfica Adreno 200
- Pantalla TFT de 3.27 pulgadas (320 x 480 pixeles) 176 ppi
- Cámara de 3 MegaPixels
- Android 4.1 Jelly Bean

Segundo dispositivo móvil de la empresa: será necesario para poder realizar las pruebas con la última versión de Android. Sus características son:

- Marca y modelo: Google Nexus 5
- Procesador Quad-Core Qualcomm Snapdragon 800 2.26GHz
- Gráfica Adreno 330
- Pantalla LCD de 4.95 pulgadas (1080x1920 pixeles) 445 ppi
- Cámara de 8 MP, 3264 x 2448 píxeles
- Android 4.4 Kitkat

Dispositivo móvil del alumno: un dispositivo más para realizar pruebas. Sus características son:

- Marca y modelo: ZTE Grand X Quad V987
- Procesador Quad-Core Mediatek MT6589 1.2 GHz Cortex-A7

- Gráfica PoweeVR SGX544
- Pantalla TFT de 5 pulgadas (720 x 1280 píxeles) 294 ppi
- Cámara 8 MP, 3264 x 2448 píxeles
- Android 4.2.1 Jelly Bean

Servidor: necesario para ofrecer una API en tiempo real a la que se conecten las aplicaciones desde los dispositivos móviles. Además, es el encargado de realizar el reconocimiento de imágenes y almacenar la base de datos. Sus potentes características son las siguientes:

- Procesador Intel Xeon E3 1245v2 de 4 Cores/8 Threads (Hyperthreading) a 3.4 GHz+
- 32 GB de memória RAM
- Sistema Operativo Linux Gentoo

4.4.2. Recursos de Software

Para la elaboración del proyecto se han utilizado una serie de programas y herramientas necesarias para las distintas partes de este. Estos son los siguientes:

Sistema Operativo Windows 8.1: Sistema Operativo instalado en el ordenador portátil.

Android Studio: Programa utilizado para la construcción de la aplicación móvil.

Android SDK: Kit de desarrollo Android, necesario para programar para Android.

Java Development Kit 7: Kit de desarrollo Java, necesario para programar para Android.

Mozilla Firefox: Navegador de Internet, destinado para la búsqueda de información.

Google Drive: Herramienta de ofimática online, utilizada para la documentación común con el equipo de desarrollo.

Atlassian JIRA: Herramienta para la gestión de incidencias, utilizada para el seguimiento del proyecto.

Atlassian Confluence: Herramienta de tipo wiki empleada para documentación de la empresa.

Pencil: Es una herramienta de prototipado utilizada para la elaboración de mockups.

Inkscape: Es una herramienta de edición de gráficos vectoriales, libre y multiplataforma, utilizada para la elaboración de gráficos y diagramas.

StarUML: Herramienta de modelado UML, con la cual se han realizado los diagramas de clases, de casos de uso y de actividad.

Adobe Photoshop: Herramienta utilizada para la edición de fotografías.

Adobe Illustrator: Editor de gráficos vectoriales, empleado para la elaboración del logo y los botones.

Spyder: Entorno de Desarrollo Integrado para el lenguaje Python, utilizado para la implementación del servidor y reconocimiento de imágenes.

RockMongo: Es un administrador PHP para bases de datos MongoDB, destinado para editar de manera rápida y sencilla la base de datos.

4.5. Seguimiento del Proyecto

En *SCRUM*, se podría apuntar que, la planificación de las tareas que se realizarán en cada *sprint* se va haciendo improvisadamente, sin plan previo, y es el cliente quien ha de decidir con la ayuda del equipo de desarrollo las tareas que se realizarán antes o después. Esto se realiza en la reunión de planificación del *sprint* y, una vez realizado el proyecto, ya se puede mostrar lo elaborado en cada *sprint*. El resultado de lo anterior puede observarse en la tabla 4.5.

Sprint	**Pila del sprint**
1	[D] Familiarizarse con el entorno de trabajo. [D] Conceptos de programación móvil. [D] Investigación tecnologías servidor. [D] Investigación algortimos reconocimiento de imágenes.
2	[U] Realizar una fotografía con mi dispositivo móvil. [U] Enviar la imagen al servidor. [U] Recibir la imagen en el servidor.
3	[U] En caso de error al enviar una imagen, se me permite reintentar o volver a empezar (No finalizada). [D] Cotejar imagen con la base de datos (No finalizada). [U] Navegar por la aplicación mediante un *action bar*. [U] Compartir el resultado de la búsqueda en Facebook y Twitter.
4	[U] En caso de error al enviar una imagen, se me permite reintentar o volver a empezar. [D] Cotejar imagen con la base de datos.
5	[U] Ver el contenido multimedia sin salir de la aplicación. [U] Marcar como favorito un resultado obtenido. [U] Acceder a mis favoritos. [C] Almacenar las fotos realizadas, la imagen a la que corresponden y desde que móvil se han enviado.
6	[U] Apartado de *acerca de* donde aparezcan datos sobre la empresa. [U] Apartado de licencias se especifiquen las licencias utilizadas para el desarrollo de la aplicación. [U] Ver tutorial al iniciar la aplicación por primera vez, pudiéndolo omitir. [U] Acceder al tutorial en cualquier momento.
7	[E] Redacción del manual de usuario. [E] Redacción de la Memoria. [E] Preparación de la presentación.

Tabla 4.5: Distribución de las historias de usuario entre los *sprints* del proyecto.

En los proyectos de desarrollo de software siempre suele haber algún desfase entre la planificación elaborada y el tiempo real final que ha durado la realización de la tarea. Este proyecto no es una excepción, ya que por una parte existe una escasez de experiencia a la hora de planificar y, por otra, la mayoría de las tecnologías utilizadas son nuevas para los desarrolladores, así que no se puede planificar con exactitud el tiempo que se requerirá para la comprensión y aplicación de estas.

Los acontecimientos más destacables de cada *sprint* se muestran a continuación:

- **Sprint 1:** Familiarizarse con el entorno de trabajo no ha sido una tarea compleja, ya que la empresa facilito un puesto de trabajo adecuando y confortable, además de que algunos de los programas utilizados ya habían sido tratados con anterioridad. Adquirir los conceptos necesarios para poder programar aplicaciones para el sistema operativo Android ha sido una tarea más costosa de lo que se pensaba anteriormente, ya que, si bien ya se tenía experiencia en el lenguaje de programación Java, con Android existen multitud de conceptos diferentes.
- **Sprint 2:** Se ha superado la estimación prácticamente en todas las historias de este Sprint. La más costoso fue la implementación de la cámara dentro de la aplicación, ya que en Android hay que tener en cuenta bastantesaspectos para gestionar la cámara de una manera adecuada. Por otra parte, realizar, enviar y recibir la fotografía también sobrepasó la estimación, en este caso debido a la poca experiencia con las tecnologías del servidor.
- **Sprint 3:** En este *sprint* se respetó bastante la estimación inicial, donde lo más costos fue la implementación de la base de datos, ya que no se había tenido ninguna experiencia anterior con la tecnología.
- **Sprint 4:** Se podría decir que la parte de más dificultad para los alumnos fue la de este *sprint*. En él se realizó la parte del reconocimiento de imágenes, aunque la estimación no se vio altamente alterada, ya que ante la dificultad de esta parte, se había hecho una estimación precavida.
- **Sprint 5:** Lo más destacable de este *sprint* fue la implementación de componentes multimedia para su visualización desde dentro de la aplicación, cosa sobre la que se demoró más tiempo del estimado al inicio del proyecto.
- **Sprint 6:** Para las historias planificadas para este *sprint* no se tuvieron dificultades y se respetaron los tiempos estimados. Aun así, al ser este el último de la estancia en prácticas, se tuvieron que realizar los pequeños ajustes para dejarlo todo dispuesto, además de realizar infinitud de pruebas, de entre las que cabe destacar las pruebas con diferentes dispositivos Android con versiones distintas de este, que supusieron números problemas.
- **Sprint 7:** Este último *sprint*, fue dedicado en su totalidad a la elaboración de la documentación para la presentación del TFG. Durante este, se multiplicarón los puntos de historia por día, concretamente a 6 puntos, ya que se trabajo todo el día. Las dificultades más destacables han sido la elaboración de la memoria técnica en LaTeX, ya que nunca se había utilizado esta tecnología.

En la tabla 4.6 se puede ver la diferencia entre los tiempos estimados y los tiempos reales (todo en puntos de historia).

Id.	Historia	P. Estimados	P. Real
HU01	[D] Familiarizarse con el entorno de trabajo.	8	6
HU02	[D] Conceptos de programación móvil.	8	7+7
HU03	[D] Investigación tecnologías servidor.	4	2
HU04	[D] Investigación algoritmos reconocimiento de imágenes.	8	5+5
HU05	[U] Realizar una fotografía con mi dispositivo móvil.	10	12
HU06	[U] Enviar la imagen al servidor.	5	6
HU07	[U] Recibir la imagen en el servidor.	5	6
HU08	[D] Cotejar imagen con la base de datos.	40	14+26
HU09	[U] Ver tutorial al iniciar la aplicación por primera vez, pudiéndolo omitir.	5	6
HU10	[U] Acceder al tutorial en cualquier momento.	1	1
HU11	[U] Ver el contenido multimedia sin salir de la aplicación.	16	18
HU12	[U] En caso de error al enviar una imagen, se me permite reintentar o volver a empezar.	12	6+2
HU13	[U] Compartir el resultado de la búsqueda en Facebook y Twitter.	8	2
HU14	[U] Asignar una puntuación a los resultados obtenidos.	8	-
HU15	[U] Marcar como favorito un resultado obtenido.	4	5
HU16	[U] Acceder a mis favoritos.	2	2
HU17	[U] Apartado de “acerca de” donde aparezcan datos sobre la empresa.	2	2
HU18	[U] Apartado de licencias se especifiquen las licencias utilizadas para el desarrollo de la aplicación.	2	1
HU19	[U] Navegar por la aplicación mediante un “action bar“.	4	4
HU20	[A] Añadir imágenes con un contenido asociado al sistema.	8	-
HU21	[A] Añadir campañas al sistema a las que asignar imágenes.	2	-
HU22	[C] Almacenar las fotos realizadas, la imagen a la que corresponden y desde que móvil se han enviado.	2	4
HU23	[C] Integrar la aplicación con Google Analytics.	2	-
HU24	[E] Redacción del manual de usuario.	4	-
HU25	[E] Redacción de la Memoria.	40	66
HU26	[E] Preparación de la presentación.	6	12

Tabla 4.6: Comparación del tiempo real y estimado de las historias de usuario.

Para el seguimiento del proyecto es muy conveniente hacer uso de alguna herramienta, de manera que se pueda saber en todo momento el estado del proyecto. Debido a que la empresa dispone de una licencia para el uso de una de las herramientas más usadas actualmente para el seguimiento de la metodología SCRUM, en ningún momento se dudó de su uso para este proyecto. La herramienta en cuestión se llama JIRA y es propiedad del gigante informático Atlassian.

JIRA es una aplicación web enfocada para el seguimiento de incidencias en proyectos. En ella se pueden insertar Errores, Funciones, Historias y Tareas. Las características más interesantes de la herramienta que se han aplicado durante el desarrollo del proyecto son:

- Gestión de historias de usuario (*backlog*) y organización en *sprints*.
- Reportes con estadísticas de los *Sprints*.
- Tablero de tareas.
- Feedback entre los integrantes del grupo.

Una de las funciones más importantes y, por tanto, imprescindibles en una herramienta para el seguimiento de SCRUM, es la gestión de la pila de producto, la cual podemos ver en la Figura **??**. JIRA permite insertar historias de usuario e ir organizándolas en los *sprints* a medida que se vaya requiriendo. Además, la gestión de las historias se realiza de una manera muy sencilla e intuitiva, simplemente podemos arrastrarlas entre los diferentes *sprints*.

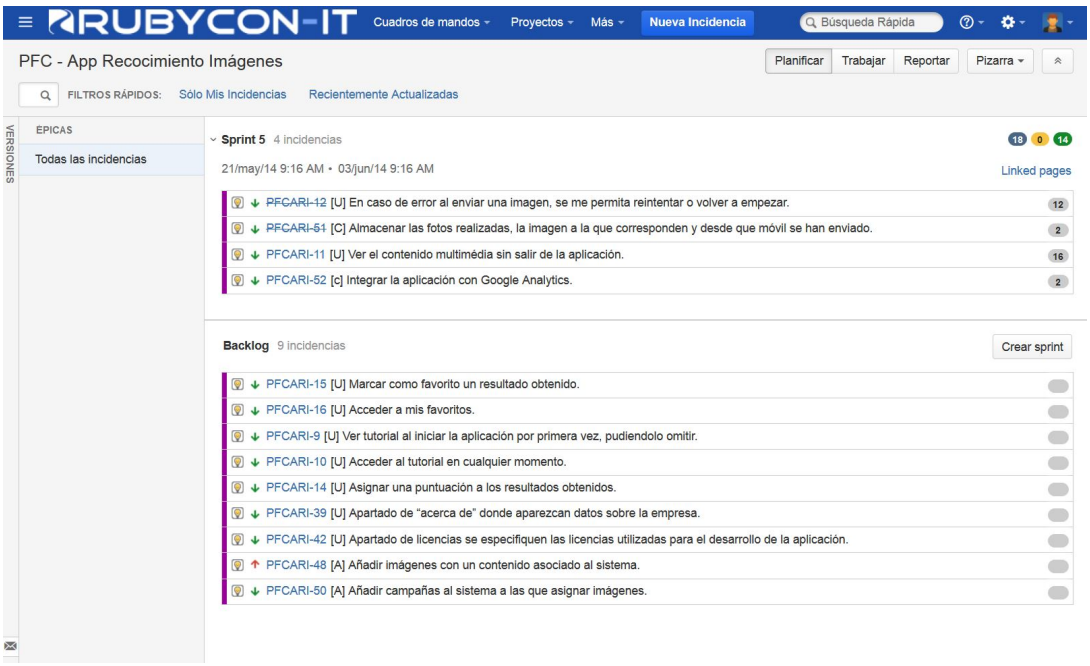
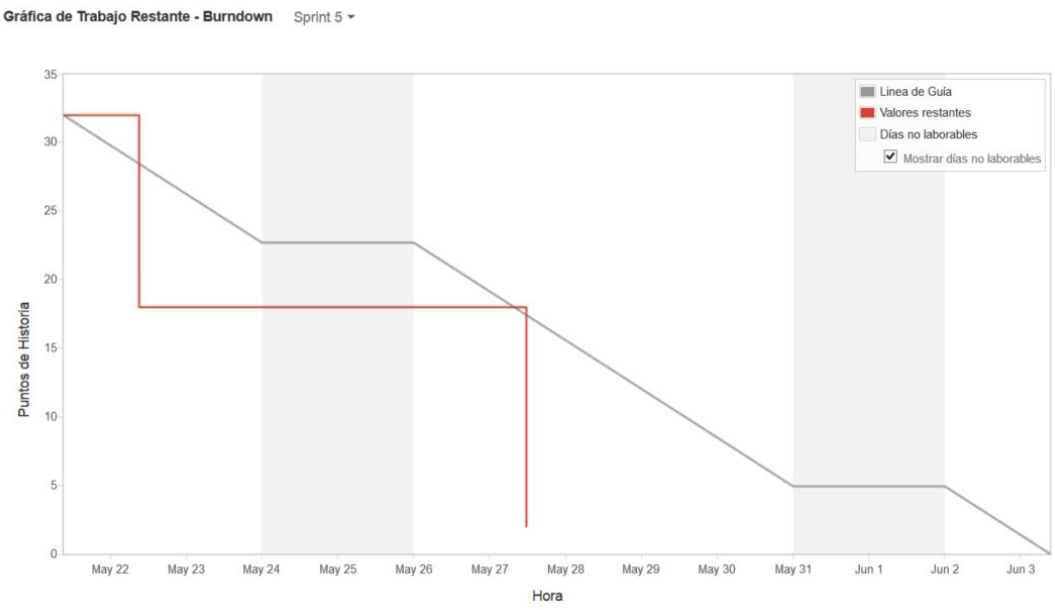


Figura 4.4: Pila del producto en la herramienta Jira.

Otra función de las que dispone la herramienta y que resultaría muy difícil de realizar sin ella, es la visualización de estadísticas al término de los *sprints*. Como se puede ver en la Figura

4.5, al terminar un *sprint* se presenta un resumen con las historias y tareas finalizadas, además de un gráfico que muestra los tiempos estimados frente a los reales.



						Puntos de Historia
Fecha	**Incidencia**	**Tipo de Evento**	**Detalle de Evento**	**Incr.**	**Decr.**	**Tiempo Restante Estimado**
21/05/2014 09:16	PFCARI-11 PFCARI-12 PFCARI-51 PFCARI-52	Comienzo de Sprint		16 12 2 2		32
22/05/2014 09:09	PFCARI-12	Trabajo Restante	Incidencia completada		12	20
22/05/2014 09:10	PFCARI-51	Trabajo Restante	Incidencia completada		2	18
27/05/2014 11:38	PFCARI-11	Trabajo Restante	Incidencia completada		16	2

Figura 4.5: Reporte de un *sprint* en la herramienta Jira.

Al mismo tiempo, esta herramienta también ofrece una tablero estilo *Kanban*, desde el cual los componentes del equipo de desarrollo pueden visualizar es estado del *sprint* en todo momento. Asimismo, desde este también se pueden mover las tareas entre las diferentes columnas a medida estas cambien su estado. En la Figura 4.6 se puede observar el tablero correspondiente al *sprint* número cinco del proyecto.

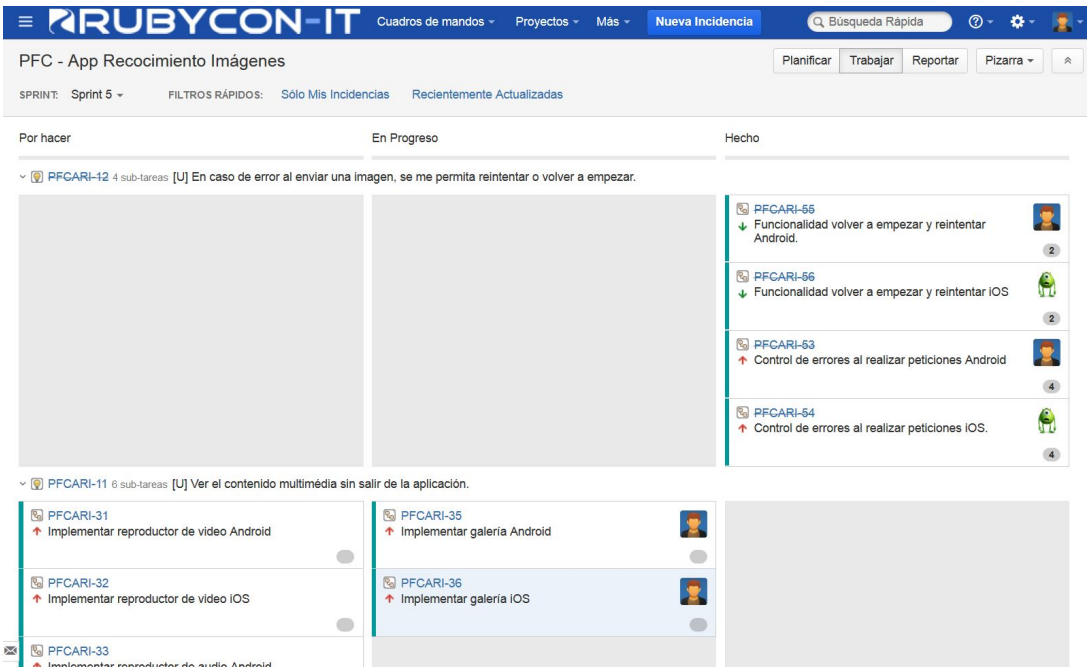


Figura 4.6: Tablero de tareas en la herramienta Jira.

Por último, todas las funciones que ofrece la herramienta, ofrecen el *feedback* necesario al resto del equipo de trabajo para que el resto del equipo de desarrollo pueda estar al corriente de todos los cambios realizados. Además, esto permite a los otros compañeros, avisar al sistema si algo no funciona correctamente, para que sea solucionado lo antes posible. En la Figura 4.7 se puede ver un ejemplo del resumen de actividades que se le muestra al usuario al entrar en la herramienta.

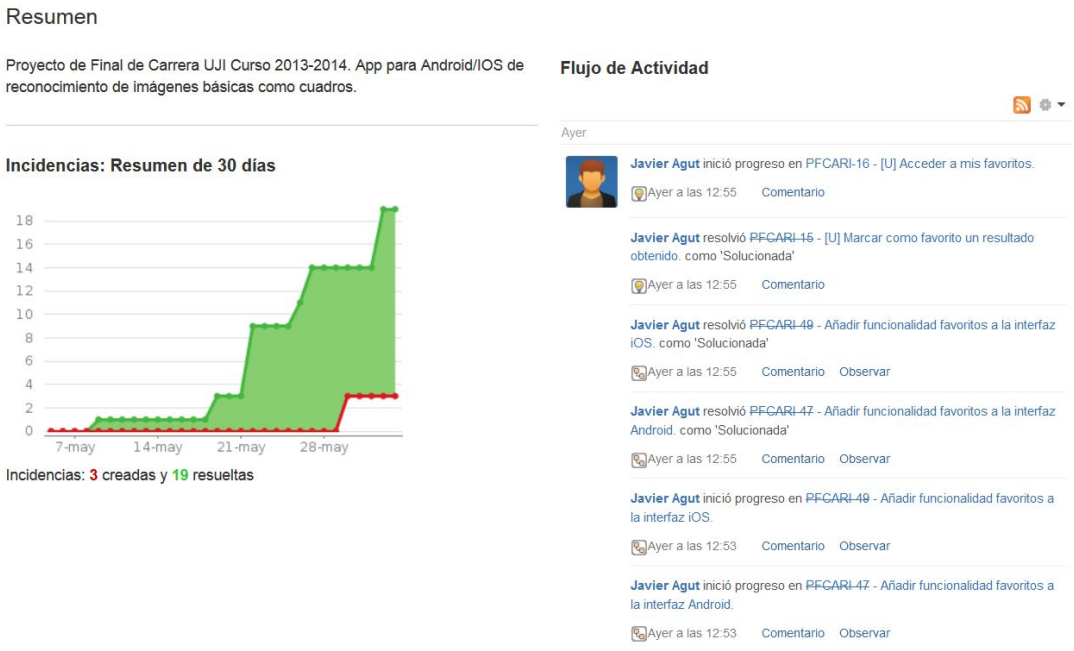


Figura 4.7: Resumen del proyecto en la herramienta Jira.

4.6. Estimación de Costes

La realización de este proyecto conlleva una serie de costes económicos. En este caso los estudiantes que han participado en el la elaboración de este no han sido remunerados, ya que se ha tratado de una estancia en prácticas, pero se supondrán todos los costes del proyecto de no ser así, para conocer el coste que tendría el proyecto en este caso.

El coste de los **recursos materiales** necesarios es el siguiente:

- Ordenador portátil: 900 €.
- Servidor: 1500 €.
- Dispositivos móviles:
  - Nexus 5: 350 €.
  - ZTE v987: 190 €.
  - Samsung Galaxy Young: 69,90 €.
- Internet (29.90 € al mes x 4 meses): 119,6 €.
- Software:
  - Windows 8.1: 119,99 €.
  - Jira (10 € al mes x 4 meses): 40 €.
  - Confluence (10 €al mes x 4 meses): 40 €.

Por otro lado, el coste de los **recursos humanos** necesarios es el siguiente:

- Alumno (294 horas x 25 €/h): 7350 €.

Por lo tanto, el **coste final** del proyecto ascendería a 10679.49 € (9329.49 + 7350). A este coste habría que sumarle los costes del alumno que ha realizado la aplicación para iOS y participado en las partes comunes, pero se han calculado solo los que pertenecen a la aplicación para Android, ya que esta memoria pertenece solo a este proyecto.

Capítulo 5

Análisis

En el siguiente capítulo se mostrará el análisis realizado para el desarrollo del proyecto. En él se recogen los requisitos del sistema, empezando por los requisitos funcionales, seguido de los requisitos de datos y terminando con los no funcionales.

5.1. Visión general de la Aplicación

Antes de elaborar los requisitos del sistema se realizó un diagrama de casos de uso, desde el cual se ofrece una visión de lo que será el sistema (Figura 5.1).

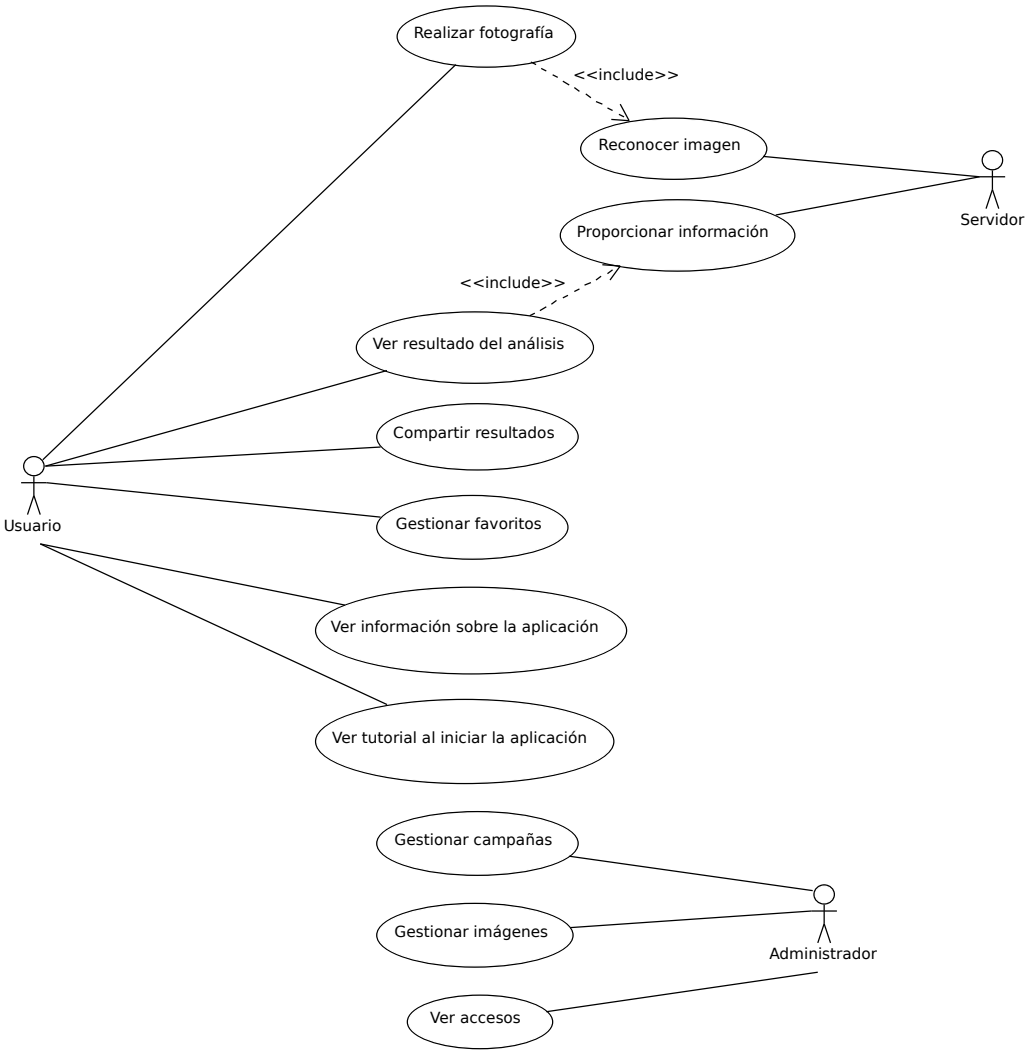


Figura 5.1: Tablero de tareas en la herramienta Jira.

El este se pueden ver tres tipos de actores: el Usuario, el Administrador y el Servidor. El usuario será la persona que utilizará la aplicación, es por ello, que se le adjudican todas las acciones relacionadas con esta. El administrador será el encargado gestionar el sistema, por lo tanto, realizará las acciones relacionados con la gestión de los datos. Por último, al servidor se le otorgan las funciones de reconocimiento de imágenes.

5.2. Análisis de requisitos

Para representar el análisis de los diferentes requisitos se utilizarán una serie de tablas como plantillas. Los requisitos funcionales se muestran siguiendo el modelo de casos de uso que se puede ver en la Tabla 5.1.

Código - Nombre del requisito	
Descripción	Breve descripción sobre el caso de uso.
Relaciones	Lista de casos de uso y requisitos relacionados.
Secuencia de pasos	Acciones.
1	Acción 1
2	Acción 2
3	Acción 3
...	Acción ...
Precondición	Condición necesaria para el cumplimiento del requisito.
Importancia	Alta, media o baja.
Comentarios	Comentarios adicionales sobre el requisito.

Tabla 5.1: Plantilla utilizada para definir los casos de uso.

Para los requisitos de datos y los no funcionales, se utilizarán otro tipos de tablas plantillas más sencillas, las cuales se pueden ver en la Tabla 5.2 y Tabla 5.3.

Identificador - Nombre del requisito	
Descripción	Breve descripción sobre el caso de uso.
Requisitos Asociados	Lista de requisitos relacionados.
Comentarios	Comentarios adicionales sobre el requisito.

Tabla 5.2: Plantilla utilizada para definir los requisitos de datos.

Identificador - Nombre del requisito	
Descripción	Breve descripción sobre el caso de uso.
Comentarios	Comentarios adicionales sobre el requisito.

Tabla 5.3: Plantilla utilizada para definir los requisitos no funcionales.

5.2.1. Requisitos funcionales

Los requisitos funcionales corresponden a los casos de uso, y representan aquellas funciones que debe realizar el sistema. La representación de estos se puede ver en las Tablas 5.4 - 5.14.

CU01 - Realizar fotografía	
Descripción	La aplicación debe permitir al usuario realizar una fotografía con la que empezar el proceso de reconocimiento.
Relaciones	CU02, CU03.
Secuencia de pasos	Acciones.
1	Hacer clic en el botón "Empezar".
2	Realizar la fotografía pulsando en el botón de hacer foto o seleccionar una foto de la galería desde el botón correspondiente.
Precondición	Ninguna.
Importancia	Alta.
Comentarios	Una vez se haya realizado o seleccionado la fotografía, automáticamente se enviará al servidor para empezar el reconocimiento.

Tabla 5.4: Caso de uso 01, Realizar fotografía.

CU02 - Reconocer imagen	
Descripción	El sistema debe, dada una imagen, realizar un proceso de reconocimiento, y devolver el resultado de este .
Relaciones	CU01, CU03.RD01, RD03
Secuencia de pasos	Acciones.
1	Recibir fotografía.
2	Comparar con las de la base de datos.
3	Determinar si está en el sistema.
4	Devolver el resultado.
Precondición	Se ha tenido que enviar una fotografía.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 5.5: Caso de uso 02, Reconocer imagen.

CU03 - Proporcionar información	
Descripción	El sistema debe proporcionar información tanto como de los resultados de los análisis, como de otras consultas sobre los datos almacenados en el.
Relaciones	CU03, CU04, CU09, CU10 RD01, RD02
Secuencia de pasos	Acciones.
1	Recibe una petición.
2	Consulta los datos.
2	Devuelve la información.
Precondición	Recibir una petición con los parámetros correspondientes.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 5.6: Caso de uso 03, Proporcionar información.

CU04 - Ver resultado del análisis	
Descripción	Desde la aplicación se debe poder mostrar el resultado del análisis. En caso de no ser positivo, se debe ofrecer la posibilidad de volver a iniciar el proceso sin tener que realizar la fotografía de nuevo.
Relaciones	CU03, CU02, CU5, CU6 RD01, RD04.
Secuencia de pasos	Acciones.
1	Recibir los datos.
2	Comprobar el tipo de datos.
2	Cargar la vista correspondiente con los datos recibidos.
Precondición	Haber iniciado el proceso de reconocimiento anteriormente.
Importancia	Alta.
Comentarios	Los resultados se tiene que mostrar sin salir de la aplicación.

Tabla 5.7: Caso de uso 03, Ver resultado del análisis.

CU05 - Compartir resultados	
Descripción	El sistema debe ofrecer la posibilidad de compartir los resultados obtenidos con las principales redes sociales.
Relaciones	CU04, CU06 RD01.
Secuencia de pasos	Acciones.
1	Pulsar el botón de compartir.
2	Elegir la red social a la que se quiero compartir el contenido.
2	Confirma la acción.
Precondición	La aplicación debe estar mostrando un resultado.
Importancia	Media.
Comentarios	Ninguno.

Tabla 5.8: Caso de uso 05, Compartir resultados.

CU06 - Gestionar favoritos	
Descripción	La aplicación debe permitir guardar y borrar los resultados como favoritos, de tal manera que el usuario pueda consultarlos en cualquier momento.
Relaciones	CU04 RD01, RD04.
Secuencia de pasos	Acciones.
1	Pulsar el botón de favoritos para añadir/eliminar este .
2	Ir a la pantalla de favoritos y comprobar que se ha realizado el cambio.
Precondición	La aplicación debe estar mostrando un resultado.
Importancia	Media.
Comentarios	Desde el mismo botón se añade o elimina el favorito.

Tabla 5.9: Caso de uso 06, Gestionar favoritos.

CU07 - Ver información sobre la aplicación	
Descripción	Desde la aplicación se debe poder consultar datos sobre la aplicación, además de las licencias/herramientas utilizadas para su implementación.
Relaciones	CU08.
Secuencia de pasos	Acciones.
1	Ir a la pantalla de "Acerca De".
2	Observar la información.
Precondición	Ninguna.
Importancia	Media.
Comentarios	Ninguno.

Tabla 5.10: Caso de uso 07, Ver información sobre la aplicación.

CU08 - Ver tutorial	
Descripción	Al iniciar la aplicación se debe mostrar un tutorial que explique el funcionamiento de la aplicación.
Relaciones	CU02, CU03.
Secuencia de pasos	Acciones.
1	Ir a la pantalla de "Acerca De".
2	Hacer clic en el botón "Ver tutorial".
2	Desplazarse por los pasos del tutorial.
Precondición	Ninguna.
Importancia	Media.
Comentarios	El tutorial se debe poder volver a ver en cualquier momento, si el usuario lo desea.

Tabla 5.11: Caso de uso 08, Ver tutorial.

CU09 - Gestionar campañas	
Descripción	El sistema debe permitir al administrador crear/modificar/eliminar campañas de publicidad, las cuales contendrán imágenes que podrán ser reconocidas.
Relaciones	CU03, CU10 RD01, RD02
Secuencia de pasos	Acciones.
1	Crear/Elegir una campaña.
2	Insertar/Modificar los datos.
3	Añadir/Modificar imágenes.
4	Pulsar el botón de "Guardar".
Precondición	Estar en la interfaz de administración.
Importancia	Baja.
Comentarios	Ninguno.

Tabla 5.12: Caso de uso 09, Gestionar campañas.

CU10 - Gestionar imágenes	
Descripción	El sistema debe permitir, al administrador, añadir imágenes a las campañas con la información que se quiera devolver de cada una.
Relaciones	CU03, CU09 RD01, RD02.
Secuencia de pasos	Acciones.
1	Crear/Elegir una imagen.
2	Insertar/Modificar los datos.
3	Asignar campaña.
4	Pulsar el botón de "Guardar".
Precondición	Estar en la interfaz de administración.
Importancia	Baja.
Comentarios	Ninguno.

Tabla 5.13: Caso de uso 10, Gestionar imágenes.

CU11 - Ver estadísticas	
Descripción	El sistema debe permitir al administrador consultar las estadísticas correspondientes a los accesos al reconocimiento y sus resultados.
Relaciones	CU02 RD01, RD03.
Secuencia de pasos	Acciones.
1	Acceder a la sección de estadísticas.
2	Observar los datos.
Precondición	Estar en la interfaz de administración.
Importancia	Baja.
Comentarios	Ninguno.

Tabla 5.14: Caso de uso 11, Ver estadísticas.

5.2.2. Requisitos de datos

En este apartado se muestran los requisitos de datos del sistema. Estos abarcan toda la información que se deberá almacenar en el sistema para poder cumplir los objetivos planteados en los requisitos funcionales. Estos requisitos se muestran en las Tablas 5.15 - 5.18.

RD01 - Imágenes	
Datos específicos	Id, Nombre, Descripción, Tipo, Valor (depende del tipo)
Requisitos Asociados	RD02, RD03, RD04
Comentarios	Los datos que se almacenarán sobre las imágenes.

Tabla 5.15: Requisito de datos 01, Imágenes.

RD02 - Campañas	
Datos específicos	Nombre, Fecha Inicio, Fecha Fin, Descripcion, Imágenes correspondientes a la campaña
Requisitos Asociados	RD01
Comentarios	Ninguno.

Tabla 5.16: Requisito de datos 02, Campañas.

RD03 - Accesos	
Datos específicos	Fecha, Dispositivo desde el que se realiza, Sistema Operativo, Versión del Sistema Operativo, Resultado del reconocimiento
Requisitos Asociados	RD01
Comentarios	Ninguno.

Tabla 5.17: Requisito de datos 03, Accesos.

RD04 - Favoritos	
Datos específicos	Id, Datos de la imagen
Requisitos Asociados	RD01
Comentarios	Ninguno.

Tabla 5.18: Requisito de datos 04, Favoritos.

5.2.3. Requisitos No Funcionales

Los requisitos no funcionales, son aquellos que especifican los criterios para juzgar las operaciones en lugar de los comportamientos específicos. Estos establecen unas restricciones baja los cuales deberá funcionar el sistema. Estos requisitos se pueden ver en las Tablas 5.19 - 5.20.

RNF01 - Usabilidad	
Descripción	La aplicación debe funcionar de una manera intuitiva, para que cualquier usuario pueda utilizarla.

Tabla 5.19: Requisito no funcional 01, Usabilidad.

RNF02 - Respuesta rápida	
Descripción	El reconocimiento de imágenes se debe realizar en un tiempo, considerablemente, corto.

Tabla 5.20: Requisito no funcional 02, Respuesta rápida.

Capítulo 6

Diseño

En el siguiente capítulo se presenta todo lo que envuelve al diseño de la aplicación. En primer lugar se muestra el diseño de la interfaz gráfica. Y, seguidamente, se muestra la parte de este referente al diseño del sistema.

6.1. Diseño de la interfaz

En esta sección se muestra todo lo que se ha tenido en cuenta a la hora de diseñar la interfaz gráfica. Primero se comenta el proceso de diseño de los elementos del sistema comunes a toda la aplicación, y después se comentarán los prototipos realizados para las distintas pantallas de esta.

Antes de profundizar en el diseño de las diferentes vistas que tendrá la aplicación, se presenta una visión general de esta, en la cual se sigue el flujo que tendrá la aplicación entre las difrentes vistas, en este caso utilizando los prototipos que se explicarán en las siguientes secciones (Figura 6.1).

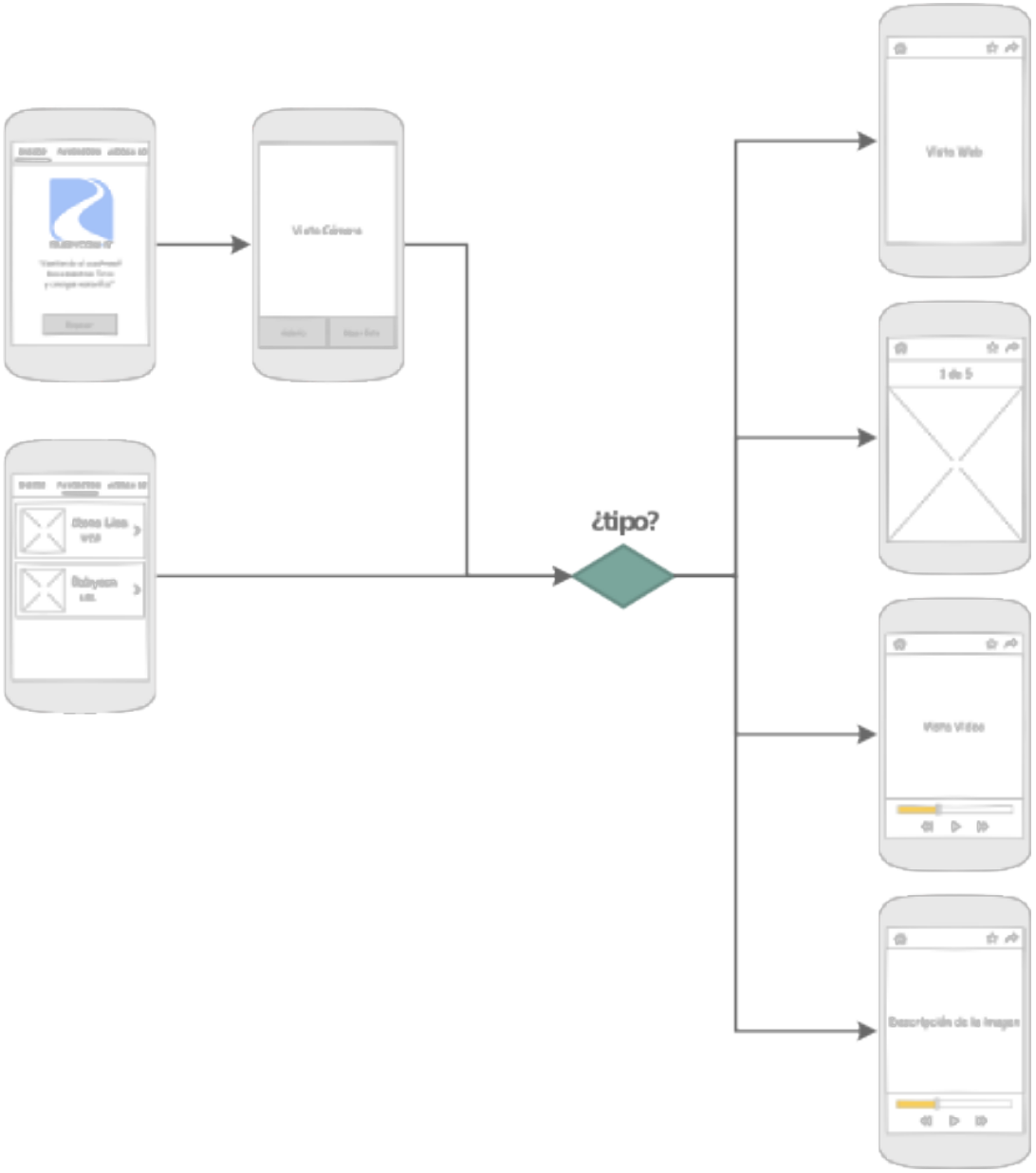


Figura 6.1: Visión general del flujo de la aplicación, mendiante prototipos.

6.1.1. Elementos comunes

La principal característica visual que se repetirá a lo largo de las diferentes pantallas de la aplicación son los colores. La elección de los colores fue una tarea rápida. Por parte de la empresa se recomendó utilizar un color de los ofrecidos por el portal de Internet, *FlatUI Colors* [4], así que siguiendo esta recomendación se eligió, como color principal para la aplicación, el azul "Peter River". Para los componentes que deben contrastar con este color principal, se tomó la decisión de aplicarles un color blanco, de manera que se distinguen perfectamente los dos colores.

Para el diseño del icono de la aplicación se utilizaron los dos colores elegidos para esta, el

azul para el fondo y el banco para el resto. El diseño de este fue una tarea complicada, ya que es difícil elegir una sola imagen que represente la funcionalidad de la aplicación. Finalmente, se optó por un diseño plano y combinando algunos iconos del conocido portal web de iconos vectoriales *FlatIcon* [17] se consiguió un resultado aceptable, el cual se puede ver en la Figura 6.2.

Figura 6.2: Icono de la aplicación.

Tanto para la realización del icono, como para los diferentes botones que se verán a lo largo de este capítulo se utilizó el programa Adobe Illustrator, que permite el diseño y edición de gráficos vectoriales.

6.1.2. Vistas Principales

Las vistas principales serán las que aparecerán al iniciar la aplicación y desde las que se navegará hacia las principales funciones de esta.

Cuando se acceda a la aplicación se mostrará una vista, a la que se le ha llamado Inicio, desde la cual se podrá acceder a la cámara para empezar el proceso principal de la aplicación. En esta se muestra el logo de la empresa, junto a un mensaje inicial para dar la bienvenida al usuario. Además, contiene un botón desde el que se accede a la cámara (Figura 6.3).

Para navegar entre las ventanas de una manera rápida y sencilla, se ha introducido en la parte superior un barra, desde la cual se pueda acceder a las vistas de "favoritos" y "acerca de". Además, no sólo se podrá acceder haciendo clic encima de esta barra, sino que simplemente pulsando con el dedo en la pantalla y arrastrando hacia los laterales, el usuario se podrá desplazar entre las distintas vistas citadas anteriormente, en lo que se conoce como un movimiento de "swype".

De esta manera se accede a la vista llamada "favoritos", en la cual se muestra al usuario todos los resultados que ha ido añadiendo en favoritos. Por cada favorito se mostrará la imagen asociada a este, junto con el título y el tipo de *feedback* que devuelve al ser reconocida. Pulsando

en cada uno de ellos se podrá ver su contenido (Figura 6.4).

Desde esta barra de navegación también se podrá acceder a un apartado llamado “acerca de”, en el cual se mostrará información sobre la aplicación.

Figura 6.3: Prototipo de la vista “inicio”.

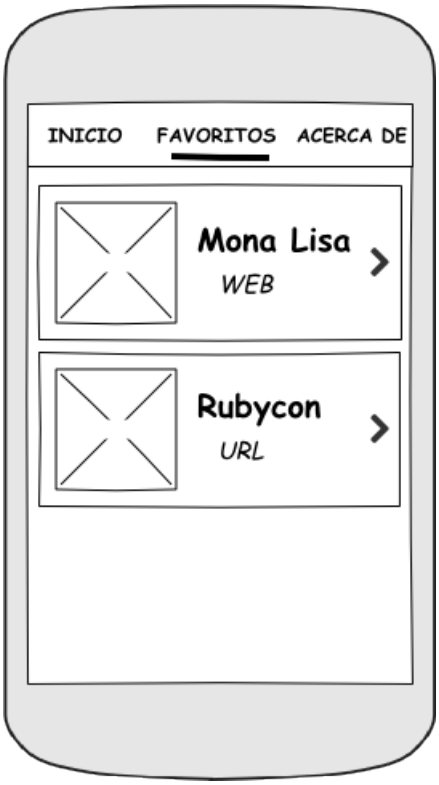


Figura 6.4: Prototipo de la vista ´´favoritos”.

Volviendo a la vista de “inicio”, pulsando en el botón “empezar”, se accede a la interfaz de cámara. En ella se podrá realizar una fotografía, simplemente pulsando en el botón “Hacer Foto”, o por otro lado, también se podrá acceder a la galería de imágenes, pulsando en el botón “Galería”, para realizar el proceso de reconocimiento con una de las imágenes contenidas en ella (Figura 6.5).

La última de las vistas principales es la correspondiente al tutorial. En la Figura 6.6 se puede ver una plantilla de cómo se mostrará la información en el tutorial, la cual será repetida tantas veces como pasos contenga el este . Para navegar en este tutorial se utilizará el mismo procedimiento que para navegar entre las pantallas principales, es decir, el usuario podrá desplazarse entre los pasos con el movimiento del dedo. Por último, se podrá saltar el tutorial en todo momento pulsando el botón “Saltar”.

Figura 6.5: Prototipo de la vista “cámara”.

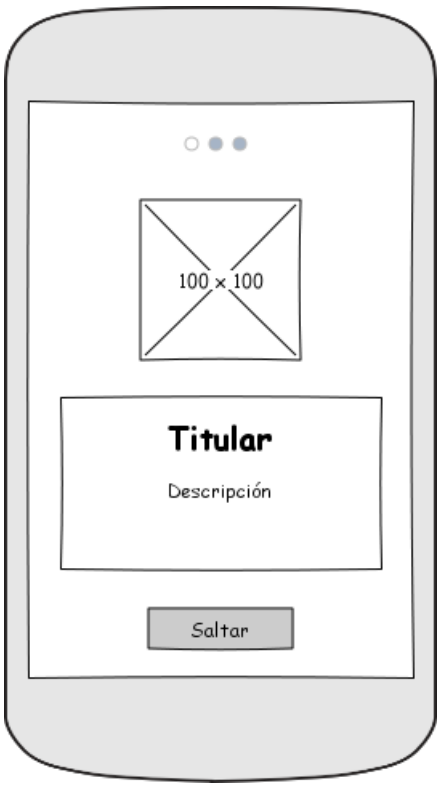


Figura 6.6: Prototipo de la vista “tutorial”.

6.1.3. Vistas de Resultados

Las vistas de resultados son las que albergan el papel de mostrar la información resultante del proceso de reconocimiento de imágenes. Como se ha comentado en el capítulo correspondiente, la aplicación ofrecerá al usuario resultados de diferentes tipos, por lo tanto deberá contener las vistas correspondientes para poder mostrar todos estos tipos.

Todas estas vistas de resultados disponen de la misma barra en la parte superior, desde la cual se podrá marcar o desmarcar el resultado como favorito, compartir el resultado mediante las principales redes sociales y volver a la pantalla anterior.

Uno de los resultados que debe mostrar la aplicación, es una página web. Para ello simplemente se ha diseñado una vista, en la que el navegador web ocupe toda la pantalla, a excepción de la barra de acciones (Figura 6.7).

Otro de estos resultados, será una galería con varias imágenes. Para mostrar estas imágenes se ha optado por una gale?ía en la que se muestra una foto a pantalla completa desde la cual se pueda navegar entre las fotos con simples gestos táctiles. Además, se mostrará también, un indicador para que el usuario pueda saber cuántas fotos puede ver, y la posición de la que está siendo visualizada (Figura 6.8).

Figura 6.7: Prototipo de la vista “resultado web”.

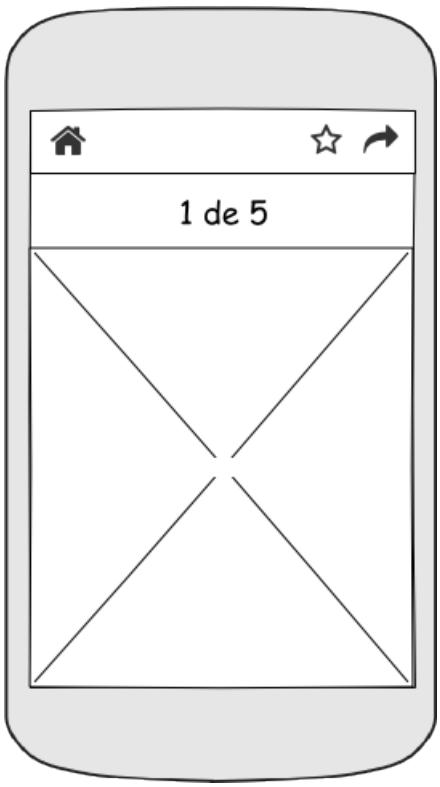


Figura 6.8: Prototipo de la vista “resultado galería”.

Por último, están los resultados correspondientes a los reproductores de audio y video. La vista correspondiente al reproductor de audio consta de unos controles multimedia en la parte de abajo, desde los cuales podrá: reproducir, pausar, retroceder y avanzar; además de una barra que muestra el segundo que se está reproduccion, y desde la cual, el usuario podrá también avanzar o retroceder el audio. Además, en el espacio restante, se mostrará la descripción asociada a la imagen correspondiente al resultado (Figura 6.9).

En cuanto al reproductor de vídeo, en la parte de abajo contendrá, exactamente, los mismos controles multimedia que el reproductor de audio. Pero, esta vez se mostrará el video en el centro de la pantalla, y no la descripción (Figura 6.10).

Figura 6.9: Prototipo de la vista “resultado audio”.

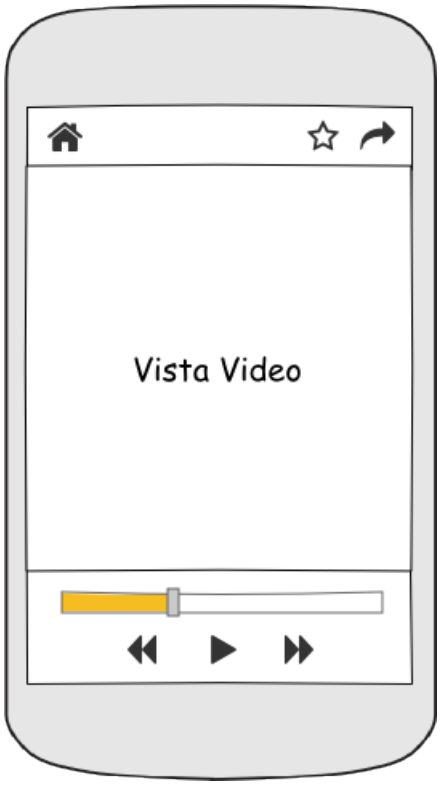


Figura 6.10: Prototipo de la vista “resultado video”.

6.2. Diseño del sistema

6.2.1. Diseño de los Procesos

El siguiente punto está dedicado a mostrar los procesos diseñados para las funcionalidades de la aplicación. Para ello se utilizarán diagramas de actividad con tal de ofrecer una mejor comprensión al lector.

En el caso de este proyecto, el proceso principal, y más importante, es el de realizar una fotografía a un objeto, que el sistema trate de reconocerla, y devuelva la información asociada a esta. Por lo tanto, este es el proceso que más tiempo de diseño ha requerido. En la Figura 6.11 podemos ver el funcionamiento que se ha diseñado para él.

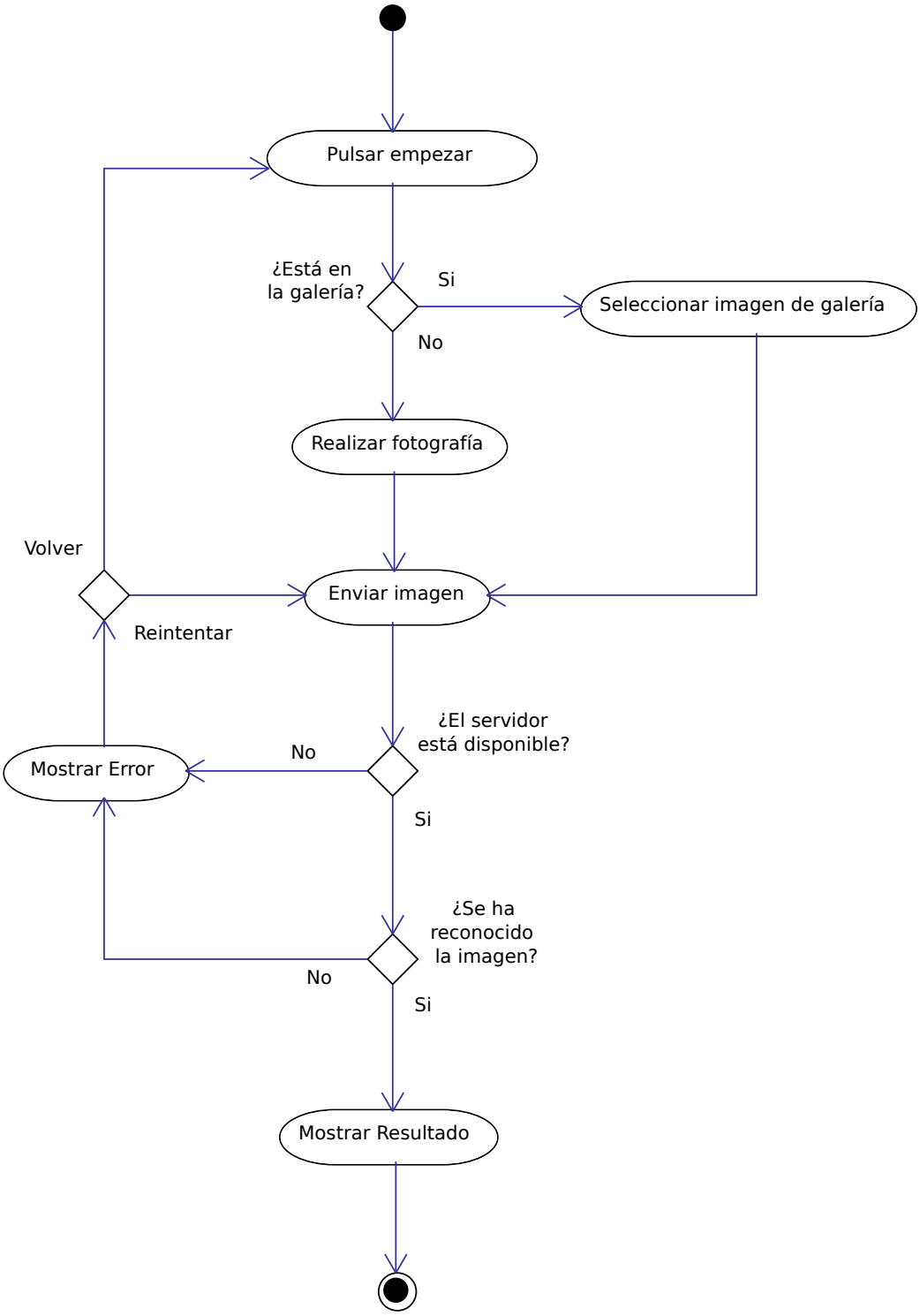


Figura 6.11: Diagrama de actividad del proceso principal de la aplicación.

En cuanto al resto de procesos que se se deben realizar, engloban tareas muy pequeñas, para las cuales no hace falta un diagrama de actividad que muestre su funcionamiento, ya que no

responden a más de uno o dos pasos para ser realizadas.

6.2.2. Diagramas de Clase

En esta sección se muestran los diagramas de clase, de los patrones de diseño utilizados. Los diagramas de clases muestran la estructura del sistema, especificando las clases que este contiene y mostrando los detalles de estas para su implementación. A continuación, se muestran los diagramas de clases más representativos, por lo que al diseño se refiere.

Para la parte del dispositivo móvil, desarrollada en Java, quizá el diagrama más constructivo sea el referente a la parte de la base de datos interna (Figura 6.12). En este se puede ver, por un lado el modelo, que corresponde a la clase *Favorito*, la cual contiene los atributos que se deben guardar en la base de datos. Por otro lado, está la clase *FavoritosSQLiteHelper*, que es la encargada de crear y acceder a la base de datos, y contiene como atributos las sentencias de creación de esta y sus columnas. Y, por último, tenemos una implementación del patrón *DAO*, el cual actúa como pasarela entre las dos clases anteriores, es decir, gracias a este se puede trabajar como si de una colección Java se tratase, añadiendo, eliminando o modificando con simples métodos y pasando directamente los objetos de tipo Favorito.

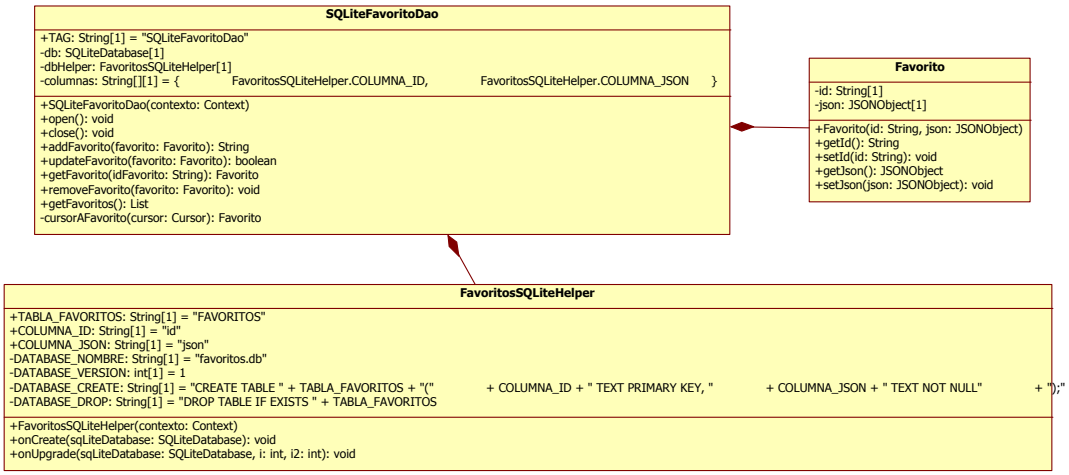


Figura 6.12: Diagrama de datos de la parte correspondiente a la base de datos.

En lo referente a la parte del servidor, el diagrama más destacable, es el correspondiente a la implementación que se realiza del patrón *Strategy*, mediante el cual conseguimos que el objeto *Reconocedor* pueda cambiar el tipo de *Detector* sin modificar su implementación interna, simplemente creando otra clase que implemente a este y pasándola como argumento o con el método *setDetector*. Esto también ocurre con el objeto Comparador, pero se ha reducido el diagrama para una mejor visualización de este, el cual se puede ver en la Figura 6.13.

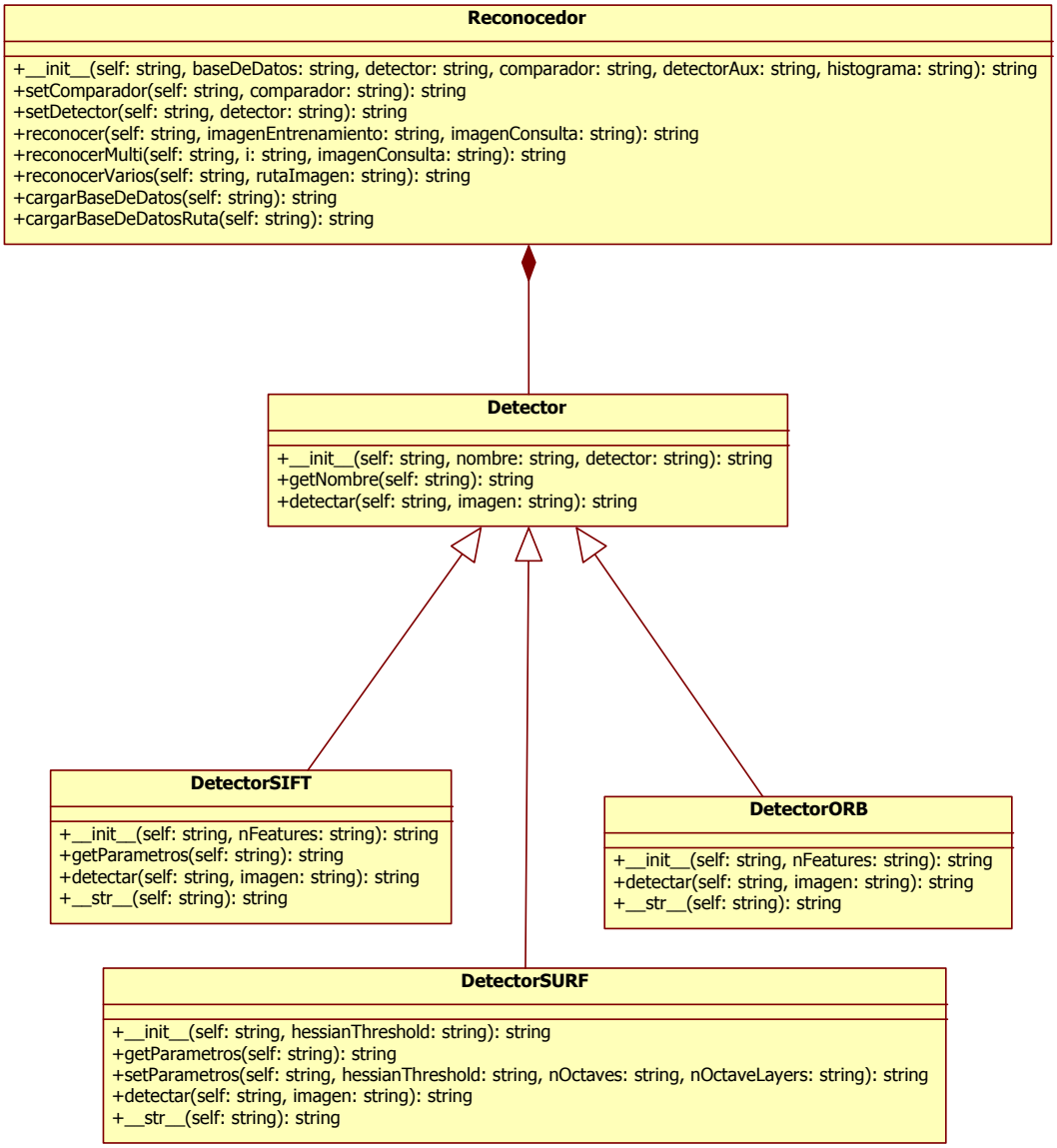


Figura 6.13: Parte del diagrama de clases del módulo de reconocimiento del servidor.

Capítulo 7

Implementación

En el siguiente capítulo se comentarán los aspectos más destacados de la implementación del sistema. Estos aspectos serán comentados desde una visión general, dándole más importancia a algunos procesos, pero sin profundizar altamente en ninguno de ellos, ya que de ser así la extensión del documento crecería enormemente.

Este sistema corresponde a una arquitectura cliente-servidor, en la cual el cliente será una aplicación móvil para el sistema operativo Android. El sistema se divide en tres partes principales: la aplicación móvil, el reconocimiento de imágenes y el servidor.

7.1. Aplicación Android

Esta sección, corresponde a la parte cliente de sistema, y esta desarrollada para ser utilizada desde un dispositivo móvil. Como se ha visto en la descripción del Proyecto (Capítulo 3, el cometido de esta será permitir al usuario realizar una fotografía y enviarla al servidor para que procese su reconocimiento. Una vez la imagen se haya analizado, será también esta parte la que muestre los resultados.

7.1.1. Interfaz Gráfico

En Android, el código que implementa la funcionalidad y las interfaces gráficas de las vistas se construye en archivos diferentes. La funcionalidad, como ya se ha comentado en alguna ocasión, se implementa mediante clases Java, mientras que las interfaces lo hacen en archivos XML. Estos archivos se llaman *layouts* y desde ellos se pueden especificar los componentes que debe contener la vista indicando su posición, incluso se puede indicar la función que será llamada al pulsar un botón. Por ejemplo, en el Código 7.1, se puede ver el *layout* correspondiente a una imagen, concretamente el logo de Rubycon, seguido de un mensaje de texto.

Código 7.1: Parte del *layout* de *InicioFragment*.

```
1 <LinearLayout
```

```
2       android:layout_width="fill_parent"
3       android:layout_height="fill_parent"
4       android:background="@color/clrPrincipal"
5       android:gravity="center_vertical|center_horizontal"
6       android:orientation="vertical"
7       android:textAlignment="center"
8       android:padding="10dp">
9
10      <ImageView
11      android:id="@+id/imgRubycon"
12      android:layout_width="200dp"
13      android:layout_height="200dp"
14      android:layout_alignEnd="@+id/lblMensajeInicial"
15      android:layout_marginTop="10dp"
16      android:src="@drawable/logo_rubycon_invertido" />
17
18      <es.rubycon.reconocimientoimg.app.TextViewRoboto
19      android:id="@+id/lblMensajeInicial"
20      android:layout_width="wrap_content"
21      android:layout_height="wrap_content"
22      android:layout_below="@+id/imgRubycon"
23      android:layout_centerHorizontal="true"
24      android:layout_marginTop="32dp"
25      android:gravity="center"
26      android:text="Bienvenido al cazaPromo!\nBusca nuestras fotos\ny
27      consigue maravillas"
28      android:textAppearance="?android:attr/textAppearanceLarge"
29      android:textColor="@color/clrSecundario" />
30
31  </LinearLayout>
```

Al igual que con las interfaces gráficas, Android utiliza el mismo sistema para los textos, los colores, los estilos, los valores, etc. Con esto se consigue una abstracción de estas partes respecto del código Java, cosa que favorece la legibilidad del código.

Así se han implementado todas las interfaces gráficas de la aplicación, pero no todos los componentes gráficos que puede contener una aplicación Android se pueden especificar directamente en estos archivos, algunos hay que añadirlos a través de código Java. Los más significativos se comentarán en las siguiente sección.

También cabe destacar que para la mayoría de los iconos de la aplicación se ha utilizado la fuente Font Awesome, que emplea los iconos como si de letras se tratasen, por tanto se pueden escalar y cambiar de color al gusto del desarrollador. Esta tarea no se puede tampoco realizar desde los archivos XML, por tanto, los iconos se han aplicado desde las clases Java [14].

7.1.2. Transición animada entre pantallas

En la fase de diseño se tomó la decisión de que entre las pantallas principales de la aplicación se pudiera navegar con un simple gesto de pasar página, es decir, arrastrando con el dedo. Para lograr esta funcionalidad existen varias alternativas, en este caso, al tratarse de las pantallas principales de la aplicación, se ha optado por implementar unas *Swipe Tags*, como se les denomina en la documentación oficial de Android, es decir, una navegación por pestañas, pero que a la vez se puedan pasar con el dedo.

Estas pestañas se añaden a la *Action Bar* de la aplicación con un simple método add en el objeto *Action Bar*, pero es esta la que contiene un *PageViewer* interno. En cada posición de este se carga un fragmento, el cual corresponde a la pestaña seleccionada en cada momento. Para realizar esto, se ha implementado un escuchador personalizado para él, de manera que dependiendo la página actual se cargue un fragmento u otro (Figura 7.1).

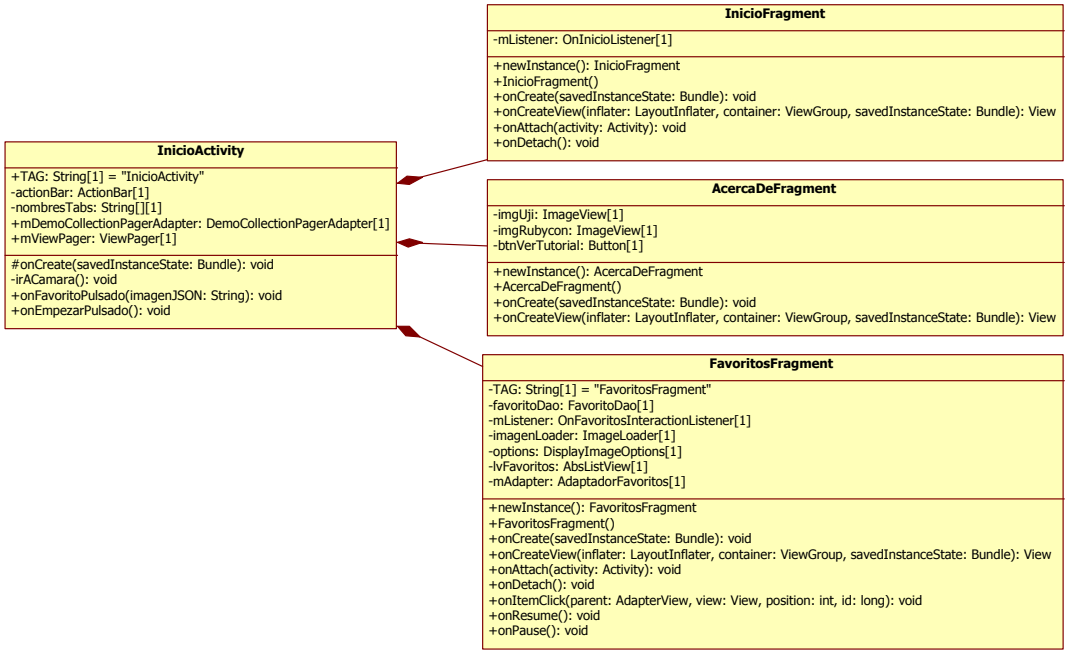


Figura 7.1: Diagrama de clases correspondiente a la transición entre pantallas.

Además de esto, para controlar que al pulsar una pestaña y al arrastrar con el dedo se cambie de página, se han implementado dos escuchadores más que realizan esta función: un *TabListener* y un *OnPageChangeListener*.

7.1.3. Realización de la fotografía

Debido a que el sistema es un reconocedor de imágenes, es necesario la realización de fotografías desde la aplicación. En Android, existen dos opciones para cumplimentar esta función. Por un lado, se podría directamente lanzar una intención implícita, mediante la cual se solicita

al sistema que se desea realizar una fotografía.Estoo abriría la aplicación de cámara del sistema Android, de manera que se saldría de la aplicación desarrollada. La otra opción existente es la implementación de una cámara dentro de la aplicación, con lo que se consigue realizar fotografías sin salir de ella. En este proyecto se ha optado por la segunda, para poder realizar fotografías desde la misma aplicación, cosa que hace que se complique un poco dicha tarea.

Para implementar una cámara desde dentro de la aplicación, primero se ha creado una actividad, la cual contiene un *SurfaceView*, donde se mostrará la vista de la cámara y los botones correspondientes para realizar la fotografía y, en este caso, también para abrir la galería. El atributo principal de esta actividad es un objeto de la clase cámara, que es la interfaz que ofrece Android para comunicarse con esta. Con este objeto ya se pueden tomar fotografías, para ello es suficiente con llamar al método *takePicture*, pasándole como argumento un escuchador que será llamado cuando la fotografía haya sido capturada (Código 7.2).

Código 7.2: Método *hacerFoto*.

```
1  public void hacerFoto() {
2      Camera.PictureCallback mPicture = new Camera.PictureCallback() {
3          @Override
4          public void onPictureTaken(byte[] data, Camera camera) {
5
6              File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
7              if (pictureFile == null) {
8                  Log.d(TAG, "Error creando el archivo multimedia,
9                      comprueba los permisos.");
10                 return;
11             }
12             try {
13                 FileOutputStream fos = new FileOutputStream(pictureFile);
14                 fos.write(data);
15                 fos.close();
16                 imageFileTemp = pictureFile;
17                 enviarImagenPorREST(imageFileTemp);
18                 ImagenUtils.actualizarGaleria(getApplicationContext());
19             } catch (FileNotFoundException e) {
20                 Log.d(TAG, "Error. No se encuentra el archivo: "
21                     + e.getMessage());
22             } catch (IOException e) {
23                 Log.d(TAG, "Error. Archivo no accesible: "
24                     + e.getMessage());
25             }
26         }
27     };
28     mCamera.takePicture(null, null, mPicture);
29 }
```

Pero no sólo ha bastado con realizar fotografías, también se deben mostrar la imágenes que va capturando objetivo de la cámara en todo momento, algo que ha resultado más complicado. Para ello, se ha extendido un objeto de la clase *SurfaceView* que se ha llamado *CamaraPreview*,

el cual en su construcción recibe una instancia del objeto cámara para mostrar su imagen en todo momento (Figura 7.2.

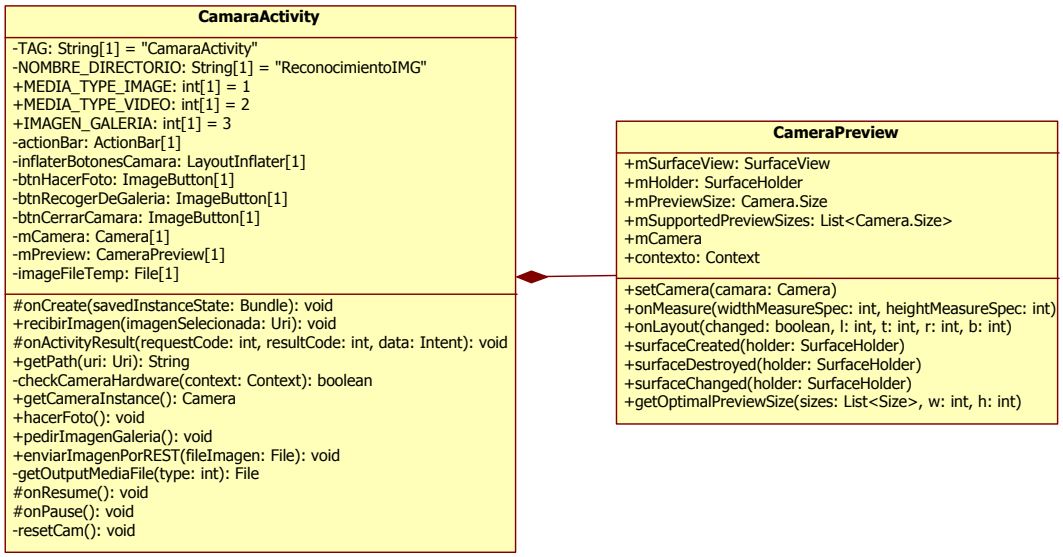


Figura 7.2: Diagrama de clases correspondiente a la implementación de la cámara.

Uno de los principales inconvenientes de esta funcionalidad ha sido que hay que liberar la cámara cuando no se use y, a su vez, recuperarla cuando sea necesario su uso. Esto conlleva mantener el objeto *Camera* contenido en la actividad y este mismo objeto contenido en el objeto *CameraPreview* perfectamente sincronizados, cosa que ha llevado más de un quebradero de cabeza. Por otro lado, otro inconveniente ha sido calcular el tamaño de la vista previa ideal para cada dispositivo, ya que existen una infinidad de dispositivos con Android, con diferentes tamaños de pantalla y diferentes tipos de cámaras.

Por último, en lo que respecta a la obtención de la imagen desde la galería, ha bastado simplemente con realizar una intención implícita, en la cual se especifique que se quiere elegir una imagen de la galería (Código 7.3).

Código 7.3: Método *pedirImagen*.

```
1 public void pedirImagenGaleria() {
2     Intent intent = new Intent();
3     intent.setType("image/*");
4     intent.setAction(Intent.ACTION_GET_CONTENT);
5     intent.addCategory(Intent.CATEGORY_OPENABLE);
6     startActivityForResult(Intent.createChooser(intent, "Seleccionar
7         imagen"), IMAGEN_GALERIA);
8     Log.d(TAG, "Mostrando imagen de la galeria");
9 }
```

7.1.4. Comunicación con el servidor REST

Respecto a la conexión entre el cliente y el servidor, esta se debe realizar en segundo plano, sin alterar el comportamiento de la aplicación, y controlando los errores que puedan suceder.

Para simplificar esta tarea, se ha optado por utilizar una de las librerías disponibles para ello. En este caso se ha optado por la librería *Android Asynchronous HttpClient* [19]. Esta está basada en la librería *HttpClient* de Apache incluida en Android, además, todas las tareas que se realicen desde ella se ejecutan en segundo plano, ahorrando esta tarea al desarrollador. Existen diferentes alternativas, por ejemplo *Volley* y *RoboSpice*, pero se ha optado por esta por su sencillez, sobre todo a la hora en enviar y recibir imágenes.

Con la utilización de esta librería, la comunicación cliente-servidor resulta una tarea sencilla. Como podemos ver en la Figura 7.3, se ha optado por seguir los cánones establecidos por el creador de la librería y se ha implementado una clase estática mediante la cual se realizarán las peticiones.

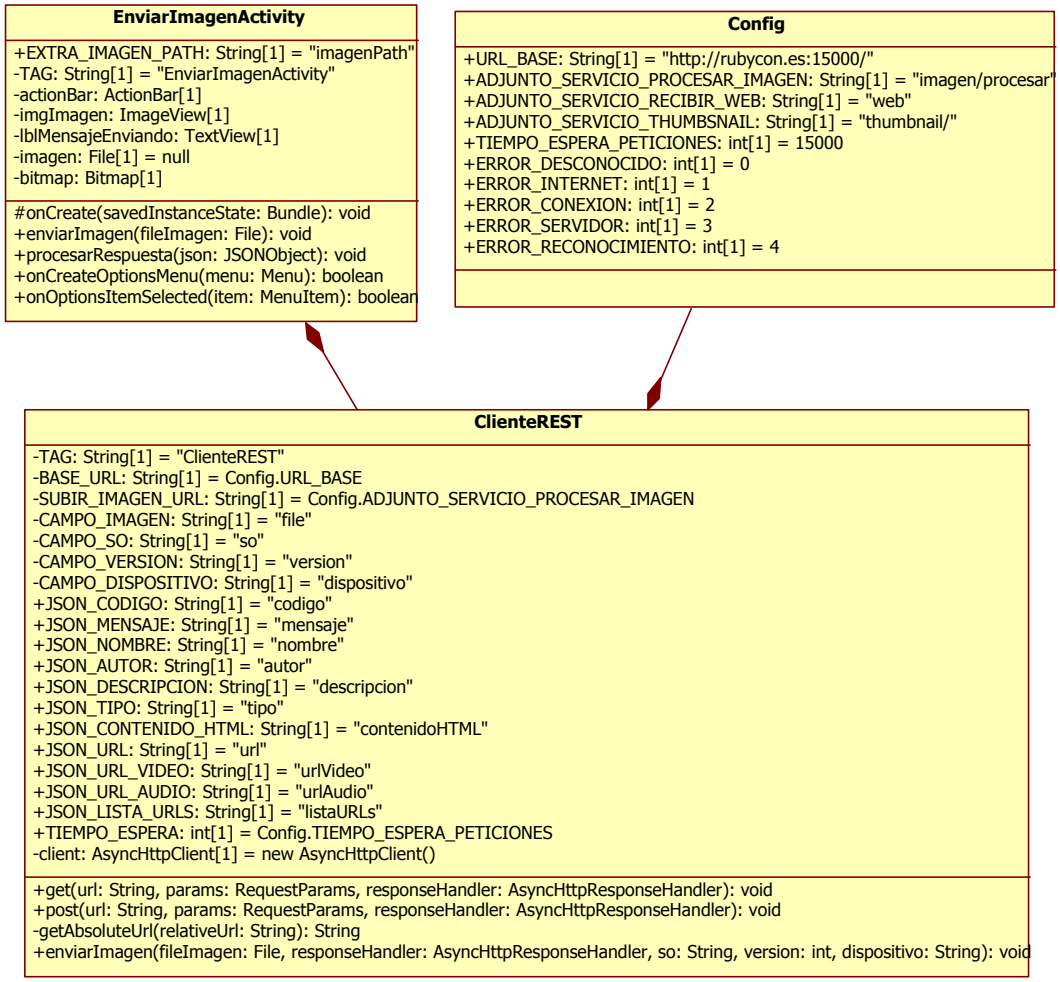


Figura 7.3: Diagrama de clases correspondiente a la comunicación con el servidor REST.

La petición más importante del sistema es la que envía la imagen para proceder a su reconocimiento, este es el cometido del método enviarImagen, el cual recibe :una imagen, en forma de File; un manejador de respuesta, que será llamado cuando se reciba una respuesta del servidor; y tres parámetros con información sobre el sistema operativo, versión y dispositivo.

Para capturar la respuesta obtenida al finalizar una petición, basta con implementar la interfaz ResponseHandler, la cual contiene diferentes métodos que serán llamados dependiendo del resultado de la petición (Código 7.4).

Código 7.4: Parte del escuchador *JsonHttpResponseHandler* utilizado para recibir la respuesta del servidor.

```
1  ClienteREST.enviarImagen(fileImagen, new JsonHttpResponseHandler() {
2
3      @Override
4      public void onStart() {
5          Log.d(TAG, "Envio iniciado.");
6          /* ... */
7      }
8
9      @Override
10     public void onSuccess(JSONObject response) {
11         Log.d(TAG, "Envio realizado con Ã©xito.");
12         /* ... */
13     }
14
15     @Override
16     public void onFailure(int statusCode, Header[] headers, byte[] responseBody,
17         Log.d(TAG, "Envio fallido. Status: " + statusCode + ". Error: " + error)
18         /* ... */
19     }
20
21     @Override
22     public void onRetry() {
23         Log.d(TAG, "Reintentado el envio...");
24     }
25
26     @Override
27     public void onProgress(int bytesWritten, int totalSize) {
28         Log.d(TAG, "Envio en proceso...");
29     }
30
31     @Override
32     public void onFinish() {
33         Log.d(TAG, "Envio finalizado.");
34         /* ... */
35     }
36 }, so, version, dispositivo);
```

Con el objetivo de estructurar bien el código y proporcionar *feedback* al usuario mientras se envía una imagen, se ha implementado una actividad llamada *EnviarImagenActivity*. Esta será lanzada en el momento que se quiera enviar una imagen al servidor y será desde donde se efectuarán todas las operaciones realizadas con esta funcionalidad. Además, cuando se lanze esta actividad el usuario verá una animación durante el tiempo que dure el reconocimiento, con lo que se consigue que la espera sea más llevadera.

7.1.5. Presentación de los resultados

Otra parte importante del cliente es la presentación de los resultados del reconocimiento. Esto conlleva la representación, desde dentro de la aplicación, de los distintos tipos de resultado, ya sea una galería de imágenes o un vídeo. Para ello, con el objetivo de repetir el mínimo código posible, se ha optado por la implementación de una sola actividad, llamada *ResultadoActivity*, y un fragmento para cada tipo de resultado, de manera que dependiendo del tipo de resultado, se cargue un fragmento u otro.

Esta actividad de resultado debe recibir como parámetro un *JSONObject* con la información sobre la imagen reconocida. Una vez recibido, se llama al método *recibirDatos*, el cual explora esto JSON y carga en variables globales toda la información que contiene. Es entonces cuando ya se sabe el tipo de resultado, cuando se llama al método *anyadirFragment*, que carga el fragmento correspondiente al tipo de resultado (Figura 7.4).

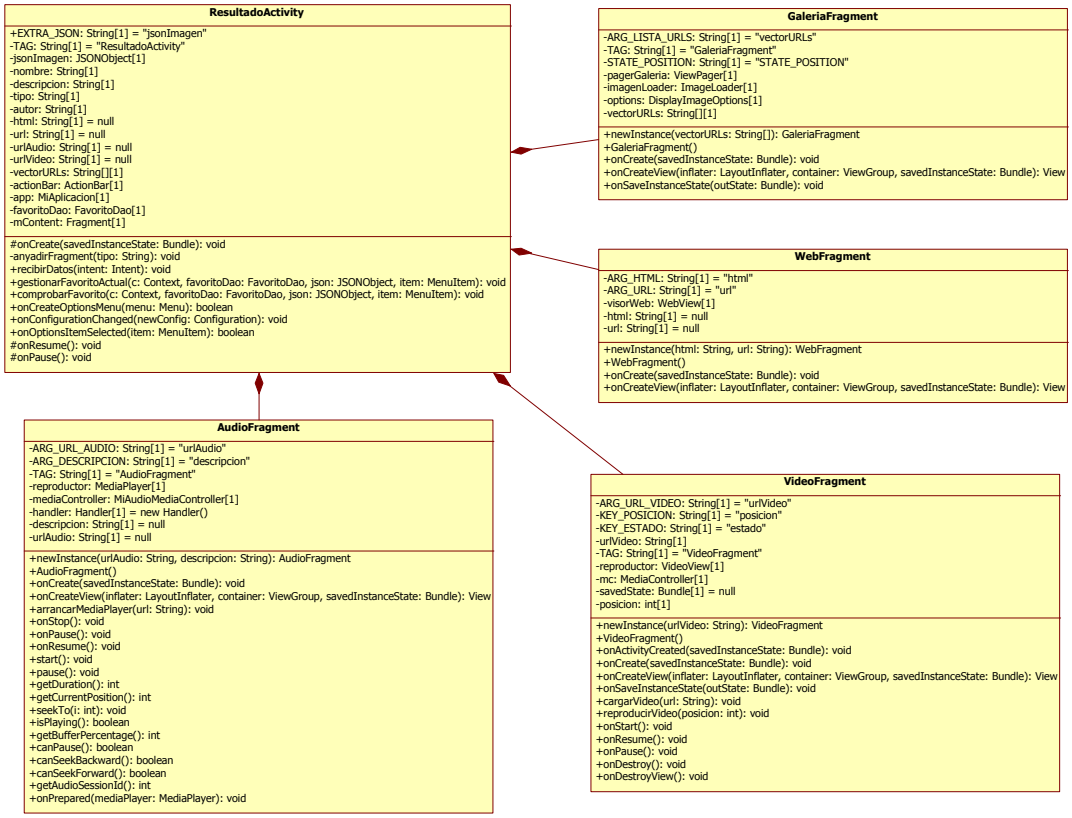


Figura 7.4: Diagrama de clases correspondiente a la presentación de resultados.

A continuación, se explica de manera resumida la implementación de cada fragmento:

- *AudioFragment*: Debe recibir como parámetros una URL, en la que se encuentra el archivo de audio que se reproducirá vía streaming, y una descripción que se quiera mostrar por pantalla. Su layout está formado por un *TextView*, en el cual se mostrará la descripción, por un *MediaController*, que será el encargado de mostrar los controles multimedia y, por último, aunque este sin representación gráfica, un *MediaController*, que será el encargado de reproducir el audio. Cabe destacar de la implementación de este fragmento, que no ha resultado una tarea fácil la integración del objeto *MediaPlayer* con el *MediaController*, ya que no está desarrollado para controlar video, pero se ha optado por esta opción para mantener una consistencia con el reproductor de video.

- *GaleriaFragment*. Este debe recibir un vector con las URL de las imágenes. Para su interfaz se ha utilizado un *ViewPager* de manera que se puedan pasar las imágenes con el dedo. Además, se han integrado los típicos gestos de galería, como hacer zoom con los dedos, gracias la librería *PhotoView* [13]. Para cargar las imágenes de manera asíncrona desde Internet, se ha utilizado la librería ImageLoader, la cual facilita esta función.

- *VideoFragment*: Este, al igual que el fragmento de audio, debe recibir una URL, en la que se encuentre un archivo de video, el cual será reproducido vía streaming. La interfaz de este fragmento, estará formada por un *VideoView* donde se mostrará el video y un *MediaController*, que contiene los controles multimédia.

- *WebFragment*: Debe recibir una URL o una cadena que contenga código HTML. Su interfaz es simple, formada por un *WebView*, donde se muestra una web o el contenido HTML, dependiendo de lo recibido.

7.1.6. Gestión de favoritos

La funcionalidad correspondiente a la gestión de favoritos es sencilla pero, a la vez, hubo que investigar sobre la creación y gestión de una pequeña base de datos en Android.

Android ofrece facilidades para la gestión de bases de datos SQLite, que son ideales para este cometido, ya que simplemente se deben almacenar favoritos. Para la implementación de esta base de datos utilizando estas facilidades, simplemente se ha extendido la clase *SQLiteOpenHelper*, en la cual se han especificado mediante sentencias SQL como se debe crear, actualizar y borrar la base de datos (Figura 7.5).

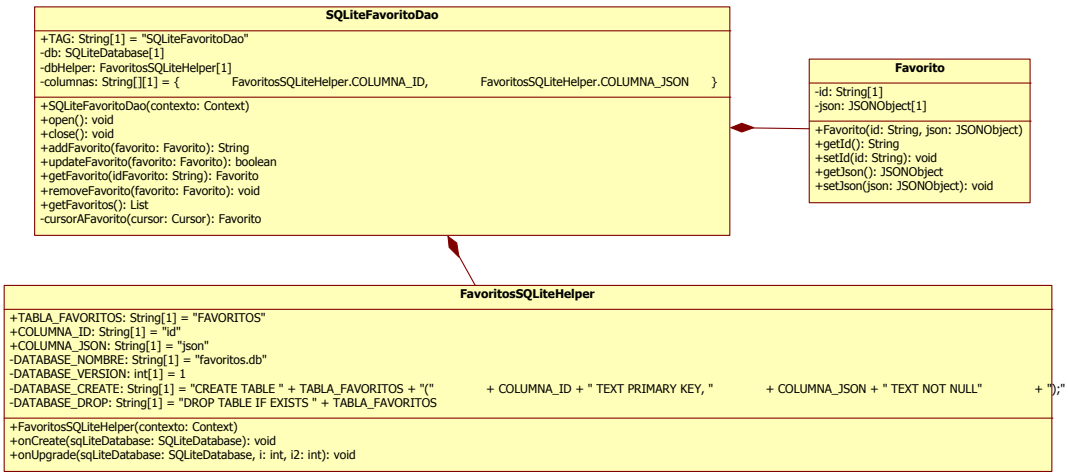


Figura 7.5: Diagrama de clases correspondiente a la gestión de favoritos.

Con este objeto creado ya se podría directamente trabajar con la base de datos, pero para facilitar aún más esta tarea, se ha elaborado una implementación del patrón de diseño *DAO* (*Data Access Object*), con el cual se consigue trabajar con la base de datos prácticamente como si de un objeto se tratase. Para ello se ha creado una interfaz, llamada *FavoritoDao*, con los métodos necesarios para trabajar como si se tratase de una colección Java (Código 7.5). Posteriormente se ha implementado esta interfaz específica para SQLite, utilizando el objeto creado anteriormente para interactuar con la base de datos.

Código 7.5: Interfaz *FavoritoDao*.

```
1  public interface FavoritoDao {
2      public void open() throws SQLException;
3
4      public void close();
5
6      public String addFavorito(Favorito favorito);
7
8      public boolean updateFavorito(Favorito favorito);
9
10     public Favorito getFavorito(String idTarea);
11
12     public void removeFavorito(Favorito favorito);
13
14     public List<Favorito> getFavoritos();
15 }
```

7.2. Reconocimiento de imágenes

Esta parte del sistema, es la encargada de reconocer las imágenes. Para ello, deberá recibir un conjunto de imágenes y una imagen para buscar en este . Este módulo del sistema,

deberá realizar el procedimiento necesario para determinar si la imagen está contenida en el conjunto de imágenes o no.

7.2.1. Detección y comparación de descriptores

Como se vio en el capítulo 2, el correspondiente al Estudio Previo, para comparar dos imágenes hace falta un algoritmo que detecte los puntos descriptores de las imágenes, y un algoritmo que compare estos puntos entre las diferentes imágenes, buscando similitudes entre ellos.

Para obtener los algoritmos más adecuados para este sistema, se han implementado los algoritmos SURF, SIFT, ORB, FLANN y BFMatcher, para poder realizar las pruebas necesarias y comprobado su funcionamiento en el ámbito de este proyecto.

Para la implementación de los algoritmos detectores de descriptores, se ha optado por crear una clase abstracta, llamada *Detector*, la cual contenga el método *detectar*. El objetivo de esta clase, es que cada algoritmo que se quiera añadir, simplemente deberá extender esta clase, por lo tanto, se podrá intercambiar con facilidad el tipo de algoritmo, cosa realmente útil, sobre todo para las pruebas.

En este caso, se han implementado los tres algoritmos mencionados anteriormente, mediante las clases *DetectorSURF*, *DetectorSIFT* y *DetectorORB*, que como se ha dicho, extienden a la clase *Detector*. En la Figura 7.6, se puede ver el diagrama de clases de esta estructura.

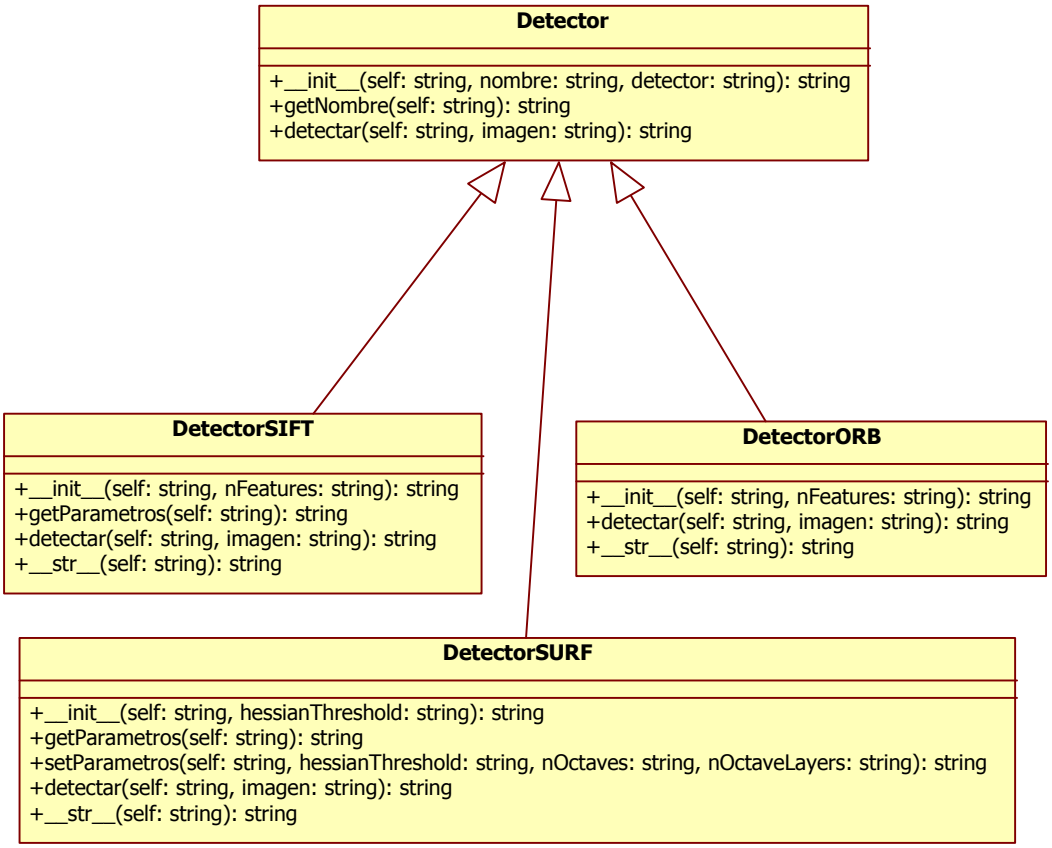


Figura 7.6: Diagrama de clases del algoritmo detector.

Por lo tanto, todas estas clases contienen un método detectar que será el encargado de analizar la imagen en búsqueda de sus descriptores y de devolver un objeto *ImagenAnalizada* que contenga el resultado de este .

La implementación de este por parte de los diferentes algoritmos, se simplifica gracias a la utilización de OpencCV. Para el caso de SURF y SIFT basta con crear una instancia de estos llamando a *cv2.SURF()* o *cv2.SIFT()* y llamar a su método *detecteAndCompute*, el cual devuelve dos vectores con los *keypoints* y los descriptores. Para el algoritmo ORB también se debe crear una instancia con *cv2.ORB()*, pero en este caso hay que realizar dos pasos: primero calcular los *keypoints* con *detect*, y luego calcular los descriptores con *compute*.

En cuanto a la comparación de descriptores, se ha seguido el misma patrón que para la de los descriptores. Como se puede observar en la Figura 7.7, se ha creado una clase abstracta *Comparador*, la cual contiene el método abstracto comparar.

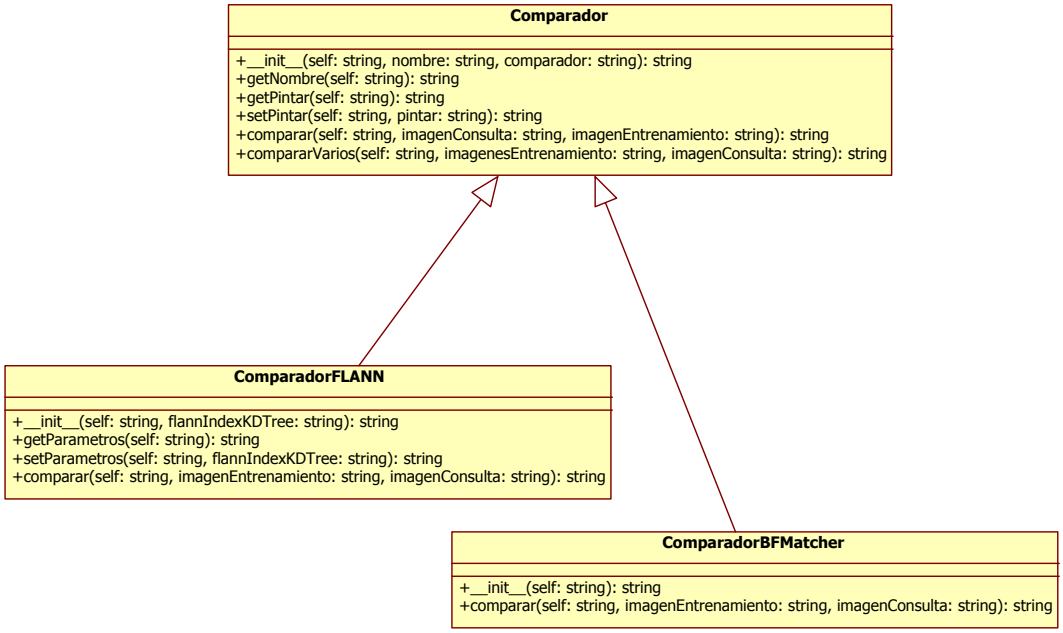


Figura 7.7: Diagrama de clases del algoritmo comparador.

Como se ha hecho con los algoritmos detectores, también se ha extendido la clase *Comparador* con los algoritmos que se desea probar, en este caso, se han creado las clases *ComparadorFlann* y *ComparadorBFMatcher*.

La implementación de estas dos clases también se simplifica gracias la librería OpenCV. Para el algoritmo FLANN se creó una instancia de él, mediante el objeto *cv2.FlannBasedMatcher* y, posteriormente, se buscan las coincidencias llamando al método *knnMatch*, que devuelve un vector con todos los descriptores comunes. Por otro lado, para *BFMatcher* se creó una instancia del objeto *cv2.BFMatcher* y, a continuación, se llamó al método *match* que, al igual que el anterior, devuelve un vector con los descriptores comunes entre las imágenes.

7.2.2. Proceso de Reconocimiento

Hasta aquí, se ha explicado el proceso por separado, pero ahora se pasa a la parte de unir estos algoritmos para obtener un algoritmo reconocedor de imágenes.

Siguiendo las pautas de diseño del software que se han seguido en la implementación de los algoritmos de detección y comparación de detectores, se ha construido un objeto *Reconocedor*, el cual será el encargado de realizar el proceso de reconocimiento, y debe recibir como parámetros: un objeto correspondiente a la base de datos, un detector, un detector más estricto y un comparador.

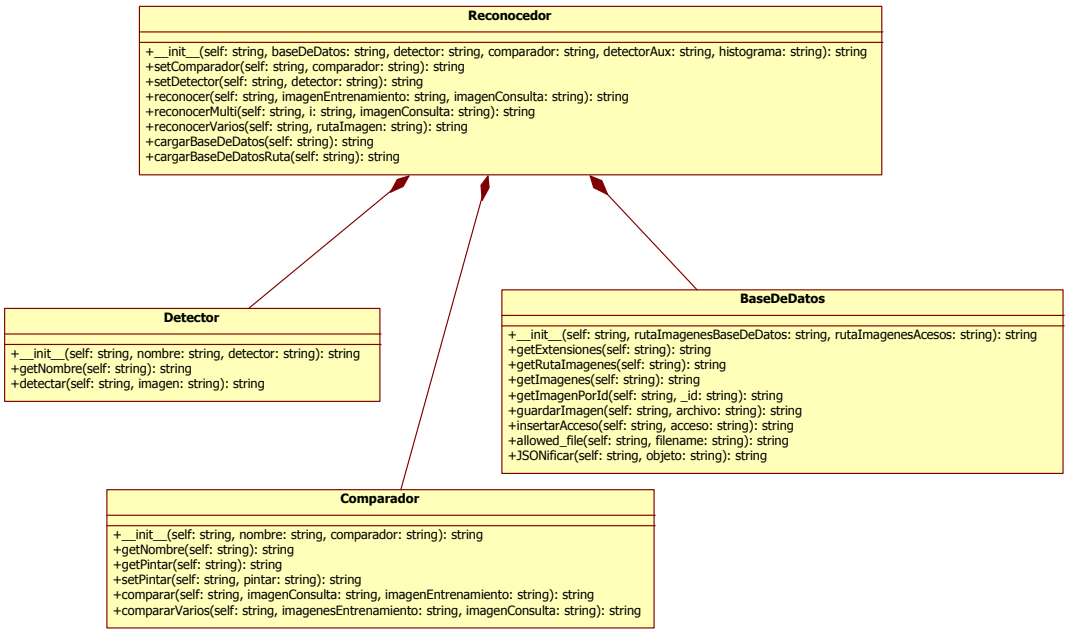


Figura 7.8: Diagrama de clases del algoritmo reconocedor.

Como se puede ver en la Figura 7.8, la clase *Reconocedor* contiene varios métodos para realizar su cometido. El método reconocer es el que realiza el proceso comparando dos imágenes y devolviendo un vector con los puntos que tienen en común (Código 7.6). En cambio, el método más interesantes es el llamado reconocerVarios, ya que es el que busca una imagen en la base de datos.

Código 7.6: Fragmento del método *reconocer*.

```
1  for i in range(len(imagenesEntrenamiento.values())):
2      if self.usarHilos:
3          thread = HiloComparador(i, "Hilo Comparador", i, imagenConsulta, self)
4          threads.append(thread)
5          thread.start()
6      else:
7          self.reconocer(i, imagenConsulta)
8
9  if self.usarHilos:
10     self.pool.map(self.reconocerMulti, listaArgs)
11     for t in threads:
12         t.join()
```

Para realizar este proceso con éxito, se han realizado los siguientes 5 pasos:

- Reducir la imagen.
- Calcular descriptores.
- Comparar con la base de datos.

- Ordenar las coincidencias.
- Determinar el resultado.

En el primer paso, siguiendo la recomendación del tutor Raúl Montoliu, experto en visión, y después de aplicar diferentes pruebas se observó que reduciendo todas las imágenes al mismo tamaño, se obtienen mejores resultados, por lo tanto lo primero que se hace es reducir la imagen al mismo tamaño, se optó por 1000 píxeles de ancho y lo correspondiente de alto, ya que de esta la imagen no será ni muy grande (muchos descriptores), ni muy pequeña (pocos descriptores).

En segundo paso, se calculan los detectores de la imagen utilizando el detector SURF. Si como resultado de este cálculo no se obtiene un número mínimo de detectores, se vuelve a realizar un segundo análisis, pero esta vez con el detector más estricto. Si aun así no se llega al umbral establecido, el proceso termina alegando de que la imagen no tiene suficiente calidad. Este número mínimo, ha sido calculado a partir de la realización de pruebas con diferentes logotipos, los cuales suelen tener pocos descriptores. Como conclusión de estas pruebas se determinó que un buen número mínimo eran 50 descriptores.

Después,en el tercer paso, se realiza un recorrido a las imágenes de la base de datos, comparando cada una de ellas con la imagen que se desea encontrar y guardando los resultados en un vector, adaptado para ello.

Una vez obtenidos los resultados de las comparaciones en un vector, en el cuarto paso, este se ordena por el porcentaje correspondiente al número de coincidencias obtenidas, respecto al número de descriptores que contiene la imagen de la base de datos (la que esta está recortada). Con tal de afinar más en el reconocimiento, se ha establecido un porcentaje mínimo de coincidencias, el cual se ha calculado comparando todas las pruebas realizadas, y se ha determinado que para una imagen ser reconocida, deben coincidir, como mínimo, el 25 % de los descriptores.

Y, por último, el último paso, es determinar el resultado. Si después de este proceso queda alguna imagen o imágenes, el resultado del reconocimiento es la que se encuentra en la primera posición de este vector. Si no queda ninguna imagen, se determina que no se ha podido reconocer la imagen.

7.2.3. Paralelización del proceso

Una vez el algoritmo funciona de una manera efectiva, se buscaron maneras de mejorar su eficiencia. Mientras la base de datos contenga un número bajo de imágenes, con el algoritmo implementado en los apartados anteriores no habrá problema. Pero si la base de datos empieza a crecer, puede que se tarde un tiempo considerable en realizar el proceso. Es por esto que se ha añadido la funcionalidad necesaria para que este se ejecute de forma paralela.

Para ello, el método *reconocerVarios* en vez de llamar al método reconocer por cada imágen, arrancará un hilo, que será el encargado de realizar la tarea. Por tanto, se lanzarán tantos hilos como imágenes contenga la base de datos.

Para que desde los diferentes hilos se pueda comparar las imágenes, se ha añadido un método

llamado *reconocerMulti*, que viene a ser la versión paralela del utilizado anteriormente, reconocer. Además, para que estos hilos puedan almacenar los resultados de manera paralela, se ha creado un vector global, donde cada hilo guardará en una posición diferentes los resultados obtenidos. Estos resultados se guardan en forma de diccionario, es decir, por cada posición del vector, este contendrá un diccionario con la imagen analizada y las coincidencias obtenidas.

Después de lanzar los hilos, el mismo método *reconocedorVarios*, tiene que esperar a que estos terminen su ejecución, y entonces puede seguir los mismos pasos que el procedimiento anterior, pero desde el vector global.

7.3. Servidor

La parte del servidor del sistema está realizada en su totalidad con el lenguaje de programación Python, y se compone principalmente de cuatro módulos: servidor, modelo, base de datos y el reconocimiento, visto en la sección anterior.

7.3.1. Servicios REST

Gracias a la utilización de Python Flask, esta parte resultó una tarea sencilla, ya que simplemente con la utilización de anotaciones en los métodos conseguimos que estos se llamen al acceder a la URL insertada en la anotación.

Por ejemplo, para el servicio que devuelve el thumbnail de una imagen a partir de la id, al añadir la anotación app.route con la dirección a la que se deben realizar las peticiones, cuando se realicen estas, directamente se llamará a este método (Código 7.7). De esta manera tan sencilla se han elaborado el resto de servicios Rest.

Código 7.7: Servicio REST para obtener el thumbnail de una imagen.

```
1 @app.route('/thumbnail/<identificador>')
2 def getThumbnail(identificador):
3     ''' Servicio que recibe la id de una imagen y devuelve la imagen
4         thumbnail correspondiente. '''
5
6     ruta = os.path.join(RUTA_IMAGENES_BASEDEDATOS, identificador,
7         'thumbnail.jpg')
8
9     return send_file(ruta, mimetype='image/jpg')
```

Además de este servicio para obtener el *thumbnail*, se ha implementado el servicio detrás del cual está la principal función del sistema, la de reconocer una imagen, al cual se le ha llamado *procesarImagen*. Este recibe una imagen y conecta con el módulo de reconocimiento para iniciar este proceso, y una vez se obtiene el resultado de este, se devuelve el resultado en forma de JSON.

7.3.2. Modelo

Para trabajar directamente con objetos se ha elaborado un módulo correspondiente al modelo del sistema. Este consiste de una serie de clases correspondientes a los objetos con los que se trabaja a lo largo de la implementación del servidor.

Los principales objetos son las imágenes. Para estas, se ha diseñado, como se pudo ver en la parte de diseño, una jerarquía en la cual existe una clase abstracta *Imagen* y un clase por cada tipo de imagen, la cual extiende a la clase abstracta (Figura 7.9).

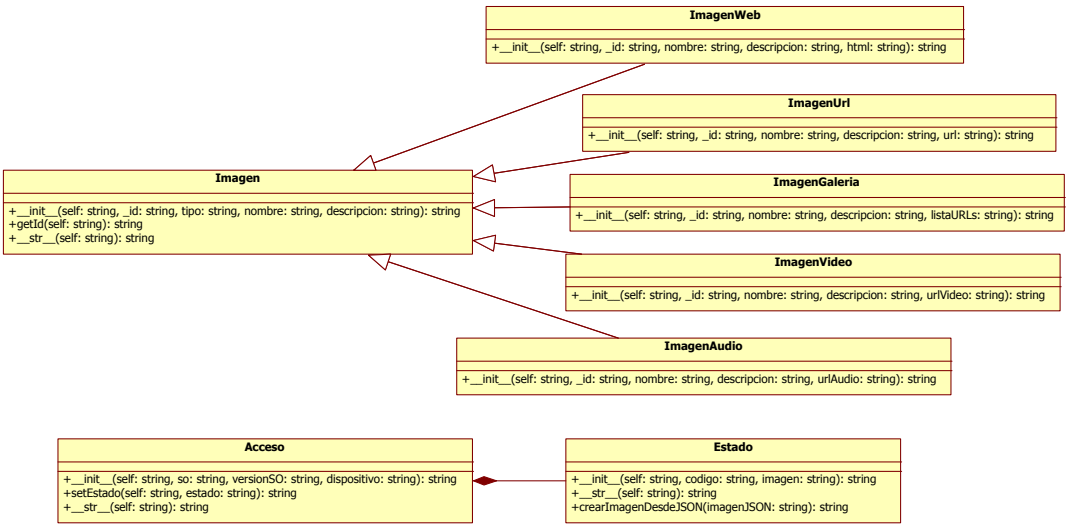


Figura 7.9: Diagrama de clases del modelo contenido en el servidor.

Por otro lado, se han implementado dos objetos más, llamados Estado y Acceso. El estado se utiliza para almacenar y devolver el resultado del análisis. Mientras que, el acceso se utiliza para almacenar estadísticas en la base de datos de las peticiones realizadas al servidor, además este contiene el estado resultante de esta.

Por último, en este módulo también se han implementado los métodos necesarios para parsear estos objetos al formato JSON y viceversa, cosa que es necesaria para almacenarlos en la base de datos MongoDB y para enviarlos al dispositivo móvil a través de los servicios Rest.

7.3.3. Base de Datos

Este módulo del sistema, se ha diseñado únicamente para conectar con la base de datos. En él, se ha implementado un objeto, llamado *BaseDeDatos*, el cual establece una conexión con la base de datos MongoDB.

Además, el objetivo de este, es actuar como pasarela entre el resto de la aplicación y la base de datos, de manera que contiene los métodos de consulta e inserción necesarios para realizar las operaciones a través de este .

Código 7.8: Clase correspondiente al objeto *BaseDeDatos*, utilizado para la conexión con la base de datos.

```
1  class BaseDeDatos(object):
2      ''' Objeto para gestionar la comunicacion con la base de datos
3      MongoDB. '''
4      def __init__(self, rutaImagenesBaseDeDatos, rutaImagenesAcesos):
5          self.connection = Connection()
6          self.db = self.connection.ServidorReconocimientoIMG
7          self.rutaImagenesBaseDeDatos = rutaImagenesBaseDeDatos
8          self.rutaImagenesAcesos = rutaImagenesAcesos
9          self.EXTENSIONES = set(['png', 'jpg', 'jpeg', 'gif'])
10
11     def getImagenPorId(self, _id):
12         ''' Devuelve una objeto de tipo Imagen con la id especificada. '''
13         return crearImagenDesdeJSON(self.db.imagenes.find_one({'_id':
14             ObjectId(_id)}))
```

Para realizar dicha conexión, se ha utilizado la librería PyMongo, la cual contiene las herramientas necesarias para conectar con una base de datos MongoDB desde el lenguaje Python. Esta, además de ser simple, es el método recomendado por los creadores de MongoDB para realizar esta conexión. http://api.mongodb.org/python/current/

En cuanto a la base de datos, simplemente se han creado dos colecciones: *Imágenes* y *Accesos*. En las imágenes se guardan los objetos del modelo correspondiente a las imágenes, los cuales, dependiendo del tipo tendrán unos campos o otros. Al ser una base de datos noSQL no hay ningún problema en hacerlo de esta manera. Por otro lado, en la colección *Accesos* se guardarán los objetos de tipo *Accesos*, que a su vez contienen un objeto estado.

7.3.4. Puesta en marcha

Cuando el servidor arranca, se crea un objeto *BaseDeDatos*, uno del tipo *Comparador* y otro del tipo *Detector*, con estos tres se crea el objeto *Reconocedor* que será el que se utilice en las funciones del sistema.

Al crearse dicho objeto, se calculan los detectores de todas las imágenes de la base de datos y se cargan en memoria, con lo se consigue por una parte acelerar el proceso de reconocimiento, pero por otra también reducir el coste de almacenamiento de la base de datos, ya que puede no resultar factible, teniendo en cuenta que almacenar en la base de datos supone mucho mayor coste que almacenar las imágenes en sí, y debido a la gran capacidad de memoria RAM existente hoy en día, esto no supone un problema para el servidor.

Por último, para ejecutar el programa constantemente en el servidor de una manera sencilla, siguiendo la recomendación del experto de la empresa, se ha utilizado la herramienta forever, mediante la cual se consigue que un programa se ejecute continuamente, sin necesidad de crear un demonio específico [28].

En el siguiente capítulo, se presentan los resultados obtenidos en las diversas pruebas realizadas con los algoritmos de detección y comparación de descriptores disponibles en la librería OpenCV, con el objetivo de obtener la configuración que mejores resultados obtiene con las imágenes de prueba del proyecto

Para la realización de dichas pruebas se ha elaborado un conjunto de imágenes de prueba, con las que poder observar los diferentes comportamientos de los algoritmos. Teniendo en cuenta que la aplicación se podrá utilizar en dispositivos con el sistema operativo Android, los cuales pueden tener diferentes cámaras con distintas resoluciones, se ha realizado para cada imagen, la misma fotografía desde tres dispositivos con resoluciones de 3, 5 y 8 megapixels. Las características de estos las podemos ver en la Tabla 7.1.

Dispositivo	**Cámara**
ZTE Grand X Quad V987	8 MP, 3264 x 2448 pixels
Samsung I8190 Galaxy S III mini	5 MP, 2592 x 1944 pixels
Samsung Galaxy Young S6310	3.15 MP, 2048 x 1536 pixels

Tabla 7.1: Características de los dispositivos utilizados para las pruebas (Fuente: *GSMArena*).

Además de esto, para someter en un mayor riesgo de equivocación a los algoritmos, en el conjunto de imágenes elegidas existen algunas muy parecidas, imágenes con mucho detalle e imágenes con poco detalle. El conjunto de imágenes elegidas para las pruebas de este documento se pueden ver en la Figura 7.10.

Figura 7.10: Conjunto de imágenes utilizadas para las pruebas.

Para todas las pruebas se han utilizado dos imágenes a comparar con el resto de imágenes del conjunto de pruebas. La primera, es la fotografía realizada al libro "Android 4 de Reto Meier", con una resolución de 8 *MegaPixeles*, ya que esta es, de las imágenes elegidas, la que más puntos descriptores puede contener. Por otro lado, la segunda imagen corresponde a la fotografía realizada al logo de la promoción "Por 1 café al día", con una resolución de 3 *MegaPixeles*, ya que esta es la que menos descriptores debería contener.

7.4. Parámetros por defecto

En esta primera prueba se compara el comportamiento por defecto de los tres algoritmos detectores de imagen: SURF, SIFT y ORB, con sus valores por defecto. Para ello se han comparado las dos imágenes elegidas con estos algoritmos.

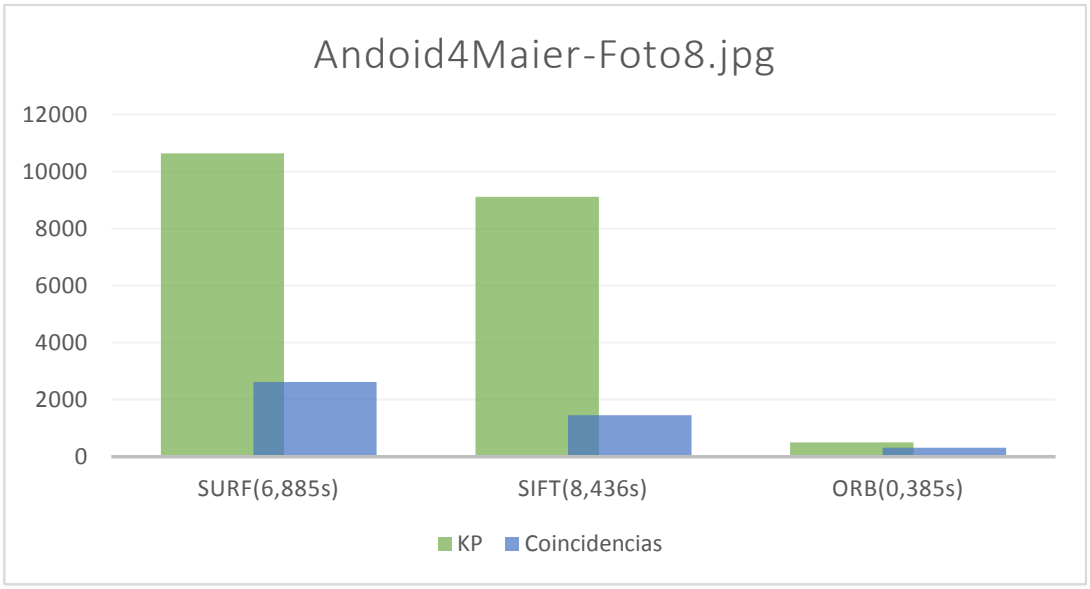


Figura 7.11: Comparación de algoritmos SURF, SIFT y ORB con la imagen *Android4Maier-Foto8*.

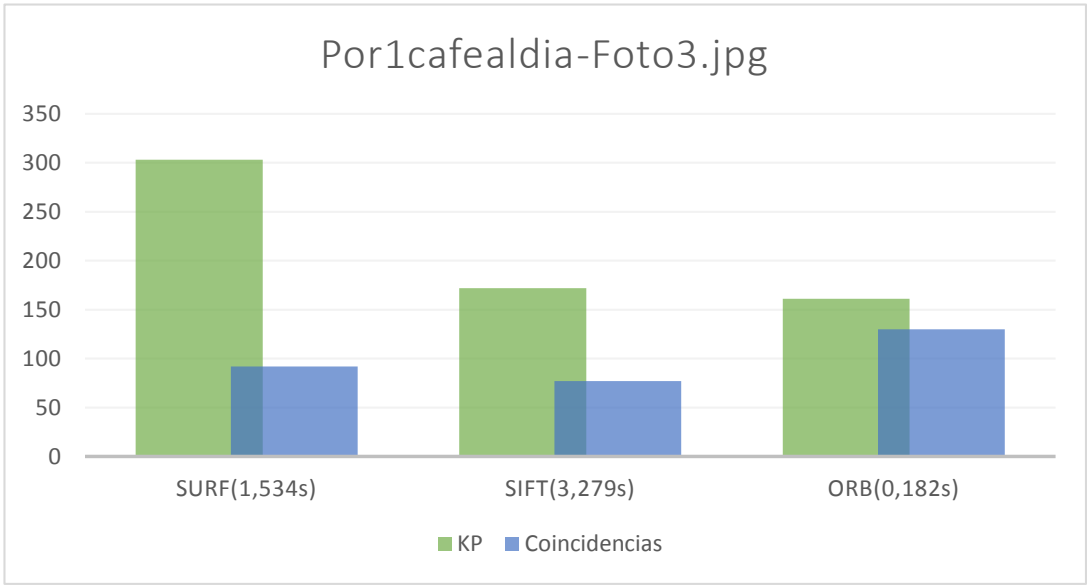


Figura 7.12: Comparación de algoritmos SURF, SIFT y ORB con la imágenes *Por1cafealdia-Foto3*.

En los gráfico de las figuras 7.11 y 7.12, se puede ver el resultado de esta primera prueba.

Como conclusión de esta, indicar que parece que el detector ORB es el que obtiene un mayor número de coincidencias entre la fotografía y la imagen de la base de datos, aunque es cierto que el total de descriptores que obtiene es menor.

Una vez realizadas las pruebas comparando dos imágenes con los diferentes algoritmos, vienen las pruebas de estos algoritmos, pero esta vez, comparando una imagen con un conjunto de imágenes, para ver cómo se comporta cada algoritmo en estos casos. Para ello, se comparan las mismas dos imágenes que en las pruebas anteriores, pero ahora contra todo el conjunto de imágenes de prueba.

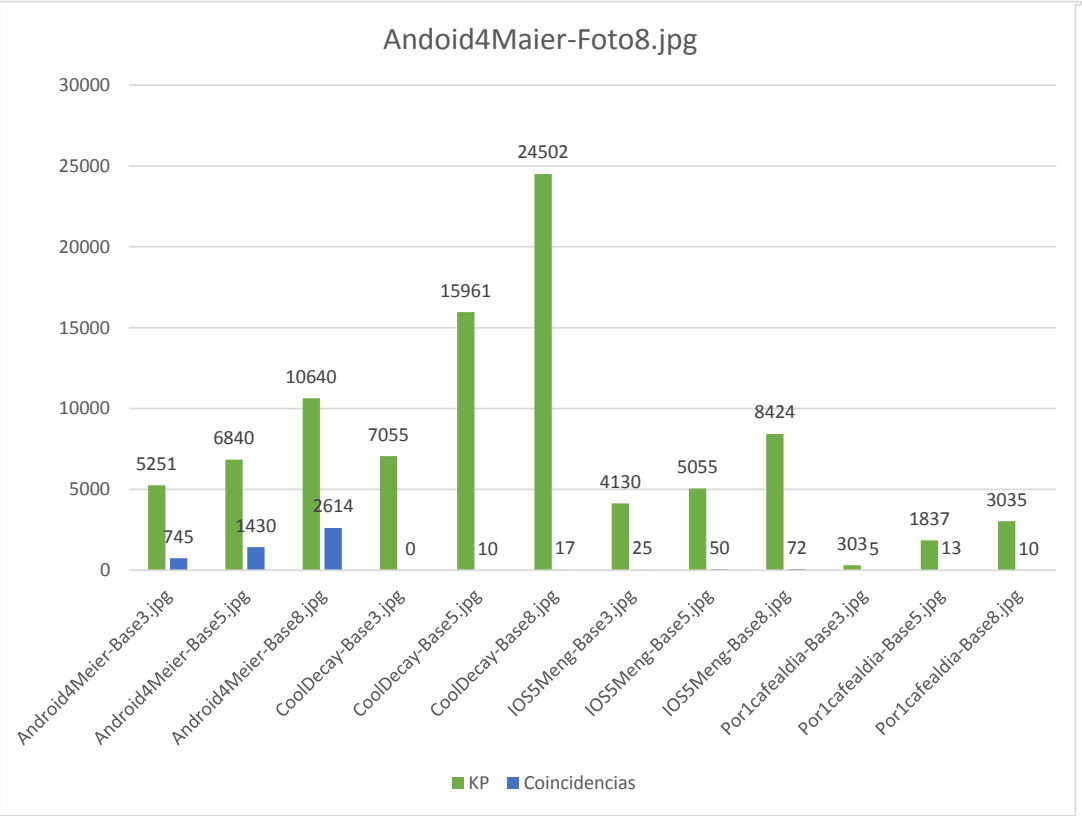


Figura 7.13: Descriptores y coincidencias resultantes de comparar la imagen *Android4Maier-Foto8* con el resto del conjunto de datos de prueba utilizando el algoritmo SURF.

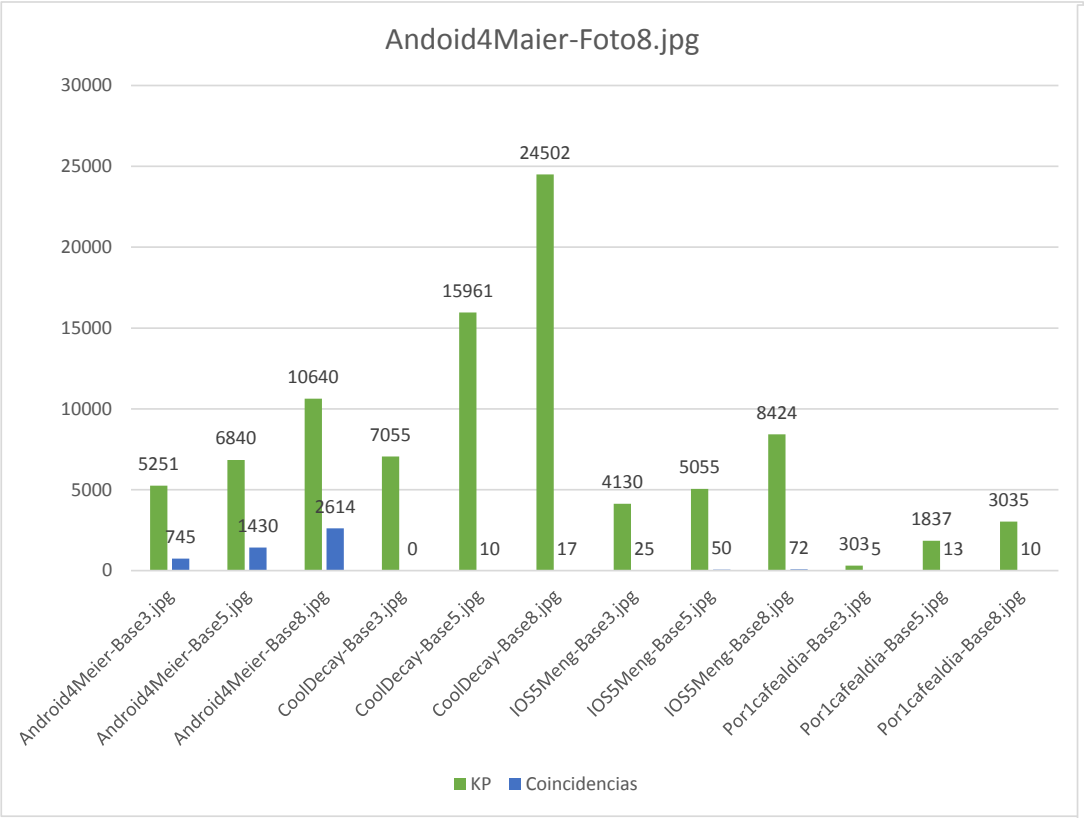


Figura 7.14: Descriptores y coincidencias resultantes de comparar la imagen *Por1cafealdia-Foto3* con el resto del conjunto de datos de prueba utilizando el algoritmo SURF.

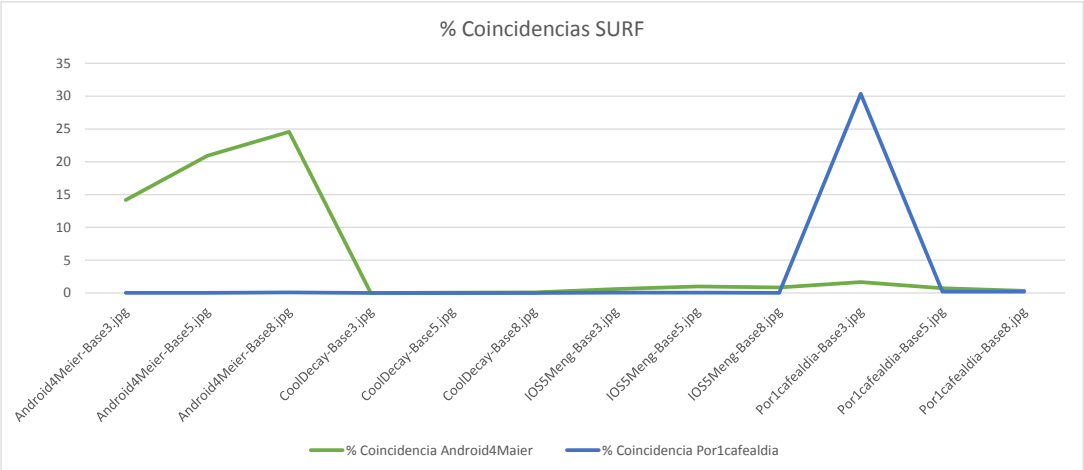


Figura 7.15: Comparación de las coincidencias obtenidas para las imágenes *Android4Maier-Foto8* y *Por1cafealdia-Foto3* con el algoritmo SURF.

En los gráficos de las Figuras 7.13 y 7.14, se puede ver la primera de estas pruebas, la correspondiente al algoritmos SURF. En estos se puede observar que se detectan muchos descriptores, pero que en los dos casos se cumplen las expectativas y se encuentran un mayor número de coincidencias en las imágenes que son semejantes a las que se desea encontrar. En

el caso de *Android4Meier*, se puede apreciar que las coincidencias también se aproximan en las resoluciones inferiores. Además, en el gráfico de la Figura 7.15 se puede ver el resultado conjunto de las dos imágenes, donde se diferencia claramente que las coincidencias aumentan con las imágenes adecuadas.

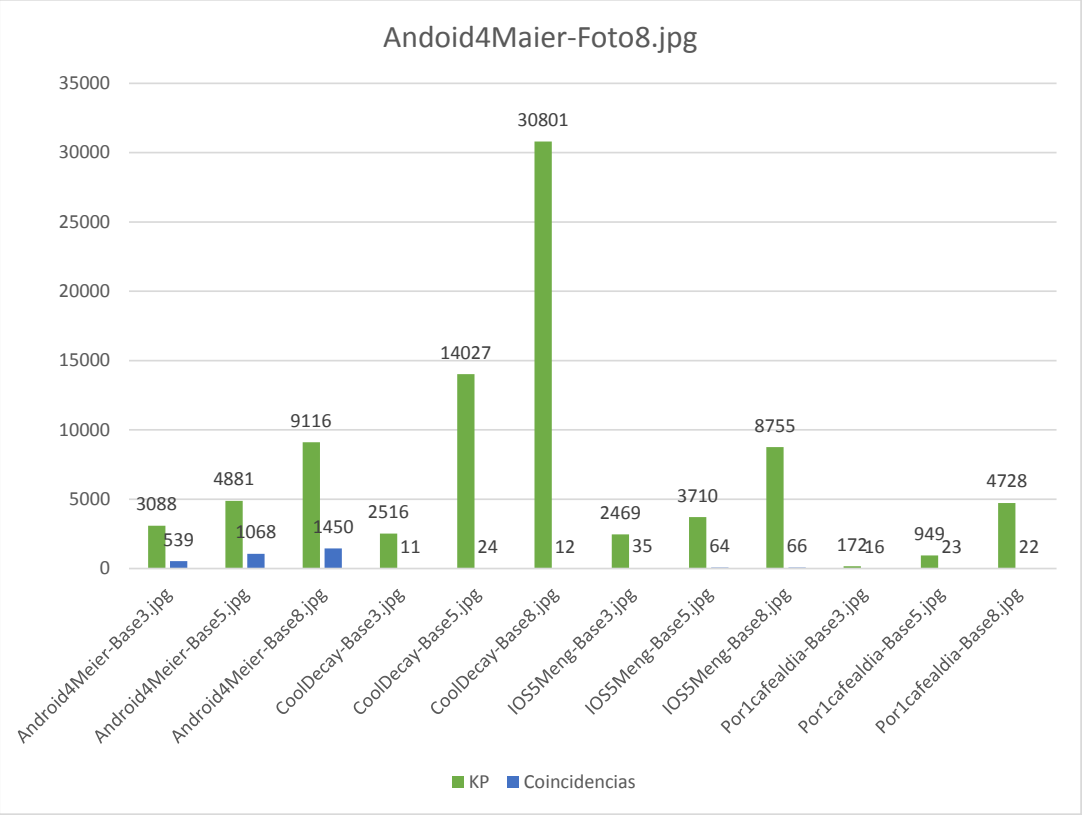


Figura 7.16: Descriptores y coincidencias resultantes de comparar la imagen *Android4Maier-Foto8* con el resto del conjunto de datos de prueba utilizando el algoritmo SIFT.

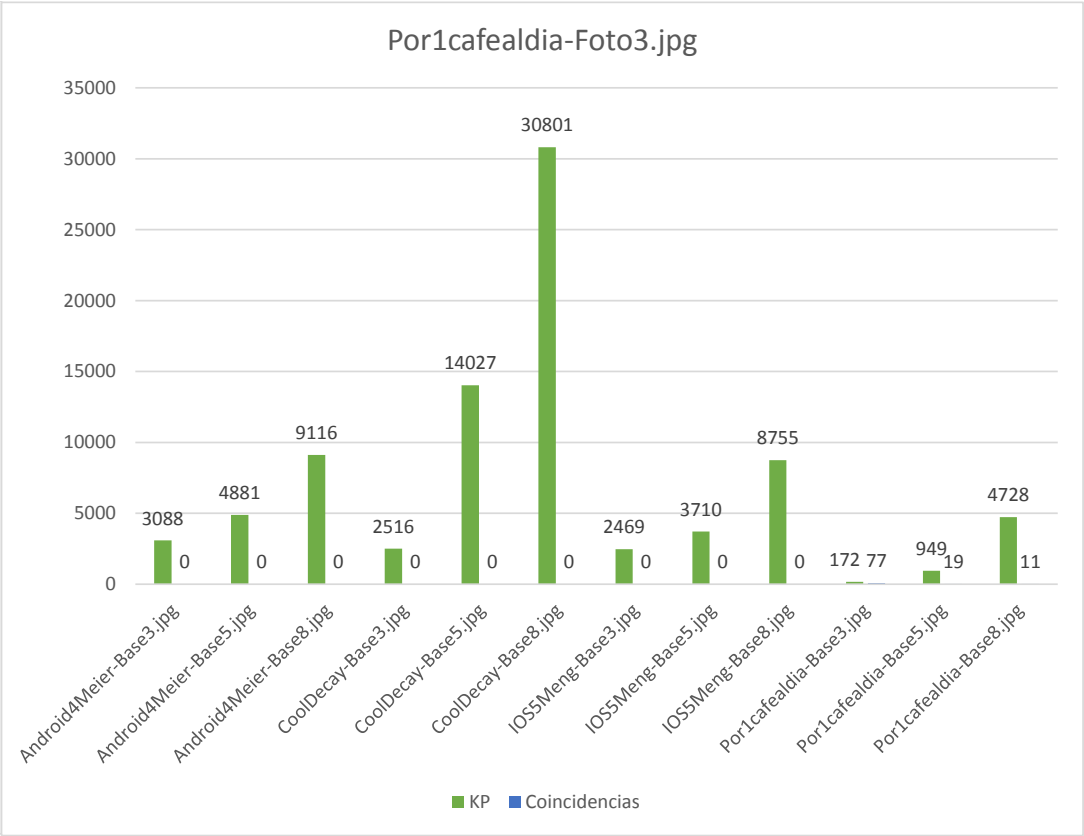


Figura 7.17: Descriptores y coincidencias resultantes de comparar la imagen *Por1cafealdia-Foto3* con el resto del conjunto de datos de prueba utilizando el algoritmo SIFT.

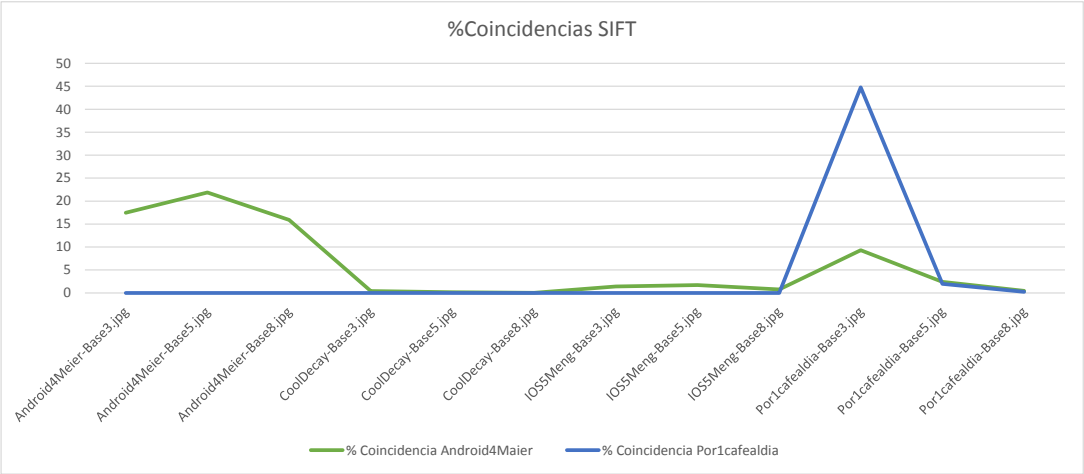


Figura 7.18: Comparación de las coincidencias obtenidas para las imágenes *Android4Maier-Foto8* y *Por1cafealdia-Foto3* con el algoritmo SIFT.

En cuanto a la prueba correspondiente al algoritmos SIFT, la de los gráficos de las Figuras 7.16 y 7.17, los resultados obtenidos son bastantes parecidos a los que se obtuvieron con el algoritmos SURF, respecto a la imagen *Por1cafealdia*. Pero, por lo que respecta a la imagen *Android4Meier*, se puede apreciar una bajada en las coincidencias en las imágenes semejantes, y una pequeña subida en las imágenes diferentes. En el gráfico de la Figura 7.18, se puede observar de una manera más clara esta bajada de coincidencias.

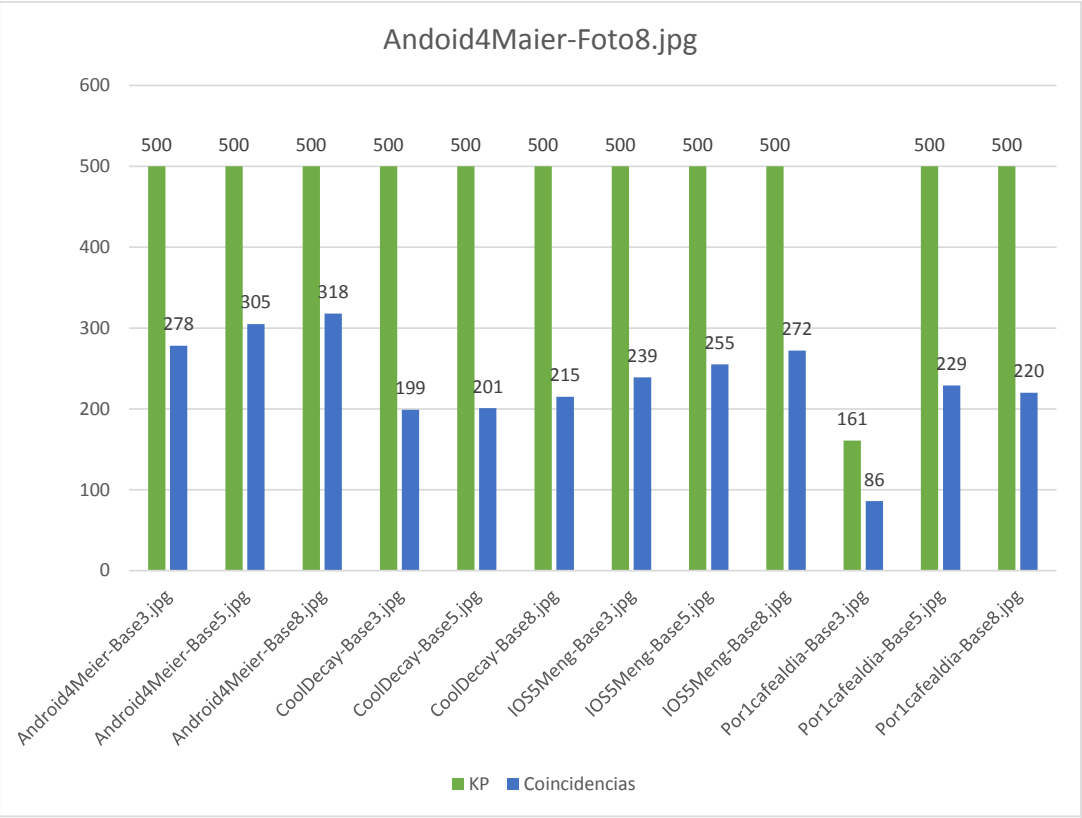


Figura 7.19: Descriptores y coincidencias resultantes de comparar la imagen *Android4Maier-Foto8* con el resto del conjunto de datos de prueba utilizando el algoritmo ORB.

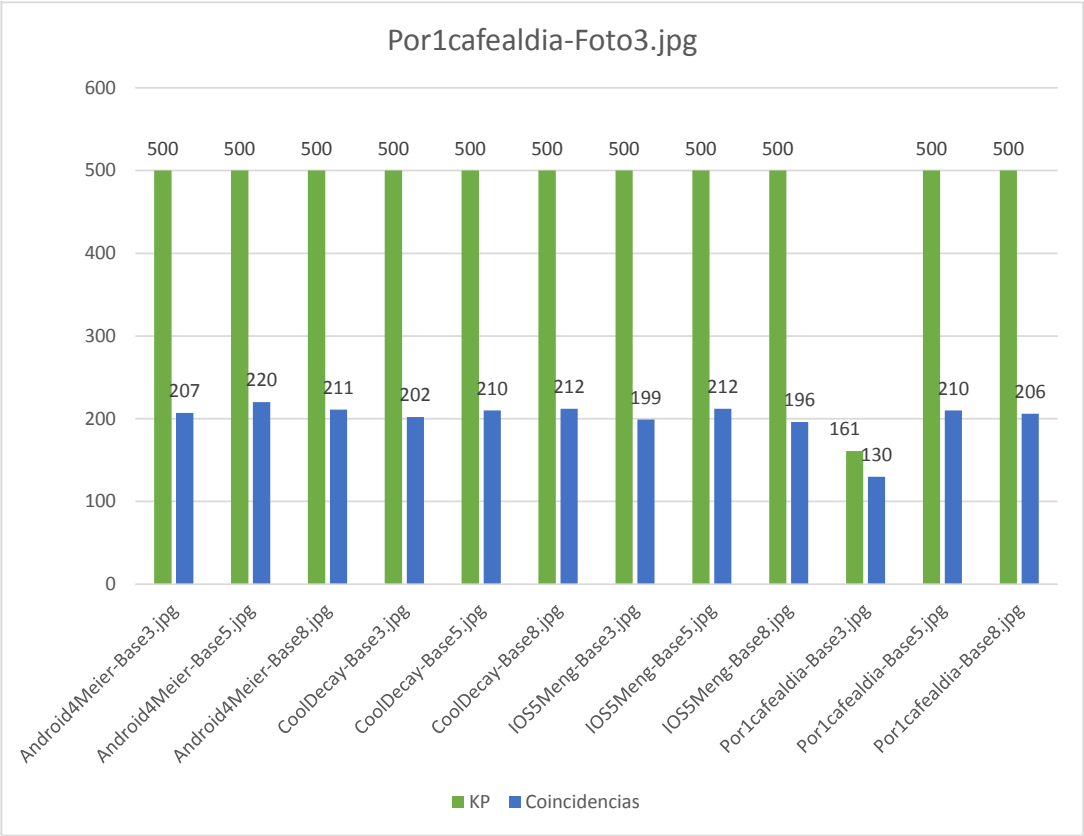


Figura 7.20: Descriptores y coincidencias resultantes de comparar la imagen *Por1cafealdia-Foto3* con el resto del conjunto de datos de prueba utilizando el algoritmo ORB.

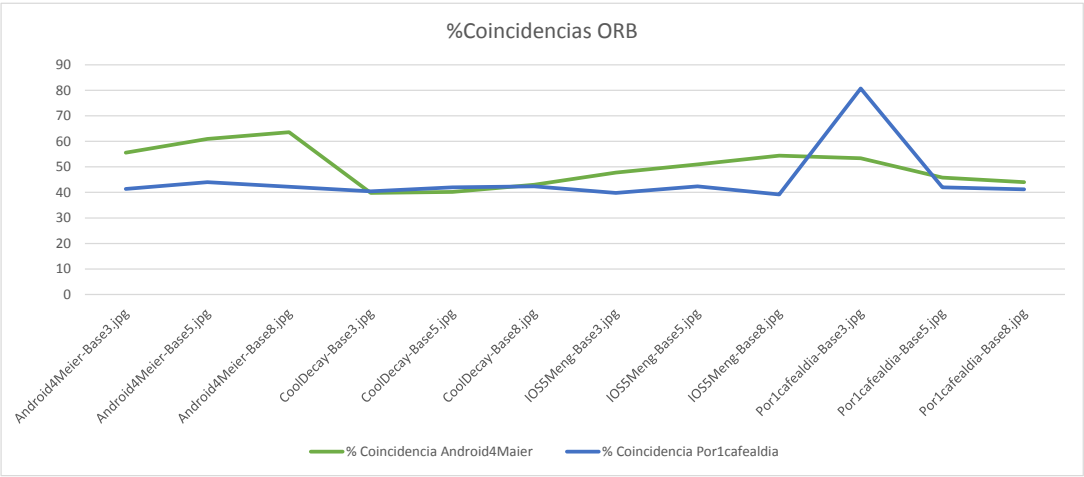


Figura 7.21: Comparación de las coincidencias obtenidas para las imágenes *Android4Maier-Foto8* y *Por1cafealdia-Foto3* con el algoritmo ORB.

Por último, en los gráficos de las Figuras 7.19 y 7.20 se pueden apreciar las pruebas realizadas para el algoritmo ORB. El resultado de estas es algo contradictorio ya que, por una parte, sí que destacan las imágenes semejantes, pero por otra parte, el resto de imágenes obtiene un mayor

número de coincidencias, cosa que hace más difícil descartar imágenes. En el gráfico de la Figura 7.20, se puede apreciar como la línea se mantiene muy alta para las imágenes que no debería.

7.5. Parámetros algoritmo SURF

Con el objetivo de mejorar la comparación de imágenes, se han realizado una serie de pruebas ajustando algunos parámetros disponibles en el algoritmos SURF. Los parámetros de los que dispone SURF se pueden observar en la Tabla 7.2.

Parámetro	Valor por defecto
hessianThreshold	400
nOctaves	4
nOctaveLayers	2
extended	true
upright	false

Tabla 7.2: Valores por defecto del algoritmo SURF.

En este caso, se han realizado pruebas solo con algunos de ellos, los que se han considerado más importantes. Estos son los siguientes:

- hessianThreshold: Umbral para la detección de descriptores.
- nOctaves: Número de octavas que el detector utilizará.
- nOctaveLayers: Número de capas de octava por cada octava utilizada.

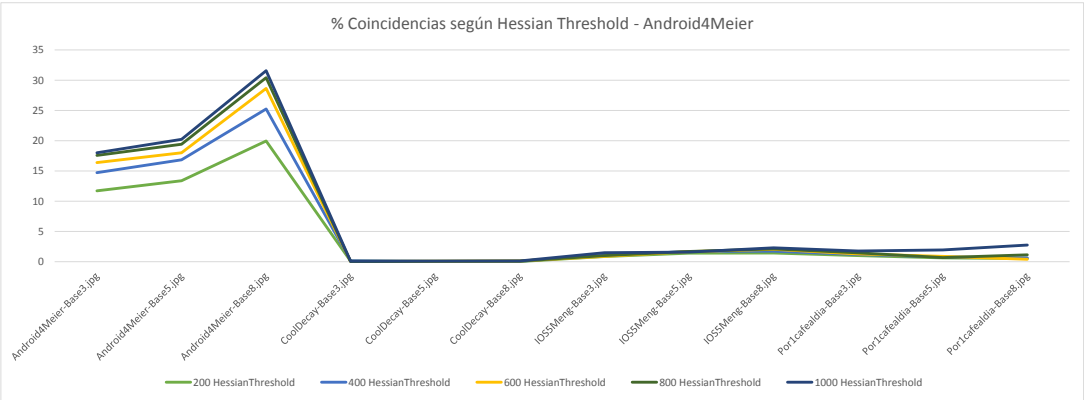


Figura 7.22: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *hessianThreshold*.

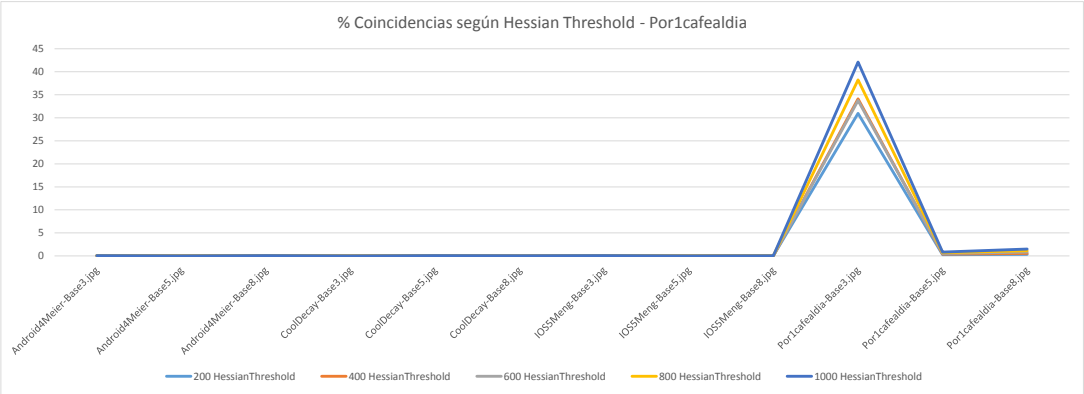


Figura 7.23: Comparación de las coincidencias obtenidas para la imagen *por1cafealdia-Foto3*, con diferentes valores de *hessianThreshold*.

En los gráficos de las Figuras 7.22 y 7.23 se pueden ver el resultado de las pruebas realizadas cambiando el valor del parámetro *hessianThreshold*. En las dos imágenes se puede observar que cuanto más alto es el valor, más coincidencias se obtienen. Pero cuando el valor es 1000, en la imagen *Android4Meier* se puede comprobar que empiezan a aumentar las coincidencias cuando no deben, por lo tanto, como conclusión se podría decir que el valor que mejor responde es 800.

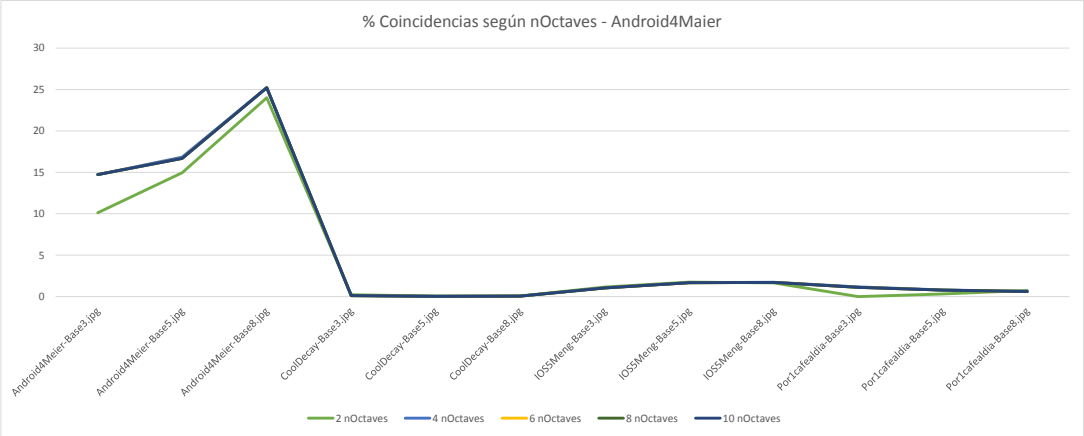


Figura 7.24: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *nOctaves*.

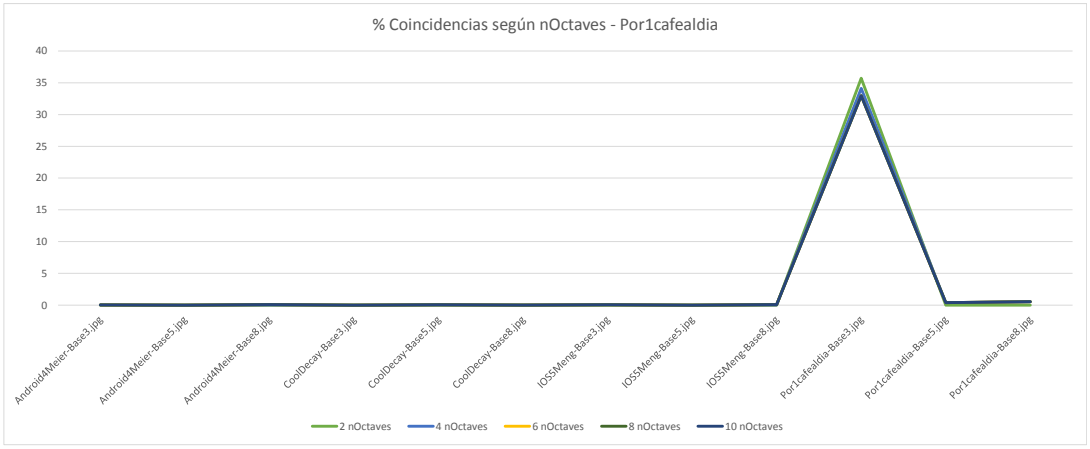


Figura 7.25: Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *nOctaves*.

En cuanto al parámetro *nOctaves*, en los gráficos de las Figuras 7.24 y 7.25, se puede ver el resultado de las pruebas. En este caso no se obtiene una gran diferencia con la variación de este parámetro. En los dos casos se podría decir que el mejor valor es 8, pero la diferencia respecto de los valores cercanos apenas se aprecia.

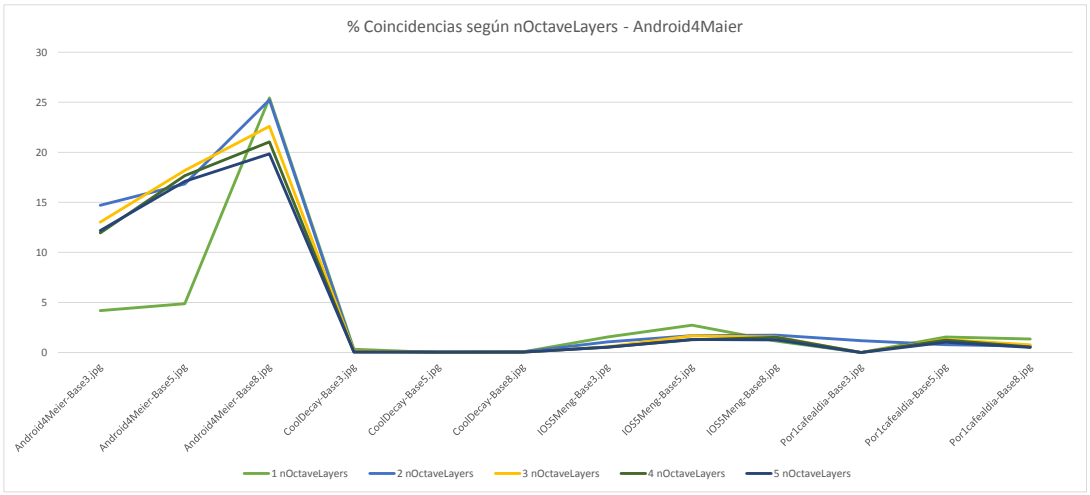


Figura 7.26: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *nOctaveLayer*.

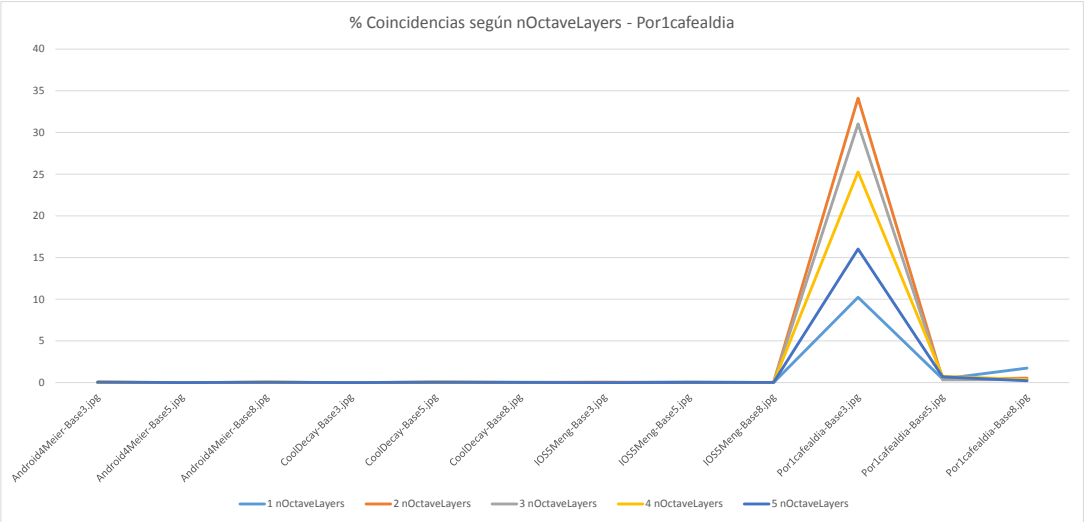


Figura 7.27: Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *nOctaveLayer*.

Para el parámetro *nOctaveLayers*, se puede observar en los gráficos de las Figuras 7.26 y 7.27, que pasa al contrario que en los parámetros anteriores, y cuanto más bajo es, mejores resultados se obtienen. Concretamente el valor más destacado de las pruebas es el 2.

7.6. Parámetros algoritmo SIFT

Al igual que en el algoritmo anterior, se han realizado pruebas con diferentes valores para los parámetros de SIFT. Los parámetros junto a sus valores por defecto se pueden ver en la Tabla 7.3.

Parámetro	**Valor por defecto**
nFeatures	0
nOctaveLayers	3
contrastThreshold	0.4
edgeThreshold	10
sigma	1.6

Tabla 7.3: Valores por defecto del algoritmo SIFT.

En este caso, las pruebas se han realizado cambiando los valores de tres parámetros. Estos son los siguientes:

- contrastThreshold: El umbral de contraste utilizado para filtrar las características débiles (con bajo contraste) en las regiones semi-uniformes.

- edgeThreshold: El umbral utilizado para filtrar las características con bordes similares.

- nOctaveLayer: Número de capas en cada octava.

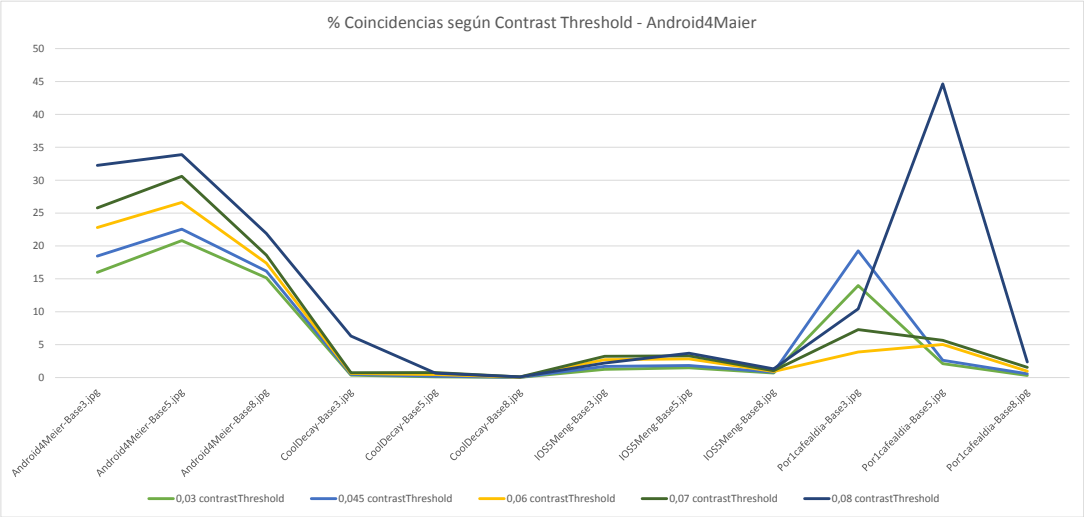


Figura 7.28: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *contrastThresold*.

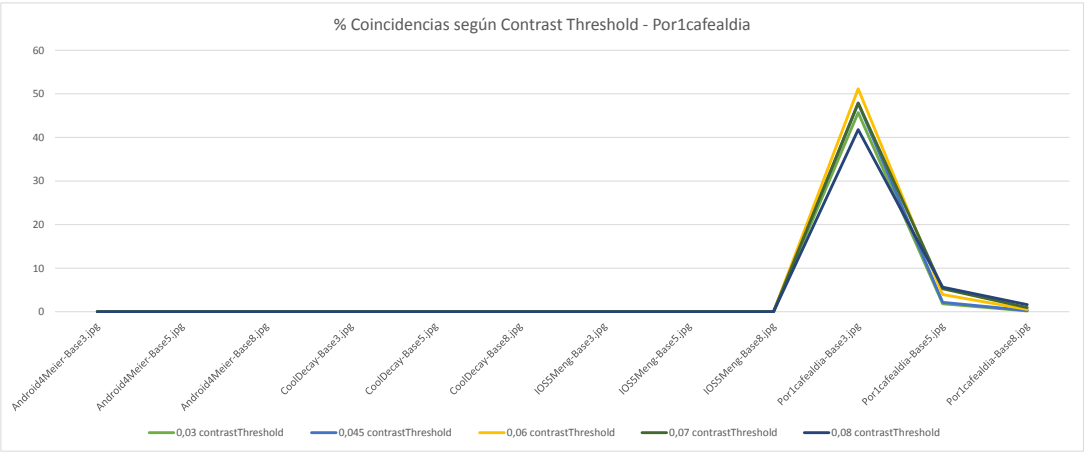


Figura 7.29: Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *contrastThreshold*.

En los gráficos de las Figuras 7.28 y 7.29 se pueden observar los resultados de las pruebas realizadas con el parámetro *contrastThreshold*. En estos, se observa que la variación de este parámetro se ve altamente reflejada en el número de coincidencias. En su valor máximo, saca un alto número de coincidencias para las imágenes semejantes, pero también para las que no lo son, por eso parece que la mejor opción es dejarlo entre 0.06 y 0.07.

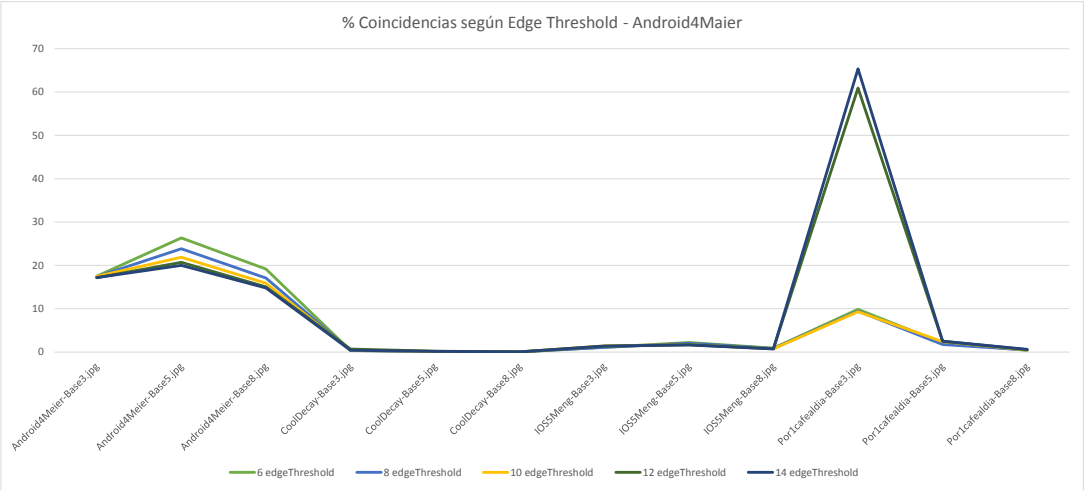


Figura 7.30: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *edgeThreshold*.

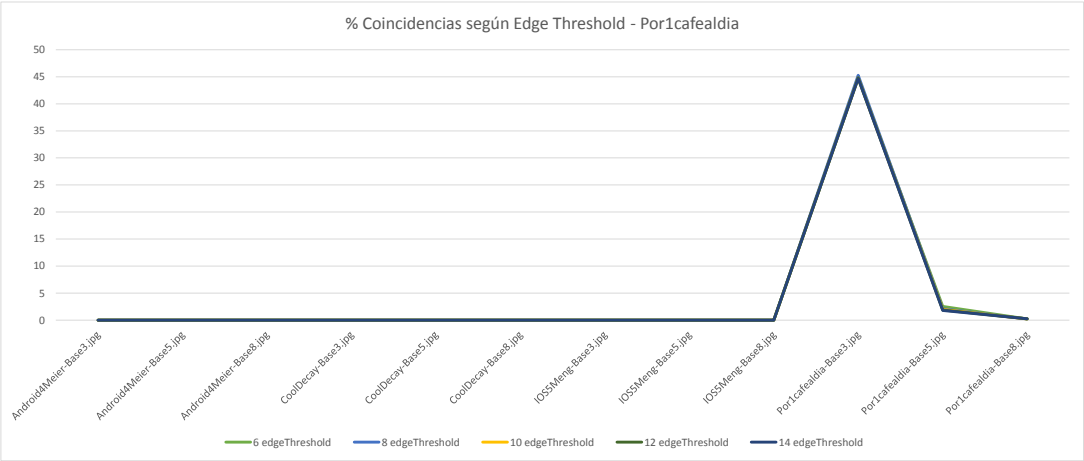


Figura 7.31: Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *edgeThreshold*.

Las siguientes pruebas se han realizado cambiando el valor del parámetro *edgeThreshold*, y se pueden ver en los gráficos de las Figuras 7.30 y 7.31. En lo referente a la imagen *Por1cafealdia* no se aprecia prácticamente las diferencias entre los diferentes valores, pero en cambio, en la imagen *Android4Meier*, se aprecia un alto factor erróneo a favor de las imágenes que no son semejantes a esta. Por lo tanto, se puede concluir que dicho parámetro debe contener un valor intermedio, en este caso 10.

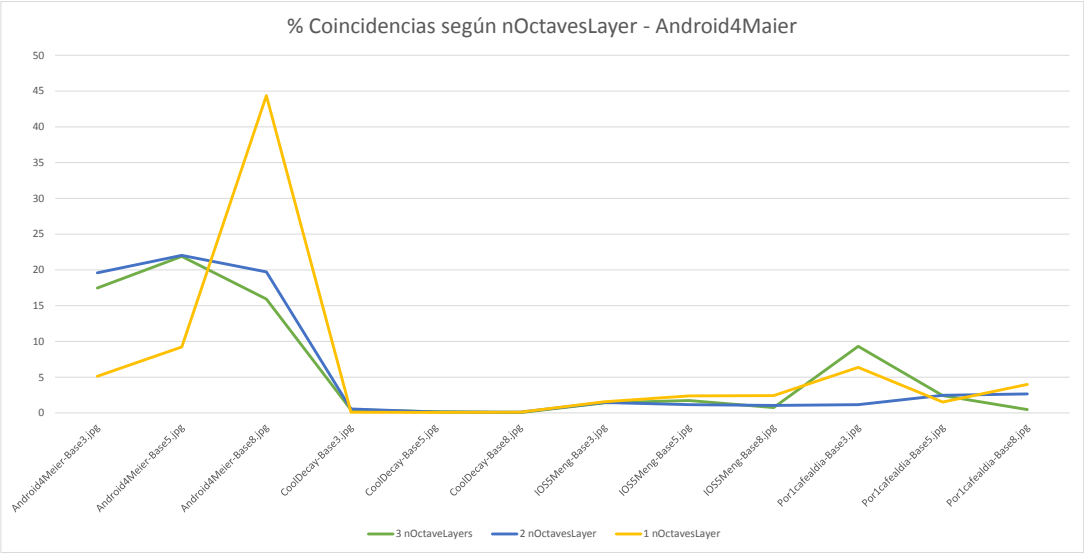


Figura 7.32: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *nOctaveLayer*.

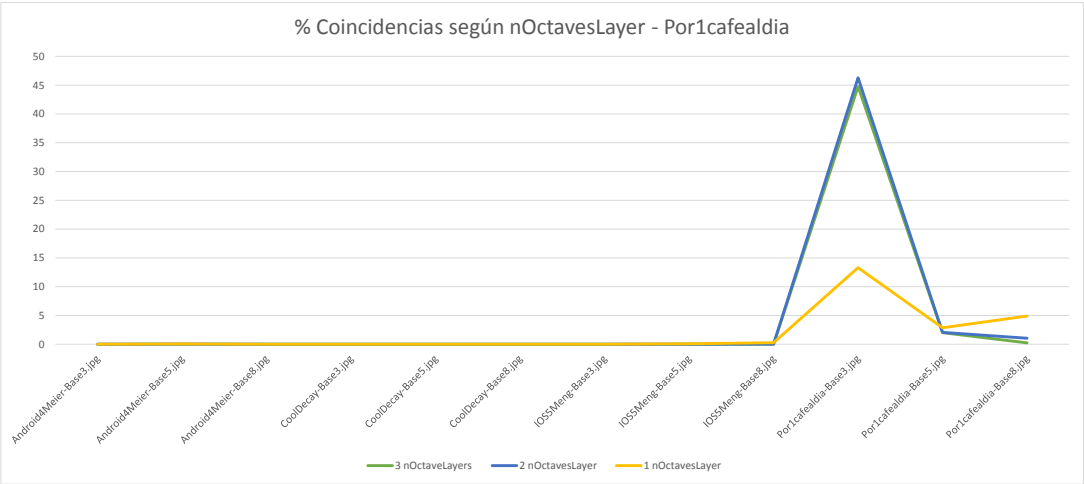


Figura 7.33: Comparación de las coincidencias obtenidas para la imagen *Poe1cafealdia-Foto3*, con diferentes valores de *nOctaveLayer*.

Por último, se hicieron pruebas referentes al parámetro *nOctaveLayer*, las cuales se pueden ver en los gráficos de las Figuras 7.32 y 7.33. En estas se puede observar que con el valor 1 saca más coincidencias para las imágenes semejantes, pero también saca más para las diferentes. Además, en la imagen *Por1cafealdia* no llega a sacar tantas como los otros valores. Por lo tanto, se ha llegado a la conclusión de que un parámetro intermedio es la mejor opción, en este caso 2.

7.7. Parámetros algoritmo ORB

Siguiendo la misma línea que los algoritmos anteriores, se han realizado pruebas con distintas valores para los parámetros del algoritmo ORB. En la Tabla 7.4 se pueden ver los parámetros de este con sus valores por defecto.

Parámetro	Valor por defecto
nFeatures	500
scaleFactor	1.2f
nlevels	8
edgeThreshold	31
firstLevel	0
WTA_ K	2
scoreType	ORB::HARRIS_ SCORE
patchSize	31

Tabla 7.4: Valores por defecto del algoritmo ORB.

En este caso los parámetros no se han estudiado en profundidad, ya que después de los resultados obtenidos en las pruebas con parámetros por defecto, se tiene poca esperanza en su uso.

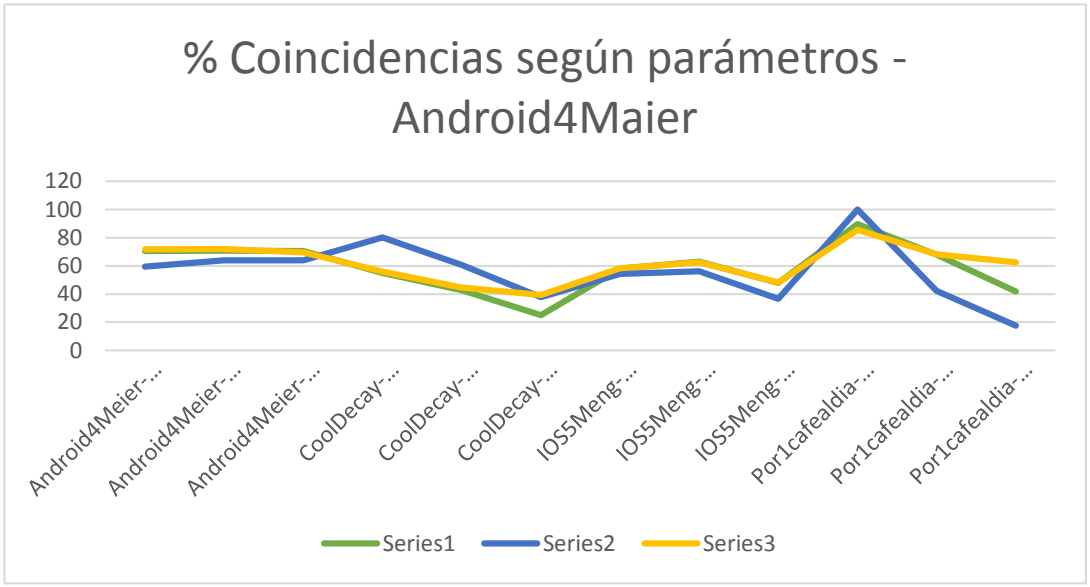


Figura 7.34: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores para sus parámetros.

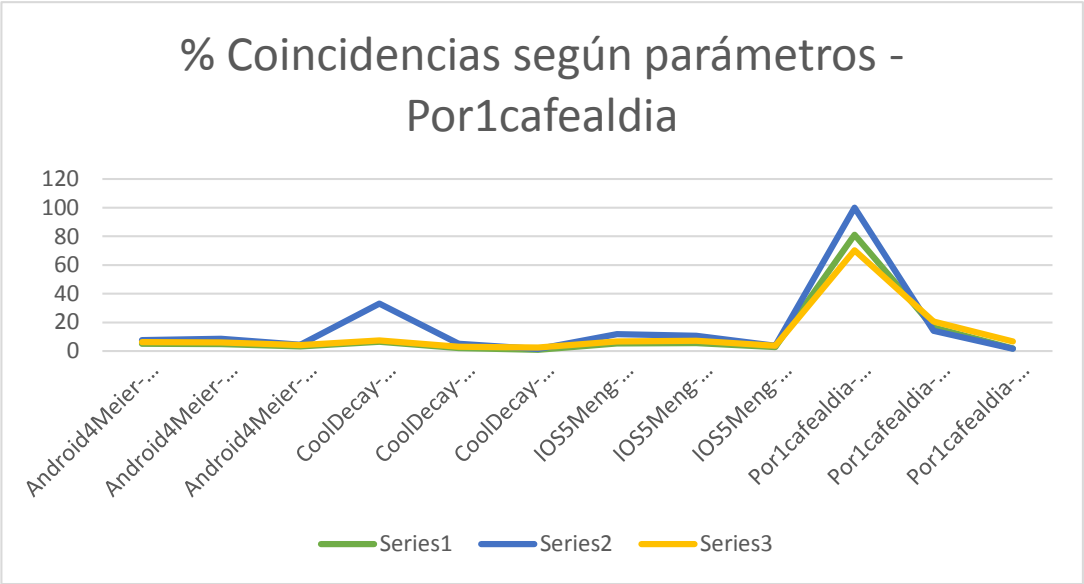


Figura 7.35: Comparación de las coincidencias obtenidas para la imagen *Pro1cafealdia-Foto3*, con diferentes valores para sus parámetros.

Para el algoritmo ORB, después de ver en las primeras pruebas con los parámetros por defecto, que los resultados no eran los esperados, no se hizo tanto hincapié en las pruebas de los parámetros y, simplemente, se realizaron pruebas con tres series de parámetros. En los gráficos de las Figuras 7.34 y 7.35, se pueden ver los resultados de estas pruebas, los cuales han servido para confirmar que el uso de ORB en este proyecto no es el más adecuado, ya que en los diferentes parámetros probados con las dos imágenes, siempre ha salido ganadora, en cuanto a coincidencias, la imagen *Por1cafealdia*. Además, se han obtenido bastantes coincidencias en las imágenes diferentes.

7.8. Parámetros algoritmo FLANN

Para el algoritmo comparador FLANN, también se han realizado pruebas con diferentes valores para sus parámetros, con el objetivo de que el número de coincidencias encontradas sea el adecuado. Los parámetros que se han tenido en cuenta son los siguientes:

- FLANN_ INDEX: Número de kd-trees que se utilizarán en paralelo.

- COEFICIENTE_ DISTANCIA: Distancia mínima que debe haber entre dos puntos, para considerarse como coincidencia.

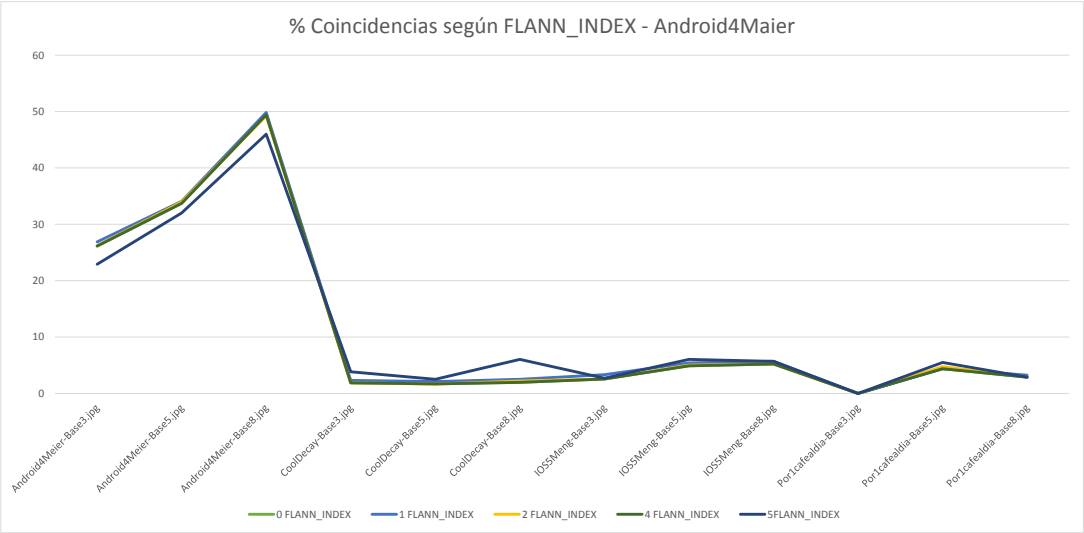


Figura 7.36: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *FLANN_ INDEX*.

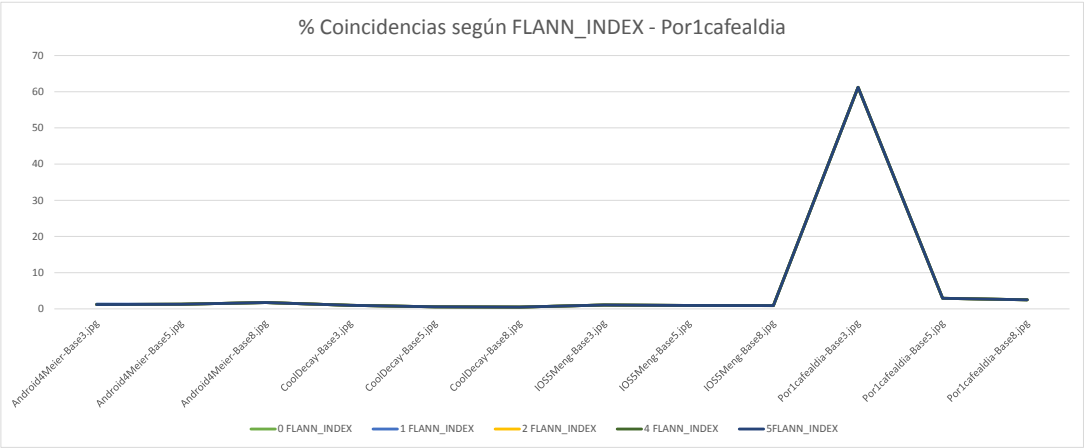


Figura 7.37: Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *FLANN_ INDEX*.

En los gráficos de las Figuras 7.36 y 7.36 se pueden ver los resultados de las pruebas realizadas sobre el parámetro *FLANN_ INDEX*. Para la imagen *Por1cafealdia* no se aprecian diferencias al cambiar los parámetros, mientras que, en la imagen *Android4Maier*, a pesar de que las diferencias son pocas, se puede concluir que los mejores resultados están entorno a los parámetros de 1 a 4.

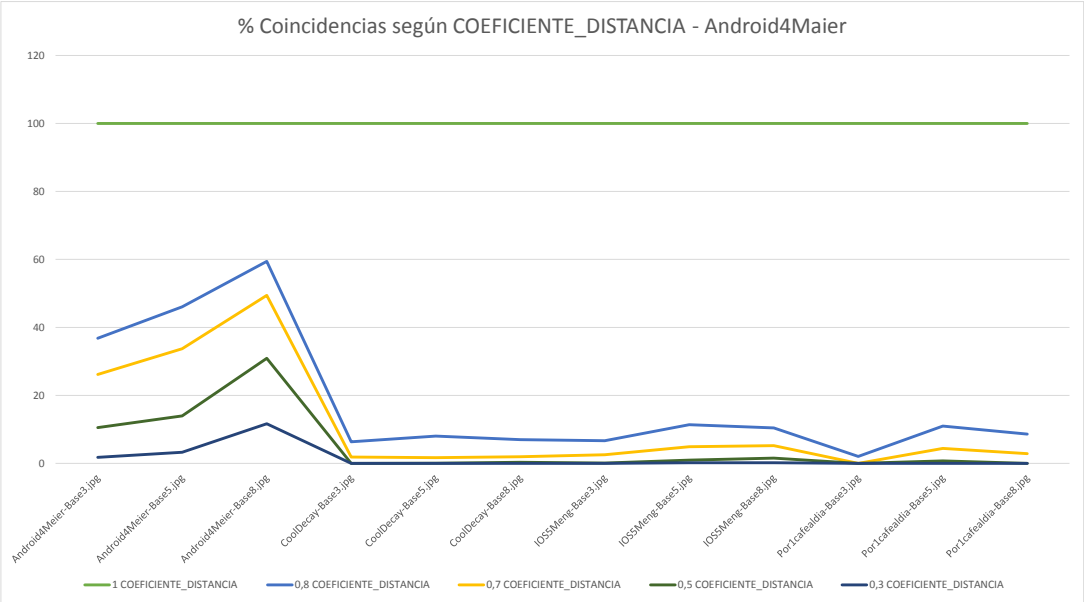


Figura 7.38: Comparación de las coincidencias obtenidas para la imagen *Android4Maier-Foto8*, con diferentes valores de *COEFICIENTE_ DISTANCIA*.

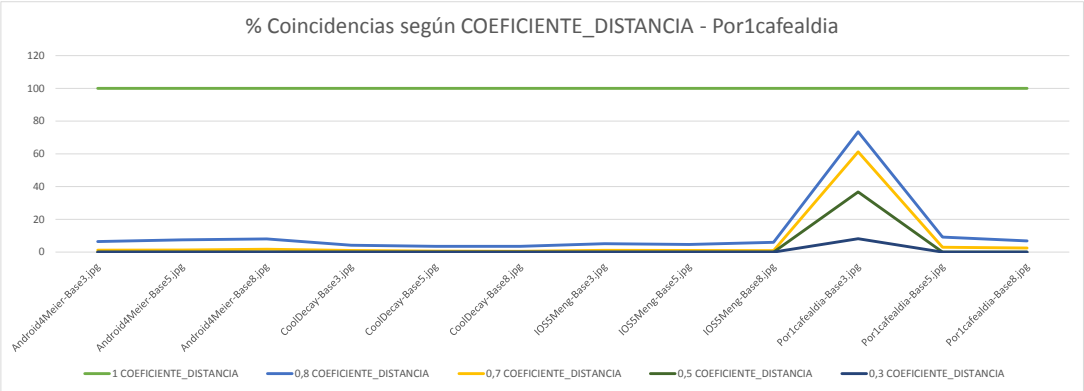


Figura 7.39: Comparación de las coincidencias obtenidas para la imagen *Por1cafealdia-Foto3*, con diferentes valores de *COEFICIENTE_ DISTANCIA*.

En cuanto a las pruebas realizadas con el valor COEFICIENTE_ DISTANCIA, las cuales se pueden ver en los gráficos de las Figuras 7.38 y 7.39. Se puede ver claramente, como es lógico, que a más distancia, más coincidencias. Viendo los resultados, parece ser que la distancia más adecuada es ente 0.7 y 0.8.

Capítulo 8

Resultado Final

En este capítulo se presenta el resultado final de la aplicación móvil. Para ello, se mostrarán capturas de pantalla de las diferentes partes de la aplicación, dividiéndolas en las distintas partes de que se compone esta.

Al igual que en el capítulo de diseño (Capítulo 6), antes de pasar a ver el resultado de la aplicación más profundamente, se presenta una visión general de esta, en la que se ha seguido el flujo que el usuario deberá realizar para ver los resultados del reconocimiento, utilizando las capturas finales que se explicarán más profundamente en las siguientes secciones (Figura 8.1).

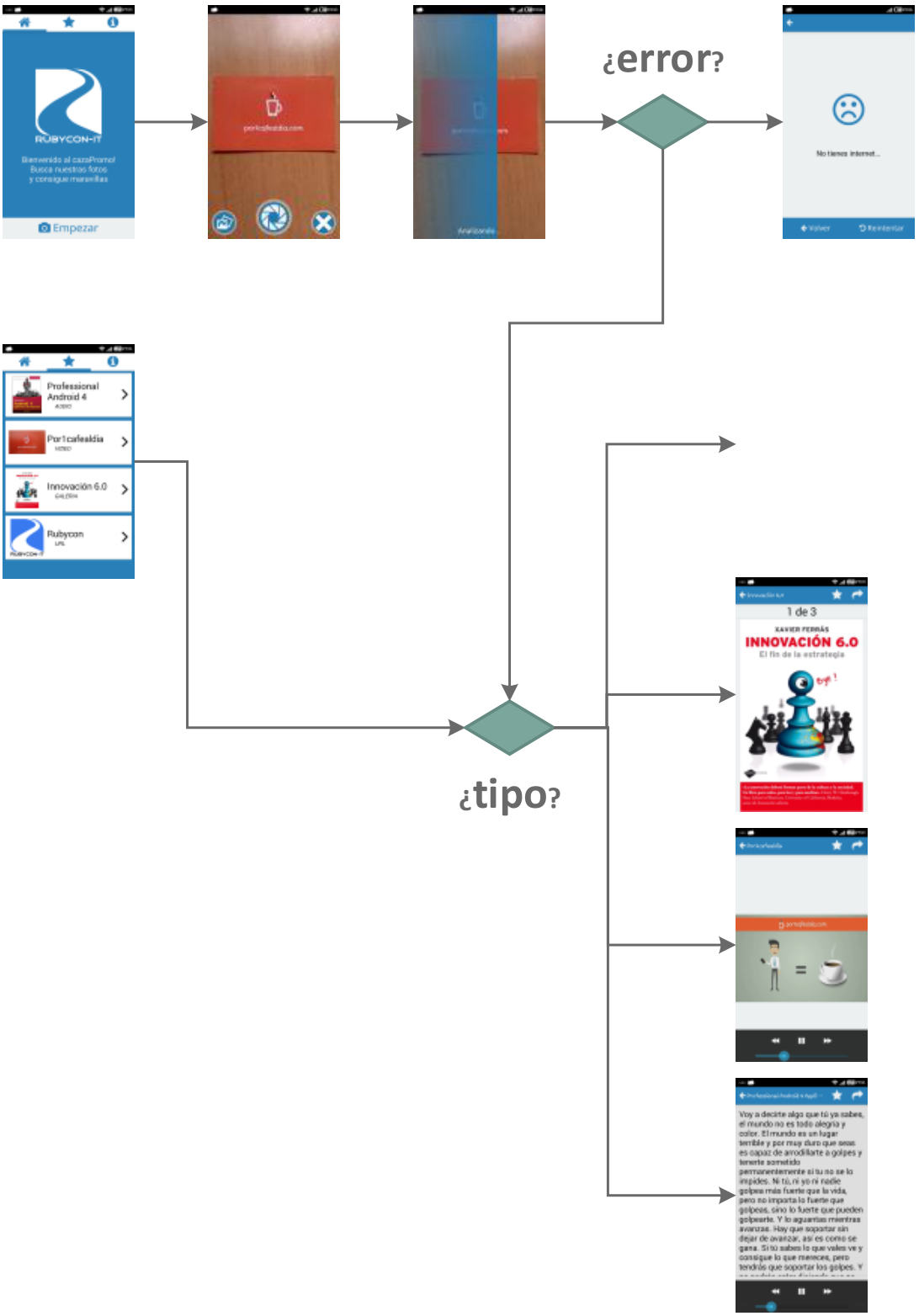


Figura 8.1: Visión general del flujo de la aplicación, mendiante capturas finales.

8.1. Tutorial

Al iniciar la aplicación, lo que primero verá el usuario será el tutorial. En este se pretende guiar al usuario sobre las utilidades de la aplicación. Para ello, se ha diseñado un estilo moderno, en el que se ha buscado la sencillez a la vez que la vistosidad. Para la transición entre los pasos se ha optado por el estándar que se sigue en todas las aplicaciones, deslizando con el dedo. En las figuras 8.2 - 8.5 podemos ver los cuatro pasos de los que se compone dicho tutorial.

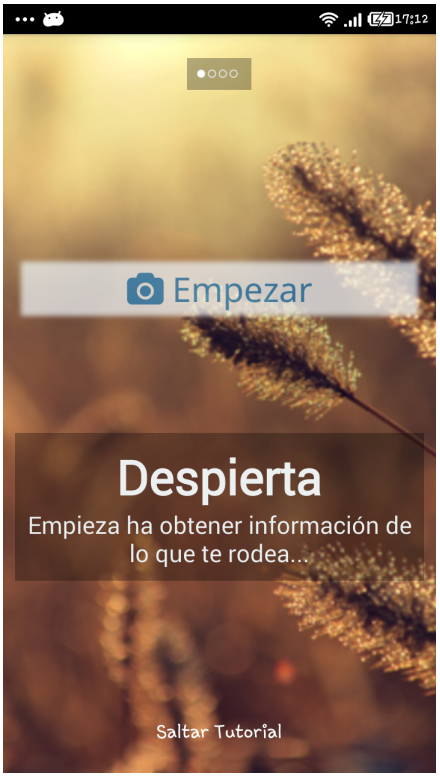


Figura 8.2: Captura final del paso 1 del tutorial.

Figura 8.3: Captura final del paso 2 del tutorial.

Figura 8.4: Captura final del paso 3 del tutorial.

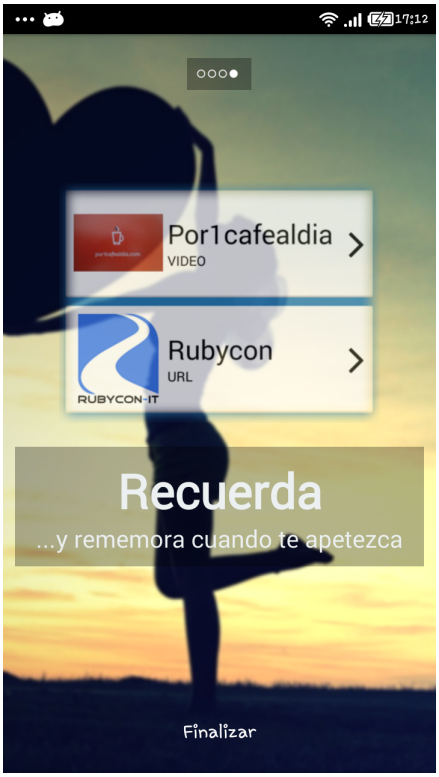


Figura 8.5: Captura final del paso 4 del tutorial.

8.2. Pantallas Principales

Una vez ya se ha visto o saltado el tutorial, se accede a las pantallas principales de la aplicación. La transición entre estas se realiza arrastrando con el dedo, como en el tutorial o, simplemente, pulsando en la pestaña correspondiente.

Primero, y quizás la pantalla más importante de estas, tenemos la pantalla de "inicio" (Figura 8.6). Esta simplemente ofrece un mensaje de bienvenida seguido de un botón desde el que se accede a la cámara, para empezar el proceso de reconocimiento.

Seguidamente, se encuentra la pantalla de "favoritos" (Figura 8.7), en la cual se muestra una lista de los resultados guardados como tal, de los que se ofrece una información mínima, y simplemente pulsando sobre ellos se puede acceder a su contenido.

Figura 8.6: Captura final de la vista "inicio".

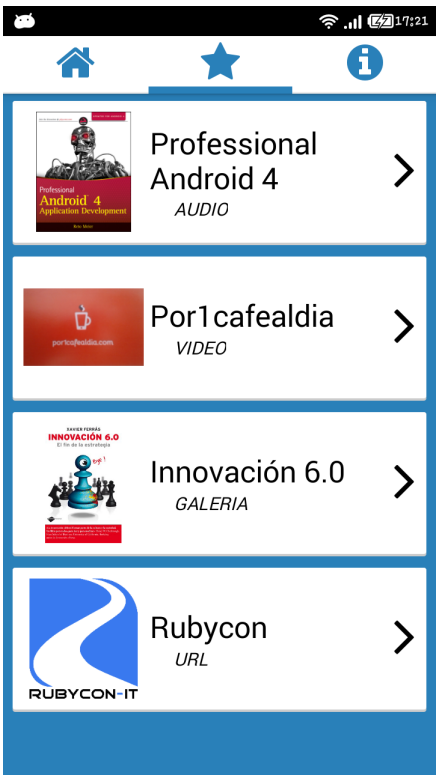


Figura 8.7: Captura final de la vista "favoritos".

Por último, tenemos la pantalla de "acerca de" (Figura 8.8), en la cual se encuentra una pequeña información sobre la aplicación y las licencias que se han utilizado para su desarrollo (Figura 8.9). Además, desde esta se ofrece la posibilidad de volver a ver el tutorial inicial.

Figura 8.8: Captura final de la vista "acerca de".

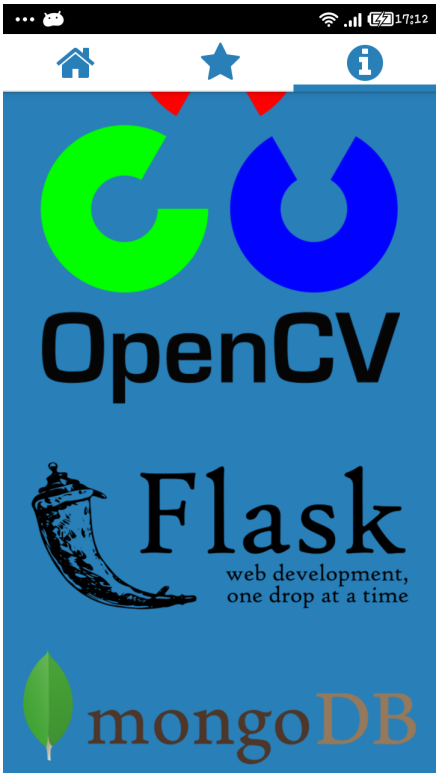


Figura 8.9: Captura final de la vista "licencias".

8.3. Cámara y envío

Una de las funciones principales de la aplicación es la de realizar una fotografía o elegir una de la galería, ya que sin esta funcionalidad no se podría enviar una imagen para su reconocimiento. Esta se encuentra en la pantalla de "cámara" (Figura 8.10), la cual ofrece una simple vista de cámara con los botones de realizar fotografía, acceder a la galería y cerrar.

Una vez el usuario ha realizado una fotografía o la ha obtenido de la galería, empieza el proceso de reconocimiento. Para hacer constancia de este hecho, se muestra una animación de "escaneo", en la cual se simula que la fotografía está siendo analizada (Figura 8.11). Cuando el proceso ha terminado se mostrará el resultado, o en caso de error, se mostrará la vista de "error", en la que muestran los diferentes tipos de errores (Figura 8.12), como "fallo en la conexión" o "no se ha podido reconocer".

Figura 8.10: Captura final de la vista "cámara".

Figura 8.11: Captura final de la vista "animación".

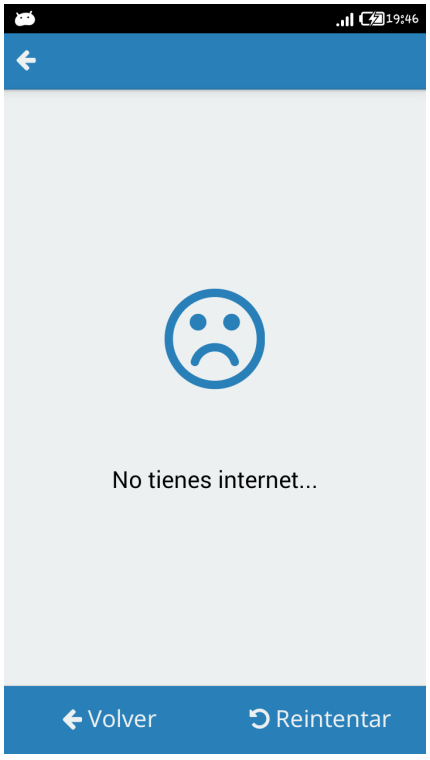


Figura 8.12: Captura final de la vista "error".

8.4. Resultados

El último paso del proceso de reconocimiento es mostrar los resultados de este. Para ello se han implementado diferentes vistas de resultado, una por cada tipo, pero siempre manteniendo la misma estructura. En la parte de arriba se dispone de una barra de acción, en la cual se muestra el título de la imagen, junto con las opciones de "compartir" y "favoritos".

Si el resultado es de tipo web, se muestra directamente dicha web desde dentro de la aplicación, pudiendo visualizarla de una manera rápida y sencilla (Figura 8.13).

En cambio, si el resultado es de tipo galería de imágenes, se muestran las imágenes a pantalla completa, pudiéndolas pasar, simplemente con el dedo, como en cualquier galería ofrecida por un dispositivo táctil (Figura 8.14). Además, también se podrán realizar los típicos gestión de zoom con los dedos.

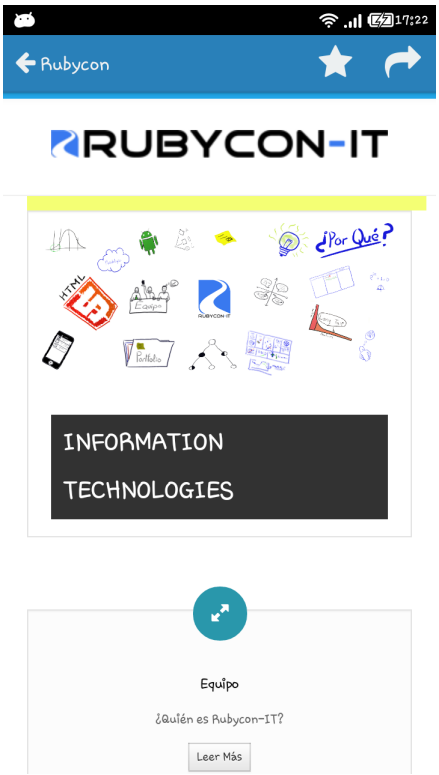


Figura 8.13: Captura final de la vista "resultado web".

Figura 8.14: Captura final de la vista "resultado galería".

Por otro lado, si el resultado es de tipo vídeo (Figura 8.15), se mostrará el video en el centro de la pantalla, junto con unos controles Multimedia en la parte de abajo, desde los que se podrá controlar las reproducción de este, y los cuales desaparecerán y aparecerán al tocar la pantalla.

Por último, si el resultado es de tipo audio (Figura 8.16), se mostrará una descripción de la imagen reconocida junto a los controles multimedia para controlar la reproducción de este.

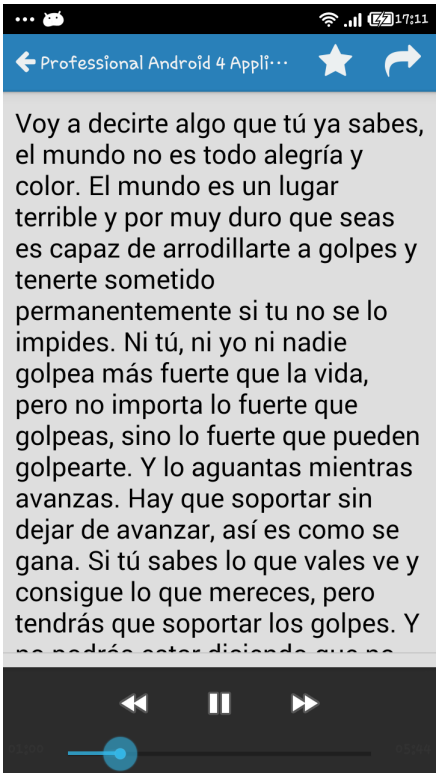


Figura 8.15: Captura final de la vista "audio".

Figura 8.16: Captura final de la vista "video".

Capítulo 9

Conclusiones

En este último capítulo se presentan las conclusiones técnicas y personales obtenidas durante el desarrollo del proyecto. Además, se planteará el posible trabajo futuro que se pueda desarrollar a partir del resultado obtenido.

9.1. Conclusiones Técnicas

Tras la finalización del proyecto, desde el punto de vista técnico, se han alcanzando los siguientes objetivos:

- Desarrollo de una aplicación para el sistema operativo Android.
- Construcción de una pequeña base de datos SQLite para Android.
- Creación de un servidor con servicios REST.
- Diseño e implementación de una base de datos MongoDB.
- Reconocimiento de imágenes utilizando los algoritmos de la librería OpenCV.

Estas metas alcanzadas complacen de manera placentera los objetivos planteados por la empresa el inicio del proyecto. Por tanto, se puede afirmar, que desde el punto de vista técnico el proyecto ha resultado ser un éxito.

9.2. Conclusiones Personales

La realización de este proyecto y estancia en prácticas ha sido una tarea muy enriquecedora, quizá la más enriquecedora en toda la carrera, ya que se han obtenido experiencia en diferentes aspectos que no se pueden aprender en las clases. Estos aspectos son los siguientes:

- **Aprender a programar para el sistema operativo Android**. Inicialmente en solitario y luego con el apoyo de los expertos de la empresa, se han aprendido los conocimientos necesarios para desarrollar aplicaciones para este.

- **Aprender a diseñar interfaces gráficas intuitivas para dispositivos móviles**. Para la implementación de la aplicación ha sido un punto importante el diseño de las interfaces de usuario.

- **Ver las bondades del trabajo en equipo en un entorno real**. A lo largo de la carrera, se han realizado numerosas prácticas de trabajo en equipo, pero durante la realización de este proyecto se ha podido comprobar los aspectos positivos de este.

- **Obtener una experiencia real con las metodologías ágiles**. Durante la estancia se han podido aplicar los conocimientos adquiridos en la asignatura Métodos Ágiles de una manera práctica.

- **Dar importancia a las librerías de terceros**. En un proyecto como este, con un tiempo acotado, resulta realmente útil la utilización de librerías estandarizadas que te ahorran el trabajo de volver a implementar algunas funcionalidades sencillas, como hacer zoom a una imagen.

- **Conocer la importancia de las pruebas**. A partir de la programación para Android, se ha podido conocer la necesidad de impartir multitud de pruebas para asegurar el correcto funcionamiento de la aplicación.

- **Utilización de un servidor real**. Durante la carrera, se ha trabajado con sistemas que incluyen la implementación de la parte correspondiente al servidor, pero no se había probado el funcionamiento de esta desde un servidor real.

Con la adquisición de estos aspectos y con las objetivos técnicos alcanzados, vistos en el punto anterior, se puede afirmar que se han superado con creces los objetivos establecidos al inicio del proyecto, y comentados en la introducción (Capítulo 1, Sección 1.4).

Durante el aprendizaje y aplicación de todos estos aspectos, se han ido viendo los conocimientos adquiridos durante la carrera, donde se ha podido comprobar la necesidad y utilidad de algunas asignaturas. Concretamente las asignaturas destacadas para la realización de este proyecto han sido las siguientes:

- Todas las asignaturas sobre programación, como **Programación Avanzada (EI1017)** o **Estructuras de Datos (EI1013)** donde se aprendieron los conceptos fundamentales sobre esta para la elaboración del proyecto.

- La asignatura **Sistemas Distribuidos (EI1021)**, donde se adquirieron y aplicaron conocimientos referentes a la arquitectura cliente-servidor, utilizada para la elaboración del proyecto.

- Las asignaturas **Bases de Datos (EI1020)** y **Diseño e Implementación de Bases de Datos (EI1041)**, donde se estudiaron los conceptos básicos sobre bases de datos y su aplicación.

- La asignatura **Algoritmia (EI1022)**, los conocimientos adquiridos sobre algoritmos y costes computacionales, tuvieron un papel fundamental en la parte del reconocimiento de imágenes.
- La asignatura **Programación Concurrente y Paralela (EI1024)**, en la que se obtuvieron los conocimientos necesarios sobre la utilización de hilos, utilizados para la paralización de las tareas de reconocimiento.
- Las asignaturas **Fundamentos de Ingeniería del Software (EI1023)** y **Análisis de Software (EI1032)**, donde se aprendieron los conocimientos referentes a la elaboración de casos de uso, diagramas de clases, diagramas de actividades, y todo lo relacionado con las fases de análisis del software.
- La asignatura **Diseño de Software (EI1039)**, en la cual se estudiaron los patrones de diseño utilizados en la implementación del proyecto. Además, se obtuvieron conocimientos sobre usabilidad, que han sido útiles a la hora de diseñar la interfaz de usuario de la aplicación.
- Las asignaturas **Gestión de Proyectos de Ingeniería del Software (EI1040)** y **Métodos Ágiles (EI1050)**, donde se han adquirido los conocimientos utilizados para la gestión y planificación del proyecto utilizando metodologías ágiles.

9.3. Trabajo Futuro

El objetivo de este proyecto es muy ambicioso, ya que no se centra en un caso particular, sino que se deja abierta su aplicación para distintas funciones. Por lo tanto, es evidente que en la duración de la estancia en prácticas no se han podido cumplir todas las posibles funciones que se quisieran implementar.

Por ello, la extensión del proyecto se podría mejorar con los siguientes puntos:

- **Aumento de la velocidad del algoritmo reconocedor de imágenes**. Para ello se debería realizar un estudio detallado de cómo se podría reducir el tiempo. Durante la elaboración del proyecto se plantearon mejoras como no analizar todas las imágenes, descartando algunas directamente durante el proceso.
- **Aumentar la efectividad del algoritmo reconocedor de imágenes**. Para este punto también se debería realizar un estudio detallado, comparando las diferentes posibilidades, como puede ser comparar también los colores de las imágenes.
- **Mejora de la paralelización del servidor**. El modulo de reconocimiento y el servidor se han realizado de manera independiente. Se han paralelizado las funciones de reconocimiento, pero utilizando variables globales. A consecuencia de esto, pueden haber problemas al realizar peticiones simultáneas al servidor. Para mejorar esto se deberían convertir estas variables globales en locales.
- **Implementación de una interfaz de administración**. Será necesaria a la hora de añadir imágenes en el sistema de una manera sencilla. No obstante, no se ha podido realizar por la duración del proyecto.

- **Prueba del reconocimiento con objetos**. Una propuesta de mejora, que no se ha probado por falta de tiempo, sería comprobar el funcionamiento del sistema con objetos e investigar sobre esta funcionalidad.
- **Adición de un elemento social al sistema**. Integrar en los resultados funcionalidades como añadir puntuación o comentarios, de manera que los diferentes usuarios puedan interactuar entre ellos.
- **Integración de la aplicación con *Google Analytics***. Añadiendo esta funcionalidad, se permitiría ver estadísticas sobre el uso de la aplicación de una manera sencilla.

Bibliografía

[1] Android tops 81 percent of smartphone market share in q3. http://www.engadget.com/2013/10/31/strategy-analytics-q3-2013-phone-share/. [Consulta: 25 de Junio de 2014].

[2] Código qr. http://blogdeviajesyturismo.com/sigue-este-blog-desde-tu-celular-con-el-codigo [Consulta: 25 de Junio de 2014].

[3] Financial distress, lender passivity and project finance: the case of eurotunnel.

[4] Flatui colors. http://flatuicolors.com/. [Consulta: 30 de Junio de 2014].

[5] Guía de diseño android. http://developer.android.com/intl/es/design/index.html. [Consulta: 25 de Junio de 2014].

[6] Imagen código de barras en libros. http://www.oseslaser.com/images/galeria/libros-etiquetados-para-inventario-control-personal.JPG. [Consulta: 25 de Junio de 2014].

[7] Métodologías ágiles en el desarrollo de software. http://www.extremeprogramming.org/rules/pair.html. [Consulta: 25 de Junio de 2014].

[8] Pair programming. http://www.extremeprogramming.org/rules/pair.html. [Consulta: 25 de Junio de 2014].

[9] Rubycon information technologies s.l. http://rubycon.es/. [Consulta: 24 de Junio de 2014].

[10] Sqlite - small. fast. reliable. choose any three. http://www.sqlite.org/. [Consulta: 26 de Junio de 2014].

[11] Superconducting super collider project.

[12] Armin Ronacher. Flask - web development one drop at a time. http://flask.pocoo.org/. [Consulta: 26 de Junio de 2014].

[13] Chris Banes. Photoview. http://github.com/chrisbanes/PhotoView. [Consulta: 8 de Julio de 2014].

[14] Dave Gandy. Font awesome, the iconic font and css toolkit. http://fortawesome.github.io/Font-Awesome/. [Consulta: 30 de Junio de 2014].

[15] Diario ABC. El número de líneas móviles en españa cae por primera vez. http://www.abc.es/economia/20130212/abci-telefonia-movil-perdida-lineas-201302121234.html/. [Consulta: 24 de Junio de 2014].

[16] Django Software Foundation. Django. https://www.djangoproject.com/. [Consulta: 26 de Junio de 2014].

[17] freepik. flaticon - free vector icons. http://www.flaticon.com/. [Consulta: 30 de Junio de 2014].

[18] itseez. Opencv. http://opencv.org/. [Consulta: 28 de Junio de 2014].

[19] James Smith. Android asynchronous http client, a callback-based http client library for android. http://loopj.com/android-async-http/. [Consulta: 8 de Julio de 2014].

[20] Jon Fingas. Para cambiar. http://oa.upm.es/20480/1/INVE_MEM_2012_135438.pdf. [Consulta: 28 de Junio de 2014].

[21] Joyent, Inc. Nodejs. http://nodejs.org/. [Consulta: 26 de Junio de 2014].

[22] Ken Schwaber and Jeff Sutherland. La guÃa de scrum. https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide-ES.pdf. [Consulta: 25 de Junio de 2014].

[23] Marius Muja and David G. Lowe. Flann - fast library for approximate nearest neighbors. http://www.cs.ubc.ca/research/flann/. [Consulta: 28 de Junio de 2014].

[24] Mathieu Labbé. Find-object. https://code.google.com/p/find-object/. [Consulta: 28 de Junio de 2014].

[25] MongoDB, Inc. Mongodb. http://www.mongodb.org/. [Consulta: 26 de Junio de 2014].

[26] Mountain Goat Software. Planning poker. http://www.planningpoker.com/. [Consulta: 25 de Junio de 2014].

[27] Nicola Iarocci. eve - python rest api framework. http://python-eve.org/. [Consulta: 26 de Junio de 2014].

[28] nodejitsu. forever, a simple cli tool for ensuring that a given script runs continuously. https://github.com/nodejitsu/forever. [Consulta: 10 de Julio de 2014].

[29] OpenCV Documentation. Feature matching. http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html. [Consulta: 28 de Junio de 2014].

[30] OpenCV Documentation. Orb (oriented fast and rotated brief). http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_orb/py_orb.html. [Consulta: 28 de Junio de 2014].

[31] Oracle. Mysql, the worldś most popular open source databas. http://www.mysql.com/. [Consulta: 26 de Junio de 2014].

[32] Pivotal Software. spring. http://spring.io/. [Consulta: 26 de Junio de 2014].

[33] Wikipedia. Android. http://es.wikipedia.org/wiki/Android. [Consulta: 27 de Junio de 2014].

[34] Wikipedia. Lean startup. http://en.wikipedia.org/wiki/Lean_startup. [Consulta: 27 de Junio de 2014].

[35] Wikipedia. Nosql. http://es.wikipedia.org/wiki/NoSQL. [Consulta: 27 de Junio de 2014].

Anexo A

Definiciones y Abreviaturas

API (Application Programming Interface): Conjunto de funciones o métodos que ofrece una biblioteca para ser utilizada por otro software como una capa de abstracción.

REST (Representational State Transfer): Técnica de arquitectura software para sistemas distribuidos, como la *World Wide Web*, basada el protocolo HTTP.

SCRUM: Modelo de desarrollo ágil que utiliza ciclos de desarrollo completos.

IDE (Integrated Development Environment)): Programa informático compuesto por un cojunto de herramientas de programación (un editor de código, un compilador, un depurador y un constructor de interfaz gráfica) el cual ofrece un marco de trabajo amigable para la construcción de software.