Grado en Ingeniería Informática

Trabajo Final de Grado

Desarrollo de aplicación Android mediante metodologías ágiles

Autor:
Aarón Molés Tormo

Supervisor:
Sergio Aguado González
Tutor académico:
Ramón Mollineda Cárdenas

Fecha de lectura: 2 de julio de 2015
Curso académico 2014/2015

Resumen

Este documento describe el proceso de desarrollo de una aplicación móvil Android colaborativa. Partiendo de una idea, se ha realizado el análisis, diseño, implementación y pruebas utilizando una metodología iterativa e incremental. Obteniendo como resultado una aplicación Android completamente funcional.

El desarrollo de la aplicación se ha realizado durante la estancia en prácticas en la empresa *Soluciones CuatroOchenta*, empresa especializada en el desarrollo de aplicaciones móviles.

La aplicación conjuntamente con este documento conforman el Trabajo de final de grado (TFG) en el Grado en ingeniería informática de la *Universitat Jaume I.*

Palabras clave

Android, trivial, wiki, ágil, incremental

Keywords

Android, trivial, wiki, agile, incremental

Índice general

Índice de figuras

Capítulo 1

Introducción

Índice

A lo largo de este documento se detallan los diferentes aspectos del proyecto informático **Quizy App** de la empresa *Soluciones CuatroOchenta*. En concreto se realiza una descripción de la empresa, del producto a construir, los objetivos, la planificación realizada, la metodología, las tecnologías utilizadas, el proceso de construcción y desarrollo, y los resultados obtenidos.

1.1. Contexto y motivación del proyecto

1.1.1. Visión general de Soluciones CuatroOchenta

Soluciones CuatroOchenta es una empresa especializada en el desarrollo integral de aplicaciones para *smartphones*, tabletas y programación avanzada a medida. Esta empresa cuenta con más de 50 aplicaciones desarrolladas durante sus 4 años de experiencia.

Esta empresa mantiene una filosofía de crear cada proyecto con una metodología de colaboración real y una altísima implicación con el cliente y sus usuarios. Aglutinando a profesionales en el campo de la **programación**, **desarrollo**, **diseño gráfico**, **marketing** y **comunicación**.

1.1.2. Motivación

Actualmente la gran mayoría de personas disponen de un *smartphone* o de cualquier otro dispositivo con características similares como un *tablet*. Según los datos del informe *Sociedad de la Información en España 2014* [4], publicado en el mes de enero del 2015, la proporción de *smartphones* en España ha aumentado del 66 % en 2013 al 81 % en 2014. Al igual sucede con las *tablets* que han experimentado un aumento del 32,6 %. Se estiman que alrededor de un 58 % de los españoles (26 millones de personas) utilizan habitualmente un *smartphone* en su vida cotidiana. En la figura 1.1 se puede ver de modo gráfico el crecimiento de los *smartphones* y tablets durante el 2014.

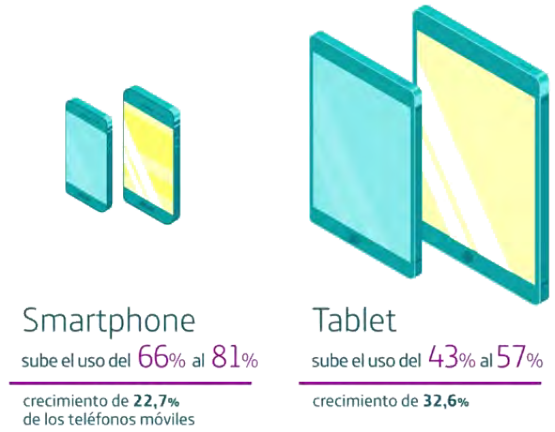

Figura 1.1: Crecimiento del uso de *Smartphone* y *Tablet* en 2014

Este fenómeno ha incrementado el uso de este tipo de dispositivos para actividades relacionadas con el ocio, la comunicación o el deporte, vida cotidiana, entre otras. Por lo tanto, el sector de las aplicaciones para *smartphones* y tabletas se ha convertido en un mercado emergente en constante crecimiento y que avanza con una gran velocidad.

Cada vez más, los usuarios de *smartphones* o tabletas, especialmente los jóvenes, utilizan con mayor frecuencia este tipo de dispositivos con fines recreativos, dejando de lado las consolas para experimentar con juegos más sencillos, basados en la habilidades mentales y que proporcionan una experiencia de usuario diferente. Este tipo de juegos y aplicaciones han tenido una mayor grado de aceptación entre los usuarios más adultos.

Además, surgen nuevas posibilidades para empresas e instituciones, mejorando su comunicación, sus servicios o su comercialización a través de aplicaciones personalizadas.

1.2. Objetivos del proyecto

El objetivo específico del proyecto es desarrollar una **aplicación móvil** para la plataforma **Android** en forma de juego, en concreto, un trivial colaborativo. A los jugadores se les ofrecerá una amplia variedad de opciones de juego. Además, se deberá dar soporte a una gran cantidad de usuarios que utilizarán el sistema concurrentemente.

El objetivo principal del proyecto se puede desglosar en los siguientes objetivos específicos:

- Proporcionar a todos sus usuarios una experiencia de usuario agradable a través de una iteración sencilla e intuitiva.
- Proveer a todos los jugadores una amplia variedad de interacción.
- Ofrecer la posibilidad a todos los jugadores de relacionarse con sus amigos de forma entretenida.
- Promover la participación de todos los jugadores para la construcción de forma colaborativa del contenido de la aplicación.

Por otro lado, el desarrollo del proyecto tiene como objetivos formativos que el alumno encargado de desarrollar la aplicación adquiera unas determinadas competencias y habilidades:

- Aprender acerca de la programación para dispositivos móviles.
- Trabajar cooperativamente en un proyecto de desarrollo de software.
- Conocer las tecnologías y herramientas que se utilizan en el mundo profesional.
- Integrar en un mismo sistema un conjunto de subsistemas.

Capítulo 2

Descripción

Índice

2.1. Descripción general

El departamento de producción de **Soluciones CuatroOchenta** ha decidido desarrollar internamente un sistema informático, que consistirá en un **juego trivial colaborativo**. A través de este sistema, los usuarios tendrán la opción de realizar diferentes pruebas intelectuales, medir sus habilidades con sus amigos, aportar nuevas preguntas o sugerir nuevos temas.

El nuevo sistema informático estará compuesto por cuatro subsistemas:

- **Backoffice**: esta parte del sistema mantiene el acceso restringido a los administradores del sistema. En este subsistema se encuentra alojada la base de datos del sistema, los servicios web a través de los cuales las aplicaciones móviles sincronizan los datos necesarios y las preferencias del sistema.
- **Página web**: *site* web accesible a cualquier usuario. En ella se podrá encontrar toda la información relativa al sistema. Además los usuarios registrados podrán identificarse en ella e interactuar con el sistema a través de cualquier dispositivo conectado a la red.
- Aplicación **Android e iOS**: este es el medio más habitual a través del cual los usuarios interactúan con el sistema. Cualquier usuario que instale la aplicación en su *smartphone* o tableta podrá disfrutar del sistema.

2.2. Descripción funcional

El sistema informático que se pretende construir tendrá tres tipos de usuarios o roles diferenciados:

- **Usuarios no registrados**: se trata de los usuarios del sistema que no están registrados como jugadores o administradores.
- **Usuarios registrados o jugadores**: usuarios que se han registrado a través a de alguno de los mecanismos de registro y autenticación del sistema.
- **Usuarios administradores**: tipo de usuarios especiales encargados de realizar tareas de supervisión y mantenimiento del sistema.

A continuación se va a realizar una breve descripción de la funcionalidad asociada a cada uno de estos roles.

Usuarios no registrados

El sistema deberá mostrar información relativa de las plataformas y mecanismos disponibles para poder jugar al juego que ofrece este sistema. Toda esta información estará disponible para todos estos usuarios a través de una plataforma web y de las correspondientes aplicaciones para Android e iOS.

Usuarios registrados o jugadores

El rol de jugador dispondrá de una amplia funcionalidad dentro del sistema a través de las aplicaciones Android e iOS y la plataforma web.

En primer lugar cabe destacar que este tipo de usuario en concreto deberá de estar registrado en el sistema para acceder a estos servicios. En concreto, los mecanismos disponibles para realizar el registro serán a través de su propia cuenta de Facebook o registrándose en la propia aplicación. Para realizar el registro en la aplicación será necesario que los usuarios introduzcan un nombre de usuario, correo electrónico y contraseña, opcionalmente puede introducir una imagen de perfil.

Los usuarios registrados o jugadores podrán interactuar o jugar de dos modos diferentes:

- **Juego individual**: modo de juego que le permitirá a los jugadores jugar individualmente para conseguir experiencia de usuario.

- **Duelo**: modo de juego en el cual un usuario se enfrentará a un amigo suyo.

Todos los juegos estarán formados por un conjunto de cuestiones o preguntas que contendrán un título, y una serie de respuestas entre las cuales solamente habrá una correcta. Las preguntas pueden tener entre 2 y 4 posibles respuestas. Cada una de las cuestiones pertenece a una única categoría.

Los jugadores podrán buscar sus amigos directamente desde la agenda del dispositivo móvil utilizando los números de teléfono y correos electrónicos o desde **Facebook**. Para que el usuario pueda delimitar su privacidad tendrá opción de bloquear a los usuarios que considere oportuno.

Cada usuario podrá visualizar sus **puntos de experiencia** y el **nivel** del juego en el que se encuentra. Los usuarios también podrán consultar los puntos de usuario obtenidos desglosados en categorías.

Usuarios administradores

Este tipo de usuario tendrá una serie de funciones básicas. En concreto, estos únicamente podrán acceder al *backoffice* del sistema a través de un sitio web en el cual podrán obtener datos y estadísticas detalladas de la aplicación y realizar funciones de mantenimiento y configuración.

En concreto las estadísticas que se podrán obtener son las siguientes:

- Cantidad de usuarios que utiliza al día la aplicación o juego, desglosado en las diferentes plataformas disponibles.
- Cantidad de preguntas aportadas por los usuarios en el sistema en los últimos días, semanas o meses.
- Comprobar que la base de datos se encuentra en un estado correcto.
- Cantidad de monedas o *tokens* totales y seguimiento de todos los usuarios para controlar que no se realizan actividades fraudulentas.

Las funciones que podrán realizar los administradores son las siguientes:

- Bloquear usuarios.
- Añadir nuevas categorías.
- Añadir nuevos idiomas.
- Añadir nuevas preguntas.

2.3. Alcance

> *"La parte más difícil en la construcción de sistemas de software es decidir precisamente qué construir"*
>
> Frederick P. Brooks Jr

Este proyecto abarca únicamente la implementación de la **aplicación Android**, interviniendo en aspectos como el análisis y diseño del sistema completo. El resto de sistema será construido por el personal de la empresa Soluciones CuatroOchenta.

Los resultados o productos esperados tras la realización del proyecto son los siguientes:

- **Aplicación Android** compatible para todos los dispositivos con una versión igual o superior a 4.0 (API 14). En el Apéndice A se detalla porque se ha elegido esta versión.
- **Documento del Análisis del sistema**: este documento contendrá todas las historias de usuario obtenidas tras el estudio del caso y toda su evolución a lo largo del desarrollo del sistema.
- **Documento de Diseño del sistema completo**: este documento contendrá el diseño de datos, de arquitectura, de componentes y de las interfaces gráficas de usuario.

El documento del análisis del sistema y el documento del diseño del sistema completo están contenidos, unificados y relacionados en este documento. Por lo tanto, los productos obtenidos son la **aplicación Android** y el **TFG** correspondiente a este proyecto.

2.4. Tecnologías y herramientas

Las principales tecnologías necesarias para el desarrollo del proyecto son las siguientes:

- **Java**: lenguaje de programación utilizado en el desarrollo de aplicaciones Android.
- **XML**: lenguaje de marcas utilizado para la construcción de múltiples recursos en las aplicaciones Android.
- **HTTP**: protocolo de transferencia de *hipertexto*. Necesario para realizar la comunicación entre la aplicación Android y el servidor que contiene los datos.

Capítulo 3

Análisis y planificación

Índice

3.1. Metodología

Alrededor del 60 % de los problemas que aparecen en los proyectos tienen su origen en los requisitos originados a partir de:

- Mala especificación de los requisitos.
- Cambios de los requisitos funcionales.
- Poca implicación de todas las partes del proyecto.
- Escasa comunicación entre el cliente y el equipo de desarrollo.

En una metodología de desarrollo tradicional antes de empezar a construir el sistema se intenta predecir y detallar todas las variables del proyecto. Además el cliente no tiene la capaci-

dad de ver el producto hasta las últimas fases de desarrollo donde el sistema ya está construido. Todas las etapas del proyecto suelen realizarse secuencialmente. Por lo tanto, si se retrasa una de ellas, se produce un retraso en el proyecto.

La **arquitectura** o las **ingenierías clásicas** (industrial, mecánica, eléctrica, puentes y caminos, etc) necesitan seguir este tipo de ciclos de vida en cascada o predictivos ya que necesitan un **diseño previo**, **exhaustivo** e **inamovible**. Por lo tanto, requieren disponer de los planos antes de empezar la fase de construcción o fabricación. Una vez construidos los cimientos de un edificio no volverá a rediseñar el plano y no se cambiará lo construido. El software tiene diferencias muy sustanciales con respecto a estos productos físicos. Estas diferencias hacen que **el proceso de construcción sea diferente**.

Todos estos problemas los intentan resolver las metodologías ágiles en la ingeniería de software **priorizando los individuos y las interacciones sobre los procesos y las herramientas**, **realizando entregas parciales completamente funcionales**, promoviendo una **colaboración activa** entre todas las partes implicadas y proporcionando una **mayor flexibilidad frente a los cambios** de requisitos.

Para la realización del proyecto se va a seguir una **metodología ágil**. Los motivos por los cuales se ha elegido utilizar este tipo de metodologías se detallan en las siguientes secciones.

3.1.1. ¿Por qué utilizar una metodología ágil en este proyecto?

Uno de los principales motivos por lo cuales se ha decidido utilizar este tipo de metodología es por el tipo de proyecto. Tal y como se ha detallado en secciones anteriores, se trata de un **proyecto interno** del departamento de producción de *Soluciones CuatroOchenta*, por lo tanto, los **requisitos pueden sufrir variaciones** significativas y **cambios en el alcance**. De este modo, debemos de estar preparados para reaccionar rápidamente ante un cambio o giro del proyecto.

Además cabe destacar que, la cultura organizativa y el personal de la empresa son propicios para aplicar este tipo de metodologías, encontrándonos con equipos de trabajo autoorganizados, multidisciplinares y multinivel (equipos formados por personas con diferentes niveles de conocimientos).

El uso de este tipo de metodologías nos aportará una serie de beneficios que enumeramos a continuación:

- Obtener un producto funcional desde las primeras fases o iteraciones.
- *Feedback* del cliente (responsable de la app) de forma muy rápida desde las primeras iteraciones.
- Gestionar los cambios de un modo flexible.

- Producción incremental (cada iteración aporta un valor incremental al producto).
- Reducción de riesgos a largo plazo.
- Gestionar la complejidad del proyecto.
- Conocer el progreso real del proyecto desde el inicio.

3.1.2. Especificación de la metodología

Para el desarrollo del proyecto se ha decidido utilizar una metodología ágil llamada **Scrum** detallada en el apéndice B. En concreto se va a realizar una adaptación y no se aplicará dicha metodología al completo por las características del proyecto y el equipo de trabajo.

Concretamente se ha seleccionado esta metodología por las siguientes características:

- **Estabilidad muy baja de los requisitos**. Como ya se ha comentado los requisitos y el alcance pueden sufrir variaciones.
- **Alto coste de prototipado**. Realizar un prototipo de una aplicación de estas características puede conllevar un consumo elevado de tiempo.
- **Tamaño del proyecto**. Se trata de un proyecto que durará alrededor de 300 horas, por lo tanto, se puede considerar un proyecto de tamaño mediano.
- **Condiciones favorables de la organización**.
- **Cultura organizativa**.
- **Entorno de desarrollo en continua evolución**. En el mundo de la tecnología y la informática, especialmente en el mundo de las aplicaciones móviles surgen nuevas tecnologías y herramientas diariamente que cambian por completo el entorno tecnológico y económico.

3.1.3. Historias de usuario (*User Story*)

En las metodologías ágiles la descripción de las necesidades se realiza a partir de las **historias de usuario** (*user story*) que son, principalmente, lo que el cliente o el usuario quiere que se implemente; es decir, son una descripción breve de una funcionalidad software tal y como la percibe el usuario (M. Cohn, 2004) [3].

De este modo, en los proyecto de desarrollo de software donde se aplican metodologías ágiles, los requisitos no se especifican a través de casos de uso, sino en historias de usuario. Normalmente, las historias de usuario se suelen escribir en tarjetas o *posits* como el que se muestra en la figura 3.1 y contienen la siguiente información:

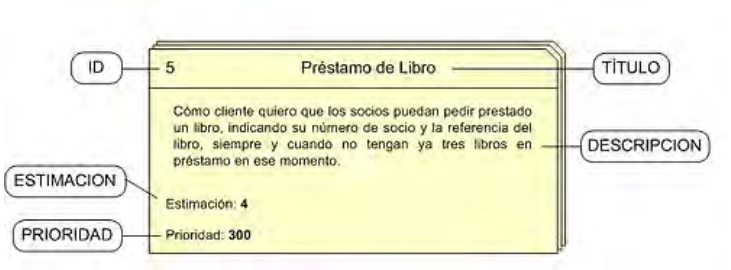

Figura 3.1: Ejemplo de una historia de usuario

- **Id**: identifica la historia de usuario.

- **Título**: describe brevemente la historia de usuario.

- **Descripción**: sintetizada de la historia de usuario. Debe responder a tres preguntas: ¿Quién? ¿Qué quiere? y ¿Cuál es el beneficio?.
  Como [rol del usuario], quiero [objetivo], para poder [beneficio].

- **Estimación**: estimación del tiempo de implementación de la historia de usuario en unidades de desarrollo o **puntos de historia** (unidades que representan el tiempo teórico de desarrollo/persona).

- **Prioridad**: valor numérico que facilita el cliente e indica el valor que le aporta esta funcionalidad dentro del sistema.

3.1.4. Puntos de historia

Las metodologías ágiles han cambiado muchos conceptos en el desarrollo del software, entre ellos la forma de realizar estimaciones. Los puntos de historia aparecieron para dimensionar y relacionar la complejidad de las historias de usuario con respecto a otras. Aunque posteriormente cada empresa o organización que aplica este tipo de metodologías asigna un valor temporal a cada punto de historia.

En nuestro caso, **un punto de historia equivale a una jornada de trabajo** (alrededor de 5 horas), teniendo en cuenta las posibles interrupciones ocasionadas, descansos, etc.

Por lo tanto, al realizar la estimación de nuestras historias de usuario le asignaremos, a cada una de ellas, un valor de **puntos de historia** equivalente al esfuerzo estimado que posteriormente podría llevarnos implementar la funcionalidad descrita en la historia de usuario.

3.1.5. Pila del producto (*Product Backlock*)

Una vez descritos los términos de **historias de usuario** y **puntos de historia** vamos a describir la **pila del producto** (*Product Backlog*).

La **pila del producto** es una lista ordenada o priorizada (según el valor para el cliente) de todas las historias de usuario que forman parte del proyecto. Es decir, para construir la pila del producto vamos a tener que realizar las siguientes acciones:

1. Analizar los requisitos del cliente.
2. Describir o detallar las historias de usuario en tarjetas o *posits*. (Actualmente existen una gran cantidad de herramientas informáticas como *Trello* o *Jira* que nos permiten gestionar las historias de usuario.)
3. Estimar las historias de usuario.
4. El cliente asigna una prioridad a cada historia de usuario asesorado por los miembros del equipo de desarrollo.

3.2. Análisis

3.2.1. Pila del producto inicial(*Initial Product Backlog*)

Tras realizar todas las operaciones necesarias hemos obtenido la siguiente **pila del producto** inicial o *Initial Product Backlog*. La estimación en puntos de historia de las historias de usuario tienen un equivalente de 5 horas de trabajo por punto de historia. Por otro lado, el valor que proporciona la historia de usuario al cliente tienen un valor entre 0 y 100, donde 100 es el valor de máxima importancia para el cliente.

1. **Registrarse como jugador**
   Como jugador quiero registrarme como usuario para poder jugar en el juego.
   Estimación: **2** puntos de historia — Valor: **80**
2. **Registrarse con Facebook**
   Como jugador quiero identificarme como usuario a través de mi cuenta de Facebook para no tener que crearme un usuario.
   Estimación: **4** puntos de historia — Valor: **80**
3. **Identificarte como usuario**
   Como jugador quiero identificarme en la aplicación como usuario registrado para poder jugar.
   Estimación: **4** puntos de historia — Valor: **80**

4. **Recordar contraseña**
   Como jugador quiero recordar mi contraseña a través del correo electrónico para poder identificarte en caso de olvidarla.
   Estimación: **2** puntos de historia — Valor: **70**

5. **Menú**
   Como jugador quiero que la pantalla del menú esté formada por 3 pantallas(tabs) que contengan la información del juego, preguntas, temas sugeridos, puntuaciones y amigos para poder ver de forma resumida las principales estadísticas.
   Estimación: **4** puntos de historia — Valor: **70**

6. **Jugar individualmente**
   Como jugador quiero jugar al juego de forma individual para mejorar mis habilidades.
   Estimación: **4** puntos de historia — Valor: **70**

7. **Ver información general**
   Como jugador quiero acceder a una pantalla de perfil de usuario donde pueda ver toda la información relativa a mi usuario como el nombre de usuario, correo electrónico y foto de perfil, y todos los resultados y estadísticas generales del juego (nivel, logros, puntos de usuario, progreso y preguntas sugeridas o corregidas) para conocer mi progreso en el juego.
   Estimación: **4** puntos de historia — Valor: **60**

8. **Añadir amigos contactos**
   Como jugador quiero buscar a mis amigos a través de los números de teléfono almacenados en mi agenda para jugar contra ellos.
   Estimación: **2** puntos de historia — Valor: **60**

9. **Añadir amigos contactos (email)**
   Como jugador quiero buscar a mis amigos a través de los correos electrónicos almacenados en mi agenda para jugar contra ellos.
   Estimación: **2** puntos de historia — Valor: **60**

10. **Añadir amigos *Facebook***
    Como jugador quiero buscar a mis amigos a través del Facebook para jugar contra ellos.
    Estimación: **2** puntos de historia — Valor: **60**

11. **Jugar Duelo**
    Como jugador quiero retar a mis amigos a un 'Duelo' para obtener puntos extra.
    Estimación: **4** puntos de historia — Valor: **60**

12. **Añadir nueva pregunta**
    Como jugador quiero añadir una nueva pregunta en el juego para obtener más puntos de usuario.
    Estimación: **4** puntos de historia — Valor: **50**

13. **Corregir pregunta**
    Como jugador quiero corregir preguntas propuestas por otros usuarios para obtener puntos

de usuario.
Estimación: **2** puntos de historia — Valor: **50**

14. **Jugar Reto**
    Como jugador quiero medir mis habilidades en un 'Reto' para obtener logros dentro del juego.
    Estimación: **8** puntos de historia — Valor: **40**

15. **Visualizar *Ranking***
    Como jugador quiero ver un *ranking* de usuarios junto a mis amigos para saber el nivel de mis amigos.
    Estimación: **4** puntos de historia — Valor: **30**

Tras construir la pila del producto hemos obtenido un total de 52 puntos de historia. En la sección de **Estimación total del proyecto** 3.3.1 se detalla la estimación realizada para desarrollar este proyecto.

Como se podrá comprobar en el capítulo de **Construcción** [6] durante la realización del proyecto, las historias de usuario evolucionan, cambian sus estimaciones (replanificación), cambia su prioridad, se añaden nuevas historias de usuario, se eliminan historias de usuario que se consideran innecesarias, cambia el alcance del proyecto, etc.

3.3. Planificación

Una de las mayores diferencias entre la planificación de un proyecto ágil como este y la planificación de un proyecto tradicional es la concepción de la actividad a realizar. En la planificación de un proyecto tradicional se trata el desarrollo de software como una **actividad predecible**, cuando el desarrollo de software se trata de una actividad de **creación y transmutación de conocimiento**. Por lo tanto, generalmente no se puede estimar software de forma precisa.

Las metodologías tradicionales intentan realizar una estimación objetiva del tamaño total del proyecto y tras trazar un plan, ejecutarlo, obteniendo el producto a entregar. En cambio, las metodologías ágiles intentan dividir todo el proyecto en pequeñas iteraciones donde se selecciona un subconjunto de historias de usuario, se planifica su ejecución y se ejecuta. De modo que al terminar la iteración existe un **producto funcional no completo** que podemos entregar y obtener *feedback* del cliente sin tener que esperar al finalizar el proyecto. Por lo tanto, los métodos ágiles buscan entregar valor incrementalmente al cliente en cada iteración.

De este modo, al terminar una iteración, se repite el proceso de seleccionar otro subconjunto de historias de usuario, planificar su ejecución y ejecutar, hasta el momento de agotar todas las historias de usuario. Esto significa que el producto está completo.

Cabe destacar que para hacer una estimación total aproximada de la duración del proyecto es posible sumar todos los puntos de historia de las historias de usuario de la pila del producto, con este valor y conociendo la cantidad de programadores que intervendrán podemos predecir **aproximadamente** el tiempo de construcción del proyecto completo.

3.3.1. Estimación total del proyecto

Tal y como se ha comentado en la sección anterior, para obtener la estimación total aproximada del proyecto debemos de sumar los puntos de historia de cada una de las historias de usuario de la pila del producto. En nuestro caso la suma total de puntos de historia es de **52**. Suponemos que nuestra la velocidad de desarrollo de software será de 1 punto de historia diario (5 horas). Obtenemos que la duración total del proyecto es de 260 (52 ph * 5 h/ph) horas aproximadamente.

Por lo tanto, esta planificación se ajusta a las 300 horas de prácticas realizadas en la empresa. Se ajusta completamente, debido a que durante las 2 primeras semanas de trabajo (40 horas) se han empleado a formación y no se han utilizado para el desarrollo del proyecto.

3.3.2. Planificación de iteraciones o *Sprints*

A lo largo de todo el proyecto se va a realizar la planificación de una serie de iteraciones o *sprints*. La cantidad total estimada de iteraciones necesarias para completar el proyecto con éxito es de aproximadamente **6** iteraciones o *Sprints*. Ya que disponemos de 260 horas de trabajo real en el proyecto y considerando que realizaremos una iteración cada 2 semanas de trabajo (40 horas) obtenemos que la cantidad total de *sprints* a realizar es de 6,5, por lo tanto, redondeando obtenemos 6 *sprints*, 5 iteraciones de 2 semanas y otra iteración más amplia de 3 semanas.

Capítulo 4

Diseño de Software

Índice

En un proyecto ágil al inicio del proyecto se realiza un diseño básico y muy sencillo del proyecto software que posteriormente se va a construir. De tal modo que el diseño ha sido refinado y perfeccionado durante las siguientes iteraciones o *sprints*.

En las siguientes secciones se va a mostrar el diseño final obtenido tras todas las iteraciones de software.

El principal **objetivo** de realizar un diseño de software es que nos permite describir cómo el sistema va a satisfacer los requisitos. Esta etapa tiene diferentes niveles de detalle. Los niveles más altos de detalle generalmente describen los componentes o módulos que formarán el software a ser producido. Mientras que los niveles más bajos, describen cada módulo con mucho detalle.

Otro de los muchos motivos por los cuales se realiza este proceso de diseño en el software es para obtener un producto:

- Funcional
- Usable

- Fiable
- Eficiente
- Mantenible

Por consiguiente, en las siguientes secciones vamos a ver el diseño de datos, la arquitectura que presenta el sistema, el diseño de las diferentes interfaces gráficas de usuario, y el diseño de componentes.

4.1. Diseño de datos o clases

El diseño de datos o clases intenta representar las entidades que conformarán el sistema, sus relaciones y las dependencias entre entidades de un modo sencillo y gráfico. En la figura 4.1 se muestra el diagrama de clases final de la aplicación Android.

Esta representación no refleja a la perfección el esquema real de la base de datos en el dispositivo Android. El principal motivo es debido al uso de una base de datos relacional y no una base de datos orientada a objetos. Este problema lo resolvemos utilizando un ORM (*Object-Relational Mapping*) herramienta que nos permite utilizar una base de datos relacional como motor de persistencia utilizando un lenguaje de programación orientado a objetos. Es decir, esta herramienta utiliza una **base de datos orientada a objetos virtual**, sobre una base de datos relacional.

Para persistir los datos en la aplicación Android se ha utilizado un ORM propio de la empresa Soluciones CuatroOchenta llamado MDF (*Model Data Framework*) encargado de construir una base de datos orientada a objetos utilizando una base de datos relacional. En el apéndice C se detalla el uso de este *framework* y la base de datos física utilizada.

4.2. Diseño de la arquitectura

La arquitectura de un sistema informático representa la estructura de los datos y los componentes del programa requeridos para construirlo. La arquitectura de un sistema constituye el estilo que tendrá el sistema, la estructura y las propiedades de los componentes que lo conforman y la interrelación entre los componentes del sistema.

De este modo, la arquitectura de un sistema es una **vista estructural de alto nivel** que define el estilo o combinación de estilos para una solución. Esta definición se concentra en requerimientos no funcionales que se satisfacen mediante el modelado y el diseño de aplicación. Por lo tanto, este proceso es esencial para conseguir el éxito de un proyecto.

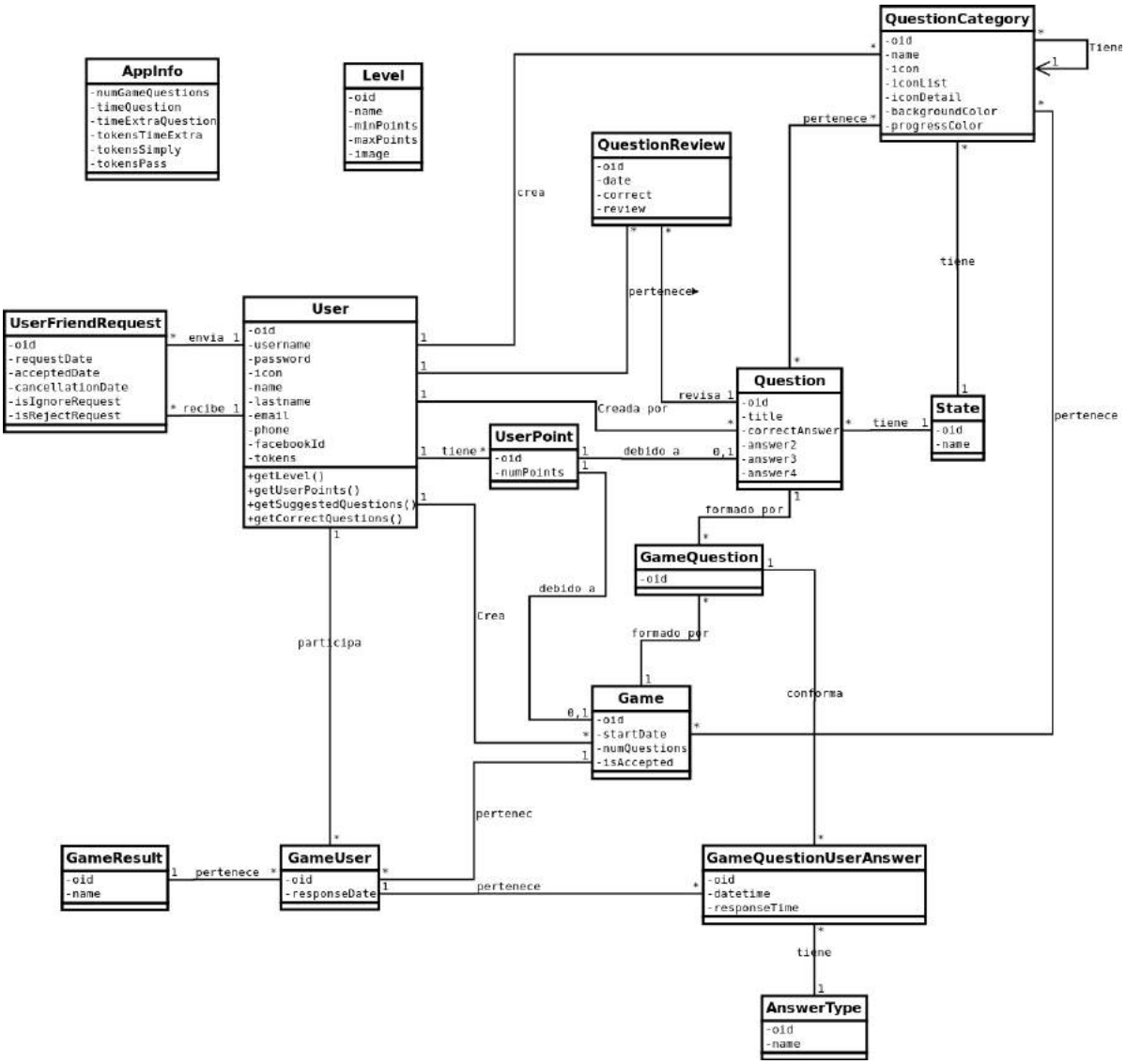

Figura 4.1: Diagrama de clases

Es muy importante definir correctamente la arquitectura de un sistema informático por tres razones:

- Las representaciones de la arquitectura facilitan la comunicación entre todos los involucrados en su desarrollo y nos permiten ver una visión global (*Big Picture*) del sistema como un conjunto de subconjuntos.

- Destaca las decisiones tempranas que tendrán un profundo impacto en el proceso de desarrollo y que, en última instancia, determinan el éxito final del sistema.

- El tamaño 'reducido' de la arquitectura permite entender la estructura del sistema y las relaciones de todos sus componentes.

4.2.1. Estilo arquitectónico

El estilo de este sistema es una combinación de varios estilos arquitectónicos. Este patrón se repite en la gran mayoría de los sistemas informáticos enfocados a dar soporte a aplicaciones móviles.

Los tres patrones o estilos arquitectónicos que se funden para dar lugar al estilo de este sistema son la **arquitectura centrada en los datos**, la **arquitectura orientada a objetos** y la **arquitectura cliente/servidor**.

La **arquitectura centrada en los datos** está construida alrededor de un almacén de datos donde los clientes operan de independiente comunicándose entre ellos a través del almacén de datos. En cambio, la **arquitectura orientada a objetos** los componentes encapsulan los datos y se utilizan las operaciones para manipularlos. La comunicación y comunicación se efectúa mediante paso de mensajes.

Por lo tanto, combinando estas dos arquitecturas obtenemos un sistema característico donde todos los componentes están construidos alrededor de un almacén de datos que contiene toda la información, permitiendo la cooperación y comunicación de todos y cada uno de los componentes conectados a él. Cada componente internamente implementa una arquitectura orientada a objetos, donde encapsula todos los datos y aplica operaciones para manipularlos.

Podemos concluir, que el sistema finalmente presentará una arquitectura **cliente/servidor** sin estado, tal y como sucede con el **protocolo HTTP** (clientes acceden al servidor para obtener información, en este caso páginas webs). Por lo tanto, existirá un servidor o almacén de datos que contendrá toda la información y una gran cantidad de clientes que se comunican con él para obtener datos.

4.2.2. Arquitectura del sistema

El sistema está formado por 4 subsistemas claramente diferenciados como se comenta en la sección 2.1 (Descripción general). Podemos ver representados gráficamente los subsistemas en la figura 4.2 donde se muestra el **sistema completo como conjunto de subsistemas** que **interactúan** entre ellos con el fin de **lograr un objetivo común**, la satisfacción total del usuario.

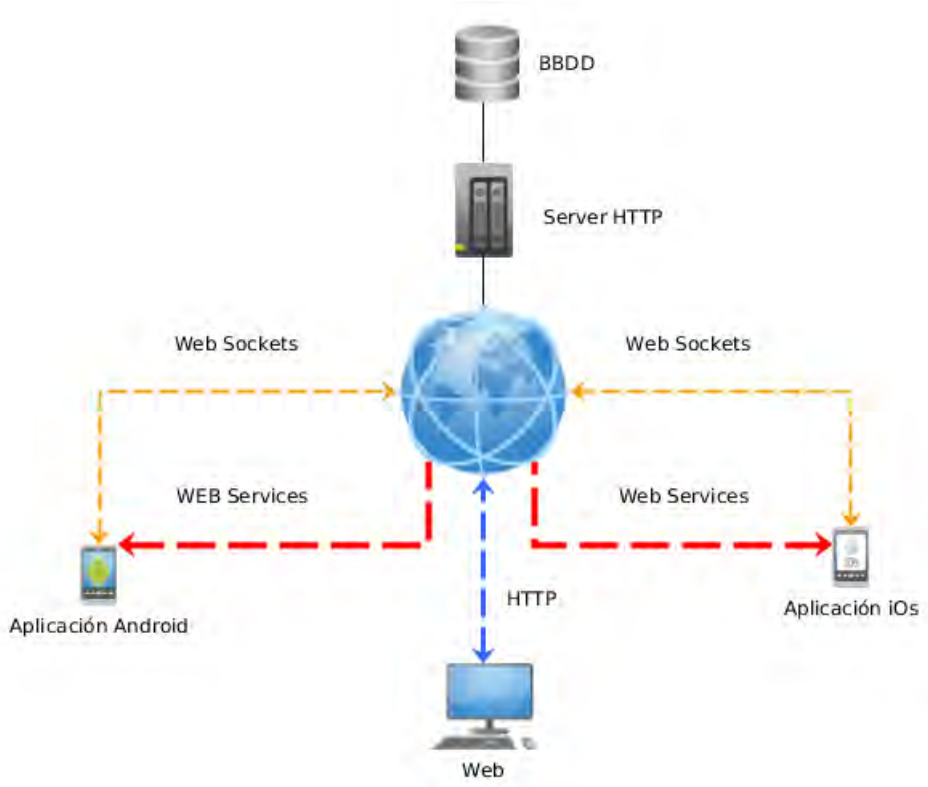

Figura 4.2: Arquitectura del sistema completo.

A continuación se describen los subsistemas y las relaciones entre ellos:

- **Backoffice** o **server HTTP**: esta parte del sistema mantiene el acceso restringido a los administradores del sistema. En este subsistema se encuentra alojada la base de datos del sistema, los servicios web a través de los cuales las aplicaciones móviles sincronizan los datos necesarios y las preferencias del sistema. Este componente realiza la función de **almacén de datos**. Por lo tanto, el resto de componentes del sistema acceden a él para obtener los datos que darán vida a la aplicación. También nos permite realizar una comunicación entre varios elementos del sistema independientemente del subsistema al que pertenezcan.
- **Página web**: *site* web accesible a cualquier usuario. En ella se muestra toda la información relativa al sistema. Los usuarios registrados podrán identificarse en ella e interactuar con el

sistema a través de cualquier dispositivo conectado a la red. Este componente se comunica con el servidor o **backoffice** a través del protocolo HTTP del mismo con el que obtenemos cualquier otra web alojada en Internet.

- Aplicación **Android e iOS**: este es el medio más habitual a través del cual los usuarios interactúan con el sistema. Cualquier usuario que instale la aplicación en su *smartphone* o tableta podrá disfrutar del sistema. Estas aplicaciones juegan el rol de **clientes**. Es decir, se conectan al servidor o **backoffice** a través de *Web Sockets* (Node.js) o *Web Services* (HTTP) para obtener los datos. Además de esto, cada uno de estos subsistemas mantendrá una arquitectura interna orientada a objetos transparente al resto de subsistemas.

En la siguiente sección [4.3] vamos a estudiar en profundidad el subsistema Android. El resto de componentes (completamente diferentes) no serán vistas en este documento, ya que quedan fuera del alcance de este proyecto.

4.2.3. Aplicación Android

Arquitectura del sistema operativo Android

En primer lugar, antes de mostrar la arquitectura de la aplicación Android y sus pequeños módulos, vamos a ver una pequeña introducción a la arquitectura del sistema operativo Android.

Android es una plataforma o sistema operativo para dispositivos móviles basado en **Linux**. Presenta una arquitectura estratificada o por capas como la mayoría de los sistemas operativos. En la figura 4.3 se pueden apreciar las capas principales que se describen a continuación:

- **Aplicaciones**: aquí se encuentran todas las aplicaciones incluidas por defecto en Android y aquellas que el usuario instale posteriormente. Todas estas aplicaciones utilizan los servicios, API y librerías de los niveles anteriores.
- ***Framework* de Aplicaciones**: representa el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android utilizan el mismo conjunto de API y el mismo "framework", representado por este nivel.
- **Librerías**: esta capa se corresponde con las librerías utilizadas por Android y proporcionan la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android. (libc, Surface MAnager, OpenGL/SL, Média libraries, SSL, SQLite, WebKit)
- **Entorno de ejecución de Android**: al mismo nivel que las librerías de Android se sitúa el entorno de ejecución. Éste lo constituyen las *Core Libraries* y la máquina virtual **Dalvik**.

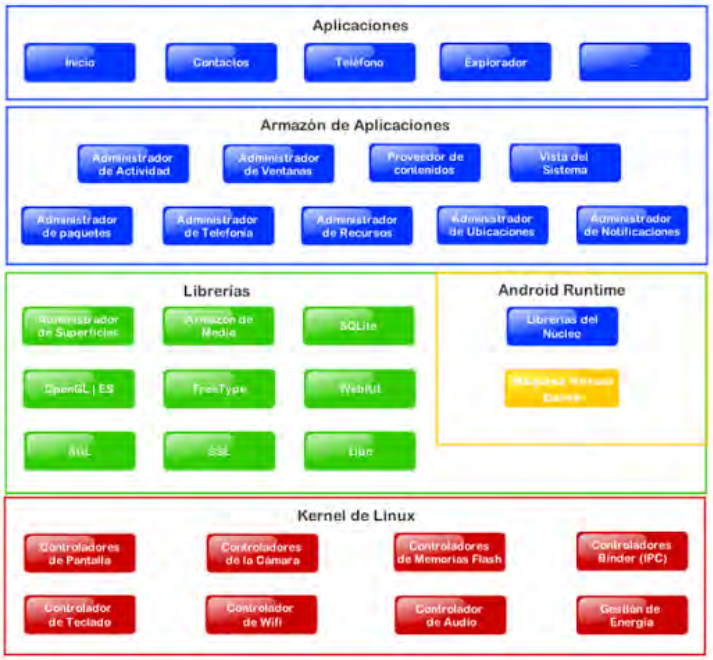

Figura 4.3: Arquitectura Android.

- **Núcleo Linux**: Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles.

4.2.4. Arquitectura de la aplicación Android

Una vez presentada la arquitectura del sistema operativo Android pasamos a ver la arquitectura de nuestra aplicación. En primer lugar, hay que destacar que la aplicación Android se va a localizar en la **capa de aplicaciones** recurriendo a la utilización de otras aplicaciones o librerías contenidas en otras capas como por ejemplo *SQLite* localizada en las librerías de Android.

En la aplicación Android se ha decidido utilizar el patrón arquitectónico **Modelo-Vista-Controlador** (*Model-View-Controller* o MVC). Este patrón arquitectónico es muy común en este tipo de aplicaciones ya que el propio sistema favorece su aplicación.

El principal objetivo de MVC es separar el modelo de datos, la interfaz gráfica de usuario y la lógica de negocio. Para ello MVC propone la construcción de tres componentes distintos:

- **Modelo**: este módulo contiene las representaciones de datos utilizadas en la aplicación (por ejemplo `User` o `Question`) y el acceso al mecanismo de persistencia. Por lo tanto, el modelo es el módulo encargado de mantener la persistencia de los datos en la base de datos *SQLite* a través de MDF (ORM) y obtenerlos de ella cuando sea necesario.

- **Vista**: representa la **interfaz gráfica de usuario** (GUI). El usuario interactúa con la aplicación a través de este componente. Este componente no tiene estado, sino que se encarga de delegar todas las acciones al controlador.

- **Controlador**: componente encargado intermediar entre la vista y el modelo. Contiene toda la lógica de negocio de la aplicación y se encarga de responder a los eventos delegados por parte de la vista (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud de información.

En la imagen 4.4 se puede observar de forma gráfica los componentes y las interacciones entre los diferentes componentes.

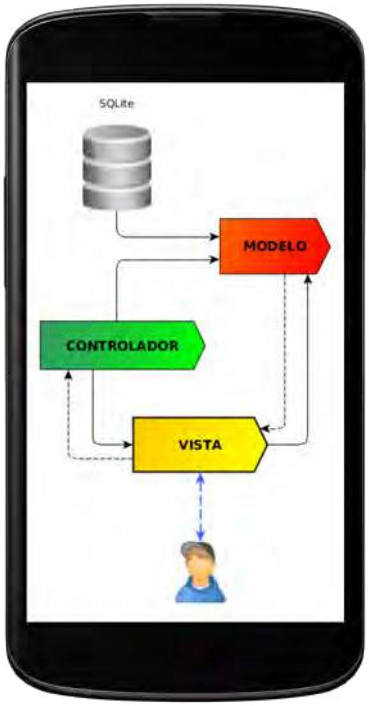

Figura 4.4: Arquitectura aplicación Android.

Aplicando este patrón arquitectónico hemos conseguido obtener una serie de beneficios como **reutilizar una gran cantidad de código**, **separación de conceptos** (*Separation of Concerns*), crear tres módulos completamente independientes y reemplazables, facilitar el desarrollo de la aplicación, obtener una mayor **mantenibilidad** y **escalabilidad** de la aplicación.

4.3. Diseño de la interfaz gráfica de usuario (GUI)

El **interfaz de usuario** es el medio o canal mediante el cual el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo.

Las interfaces gráficas de usuario deben de seguir una serie de reglas que se enumeran a continuación:

- Proporciona una interacción flexible.
- Permitir la interacción directa con objetos que se muestran en pantalla.
- Permitir que el usuario interrumpa o deshaga acciones.
- Reducir la carga de memoria del usuario.
- El formato visual de la interfaz se deberá basar en una metáfora del mundo real.
- Organizar jerárquicamente la interfaz.
- Mantener la consistencia de la interfaz.

Es muy importante realizar un diseño previo de la interfaz gráfica de usuario (GUI) antes de la fase de construcción o desarrollo por varios motivos. En primer lugar, realizar este diseño le permite a los programadores **ver el aspecto final de la aplicación**, de este modo, se construirá una interfaz gráfica similar al diseñado. En segundo lugar, permite enfatizar con el cliente final del sistema para estudiar sus necesidades y **proporcionar una interfaz gráfica que se adapte a su experiencia y a sus necesidades**.

En muchos entornos profesionales, como el diseño de páginas webs, son los propios diseñadores los encargados de escribir el código de la interfaz gráfica.

4.3.1. Interfaz gráfica

El diseño de la interfaz gráfica de la aplicación Android ha sido **realizado por el departamento de diseño** como sucede con todas las aplicaciones desarrolladas en la empresa. Se trata de profesionales con amplia experiencia en el sector y con conocimiento total acerca de la usabilidad y otros aspectos clave en el diseño de interfaces gráficas para dispositivos móviles.

Durante el desarrollo de la aplicación se ha necesitado una colaboración activa con los responsables del diseño gráfico para lograr el objetivo común: el éxito de la aplicación.

El diseño gráfico de la aplicación está inspirado en el tiempo meteorológico. En las siguientes figuras se muestran todos los diseños realizados.

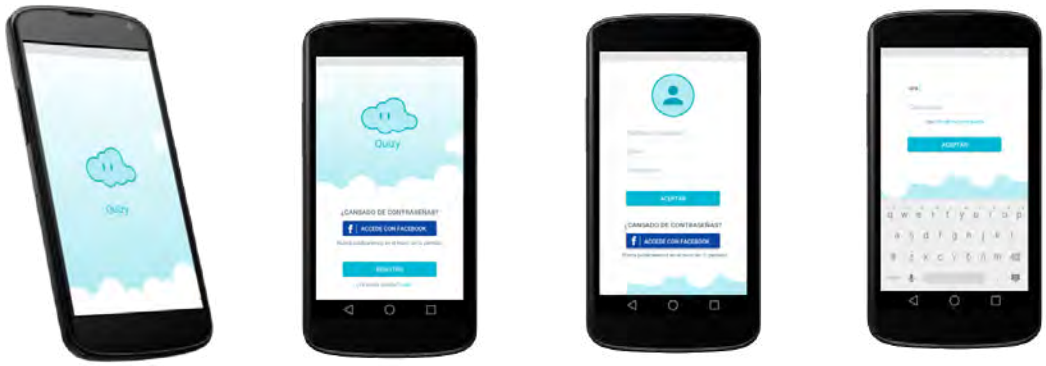

Figura 4.5: Diseño de pantallas de *Splash*, inicio, registro y *login*.

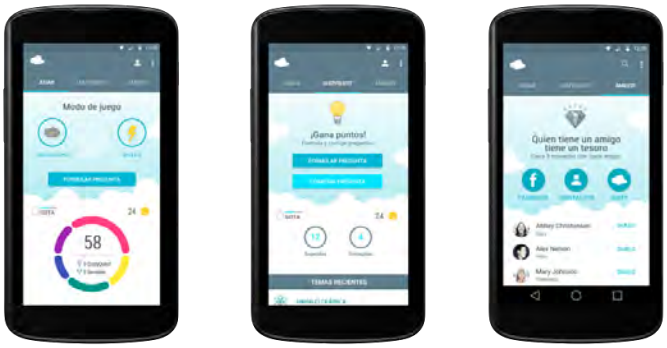

Figura 4.6: Diseño de pantallas de Menú.

4.3.2. Jerarquía de pantallas

Finalmente,en la figura 4.12 se muestra la jerarquía de pantallas que presenta la aplicación mediante los diseños presentados anteriormente. Gracias a esta imagen conocemos el flujo de pantallas a través del cual el usuario final podrá navegar.

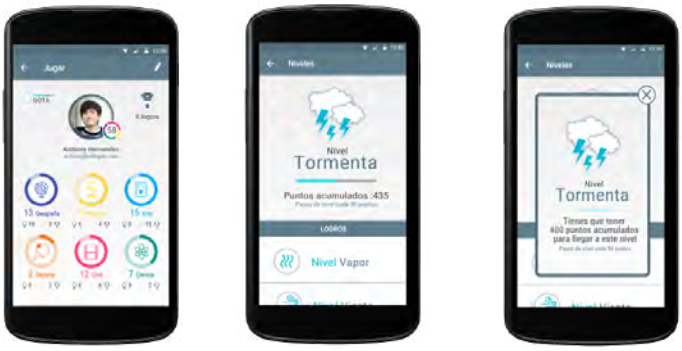

Figura 4.7: Diseño de pantalla de perfil y detalles de niveles.

Figura 4.8: Diseño de pantallas de juego individual.

Figura 4.9: Diseño de pantallas de Duelo.

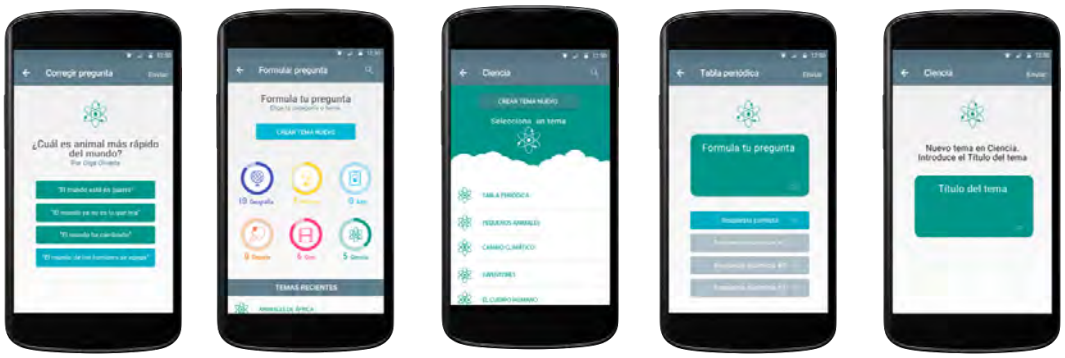

Figura 4.10: Diseño de pantallas de añadir preguntas y corregir preguntas.

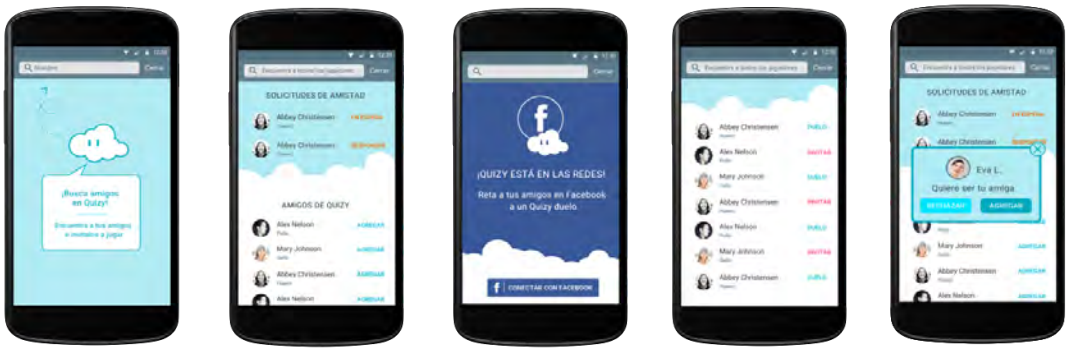

Figura 4.11: Diseño de pantallas de gestión de usuarios.

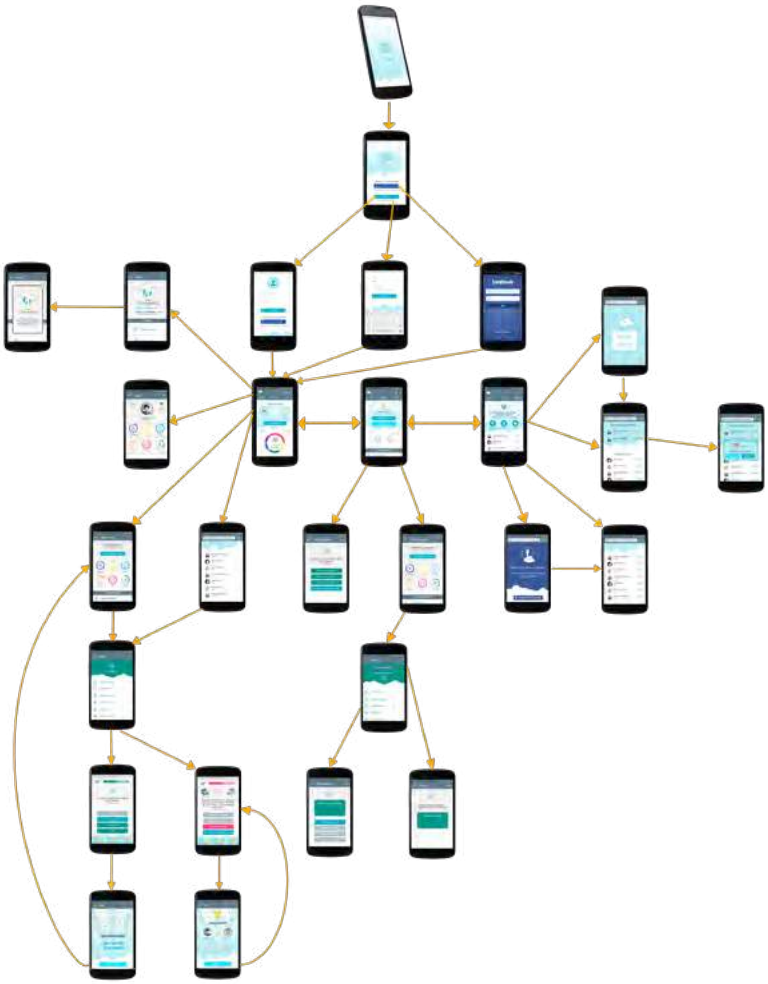

Figura 4.12: Jerarquía de pantallas.

Capítulo 5

Implementación y pruebas

Índice

ALTO NIVEL (BIG PICTURE)

5.1. Tecnologías y herramientas utilizadas

Las tecnologías y herramientas que se han utilizado en la construcción de la aplicación Android son las siguientes:

- **Java**: lenguaje de programación utilizado para programar la aplicación Android.
- **XML**: lenguaje de marcas utilizado para la construcción de aplicaciones Android. Los elementos más comunes que se construyen a partir de ficheros XML son las interfaces gráficas de usuario (GUI) y ficheros de configuración.
- **HTTP**: *Hyper Text Transfer Protocol* es el mecanismo a través del cual la aplicación transmite datos con el servidor web. Este tipo de operaciones son síncronas y sin estado.
- **Node.js y SocketIO**: Node.js es un nuevo *framework* Javascript del lado del servidor que nos permite realizar comunicaciones cliente/servidor de forma asíncrona y crear programas de red altamente escalables. Este *framework* utiliza la tecnología SocketIO para realizar la comunicación, por lo tanto, podemos realizar comunicaciones con cualquier tipo de

plataforma gracias a la gran cantidad de librerías SocketIO disponibles para una gran cantidad de lenguajes de programación como Java, Swift, C, C++ o Python.

- **Android Studio**: IDE de desarrollo.
- **CocoaRestClient**: cliente de peticiones HTTP, utilizado para hacer pruebas sobre *Web-Service*.
- **SqlPro for SQLite**: herramienta utilizada para mostrar el esquema y contenido de bases de datos *SQLite*.
- **Android Assets Studio**: sitio de web que nos permite obtener los iconos y gráficos de la aplicación específicos para cada resolución de pantalla.
- **Adobe Ilustrator**: editor de gráficos vectoriales, utilizado para la explotación de los diseños gráficos de usuario.

5.2. Detalles de implementación

A lo largo de esta sección se presentan detalles de la implementación, así como la organización en pequeños módulos y algunos de los patrones de diseño utilizados a lo largo del código.

5.2.1. Módulos de software

La estructura de un proyecto Android nos proporciona los mecanismos correctos para separar ya todos los recursos (vistas, imágenes, audio, ...) del código de nuestra aplicación. A pesar de esto el código Java hemos de mantenerlo organizado en un conjunto de módulos. Los diferentes módulos que hemos utilizado para organizar nuestro código son los siguientes:

- `src`
  - `activities`: contiene actividades comunes de la aplicación.
  - `alerts`: este módulo aloja todas las clases necesarias para mostrar alertas a los usuarios.
  - `play`: en el interior de este módulo se encuentran todos los elementos y controladores necesarios para que el juego funcione correctamente.
  - `model`: contiene las clases del modelo.
  - `notifications`: contiene todos los elementos necesarios para recibir y mostrar las notificaciones de usuario.
  - `services`: contiene todos los *Web Services* de la aplicación.

- `social`: contiene todos los controladores y adaptadores necesarios para implementar las pantallas de amigos.
- `utils`: contiene clases de utilidad como `SocketUtils` o `LoginUtils`.
- `wiki`: contiene los controladores y adaptadores necesarios para implementar las pantallas de añadir preguntas y corregir preguntas.

▪ `res`

- `drawable`: contiene todos los recursos visuales de la aplicación (imágenes, iconos, botones personalizados, ...).
- `layout`: contiene todas las vistas de la aplicación.
- `values`: contiene ficheros de colores, dimensiones y palabras, fundamentales para la representación de las vistas y la reutilización de código en las vistas.
- `menú`: contiene los menús de la aplicación.
- `xml`: contiene ficheros XML que no encajan en cualquier otro directorio. Suelen ser ficheros de configuración de herramientas externas al sistema Android.

5.2.2. Patrones de diseño

Los patrones de diseño son soluciones genéricas para problemas típicos y recurrentes que nos podemos encontrar a la hora de desarrollar una aplicación. A continuación se describen los patrones de diseño utilizados durante la implementación de la aplicación Android con el fin de proporcionar una solución sencilla, mantenible y flexible.

Singleton

Se trata de un patrón de diseño de la categoría de **creación**. El principal objetivo es asegurarse que sólo existe una única instancia de una determinada clase y proporcionar un punto de acceso a dicha instancia. En la figura 5.1 se muestra la estructura de dicho patrón.

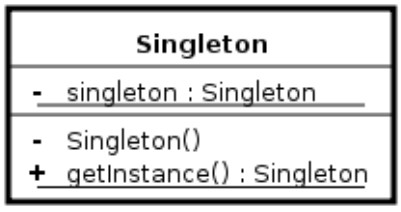

Figura 5.1: Estructura patrón de diseño *Singleton*.

Este es, sin duda, uno de los patrones de diseño más utilizados en Android. A lo largo del proyecto este patrón se utiliza en un gran conjunto de clases, sobre todo clases de utilidad, como

`SocketUtils` (permite comunicación con servidor a través de sockets. El coste computacional y temporal de de crear este objeto es muy elevado, por lo tanto, es muy importante mantener una única instancia), `FontManagerUtils` (se encarga de manejar las fuentes de texto. El coste computacional y temporal de crear este objeto es muy alto.) o `I18Utils` (permite gestionar las cadenas dentro de nuestra aplicación. El coste computacional y temporal de crear este objeto es muy alto.).

El motivo de utilizar este patrón ha sido el gran coste de inicialización de varios objetos y la necesidad de mantener una única referencia en todo el sistema.

Builder

Se trata de un patrón de creación. Pretende construir un objeto complejo sin tener en cuenta su representación, de modo que, el mismo proceso de construcción permite crear diferentes representaciones. Es otro elemento el encargado de obtener la instancia y representar el objeto.

A lo largo de este proyecto se ha utilizado este patrón concreto para delegar en una clase `Builder` la responsabilidad de crear todos los diálogos modales de la aplicación. En la figura 5.2 se presenta la estructura de este patrón y la aplicación o utilidad que representa en este proyecto. Como se puede apreciar en la figura, existe un cliente anónimo que necesita un diálogo para mostrarlo en la pantalla. Este cliente llama a un *builder* que se encarga de crear (creación) y configurar el nuevo diálogo y devolverlo al cliente. A partir de este momento ya es el cliente el encargado de representarlo o de realizar las acciones oportunas con esta instancia.

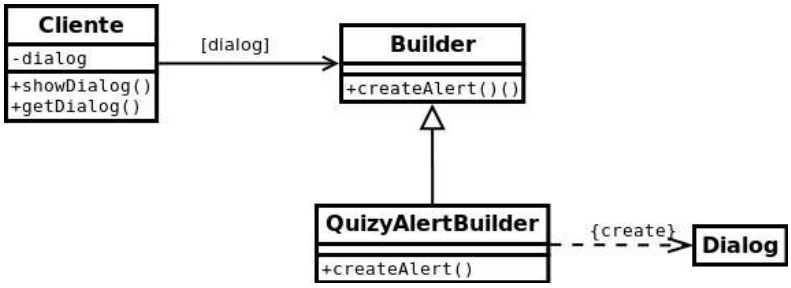

Figura 5.2: Estructura patrón de diseño *Builder*.

A lo largo del proyecto, el cliente se encarga de mostrar el diálogo en la ventana principal de la aplicación como se muestra en la figura 5.3. Este patrón nos ha sido muy útil ya que la responsabilidad de crear y mostrar el diálogo están separadas. De modo que si decidimos cambiar el comportamiento o el formato visual solamente será necesario cambiar uno de los elementos implicados y no todo el sistema de alertas.

Figura 5.3: Diálogo de “Solicitud de amistad”.

5.3. Pruebas unitarias

Las pruebas unitarias o test unitarios son un modo de comprobar el correcto funcionamiento de un módulo de código independiente. Este tipo de pruebas son útiles para asegurar todos y cada uno de los módulos funcione correctamente por separado. Este tipo de pruebas han de seguir los principios **FIRST** (*fast*, *independient*, *repeatable*, *self-validating* y *timely*). Estos principios recomiendan pruebas rápidas, independientes, repetibles, autovalidables y escribirse antes o junto con el código que pretenden probar.

Para el diseño de las pruebas unitarias de este proyecto se han utilizado **JUnit**. JUnit es un *framework* de código abierto que permite definir test unitarios para una gran cantidad de lenguajes de programación, entre los cuales se encuentra Java.

A lo largo del proyecto se han realizado test unitarios a los módulos de comunicación (Sockets), módulo de análisis (encargado de obtener los datos y estadísticas del usuario identificado) y al módulo de juego (implementa la funcionalidad del juego de trivial).

5.4. Pruebas de aceptación

Las pruebas de aceptación se encargan de verificar que un producto o sistema cumpla con los estándares necesarios y satisfaga a los usuarios cumpliendo los requisitos establecidos. En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique.

A lo largo del desarrollo de este proyecto se han realizado una serie de iteraciones o *sprints*. Durante la reunión de planificación se diseñan las pruebas de aceptación relativas a cada historia de usuario. Al finalizar el *sprint* se realiza una comprobación del producto. Si el sistema satisface las pruebas de aceptación se considera que el *sprint* se ha terminado con éxito. En caso contrario, en la siguiente iteración se deben resolver los problemas para satisfacer la prueba de aceptación.

Las pruebas de aceptación tienen la siguiente estructura:

> **DADO un escenario**
> **CUANDO se produzca un evento concreto**
> **ENTONCES se produce una salida y/o una actualización del estado del sistema**

Para el diseño de los test de aceptación se han seguido los principios INVEST para la definición de *user stories*:

- **Independent**: puede desarrollarse de forma independiente.
- **Negotiable**: resultado de análisis conjunto de todas las partes.
- **Valuable**: tiene valor de negocio para el cliente.
- **Estimable**: debe poderse estimar el esfuerzo que requiere su desarrollo .
- **Small**: debe poder completarse en una única iteración o ciclo.
- **Testable**: el cliente/usuario debe poder confirmar que se ha completado.

Capítulo 6

Construcción e iteraciones

Índice

A lo largo de este capítulo se va a realizar un seguimiento de las iteraciones o *Sprints*

realizados a lo largo de la ejecución del proyecto. De este modo, en cada sección se va a presentar una iteración o *Sprint* indicando las historias de usuario seleccionadas y planificadas (pila del sprint), la descomposición en tareas de cada historia de usuario, la estimación detallada de cada historia de usuario por tareas, las pruebas de aceptación de cada una de las historias de usuario y el resultado final obtenido tras la realización del sprint. Además de esto, a lo largo de las 6 iteraciones se va a realizar un seguimiento de la evolución de la pila del producto, es decir, los cambios que experimenta en cada iteración la pila del producto a la cual se le añaden nuevas historias de usuario, se eliminan historias existentes, cambios de estimaciones, etc.

6.1. Sprint 1

6.1.1. Planificación del Sprint

A continuación se muestra la pila del sprint 1, formada por 4 historias de usuario descompuestas en tareas estimadas que nos proporcionarán una estimación aproximada del tiempo que nos llevará a implementarlo.

Yo como jugador quiero registrarme como usuario para poder jugar en el juego.

- Crear pantalla de Bienvenida (ventana de inicio) con todos sus componentes tal y como detallan los diseños [3 horas].
- Crear una actividad (*Activity*) para manejar la lógica de negocio de los componentes gráficos y los mecanismos de traducción de los textos [1 hora].
- Crear petición y realizar la llamada al servicio web (ya implementado) de registro de usuario [2 horas].
- Procesar valor devuelto por el servicio web de la petición de registro de usuario y mostrar el resultado [2 horas].

Tiempo total: [8h] $\Rightarrow$ 2 puntos historia (10 horas)

Yo como jugador quiero identificarme como usuario a través de mi cuenta de Facebook para no tener que crearme un usuario.

- Añadir e investigar el funcionamiento de la librería de Facebook en aplicaciones Android [3 horas].
- Creación de la aplicación en la plataforma Facebook y configuración de todos sus parámetros [2 hora].
- Añadir elementos gráficos a las interfaces de usuario de login e inicio de la aplicación y configurar su funcionamiento[1 hora].
- Realizar la llamada y recepción de identificación de usuario a través de un servicio web [2 horas].
- Mostrar mensajes de progreso y de error en caso de fallar el proceso de identificación [2 horas].
- Pruebas de test y debug [2 hora].

Tiempo total: [12 h] ⇒ 4 puntos de historia

Yo como jugador quiero identificarme en la aplicación como usuario registrado para poder jugar.

- Crear la pantalla de identificación de usuarios (Login) [2 hora].
- Añadir lógica de negocio de los componentes de la pantalla de login [2 hora].
- Realizar la llamada y recepción de llamada a un servicio web para la identificación de usuario [3 horas].
- Mostrar mensajes de progreso y de error en caso de fallar el proceso de identificación [2 horas].

Total [9 horas] ⇒ 2 puntos de historia (*)

Yo como jugador quiero recordar mi contraseña a través del correo para poder identificarme en caso de olvidarla.

- Añadir un nuevo componente en la pantalla de login [1 hora].
- Añadir lógica de negocio al componente añadido [2 hora].
- Realizar la llamada y recepción de llamada al servicio web correspondiente para recordar contraseña [3 horas].

Tiempo total: [6 h] ⇒ 2 puntos de historia

(*) El valor de punto de historia al realizar la planificación ha variado en comparación a la estimación inicial debido a que al descomponer la historia de usuario en pequeñas tareas se ha observado que su estimación era muy pesimista y se ha rectificado el valor de puntos de historia.

Un aspecto a tener en cuenta en la planificación es que al calcular los puntos de historia de cada una de las historias de usuario este valor debe de ser potencia de 2 (1, 2, 4, 8, 16, ...), por lo tanto, si la suma de la estimación de todas sus tareas es un número diferente de una potencia de 2 se selecciona la potencia inmediatamente superior. Esto se realiza para estimar las historias de usuario de modo optimista y prevenir pequeños contratiempos que puedan aparecer.

6.1.2. Pruebas de aceptación

Tras la finalización de este primer sprint la aplicación debe cumplir con las siguientes pruebas de aceptación descritas, sino, no se puede considerar como completado las historias de usuario

correspondiente. A continuación se muestra una lista con las pruebas de aceptación. Las pruebas marcadas de color verde se tratan de pruebas que tras el sprint se satisfacen correctamente y en rojo las pruebas de aceptación que no satisfacen la prueba o no pueden ser validadas por falta de datos.

- **DADO** que un usuario introduce su nombre de usuario, email y una contraseña, **CUANDO** este selecciona la opción de registrarse **ENTONCES** el usuario es registrado como un jugador de la aplicación.
- **DADO** que un usuario no introduce su email, contraseña o nombre de usuario **CUANDO** proceda a registrarse en la aplicación **ENTONCES** se mostrará un mensaje de error indicando el problema sucedido.
- **DADO** un usuario no registrado que quiere jugar al juego a través de su cuenta de Facebook **CUANDO** se identifique con los datos de Facebook **ENTONCES** podrá acceder al juego como un usuario registrado.
- **DADO** un usuario registrado **CUANDO** se identifique proporcionando los datos correctos **ENTONCES** podrá acceder al juego.
- **DADO** un usuario registrado **CUANDO** trate de identificarse en el juego indicando mal su correo o contraseña **ENTONCES** se muestra un mensaje de error indicando que el nombre de usuario o contraseña son incorrectos.
- **DADO** un usuario no registrado **CUANDO** intenta identificarse como usuario **ENTONCES** se muestra un mensaje de error indicando que el usuario no está registrado.
- **DADO** un usuario registrado intenta acceder a la aplicación pero no recuerda la contraseña **CUANDO** selecciona la opción de recordar contraseña e introduce el correo **ENTONCES** es enviado al correo indicado un formulario para restablecer la contraseña.
- **DADO** un usuario no registrado **CUANDO** selecciona la opción de recordar contraseña e introduce el correo **ENTONCES** se muestra un mensaje de error indicando al usuario que este correo electrónico no se encuentra en el sistema.

6.1.3. Resultados obtenidos

Tras la realización de la primera iteración y que la primera versión de la aplicación satisfaga todas las pruebas de aceptación definidas durante la reunión de planificación del *sprint* se puede comprobar en la figura 6.1 las pantallas finalizadas con respecto a la aplicación final obtenida. Este gráfico simplemente tiene como intención mostrar el avance del proyecto en respecto a las pantallas implementadas.

Además de esto, la cantidad de puntos de historia implementados es de 10, teniendo en cuenta que la suma total de puntos de historia es de 50 podemos concluir que durante la primera iteración tenemos completado alrededor del 20 % del sistema.

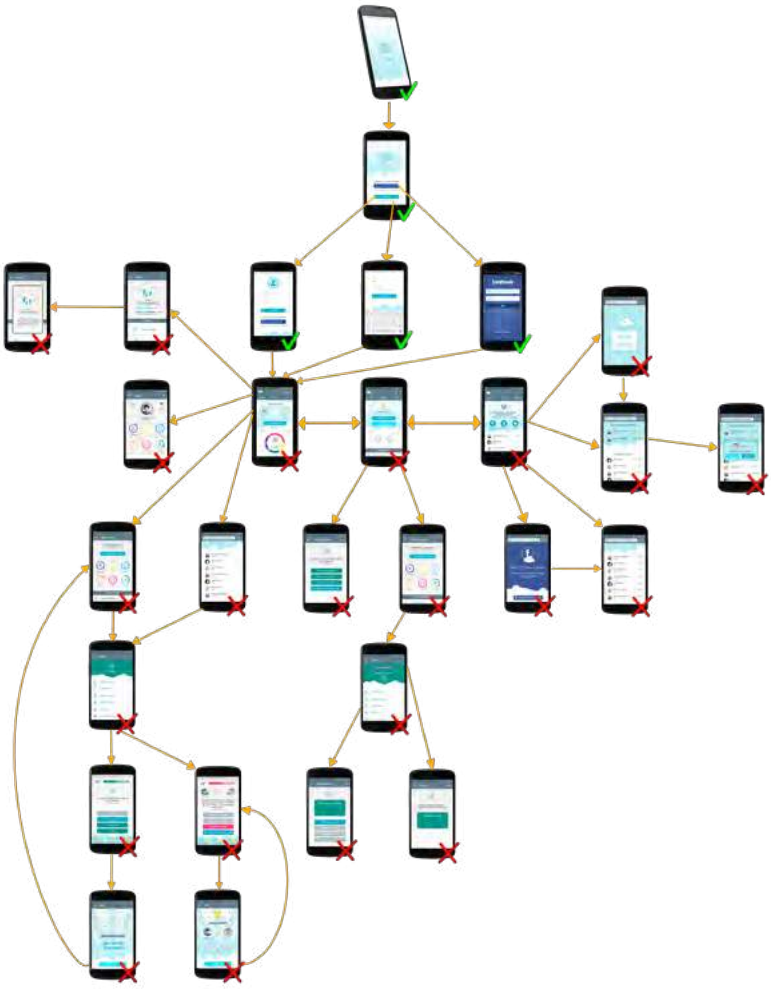

Figura 6.1: Jerarquía de pantallas completada Sprint 1.

Otro aspecto interesante a destacar es que hemos obtenido ya un producto funcional y entregándose al cliente una primera versión incompleta, aunque **funcional**.