Universidad de Valladolild

Escuela de Ingeniería Informática

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática, Mención en Tecnologías de la Información y de la Comunicación

Desarrollo de una app móvil para la explotación de una base de datos de pigmentos

Autor:
Sergio Esteban Pellejero

Universidad de Valladolild

Escuela de Ingeniería Informática

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática, Mención en Tecnologías de la Información y de la Comunicación

Desarrollo de una app móvil para la explotación de una base de datos de pigmentos

Autor:
Sergio Esteban Pellejero
Tutor:
Joaquín Adiego Rodriguez
Angel Carmelo Prieto Colorado

Resumen

El fin de este trabajo de fin de grado es recrear una antigua página web que explotaba una base de datos orientada a la pigmentación. La aplicación antigua hecha en forma de web ha sido trasladada a una aplicación móvil capaz de ejecutarse en cualquier dispositivo con Android que hay en la actualidad.

La utilidad principal de desarrollo de esta aplicación es permitir que cualquier científico o persona especializada en el campo de la restauración, tenga de manera accesible, rápida y sencilla la información necesaria acerca de todos los pigmentos naturales que se encuentran en la Tierra.

Este propósito se ha conseguido rediseñando una base de datos ya existente con toda esta información además de generar la información que faltaba o estaba corrupta. Desarrollando en Android de la mano del cliente una aplicación que cumple los requisitos del mismo y ofreciéndola de manera gratuita en la tienda de Google para que cualquier persona la tenga a su disposición.

Uno de los principales retos que se han planteado durante el desarrollo de la aplicación ha sido tratar con cantidades muy grandes de datos en tiempos de reacción para el ojo humano casi imperceptibles.

Índice

Índice **2**

Índice de figuras **4**

1. Introducción **1**
1.1. Prefacio . . . 1
1.2. Visión global . . . 1
1.3. Marco de desarrollo . . . 2
1.4. Objetivos . . . 2
1.5. Beneficios . . . 2
1.6. Técnicas experimentales y medidas . . . 3

2. Fundamentos y herramientas **5**
2.1. Metodología . . . 5
2.1.1. Kanban . . . 5
2.1.2. Gitflow . . . 7
2.2. Herramientas y tecnologías . . . 9
2.2.1. Taiga . . . 9
2.2.2. SonarQube . . . 11
2.2.3. SonarLint . . . 13
2.2.4. Jenkins . . . 14
2.2.5. Android Studio . . . 16
2.2.6. Github . . . 17
2.2.7. Overleaf . . . 17
2.2.8. SQLite . . . 18
2.3. Balsamiq Mockups 3 . . . 18
2.4. Resumen de la memoria . . . 19

3. Planificación y gestión del proyecto **20**
3.1. Necesidad de planificación . . . 20
3.2. Definición del alcance y objetivos . . . 21
3.3. Tiempo de desarrollo . . . 21
3.4. Recursos y costes . . . 21
3.4.1. Recursos y tipos . . . 21
3.4.2. Estimación de los costes . . . 22
3.5. Tratamiento de los riesgos . . . 23
3.6. Control de configuraciones . . . 24
3.7. Calidad de los productos entregados . . . 25

4. Diseño **27**
4.1. Volviendo al desarrollo en cascada . . . 27
4.2. Definición de criterios . . . 32
4.2.1. Definición de hecho . . . 32
4.2.2. Criterios de aceptación . . . 33

4.3. Épicas e historias de usuario 34
4.3.1. Épicas 35
4.3.2. Historias de usuario 35
4.4. Backlog 37
4.5. Bocetos iniciales 43
4.5.1. Logo de la aplicación y nombre 43
4.5.2. Diseño de las primeras interfaces 45
4.6. Sistema de Integración continua 51

5. Diseño de la base de datos 53
5.1. Suposiciones y convenciones 53
5.2. Posibles consultas contra la BBDD 53
5.3. Modelado conceptual 54
5.3.1. Entidades y atributos 54
5.3.2. Diagrama de Entidad Relación 56
5.4. Implementación de la base de datos 57
5.4.1. ¿Que son los POJO? 58
5.4.2. Ejemplo 59
5.5. Otras consideraciones 62
5.5.1. Diseño antiguo VS diseño actual 62
5.5.2. Código frente a eficacia 63
5.5.3. Scripts necesarios 64
5.6. Tratamiento de las BBDD en Android 65

6. Guía de Instalación y Manual de Usuario 67
6.1. Guía de Instalación 67
6.2. Manual de Usuario 67

7. Problemas durante el desarrollo 69

8. Conclusiones Finales 71

Referencias 72

Índice de figuras

1. Patrón completo . . . 8
2. Patrón Derivado . . . 9
3. Tablero de Taiga.io . . . 10
4. Configuración del fichero build.gradle . . . 12
5. Primer análisis del código inicial generado por Android Studio . . . 13
6. Tarea Gradle para Sonar . . . 13
7. SonarLint en Android Studio . . . 14
8. Flujo seguido por Jenkins . . . 15
9. Panel General de jenkins . . . 16
10. Interfaz principal de Balsamiq Mockups . . . 18
11. Flujo de Integración continua . . . 26
12. Diagrama de Casos de Uso . . . 28
13. Diagrama de Paquetes a alto nivel . . . 30
14. Diagrama de secuencia de alto nivel para ver la información de un pigmento . . . 31
15. Relación entre épicas, historias y temas . . . 34
16. Formato que seguirán las historias de usuario . . . 36
17. Diagrama de precedencia de las principales tareas . . . 42
18. Diseño 1 del logo de la aplicación . . . 43
19. Diseño 1 del logo de la aplicación . . . 43
20. Diseño final del logo de la aplicación . . . 44
21. Menú principal de la aplicación . . . 47
22. Lista principal con todos los pigmentos del sistema . . . 47
23. Información de los pigmentos . . . 48
24. Información gráficas de pigmentos . . . 48
25. Menú para hacer una consulta simple . . . 49
26. Menú de selección de colores . . . 49
27. Menú de selección de elementos . . . 49
28. Menú de selección de colores . . . 50
29. Menú de selección de elementos . . . 50
30. Menú de reporte de fallos . . . 51
31. Entidades y cardinalidades del modelo . . . 56
32. Diagrama de Entidad-Relación de la base de datos . . . 57
33. Diseño propuesto en Access de la Base de Datos . . . 63
34. Diseño propio de la base de datos . . . 63

1. Introducción

En esta sección vamos a comentar, el objetivo de este trabajo de fin de grado, la metodología utilizada para llevar a cabo el desarrollo, ya que es un proyecto puramente de desarrollo de una aplicación móvil con un fin determinado y focalizado. También se expondrán los principales objetivos que han motivado al alumno.

1.1. Prefacio

Durante estos últimos años debido a la situación económica y política de España, el sector de la conservación y la restauración del patrimonio cultural se ha dejado un poco de lado, aunque sigue a la cabeza de tecnología para restauración.

Al mismo tiempo, se trata de un campo de trabajo que demanda amplia información, especialmente en lo relativo a la utilización de nuevas tecnologías, tanto de tratamiento como de estudio, para evitar intervenciones desafortunadas que han hecho peligrar el estado de conservación de muchas obras de nuestro rico patrimonio. A día de hoy contamos con tecnología puntera en Europa como pueden ser los escáneres en 3D de zonas muy amplias, estos escáneres nos ofrecen imágenes de muy alta resolución que los historiadores y profesionales de la materia precian enormemente para evitar deteriora las obras reales.

Por citar algunas de las otras tecnologías que podemos encontrar en el ámbito de la restauración pueden ser: tomografía, microscopía óptica, envejecimiento artificial (cada vez mas desarrollado con los potentes ordenadores con los que contamos), espectrofotometría infrarroja por transformación de Fourier son algunos de los métodos que podemos aplicar cuando se trata de restaurar nuestro patrimonio.

Este proyecto trata de facilitar tanto la búsqueda como la utilización de información relacionada con el sector de la restauración y conservación del patrimonio, para que profesionales dedicados a este campo puedan encontrar un punto de referencia práctico y de fácil acceso que les ayude en sus investigaciones y proyectos y que esté al alcance de todas las personas.

1.2. Visión global

Este proyecto consiste en la creación de una aplicación móvil que explote de una manera adecuada una base de datos ya existente sobre pigmentos naturales y sintéticos. Dicha base de datos posee información tanto de las propiedades colorimétricas como estructurales de cada pigmento, dichos datos han sido recogidos mediante la aplicación de técnicas experimentales en laboratorios con los instrumentos adecuados. Más adelante hablaremos brevemente.

Esta aplicación tiene que ofrecer al usuario la capacidad de obtener de manera sencilla e intuitiva información compleja acerca de los diferentes pigmentos. Además de cada uno de ellos vamos a

hablar también acerca de sus diferentes usos y de las técnicas por las que se han obtenido dichas medidas, y quizá en algunos de ellos (en los que al científico director le parezca mas relevante) se podrán añadir notas curiosas, como por ejemplo de la obtención de dicho pigmento o de usos pintorescos del mismo a lo largo de la historia.

La base de datos, como hemos dicho ya existe, la idea y objeto principal de este proyecto es el desarrollo de una aplicación capaz de funcionar en un porcentaje elevado de los dispositivos móviles actuales sino en todos.

1.3. Marco de desarrollo

El actual proyecto esta asentado sobre un proyecto más grande y ya existente, que como dijimos era el que nos aportaba la información acerca de los pigmentos y de las muestras. Dicho proyecto: "La aplicación de la técnica láser a la conservación del patrimonio"fue desarrollado por el Departamento de Física de la Materia Condensada, Cristalografía de la Universidad de Valldolid en colaboración con el Centro Tecnología Láser. Aunque esta información es pasada, me parece como menos importante nombrar dicho proyecto, ya que asienta las bases actuales y todas las medidas e información presente sobre las que se va a desarrollar la aplicación móvil.

1.4. Objetivos

EL objetivo principal es facilitar tanto la búsqueda como al utilización y acceso a la información por parte de toda la población de posibles usuarios. La manipulación de la base de datos actual es bastante sencilla, por lo que con el paso de los años si futuras investigaciones en este campo lo desean pueden ir actualizando la información. Además la presentación sencilla, accesible e intuitiva de la información hacen que no se requieran conocimientos previos en la materia para sacar conclusiones u obtener información de manera rápida, precisa y veraz.

Otro objetivo es poner a disposición de empresas y científicos la información recopilada por otros científicos. Al final la ciencia no progresa si las personas no ponemos de nuestro lado, si cada una de las futuras investigaciones en la materia aporta información a la base de datos, por poca que sea, la ciencia se perpetúa.

1.5. Beneficios

A continuación destacaremos algunas de las posibles aplicaciones que tiene este proyecto, incluyendo las posibles utilidades que supondrá su culminación para diversos colectivos.

La utilización de determinados pigmentos y aglutinantes, así como sus mezclas, es en muchas ocasiones característico de la obra de un autor, así como de la época, zona de influencia y desarrollo del arte en un periodo histórico concreto. De esta forma, los estudios realizados a través de las

técnicas instrumentales descritas anteriormente pueden ser aplicados en tareas de identificación, datación y autentificación de obras de arte. Los expertos encargados de realizar estas tareas pueden beneficiarse de la creación de la base de datos, ya que podrán recurrir a ella para realizar contrastes entre los registros obtenidos en el laboratorio y aquellos recogidos en la base.

Los profesionales de empresas de restauración necesitan en ocasiones reintegrar partes dañadas siguiendo las técnicas originales. Requieren por lo tanto conocer los orígenes de los pigmentos empleados con el fin de acometer las obras de restauración en las mejores condiciones. Estos expertos pueden beneficiarse de la creación de la base de datos, puesto que podrán acceder a diverso tipo de información obtenida a través de potentes herramientas analíticas de una forma fácil y con una disponibilidad total una vez registrados como usuarios autorizados.

Generar una aplicación móvil disponible en un dispositivo que podemos llevar a cualquier lugar es una importante innovación por el hecho de que esta herramienta puede ser utilizada de manera directa e inmediata por las empresas y por los expertos de instituciones y organismos dedicados a la restauración y conservación del patrimonio histórico-artístico, que muchas veces deben de decidir en un corto espacio de tiempo el tratamiento más adecuado para restaurar una determinada obra. Cada uno de los restauradores que tenemos en el territorio puede tener instalada dicha aplicación y visualizar métricas y datos mientras realiza trabajos de campo, sin necesidad de acceso a Internet, de una manera rápida y sencilla. La disponibilidad de la información es crucial en el desarrollo de este proyecto.

De la misma forma, la aplicación móvil desarrollada permitirá poner en contacto a diversos colectivos de profesionales que realizan sus investigaciones en todo el mundo, así como obtener información complementaria a estas como bibliografía, ya sea en forma de libro o medio digital.

1.6. Técnicas experimentales y medidas

En esta parte vamos a hablar de las diferentes técnicas experimentales que han sido usadas para obtener los pigmentos, junto con algunas de las medidas que vamos a presentar dentro de la aplicación. Esto no es del ámbito informático pero tiene una cierta importancia a la hora de entender las gráficas que más adelante se mostrarán a los diferentes usuarios.

Las principales técnicas y métodos de análisis de los que se han dispuesto para la elaboración de las bases de datos son los siguientes:

- **Análisis micro colorimétrico - Determinación de coordenadas tricrómicas**: las coordenadas tricrómicas permiten situar al pigmento dentro de un mapa cromático sin ambigüedad alguna, posibilitando el estudio del grado de evolución cromófora del pigmento en función de su antigüedad, composición mineralógica, procedencia y posible evolución temporal y espacial, e interacción con otros pigmentos.

- **Difracción de rayos-x**: el análisis del difractograma de rayos X, permiten obtener con rigor las fases mineralógicas presentes en el material pigmentante. Los minerales accesorios a veces son de relevante importancia dado que posibilitan determinar áreas de procedencia

comunes y, por tanto, determinar si un pigmento concreto fue usado de forma generalizada en diversas obras de arte o estuvo restringido a obras y autores determinados.

- **Espectroscopía infrarroja por transformada de Fourier con transmisión atenuada de reflectancia**: la técnica FTIR-ATR (Infrarrojo por Transformada de Fourier con Reflectancia Total Atenuada) permite identificar las fases iónicas y moleculares presentes y, a través de los parámetros espectrales característicos, se obtiene información suplementaria sobre el estado del pigmento y su proceso evolutivo temporal.

- **Especstroscopía Raman**: la espectroscopía micro-Raman es una técnica que proporciona información sobre la estructura dinámico vibracional de las especies químicas que constituyen el pigmento, complementándose con la espectroscopía FTIR ATR. Los espectros dan una información complementaria a la estructura estática obtenida mediante DRX. Además, la espectroscopía micro-Raman permite la detección de posibles cambios físico-químicos inducidos en el material por la acción de la radiación láser, con la ventaja de carácter de inocuidad y su alta resolución espacial 1μ.

La información de este apartado se puede encontrar de una manera mejor explicada en la memoria de la Tesina en la que se basó la creación de la página web del mismo fin que la aplicación que yo desarrollo. Dicho trabajo de fin de grado se titula: "Sistema telemático de búsqueda e identificación de pigmentos mediante parámetros colorimétricos y estructurales", escritor por Eduardo Cabañas Palacios y Pablo Porro Plaza en Septiembre del 2000.

2. Fundamentos y herramientas

En esta sección vamos a comentar, la metodología utilizada para llevar a cabo el desarrollo, ya que es un proyecto puramente de desarrollo de una aplicación móvil con un fin determinado y focalizado. En cuyo desarrollo van a entrar temas de casi todas las materias e información aprendida a lo largo de estos años de estudio.

2.1. Metodología

En esta sección se va a explicar la metodología utilizada a lo largo del proceso de desarrollo de software. Lo primero cabe destacar que se va a utilizar una herramienta de gestión de proyectos, de código abierta y gratuita. La idea del uso de esta herramienta tiene que ver con la metodología de desarrollo que se va a llevar a cabo durante el transcurso de este trabajo de fin de grado. La metodología usada en cuestión es Kanban. A continuación hay una breve descripción para las personas que la desconozcan.

2.1.1. Kanban

El modelo de proceso que se va a utilizar para el desarrollo del proyecto será Kanban. Se trata de un modelo similar a Scrum pero con unas directivas más flexibles. La idea en la que se basa Kanban es visualizar el flujo de trabajo mediante segmentos o estados definidos, representados por columnas en un tablero. Los segmentos que se suelen utilizar son los siguientes:

- **Nuevas**: (new) son las tareas que se han ido añadiendo al backlog, el backlog es la lista de tareas e historias de usuario que se crean en el proyecto. Las nuevas tareas se crean por defecto en el estado de nuevas.

- **Preparado**: (ready) tareas que aún no han comenzado, pero que lo harán en algún momento del futuro. En esta columna se pondrán las siguientes tareas a ser desarrolladas.

- **En desarrollo**: (in progress) a esta columna también se le llama WIP (Work in Progress). La conforman las tareas que se están llevando a cabo en un instante de tiempo determinado.

- **Bloqueadas**: en la que se mostrarán las tareas que están siendo bloqueadas debido a razones externas. Por ejemplo la tarea depende de la finalización de otra actividad, o tenemos que preguntar alguna duda al cliente, o se ha caído un servicio del que depende la tarea. En este caso concreto, no tendremos columna de bloqueadas, porque la herramienta las muestra de otra manera, pero es común que en los tableros kanban aparezca dicha columna.

- **En revisión**: (in review) en esta columna se dispondrán las actividades que están acabadas pero pendientes de revisión por parte de otros compañeros. Las actividades de esta columna tendrán un enlace a la PR asociada para ganar una mayor continuidad del trabajo. Como

este trabajo va a ser desarrollado por una única persona, o como mucho el trabajo será supervisado por el tutor, no contaremos con dicha columna en nuestro tablero.

- **Preparadas para probar**: tareas que han sido realizadas y aprobadas y que tienen que ser testeadas, en el caso de tener una CI/CD, es conveniente tener esta columna. La idea del proyecto es contar con un CI/CD, por lo menos cuando se tenga el suficiente código desarrollado como para que se permita.

- **Hechas**: (done) tareas completadas, dicha columna se irá vaciando y las tareas se pasarán a archivadas, simplemente para que no se sature la vista de la columna de Hecho.

El principio básico de Kanban es que separando las tareas del proyecto en estos segmentos se garantiza que el trabajo del equipo se visualice de forma clara. Además, permite que el workflow se unifique en un solo lugar y que se identifiquen y resuelvan inmediatamente todos los factores que lo bloqueen y de los que dependan.

Cada elemento de trabajo en Kanban se representa en una tarjeta distinta y esta se coloca en una columna para indicar su posición en el flujo de trabajo. Estas tarjetas son las llamadas User stories, que son tareas inicialmente sencillas, que más adelante se subdividen en subtareas para su implementación. Estas tarjetas proporcionan visibilidad a todo el equipo sobre quién está a cargo de qué elementos, la descripción del trabajo y una estimación de la duración del mismo. A mayores (en tablas virtuales Kanban) se suelen añadir capturas de pantalla y otros datos técnicos de valor para el destinatario de la asignación. También en las tarjetas Kanban se permite una estimación de la dificultad de implementación de la actividad, además de una estimación del tipo de tareas que va a desempeñar. La estimación de la dificultad se hace siguiendo la serie de fibonacci. Esto se hace así porque la complejidad normalmente crece en el mismo orden que esta serie.

El segmento central (WIP) y, opcionalmente, el resto de segmentos, tienen especificado un límite de tareas que pueden contener en un instante. Un elemento no podrá pasar a la siguiente columna a no ser que haya espacio disponible. De esta forma se evita el exceso de trabajo activo sin necesidad de sprints que definan la cantidad de tareas asignadas. Normalmente el WIP para cada uno de los integrantes del equipo será de 2 tareas como máximo.

Kanban nos proporciona una reducción de los cuellos de botella. Cuantos más elementos de trabajo haya en curso más se cambia el contexto, por lo tanto una de las cosas que propone Kanban es limitar la cantidad de trabajo en curso. La limitación del trabajo hace que resalten los cuellos de botella.

Otra de las ventajas es una mejor estimación de las entregas futuras debido a que se puede ver cuánto tarda en viajar una actividad de trabajo a través del workflow desde que comienza hasta que se envía.

los principios básicos de Kannban se asientan en los principios básicos del manifiesto ágil [1] y en los principios seguidos por Scrum.

A nivel personal, las referencias de la empresa Atlassian [2] para gestión de proyectos y para información de metodologías me parecen muy buenas, tanto para Kanban [3] como para Scrum [4].

En nuestro caso, la herramienta que vamos a utilizar para tener un poco de control en el proyecto y seguir esta aproximación a Kanban, va a ser Taiga, que se describe mas adelante de manera breve. Por ejemplo, la empresa Atlassian, tiene su propio software de control de proyectos grupales ágiles, llamado Jira [5], usado por grandes compañías a nivel mundial.

Ahora que hemos descrito de manera breve, la metodología que vamos a utilizar, vamos a hablar de las tecnologías principales que se van a utilizar en este proyecto de software.

2.1.2. Gitflow

En los proyectos de software principalmente se trabaja desarrollando código y una vez completados los módulos principales, se van mejorando. Dentro de estos proyectos hay ciertas maneras y formas de trabajar de manera adecuada sobre dicho código. Las diferentes formas de desarrollo que un proyecto y miembros del equipo siguen a la hora del desarrollo del código se denomina Gitflow.

Originalmente el termino de Gitflow [17] fue acuñado por Vincent Driessen en Nvie. Lo que esta metodología define es la creación de un conjunto de ramas especificas, cada una con unas reglas determinadas a la hora de programar con el objetivo de tener un control de las versiones de código del proyecto junto con un control total sobre las versiones finales o de prueba que el equipo va consiguiendo.

El Gitflow es ideal para los proyectos en los que hay marcas de tiempo bastante fuertes. No se introducen conceptos nuevos a la hora del control del código. Simplemente se marca un patrón bien definido, junto con una serie de roles para cada una de las ramas. Con esto conseguiremos los objetivos de tiempo con una mayor facilidad y rapidez.

El patrón definido por Gitflow no es demasiado complejo, para proyectos pequeños es incluso mas tedioso aplicarlo correctamente que simplemente empezar a desarrollar código e ir actualizando el repositorio. Pero incluso para equipos pequeños, de 2 o 3 personas, es muy útil e impide problemas en el futuro derivados de un uso incorrecto de las herramientas de control de versiones.

Primero vamos a mostrar un poco como es el patrón original al completo y luego vamos a explicar la adaptación que yo voy a llevar a cabo en este proyecto, ya que en mi opinión no es necesario aplicar el patrón completo al 100 %.

Patrón original

Este patron nos define un conjunto de ramas sobre los que vamos a trabajar:

- **Master**: la rama master es la que lleva el control de las versiones finales con capacidad suficiente como para existir por ellas solas y que están en el entorno de producción del cliente. Normalmente la rama master no va nunca sola, y en este modelo se encuentra bloqueada,

es decir no se pueden hacer commits (cambios) en ella de manera directa, para cambiar esta rama siempre tiene que haber una solicitud de cambio de otra rama.

- **Develop**: en esta rama se van desarrollando las diferentes partes de la aplicación. Por ejemplo imaginemos que estamos desarrollando un servicio fullstack para una empresa, en develop es donde se verian reflejados todos y cada uno de los cambios, pero como veremos mas adelante, no siempre se hacen los cambios directamente sobre develop.

- **Features**: para llevar un control de grano mas fino acerca de los diferentes cambios, quien los hace y cuando, para cada una de las partes principales de la aplicación se suele sacar una rama llamada feature. Por ejemplo, continuando con el desarrollo de la aplicación fullstack, podríamos sacar una rama develop para el front y una rama develop para el back y a su vez, ramas de funcionalidad de cada una de las ramas de desarrollo de front y de back. Por ejemplo una rama valida de funcionalidad para el back seria login, en la que se implementarían las funciones de login de la aplicación.

- **Hotfix**: estas ramas son ramas que se sacan directamente de master y se acaban fusionando con master, porque aplican cambios importantes que arreglan fallos graves en el entrono de producción que necesitan ser arreglados en el menor tiempo posible.

Podemos observar este patrón al completo en la siguiente figura:

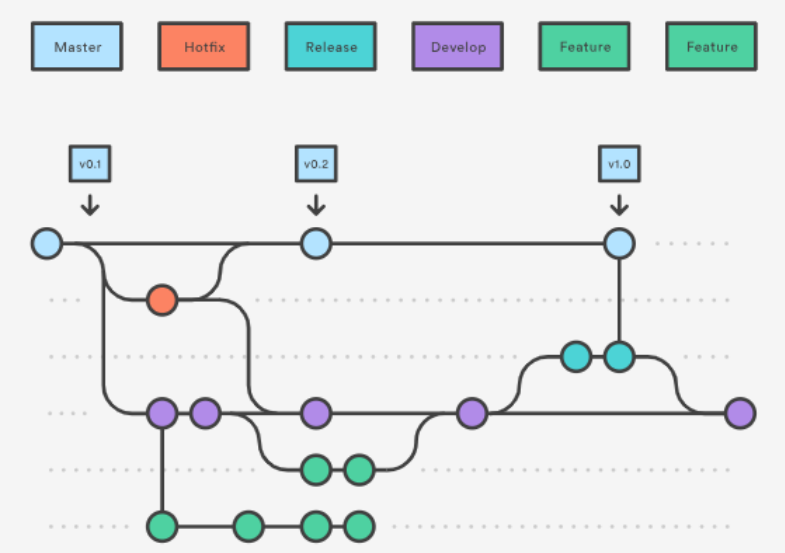

Figura 1: Patrón completo

Patrón derivado

En mi caso se va a seguir un patrón mas flexible ya que no es un proyecto demasiado grande y en mi opinión no hace falta cumplir con el patrón al completo. La idea es la siguiente:

- **Master**: rama en la que existirán las versiones principales y funcionales de la aplicación.

Cada una de las diferentes versiones que se consideren finales se etiquetaran con un tag del tipo: V1.0.0. Esta rama estará bloqueada y solo se podrán realizar cambios mediante peticiones de cambio.

- **Features**: ramas de funcionalidad, son sacadas directamente desde master y desarrollan las diferentes funcionalidades que tendrá la aplicación. Se fusionaran directamente con master cuando se considere necesario.

- **Hotfix**: las ramas de hotfix son necesarias en todos los proyectos. Siempre surgen problemas que hay que resolver en el menor tiempo posible sin tener en cuenta el resto de funcionalidades que se están implementando en el momento. No serán las mas comunes, pero alguna rama de este tipo habrá en el proyecto.

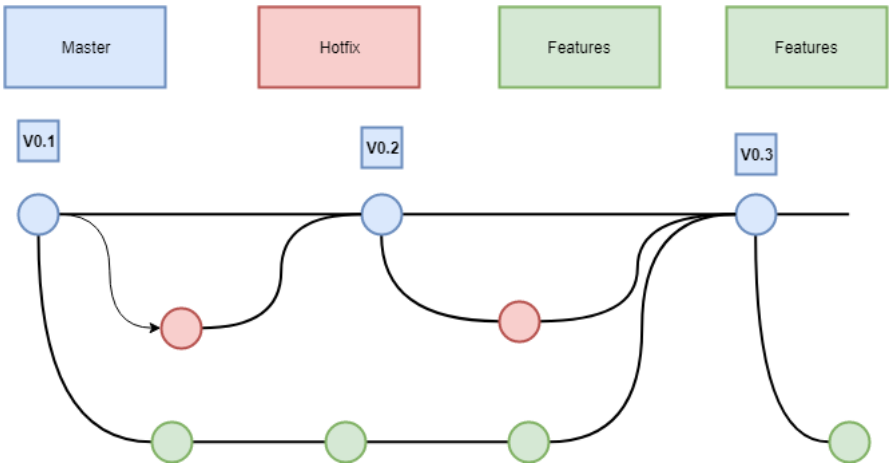

Figura 2: Patrón Derivado

2.2. Herramientas y tecnologías

En esta sección vamos a hablar de las tecnologías principales que se van a utilizar en este proyecto. Se va a intentar dar una breve descripción de las mismas, ya que el objetivo no es aprender a usarlas sino decir y nombrar el porque me van a valer en este proyecto, porque he decidido esas herramientas y no otras.

Alguna de las herramientas presentadas han sido claves para el desarrollo de dicha aplicación mientras que otras simplementa han servido de apoyo.

2.2.1. Taiga

Taiga [6] es una herramienta que hemos nombrada anteriormente que nos permite el control de proyectos, normalmente orientado a proyectos software, de una manera sencilla, cómoda y visual.

Taiga ofrece registrarse en su plataforma de manera gratuita. También da la posibilidad de

creación de proyectos públicos o privaos como en Github. Cada uno de estos proyectos te deja seleccionar si ha de seguir el modelo Kanban o Scrum, para poner la configuración por defecto del mismo.

Una vez creado el proyecto, el tablero principal de control de proyecto, con algunas tareas añadidas queda de la siguiente manera:

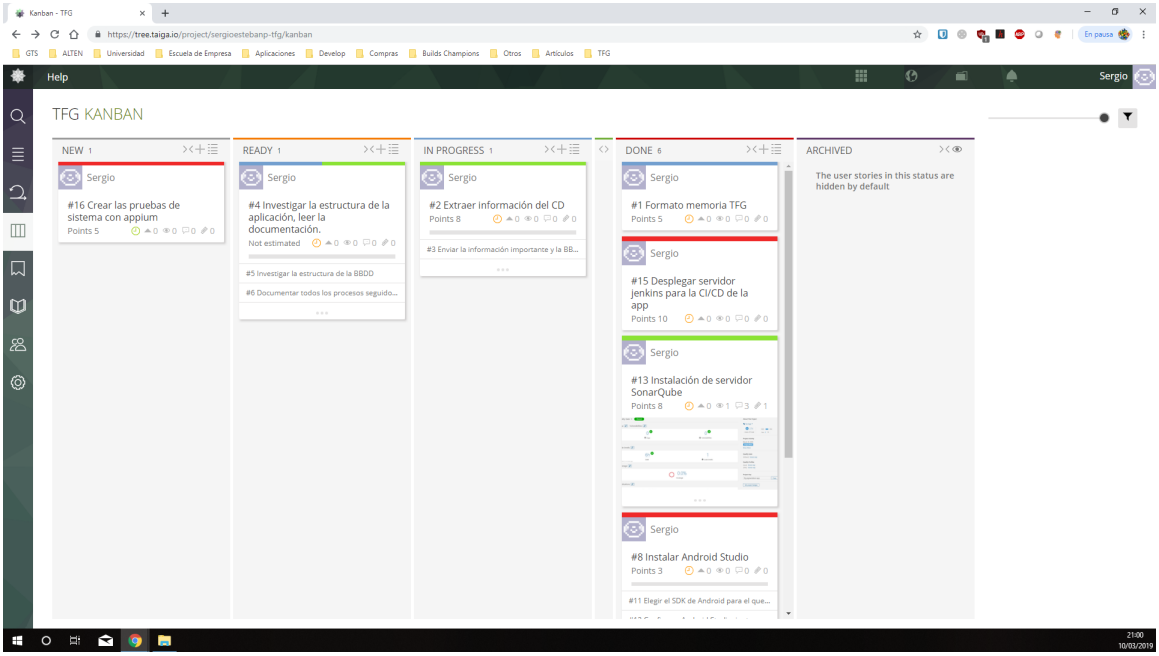

Figura 3: Tablero de Taiga.io

En la figura anterior podemos ver el proyecto publico donde se ira actualizando el proceso de creación de este mismo documento y de todo el trabajo de fin de grado []. Además podemos observar las principales columnas que hemos nombrado anteriormente cuando describíamos la metodología de trabajo.

Además del tablero principal, esta herramienta nos permite muchas mas opciones, como las siguientes:

- **Timeline**: en este apartado, podemos ver la cronología de todas las actividades realizadas en el proyecto, desde cuando se ha creado una tarea, cuando se ha actualizado el ultimo issue encontrado o cuando se ha añadido o quitado información a la wiki del proyecto.

- **Backlog**: aquí podemos encontrar todas las tareas, historias de usuario y issues que hemos ido creando en el proyecto. Se presentan de forma resumida, con el estado principal, fecha de creación, etc.

- **Issues**: los issues o problemas, son los diferentes problemas o bugs que nos vamos encontrando conforme el desarrollo de la aplicación se va realizando.

- **Wiki**: nos proporciona una herramienta de documentación interna para el proyecto. Podemos asemejar esta wiki a Confluence [14], que es una herramienta de pago creada por la empresa Atlassian, para tener información y documentación en la nube. Nosotros usaremos esta wiki para anotar cosas puntuales de desarrollo o configuración.

- **Members**: podemos ver a los demás miembros del equipo, en este caso solo voy a ser por lo que este apartado carece de especial importancia.

Podemos encontrar el proyecto de este trabajo de fin de grado en la siguiente dirección [1].

2.2.2. SonarQube

SonarQube [7] es una herramienta que nos permite obtener métricas y medidas de la calidad de nuestro código. Nos permite obtener métricas generales de la calidad el código, a nivel de buenas y malas practicas que estamos cumpliendo, posibles vulnerabilidades junto con el nivel de criticidad de las mismas, nivel de complejidad de diversos tipos, como por ejemplo la ciclomática [2].

He decidido integrar SonarQube y otras tecnologías en contenedores Docker [15] En realidad no se gana demasiado dockerizando las herramientas, pero si que es verdad que permite una mayor flexibilidad, a la hora de recrear esa instancia, simplemente tenemos que copiar y pegar el bakcup de la imagen. Dockerizar los servidores en contenedores individuales es una buena medida de seguridad si la aplicación estuviera en producción y se sirvieran los servicios hacia el exterior.

Nosotros vamos a integrar SonarQube y medir la calidad de nuestro código de 3 maneras diferentes. La primera y mas directa es en nuestro proyecto de Android Studio junto con Gradle [16]. Ya que los proyectos de aplicaciones móviles de Android se hacen con el constructor Gradle, y Sonar (acortación de SonarQube de aquí en adelante) tiene integración con dicha herramienta, solo tenemos que crear el proyecto en el servidor previamente desplegado en docker e introducir las siguientes lineas de código en el fichero build.gradle de nuestra aplicación, mostrado en la figura 6.

Creamos la tarea en gradle correspondiente para enviar la información a nuestro servidor de análisis y el resultado es el mostrado en la siguiente figura:

[1] https://tree.taiga.io/project/sergioestebanp-tfg/kanban

[2] La complejidad ciclotímica se mide en la cantidad de caminos individuales que se pueden recorrer en nuestro código. Cuanta menor complejidad ciclotímica mejor es la calidad del código

```
1  // Top-level build file where you can add configuration options common
2  // to all sub-projects/modules.
3
4  buildscript {
5      repositories {
6          google()
7          jcenter()
8
9      }
10     dependencies {
11         classpath 'com.android.tools.build:gradle:3.3.2'
12
13         // NOTE: Do not place your application dependencies here;
14         // they belong
15         // in the individual module build.gradle files
16     }
17 }
18
19 plugins {
20     id "org.sonarqube" version "2.6"
21 }
22
23 allprojects {
24     repositories {
25         google()
26         jcenter()
27
28     }
29 }
30
31 task clean(type: Delete) {
32     delete rootProject.buildDir
33 }
```

Figura 4: Configuración del fichero build.gradle

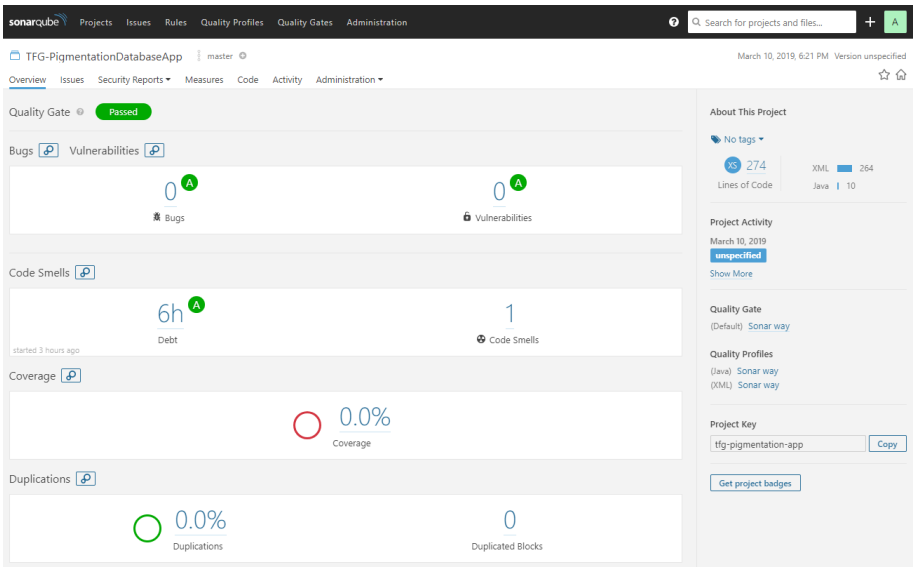

Figura 5: Primer análisis del código inicial generado por Android Studio

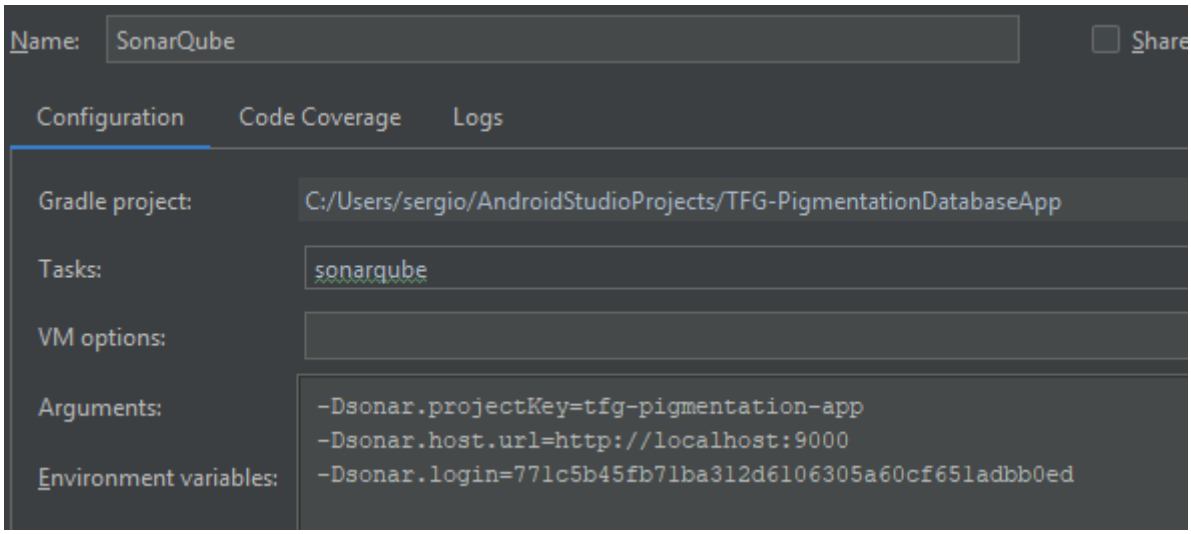

Figura 6: Tarea Gradle para Sonar

Una vez tenemos estos parámetros colocados y el servidor funcionando, podemos pasar a la configuración de SonarLint.

2.2.3. SonarLint

SonarLint [8] es un plugin para los entornos de desarrollo proporcionados por JetBrains, como Intellij IDEA o Android Studio. SonarLint nos proporciona la potencia de SonarQube pero sin tener que estar ejecutando el codigo en el servidor de manera constante. Lo que hacemos es configurar nuestro servidor de SonarQube en el plugin, y cada vez que abrimos un fichero de tipo

Java, el plugin pasa el fichero por el servidor y nos devuelve los resultados de una manera muy agradable. Podemos ver un ejemplo del uso de SonarLint en la siguiente figura:

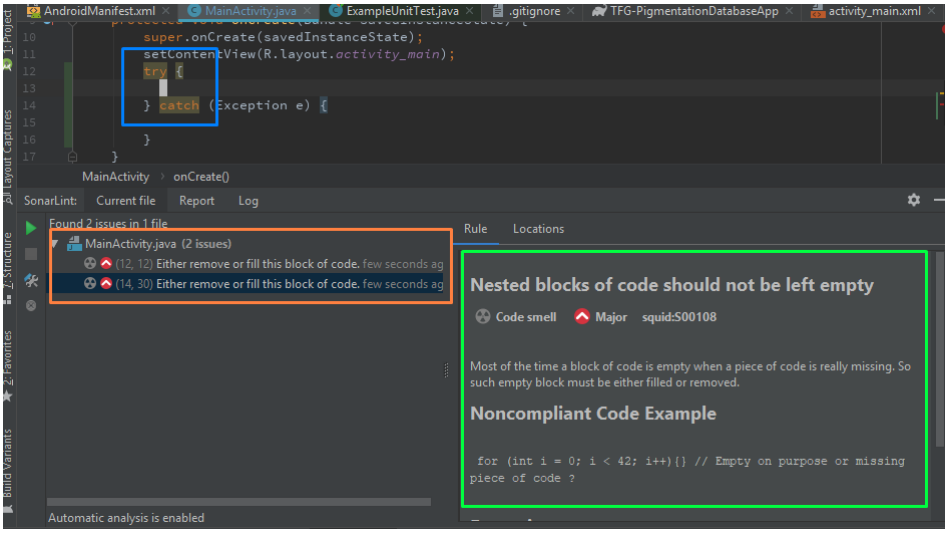

Figura 7: SonarLint en Android Studio

En el recuadro de color azul, podemos apreciar como el plugin nos muestra las vulnerabilidades de código. En el recuadro naranja nos lista todas las posibles vulnerabilidades o fallos que tiene nuestro software, y en el recuadro verde nos pone lo que significan, argumentos de porqué ese código no es de calidad y lo más importante, es que nos pone ejemplos para solucionarlo de manera sencilla y eficaz.

2.2.4. Jenkins

Jenkins [9] es una herramienta basada en la integración continua de productos software y en el desarrollo continuo. En nuestro caso lo vamos a usar como tal. Como todo producto software, nuestra aplicación va a requerir pruebas, tanto unitarias como de otros tipos. La idea es meter toda esta suite de pruebas en jenkins e integrarla tanto con github como con sonar. Lo que vamos a hacer es definir una serie de casos de prueba, tanto a nivel unitario como test de regresión, para que cada vez que el código se vea modificado por una Pull Request (nada más allá de una solicitud de cambio), la aplicación tenga que pasar los test satisfactoriamente para comprobar que no hemos roto lo anterior. Esto es para lo que vamos a usar Jenkins principalmente. Además como Jenkins tiene muchas utilidades, también podemos integrar SonarQube para que nos notifique en caso de que en una Pull Request la calidad de nuestro código se vea afectada a peor.

Un ejemplo de la estructura y flujo que sigue Jenkins es el siguiente:

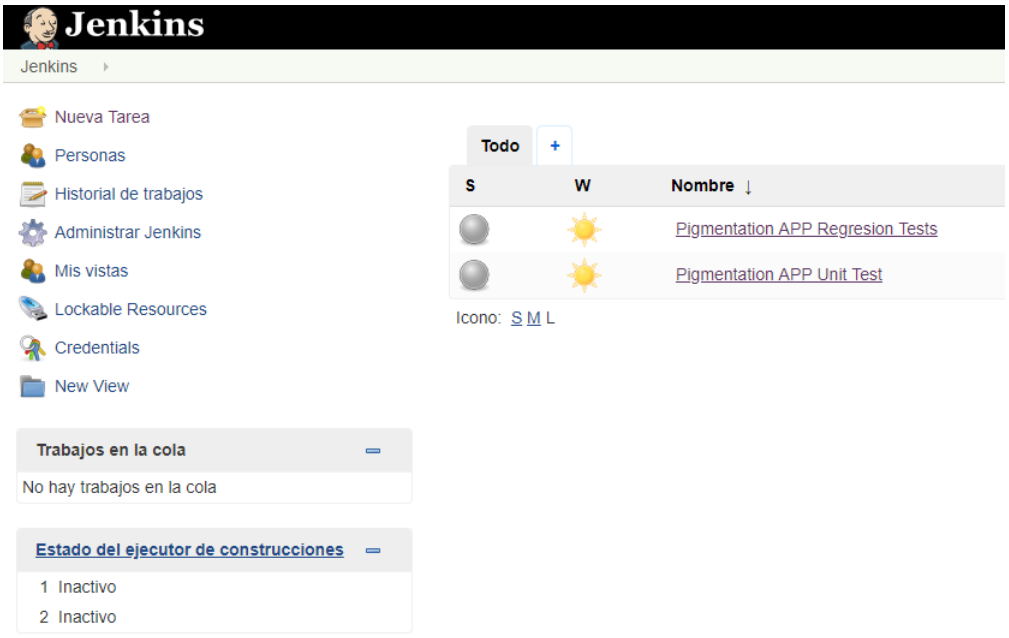

Figura 8: Flujo seguido por Jenkins

En Jenkins nosotros lo que definimos son un flujo por el que pasará nuestra aplicación. En el caso de nuestra aplicación seguramente tendremos dos flujos diferentes, uno para pasar los test unitarios cada vez que hagamos una propuesta de cambio a nuestro código; y otro flujo separado que serán las pruebas de regresión. Las pruebas de regresión las haremos cuando tengamos una base de funcionamiento de la aplicación, ya que las haremos sobre la interfaz. Por ejemplo la descripción de los flujos seguidos serían:

- **Pipeline ejemplo**: Obtener el código → pasar las pruebas unitaras → pasar las pruebas de calidad → generar los reportes → desplegar los cambios o dar el visto bueno.

Los flujos de Jenkins se describen mediante unos ficheros llamados Jenkinsfile, también pueden ser especificados en el propio servidor, pero no es una practica recomendable. Normalmente se integra en un archivo separado en el repositorio, precisamente para tener un seguimiento de los cambios que se hacen en el mismo. Dichos ficheros son escritos en groovy, que se basa en Java, pero simplificado.

Un vistazo al panel general de Jenkins, es el siguiente:

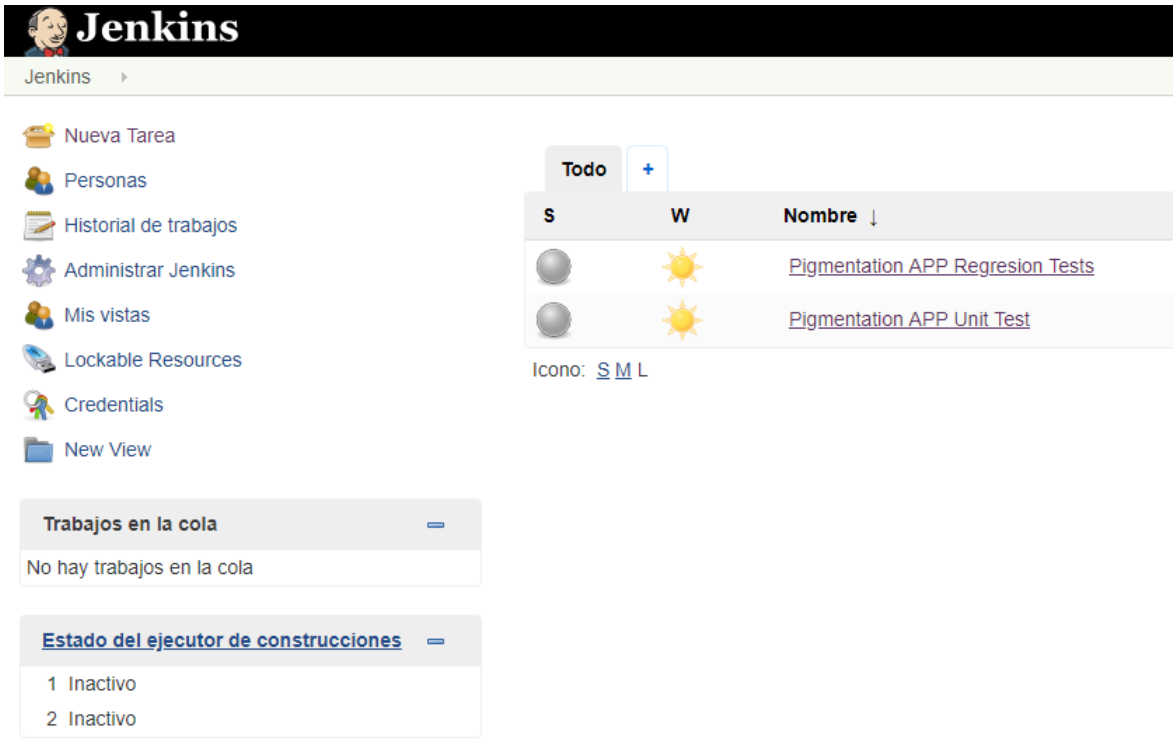

Figura 9: Panel General de jenkins

2.2.5. Android Studio

Android Studio [10], no hay mucho que decir, simplemente que es un entorno de desarrollo integrado que trae todas las herramientas necesarias casi de manera nativa la la programación y testeo de aplicaciones para dispositivos multiplataforma, como pueden ser los productos del sistema operativo Android. Las principales herramientas que utilizaremos serán las ejecuciones gradle automáticas, la capacidad de simulación de un dispositivo de la familia Android de manera casi nativa, el debugger que tiene, junto con el profiler son de gran calidad desde mi punto de vista, la integración con cantidad de plugins como el ya mencionado de SonarLint o VIM son también de una calidad excelente. En general para la realización de este proyecto me he decantado por Android Studio por la centralización de las herramientas que trae el propio IDE.

Al igual que estamos desarrollando con este IDE se pueden configurar otros como Intellij o Visual Studio Code para trabajar de manera similar.

En mi caso voy a utilizar Android Studio por la familiaridad a la hora de trabajr con él.

Además incluye ciertas herramientas de manera nativa, que de otra tendríamos que instalar a parte. Estas herramientas son por ejemplo el gestor del Android Software Development Kit [23] y el Android Virtual Device emulator (AVD) [22].

2.2.6. Github

Github [11] es generalmente conocido (o debería serlo) para todos los desarrolladores que trabajan en el gremio. Será utilizado como gestor de versiones de código y gestor de cambios del proyecto. Como hemos mencionado anteriormente en la metodología (Sección 2.1), vamos a seguir un patrón para la creación de ramas y gestionar los cambios en el código. Github nos permite seguir estos patrones e implementarlos de una manera sencilla y eficaz. Además cabe destacar que Android Studio tiene integración con Github y permite de manera sencilla hacer commits, cambiar entre ramas y demás operaciones importantes.

El repositorio de código de nuestra aplicación se encontrará en la siguiente dirección:

https://github.com/SergioEstebanP/TFG-PigmentationDatabaseApp

2.2.7. Overleaf

Overleaf [13] es un editor de textos online que nos permite generar documentos usando LaTeX. Por comodidad a la hora de gestionar la configuración del documento, se ha determinado usar latex como sistema de composición de textos, ya que word no ofrece las características necesarias para ello. Por lo menos no para mi.

Algunas de las razones elegidas son:

- Gestión automática de contenidos tales como figuras, tablas, títulos o secciones.
- Generación y gestión fácil, automatice y simple de la bibliografía y las referencias.
- Facilidad para insertar formulas matemáticas en caso de que fuera necesario.
- Configuración global del documento mas sencilla y flexible que en otros sistemas como word.
- Inadaptación de cambios gestionada por el propio sistema.
- Overleaf tiene plugin para escribir utilizando los atajos de VIM.

El enlace al proyecto de Overleaf, donde se encuentra este mismo documento es el siguiente:

https://es.overleaf.com/read/tjcbwvvdxwtq

Overleaf permite la edición múltiple de documentos y además también tiene un gestor de cambios integrado. El problema es que la línea de cambios en el tiempo que guarda es relativamente pequeña, y además muestra los cambios de una manera poco intuitiva y eficaz. La ventaja frente subir el código fuente a Github es que sigue los cambios de manera automática, mientras que subiéndolo a Github hay que hacerlo manualmente.

2.2.8. SQLite

SQLite [12] es un sistema gestor de base de datos muy sencillo, que integra Android por defecto. La idea es que la base de datos que tenemos que explotar esté integrada junto con el código de la aplicación. La base de datos parece pequeña, por lo que no ocupará nada más que unos pocos de Mega Bytes. Además, como el fichero generado para usar la aplicación es un binario, la información no será accesible al exterior. La información solo será proporcionada por la aplicación y de la manera propuesta en los capítulos posteriores.

Como describiremos en los capítulos posteriores, una de las primeras tareas es investigar la estructura de la base de datos y exportarla a SQLite o MySQL, ya que son SGBD a los que estoy más acostumbrado y necesarios para la compatibilidad con y buena comunicación de datos con la aplicación.

2.3. Balsamiq Mockups 3

Esta va a ser la herramienta utilizada para el desarrollo de los bocetos de la aplicación. Las opciones eran o la plataforma draw.io el programa de escritorio balsamiq mock up [18]. Como en anteriores proyectos ya he usado esta herramienta es la que usaré en este también. A grandes rasgos nos permite diseñar de una manera sencilla y rápida todo tipo de interfaces y entradas de datos visuales para un usuario. Además nos permite simular de una manera bastante sencilla las diferentes interacciones que hay entra las pantallas de una misma aplicación o página web.

Un ejemplo de la interfaz que nos ofrece esta aplicación es la que se observa en esta figura:

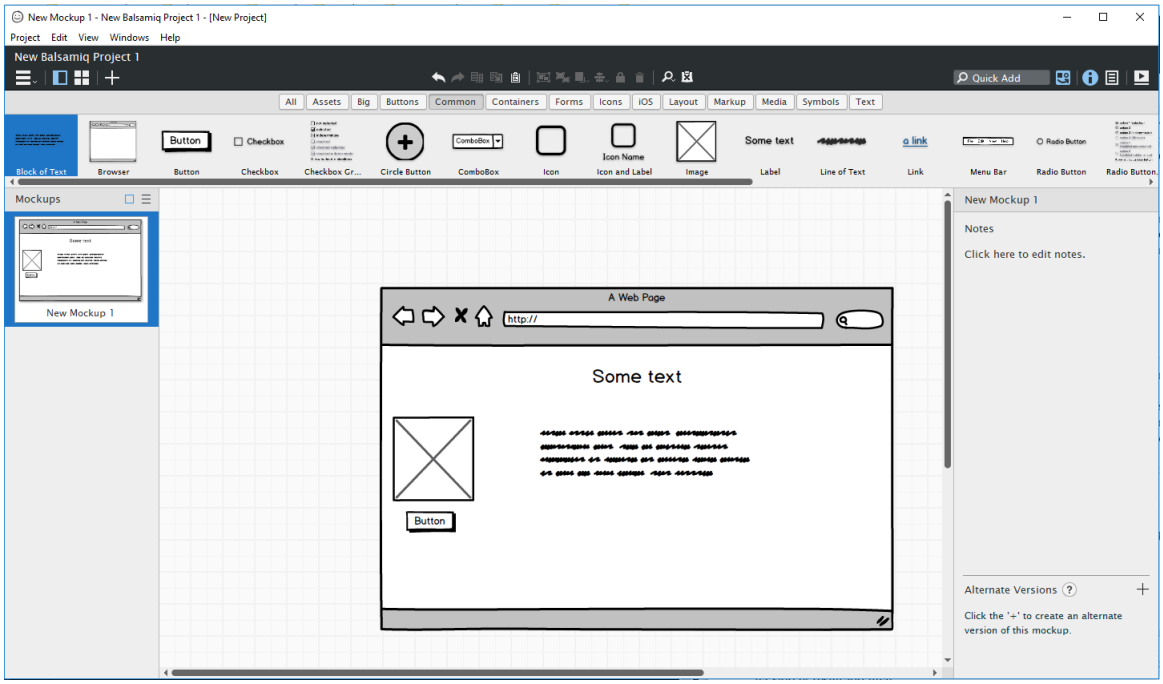

Figura 10: Interfaz principal de Balsamiq Mockups

2.4. Resumen de la memoria

En esta memoria se presentan de forma clara el desarrollo de la aplicación presentada en los puntos anteriores. La idea de esta memoria no es solos describir el desarrollo sino también documentar todos los problemas que se han tenido durante el proceso, como se han resulto y cual ha sido el resultado final.

En esta memoria, a modo de guía se encuentra la siguiente información, ordenada aproximadamente de la siguiente manera:

- Introducción y presentación de las herramientas: es la sección que estamos cerrando, simplemente se ha hecho un repaso de las herramientas que se están usando para el proyecto, para que luego cuando se hable de ellas, no se tenga duda alguna de porque las estamos utilizando o porque no las hemos nombrado anteriormente.

- Investigación de los documentos e información que existen a cerca de la aplicación y de la base de datos. Primero haremos una pequeña investigación para saber en que formato esta la base de datos actualmente, como importarla en caso necesario a un SGBD que nos sea mas adecuado para su tratamiento y además revisar la documentación actual. Dicha base de datos fue explotada anteriormente por una pagina web (año 2000), de la cual existe documentación a día de hoy. Hay que revisar esa información y ver si se puede aprovechar algo de lo que hay ya implementado o simplemente orientar el trabajo para implementar una funcionalidad u otra.

- Definición del objetivo y del alcance del proyecto, creación de los primeros diagramas, bocetos y modelos iniciales. Presentación al cliente para conocer su impresión inicial. En esta parte no solo vamos a centrarnos en los modelos, también desarrollaremos algo de código, como prueba de concepto para ver si todo funciona. Podemos hacer esto ya que seguimos un modelo de desarrollo ágil, lo que nos permite empezar a desarrollar productos software funcionales en poco tiempo e ir presentándoselos al cliente de manera regular. Esta fase estará acompañada de material gráfico, mostrando los diferentes bocetos y diagramas iniciales que compondrán la aplicación.

- En las subsiguientes partes de la memoria, se ira directamente a la implementación y a la evolución de las fases iniciales. Es evidente que un sistema de software va evolucionando con el paso del tiempo y conforme se van entendiendo los requisitos y las preferencias de funcionamiento o estéticas del cliente. Todo esta evolución, los problemas que van surgiendo y las decisiones tomadas son el objetivo de esta memoria.

- Al final de la memoria podemos encontrar las conclusiones finales. Las conclusiones no solo se van a centrar en si hemos logrado el objetivo principal que es el desarrollo de la aplicación como tal, sino que se hará un análisis introspectivo tanto al flujo de desarrollo como a las diferentes sensaciones que se han sentido durante las fases de las que ha constado el proyecto. Intentando saber si las decisiones que se han tomado han sido las correctas o no. Evidentemente este apartado constará de una parte en la que se indicara si hemos logrado los objetivos principales.

3. Planificación y gestión del proyecto

Los proyectos de software, evidentemente son proyectos ante todo, y por lo tanto van a tener características y procesos comunes a cualquier proyecto de ingeniería en general. Algunos de estos procesos que vamos a tratar en esta sección son: breve resumen de la necesidad de planificación, estimaciones acerca del tiempo que ha costado desarrollar la aplicación y en general invertido en el proyecto, estimaciones de costes y recursos necesarios para llevar a cabo dicho trabajo, estimación de los posibles riesgos que pueden surgir junto con sus posibles planes de contingencia y para finalizar hablaremos de la necesidad de la calidad del producto entregado cuando el proyecto tiene un marco temporal fijo y dinero limitado, que son las principales restricciones de los proyectos de este tipo.

3.1. Necesidad de planificación

Los proyectos software, como su nombre bien lo indica son proyectos, aunque a lo largo de la historia esta ultima parte se olvida a menudo. Desde que la informática comenzó en sus días, las empresas se embarcaban en grandes macro proyectos de cientos de miles de euros, pero la realidad es que muchos de ellos fracasaban y la principal de las razones era la falta de una adecuada planificación. Por lo menos con el tiempo, tanto las empresas como los profesionales y expertos en planificación han conseguido que esto no pase tanto, es decir, hay una tendencia alcista en lo que a éxitos en los proyectos de software se refiere, junto con un descenso significativo de los proyectos que al final no se acaban por unas u otras razones.

La planificación es necesaria en todos los niveles, no hay ningún proyecto grande ni pequeño que se pueda empezar sin una planificación mínima. Incluso el proyecto más pequeño parte de una idea inicial que una persona tiene estructurada en la cabeza, lo cual es planificación.

Cuando intentamos abordar proyectos de software que involucran a cientos de personas, recursos, cantidades grandes de dinero, se espera que ese proyecto salga adelante, pero la verdad es que ese proyecto solo saldrá adelante si se ha planificado y ejecutado de acuerdo a dicha planificación.

Uno de los principales problemas de la falta de planificación en los proyectos de software es la invisibilidad, por lo general planificar la construcción de un puente o el propio proyecto es mucho más vísale que la construcción de un artefacto de software. Otra de las principales características que dificultan la planificación es la complejidad. No es lo mismo intentar coordinar a 2 personas para que levanten un muro, que desarrollar una aplicación multiplataforma, están objetivamente demostrado que los proyectos de software son más complejos que los tradicionales por unidad de coste.

Estas y otras características son las que al final llevan a tener una planificación de proyectos pobre o deficiente, lo cual puede desembocar en la no realización del proyecto, o en la realización del mismo pero son sobrecostes o funcionalidad que no se han llegado a desarrollar.

3.2. Definición del alcance y objetivos

3.3. Tiempo de desarrollo

La estimación en genera es una tarea complicada y difícil, yo en este documento voy a intentar reflejar mi mejor aproximación del tiempo que se va a invertir en este proyecto y además se contrastará con el tiempo que hasta ahora se está invirtiendo.

Este proyecto cuenta con un marco temporal que en principio se considera fijo. Pongamos que la fecha límite que no se puede rebasar, para la cual todos los entregables y artefactos del proyectos han de estar terminados es el:

3.4. Recursos y costes

3.4.1. Recursos y tipos

En lo que a estimación de recursos y costes se refiere, este proyecto es modesto, es decir que no va a disponer ni depender de caros y grandes recursos. Podemos tener varios tipos de recursos, al ser un proyecto pequeño, se van a dividir en 3 grandes categorías:

- **Personales**: a nivel de personal solo se involucran a 3 personas reales. Una es el director del TFG, a la cual no se le puede contar casi como recurso, ya que es mas la persona que desbloquea los principales problemas que nos podemos ir encontrando conforme vamos realizando el proyecto. Otra de las personas es el profesor asociado al cual va destinada la aplicación, esta persona si que es más importante porque hay otros recursos que han dependido de ella. Otro de los recursos personales es el propio autor del documento, el cual ha tenido que poner en práctica conocimientos de diseño de interfaces, software arquitectura, redacción... para lograr este documento. Luego haremos los cálculos de coste, de momento simplemente estamos planteando los recursos.

- **Físicos**: a nivel de recursos físicos solo vamos a precisar de un ordenador para realizar el proyecto y la documentación del mismo, el ordenador no tiene que ser demasiado potente ya que solo tiene que cargar un IDE para desarrollar en aplicaciones Android. También podemos tener en cuenta los periféricos asociados y finalmente podemos añadir el móvil de pruebas en el que se van a realizar las principales pruebas físicas durante el desarrollo de la aplicación.

- **Datos**: voy a considerar como recursos los datos disponibles. Cuando empezó el trabajo la base de datos venía en forma de datos almacenados en un CD, pero en el futuro se ha visto que había por ejemplo falta de ese tipo de recursos.

- **Otros**: podemos entrar a considerar recursos que para este proyecto no tiene casi sentido tener en cuenta. Dichos recursos pueden ser la electricidad gastada durante se desarrollaba

el proyecto, o simplemente la estancia física donde se realizaba la acción. Repito que en este caso no tiene sentido tenerlos en cuenta pero si que me gustaría que quedase claro que en proyecto más grandes con muchas más personas, este tipo de recursos hay que contabilizarlos, controlarlos y tenerlos muy en cuenta (el precio de alquiler de una oficina puede disparar el precio de desarrollo por ejemplo).

3.4.2. Estimación de los costes

Intentar dar una estimación lo más adecuada posible de el coste de desarrollo total de este proyecto.

1. **Costes físicos**: estimar los costes de los recursos físicos anteriormente nombrados mediante la siguiente suma:
$$C_{fisico} = 550 + 20 + 15 + 150 = 735EUR$$
El desglose sería el siguiente:

   a) **Ordenador**: por unos 550 EUR se pueden encontrar ordenadores con las prestaciones que nosotros buscamos. Mas o menos tienen que tener 8GB de RAM y con un procesador del tipo Intel Core i3 o i5 es suficiente. Con un almacenamiento de 500GB de disco HDD (no se precisan tantos) se puede desarrollar perfectamente este trabajo en condiciones buenas.

   b) **Ratón**: 20 EUR es el precio de mercado medio de un ratón inalámbrico para usarlo con el ordenador.

   c) **Teclado**: 20 EUR es el precio de mercado medio de un teclado externo para usarlo con el ordenador.

   d) **Móvil**: 150 EUR es el precio medio de un móvil con prestaciones normales, es decir entre 1GB y 2GB de RAM, almacenamiento ROM de entre (y 16 GB y un procesador, junto con pantalla y cámaras de prestaciones medias.

2. **Costes personales**: para los costes personales primero tenemos que tener una buena estimación temporal, que en nuestro caso es: 34 días. Para ver cuanto cuesta pagar a una persona y que desarrollo dicho proyecto vamos a suponer que la hora a la que ese trabajador ha desarrollado el proyecto ha sido pagada a: 5 EUR, que dedicando una jornada laboral completa suponen unos beneficios de 40 EUR/dia y 1200 EUR/mes (no vamos a contabilizar los impuesto para facilitar las operaciones). Como nuestro trabajador va a dedicar unos X días, el coste de esa persona es aproximadamente:
$$C_{personal} = 34dias \cdot 40EUR/dia = 1360EUR$$

3. **Costes totales**: los costes totales son los siguientes:

$$C_{total} = C_{fisico} + C_{personal} = 735 + 1360 = 2095EUR$$

3.5. Tratamiento de los riesgos

La gestión de riesgos es un tema complicado, que intentaré abordar según la mayor brevedad y facilidad posible. Aunque la planificación de un proyecto se basa en describir, planificar, asignar y revisar las tareas y recursos conocidos, existen muchos recursos desconocidos o difíciles de establecer que pueden incidir directamente en la consecución de los resultados en el tiempo previsto. Un riesgo implica dos factores claves:

- **Incertidumbre**: probabilidad de que dicho riesgo se manifieste con el paso del tiempo.
- **Pérdida**: si el riesgo se convierte en una realidad, ocurrirán consecuencias o pérdidas no deseadas.

EL proceso de tratamiento de los riesgos tiene unas fases bien claras y definidas. Muchas empresas optan por la negación de los riesgos y la corrección de los mismos junto con los problemas ocasionados mientras estos van apareciendo. Este comportamiento no es mas que el reflejo de una pobre gestión y planificación ya que los riesgos son potenciales problemas que pueden surgir durante el desarrollo y si no se tienen en cuenta, pueden dar lugar a la no consecución de los objetivos del proyecto o incluso a la no finalización del mismo.

Vamos a dar unas pequeñas clasificaciones de posibles riesgos y luego presentaremos un análisis de riesgos inicial. Estamos por lo tanto en la **fase de análisis de riesgos**:

- *Directos*: depende enteramente de tu proyecto.
- *Indirectos*: tu proyecto no puede hacer nada, están fuera del alcance.

Otro tipo de clasificación que podemos dar:

- *De proyecto*: amenazan el plan del proyecto, aumentan el tiempo y el coste generalmente.
- *Técnicos*: potenciales riesgos de implementación, arquitectura, diseño ...
- *De negocio*: no entender bien los requisitos, no es de calidad, no es intuitivo.

Hemos hablado de que estamos en la fase de análisis de riesgos, una vez que hemos clasificado los riesgos deberíamos de estimar la probabilidad de que estos sucedan y el impacto al proyecto en caos de que se manifiesten. Al terminar esta fase deberíamos de pensar en posibles soluciones y alternativas para los problemas que se pueden ocasionar si dicho riesgo se manifiesta en el futuro. En conclusión la lista de riesgos, junto con la probabilidad, el impacto y la posible solución la podemos revisar a continuación. Además dicha lista está ordenada por prioridad:

- **Pérdida repentina de datos del PC**: 10 % de que ocurra - 0.9 de impacto - Posible solución: usar tecnologías de almacenamiento en la nube junto con la replicación de los datos en varios soportes de almacenamiento.

- **Mala gestión del tiempo**: 40 % de que ocurra - 0.95 de impacto - Posible solución: posponer la presentación de dicho trabajo.

- **Diseño poco intuitivo de la interfaz**: 10 % de que ocurra - 0.5 de impacto - Posible solución: organizar reuniones periódicas con el cliente para prevenir la aparición del riesgo.

- **Mal diseño de la base de datos**: 10 % de que ocurra - 0.2 de impacto - Posible solución: para prevenir podemos diseñar la base de datos de las maneras aprendidas en la carrera, aunque a veces no sean los mejores diseño nos podemos aferrar a buenos diseños de manera objetiva si hemos cumplido ciertas reglas.

- **Falta de datos**: 50 % de que ocurra - 0.8 de impacto - Posible solución: hablar con los profesores encargados para buscar o generar dichos datos. (manifestado)

- **Explotación pobre de la base de datos**: 10 % de que ocurra - 0.2 de impacto - Posible solución: organizar charlas con el cliente para que presente todas las ideas posibles.

- **Bloqueo a nivel técnico por el lenguaje de programación**: 5 % de que ocurra - 0.1 de impacto - Posible solución: usar otro lenguaje de programación o mejorar las habilidades en el que se está usando.

- **Mala arquitectura de la aplicación**: 15 % de que ocurra - 0.5 de impacto - Posible solución: aplicar patrones bien definidos y estudiados por personalidades importantes dentro del mundo de la arquitectura de software. Esta comprobado de manera objetiva que son buenas soluciones a problemas que ya se han planteado con anterioridad.

- **Bajo rendimiento**: 5 % de que ocurra - 0.3 de impacto - Posible solución: optimizar las búsquedas en la base de datos, estructuras de datos de la aplicación ...

- **Baja calidad del producto**: 20 % de que ocurra - 0.1 de impacto - Posible solución: someter a la aplicación a un sistema de integración continua donde se prueba una cierta y mínima calidad, tanto a nivel de código como de funcionamiento de software.

3.6. Control de configuraciones

La gestión de la configuración en este proyecto no va a ser su punto fuerte. El objetivo de la gestión de la configuración es mantener la integridad de los artefactos que se obtienen en cada uno de los proyectos que integran el área garantizando que no se realizan cambios no controlados y que todos los participantes del proyecto disponen de la versión adecuada de los productos y artefactos que manejan.

Así, entre los elementos de configuración software, se encuentran no únicamente ejecutables y código fuente, sino también los modelos de datos, modelos de procesos, especificaciones de requisitos, pruebas, etc. La gestión de configuración es una actividad continúa ya que se realiza durante todas las actividades asociadas al desarrollo de un sistema, y continúa registrando los cambios hasta que éste deja de utilizarse. Es una actividad de garantía de calidad que se aplica en todas las fases del proceso de ingeniería del software.

En nuestro caos no se va a usar un software específico de gestión de configuraciones. Además la gestión de configuraciones no es solo una base de datos donde se mantiene la información. Es un proceso continuo que avanza en paralelo al desarrollo del proyecto. Un ejemplo de una buena gestión de configuraciones habla de la necesidad de realizar auditorías, pero en un proyecto tan pequeño como el que vamos a desarrollar no tiene sentido.

En nuestro caso para mantener un mínimo control sobre las configuraciones y las versiones, no solo del código fuente y y de los ejecutables, sino también de los diagramas y de los documentos vamos a usar el mismo SCV que para el código fuente. Git.

3.7. Calidad de los productos entregados

Hemos hablado de la posibilidad de una pobre calidad de los productos que entreguemos al final de nuestro proyecto. Hemos hablado que es un riesgo potencial y que en principio puede ser evitado o como mínimo prevenido. Durante el desarrollo de la aplicación vamos a asegurar el control de la calidad sometiendo el propio software a un proceso de desarrollo e integración continua. Como hemos mencionado de pasada en la parte de tecnologías, vamos a usar diferentes tecnologías para asegurarnos de que la calidad de nuestra aplicación es la máxima que le podemos ofrecer a nuestro cliente.

La idea es que nosotros vamos a seguir un proceso de pruebas casi paralelo, aunque siempre un poco retrasado por definición en lo que a pruebas de sistema respecta, ya que probar la funcionalidad de algo, primero tenemos que desarrollarlo.

Una vez estén implementadas las primeas interfaces de usuario, con sus principales interacciones es cuando se introducirán y diseñarán las principales pruebas de sistema. Además de estas pruebas se desarrollaran, aunque en menor medida pruebas funcionales para los módulos que se considere oportunos y que tengan un comportamiento esperado para una entrada dada.

Todo este conjunto de pruebas se intentará que queden lo más automatizado posible. Como el objetivo principal es el desarrollo de la aplicación, no se le dará una importancia grande a esta parte, pero si que me gustaría introducir unas cuantas tecnologías de uso diario en el desarrollo ágil de productos software como puede ser Jenkins. Esta tecnología será la principal responsable de la automatización de toda nuestra suite de pruebas. Evidentemente no solo sirve para pruebas, sirve para automatizar procesos de software en general, otro ejemplo que podemos observar es el despliegue de un determinado servicio web en un entorno de desarrollo o producción cuando los cambios necesarios han sido realizados correctamente.

Podemos ver en la **Figura 11**, una aproximación del flujo al que se va a someter las diferentes partes de código que desarrollemos en nuestra aplicación.

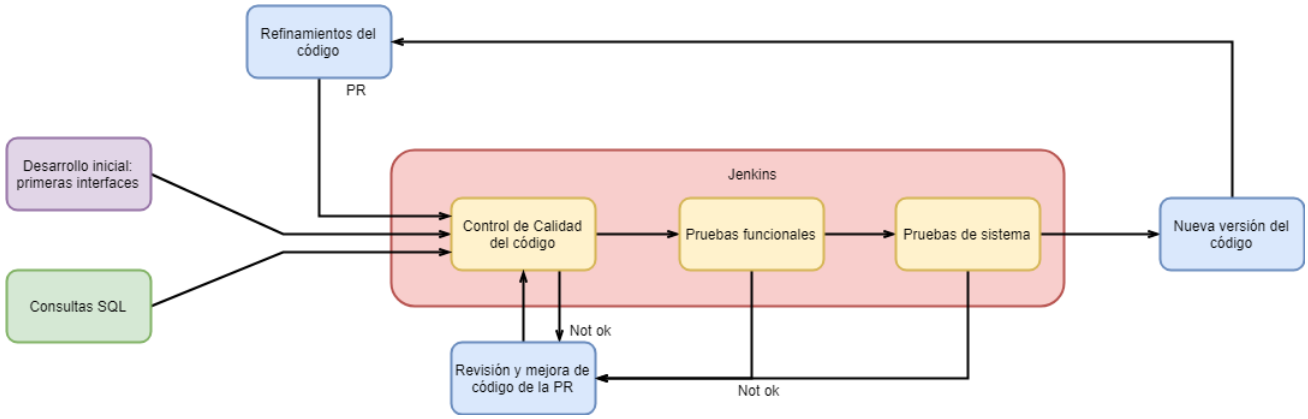

Figura 11: Flujo de Integración continua

4. Diseño

En esta sección vamos a hablar del diseño de la aplicación. Volveremos un poco al diseño más clásico que es el diseño en cascada, pero seguiremos hablando en todo momento de metodología Kanban.

4.1. Volviendo al desarrollo en cascada

Hemos dicho que la metodología de diseño que vamos a usar es Kanban, pero a mi personalmente me gusta siempre dar una serie de diagramas y de ideas procedentes del clásico desarrollo en cascada. Ya que las metodologías ágiles tienen unos entregables determinados y en ninguno de ellos aparecen los que yo voy a exponer a continuación, pero me parecen de una gran importancia y además expresan conceptos a nivel de arquitectura que a mi parecer deben de tener toda construcción de software.

Elicitación de requisitos

Aunque la metodología que está siguiendo el proyecto es Kanban, en todos los proyectos que he realizado me gusta introducir y recordar algunas cosas del proceso unificado. Por ejemplo la elicitación de los requisitos es algo que se hace en todos los proyectos. Por ejemplo en el caso de las metodologías ágiles el Product Owner es el encargado de hablar con el cliente y especificar al equipo cuales son los requisitos del mismo. Por lo tanto los principales requisitos básicos que ha de tener nuestra aplicación son los siguientes:

- **Requisitos Funcionales**:
  - El usuario tiene que poder ver todos los pigmentos.
  - El usuario tiene que poder ver información de las gráficas de datos de los pigmentos.
  - El usuario tiene que poder buscar en la base de datos por nombre.
  - El usuario tiene que poder buscar los pigmentos en base al elemento químico principal del pigmento.
  - El usuario tiene que poder buscar los pigmentos en base al color principal del pigmento.
  - El usuario tiene que poder añadir notas de cada uno de los diferentes pigmentos, para su propia colección.
- **Requisitos no Funcionales**:

- La aplicación tiene que ser capaz de ejecutarse en un dispositivo móvil.
- La base de datos tiene que estar implementada en SQLite, gestor principal de los dispositivos móviles.

Diagrama de Casos de Uso

El diagrama de casos de uso va a tener una gran semejanza al backlog y las historias de usuario que tenemos en las metodologías ágiles, pero desde mi punto de vista, este diagrama es intuitivo a la par que simple y nos permite saber lo que puede hacer un usuario de un golpe de vista.

Figura 12: Diagrama de Casos de Uso

Diagrama de paquetes de alto nivel

Uno de los diagramas que refleja muy bien la arquitectura que va a tener el sistema es el diagrama de paquetes, en el se suelen representar los diferentes paquetes y clases principales de las que se va a componer la aplicación. Podemos observar de un vistazo los principales patrones utilizados, así como las principales interacciones entre las diferentes partes que componen el sistema.

El diagrama de paquetes, con los principales patrones que más se adecúan a la solución es el mostrado en la **Figura 13**.

En esta arquitectura a alto nivel se puede apreciar el uso del modelo MVC, donde habrá clases que representen a la vista y accedan al modelo usando un controlador general de aplicación. Luego dicho controlador de aplicación es el que se encargará de interactuar con el resto de elementos y de proveer la información.

Una de las partes más importantes de esta aplicación es el acceso a los datos, ya que vamos a estar interactuando de manera permanente contra una base de datos. Dichas peticiones tienen que ser eficientes y realizarse en el menor tiempo posible y de una manera sencilla. El patrón por el que he optado ha sido un patrón basado en objetos, es decir nosotros vamos a crear un ORM (Object Request Manager), que se va a encargar de atender las peticiones de información y va a devolver las filas de la base de datos en forma de objetos.

Nuestra aplicación verá la base de datos como grandes listas de objetos. Cada de una base de datos será representada por un objeto diferente, con tantos atributos como columnas tenga dicha tabla. Cuando la aplicación necesite consultar una de esas filas, lo que hace nuestro ORM es devolver una lista con todas las filas de la base de datos en forma de lista de objetos. Entonces la aplicación se encarga de realizar la búsqueda dentro de esa lista.

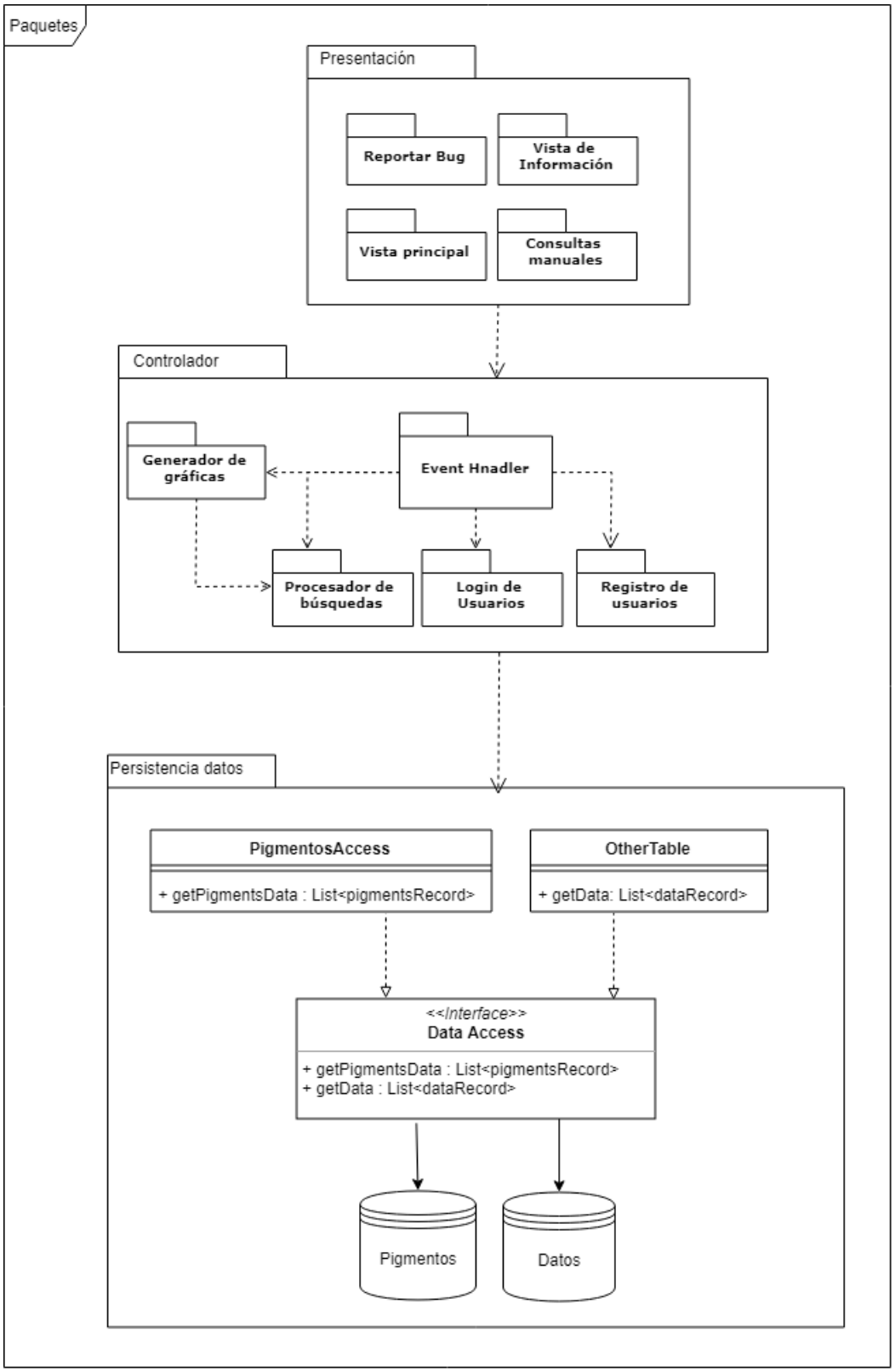

Figura 13: Diagrama de Paquetes a alto nivel

Diagramas de secuencia de alto nivel

En esta sección vamos a ilustrar como sería la interacción básica de los componentes de alto nivel de la aplicación. No buscamos una fuente detalla de los datos sino simplemente una aproximación a la solución que hemos planteado para que se entienda lo que hemos desarrollado en la aplicación, pero sin entrar a detalles de bajo nivel que pueden ofuscar la presentación.

En este caso vamos a especificar un diagrama de secuencia para el supuesto de que el usuario quiere observar las propiedades químicas de un pigmento determinado. Esta secuencia de actividades la podemos ver reflejada a modo de ejemplo en el diagrama de la **Figura 14**

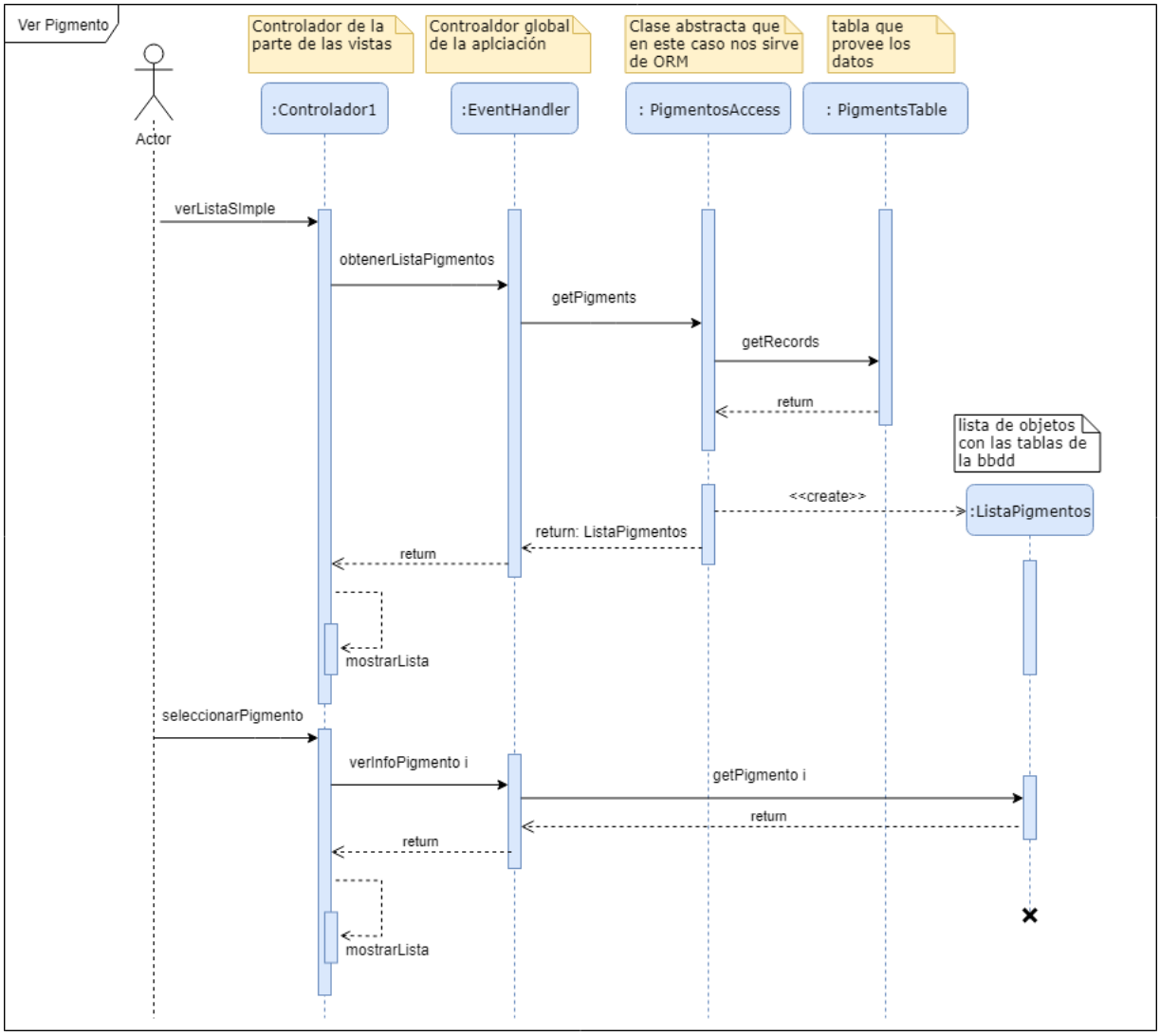

Figura 14: Diagrama de secuencia de alto nivel para ver la información de un pigmento

En este diagrama a de secuencias para uno de los casos más básicos, podemos observar como el actor interactúa con la aplicación y toca en el botón de ver todos los pigmentos, esta acción desencadena que el controlador del MVC (Model View Controller) [24] se ponga en contacto con el controlador principal de la aplicación. Dicho controlador interactúa con el ORM que hemos dicho

antes. Este ORM (Object Request Manager) [25] hace la consulta necesaria MySQL en la base de datos y le devuelve al controlador de aplicación una lista con todos los objetos representando las filas de la tabla que se haya consultado, en este caso la general de los pigmentos. El controlador de aplicación devuelve la información al controlador de la vista y presenta la información de una manera adecuada para el usuario.

Ahora el usuario selecciona un pigmento en concreto, para ello el controlador de las vistas se pone en contacto con el controlador de la aplicación, el cuál no necesita interactuar con la base de datos porque ya tienes los datos disponibles. Buscamos el pigmento en cuestión, devolvemos la información al controlador de aplicación y este devuelve la información al controlador de las vistas. Hecho esto la el usuario tiene a su disposición la información que quería.

Una vez que hemos presentado los principales diagramas e ideas en el formato tradicional, vamos a seguir con nuestra metodología inicial que es Kanban.

4.2. Definición de criterios

En Kanban, antes de definir las historias de usuario, tenemos que definir brevemente alguno de los criterios que usaremos en dichas historias y que son muy importantes.

Los más importantes que tenemos que definir son dos, los criterios de aceptación y los criterios de finalización.

4.2.1. Definición de hecho

La definición de hecho es una norma que se aplica a las Historias de Usuario, que desarrollaremos más adelante, pero para ello es necesario tener bien fijada esta definición con anterioridad. Normalmente esta definición es fijada de manera conjunta con el cliente final y el líder del equipo y muchas veces está presente el euqipo de desarrollo como tal para verificar las diferentes posibilidades, ya que nos permite:

- **Tener un producto potencialmente entregable y usable** al finalizar cada iteración. En el caso de este proyecto no tenemos iteraciones como tal, pero si nosotros establecemos unas ciertas normas que tiene que cumplir cada una de las tareas que vamos a desarrollar, cuando nosotros finalicemos esas tareas podemos decir que están acabas y entregadas de manera adecuada y conforman el producto final perfectamente entrgeable. De esta forma el cliente, conforme tengamos tareas acabadas puede tomar decisiones de qué desarrollar en el futuro y qué no de manera clara y sencilla.

- **Establecer unos criterios de calidad**: define que entregables y mínimos de calidad se tienen que cumplir en todos los objetivos/requisitos que se van a ir aceptando conforme el proyecto vaya avanzando en el tiempo.

Ahora que sabemos lo que son y las posibilidades de la definición de hecho, pasamos a ver cuales serán nuestros criterios de hecho:

1. El trabajo de cada miembro del equipo ha sido revisado y aceptado por lo menos por una persona más del proyecto, que en este caso pueden ser tanto el tutor del proyecto como el profesor asociado.

2. Todo el equipo considera que para cada objetivo/requisitos se cumplen sus criterios de aceptación al 100 %.

3. El requisito o tarea tiene que estar probado.

4. Si el requisito lo permite, tiene que superar todas las pruebas unitarias que le hayan sido diseñadas.

5. Si el requisito lo permite, tiene que superar todas las pruebas de aceptación que le hayan sido diseñadas.

6. Si el requisito lo permite, tiene que superar todas las pruebas de regresión que le hayan sido diseñadas.

7. El requisito o tarea tiene que estar documentado.

8. El requisito o tarea supera una calidad respecto a código del 80 %.

9. El product owner ha validado y aceptado el objetivo/requisito.

4.2.2. Criterios de aceptación

Mientras que los criterios de finalización tienden a ser criterios mas objetivos, medibles y triviales. Los criterios de aceptación dependen en gran medida del deseo e idea que el cliente tiene sobre el producto final.

Los criterios de aceptación definen los requisitos del Product Owner sobre cómo debe comportarse la aplicación para que una determinada acción se pueda llevar a cabo, normalmente por parte de un usuario de la aplicación. Generalmente ayudan al equipo de desarrollo a responder a las preguntas:

- **¿He construido el producto correcto?**

- **¿He construido el producto correctamente?**

Los criterios de aceptación deben describir siempre un contexto, un evento y la respuesta o consecuencia esperada del sistema. La forma más utilizada para describir los criterios de aceptación es conocida como Given-When-Then, que veremos más adelante cuando mostremos las historias de

usuario. Los criterios de aceptación son la clave de las historias de usuario, por lo tanto se tienen que cumplir todos los criterios de aceptación para dar una historia de usuario como terminada y correcta. El cumplimiento de los criterios de aceptación de las historias de usuario, a nivel de funcionalidad se verá de una manera muy clara cuando se presente la suite de pruebas asociadas a la aplicación. Como adelanto, dicha suite de pruebas se va a basar en un lenguaje llamado Gherkin en el que los escenarios de prueba se describen en un lenguaje de alto nivel basado en el Given-Then-When.

Habrá una asociación directa entre los criterios de aceptación de una historia de usuario y el test concreto que pruebe la funcionalidad de dicha historia de usuario. Todo esto lo podremos ver más en detalle en los capítulos posteriores, en concreto en el relacionado con las pruebas de la aplicación.

4.3. Épicas e historias de usuario

A la hora de estructurar el trabajo que tenemos que hacer en un proyecto de gran magnitud, tenemos que tener en cuenta desde las actividades más ambiciosas hasta las mas detalladas. Por eso, existen diferentes estructuras de datos y formas de organización que nos ayuda a conseguir dicho objetivo. Como bien nos dice el *Atlassian Agile Couch*, para conseguirlo nos podemos servir de los temas, iniciativas, épicas, historias de usuario y tareas. Podemos ver una descomposición visual de estos elementos en la **Figura 15**

Figura 15: Relación entre épicas, historias y temas

A modo de resumen, cada una de las estructuras significa:

- **Temas**: son grandes áreas de enfoque que abarcan a toda la organización.
- **Iniciativas**: son conjuntos de épicas que conducen hacia un objetivo común.

- **Épicas**: son grandes cantidades de trabajo que se pueden desglosar en un número de tareas más pequeñas, llamadas historias.

- **Historias**: son breves requisitos o solicitudes escritas desde el punto de vista del usuario final.

- **Tareas**: las tareas son actividades que hay que hacer para conseguir desarrollar las historias, no tienen porque estar escritas desde el punto de vista del cliente.

En este proyecto por temas de magnitud, ya que es un proyecto pequeño, solo vamos a contar con historias, tareas y como mucho con unas cuantas épicas que nos van a venir bien a nivel de planificación y explicación del proyecto.

4.3.1. Épicas

Las épicas que nosotros vamos a presentar en nuestro proyecto son las siguientes:

- **Desarrollo**: en esta épica se incluirán las tareas relacionadas con las actividades de desarrollo.

- **Documentación**: se relacionarán tareas de documentación de las diferentes partes que componen el sistema, tanto documentación interna como documentación externa.

- **Configuración**: se añadirán las tareas relacionadas con la configuración de entornos o sistemas necesarios para la consecución de los objetivos del proyecto.

- **Investigación**: puede que haya algunas tareas de investigación en ciertos puntos del proyecto, en esta épica habrá seguramente muy pocas tareas, pero también considero necesaria añadirla.

Cabe destacar que tanto las épicas del proyecto como las diferentes historias de usuario y tareas, las podemos

4.3.2. Historias de usuario

Primero vamos a describir brevemente lo que son las épicas y las historias de usuario. Que son la base de la metodología que estamos siguiendo.

Las historias de usuario son la base trabajo de nuestro equipo de desarrollo. Para presentarlas hemos hecho una pequeña plantilla que muestra las características esenciales de una historia de usuario, y tiene los siguientes campos:

- **Id**: cada una de las historias de usuario de nuestro proyecto va a tener un ID único que la

identificará. Esto mejora la búsqueda de las mismas cuando hagamos referencias y siempre sepamos de que elemento estamos hablando.

- **Prioridad Dev.**: número que seguirá la serie de Fibonacci (1, 3, 5, 8, 13, 21) que indicará la prioridad a la hora de desarrollar esa historia de usuario para los desarrolladores. Aclaramos que el 10 será la prioridad máxima y el 1 será la prioridad mínima.

- **Valor**: número que seguirá la serie de Fibonacci (1, 3, 5, 8, 13, 21) que indicará la prioridad a la hora de desarrollar esa historia de usuario para el cliente final. Aclaramos que el 10 será la prioridad máxima y el 1 será la prioridad mínima.

- **Estimación**: una ligera estimación del coste de implementación/finalización de esa Historia de Usuario. Podremos tener diferentes medidas, la mínima serán H (horas) y la máxima serán W (semanas). Ej.: 2H, 3D (días) o 1W.

- **Asignado a**: desarrollador que tendrá asignado esa historia de usuario, aunque las tareas en las que se subdivida puedan estar asociadas a otros desarrolladores.

- **Usuario**: el usuario al que va referida esta historia de usuario dentro de la aplicación final.

- **Descripción**: una breve descripción del objetivo que tiene que conseguir esa historia de usuario en concreto.

- **Dependencias**: las diferentes historias de usuario de las que depende la historia de usuario especificada.

- **Criterios de aceptación**: los criterios que marcarán cuando la historia de usuario podrá considerarse cerrada y acabada.

- **Épica relacionada**: se mostrará la épica con la que está asociada dicha tarea de usuario.

Podemos ver una plantilla del formato que van a tener nuestras historias de usuario en la siguiente **Figura 16**

Título de la historia				
Descripción: descripción breve de la historia de usuario que se va a desarrollar.		**Criterios de aceptación**: criterios de aceptación de la historia en cuestión. Seguirán el formato cuando-dado-entonces.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
02	5	13	3D	SE

Figura 16: Formato que seguirán las historias de usuario

Una vez que hemos definido el formato de nuestras historias de usuario, los criterios que van a tener y todas las partes de las que van a constar, podemos pasar a mostrar todas las que tenemos.

4.4. Backlog

El backlog es una de las herramientas mas importantes cuando hablamos de desarrollo de proyectos bajo el marco de las metodologías ágiles. El backlog es una estructura de datos, normalmente una lista que contiene todas las historias de usuario y tareas que se van desarrollando en la aplicación. El bakclog normalmente se ordena de manera inicial por la prioridad que el cliente asigna, pero sin embargo es común que dentro de esa prioridad del cliente se reordene también por la prioridad que los desarrolladores consideran necesaria. Esto se debe a que el cliente puede considerar muy necesaria una de las historias, pero sin embargo es muy fácil de desarrollar en comparación a otras tareas que pueden conllevar una carga de desarrollo mayor.

A continuación voy a mostrar el backlog con todas las historias de usuario que consideramos necesarias. Este backlog estará replicado en taiga.io. Además en el backlog de la aplicación se podrán observar las diferentes épicas, y tareas relacionadas con cada una de las diferentes historias, que también estarán en el backlog.

Consulta por coordenadas tricrómicas				
Descripción: el usuario tiene que ser capaz de obtener los datos a partir de las coordenadas tricrómicas proporcionadas.		**Criterios de aceptación**: dado que el usuario quiere consultar todos los pigmentos con unas determinadas coordenadas tricrómicas, cuando el usuario introduce los 3 valores: L, a y b, entonces el sistema muestra todos los pigmentos donde alguna de sus coordenadas coincide con las introducidas.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
01	13	21	2D	SE

Consulta por parámetros del Espectro Raman				
Descripción: el usuario tiene que ser capaz de obtener los pigmentos a partir de los datos del Espectro Raman introducidos.		**Criterios de aceptación**: dado que el usuario quiere consultar todos los posibles pigmentos dentro de un rango de tolerancia determinado, cuando introduce la posición que quiere explorar, el sistema muestra todas las posibles coincidencias.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
02	13	21	2D	SE

Consulta por el espaciado DHKL del difractograma				
Descripción: el usuario quiere obtener todas las posibles concordancias de pigmentos que cumplen un espaciado de DHKL proporcionado.		**Criterios de aceptación**: dado que el usuario quiere consultar todos los posibles pigmentos cuyo espaciado de DHKL se encuentra en un determinado rango, cuando el usuario introduce los datos, el sistema tiene que mostrar todos los pigmentos que cumplen dicha condición.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
03	13	21	2D	SE

Consulta por parámetros del Espectro de FTIR				
Descripción: el usuario tiene que ser capaz de obtener todos los pigmentos que tengan un pico en la posición del espectro FTIR introducido.		**Criterios de aceptación**: dado que el usuario quiere obtener todos los pigmentos que tienen un pico en el espectro FTIR en la región introducida, cuando el usuario proporciona los datos, el sistema tiene que mostrar una lista con todos los pigmentos que cumplen la condición.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
04	13	21	3D	SE

Consulta por elementos químicos				
Descripción: el usuario tiene que ser capaz de visualizar los pigmentos que contienen los elementos químicos proporcionados.		**Criterios de aceptación**: dado que el usuario quiere ver todos los pigmentos que contienen los elementos químicos seleccionados, cuando el usuario selecciona los elementos químicos y pulsa en buscar, el sistema muestra una lista de todos los pigmentos que cumplen la condición anterior.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
05	11	13	1D	SE

Consulta por nombre				
Descripción: el usuario tiene que ser capaz de recuperar todos los pigmentos que concuerden con el nombre proporcionado.		**Criterios de aceptación**: dado que el usuario quiere obtener una lista de pigmentos que concuerden con el nombre especificado, cuando el usuario introduce el nombre y presiona en buscar, el sistema muestra todos los pigmentos cuyo nombre contiene o concuerda con el pasado como parámetro de búsqueda.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
06	11	13	2D	SE

Consulta por color				
Descripción: el usuario tiene que ser capaz de recuperar todos los pigmentos que concuerden con el color proporcionado.		**Criterios de aceptación**: dado que el usuario quiere obtener una lista de pigmentos que concuerden con el color especificado, cuando el usuario selecciona los colores y presiona en buscar, el sistema muestra todos los pigmentos cuyo color coincide con el pasado como parámetro de búsqueda.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
07	11	13	1D	SE

Seleccionar vista rápida de pigmentos				
Descripción: el usuario quiere visualizar un lista completa y rápida de todos los pigmentos que hay en la base de datos actual.		**Criterios de aceptación**: dado que el usuario se encuentra en el menú principal de la aplicación y quiere ver una lista de todos los pigmentos disponibles, cuando pulsa en el botón de Pigmentos, el sistema muestra una lista con todos los pigmentos disponibles.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
08	13	11	2D	SE

*Seleccionar consulta simple de pigmentos ****				
Descripción: el usuario quiere realizar una búsqueda simple de pigmentos, no tiene porque saber ninguna característica fisicoquímica de los mismos.		**Criterios de aceptación**: dado que el usuario quiere obtener la información de un pigmento de manera rápida, cuando pulsa en el botón de consulta simple el sistema muestra las dos posibilidades de búsqueda simple, que es por color y por nombre, el usuario selecciona la que prefiera y ve la pantalla para introducir la información al sistema.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
09	13	13	1D	SE

Seleccionar consulta avanzada de pigmentos				
Descripción: el usuario quiere realizar consultas más complejas basadas en las características físicas y químicas de los pigmentos.		**Criterios de aceptación**: dado que el usuario quiere realizar consultas mas complejas para obtener pigmentos determinados, cuando el usuario pulsa en el botón de búsquedas avanzadas, el sistema muestra una lista con los diferentes tipos de búsquedas posibles, el usuario selecciona la que quiere, introduce los datos al sistema y este devuelve la información al usuario.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
10	13	21	4D	SE

Ver información de un pigmento				
Descripción: dado que el usuario quiere obtener la información completa y detallada de un pigmento de la base de datos, supondremos el caso más simple en el que el usuario conoce el nombre del pigmento en cuestión.		**Criterios de aceptación**: dado que el usuario quiere obtener la información detallada de un pigmento cuyo nombre conoce, el usuario pulsa sobre búsqueda simple, el sistema muestra las posibilidades de búsqueda, el usuario selecciona la búsqueda por nombre, el usuario introduce el nombre del pigmento y el sistema muestra la lista con los resultados, en caso de que sean varios.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
11	11	13	1D	SE

Reportar un bug				
Descripción: el usuario tiene que ser capaz de informar al creador de la aplicación en el caso de que detecte un fallo de funcionamiento de esta.		**Criterios de aceptación**: dado que el usuario ha detectado un fallo de funcionamiento, cuando presiona sobre el botón de reportar un fallo, el sistema muestra un pequeño formulario, cuando el usuario ha introducido la información pertinente y presiona sobre el botón enviar, el sistema muestra un mensaje de retroalimentación hacia el usuario.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
12	8	5	5H	SE

Añadir pigmento				
Descripción: el usuario es un científico y quiere añadir información acerca de un pigmento a la aplicación. Dicha información permanecerá en local, no se actualizará en la aplicación en producción.		**Criterios de aceptación**: dado que el usuario quiere introducir un pigmento en la aplicación, cuando pulsa sobre el botón de añadir pigmento, el sistema muestra un formulario donde se tienen que rellenar algunos de los campos, cuando el usuario presiona sobre guardar, el sistema muestra la información actualizada de dicho pigmento.		
ID	**Prioridad DEV**	**Valor**	**Estimación**	**Asignada**
13	5	5	1D	SE

Ahora que ya tenemos el backlog ordenado, sabemos las tareas que tenemos que desarrollar, como hay que hacerlo y en que orden, podemos pasar a empezar a desarrollar la aplicación y

pensar en cosas de más bajo nivel. También podemos observar, que estas historias de usuario tiene diferentes dependencias entre ellas, como es esperable en el desarrollo de una aplicación, por ejemplo podemos completar la parte de las consultas avanzadas si no tenemos el desarrollo de la interfaz principal.

La precedencia de tareas la podemos observar en la **Figura 17**

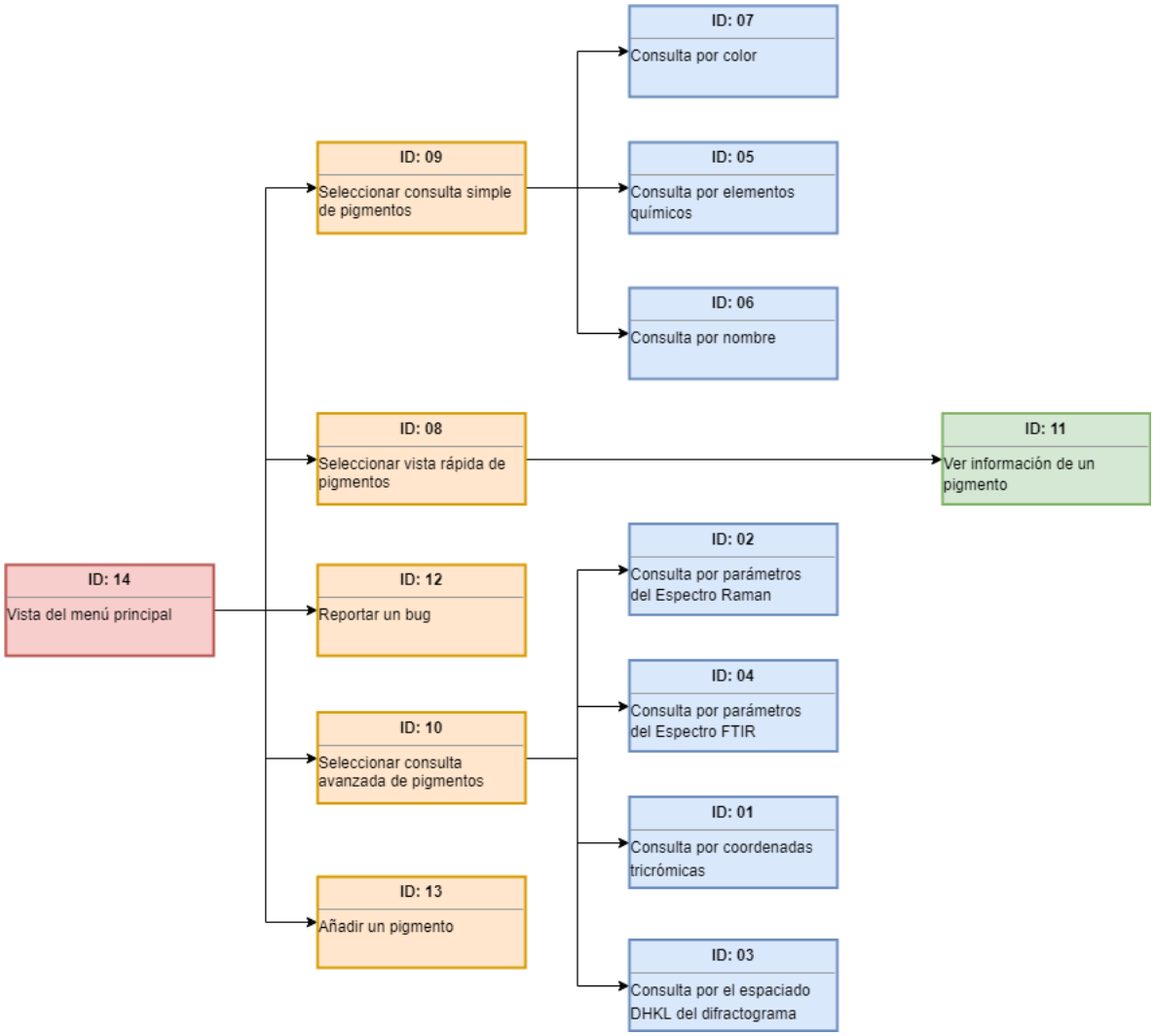

Figura 17: Diagrama de precedencia de las principales tareas

Muchas de las tareas, como la ID:07, ID:05 o ID:02 desembocan también en la ID:11, pero no están relacionadas porque no es estrictamente necesario que todas ellas estén terminadas para poder empezar la 11, la única que si que tiene que estar necesariamente terminada para poder empezar con la ID:11 es la ID:08.

4.5. Bocetos iniciales

Ahora que tenemos planificadas la gran mayoría de tareas principales y a más alto nivel, vamos a empezar con el desarrollo como tal de la aplicación. En esta sección presentaremos el logo de la app junto con el nombre y los diseños de las principales interfaces.

4.5.1. Logo de la aplicación y nombre

Una de las primeras cosas en las que pensamos cuando escuchamos aplicación móvil es en el nombre y el logo de las grandes aplicaciones. En la época actual los logos tienen a ser minimalistas, simples y con colores pastel y suaves. Así mismo los nombres de las aplicaciones tienden a ser cortos, ya que los dispositivos móviles suelen tener un tamaño reducido y no se pueden meter frases largas.

Logo

Los dos diseños iniciales para el logo de la aplicación fueron los siguientes:

Figura 18: Diseño 1 del logo de la aplicación

Figura 19: Diseño 1 del logo de la aplicación

Pero después de una reunión con el cliente final, se llego a la conclusión de que el logo debería de ir mas por el estilo de la paleta, pintada de diferentes colores:

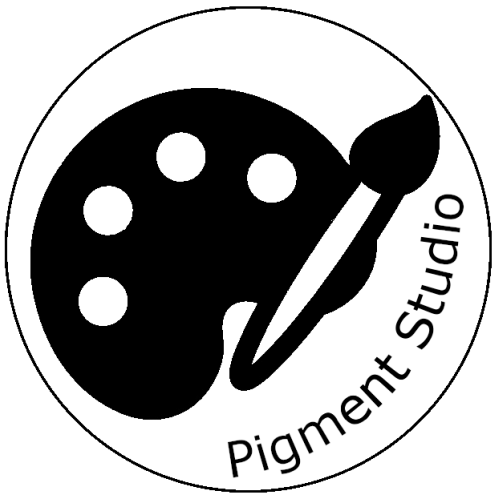

Figura 20: Diseño final del logo de la aplicación

En las reuniones posteriores se ha decidido dejar como logo una paleta pero coloreada, en lugar de en tonos blancos y negros. Y quitar el Titulo dentro del logo.

Nombre

En la sección anterior ya hemos dejado ver cual iba a ser el nombre de la aplicación. Una de las ideas iniciales era:

- **PigemtsDB**: pero ni al cliente ni al tutor del TFG les convencía esta opción, ya que al contener la acortación DB, procedente de Data Base incluía unas connotaciones un poco más técnicas.

Al final se ha dejado como **Pigment Studio** ya que tiene un significado ligeramente más general y no restringe a ningún tipo de colectivos.

Es un nombre corto y que haces una buena referencia a lo que el usuario se va a encontrar cuando abra la aplicaicón, que no es más que un estudio bastante avanzado y claro de los diferentes pigmentos que tenemos en el mundo.

4.5.2. Diseño de las primeras interfaces

Una de las partes más críticas cuando estamos desarrollando aplicaciones que va a utilizar el usuario final es la interfaz que van a tener que usar. Va a ser una pantalla contra la que el usuario va a interactuar bastante a menudo, por lo tanto tiene que ser cómoda, clara y sencilla. Algunas de las normas que tienen que tener el diseño de interfaces para que el usuario tenga una buena experiencia (UX) las podemos encontrar en la siguiente lista:

- La aplicación tiene que ser fácil de utilizar incluso si el usuario no tiene conocimiento previo de cómo se usa la misma.
- La aplicación tiene que ir fluida y responder en tiempos pequeños.
- La aplicación tiene que tener las opciones mas importantes resaltadas y en la parte derecha, cerca del pulgar.
- El logo de la aplicación tiene que ser atractivo y relacionado con la información que va a contener.

Podemos encontrar algunos consejos útiles y que yo he utilizado en las referencias de diseño de Adobe [26].

Las principales interfaces que vamos a encontrar en nuestro sistema son las siguientes, junto con una descripción de lo que harán.

Menú Principal

En el menú principal de la aplicación podremos encontrar las diferentes opciones que nos presenta para explotar la base de datos de la manera más adecuada. Entre ellos los botones que nos podemos encontrar (recordar que solo son bocetos):

- **Pigmentos**: en esta sección encontraremos una lista con todos los pigmentos que tenemos en la base de datos con algunas características en miniatura.
- **Consulta Simple**: nos permitirá realizar las principales consultas, que son por color y por el elemento químico principal del pigmento en cuestión.
- **Consulta Avanzada**: nos permitirá realizar búsquedas más avanzadas, como por las coordenadas tricrómicas de un pigmento o por determinadas características de las gráficas de los mismos.
- **Gráficas**: mostrará al usuario una lista de las diferentes gráficas que se disponen de cada pigmento.
- **Añadir Pigmento**: el usuario podrá añadir un nuevo pigmento a la base de datos, en el

caso de que se disponga de los datos suficientes para hacerlo.

Podemos ver la vista del menú principal en la **Figura 21**

Figura 21: Menú principal de la aplicación

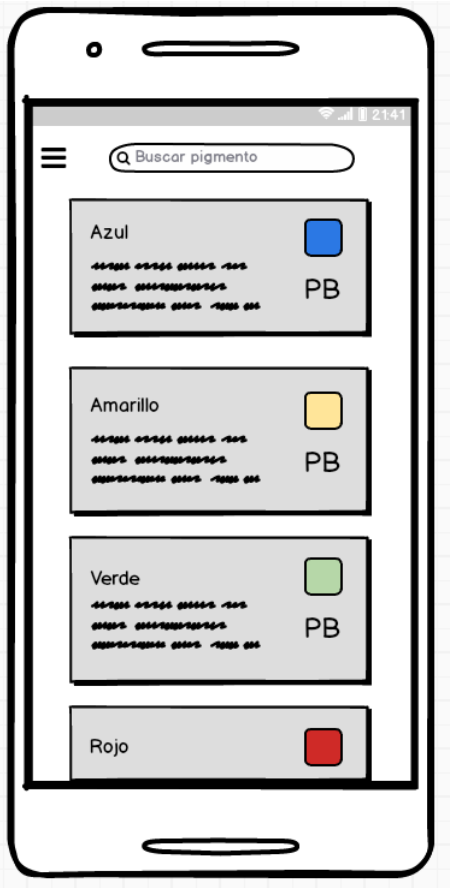

Figura 22: Lista principal con todos los pigmentos del sistema

Vista rápida de pigmentos

Si el usuario pulsa sobre el botón de pigmentos, entonces lo que se va a encontrar es una sencilla lista con todos los pigmentos que se encuentran disponibles en el sistema. La entrada de cada pigmento constará del nombre del mismo, un color aproximado, para que se le haga mas visual al usuario, junto con una breve descripción y en miniatura el elemento químico principal que compone dicho pigmento. Podemos observar dicho menú en la **Figura 22**. En esta vista el usuario puede buscar por nombre un determinado pigmento, puede conseguir esto introduciendo el nombre del mismo en la barra superior que tenemos en el menú.

Información de los pigmentos

Una vez que en la pantalla que hemos presentado anteriormente, el usuario pulsa sobre uno de los pigmentos, el sistema muestra toda la información detalla del mismo según la **Figura 23**

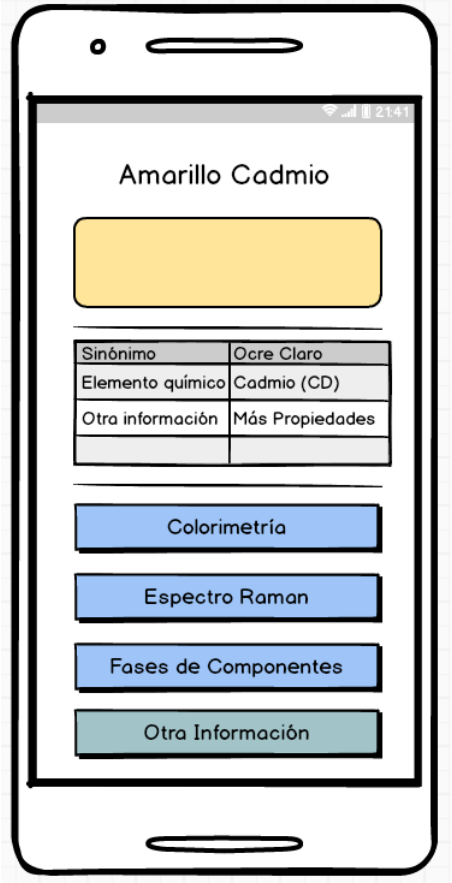

Figura 23: Información de los pigmentos

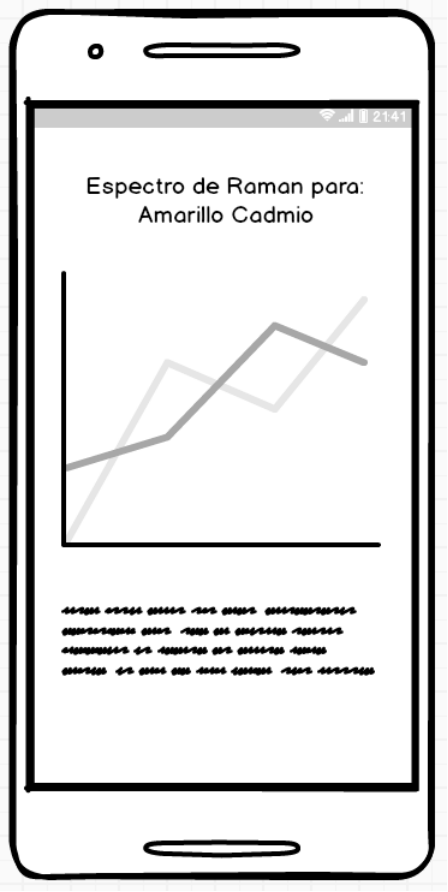

Figura 24: Información gráficas de pigmentos

Consulta Simple - Opción 1

Volviendo al menú principal, si el usuario pulsa sobre el botón de consulta simple entonces se le muestran las diferentes opciones, que en este caso solo son 3, una es la búsqueda por color, otra la búsqueda por un elemento químico y la tercera es la opción que tendríamos en el menú principal de verlos todos. Además una vez que el usuario pulsa sobre una de las dos opciones, el sistema le muestra los diferentes formularios para ejecutar la búsqueda.

En esta sección tenemos que presentar dos bocetos diferentes, que son los que se habían pensado en el principio, y luego se darán más detalles para decir con cual nos hemos decantado.

Podemos ver los primeros bocetos en las Figuras siguientes:

Figura 25: Menú para hacer una consulta simple

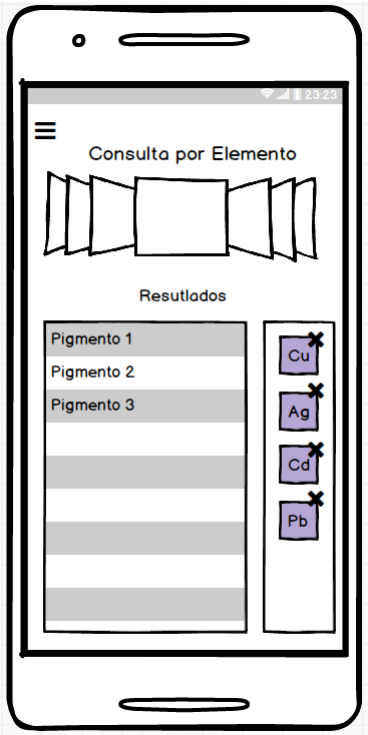

Figura 26: Menú de selección de colores

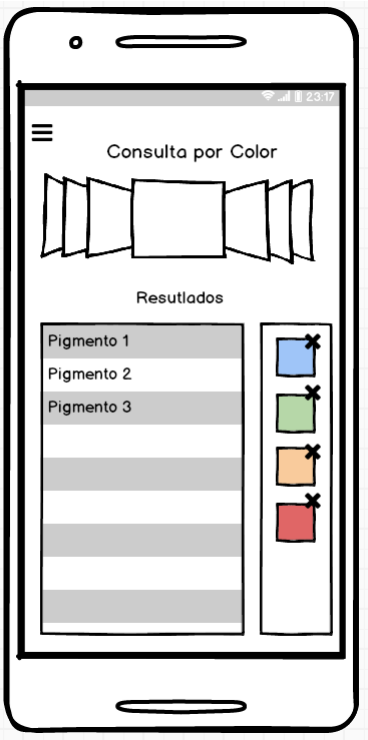

Figura 27: Menú de selección de elementos

En la **Figura 25** podemos ver el menú principal que hemos descrito anteriormente, donde el usuario elije el tipo de búsqueda que quiere realizar. Mientras que en la **Figura 26** podemos ver la selección de colores y en la **Figura 27** podemos ver como sería la pantalla de selección de elementos.

La ventaja que tienen estas interfaces es que puedes un solo diseño de la vista de la aplicación para dos funcionalidades diferentes. Además es más dinámica ya que evita tener que pulsar tantos botones como las opciones que mostraremos a continuación. El usuario simplemente desliza en busca de los colores o elementos que quiere, va pulsando sobre ellos, dichos elementos se van añadiendo a la lista vertical de la derecha, y la lista de resultados se va actualizando automáticamente en función de los filtros introducidos.

Consulta Simple - Opción 2

Sin embargo en las **Figuras 28** y **Figuras 29** podemos ver como era otra de las ideas iniciales para estas consultas.

Figura 28: Menú de selección de colores

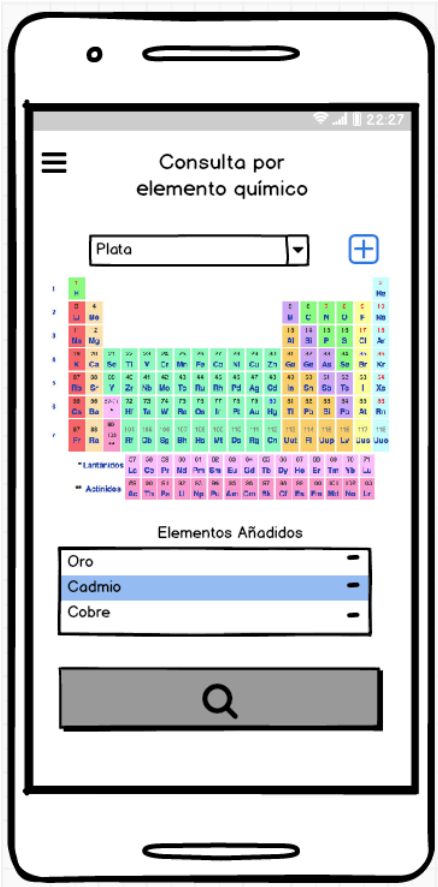

Figura 29: Menú de selección de elementos

Las desventajas que podemos encontrar al diseñar esta interfaz son varias. Primero es menos dinámica que la opción anterior ya que el usuario tiene que hacer más pulsaciones en la pantalla para obtener el mismo resultado, además de que tiene más pantallas intermedias, ya que se tienen que mostrar los resultados en una vista diferente. Otra de las desventajas desde el punto de vista de desarrollo es que requieren funcionalidades diferentes, en una tendríamos que añadir un selector de color e introducir cosas más específicas de este elemento mientras que la otra opción tendríamos que ingeniarnos un selector de elementos químicos, lo cual puede ser complicado. La opción anterior unifica estas opciones, ya que saca los colores y los elementos de una base de datos y solo tiene que cargar la información para que el usuario la elija como filtro de la búsqueda.

Recordamos que las figuras anteriormente presentadas son solo bocetos de la aplicación. La idea es que la aplicación final quede de manera aproximada a como se ha mostrado, pero por problemas de desarrollo, tiempo u otras causas es posible que los resultados finales se vean afectados o no sean completamente fieles a los presentados en las figuras anteriores.

Reportar Bug

Esta pantalla no tiene mucha importancia a nivel de funcionalidad, pero si que la considero bastante importante a nivel de peticiones de cambio futuras y cuyo objetivo es mejorar la aplicación el tiempo.

Los reportes de fallos o bugs de la aplicación, que sea reportados por los usuarios serán almacenados en un servidor central que gestionará una base de datos de fallos y bugs. Esta información se mandará de manera completamente anónima por la red, y no se almacenará ningún posible dato del usuario.

La pantalla que tengo pensada para implementar esta funcionalidad la podemos observar en la **Figura 30**

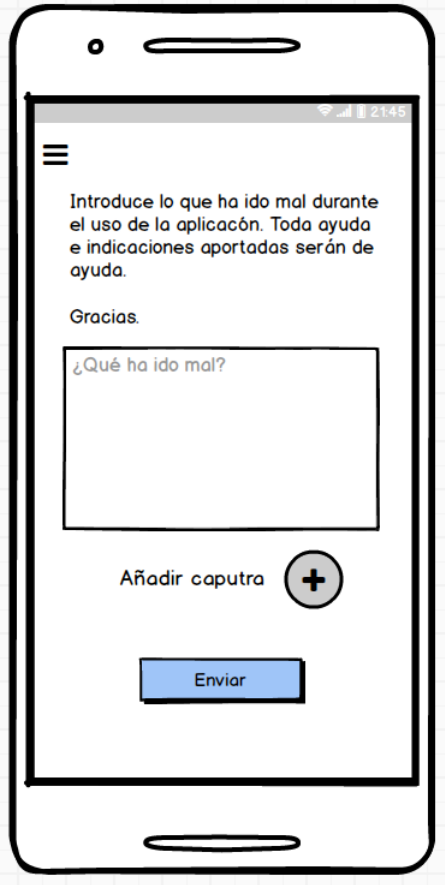

Figura 30: Menú de reporte de fallos

4.6. Sistema de Integración continua

El sistema de integración continua ha sido ligeramente descrito en los capítulos anteriores (**Sección 2.2.4**) cuando hablábamos de la tecnología que íbamos a utilizar, que en este caso es

Jenkins.

Cabe destacar que los sistemas de integración continua y entrega continua son más relevantes en equipos de desarrollo grandes con muchos servicios diferentes. Por ejemplo si el equipo de desarrollo hace un despliegue de una nueva versión de los servicios en un entorno determinado, entonces se pasan una serie de pruebas automáticas para ver que nadie ha roto nada, etc. Sin embargo ya que la magnitud de nuestro proyecto es pequeña, esta parte tampoco tiene demasiada importancia, aunque intentaré reflejarla tal cual lo hacen las empresas en la actualidad.

Si nos basamos en la teoría fundamental de la integración continua y del proceso de pruebas, las pruebas del sistema deberían de empezar en paralelo con el desarrollo de la aplicación, mientras que las pruebas unitarias de los diferentes métodos se diseñan antes de la implementación de dichos métodos [27].

En este caso lo que voy a hacer es desarrollar la interfaces más principales de la aplicación. Una vez desarrolladas esas interfaces se diseñarán los casos de prueba más simples para dichas pantallas, los cuales se introducirán en Jenkins y desde entonces se pasarán de manera automática cada vez que se haga un push en el sistema de control de versiones.

Cuando tengamos que desarrollar algún algoritmo o método, la idea es que se desarrollen algunas pruebas unitarias mínimas para probar la funcionalidad de dicha parte del método. Estas pruebas también se añadirán en trabajos diferentes a la parte de Jenkins para que se pasen de manera automática.

5. Diseño de la base de datos

En esta sección vamos a mostrar como se ha llevado a cabo el diseño de la base de datos y los diferentes procedimientos utilizados.

Tenemos que destacar que contamos con una implementación de la base de datos de pigmentos. El problema es que dicha base de datos esta implementada en AccessDB, que es un sistema gestor de bases de datos propietario de Microsoft. Existen herramientas que permiten la integración de dichos tipos de bases de datos en otros gestores como MySQL o SQLite.

En este caso la base de datos se va a diseñar desde 0 para una mejor optimización de los recursos, además se utilizará como sistema gestor de bases de datos SQLite, ya que tiene una integración completa de serie con los terminales Android.

5.1. Suposiciones y convenciones

En este apartado se van a aclarar ciertas cosas acerca de los identificadores de las relaciones, formatos que hemos decidido usar y algunas otras convenciones que me parece apropiado comentar en relación al diseño de la base de datos.

- Los identificadores tanto de los pigmentos, que actúan como clave primaria de la mayoría de las relaciones quedan representados por el descriptor breve. En este caso el descriptor es un código numérico que se ira generando de manera automática en la base de datos. De esta forma nos aseguramos de que cada uno de los pigmentos y de los datos relacionados están representados de manera unívoca por un número identificativo.

- Cabe destacar que en el diagrama de entidad relación el formato que se ha seguido para representar las cardinalidades de las relaciones queda representado por el formato típico de UML, y no el que dicta el diseño propio de las bases de datos convencionales.

5.2. Posibles consultas contra la BBDD

Vamos a analizar de manera breve las diferentes consultas que podemos realizar contra la base de datos que tenemos que diseñar. En nuestro caso, las consultas quedan determinadas por las diferentes interfaces que vamos a tener en nuestra aplicación. En este caso la principal información que vamos a tener que extraer de la base de datos es la siguiente:

- Información acerca de cada uno de los pigmentos: en nuestra aplicación tenemos una vista en la que se muestra una lista con todos los pigmentos junto con la información más básica de cada uno de ellos, como puede ser el nombre, el color y el elemento químico principal por el que están compuestos. Esta información la podemos extraer directamente de la relación Pigmentos, donde tenemos toda la información básica de todos ellos.

- Coordenadas colorimétricas: cada uno de los pigmentos consta de una serie de coordenadas que determinan el color y la luminosidad del mismo. Estas coordenadas serán almacenadas en una relación diferente porque la considero una característica un poco más avanzada que por ejemplo el nombre del pigmento. Cada uno de los pigmentos tendrá una relación directa con sus coordenadas colirimétricas. Esta relación será posible gracias al identificador único del pigmento que será utilizado como clave primaria en todas las relaciones.

- Difractograma de rayos X: cada uno de los pigmentos tiene información relacionada con su generación del difractograma de rayos X característico. Por ello esta información se ha almacenado en una tabla diferente y el método de acceso será equivalente al mencionado en la relación anterior. Tenemos que ser capaces de extraer una gráfica para poder mostrar el espectro de rayos X de cada uno de los elementos que tenemos en la base de datos.

- Infrarojos: otra de las características representativa de cada uno de los pigmentos es su espectro de infrarojos, básicamente va a estar compuesto por dos valores a partir de los cuales se podrán generar una serie de gráficas. Tenemos que ser capaces de extraer las gráficas correspondientes al espectro de infrarojos para cada uno de los pigmentos.

- Espectro de Raman: cada uno de los pigmentos tiene asociado un espectro de Raman característico. Este espectro se puede generar gracias a dos valores, ya que es una gráfica simple, en la que luego tenemos que dar los puntos más representativos. Tenemos que ser capaces de extraer los 3 picos de Raman más importantes junto con las gráficas correspondientes.

5.3. Modelado conceptual

5.3.1. Entidades y atributos

Las diferentes entidades junto con sus atributos son las siguientes:

- **Pigmento**: tenemos que almacenar de una manera segura y persistente la información relacionada con cada uno de los pigmentos. Para ello simplemente vamos a exportar dichos datos de las bases de datos ya existentes y creadas en AccessDB. La información que tenemos que almacenar de cada uno de los pigmentos es la siguiente:

  - *Nombre*: nombre con el que se identifica a cada uno de los pigmentos dentro de la base de datos. Dicho nombre es único para cada uno de los pigmentos.

  - *Id Color*: id representando el color principal de cada uno de los pigmentos

  - *Notas*: los pigmentos pueden o no tener notas adjuntas, de hecho se permite a los usuarios añadir notas sobre dichos pigmentos. Es por ello que se permiten almacenar notas separadas de cada uno de los elementos de la base de datos.

  - *Fórmula*: los pigmentos al estar compuestos por elementos naturales, tienen fórmulas

químicas determinadas. Es un dato importante si queremos saber su composición o como van a reaccionar cuando están en presencia de cierto elementos químicos, también la mostraremos.

- *Sinónimos*: los pigmentos pueden tener sinónimos a la hora de ser llamados. Es por esto que cada uno de ellos tendrá una relación con sus sinónimos.

- *Elemento químico*: cada uno de los pigmentos tiene un elemento químico predominante en su composición química. Tenemos que almacenarlo para mostrarlo cuando proporcionemos la información relacionada con el pigmento.

- **Colorimetría**:

  - L: primer parámetro dentro del espacio de color CIELAB, que nos indica la luminosidad, L = 0 es equivalente a negro, y L = 100 equivale a blanco.

  - a: segundo parámetro dentro del espacio de color CIELAB, que nos indica su posición entre el rojo y el verde. Valores negativos indican verde mientras que valores positivos indican rojo.

  - b: tercer parámetro dentro del espacio de color CIELAB, que nos indica su posición entre el amarillo y el azul. Valores negativos indican azul mientras que valores positivos indican amarillo.

  - X: primera de las coordenadas del espacio de color CIE XYZ, valor triestímulo calculado a partir de los datos Y - y - x.

  - Y: segunda de las coordenadas del espacio de color CIE XYZ, valor triestímulo calculado a partir de los datos X - y - x.

  - Z: tercera de las coordenadas del espacio de color CIE XYZ, valor triestímulo calculado a partir de los datos Y - y - x - y.

- **Difractograma de rayos X**: pares de valores reales obtenidos a partir de técnicas experimentales concretas.

- **Espectro de Raman**: pares de valores reales obtenidos a partir de técnicas experimentales concretas.

- **FTIR-ATR**: pares de valores reales obtenidos a partir de técnicas experimentales concretas.

- **Notas**: el usuario tiene que ser capaz de escribir notas rápidas para sí mismo acerca de cada uno de los pigmentos. Es por ello que una de las opciones de la aplicación es almacenar notas de uno de los pigmentos seleccionados. Dichas notas podrán ser recuperadas con posterioridad por el usuario para ser editadas o borradas.

  - *Título*: cada una de las notas que almacenamos de un pigmento consta de un título,

que no es único y va a tener una longitud limitada a 100 caracteres. Tiene que ser algo corto y determinado. Ya que lo primero que ser verá cuando mostremos la vista de las notas es el título y estarán ordenadas por la fecha de creación, por lo que la última nota creada es la que se mostrará por completo.

- *Valor*: el valor es la nota en sí. Lo que el usuario introduzca en el formulario de escritura.

- *Fecha*: la fecha no será necesario mostrarla al usuario, pero si que tiene sentido almacenarla en la base da datos para tener un histórico de creación de las mismas y luego poder recuperarlas en diferente orden por la aplicación.

- **Sinónimos**: cada uno de los pigmentos puede tener nombres diferentes con los que ha sido referenciado a lo largo del tiempo. Es por esto que tendremos un lista para cada uno de los pigmentos con todos los sinónimos que tenga.

  - *Valor*: como cada uno de los sinónimos puede tener 0 o más sinónimos, he decidido hacer una relación separada para cada uno de los pigmentos. Sino otra forma es hacer un atributo sinónimo dentro de la entidad Pigmento conteniendo una lista separada por comas con todos los sinónimos, pero me parece una mejor opción hacer una relación separada.

Antes de mostrar el diagrama de entidad relación vamos a mostrar una pequeña tabla con las diferentes cardinalidades y entidades relacionadas. Podemos encontrar dicho gráfico en la **Figura 31**

Entidad	Cardinalidad	Entidad	Cardinalidad
Pigmento	1-N	*Nota*	0-N
Pigmento	1-N	*Sinónimo*	0-N
Pigmento	1-N	*Infrarrojos*	1
Pigmento	1-N	*Raman*	1
Pigmento	1-N	*RayosX*	1
Pigmento	1-N	*Colorimetría*	1

Figura 31: Entidades y cardinalidades del modelo

5.3.2. Diagrama de Entidad Relación

Una vez que hemos presentado tanto las entidades, como las cardinalidades y como están relacionadas las mismas, podemos mostrar como se relacionan las unas con las otras de una manera gráfica, que es presentado el diagrama de entidad relación de la base de datos. En este caso se ha decidido usar el formato de UML y no sigue las reglas estrictas del modelo de Entidad Relación, pero me parece que es más visual, la herramienta y el formato es más universal y entendible que el propio para expresar este tipo de contenidos. Podemos encontrar esto en la **Figura 32**

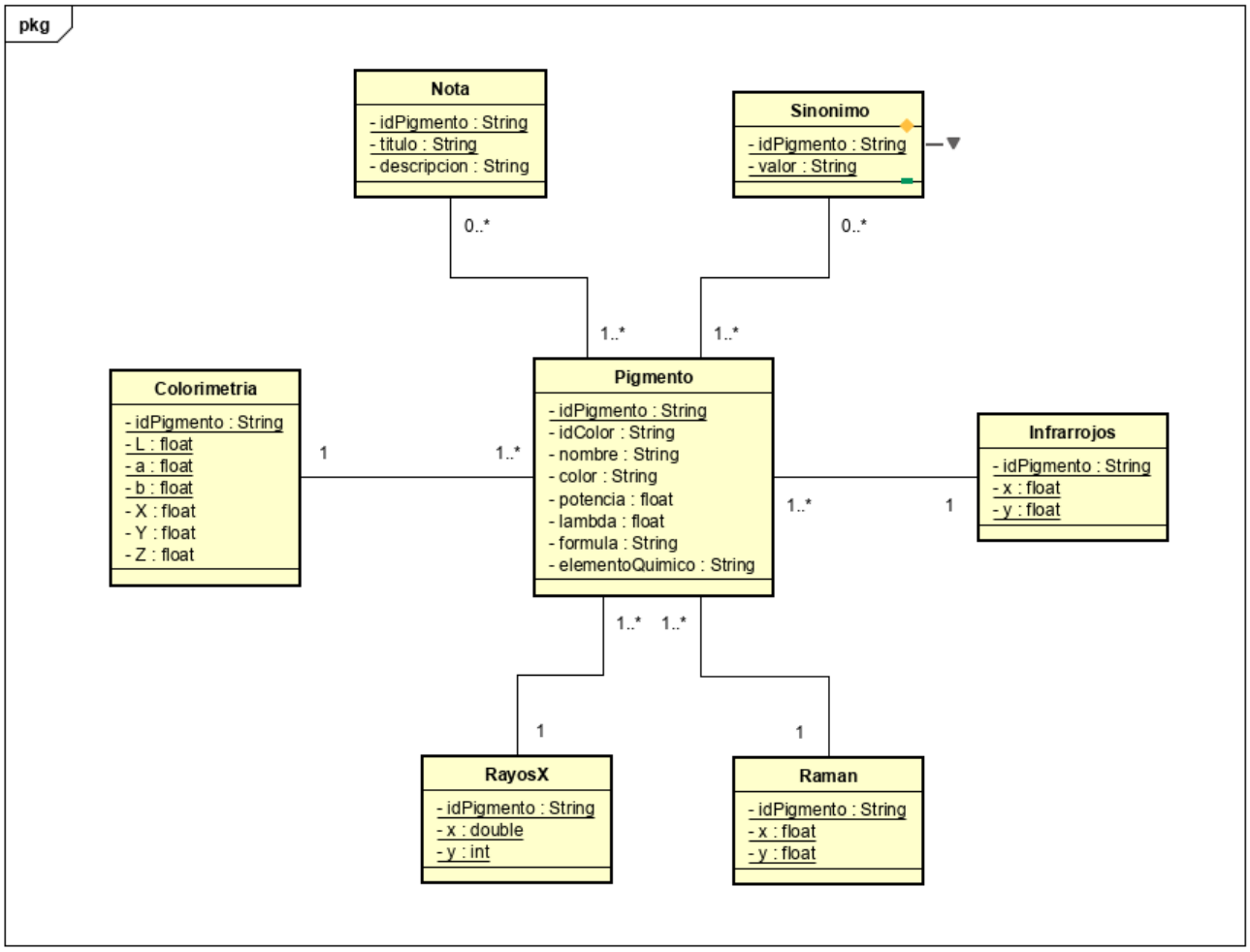

Figura 32: Diagrama de Entidad-Relación de la base de datos

Podemos observar que la clave primaria de la relación pigmento es el id de dicho pigmento, que simplemente es un identificador numérico y unívoco para cada una de las entradas de la relación. Además para todo el resto de relaciones hemos añadido también el id de dicho pigmento como atributo para poderlo usar como clave foránea y poder relacionar las tablas de una manera sencilla.

Las claves primarias del resto de relaciones las podemos observar en el diagrama ya que están subrayados los atributos que pertenecen a ella.

Una vez que tenemos el diseño de la base de datos podemos pasar a explicar la implementación de la misma, en lineas generales.

5.4. Implementación de la base de datos

Una vez que tenemos la base de datos diseñada tenemos que implementarla dentro de nuestra aplicación. Para ello lo primero que tenemos que hacer es crear una clase auxiliar que nos permita

la creación y en el futuro la conexión entre la propia aplicación y los datos almacenados en la base de datos.

Para implementar la base de datos he optado por hacerlo de una manera más sencilla que la aprendida en la asignatura de SMOV. La base en realidad es la misma, pero la forma es la misma, bien es sabido que dentro de la informática podemos obtener los mismos resultados pero con algoritmos diferentes. En este caso vamos a realizar una mezcla entre uno de los patrones de acceso comentados al principio, DAO y los POJO.

Como hemos dicho en los capítulos anteriores, cada uno de los elementos de la base de datos, que en este caso son pigmentos, va a estar representado por un objeto típico de Java. Esta forma de representar los objetos para el almacenamiento de datos se denomina POJO (**P**lain **O**ld **J**ava **O**bjects).

Además cada vez que recuperemos los elementos de la base de datos lo hacemos devolviendo una lista de objetos del tipo adecuado. Por ejemplo para devolver todos y cada uno de los pigmentos que hay en la base de datos lo que hacemos es mandar la consulta al SGBD de SQLite y lo que nos devuelve es una lista de POJOs del tipo deseado.

De esta manera podemos acceder de una manera sencilla a todos los atributos de los datos que acabamos de recuperar.

5.4.1. ¿Que son los POJO?

Dicho término [19] fue acuñado por Martin Fowler, Rebecca Parsons y Josh MacKenzie en Septiembre del 2000. Se usa para referirse a un objeto Java que no tiene limitaciones de por requerir librerías ni clases externas. Son objetos simples y básicos de java, con atributos, funciones para recuperar los valores y para cambiarlos en caso necesario y nada más. Este tipo de objetos son muy utilizados cuando tenemos que convertir objetos entre diferentes lenguajes o cuando tenemos que serializarlos para enviarlos por la red.

Idealmente y según dice la definición de los mismos, un objeto Java es POJO si cumple estos 3 requisitos:

- No extiende de ninguna clase.

```
1 public class Foo extends javax.servlet.http.HttpServlet { ...
```

- No implementa ninguna clase.

```
1 public class Bar implements javax.ejb.EntityBean { ...
```

- No contiene anotaciones.

```
1 @javax.persistence.Entity public class Baz { ...
```

Los POJO fueron inicialmente utilizados en J2EE para representar los JavaBeans, que son objetos java muy simples y que ofrecen servicios por la red. Se siguen utilizando aunque han cambiado un poco las formas de hacerlo. Ahora los POJO también son ampliamente utilizados para representar objetos JSON recibidos por parte de microservicios web y ser utilizados en un servidor. Este es uno de los muchos ejemplos de utilización real a día de hoy de los POJO.

5.4.2. Ejemplo

Vamos a explicar con un sencillo ejemplo lo que son los POJO y como se hace la consulta de los mismos para luego mostrar la información dentro de la aplicación.

En este caso uno de los atributos que nos interesa son las diferentes notas que tiene cada uno de los pigmentos, para ello habrá una tabla con todas las notas de todos los pigmentos. Como hemos enseñado en el modelado conceptual, de cada una de las diferentes notas almacenamos y guardamos los siguientes datos:

- IdPigmento: para poder relacionar la nota con el pigmento, este dato es interno para la gestión de los datos.
- Título: breve descripción.
- Descripción: contenido completo de la nota.
- Fecha: fecha de creación de la misma.

La descripción anterior la podemos encontrar representada por un POJO en el siguiente fragmento de cñodigo Java:

```
1  package com.dbpigmentationapp.dataModel;
2
3  public class Nota {
4
5      private String idPigmento;
6      private String titulo;
7      private String descripcion;
8
9      // Constructor por defecto
10     public Nota () {
11         this.idPigmento = "Id Pigmento";
12         this.titulo = "T tulo";
13         this.descripcion = "Breve descripci n";
14     }
15
16     // Constructor con parametros
17     public Nota (String idPigmento,String titulo,String descripcion) {
18         this.idPigmento = idPigmento;
19         this.titulo = titulo;
20         this.descripcion = descripcion;
21     }
22
23     // FUNCIONES DE ACCESO A DATOS
24     public String getTitulo() {
25         return titulo;
26     }
27
28     public void setTitulo(String titulo) {
29         this.titulo = titulo;
30     }
31
32     public String getDescripcion() {
33         return descripcion;
34     }
35
36     public void setDescripcion(String descripcion) {
37         this.descripcion = descripcion;
38     }
39
40     public String getIdPigmento() {
41         return idPigmento;
42     }
43
44     public void setIdPigmento(String idPigmento) {
45         this.idPigmento = idPigmento;
46     }
47 }
```

Una vez que hemos diseñado todos los POJOS que va a contener nuestra aplicación, que simplemente es uno por cada entidad del diagrama de entidad relación, además con los mismos atributos que se muestrán en el esquema. Ahora podemos implementar todas las operaciones de acceso a esos datos. Las operaciones para cada uno de las diferentes entidades que tenemos en nuestro sistema son:

- **Pigmento**:
  - Añadir pigmento.
  - Obtener todos los pigmentos para mostrarlos en la lista principal.
  - Obtener los pigmentos que coincidan con un nombre.
  - Obtener los pigmentos que coincidan con un color.
  - Obtener los pigmentos que coincidan con un elemento químico.
- **Notas**:
  - Añadir notas.
  - Obtener todas las notas de un pigmento dado su nombre (en la consulta se usará el id del pigmento).
- **Sinónimos**:
  - Añadir notas.
  - Obtener todos los sinónimos de un pigmento dado su nombre (en la consulta se usará el id del pigmento).
- **Infrarrojos**:
  - Añadir infrarrojos.
  - Obtener todos los datos de los infrarrojos dado un pigmento, para luego poder representarlos en una gráfica de manera adecuada.
  - Obtener todos los elementos dadas una coordenadas de infrarrojos determinadas y mostrar los elementos que corresponden a esos valores.
- **Rayos X**:
  - Añadir rayos X.
  - Obtener todos los datos de los rayos X dado un pigmento, para luego poder represen-

tarlos en una gráfica de manera adecuada.

- Obtener todos los elementos dadas una coordenadas de infrarrojos determinadas y mostrar los elementos que corresponden a esos valores.

- **Raman**:
  - Añadir coordenadas de Raman.
  - Obtener todos los datos del espectro de Raman dado un pigmento, para luego poder representarlos en una gráfica de manera adecuada.
  - Obtener todos los elementos dadas una coordenadas de del espectro de Raman determinadas y mostrar los elementos que corresponden a esos valores.

- **Colorimetria**:
  - Añadir coordenadas de colores.
  - Obtener todos los datos colorimétricos dado un pigmento, para luego poder representarlos en una gráfica de manera adecuada.
  - Obtener todos los elementos dadas una coordenadas de de colores determinadas y mostrar los elementos que corresponden a esos valores.

5.5. Otras consideraciones

Algunas otras decisiones de diseño que hemos tenido que tener en cuenta, y que afectan tanto al rendimiento de la aplicación como al diseño de la base de datos, son las siguientes:

5.5.1. Diseño antiguo VS diseño actual

En el diseño antiguo de la base de datos (**Figura 33**), de manera resumida se tiene una tabla con una entrada para cada pigmento y esta entrada tiene un atributo que referencia a un fichero de datos. Por ejemplo si queremos obtener el difractograma de rayos X del pigmento número 3, nos va a referenciar a un fichero de texto que contiene aproximadamente unas 4000 líneas con 2 valores en cada línea. Es decir que en total vamos a tener unos 50 ficheros cada uno de 4000 líneas. Según el gestor de almacenamiento de Windows estos ficheros tienen un tamaño medio de unos 30 Kb. En total tenemos unos 150 ficheros lo que hace que el total de los ficheros si los tuviéramos que almacenar dentro de la base de datos para su posterior procesado, además de tener aparte la instancia de la base de datos creada supondría un aumento de unos 5Mb en la aplicación.

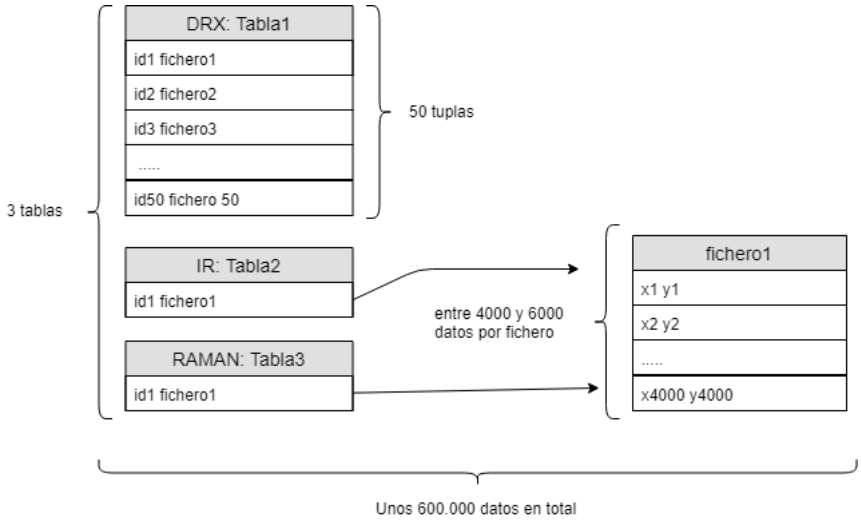

Figura 33: Diseño propuesto en Access de la Base de Datos

En el diseño que yo he propuesto (**Figura 34**) lo que tenemos son 3 tablas, cada una de 200.000 entradas (50 x 4000 en el peor de los casos) donde tenemos disponibles todos y cada uno de los datos. Además una de las ventajas de esta forma es que la aplicación no tiene que hacer tantas operaciones de entrada salida para procesar todos los datos al realizar las consultas. Con el diseño propuesto solo se procesan una única vez para poder cargar y generar la base de datos.

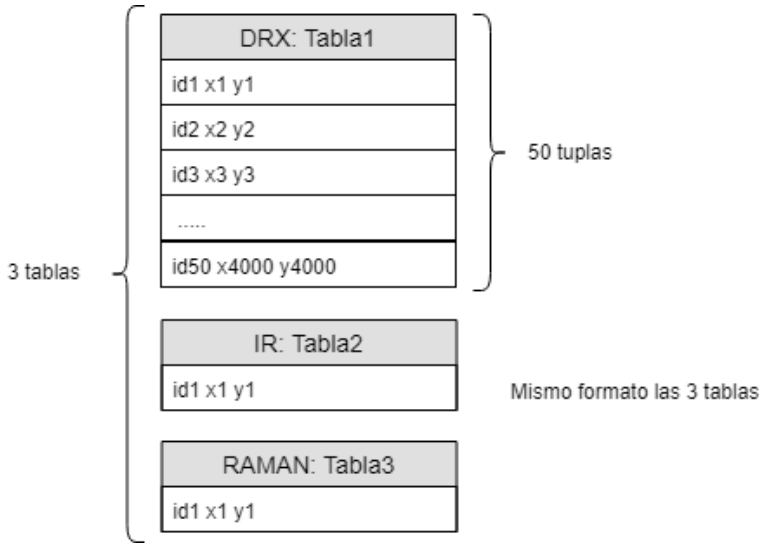

Figura 34: Diseño propio de la base de datos

5.5.2. Código frente a eficacia

Otro de los problemas es la eficacia y el diseño frente a la limpieza y entendibilidad del código. Podemos implementar la solución de dos maneras diferentes.

La primera es creando una única base de datos con todas las tablas y todas las operaciones necesarias. El problema es que esa base de datos y las operaciones se tienen que crear en una única clase que se encarga de ello, esta clase va a ser relativamente grande y uno de los principios de diseño que por ejemplo podemos encontrar en el Libro Clean Code [20] es que las clases tienen que ser pequeñas, autocontenidas y hacer una única cosa. Sin embargo nuestra clase va a crear una base de datos, inicializar 4 tablas y ofrecer todos los métodos de acceso y modificación de todas las tablas.

De esta forma el código se ve afectado pero sin embargo el diseño de la base de datos y la eficacia de las consultas no se ve afectada.

La otra manera es crear una clase para cada una de las tablas, pero de esta manera tendríamos una base de datos por tabla, lo que no concuerda con el diseño propuesto de la base de datos y además tendríamos que procesar las consultas via software usando la JVM lo cual puede traducirse en un decremento de la eficiencia de la aplicación.

La solución por la que se ha optado es la primera.

5.5.3. Scripts necesarios

Por como he decidido hacer el diseño de la base de datos, ha sido necesario adaptar de una cierta manera los datos para luego ser procesados de manera automática por la aplicación e introducirlos en la base de datos. Ha sido necesaria la creación de una serie de scripts, para evitar un trabajo manual considerable. Los programas necesarios son los siguientes:

- **Generador de notas**: en nuestra base de datos estamos almacenando una breve descripción de los pigmentos. Esta descripción en la base de datos original esta como una tabla separada con 2 columnas. La primera columna referencia al pigmento mientras que la segunda referencia al archivo donde se puede encontrar la descripción. A mi me parece ineficiente la idea de tener los datos separados, por eso en mi base de datos la descripción esta dentro de la propia base de datos. Pero para poder generarla hay que tener los ficheros en un formato correspondiente, que en este caso es CSV. Por lo que para ello he tenido que generar el siguiente script, que recorre todos los ficheros y pone las descripciones de los pigmentos en una única línea, que luego la base de datos tomará como string y almacenará.

```
import os

files = {}

for filename in os.listdir("./generador"):
    file = open("./generador/" + filename, "r")
    files[filename] = file.read()

for filename, text in files.items():
    print(';' + text.replace('\r', "").replace('\n', ""))
```

- **Generador de datos**: en nuestra base de datos, como hemos comentado anteriormente, tenemos una tabla para almacenar los datos de IR, otra para los de RX y otra para Raman. Estos datos se han obtenido ejecutando scripts en python donde podemos juntar en un único fichero en formato CSV todos los datos. Por ejemplo para obtener los datos de la tabla de DRX y luego e introducirlos de una manera sencilla en nuestra base de datos, hemos generado un único fichero que tiene la siguiente estructura: idPigmento;valorX;valorY. Esta estructura se ha seguido con todos los ficheros y se ha conseguid gracias al siguiente script.

```
import os
import glob

ids = []
idFiles = open("./id.txt", "r")
for y in idFiles:
    ids.append(y.replace("\n", "").replace("\r", "."))

path = './generador/*.txt'
files = glob.glob(path)
i = 0
for file in files:
    f = open(file, 'r')
    for x in f:
        if ("#" not in x):
            print(ids[i] + ";" + x.replace("\r", "").replace("\n", "")
                .replace("\t", ";"))
    i=i+1
```

5.6. Tratamiento de las BBDD en Android

Como es evidente la mayoría de las aplicaciones tiene que almacenar datos entre sus ejecuciones, o para compartirlas entre el resto de aplicaciones. Android nos ofrece una serie de formas de almacenar los datos [21], que son las siguientes:

- **Preferencias compartidas**: almacenamiento de datos primitivos en pares de datos clave valor.
- **Almacenamiento interno**: almacenamiento de datos privados en la memoria del dispositivo.
- **Almacenamiento externo**: almacenamiento de datos públicos en el almacenamiento externo compartido.
- **Bases de datos SQLite**: almacenamiento de datos estructurados en una base de datos privada.
- **Conexión de red**: almacenamiento de datos en la Web mediante tu propio servidor de red.

En nuestro caso vamos a realizar una mezcla de tipos de almacenamiento. Por una parte nuestra aplicación contendrá los ficheros de datos necesarios para generar la base de datos, pero a su vez la aplicación interactuara con esos datos mediante una base de datos SQLite 3 que se generará la primera vez que se inicie la aplicación en el dispositivo móvil.

Nuestra aplicación va a disponer de la información necesaria y suficiente como para crear la base de datos completa y almacenarla de manera inaccesible para el usuario en el almacenamiento interno privado del dispositivo.

La primera ejecución de la aplicación puede ser un poco más lenta de lo normal, precisamente porque se está creando la base de datos. Una de las mejoras futuras podría ser que la primera ejecución mostrase un mensaje de configuración del tipo: "Se está configurando la base de datos". De momento no se va a implementar.

En las posteriores ejecuciones la aplicación va a volver a tratar de introducir todos esos datos en la base de datos, pero es el sistema operativo el que ve que ya existen y no los introduce, cancelando así la operación y no gastando tiempo de ejecución. Otra de las comprobaciones que se podrían hacer en un futuro es comprobar si la base de dato ya existe, de esa manera se ahorraría aún más tiempo. Ya que las bases de datos de las aplicaciones se almacenan en una ruta determinada: *$ /data/data/com.dbpigmentationapp/databases/pigments.db* La parte de çom.dbpigmentationapp"tiene que ser reemplazada con el nombre del paquete con el que hemos generado nuestra aplicación, que es el mismo que nos muestra AndroidStudio mientras la estamos desarrollando.

6. Guía de Instalación y Manual de Usuario

6.1. Guía de Instalación

Vamos a presentar una pequeña guía de instalación. Lo que se ha entregado de la aplicación han sido dos cosas (a nivel de instalación). Tenemos dos formas de instalar la aplicación en un dispositivo Android:

- **Fichero APK**: uno de los fichero mas importantes generados de manera automática por el programa Android Studio es el debug.apk. Que lo podemos encontrar como fichero de salida dentro de nuestro proyecto. Simplemente tenemos que arrastrar dicho fichero al móvil o pasarlo de alguna manera, intentar abrirlo y otorgando los permisos necesarios, podemos instalar la aplicación considerada DEBUG. Es decir, esta aplicación no se puede usar para fines comerciales, ya que no esta registrada en la tienda oficial de Google.

- **Proyecto Android Studio**: otra de las formas posibles de instalar la aplicación en un dispositivo real es importar el proyecto en Android Studio. Simplemente tenemos que hacer clic en: Archivo ¿Abrir y elegir la carpeta donde se encuentra el proyecto que he adjuntado al entregar este documento. Una vez hecho esto, la compilación y construcción del proyecto no debería de dar ningún fallo. En la barra de herramientas superior podemos ver un símbolo de Play. Antes de pulsarlo tenemos que tener el dispositivo objetivo conectado al ordenador con los permisos de desarrollador activados. Una vez dado al play y seleccionado el dispositivo objetivo, la aplicación se abrirá de manera automática en el móvil conectado.

6.2. Manual de Usuario

Para abrir la aplicación simplemente tenemos que pulsar en el icono correspondiente dentro del menú de aplicaciones del móvil en el que la tengamos instalada.

Una vez abierto el menú podemos realizar cuatro acciones importantes:

1. **Ver todos los pigmentos**: situado en el menú principal de la aplicación. Como usuario, tienes que pulsar en el botón de color azul que pone Pigmentos. Una vez pulsado el sistema mostrará una lista descendente con la información básica y principal de todos los pigmentos de los que se dispone información. El usuario puede pulsar en cualquiera de los elementos de la lista, acto seguido se mostrara una pantalla con la información principal relacionada con ese pigmento. El usuario puede interactuar con esta pantalla. Si desliza hacia abajo verá una serie de desplegables, si se pulsa en ellos se puede ver la información de las gráficas más importantes que se presentan como información de cada uno de los pigmentos.

2. **Buscar un pigmento por color**: situado en el menú principal de la aplicación. Como usuario tienes que pulsar en el botón de color naranja que pone filtrar por color. Entonces se mostrará una ventana con un círculo cromático donde el usuario puede elegir el color

deseado junto con la luminosidad. Hecho esto, el usuario pulsa en el botón de buscar pigmentos. Entonces el sistema muestra los resultados relacionados con el pigmento que se ha seleccionado.

3. **Buscar un pigmento por elemento químico**: situado en el menú principal de la aplicación. Como usuario, tienes que pulsar en el botón de color violeta que pone buscar por elemento químico. Entonces el sistema muestra una ventana que simula una pequeña tabla periódica donde tenemos todos los elementos excepto los gases nobles, hidrógeno y las tierras raras. La tabla periódica esta dividida en diferentes trozos para que el usuario pueda visualizarla de una manera sencilla. Si el usuario pulsa en uno de los elementos de la tabla, entonces el sistema realiza la consulta en la base de datos y muestra la información.

4. **Buscar un pigmento por nombre**: situado en el menú principal de la aplicación. Como usuario, tienes que pulsa en el botón de color verde que pone filtrar por nombre. Hecho esto el sistema muestra una pantalla donde el usuario puede introducir un nombre. Una vez introducido si pulsa en el botón de buscar pigmento entonces el sistema muestra la ventana con los resultados de la búsqueda según el nombre que se ha introducido.

7. Problemas durante el desarrollo

Como es evidente todos los proyectos de ingeniería tiene dificultades y no es un proceso completamente planificado y sin problemas. En esta sección vamos a comentar algunos de los problemas más destacables que se han ido presentando conforme se desarrollaba la aplicación.

- **Base de datos**: al principio pensaba que la base de datos iba a ser reutilizable ya que lo primero en lo que pienso cuando escucho base de datos es SQL. El problema fue encontrar que la base de datos estaba implementada en Access. Que es el sistema gestor de bases de datos principal de Microsoft, orientado a un objetivo no tan técnico. Además otro de los problemas relacionados fue que el diseño de la base de datos era bastante pobre, había mucha información desperdigada en sitios diferentes y almacenada en ficheros. Esto en mi opinión no tiene sentido porque además estaba siendo soportado por un servidor entero. El problema principal en esta fase fue como juntar todos los ficheros para poder tratarlos como tablas únicas en SQLite. Lo resolví escribiendo unos pequeños scripts en python para parsear los datos al formato que a mi me convenía.

- **Rendimiento y cantidad de datos**: uno de los principales problemas que no se tuvieron en cuenta en las etapas iniciales es que los datos almacenados en la base de datos son muchos. Aproximadamente estamos hablando de unos 2 millones de datos si tenemos en cuenta todos los valores para todas las gráficas de todos los pigmentos. Uno de los primeros problemas de rendimiento fue tratar de leer todos los valores de los ficheros, introducirlos en la base de datos para que4 quedasen de manera permanente para que después la aplicación los recuperase de manera sencilla. El problema es que al intentar realizar este flujo cualquier dispositivo Android dejaba de funcionar. La solución fue leer los datos por trozos e introducirlos por partes en la base de datos. En vez de introducirlos todos a la vez, los datos se recuperan según el pigmento que se selecciona, lo cual decrementa el tiempo de ejecución y de búsquedas en aproximadamente un 1000 %.

- **Problemas de gráficas**: otro de los problemas principales relacionado con el rendimiento de los dispositivos móviles es que una de las funciones de búsqueda que se iba a implementar en un principio no ha acabado desarrollándose. Esto se debe principalmente al rendimiento, las operaciones de búsqueda en cantidades de datos tan grandes daban como resultado unos tiempos de carga demasiado elevados, por lo que con la aprobación del cliente, esta funcionalidad se ha quitado y se puede estudiar como una posible mejora de la aplicación.

- **Tiempo**: el tiempo ha sido un recursos muy preciado. Evidentemente el tiempo ha jugado un papel clave en el desarrollo de la aplicación, aunque al final no ha sido un factor decisivo para poder entregar el producto final o no en la fecha establecida.

A grandes rasgos, los problemas que se han encontrado a lo largo del desarrollo de la aplicación han sido esos. El resto han sido problemas menores, no destacables, relacionados o con carencias en el lenguaje de programación o también en la información que se estaba tratando.

La información contenida en la base de datos es información generada por investigadores y científicos especializados en dicho campo. Al no haber tratado nunca esos temas he tenido que

preguntar muchas veces al cliente para poder continuar o saber si el resultado era el esperado o no.

8. Conclusiones Finales

Y para concluir podemos resumir las siguientes conclusiones y sensaciones finales de este proyecto:

- Es un proyecto de ingeniería, que ha sido planificado y desarrollado conforme la estimación inicial. Una de las cosas que me sorprende es que por muy simple que parezca el resultado final, o el trabajo realizado y plasmado en un documento, son muchas horas de trabajo, esfuerzo y dedicación de una o varias personas. Pero la gratificación de ver un proyecto finalizado en el tiempo estimado es muy reconfortante.

- Una de las conclusiones a las que seguramente llegan todos los alumnos es que aunque te esfuerces en hacerlo lo mejor posible, la gestión del tiempo es algo muy complicado. Ahora se entiende porque hay gentes especializada únicamente en gestionar tanto equipos como recursos para que los proyectos se desarrollen en el tiempo estipulado.

- Si volvemos al campo un poco más técnico, una de las conclusiones más importantes que he aprendido es que los dispositivos Android no pueden manejar grandes cantidades de datos en el espacio de almacenamiento local. Si que es cierto que disponen de mucho almacenamiento, pero cuando se requieren búsquedas en millones y millones de elementos, entonces el rendimiento se empieza a ver bastante afectado. Hasta donde he podido comprobar búsquedas entre unos 10.000 elementos se pueden llevar a cabo de una manera sencilla y rápida si tienes el cuidado suficiente. A partir de esas cifras, el rendimiento se ve seriamente comprometido. Es por eso que la mayoría de aplicaciones son simplemente interfaces gráficas y sacan los datos de bases de datos distribuidas que evidentemente no están almacenadas en el espacio de almacenamiento local del propio dispositivo.

- Una de las conclusiones mas valiosas es la capacidad que he tenido de demostrar y juntar todos los conocimientos que se han ido adquiriendo durante el estudio del grado.

- También cabe destacar el gran conocimiento que se ha adquirido sobretodo de la programación en Android. Es cierto el lenguaje principal es Java, pero muchos de los patrones que tienes que seguir obligatoriamente para poder desarrollar una aplicación en Android son únicos del desarrollo de este tipo de software. A parte de los conocimientos propios obtenidos a nivel de programación, también de la combinación de herramientas de integración continua y testing. Aunque en esta memoria no queden lo suficientemente reflejadas (ya que no era la idea del proyecto) el proyecto ah tenido integración continua y testing automatizado.

- Mencionar también que el desarrollo de interfaces de usuario simples e intuitivas a la vez que bonitas es todo un reto. Además tienes que llegar a la mayor cantidad de personas posibles dentro del mercado objetivo, y no todas tienen los mismos gustos estéticos.

Referencias

[1] **Manifiesto Ágil**. *Manifiesto ágil con los principios de agilidad software.*
https://agilemanifesto.org/iso/es/manifesto.html

[2] **Attlasian Company**. *Información de la empresa líder en herramientas de PM.*
https://es.atlassian.com/

[3] **Kanban - Atlassian**. *Información de Kanban de Atlassian.*
https://es.atlassian.com/agile/kanban

[4] **Scrum - Atlassian**. *Información de Scrum de Atlassian.*
https://es.atlassian.com/agile/scrum

[5] **Jira**. *Información de Jira de Atlassian.*
https://es.atlassian.com/software/jira

[6] **Taiga Software**. *Software para control de proyectos ágiles.*
https://tree.taiga.io/discover

[7] **SonarQube**. *Software para el control de la calidad del software.*
https://www.sonarqube.org/features/clean-code/

[8] **SonarLint**. *Plugin agregado de SonarQube para el control de la calidad en el IDE.*
https://www.sonarlint.org/

[9] **Jenkins**. *Softwre para la integración continua y la entrega continua de software.*
https://jenkins.io/

[10] **Android Studio**. *Entorno de desarrollo integrado para programación de aplicaciones multiplataforma.*
https://developer.android.com/studio

[11] **Github**. *Software de control de versiones en la nube.*
https://github.com/

[12] **SQLite**. *SGBD ligero integrado en Android.*
https://www.sqlite.org/index.html

[13] **Overleaf**. *Sistema de composición de textos basado en LaTeX.*
https://es.overleaf.com/

[14] **Confluence**. *Sistema de documentación en la nube propietario de Atlassian.*
https://es.atlassian.com/software/confluence

[15] **Docker**. *Sistema de contenedores open source.*
https://www.docker.com/why-docker

[16] **Gradle**. *Sistema de construcción de proyectos y gestor de dependencias.*
https://gradle.org/

[17] **Gitflow - Atlassian**. *Gitflow worflow de la empresa líder en Agilidad.*
https://es.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[18] **Balsamiq Mocuk Up**. *Página principal de Balsamiq.*
https://balsamiq.com/wireframes/

[19] **POJO**. *Definición de POJOs.*
https://en.wikipedia.org/wiki/Plain_old_Java_object

[20] **Libro Clean Code**. *Libro Clean Code - Robert C. Martin.*
https://www.oreilly.com/library/view/clean-code/9780136083238/

[21] **Almacenamiento Android**. *Android Developers - Formas de almacenear contenido.*
https://developer.android.com/

guide/topics/data/data-storage#db

[22] **Android Virtual Device**. *Android Developers - Como crear un AVD.* https://developer.android.com/studio/run/managing-avds?hl=es-419

[23] **Android Software Development Kit**. *Android SDK.* https://developer.android.com/studio

[24] **Model View Controller**. *Patrón MVC.* https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[25] **Object Request Manager**. *Patrón ORM.* https://es.wikipedia.org/wiki/Object-role_modeling

[26] **Consejos de diseño de interfaces**. *Guía de diseño de Experiencia de Usuario de Adobe.* https://theblog.adobe.com/15-rules\-every-ux-designer-know/

[27] **International Software Testing Qualification Boards**. *Certificaci'on de testing internacional.* https://www.istqb.org/