



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

 etsinf

Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación móvil multiplataforma para la creación y resolución de nonogramas

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Ignacio Ferrer Sanz

Tutor: Germán Francisco Vidal Oriola

Curso 2020-2021

Resumen

WIP

Palabras clave: WIP

Resum

WIP

Paraules clau: WIP

Abstract

WIP

Key words: WIP

Índice general

Índice general	v
Índice de figuras	vii
Índice de tablas	vii
1 Introducción	1
1.1 Contexto y motivación	1
1.2 Objetivos	2
1.3 Metodología	2
1.3.1 Ciclo de vida de desarrollo	2
1.3.2 Modelo en casacada	3
1.4 Estructura de la memoria	4
2 Estudio estratégico	5
2.1 Nonogramas en la era Digital	5
2.1.1 Nonograms Katana	5
2.1.2 Nonogram.com - Picture cross number puzzle	7
2.1.3 Nono Infinite	8
2.1.4 Family Crest Nonogram	9
2.2 Análisis de las aplicaciones	10
2.3 Propuesta	11
3 Análisis del problema	13
3.1 Especificación de requisitos	13
3.1.1 Propósito	13
3.1.2 Ámbito	13
3.1.3 Terminología	13
3.1.4 Modelo de Dominio	14
3.1.5 Límites del Sistema	16
3.1.6 Restricciones del Sistema	16
3.1.7 Características del Sistema	17
4 Diseño de la solución	19
4.1 Tecnología empleada	19
4.1.1 Flutter y Dart	19
4.1.2 Firebase	20
4.1.3 Visual Studio Code	20
4.1.4 GitHub	21
4.2 Arquitectura del Sistema	21
4.2.1 Arquitectura por capas	21
4.2.2 Manejador de estados	23
Bibliografía	25

Índice de figuras

1.1	Esquema de un Modelo en cascada	3
2.1	Pantalla principal de Nonograms Katana	5
2.2	Pantallas de Nonograms Katana con modalidad a color.	6
2.3	Modal de restricción en publicación en Nonograms Katana	6
2.4	Pantallas de Nonogram.com	7
2.5	Pantalla de Evento Primavera de Nonogram.com	7
2.6	Pantallas de selección de nivel y juego de Nono Infinite	8
2.7	Pantallas de tutorial en Nono Infinite	9
2.8	Pantallas de Family Crest	9
3.1	Diagrama del modelo de dominio	14
3.2	Diagrama de contexto del aplicativo	16
3.3	Diagrama de casos de usos principales	17
3.4	Diagrama de casos de usos en pantalla de resolución de nivel	17
3.5	Diagrama de casos de ajustes	18
4.1	Diagrama de renderización de <i>Flutter</i> [11]	19
4.2	Diagrama del funcionamiento de <i>GitFlow</i>	21
4.3	Diagrama de <i>Arquitectura basada en cuatro capas</i>	22

Índice de tablas

2.1	Comparativa de características entre aplicaciones de interés y su inclusión como requisito	10
3.1	Glosario de términos ontológicos del aplicativo	14
3.2	Glosario de términos técnicos del aplicativo	14
3.3	Atributos de la clase Usuario	15
3.4	Atributos de la clase Nivel	15
3.5	Atributos de la clase Progreso	15
3.6	Atributos de la clase Nonograma	16
3.7	Restricciones del sistema	16
4.1	Funcionalidades <i>en nube</i> cubiertas por <i>Firebase</i>	20
4.2	Función de las capas de la arquitectura del sistema	22

CAPÍTULO 1

Introducción

Descubrir imágenes hechas píxel de situaciones del día a día, naturaleza, edificios famosos, personas, y cuantas cosas más, esta es la verdadera esencia de los nonogramas, también conocidos como hanzies, picross o griddlers.

1.1 Contexto y motivación

No importa lo complejo que sea resolver un nonograma, la clave de estos rompecabezas reside en que su resolución pueda efectuarse por simple lógica. [4] Este era el principal propósito de James Dalgety y su equipo de diseñadores, responsables de dar a conocer a occidente este conocido pasatiempo nipón, impulsado por el arte de la diseñadora Non Ishida, más adelante, responsable de su principal denominación: "Non" Ishida y Dia "gram".

No fue hasta mediados del año 1990, cuando finalmente se dio a conocer los *nonogramas* a escala mundial, a través de una publicación del periódico británico *The Sunday Telegraph*. Más adelante, el mismo noticiero adoptó el término bajo el seudónimo de *griddlers*, publicándolos semanalmente.

A partir de estas publicaciones, se fue difundiendo exponencialmente el famoso puzzle y se puede encontrar en revistas, otros periódicos y libros. Fue tan notable su crecimiento que, como otros rompecabezas, alcanzó con prontitud el formato digital, en forma de sitios web, videojuegos y aplicaciones.

La capacidad creativa que ofrece resulta ilimitada, ya que con tan solo sus celdas dispuestas en forma de matriz (*filas y columnas*), permite representar todo tipo de figuras, siluetas y formas, como si de un lienzo se tratara.

Sorprendentemente y a pesar de que nos encontramos en plena era digital, son pocos los medios que ofrecen una capacidad de creación, más allá del simple tradicional método del lápiz y papel y así esta herramienta ayuda y otorga al jugador no solo el rol de *resolutor* sino de *creador*, e impulsa que la cuantía de *nonogramas* a resolver no ceda.

Un medio digital es ideal para, no solo hacer que esta propiedad de creación sea posible, sino de facilitar su proceso y que sea lo más liviano y recreativo posible, de esta forma las aplicaciones móviles, cada vez más conocidas y presentes en nuestra sociedad, constituyen el entorno perfecto.

Siguiendo esta premisa, lo que sería adecuado es que cualquier usuario pudiera, dentro de lo posible, emplear estos medios independientemente de cual sean las características técnicas, sistemas operativos y prestaciones de sus dispositivos.

1.2 Objetivos

El presente trabajo explora el submundo de las aplicaciones móviles y propone una solución software para: i) permitir al usuario de resolver *nonogramas* de forma interactiva ii) dar la oportunidad de crear sus propios puzzles como lo hizo *James Dalgety* y su equipo y compartirlos con todos los demás usuarios, promoviendo así una comunidad de entusiastas de este divertido rompecabezas.

Por consiguiente, para el correcto desarrollo y funcionamiento de la aplicación se deben de cumplir una serie de requerimientos a nivel técnico bien diferenciados:

- Estudiar y desarrollar un algoritmo integrado en el aplicativo, con el fin de posibilitar al usuario crear y resolver *nonogramas* en un dispositivo móvil, bajo unas variables determinadas, siempre dando prioridad a la experiencia de juego.
- Implementar un *backend* con el que el aplicativo pueda complementar sus funcionalidades con *servicios en nube*, tales como la base de datos, sincronización o inicios de sesión.
- Seguir los principios de *Clean Arquitecture* durante el proceso de desarrollo, implementando patrones de diseño y aplicando *suites de test* con el objetivo de aminorar el proceso de mantenimiento.
- Definir y realizar un MVP (*Mininum Viable Product*), con el que usuario pueda, en una primera versión, hacer uso de sus funciones principales.

Finalmente, encontramos requisitos de índole personal, que progresivamente se considerarán como cumplidos durante todo el desarrollo del proyecto, tales como:

- Ahondar en el desarrollo de una aplicación móvil, desde su inicio hasta su finalización, bebiendo de buenas prácticas y recomendaciones propuestas por artículos, documentaciones y comunidades de desarrolladores.
- Comprender y profundizar en el extenso mundo de desarrollo de juegos de puzzles, haciendo frente y reduciendo su marcada complejidad.

1.3 Metodología

El aplicativo resultante, como cualquier otra solución software, debe satisfacer una serie de requerimientos frente un problema o problemas concretos. Esta solución podría residir o enfocarse en diferentes plataformas, bajo una perspectiva tanto de *software* como de *hardware*.

Así mismo, antes de que esté considerada preparada la aplicación para su uso, ha de atravesar por una serie de procesos, que difieren mucho de un simple desarrollo y en su conjunto reciben el nombre de *System Development Life Cycle (SDLC)*, que corresponden a su ciclo de vida.

1.3.1. Ciclo de vida de desarrollo

Comprender esta serie de procesos sucesivos, junto a los requisitos recién comentados, es definitorio para elegir una metodología ideal, que sirva como guía para el correcto desarrollo total del producto.

- **Planificación:** identificar los requisitos necesarios para la aplicación, considerando y comparando otras soluciones disponibles, para así establecer un *target* o perfil de usuario ideal para nuestra aplicación, sin entrar en el apartado técnico.
- **Análisis:** establecer los requisitos funcionales de la aplicación, recopilando y anticipándose a aquellos que puedan suponer un problema para la evolución del ciclo de vida.
- **Diseño:** documentar las, ya definitivas, características, partes y componentes a integrar en el aplicativo.
- **Codificación:** seguir los requisitos ya debidamente documentados e implementarlos creando así el aplicativo.
- **Testing:** una vez desarrollada la solución, realizar *suites de tests* con el fin de encontrar o mostrar posibles errores y *bugs*. Además de verificar que los requisitos de la fase de diseño están presentes en la solución.
- **Implementación:** desplegar una primera versión de la aplicación y hacerla disponible en las tiendas principales de aplicaciones.
- **Mantenimiento:** monitorizar la experiencia de usuario, contemplando y dando solución a posibles errores, además de realizar cambios y mejoras en forma de nuevas versiones.

Puesto que en el presente trabajo, durante la primera fase de Planificación, los requisitos han sido detectados y establecidos bajo unos tiempos y conocimientos concretos, se ha optado por el modelo clásico de *Modelo en Cascada*.

1.3.2. Modelo en casacada

Siguiendo esta metodología, como se muestra en la Figura 1.1, se implantaría un modelo secuencial de todas las diferentes fases del *SDLC* del aplicativo, desde su fase de Planificación hasta su Mantenimiento, como si se tratara de una cascada.

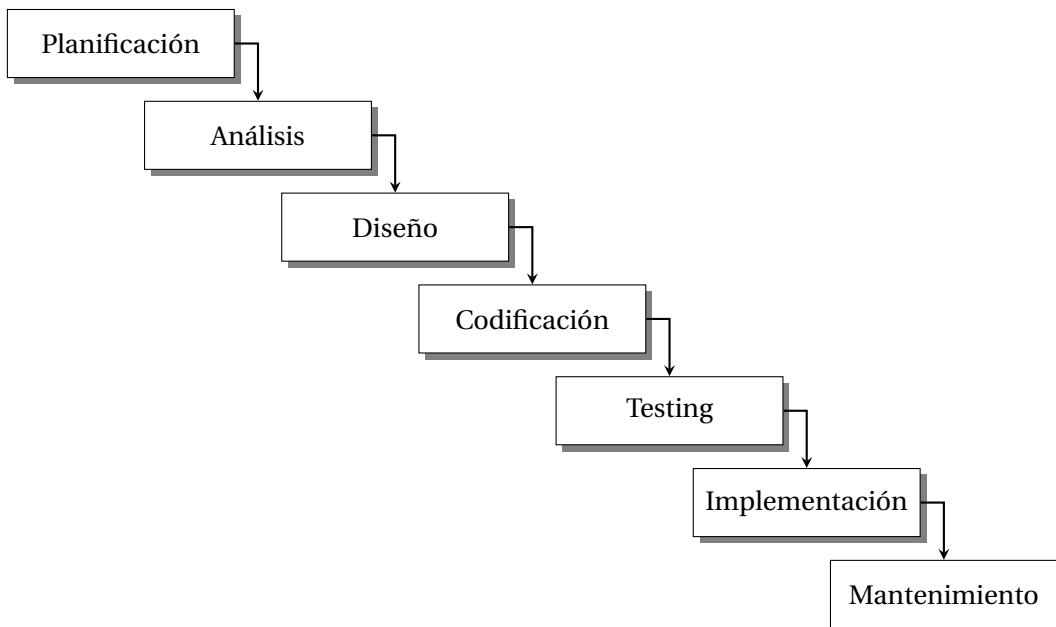


Figura 1.1: Esquema de un Modelo en cascada

Sin embargo, como cualquier otra metodología software, es necesario tener en cuenta ciertas consideraciones imprescindibles para el transcurso del proyecto:

- Como ya se había comentado, es necesario hacer hincapié en las etapas tempranas del modelo, ya que son las que van a definir de manera correcta los requisitos de la solución. De forma que, si no están bien establecidos, pueden aparecer problemas en el resto de fases [14].
- Esta metodología carece de iteraciones por lo que no es posible retroceder a las etapas anteriores. Hay que cerciorarse de que cada una de las fases se aprueban correctamente, de forma que sea seguro pasar a la fase siguiente.
- En la práctica, es recomendable marcar bien los tiempos de cada fase, marcando estimaciones de cada fase y contrastarlas con el tiempo real que ha supuesto realizarlas. Esta especie de monitorización se realizará periódicamente y se puede encontrar en el ??.

1.4 Estructura de la memoria

El resto de capítulos que conforman la memoria, son los que se resumen a continuación:

- **Capítulo 2. Estudio estratégico:** corresponde al conocido apartado *estado del arte*, en el que se realiza una labor de estudio de aquellas soluciones relacionadas con *nonogramas* dentro del mundo digital y que puedan ayudar a la identificación y extracción de requisitos.
- **Capítulo 3. Análisis del problema:** en este capítulo se expone la parte de especificación de requisitos, identificando aquellos que puedan repercutir negativamente al transcurso del proyecto.
- **Capítulo 4. Diseño de la solución:** se muestran las decisiones que se han tomado a nivel de arquitectura, patrones de diseño, y tecnologías empleadas.
- **Capítulo 5. Desarrollo de la solución:** se comentan las distintas partes que han compuesto el aplicativo, cómo se comportan y el funcionamiento interno de cada una de ellas, entrando en el apartado técnico.
- **Capítulo 6. Pruebas:** se muestran el conjunto de pruebas, que se han desarrollado para la posible identificación de errores y posibles fallos en su ejecución, todas ellas divididas en tipos.
- **Capítulo 7. Implementación y mantenimiento:** se explica el paso que se ha realizado para hacer que el aplicativo sea accesible para los usuarios y las medidas para su mantenimiento.
- **Capítulo 8. Manual de uso:** presenta una pequeña demo con el fin de que el usuario se familiarice con el uso de la solución, mostrando capturas del mismo.
- **Capítulo 9. Conclusión y Trabajo Futuro:** contempla las conclusiones que se han obtenido durante la realización del trabajo y las mejoras que se tomarán para siguientes versiones del mismo.
- **Apéndices:** se muestran anexos relacionados con análisis de tiempos y apartados relacionados con el de Codificación del producto.

CAPÍTULO 2

Estudio estratégico

Los nonogramas junto otros gigantes rompecabezas de «papel y lápiz» tales como: sudoku, crucigramas, hundir la flota, ahorcado... se han adaptado a una era en la que está gobernada por las nuevas plataformas tecnológicas. Y más concretamente, en estos tiempos de incertidumbre, confinamientos e incluso ocio han propiciado que estos puzzles se hagan cada más presentes en nuestras vidas, alejándose una posible obsolescencia.

2.1 Nonogramas en la era Digital

Pese a que la era Digital ofrece un amplio abanico de medios o plataformas que han incluido y popularizado estos rompecabezas, en este apartado nos enfocaremos en el de las aplicaciones móviles.

Para que el estudio sea exhaustivo, se explorarán aquellas aplicaciones disponibles en las principales tiendas de aplicaciones para las plataformas *Android* e *iOS*, *Google Play* y *App Store* respectivamente.

2.1.1. Nonograms Katana

Nonograms Katana es una aplicación con una remarcada temática nipona, que permite al usuario resolver una gran variedad de *nonogramas*, de una gran variedad de categorías y dimensiones, como se puede comprobar en la Figura 2.2a.

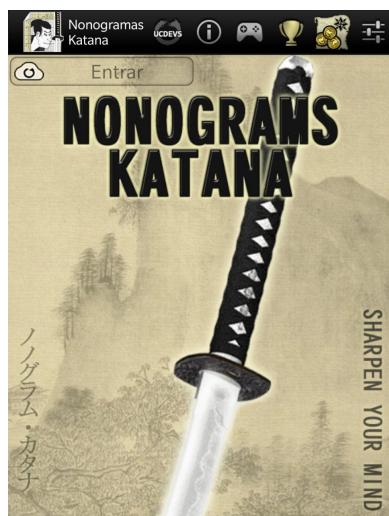


Figura 2.1: Pantalla principal de Nonograms Katana

Además, como se puede apreciar en la Figura 2.2b, se incluye la resolución de *nonogramas* a color, en el que mediante un selector de colores el usuario pinta cada una de las celdas, resolviendo de este modo el *nonograma*.

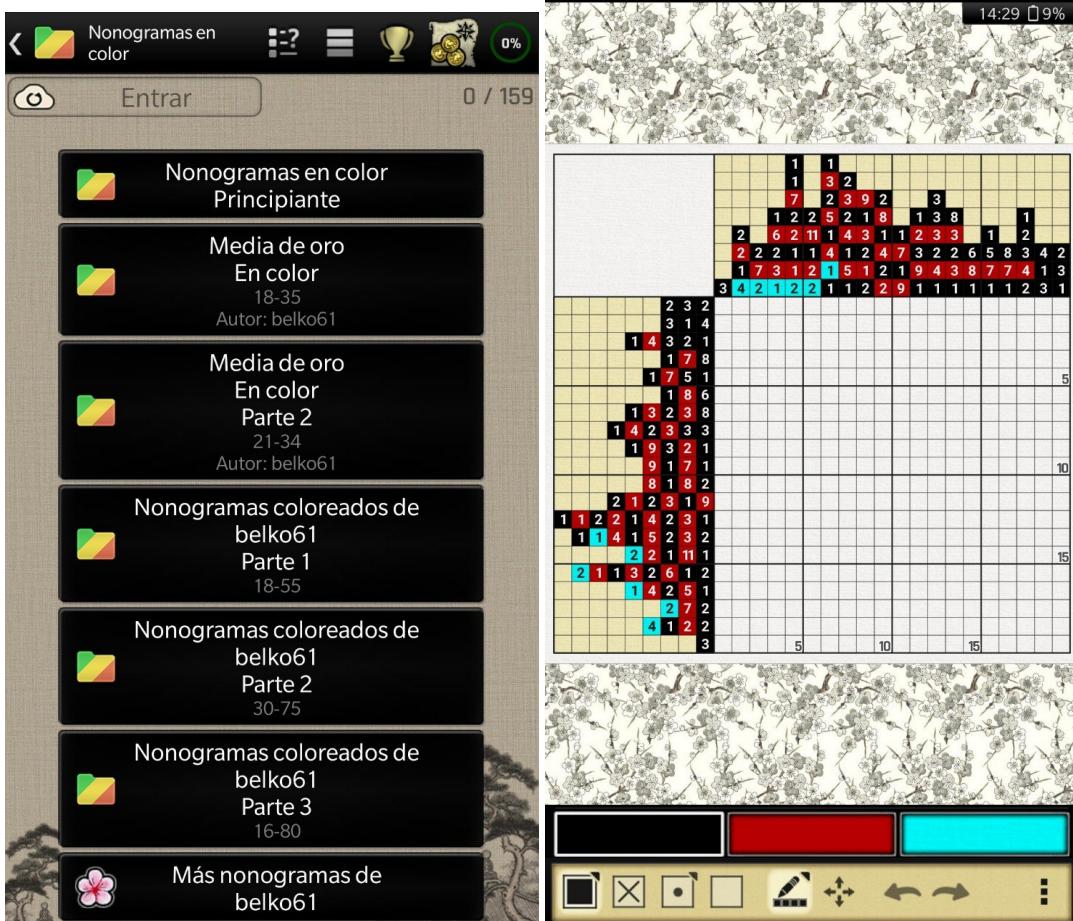


Figura 2.2: Pantallas de Nonograms Katana con modalidad a color.

El aplicativo sigue la corriente clásica y no permite al usuario resolver los *nonogramas* basado en un número determinado de *vidas* o intentos (*disminuyendo su valor al pulsar sobre celdas erróneas*), siendo algo tediosa la experiencia de juego, ya que puede acarrear fallos durante su resolución.

Una característica notable de la aplicación es la de permitir al usuario crear sus propios *nonogramas*, compartirlos con la comunidad, y resolver los de otros usuarios. Sin embargo, esta propiedad no es su principal función y aparece bloqueada si no estás registrado, además de estar limitada por restricciones como los de la Figura 2.3.

El aplicativo presenta la propiedad de multilenguaje, no obstante, este presenta errores en sus traducciones, como se puede comprobar en la figura anteriormente citada.



Figura 2.3: Modal de restricción en publicación en Nonograms Katana

2.1.2. Nonogram.com - Picture cross number puzzle

Una de las aplicaciones más descargadas dentro de la categoría puzzle con más de diez millones de descargas en la tienda *Google Play*, presenta una interfaz amigable y limpia.

Hace un buen uso de animaciones, destacando la experiencia de juego, algunas bastante remarcables como cuando la que se muestra en la Figura 2.4b, en cuanto finalizas un nivel.

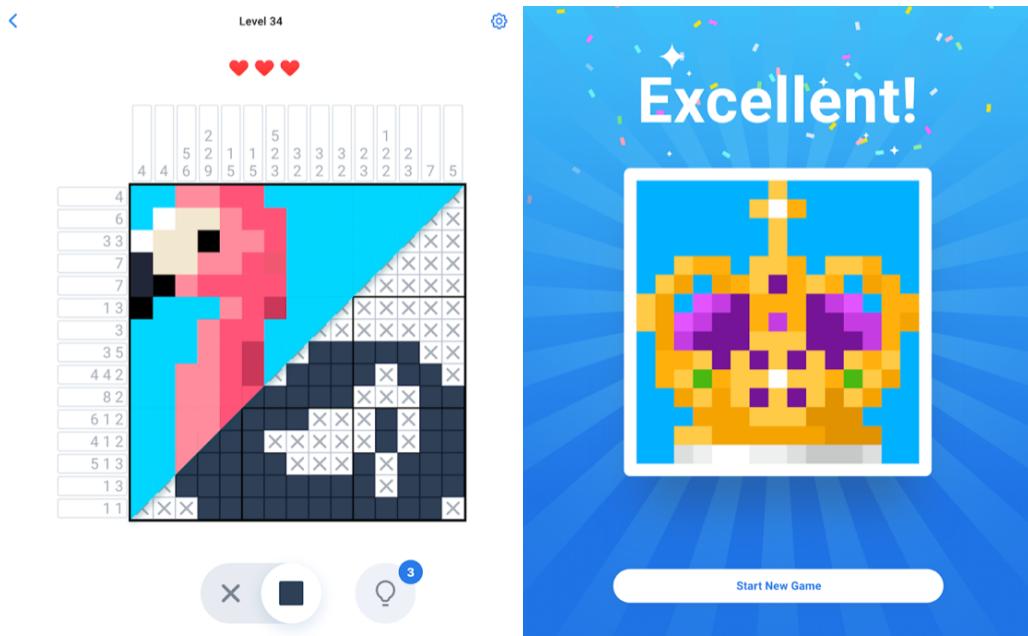


Figura 2.4: Pantallas de Nonogram.com

Como característica extra de juego, como se visualiza en la Figura 2.4a, la barra inferior de juego incorpora un botón llamado *Pista*, con el que después de ser seleccionado, el usuario puede clicar sobre una celda determinada y descubrir si es correcta sin restar una vida, teniendo limitada esta opción a tres usos por nivel.

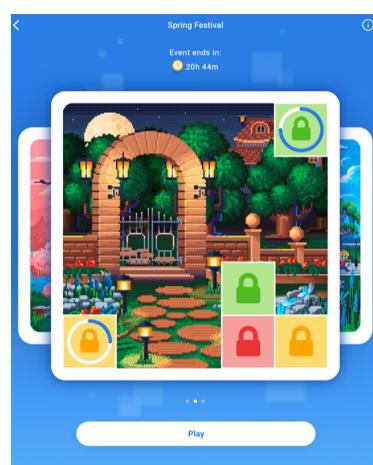


Figura 2.5: Pantalla de Evento Primavera de Nonogram.com

En el aplicativo, de forma recurrente, aparecen *Eventos* en los que el usuario puede resolver un conjunto de *nonogramas* especiales relacionados con una temática concreta, representada en la Figura 2.5.

La solución, como se ha podido comprobar, es de las más completas de las disponibles, no obstante, incluye publicidad excesivamente intrusiva para el usuario, presente en casi todas sus funcionalidades, que entorpecen la experiencia de juego, incluso llegando a entorpecer al jugador.

2.1.3. Nono Infinite

Este producto destaca por su interfaz *arcade*, con una clara intención de ser dirigida para todos los públicos, constraintando colores muy vivos, con formas que recuerdan mucho a juegos clásicos para niños, se puede ver reflejado en las Figuras 2.6.



(a) Pantalla selector de nivel

(b) Pantalla de resolución de un nivel 5x10

Figura 2.6: Pantallas de selección de nivel y juego de Nono Infinite

Como peculiaridad, presenta *nonogramas* de dimensiones poco usuales que difieren mucho de los clásicos, como el 5x10 presente en la Figura 2.6b. Así mismo, permite cambiar la dificultad de los niveles manteniendo las dimensiones del mismo, sin embargo, esta puede parecer un poco "artificial" ya que únicamente aumenta las distancias de las celdas correctas.

Resulta interesante cómo la solución aprovecha el apartado del tutorial, para habituar al usuario de forma interactiva con las reglas clásicas de los *nonogramas*, controles del mismo y consejos más avanzados, sobretodo para niveles más complejos, Figura 2.7.

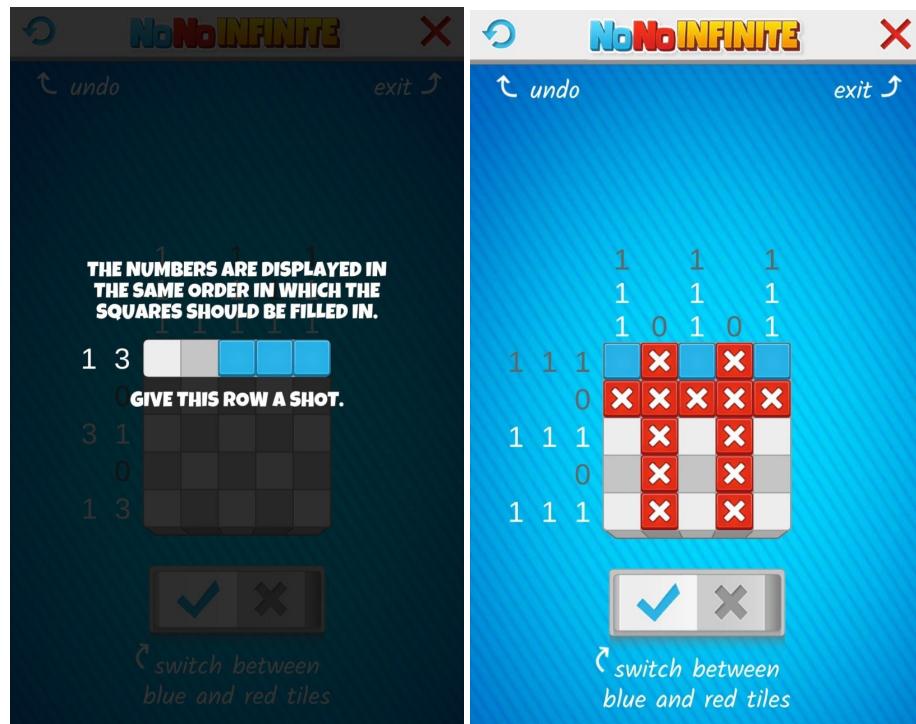


Figura 2.7: Pantallas de tutorial en Nono Infinite

2.1.4. Family Crest Nonogram

Family Crest Nonogram es un juego que, pese a que no destaque por su apartado gráfico e interfaz de usuario, presenta muchas características que lo diferencian del resto de soluciones del género.

Lo más remarcable de la aplicación es que tenga establecido su uso en modo *landscape* (apaisado), adaptándose al formato de la mayoría de géneros de juegos de móvil. Sin embargo, este puede ser un punto negativo para muchos, ya que priva al jugador la esencia de estar jugando a un pasatiempo, recordándolo más a la de un *videojuego*.



(a) Pantalla principal

(b) Pantalla de resolución de un nivel 20x20

Figura 2.8: Pantallas de Family Crest

No obstante, su experiencia de juego es muy notable, ya que presenta una *cruceta* (para moverse por cada una de las celdas) y botones físicos de acción que facilitan un mayor control, Figura 2.8b, además de incorporar una música ambiente que acompaña al usuario cada vez que está jugando un nivel.

2.2 Análisis de las aplicaciones

Al analizar las aplicaciones enfocadas al submundo de los *nonogramas*, vemos que siguen un patrón de similitudes, que resultan sustanciales para el desarrollo de una solución del género rompecabezas, además de ciertas peculiaridades que parecen interesantes para su inclusión en la solución. Es importante también remarcar y apartar aquellas características que puedan ser perjudiciales para una primera versión de la solución. A continuación, se muestra un tabla representativa de cada una de las características estudiadas, donde se objetarán su final inclusión en el aplicativo:

Tabla 2.1: Comparativa de características entre aplicaciones de interés y su inclusión como requisito

Característica encontrada	Nonogram Katana	Nonograma.com Picture cross	Nono Infinite	Family Crest	Incluido como requisito
Aplicación multiplataforma	✓	✓	✗	✓	✓
Temática especial	✓	✗	✓	✓	✓
Creación nonogramas	✓*	✗	✗	✗	✓
Opción de autoguardado	✓	✓	✗	✗	✓*
Registro de usuario	✓	✗	✗	✗	✗
Sincronización de niveles en nube	✓	✗	✗	✗	✓
Juego con <i>vidas</i>	✗	✓	✗	✗	✓*
Música ambiente	✗	✗	✗	✓	✗
Botón de pistas	✗	✓	✗	✗	✗
Botones físicos de acción	✓	✓	✓	✓	✗
Variedad de visuales	✓	✗	✗	✗	✓
Multi-idioma	✓*	✓	✗	✗	✓
Eventos especiales	✗	✓	✗	✗	✗
Nonogramas a color	✓	✗	✗	✗	✗
Tutorial de juego	✓	✓	✓	✗	✓
Sección Leaderboard	✓	✓	✗	✗	✗
Selector de niveles	✓	✗	✗	✓	✓

La inclusión de las características de la Tabla 2.1 se ven indicadas mediante la siguiente simbología:

1. ✓: Característica incluida en el aplicativo.
2. ✗: Característica no incluida en el aplicativo.
3. ✓*: Característica incluida en el aplicativo con ciertos matices.

2.3 Propuesta

Visualizando las características reflejadas en la Tabla 2.1 sacamos la siguientes conclusiones de cara a la creación de la primera versión de la aplicación como producto:

- Ya que la mayoría de apps estudiadas son multiplataforma, el aplicativo funcionará para ambas plataformas *Android* e *iOS*.
- Seguirá una temática especial con el fin de diferenciarlas del resto, además de ir alternando entre temas diversos cambiando la interfaz de usuario.
- Tanto la resolución como la creación de los nonogramas será una de las funciones de la solución, sin ningún tipo de restricción.
- El usuario podrá usar *servicios in-cloud* como: sincronización en nube o acceso a la base de datos sin tener que registrarse.
- La aplicación tendrá posibilidad multi-idioma, teniendo disponible inglés y español para esta primera versión, sin ningún tipo de errata.
- Se podrán jugar niveles de *nonogramas* de diferentes dimensiones clásicas, recordando a los de los medios físicos tradicionales, previamente explicado su uso por un tutorial intuitivo.
- Durante la resolución del nonograma, el usuario podrá resolverlo con una interfaz minimalista, en ausencia de botones físicos, además de poder alternar entre resolución con y sin *vidas*.
- Muchas de las características estudiadas se han tomado como no esenciales y quedarán descartadas para la primera versión.

CAPÍTULO 3

Análisis del problema

La Ingeniería de Requisitos (IR) es el área más importante de la Ingeniería de Software y posiblemente de todo el ciclo de vida de una solución software (SDLC) [3]. Esta etapa es la responsable de que los requisitos recién detectados, aún incompletos e imprecisos, se transformen en especificaciones formales del aplicativo final.

3.1 Especificación de requisitos

Para llevar a cabo la especificación total de requisitos se seguirá la norma tradicional establecida por el estándar internacional IEEE Std 830-1998 [7], elegido por una gran mayoría de jefes de departamentos software por su gran agilidad en la fase de gestión de requisitos [5].

A continuación, de acuerdo a la normativa ISO elegida, se mostrarán los contenidos acompañados por diagramas y buenas prácticas.

3.1.1. Propósito

El propósito de esta sección es la de definir y formalizar los requerimientos que debe incorporar el *MVP* del aplicativo, facilitando y guiando el desarrollo del mismo.

3.1.2. Ámbito

El ámbito, como se ha comentado en capítulos anteriores, es el de las aplicaciones móviles disponibles en plataformas *iOS* y *Android*.

El aplicativo en su versión *MVP* adoptará el nombre provisional de *NonoChallenge*, compuesto por el juego de palabras: *Nonograma* junto con el término anglosajón *Challenge* (reto). En el cual, el usuario hará uso de sus servicios propios y *en nube* tales como inicios de sesión, interacción con base de datos y sincronización.

3.1.3. Terminología

Los términos relacionados con la ontología del sistema se ven enumerados y descritos por la Tabla 3.1.

Tabla 3.1: Glosario de términos ontológicos del aplicativo

Término	Descripción
Usuario	Persona que hará el uso del conjunto de funcionalidades del aplicativo final
Nonograma	Rompecabezas de MxN dimensiones
Nivel	Nonograma a crear o resolver por el Usuario
Progreso	Datos relacionados con la persistencia de la resolución de un nivel

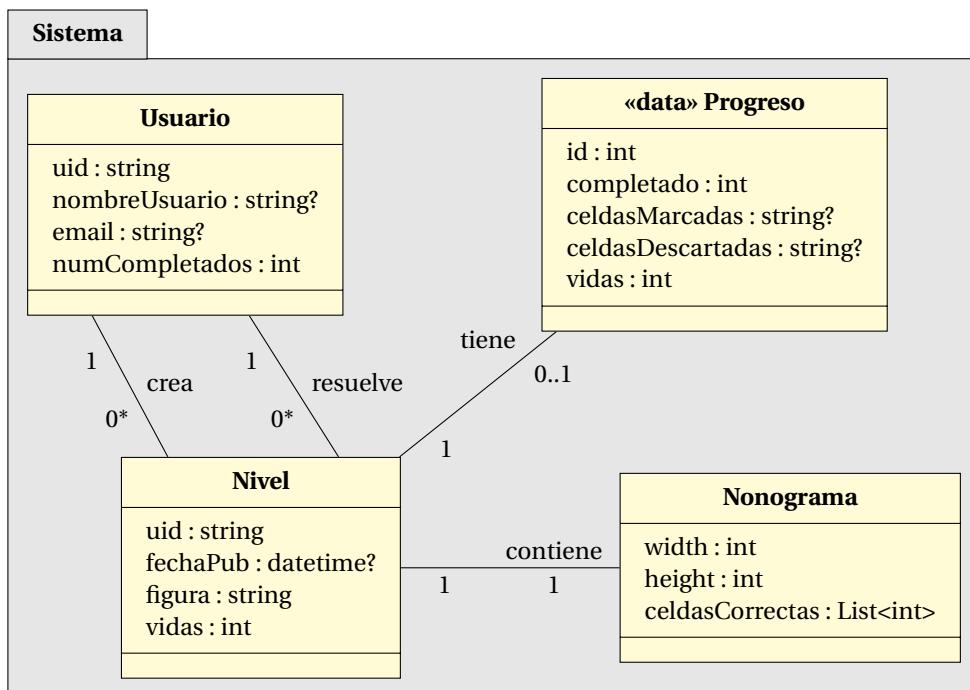
Por otra parte, los términos técnicos que componen el sistema son los que siguen:

Tabla 3.2: Glosario de términos técnicos del aplicativo

Término	Descripción
Nombre de usuario	Nombre que adoptará el Usuario de forma opcional para su identificación en el aplicativo.
Email	Cuenta de correo que hará uso el usuario para acceder a los servicios <i>en nube</i>
Fecha de publicación	Fecha en la que el usuario crea y publica un nivel
Figura	Nombre identificativo de un nonograma
Vidas	Número de intentos en la resolución de un nivel

3.1.4. Modelo de Dominio

Una vez introducida la terminología propia del sistema, para un mayor entendimiento del contexto del mismo, se representa un diagrama de clases de acuerdo a las reglas clásicas UML, como se puede visualizar en la Figura 3.1.

**Figura 3.1:** Diagrama del modelo de dominio

A continuación, se comenta la función de cada una de las clases conceptuales y se enumeran en las Tablas 3.3- 3.6 los atributos que las componen. (*La anotación ? al lado del tipo refleja su posibilidad de nulidad en el sistema.*)

- **Clase Usuario:** Conjunto de datos del usuario relacionados con el sistema.

Tabla 3.3: Atributos de la clase Usuario

Atributos de Usuario	Descripción
uid <i>string</i>	Cadena de caracteres único que identifica al usuario a nivel interno
nombre de Usuario <i>string?</i>	Pseudónimo único a elegir por el usuario en el sistema
email <i>string?</i>	Cuenta de correo de registro única para el usuario
numCompletados <i>int</i>	Número de niveles completados por el usuario

- **Clase Nivel:** Muestra las características de un determinado nivel a resolver.

Tabla 3.4: Atributos de la clase Nivel

Atributos de Nivel	Descripción
uid <i>string</i>	Cadena de caracteres único que identifica al nivel
fecha de Publicación <i>dateTime?</i>	Fecha de creación del nonograma en caso de llegar a publicarse
figura <i>string</i>	Nombre del nivel mostrado una vez resuelto
vidas <i>int</i>	Número de intentos que dispone un usuario para la resolución de un nivel

- **Clase Progreso:** Clase de persistencia que contiene los datos durante la resolución de un nivel.

Tabla 3.5: Atributos de la clase Progreso

Atributos de Progreso	Descripción
id <i>int</i>	Número de identificación del progreso de un determinado nivel
completado <i>int</i>	Indica con un 1 si el nivel está resuelto y 0 si no lo está (simulando un dato de tipo booleano)
celdasMarcadas <i>string</i>	Números de las celdas pintadas separados por delimitadores (simulando un dato de tipo lista de enteros)
celdasDescartadas <i>string</i>	Números de las celdas descartadas separados por delimitadores (simulando un dato de tipo lista de enteros)
vidas <i>int</i>	Números de vidas que dispone el usuario en ese momento

- **Clase Nonograma:** Contiene los datos del nonograma de un determinado nivel.

Tabla 3.6: Atributos de la clase Nonograma

Atributos de Nonograma	Descripción
width <i>int</i>	Número de filas de celdas que compone el Nonograma
height <i>int</i>	Número de columnas de celdas que compone el Nonograma
celdasCorrectas <i>List<int></i>	Lista de los números de celdas que componen la solución del Nonograma

3.1.5. Límites del Sistema

Para representar los límites de la aplicación, se emplea un diagrama de contexto en el que se muestra la jerarquía de los actores que van a interactúan con el aplicativo.

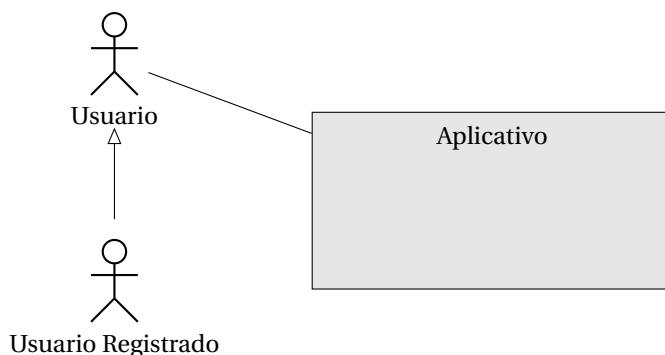


Figura 3.2: Diagrama de contexto del aplicativo

Como se puede apreciar en el diagrama de la Figura 3.2, únicamente van a interacutuar dos actores con el sistema: i) el usuario sin registro, que emplea únicamente las funciones locales del sistema y ii) el usuario registrado, que podrá emplear tanto las funciones locales como *en nube* del aplicativo.

3.1.6. Restricciones del Sistema

En este apartado se muestran algunas de las restricciones que pueden impedir algunas de las funcionalidades que ofrece el sistema al usuario, vistos en la Tabla 3.7.

Tabla 3.7: Restricciones del sistema

Restricción	Descripción
Memoria del dispositivo	Posibilidad de que el dispositivo no tenga suficiente memoria disponible para guardar el progreso de la resolución de los niveles. <i>Probabilidad prácticamente despreciable</i>
Conexión a internet	Las funcionalidades <i>en nube</i> requieren de conexión a internet para poder llegar a usarse.
Cuenta de Google	El usuario puede carecer de una cuenta de correo de Google necesaria para hacer uso de las funcionalidades <i>en nube</i> .

3.1.7. Características del Sistema

Una vez vistas las limitaciones y restricciones del sistema, se pueden analizar las características finales a incorporar en la aplicación, mediante *casos de uso*.

3.1.7.1. Casos de uso principales

Estos, son los que se encontrarán en la *Pantalla Principal* y *Pantalla de Selección de Nonogramas* del aplicativo.

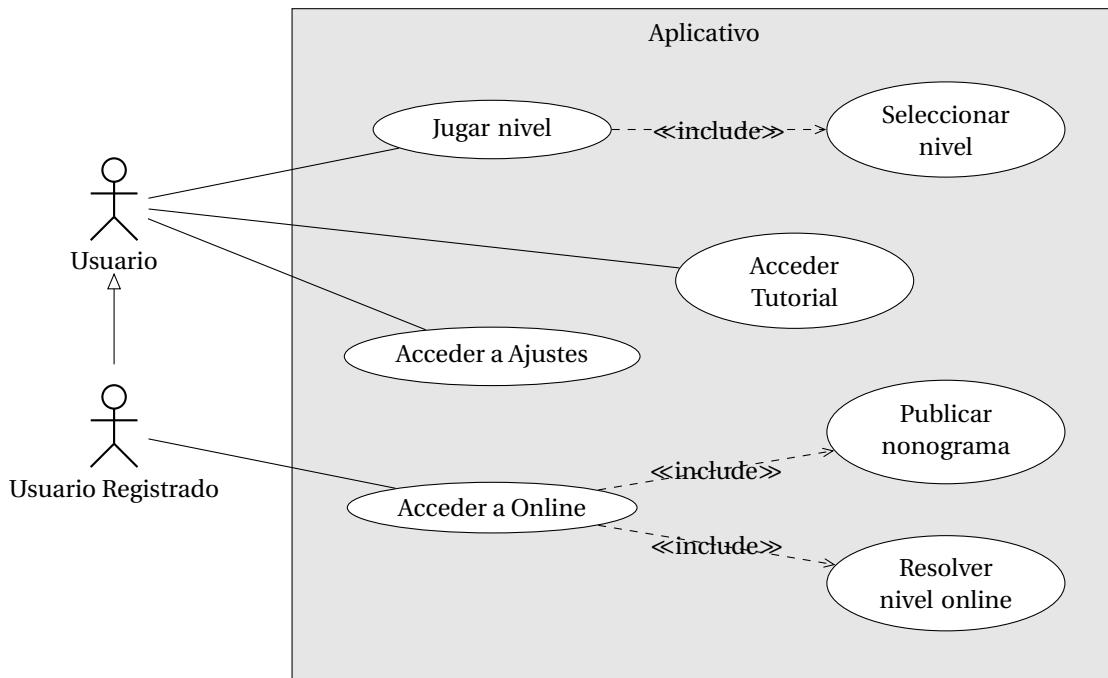


Figura 3.3: Diagrama de casos de usos principales

3.1.7.2. Casos de uso de resolución de nivel

Estarán disponibles en la pantalla de resolución de nivel.

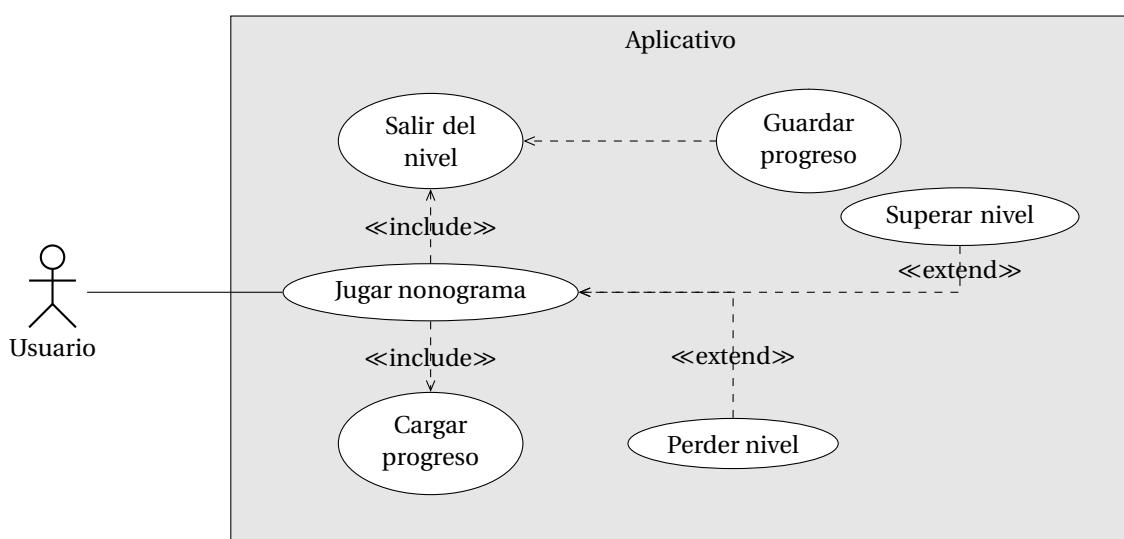


Figura 3.4: Diagrama de casos de usos en pantalla de resolución de nivel

3.1.7.3. Casos de uso de ajustes

Casos de uso visibles en la *Pantalla de Ajustes* del sistema.

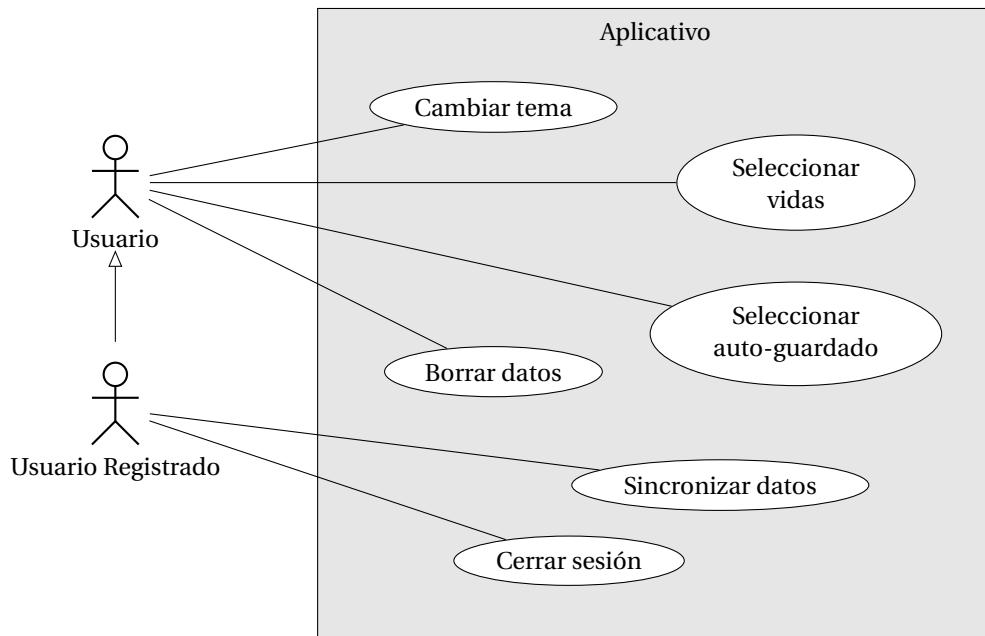


Figura 3.5: Diagrama de casos de ajustes

CAPÍTULO 4

Diseño de la solución

El desarrollo de una aplicación multiplataforma es una opción cada vez más a tener en cuenta dentro del ámbito del desarrollo móvil [1]. La mayor parte de desarrollos se van a querer destinar a las principales plataformas móviles, con el fin de llegar al máximo número de usuarios posibles.

4.1 Tecnología empleada

Como se había decidido en el segundo capítulo, la realización de este *MVP* irá dirigido para las plataformas móviles: *Android* e *iOS*. El desarrollo de un aplicación *nativa* para cada sistema, evidentemente, no sería factible, ya que implicaría el doble de esfuerzo en la codificación, testing y mantenimiento del producto [6].

Por ello, es necesario el uso de un *framework* de ámbito multiplataforma que nos proporcione un soporte para ambas plataformas. En este caso, se ha optado por la prometedora herramienta de desarrollo impulsada por *Google*: *Flutter*.

4.1.1. Flutter y Dart

Flutter junto a su lenguaje de programación principal, *Dart*, buscan aproximar al usuario la apariencia y rendimiento propio al de un desarrollo *nativo*, aprovechando todas las ventajas que proporciona un *framework multiplataforma* [10].

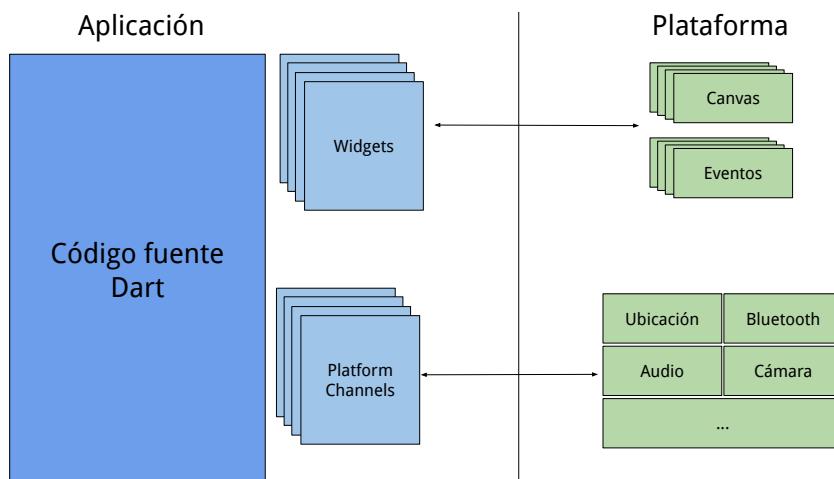


Figura 4.1: Diagrama de renderización de *Flutter*[11]

Esta relación *apariencia/rendimiento*, la consigue *Flutter* de forma exitosa *renderizando* directamente sus componentes, *widgets*, en un *canvas*, permitiendo visualizar el mismo componente en las diferentes plataformas. Del mismo modo, realiza la conversión directa de servicios propios, *Platform Channels*, en librerías nativas que interactúan con las funcionalidades propias del dispositivo [11], tal y como se puede visualizar en la Figura 4.1.

Esta es una de las razones principales por las que se ha elegido el uso de este herramienta frente a otros *frameworks* de desarrollo móvil como:

- Los *frameworks interpretados* como: *React Native* y *Vue Native*, que necesitan de una capa adicional para realizar la conversión a *componentes nativos*, afectando al rendimiento.
- Los *frameworks embebidos* como *Ionic* que emplea un *WebView* para mostrar los componentes dependiendo de la conexión a Internet del dispositivo.

4.1.2. Firebase

Para todas las funcionalidades relacionadas con servicios *en nube*, vistos en la fase de Análisis, se empleará la plataforma de desarrollo backend impulsada y recomendada por *Google*: *Firebase*.

Gracias a que *Firebase* presenta una base de datos del tipo *real-time*, permitirá la sincronización de los datos durante la ejecución del aplicativo [8]. Esta característica resulta ideal, ya que *Flutter* al tratarse de un *framework declarativo* no requiere de eventos para sincronizar los datos, permitiendo al usuario visualizar los datos, en todo momento, sincronizados en pantalla, sin necesidad de que el usuario realice un *input de refrescar*.

Además, esta, al pertenecer al tipo de las *no relacionales*, presenta ciertos beneficios frente a las *relacionales*, a tener en cuenta en la definición de los modelos, tales como flexibilidad, seguridad y rendimiento.

La funcionalidades *in-cloud* del *back-end* de la aplicación final quedarán cubiertas por los apartados mostrados en la Tabla 4.1

Tabla 4.1: Funcionalidades *en nube* cubiertas por *Firebase*

Funcionalidad	Tecnología
Autentificación	Firebase Authentication , permite al usuario el acceso a las funciones de la plataforma mediante inicio sesión tradicional (cuenta de correo y contraseña) o por RRSS.
Base de datos	Firebase Firestore , permite al usuario interactuar mediante funciones CRUD la base de datos del sistema.
Analíticas	Firebase Analytics , monitoriza la experiencia de usuario y extrae estadísticas de cara al lanzamiento de futuras versiones tales como: porcentaje de usuarios libre de errores, número de usuarios activos...

4.1.3. Visual Studio Code

Como *IDE* de desarrollo se empleará la herramienta *Visual Studio Code*, desarrollado por *Microsoft*, frente a otros entornos como: *XCode* y *Android Studio*, ya que este dispone de una gran cantidad de *extensiones o plugins* que facilitan notablemente el proceso de *codificación*, además de estar mejor optimizado para labores de *compilación y depuración*.

4.1.4. GitHub

Se empleará *GitHub* para el *control de versiones*, siguiendo la metodología propias de *GitFlow*, en el que tendremos, como se puede visualizar en la Figura 4.2: i) una rama *master* para versiones finales del aplicativo, ii) una rama *develop* en la que aunar y testar las funcionalidades desarrolladas y iii) las ramas *feature* que se crearán por cada caso de uso a desarrollar.

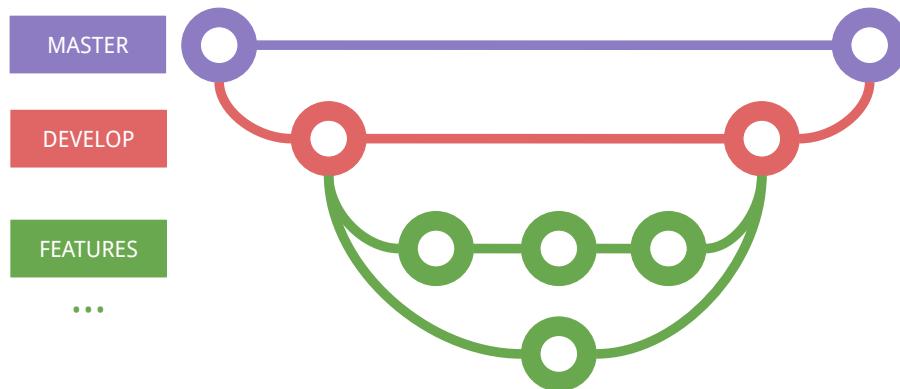


Figura 4.2: Diagrama del funcionamiento de *GitFlow*

Además, nos nutriremos de las buenas prácticas, *librerías* y *repositorios* disponibles en la plataforma, aprovechando la subida exponencial de la comunidad *Flutter* en este último año, posicionándose dentro de las tecnologías con más *stars* en el *sitio web* [12].

4.2 Arquitectura del Sistema

Una vez definidas las tecnologías que se van a utilizar para el desarrollo de la solución, es importante definir una arquitectura que encaje dentro del abanico de requisitos que se desean incluir, además de las funcionalidades que aportan las tecnologías estudiadas.

Se requiere de una arquitectura de *software* que no tenga tanta complejidad, ya que no se trata de un proceso industrial y el desarrollo va a ser realizado por una persona, además de estar bien establecida, ya que no se quiere entorpecer la escalabilidad del sistema, facilitando, de esta forma, las fases de testeo y mantenimiento.

Para un desarrollo con *Flutter* como *framework* de desarrollo *front-end* y *Firebase* como infraestructura de servicios *back-end*, se empleará, la *arquitectura por capas*, o *Layered Architecture* una de las más arquitecturas más conocidas y elegidas en entornos de desarrollo móvil [9].

4.2.1. Arquitectura por capas

El objetivo de esta arquitectura es la de abstraer lo máximo posible los diferentes bloques o componentes que van a conformar el sistema.

Empleando la siguiente arquitectura se pretenderá obtener, a corto y largo plazo, los siguientes beneficios:

- Componentes que desempeñan una única funcionalidad bien definida.
- Mayor escalabilidad, favoreciendo la inclusión de nuevas funcionalidades.

- Facilidad en la identificación de posibles *bugs* o errores relativos a la ejecución del aplicativo.
- Permite la inclusión de *patrones de diseño* en el sistema.
- Código más legible y fácil de entender.

Mediante esta arquitectura es posible dividir el sistema mediante N bloques, para el aplicativo final se ha optado por una arquitectura dividida en cuatro capas, como se puede visualizar en la Figura 4.3.

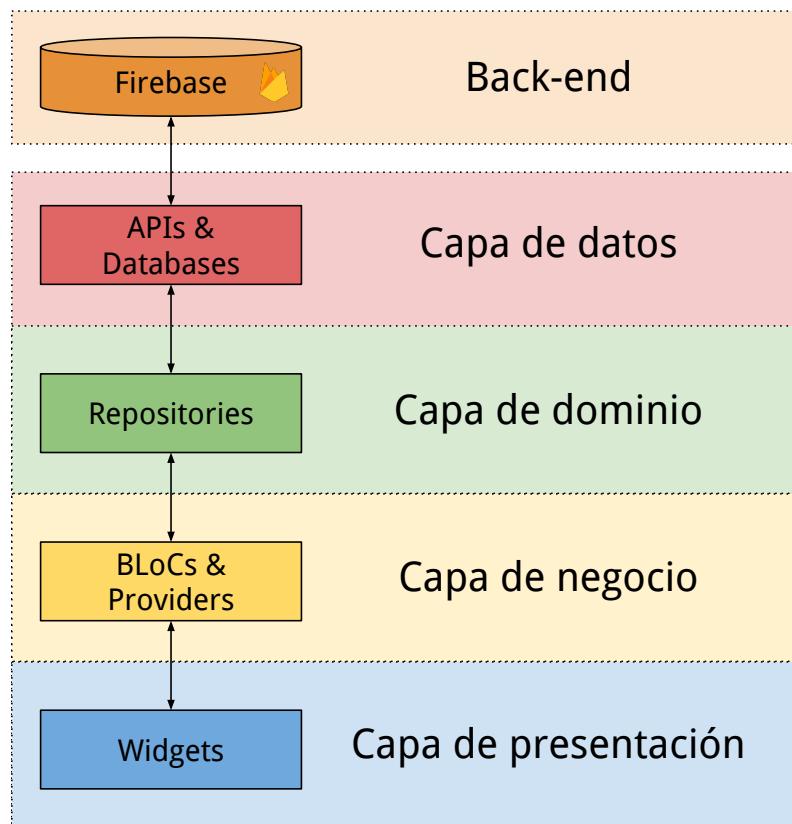


Figura 4.3: Diagrama de *Arquitectura basada en cuatro capas*

A continuación, se comentará la funcionalidad de cada una de las capas, en la Tabla 4.2:

Tabla 4.2: Función de las capas de la arquitectura del sistema

Capa	Funcionalidad
Capa de presentación	Mediante <i>widgets</i> se presentarán todas las vistas y componentes que contiene el aplicativo.
Capa de negocio	A partir de <i>manejadores de estados</i> se controlarán todos los cambios de estado presentes en la capa de <i>presentación</i> .
Capa de dominio	Las interfaces <i>repositorio</i> servirán de esqueleto para las clases relacionadas con la capa de <i>datos</i> .
Capa de datos	Implementando las interfaces de la capa de <i>dominio</i> se obtendrán los datos de fuentes externas como nuestro <i>back-end</i> de <i>Firebase</i>

Es importante remarcar que, en algunas arquitecturas como *Clean Architecture*, se suele abstraer un nivel más la capa de dominio aislando del sistema, en forma de paquetes externos, con el fin de reutilizarlo en otras soluciones [2].

4.2.2. Manejador de estados

Es cierto que *Flutter*, para aplicaciones más simples permite *manejar estados* directamente en la *capa de presentación*, mediante la función `setState()`[13], sin embargo, para nuestra aplicación y otras soluciones de gran envergadura y escalabilidad es importante establecer un *manejador de estados*, que esté presente en la capa de *negocio*, que se encargue de:

- Controlar todos los eventos presentes en el aplicativo para transformarlos en estados y transmitirlos a la capa de *presentación*.
- Obtener los datos sincronizados transmitidos por la capa de *dominio*.

Dentro del gran abanico de librerías e implementaciones que ofrece *Flutter* se ha optado por, como se puede visualizar en la Figura 4.3, los patrones de diseño:

- **Provider:** usado para la *inyección de dependencias* directamente en la capa de *presentación* transmitiendo estados a la misma.
- **Patrón BLoC:** empleado para centralizar los cambios de estado, recibiendo *eventos* de la capa de *presentación* y transmitiendo *estados* que cambien los componentes de la misma.

En la práctica, se empleará la combinación de ambos patrones mediante el paquete *flutter_bloc*, donde se podrán visualizar algunos ejemplos de uso en el siguiente capítulo de *Desarrollo de la solución*.

Bibliografía

- [1] A. Biørn-Hansen, T.-M. Grønli, and G. Ghinea, “A survey and taxonomy of core concepts and research challenges in cross-platform mobile development,” *ACM Comput. Surv.*, vol. 51, no. 5, Nov. 2018. [Online]. Available: <https://doi.org/10.1145/3241739>
- [2] S. Boukhary and E. Colmenares, “A clean approach to flutter development through the flutter clean architecture package,” in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2019, pp. 1115–1120.
- [3] A. Chakraborty, M. Baowaly, U. A Arefín, and A. N. Bahar, “The role of requirement engineering in software development life cycle,” *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, pp. 723–729, 2012.
- [4] J. Dalgety. (2017) Griddler puzzles and nonogram puzzles. [Online]. Available: <https://www.puzzlemuseum.com/griddler/gridhist.htm>
- [5] E. D. Guzmán Chamorro, “Impacto de la implementación del software de gestión para la fase de análisis de requerimientos funcionales en la Cooperativa Financiera Atuntaqui,” Master’s thesis, 2018.
- [6] H. Heitkötter, T. A. Majchrzak, and H. Kuchen, “Cross-platform model-driven development of mobile applications with md2,” in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 526–533. [Online]. Available: <https://doi.org/10.1145/2480362.2480464>
- [7] IEE, *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Std 830-1998, 1998.
- [8] C. Khawas and P. Shah, “Application of firebase in android app development-a study,” *International Journal of Computer Applications*, vol. 179, no. 46, pp. 49–53, 2018.
- [9] B. A. Kumar, “Layered architecture for mobile web based applications,” in *Asia-Pacific World Congress on Computer Science and Engineering*, 2014, pp. 1–6.
- [10] M. Latif, Y. Lakhrissi, E. H. Nfaoui, and N. Es-Sbai, “Review of mobile cross platform and research orientations,” in *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, 2017, pp. 1–4.
- [11] W. LELER, “What’s revolutionary about flutter,” *Hacker Noon*, 2019. [Online]. Available: <https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514>
- [12] T. Sneath. (2020) Github repo tracker. [Online]. Available: <https://github.com/timsneath/github-tracker>
- [13] F. D. Team. Adding interactivity to your flutter app. [Online]. Available: <https://flutter.dev/docs/development/ui/interactive>

- [14] K. Wiegers and J. Beatty, *Best Practises: Software requirements.* Microsoft Press, 2013.