



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Kairos
Documentación Técnica**



Presentado por Rodrigo Pérez Ubierna
en Universidad de Burgos — 4 de julio de 2024
Tutor: Dra. Sandra Rodríguez Arribas
y Dr. José Antonio Barbero Aparicio

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	v
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	7
Apéndice B Especificación de Requisitos	11
B.1. Introducción	11
B.2. Objetivos generales	12
B.3. Catálogo de requisitos	12
B.4. Especificación de requisitos	14
Apéndice C Especificación de diseño	29
C.1. Introducción	29
C.2. Diseño de datos	29
C.3. Diseño procedimental	34
C.4. Diseño arquitectónico	44
C.5. Diseño visual	45
Apéndice D Documentación técnica de programación	55
D.1. Introducción	55
D.2. Estructura de directorios	55

D.3. Manual del programador	57
D.4. Compilación, instalación y ejecución del proyecto	64
D.5. Pruebas del sistema	64
Apéndice E Documentación de usuario	65
E.1. Introducción	65
E.2. Requisitos de usuarios	65
E.3. Instalación	65
E.4. Manual del usuario	66
Apéndice F Anexo de sostenibilización curricular	75

Índice de figuras

A.1. Registro de sprints	2
C.1. Pantalla principal Firestore Database	29
C.2. Forma datos usuarios	30
C.3. Forma datos relojes	31
C.4. Forma datos subasta	31
C.5. Conexión con Flutter parte 1	32
C.6. Conexión con Flutter parte 2	33
C.7. Esquema general de casos de uso	35
C.8. Iniciar sesión	36
C.9. Registrar usuario	37
C.10.Ver relojes	37
C.11.Añadir reloj	38
C.12.Editar reloj	38
C.13.Eliminar reloj	39
C.14.Ver subastas	39
C.15.Crear subasta	40
C.16.Eliminar subasta	40
C.17.Pujar en una subasta	41
C.18.Comprar reloj directamente en una subasta	42
C.19.Editar información personal	43
C.20.Consultar precio reloj	44
C.21.Esquema conexión de elementos	44
C.22.Diseño inicio de sesión y registro de cuenta	46
C.23.Diseño recuperación de contraseña y página principal	47
C.24.Diseño ver mis relojes y añadir nuevos	48
C.25.Diseño creación de subasta y estado de ventas	49
C.26.Diseño de aplicar a una subasta y predicción de precio	50

C.27.Diseño configuración de información personal	52
D.1. Estructura de directorios	56
D.2. Definición de atributo Firestore	58
D.3. Forma de consulta de datos a base de datos	58
D.4. Función para extraer datos de dataset	59
D.5. Forma de hacer la aplicación responsive	60
D.6. Forma de actualizar un widget	61
D.7. Funcion para visualización de diálogos	61
D.8. Forma de hacer consulta a la API	62
D.9. Variables booleanas para contraseña	62
D.10.Lógica de ocultar y mostrar contraseña	63
D.11.Expresiones regulares	63
D.12.Funció n que inicializa Firebase	64
E.1. Inicio de sesión	66
E.2. Registro de usuario	67
E.3. Página principal	68
E.4. Edición de la información del usuario	69
E.5. Página listado de relojes	70
E.6. Registro de un reloj	70
E.7. Edición de un reloj	71
E.8. Página listado de subastas	72
E.9. Creación de una subasta	72
E.10.Vista de subasta respecto a otro usuario	74

Índice de tablas

B.1. Caso de Uso 1: Iniciar sesión	15
B.2. Caso de Uso 2: Registrar usuario	16
B.3. Caso de Uso 3: Ver pantalla principal	17
B.4. Caso de Uso 4: Registrar reloj	18
B.5. Caso de Uso 5: Eliminar reloj	19
B.6. Caso de Uso 6: Editar reloj	20
B.7. Caso de Uso 7: Vender reloj en subasta	21
B.8. Caso de Uso 9: Eliminar venta en curso	22
B.9. Caso de Uso 10: Pujar en subasta	23
B.10.Caso de Uso 11: Comprar reloj de manera directa	24
B.11.Caso de Uso 12: Estimar precio de un reloj	25
B.12.Caso de Uso 13: Editar datos personales	26
B.13.Caso de Uso 14: Cambiar contraseña	27

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Este apartado del informe recoge cual ha sido la planificación temporal estipulada, así como el estudio de viabilidad tanto económica como legal llevado a cabo. Es importante marcar que Kairos nace de un proyecto educativo sin recursos, por lo que se ha buscado utilizar herramientas en sus versiones gratuitas. En cuanto a la planificación temporal, el número de días por sprint ha sido bastante uniforme, pero el número de horas ha variado mucho debido a factores externos a este trabajo.

A.2. Planificación temporal

Tras la primera reunión con los tutores, decidimos llevar a cabo el proyecto siguiendo la metodología ágil *Scrum*, vista en diversas asignaturas del grado. Sin embargo, debido a ser un proyecto con un único desarrollador y con un contexto cambiante, los ciclos de trabajo han sido diferentes. Al trabajar con GitHub, los *sprints* se han definido como *milestones*, tal y como se representa en la Figura A.1.

Labels	Milestones				New milestone
↑ 8 Open	✓ 0 Closed				Sort ↴
Sprint 8		100% complete	0 open	5 closed	
↳ Past due by 8 days. (⌚ Last updated 5 days ago)					Edit Close Delete
Sprint 7		100% complete	0 open	5 closed	
↳ Past due by 14 days. (⌚ Last updated 11 days ago)					Edit Close Delete
Sprint 6		100% complete	0 open	3 closed	
↳ Past due by 29 days. (⌚ Last updated 24 days ago)					Edit Close Delete
Sprint 5		100% complete	0 open	3 closed	
↳ Past due by about 2 months. (⌚ Last updated about 1 month ago)					Edit Close Delete
Trabajo full-end desarrollo de la app					
Sprint 4		100% complete	0 open	5 closed	
↳ Past due by 2 months. (⌚ Last updated about 2 months ago)					Edit Close Delete
Sprint 3		100% complete	0 open	4 closed	
↳ Past due by 2 months. (⌚ Last updated 3 months ago)					Edit Close Delete
Programación y documentación relacionada con Machine Learning.					
Sprint 2		100% complete	0 open	3 closed	
↳ Past due by 4 months. (⌚ Last updated 3 months ago)					Edit Close Delete
Práctica en lenguajes, refaciendo uso del lenguaje de datos del D. [secreto]					
Sprint 1		100% complete	0 open	3 closed	
↳ Past due by 4 months. (⌚ Last updated 4 months ago)					Edit Close Delete
Primer sprint del proyecto. No tiene prioridad, se encuadra en [secreto]					
Iniciación		100% complete	0 open	1 closed	
↳ Past due by 4 months. (⌚ Last updated 4 months ago)					Edit Close Delete
Prueba de iniciación					

Figura A.1: Registro de sprints

Sprint iniciación : 14/02/2024 - 28/02/2024

El primer *sprint* duró dos semanas. El conjunto de tareas realizadas acabo fueron:

1. Buscar información teórica sobre Flutter.
2. Crear el repositorio del trabajo en GitHub.
3. Aprender cómo crear un proyecto Flutter.
4. Aprender a subir los cambios en GitHub a través de comandos.
5. Definir mi primer *milestone* en GitHub.
6. Crear mi primera *issue* y adjudicarla *labels* creadas por mí mismo.
7. Completar documentos necesarios para la universidad relacionados con este trabajo.

En cuanto a problemas encontrados, la tarea más complicada fue la instalación de Flutter y la conexión del equipo con la cuenta de GitHub. Flutter consta de muchos requerimientos para su instalación y su documentación de creación de proyectos es algo compleja, cosa que ralentizó el trabajo.

Sprint 1 : 28/02/2024 - 13/03/2024

Este *sprint* duró también dos semanas. Las tareas realizadas durante ello fueron:

1. Buscar un *dataset* adecuado para la realización de un modelo de predicción de precios de un producto.
2. Buscar alternativas para afrontar la parte *back-end* del proyecto.
3. Una vez completa la tarea anterior, conectar la base de datos con el proyecto.
4. Configurar LaTeX para el desarrollo del informe.
5. Seguir trabajando con la herramienta Flutter.

La mayor parte del tiempo fue destinada a la búsqueda de un *dataset* con mucha información. El desconocimiento en la materia hizo que no se supiera bien qué características debía tener un buen *dataset*. Tras varias opciones, se decidió centrar la aplicación en el mundo de relojes y se eligió el óptimo para ello.

Sprint 2 : 13/03/2024 - 03/04/2024

El tercer *sprint* del proyecto se dedicó únicamente a la parte del aprendizaje automático. Duró tres semanas y las tareas realizadas durante este fueron:

1. Buscar información teórica relativa al aprendizaje automático.
2. Conseguir un *dataset* donde los valores fueran numéricos en vez de cadenas de texto.

Durante este sprint se invirtió mucho tiempo para entender qué era el aprendizaje automático y cómo podía ser empleado en este proyecto. El problema principal que surgió fue la gran cantidad de información no relevante que albergaba el *dataset*. En esta tarea se invirtió mucho tiempo en estudiar la estipulación de las características de más de 280000 registros de relojes.

Sprint 3 : 03/04/2024 - 24/04/2024

El cuarto *sprint* del trabajo fue también dedicado al aprendizaje automático. Al igual que el anterior, duró tres semanas. Las tareas realizadas fueron:

1. Buscar una forma de completar aquellos datos sin valor en el *dataset*.
2. Intentar aplicar *cross validation* al futuro modelo.
3. Crear un modelo y entrenarlo.
4. Definir métricas para valoración del modelo.

El tiempo invertido en este *sprint* fue mucho ya que, como se ha indicado en otros puntos del trabajo, muchas de las cosas aprendidas han sido a base de prueba y error. Se consiguió crear un preprocesamiento de los datos, la creación de un modelo y la obtención de unos resultados al aplicar distintas métricas. Tras la reunión, la métrica que más nos interesaba devolvía unos resultados esperanzadores, por lo que decidimos comenzar con la parte principal del proyecto.

Sprint 4 : 24/04/2024 - 08/05/2024

El quinto *sprint* duró dos semanas y se decidió comenzar a ver la parte protagonista del trabajo y para con la creación del modelo. Las tareas llevadas a cabo fueron:

1. Documentar los requisitos de la aplicación.
2. Conseguir las funciones *get* y *add* en Flutter relacionadas con el usuario.
3. Documentar todo el trabajo del aprendizaje automático realizado hasta la fecha.
4. Crear bocetos que representasen las futuras vistas de la aplicación.

En cuanto a los requisitos y los bocetos, realizarlos fue una tarea importante para establecer con claridad que se buscaba ver en la aplicación. Por otra parte, la tarea relacionada con las funciones *get* y *add* llevó bastante tiempo ya que, aunque las funciones fueran sencillas al final, llegar hasta ese momento necesitaba de conocer cosas previas no vistas. Un problema a destacar fue el archivo de dependencias del proyecto, el cual es bastante complejo de configurar sin saber nada acerca de la materia.

Sprint 5 : 08/05/2024 - 22/05/2024

El sexto *sprint* duró dos semanas y siguió la dinámica del anterior. Las tareas realizadas fueron:

1. Desarrollo de la vista de inicio de sesión
2. Desarrollo del registro de un usuario.
3. Corrección de errores en la documentación.

El trabajo de este *sprint* no fue sencillo, pero mereció mucho la pena para entender ciertas cosas sobre Flutter que se desconocían hasta el momento. Tras la reunión de este ciclo de trabajo, vimos que había una serie de errores que eran necesarios tener en cuenta y corregir para el siguiente *sprint*. El problema más destacable fue la incapacidad por mi parte de llevar variables de una ventana a otra, lo que ralentizó todo buscando distintas formas de conseguir esta parte fundamental.

Sprint 6 : 22/05/2024 - 31/05/2024

El séptimo *sprint* duró solo una semana debido a la gran cantidad de tiempo que tuve para afrontarlo. Las tareas realizadas durante este fueron:

1. Crear acciones relacionadas con relojes.
2. Crear acciones relacionadas con subastas.
3. Crear la vista de la lista de relojes.
4. Crear la vista de la lista de subastas.

Debido al buen momento que atravesaba, fueron surgiendo tareas intermedias que significaban nuevas implementaciones al trabajo. Lo más complicado del *sprint* fue la forma de relacionar los relojes con las subastas, ya que la base de datos utilizada era NoSQL y mis conocimientos se centraban en bases de datos SQL.

Sprint 7 : 31/05/2024 - 13/06/2024

Tras una demo con los tutores y una puesta en común de las tareas a afrontar, se realizó un *sprint* de dos semanas donde se realizaron las siguientes tareas:

1. Incluir campo *wallet* al usuario.
2. Tratar cambios en los estados de los relojes y subastas tras realizar distintas acciones.
3. Mejorar validaciones de la creación de relojes.
4. Insertar desplegables con opciones para completar el registro de relojes.

Lo más complicado de este sprint fue la conexión de un archivo externo al proyecto para poder crear los desplegables de los campos como *brand* o *model*. Lo más satisfactorio de este ciclo de trabajo fue ver como conseguía realizar una subasta con distintos usuarios.

Sprint 8 : 13/06/2024 - 19/06/2024

El noveno y último *sprint* duró una semana. Las tareas realizadas fueron:

1. Resolver errores tenidos en el anterior sprint.
2. Incluir condicionantes al campo *wallet* del usuario para controlar mejor las subastas.
3. Tratar la decoración y respuesta de la aplicación a cualquier tipo de dispositivo.
4. Conectar modelo a la aplicación.

Las tres primeras tareas no me llevaron mucho tiempo y pude dedicar casi todas las horas a la conexión del modelo. Sin duda, la conexión del modelo a la aplicación ha sido el punto más problemático de todo el trabajo. Tras los resultados “esperanzadores” obtenidos en anteriores *sprints*, vimos que no iban a servir de mucho porque no me fue posible conectarlo como yo creía hacerlo. Se intentaron diversas formas de conectar, ya sea cambiando el formato, la limpieza de datos... llegando a definir un sencillo modelo de predicción que se tuvo que lanzar a producción para que llegase información a la aplicación. Puede que el camino fueran escasos pasos, pero el desconocimiento de esto hizo que el trabajo se complicase mucho.

A.3. Estudio de viabilidad

Definir un estudio de viabilidad del trabajo realizado es una tarea difícil cuando se trata de una aplicación con carácter educativo. Sin embargo, se intentará explicar con un ejemplo como podría haberse dado tal caso si la aplicación hubiera sido requerida a una empresa externa.

Viabilidad económica

Cuando estudiamos la viabilidad económica de un proyecto, debemos centrarnos en tres puntos fundamentales:

1. Costes iniciales
2. Costes operativos
3. Ingresos y análisis de rentabilidad

En cuanto a los costes iniciales, se debe tener en cuenta el número de desarrolladores necesarios para llevar a cabo el trabajo. En este caso he sido yo mismo el único desarrollador pero podría ser necesario contratar más desarrolladores si quisiera escalarse la aplicación. A estos costes debemos añadir el precio de adquisición de licencias y herramientas de software. En nuestro caso:

1. Visual Studio Code
2. Android Studio
3. Flutter
4. Firebase Firestore
5. Render

Todas han sido utilizadas en sus versiones gratuitas aunque alguna de ellas ofrece planes con costes añadidos si la aplicación lo requiriese. Ejemplos de programas son Adobe XD para el diseño y prototipado o Google Ads para la realización de campañas.

En cuanto al coste operativo, englobamos costes destinados al mantenimiento de la aplicación y actualizaciones de seguridad. También entran dentro de estos costes el personal para atender al cliente, lanzamiento de

campañas para mantener vivo el marketing de la aplicación y/o comisiones de distintas plataformas de pago.

Si nos centramos tanto en los ingresos como en el análisis de rentabilidad, ambos pilares van ligados. Por un lado, es importante tener en cuenta las comisiones de venta, estipulación de tarifas para la publicación de productos y/o espacios publicitarios dentro de la aplicación. De esta forma podríamos generar ingresos pasivos que ayudaran al mantenimiento de esta.

Por tanto, según datos recientes, los costes anuales serían:

1. Sueldo medio de un desarrollador *web* en España : 30000 euros
2. Coste del ordenador donde se realiza el trabajo: 500 euros
3. Coste del *Hosting* y dominio: 200 euros
4. Costes de marketing y publicidad (campaña básica): 1000 euros
5. Costes de mantenimiento: 500 euros

Por otro lado, los supuestos ingresos anuales esperados son:

1. Publicidad : 500 euros
2. Tarifas para publicación de productos: 100 usuarios a 5 euros el mes
3. Comisiones por venta: varía mucho porque se definiría un porcentaje del precio total. Suponemos 300 ventas de 1000 a un 10 por ciento de comisión.

Por tanto:

$$\text{Costes Totales Anuales} = 30,000 + 500 + 200 + 1,000 + 500 = 32,200 \text{ euros} \quad (\text{A.1})$$

$$\text{Ingresos por Publicidad} = 500 \text{ euros} \quad (\text{A.2})$$

$$\text{Ingresos por Tarifas} = 100 \times 5 \times 12 = 6,000 \text{ euros} \quad (\text{A.3})$$

$$\text{Ingresos por Comisiones} = 50 \times 1,000 \times 0,10 = 5,000 \text{ euros} \quad (\text{A.4})$$

$$\text{Ingresos Totales Anuales} = 500 + 6,000 + 5,000 = 11,500 \text{ euros} \quad (\text{A.5})$$

$$\text{Balance Anual} = \text{Ingresos Totales Anuales} - \text{Costes Totales Anuales} \quad (\text{A.6})$$

$$\text{Balance Anual} = 11,500 - 32,200 = -20700 \text{ euros} \quad (\text{A.7})$$

Según los cálculos, el proyecto no es viable con una pérdida el primer año de 20700 euros. Sin embargo, la ayuda de futuros inversores para comenzar con algo más de capital y ofreciendo mejoras tras ese ingreso, cambiaría mucho el balance ofrecido.

Viabilidad legal

Al igual que el apartado anterior, para estudiar la viabilidad legal del proyecto también es importante definir cinco pilares clave para tomar una decisión tras este estudio:

1. Registro de empresa y regulación de las subastas.
2. Protección al consumidor
3. Propiedad intelectual
4. Seguridad
5. Aspectos fiscales

Como la aplicación va a ser desplegada en España y está siendo lanzada por una empresa, es necesario que esta se encuentre registrada en el país donde vaya a operar. Hay que añadir que, además del registro, es necesario la adquisición de licencias como Licencia Comercial o Registro de IVA entre otras. Por otro lado, al ser una aplicación de subastas, es necesario el cumplimiento de las leyes de subastas locales y nacionales, así como atender a las regulaciones específicas sobre venta de productos de lujo. Ejemplos de ellas son:

1. Ley de Contratos del Sector Público (LCSP) según Real Decreto Legislativo 3/2011, de 14 de noviembre.
2. Ley de Enjuiciamiento Civil (LEC) según Ley 1/2000, de 7 de enero.
3. Reglamento General de la Ley de Contratos de las Administraciones Pùblicas según Real Decreto 1098/2001, de 12 de octubre.

En cuanto a la protección del consumidor, se deben establecer de manera clara tanto los términos de uso como las políticas de devolución y reembolsos. Las políticas de privacidad deben ser claras y se deben cumplir la ley de protección de datos poniendo como ejemplo la *General Data Protection Regulation* (GDPR) para la protección de datos de individuos dentro de la UE.

La defensa por la propiedad intelectual también es importante tenerla en cuenta a la hora de estudiar este punto. Se deben tomar medidas para la protección de usuarios y, ya que Kairos realiza transacciones, estas deben cumplirse según los estándares de seguridad en transacciones en línea.

Por último, no debemos olvidar aspectos fiscales como el cumplimiento de las obligaciones fiscales tanto locales como internacionales, así como la gestión de impuestos sobre ventas ingresos y demás de aspectos relacionados.

Apéndice B

Especificación de Requisitos

B.1. Introducción

Una buena práctica a la hora de realizar cualquier proyecto es definir previamente qué se busca. En cuanto al desarrollo software, es muy importante definir previamente información esencial que, bien estructurada, va a ser de gran ayuda para afrontar mejor el proyecto. Como marcábamos en anteriores apartados, durante la realización del trabajo se ha explicado la metodología ágil a seguir. Sin embargo, existen ciertos conceptos que deben conocerse para entender el porqué de este apartado.

Una vez reunidos tanto tutores como alumno, se decide definir en grupo cuáles son los requisitos del programa. Denominamos “requisitos funcionales” a todo aquello que esperamos que sea realizado por nuestro programa final. Un ejemplo es: “el programa debe permitir crear una cuenta”. Por otro lado, los “requisitos no funcionales” complementan a los anteriores marcando cómo un software debe realizar ciertas actividades. En este caso, un ejemplo sería: “el cambio de una página a otra de la aplicación no puede ser mayor de 4 segundos”.

Definir previamente esto nos ayuda a entender mejor qué queremos como usuario final y nos permite estructurar mejor el desarrollo del proyecto. Sin embargo, esto es solo un parte. A esto debe sumar la definición de las historias de usuario, los casos de uso y los diagramas de interacción.

Una “historia de usuario” no es más que la explicación del requisito funcional en un lenguaje comprensible para el usuario. Por otra parte, conocemos como “casos de uso” a la forma de describir como el usuario

realiza una actividad dentro del sistema, mientras que los “diagramas de interacción” representan las interacciones que se dan dentro de este.

Con todo esto definido, podemos dar comienzo a la definición de objetivos, definición de los requisitos y desarrollo de estos, así como un esquema gráfico de cómo debería ser la aplicación final.

B.2. Objetivos generales

Como indicamos en la presentación formal de la idea para este trabajo, los objetivos de este proyecto podrían resumirse como:

- Realizar una aplicación multiplataforma apoyándose en el uso de un SDK denominado Flutter.
- Permitir al usuario subir su producto y crear una subasta para la venta de este, así como un precio de venta directa si el comprador no quiere aplicar a la subasta y conseguir el producto directamente.
- Incorporar un apartado de predicción de precios de un reloj según sus características.

Se detallan con más precisión en el documento memoria de este trabajo.

B.3. Catálogo de requisitos

A modo de ser lo más ordenado posible, se enumeran a continuación todos los requisitos establecidos durante el proyecto:

Requisitos funcionales

- RF-1) Un usuario accederá a la aplicación introduciendo su email y su contraseña en una pantalla de inicio de sesión.
- RF-2) La pantalla de inicio de sesión mostrará un mensaje de error si el nombre de usuario no existe y/o si la contraseña es errónea.
- RF-3) Un usuario podrá registrarse accediendo desde un botón específico para ello situado en la pantalla de inicio de sesión.

- RF-4) El registro de un usuario debe contener los siguientes datos personales: nombre, apellidos, fecha de nacimiento, país, email, *password*, número de cuenta donde se realizarán las transacciones y monedero.
- RF-5) El email deben ser único en la aplicación.
- RF-6) La contraseña del usuario debe seguir las siguientes directrices: longitud mínima de 8 caracteres y contener al menos una mayúscula, una minúscula, un número y un carácter especial.
- RF-7) Si no se completa de manera correcta el registro, la aplicación lanzará un mensaje de error indicando el origen del problema.
- RF-8) La aplicación contará con una pantalla principal donde dispondrá de las siguientes funcionalidades: listado de mis relojes, listado de subastas y actualización de la información de usuario.
- RF-9) Un usuario podrá subir su reloj a la aplicación marcando: nombre, modelo, referencia, movimiento, material de la caja, material del brazalete, año de fabricación, estado y género. Son obligatorios marca, modelo, año y estado.
- RF-10) Un usuario podrá eliminar un reloj.
- RF-11) Un usuario podrá editar la información de un reloj.
- RF-12) Un usuario podrá vender su reloj previamente subido a través de una subasta creada por él.
- RF-13) Un usuario podrá vender su reloj de manera directa marcando un precio de venta directa.
- RF-14) Un usuario podrá eliminar una venta en curso siempre que no se haya recibido una primera respuesta a la subasta.
- RF-15) Un usuario podrá comprar un reloj pujando una cantidad de dinero a través de una subasta.
- RF-16) Un usuario podrá comprar un reloj de manera directa aceptando el precio de venta directa.
- RF-17) La aplicación integrará un modelo que marque cuál es el precio recomendado de venta directa de un producto.

- RF-18) La aplicación contará con un apartado donde se introducirán los siguientes datos para la estimación de un precio de un reloj: nombre, modelo, referencia, movimiento, material de la caja, material del brazalete, año de fabricación, estado y género. Son obligatorios marca, modelo, año y estado.
- RF-19) La aplicación contará con un apartado de edición de datos personales.
- RF-20) La aplicación contará con un apartado de edición de contraseña.

Requisitos no funcionales

- RNF-1) Los cargos en los monederos deben ser seguros.
- RNF-2) La aplicación debe ser intuitiva y al alcance de todo tipo de persona mayor de edad.

B.4. Especificación de requisitos

CU-1	Iniciar sesión
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-1, RF-2
Descripción	El usuario inicia sesión introduciendo su correo electrónico y su contraseña
Precondición	El usuario debe tener una cuenta registrada.
Acciones	<ol style="list-style-type: none"> 1. El usuario abre la aplicación. 2. El usuario introduce su email y contraseña donde corresponda. 3. El usuario pulsa el botón “Iniciar Sesión”.
Postcondición	El usuario accede a la pantalla principal de la aplicación.
Excepciones	<ul style="list-style-type: none"> ■ Si el email no existe, se muestra un mensaje de error. ■ Si la contraseña es incorrecta, se muestra un mensaje de error.
Importancia	Alta

Tabla B.1: Caso de Uso 1: Iniciar sesión

CU-2	Registrar usuario
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-3, RF-4, RF-5, RF-6, RF-7
Descripción	Un futuro usuario se registra en la aplicación aportando sus datos personales
Precondición	El usuario no estar registrado en la aplicación.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla de inicio de sesión. 2. El usuario pulsa el botón “Registrarse”. 3. El usuario introduce sus datos personales (nombre, apellidos, fecha de nacimiento, país, email, contraseña, número de cuenta, monedero). 4. El usuario pulsa en el botón “Registrarse”.
Postcondición	Se crea la cuenta y ya puede iniciar sesión.
Excepciones	<ul style="list-style-type: none"> ■ Si el email ya está registrado, se muestra un mensaje de error. ■ Si la contraseña no cumple con los criterios, se muestra un mensaje de error. ■ Si faltan datos obligatorios, se muestra un mensaje de error. ■ Si es menor de edad, se muestra un mensaje de error.
Importancia	Alta

Tabla B.2: Caso de Uso 2: Registrar usuario

CU-3	Ver pantalla principal
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-8
Descripción	El usuario puede acceder a la pantalla principal donde verá sus relojes, subastas y opciones de actualización de información.
Precondición	El usuario debe haber iniciado sesión.
Acciones	<ol style="list-style-type: none"> 1. El usuario inicia sesión. 2. La aplicación muestra la pantalla principal con las funcionalidades disponibles.
Postcondición	El usuario puede interactuar con las funcionalidades de la pantalla principal.
Excepciones	<ul style="list-style-type: none"> ■ Si el usuario no ha iniciado sesión, se redirige a la pantalla de inicio de sesión.
Importancia	Alta

Tabla B.3: Caso de Uso 3: Ver pantalla principal

CU-4	Registrar reloj
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-9
Descripción	El usuario puede subir un reloj a la aplicación proporcionando los datos necesarios.
Precondición	El usuario debe haber iniciado sesión.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario selecciona la opción ver listado de relojes. 3. El usuario selecciona la opción de subir un reloj. 4. El usuario introduce los datos del reloj (nombre, marca, modelo, referencia, movimiento, material de la caja, material del brazalete, año de fabricación, estado, género y precio). 5. El usuario envía el formulario para subir el reloj.
Postcondición	El reloj es añadido a la colección del usuario en la aplicación.
Excepciones	<ul style="list-style-type: none"> ■ Si faltan datos obligatorios, se muestra un mensaje de error.
Importancia	Alta

Tabla B.4: Caso de Uso 4: Registrar reloj

CU-5	Eliminar reloj
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-10
Descripción	El usuario puede eliminar un reloj de su colección.
Precondición	El usuario debe haber iniciado sesión y tener relojes en su colección.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario selecciona la opción ver listado de relojes. 3. El usuario selecciona el reloj que desea eliminar. 4. El usuario pulsa el botón “Eliminar”.
Postcondición	El reloj es eliminado de la colección del usuario.
Excepciones	<ul style="list-style-type: none"> ■ Si existe algún contratiempo, el reloj no se elimina.
Importancia	Media

Tabla B.5: Caso de Uso 5: Eliminar reloj

CU-6	Editar reloj
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-11
Descripción	El usuario puede editar la información de un reloj en su colección.
Precondición	El usuario debe haber iniciado sesión y tener relojes en su colección.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario selecciona la opción ver listado de relojes. 3. El usuario selecciona el reloj que desea editar. 4. El usuario pulsa el botón “Editar”. 5. El usuario modifica la información del reloj. 6. El usuario guarda los cambios.
Postcondición	La información del reloj se actualiza en la colección del usuario.
Excepciones	<ul style="list-style-type: none"> ■ Si faltan datos obligatorios, se muestra un mensaje de error.
Importancia	Media

Tabla B.6: Caso de Uso 6: Editar reloj

CU-7	Vender reloj en subasta
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-12
Descripción	El usuario puede vender un reloj creando una subasta.
Precondición	El usuario debe haber iniciado sesión y tener relojes en su colección.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario accede a la lista de subastas. 3. El usuario accede al apartado “Crear subasta” 4. El usuario completa los detalles de la subasta (nombre del reloj, precio inicial, precio de venta directa y fecha de cierre). 5. El usuario confirma la creación de la subasta pulsando en el botón “Crear”.
Postcondición	La subasta se crea y el reloj se pone a la venta.
Excepciones	<ul style="list-style-type: none"> ■ Si faltan datos obligatorios para la subasta, se muestra un mensaje de error.
Importancia	Alta

Tabla B.7: Caso de Uso 7: Vender reloj en subasta

CU-9	Eliminar venta en curso
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-14
Descripción	El usuario puede eliminar una venta en curso si no se ha recibido una oferta en la subasta.
Precondición	El usuario debe haber iniciado sesión y tener una venta en curso sin ofertas.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario accede a la lista de subastas. 3. El usuario selecciona la venta que desea eliminar. 4. El usuario pulsa el botón “Eliminar Venta”.
Postcondición	La venta es eliminada y el reloj se retira de la venta.
Excepciones	<ul style="list-style-type: none"> ■ Si la venta ya tiene una oferta, no se puede eliminar y el botón aparece bloqueado.
Importancia	Media

Tabla B.8: Caso de Uso 9: Eliminar venta en curso

CU-10	Pujar en subasta
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-15
Descripción	El usuario puede pujar una cantidad de dinero en una subasta para comprar un reloj.
Precondición	El usuario debe haber iniciado sesión y la subasta debe estar activa.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario accede a la lista de subastas. 3. El usuario selecciona una subasta activa. 4. El usuario introduce una cantidad para pujar. 5. El usuario confirma la puja.
Postcondición	La puja del usuario se registra en la subasta.
Excepciones	<ul style="list-style-type: none"> ■ Si la subasta no está activa, se muestra un mensaje de error. ■ Si la puja es menor que el mínimo y/o valor actual, se muestra un mensaje de error. ■ Si la puja es mayor o igual al precio de venta directa, se muestra un mensaje de error. ■ Si el usuario no tiene suficiente dinero para afrontar todas sus pujas, se muestra un mensaje de error. ■ Si el usuario tiene menos dinero en su monedero que la cantidad a pujar, se muestra un mensaje de error.
Importancia	Alta

Tabla B.9: Caso de Uso 10: Pujar en subasta

CU-11	Comprar reloj de manera directa
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-13, RF-16
Descripción	El usuario puede comprar un reloj directamente al aceptar el precio de venta directa establecido.
Precondición	El usuario debe haber iniciado sesión y el reloj debe estar a la venta.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario accede a la lista de subastas. 3. El usuario selecciona una subasta. 4. El usuario confirma la compra directa pulsando sobre el botón de “Compra directa”.
Postcondición	El reloj es comprado por el usuario.
Excepciones	<ul style="list-style-type: none"> ■ Si el usuario no tiene suficientes fondos o la transacción no es posible por alguna razón, se muestra un mensaje de error. ■ Si el usuario no tiene suficiente dinero para afrontar todas sus pujas, se muestra un mensaje de error.
Importancia	Alta

Tabla B.10: Caso de Uso 11: Comprar reloj de manera directa

CU-12	Estimar precio de un reloj
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-17
Descripción	El usuario puede obtener una estimación del precio de un reloj ingresando sus detalles a la hora de querer subirlo a la aplicación.
Precondición	El usuario debe haber iniciado sesión.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla principal. 2. El usuario selecciona la opción ver listado de relojes. 3. El usuario selecciona la opción de subir un reloj. 4. El usuario introduce los datos del reloj (marca, modelo, año de fabricación y estado). 5. El usuario pulsa sobre el botón “Predecir precio”.
Postcondición	El sistema proporciona un mensaje con una estimación del precio del reloj introducido.
Excepciones	<ul style="list-style-type: none"> ■ Si no se pueden calcular los datos necesarios para la estimación, se muestra un mensaje de error. ■ Si hubiera fallos en la conexión con el modelo y/o la API, se muestra un mensaje de error.
Importancia	Media

Tabla B.11: Caso de Uso 12: Estimar precio de un reloj

CU-13	Editar datos personales
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-18
Descripción	El usuario puede editar sus datos personales en la aplicación.
Precondición	El usuario debe haber iniciado sesión.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla de edición de datos personales. 2. El usuario modifica los campos deseados. 3. EL usuario introduce la contraseña a modo de seguridad. 4. El usuario guarda los cambios.
Postcondición	Los datos personales del usuario se actualizan en la aplicación.
Excepciones	<ul style="list-style-type: none"> ■ Si hay algún problema con la modificación de datos, se muestra un mensaje de error.
Importancia	Media

Tabla B.12: Caso de Uso 13: Editar datos personales

CU-14	Cambiar contraseña
Versión	1.0
Autor	Rodrigo Pérez Ubierna
Requisitos asociados	RF-19
Descripción	El usuario puede cambiar su contraseña en la aplicación.
Precondición	El usuario debe haber iniciado sesión.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la pantalla de edición de datos personales. 2. El usuario introduce su contraseña actual y la nueva contraseña. 3. El usuario confirma la nueva contraseña.
Postcondición	La contraseña del usuario se actualiza en la aplicación.
Excepciones	<ul style="list-style-type: none"> ■ Si la nueva contraseña no cumple con los requisitos, se muestra un mensaje de error.
Importancia	Alta

Tabla B.13: Caso de Uso 14: Cambiar contraseña

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado del trabajo se van a marcar varios aspectos importantes como la forma de guardar los datos en la base de datos, los diagramas de casos de uso o la conexión de los distintos elementos, entre otras cosas.

C.2. Diseño de datos

Como se ha indicado en otros puntos de este trabajo, la base de datos utilizada ha sido Firestore Database. Esta base de datos, albergada en la nube y propiedad de Google, responde a la filosofía NoSQL. Para situarnos, una vez que entramos en la base de datos nos encontramos la ventana representada en la Figura C.1.

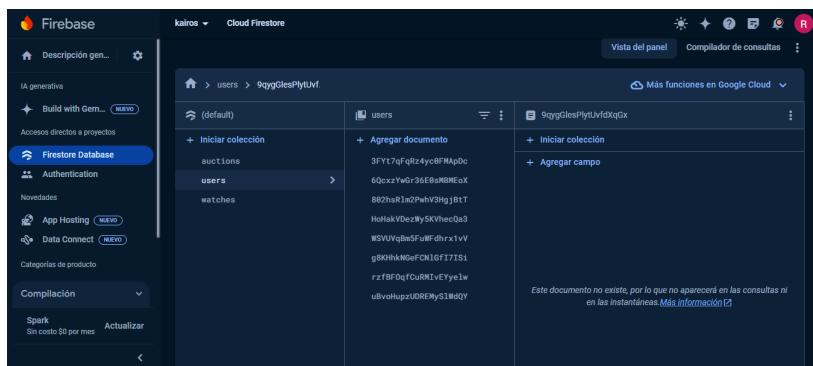


Figura C.1: Pantalla principal Firestore Database

Esta base de datos denomina las tablas con el nombre “colecciones”, mientras que los registros de cada una de ellas reciben el nombre de “documentos”. Si queremos ingresar los datos de manera manual, simplemente debemos seguir lo siguientes pasos:

1. Situarnos en la colección donde queramos guardar un nuevo registro.
2. Pulsar en el botón de la columna central “+ Agregar documento”
3. En la ventana modal, establecer un identificador único a cada “documento” (suelen generarse de manera automática) y completar añadiendo los campos que se deseen siguiendo: campo-tipo-valor

En relación a este trabajo, se han necesitado tres colecciones: usuarios, relojes y subastas. A continuación se presentan la Figura C.2, la Figura C.3 y la Figura C.4 que marcan los tipos de datos de cada colección respectivamente.

The screenshot shows the Firebase Cloud Firestore interface. On the left, there's a sidebar with project settings like 'IA generativa', 'Build with Gemini', 'Authentication', 'Firestore Database' (which is selected), 'App Hosting', 'Data Connect', 'Categorías de producto', 'Compilación', and 'Spark'. The main area shows a tree view of collections: 'users' (selected), 'auctions', and 'watches'. Under 'users', several document IDs are listed: '3FYt7qfqrZ4yc8FMaqDc', '60exxYwGr36E8mMEoX', '882hsR12pWv3HgJBtT', 'HoHakVde2By5kVheQa3', 'WSVUJqba5rUlfdrhx1vV', 'g8KlhkNgfCNlGf77sI', 'rzFBF0qfcuRKiVETyeIw', and 'uBvoIupzUDREMyS1IdQY'. To the right of the list, a modal window is open for a new document. The 'Agregar campo' section contains the following fields with their values:

bankCode	: 'N-123456789-RPU'
birthdate	: '25/2/2001'
country	: 'Spain'
email	: 'rodrigo.perez@kairos.com'
name	: 'Rodrigo'
password	: 'P@sswOrD'
surname	: 'Pérez Uberna'
wallet	: 50

Figura C.2: Forma datos usuarios

```

document u55JvFqYt9ZkHmsoHH
  + Agregar campo
    braceM: "Ostrich skin"
    brand: "Cartier"
    caseM: "Platinum"
    condition: "Good"
    model: "Ballon Bleu"
    movement: "Automatic"
    price: 8540
    reference: "CAR-89343"
    saleStatus: "At auction"
    sex: "Men's watch/Unisex"
    userEmail: "rodrigo.perez@kairos.com"
    watchNickName: "Reloj de Rodrigo"
    yop: 2010
  
```

Figura C.3: Forma datos relojes

```

document Ct1eiBqBJ#8YJYRHMpdn
  + Agregar campo
    actualValue: 0
    auctionStatus: "Active"
    buyerEmail: ""
    limitDate: 28 de junio de 2024, 3:46:00 p.m. UTC+2
    maximumValue: 12000
    minimumValue: 8000
    vendorEmail: "rodrigo.perez@kairos.com"
    watchNickName: "Reloj de Rodrigo"
  
```

Figura C.4: Forma datos subasta

Sin embargo, los datos no se guardan de manera manual. Desde Flutter se han creado dos funciones capaces de enviar y recibir los datos entre ambas partes. Como ejemplo, la Figura C.5 y la Figura C.6 representan el código necesario para establecer tal comunicación con la colección *auctions*.

```
1 import 'package:cloud_firestore/cloud_firestore.dart';
2
3 class Auction {
4   final String idAuction;
5   final String vendorEmail;
6   final String watchNickName;
7   final DateTime limitDate;
8   final int minimumValue;
9   final int maximumValue;
10  String buyerEmail;
11  String auctionStatus;
12  int actualValue;
13
14  Auction({
15    required this.idAuction,
16    required this.vendorEmail,
17    required this.watchNickName,
18    required this.limitDate,
19    required this.minimumValue,
20    required this.maximumValue,
21    required this.buyerEmail,
22    required this.auctionStatus,
23    required this.actualValue,
24  });
~
```

Figura C.5: Conexión con Flutter parte 1

```
26  /// Firestore Database -> Flutter
27  factory Auction.fromFirestore(DocumentSnapshot doc) {
28      Map data = doc.data() as Map<String, dynamic>;
29      return Auction(
30          idAuction: doc.id,
31          vendorEmail: data['vendorEmail'] ?? '',
32          buyerEmail: data['buyerEmail'] ?? '',
33          limitDate: (data['limitDate'] as Timestamp).toDate(),
34          auctionStatus: data['auctionStatus'] ?? '',
35          watchNickName: data['watchNickName'] ?? '',
36          minimumValue: data['minimumValue'] ?? 0,
37          actualValue: data['actualValue'] ?? 0,
38          maximumValue: data['maximumValue'] ?? 0);
39      }
40
41  /// Flutter -> Firestore Database
42  Map<String, dynamic> toMap() {
43      return {
44          'vendorEmail': vendorEmail,
45          'buyerEmail': buyerEmail,
46          'limitDate': limitDate,
47          'auctionStatus': auctionStatus,
48          'watchNickName': watchNickName,
49          'minimumValue': minimumValue,
50          'actualValue': actualValue,
51          'maximumValue': maximumValue
52      };
53  }
54 }
55 }
```

Figura C.6: Conexión con Flutter parte 2

La primera parte define el constructor para poder crear instancias de subastas, especificando dentro de ella tanto los campos que debe albergar como si son obligatorios de completar. Por otro lado, el método *fromFirestore* es el método que crea una instancia a partir del documento que recibe de la base de datos en la nube.

Si se quisiera realizar el sentido contrario, se utilizaría la función *toMap*, la cual recoge una instancia y la convierte en un mapa. El porqué de esta conversión es que es el formato requerido para almacenar datos en Firestore Database.

C.3. Diseño procedimental

No hay mejor forma de presentar el diseño procedimental que esquematizando el diagrama de casos de uso que pueden darse a lo largo del uso de la aplicación junto con sus diagramas de secuencia. A continuación, de la Figura C.7 hasta la Figura C.20 (ambas incluidas) presentaran todo esto.

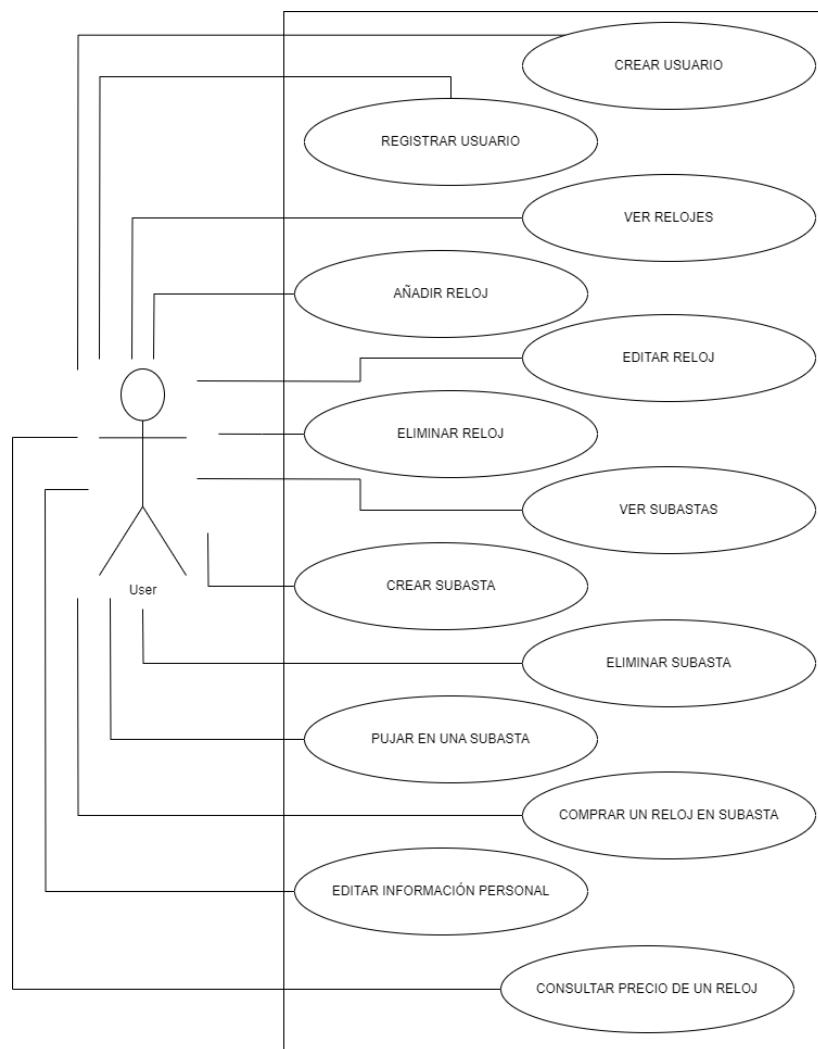


Figura C.7: Esquema general de casos de uso

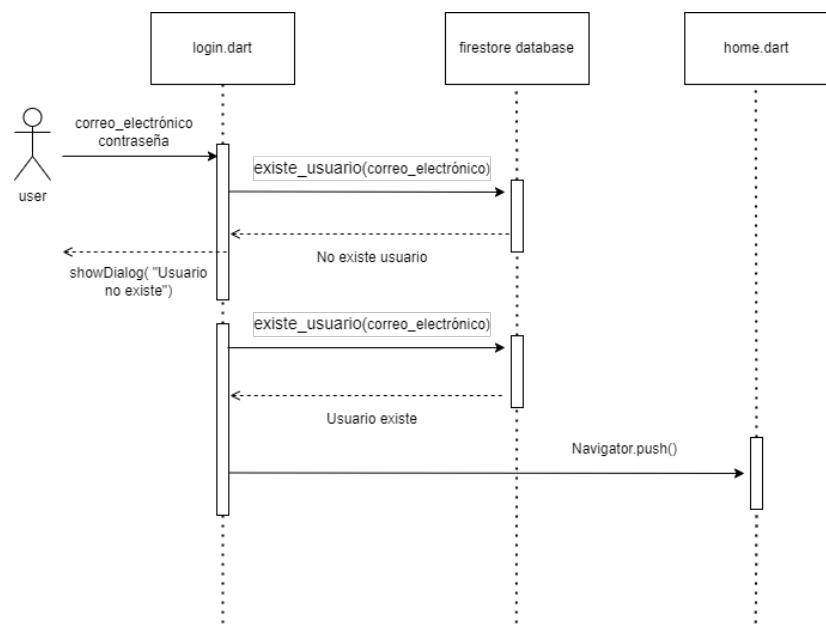


Figura C.8: Iniciar sesión

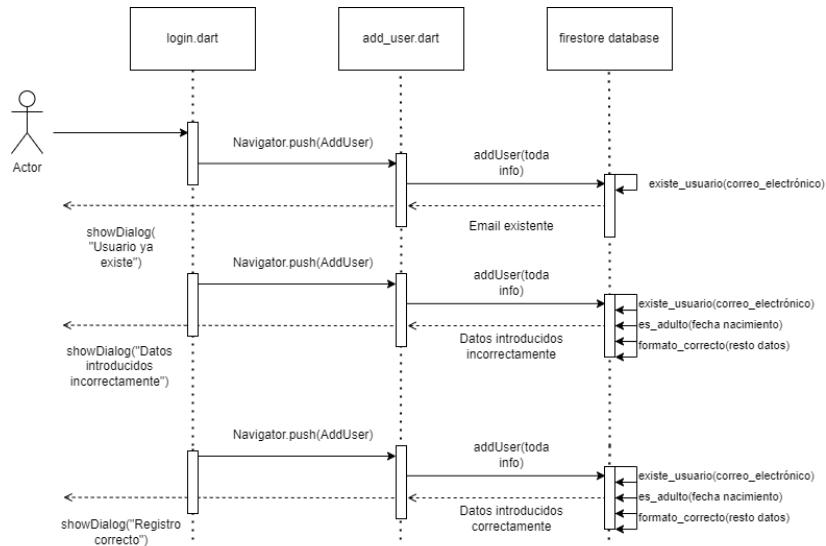


Figura C.9: Registrar usuario

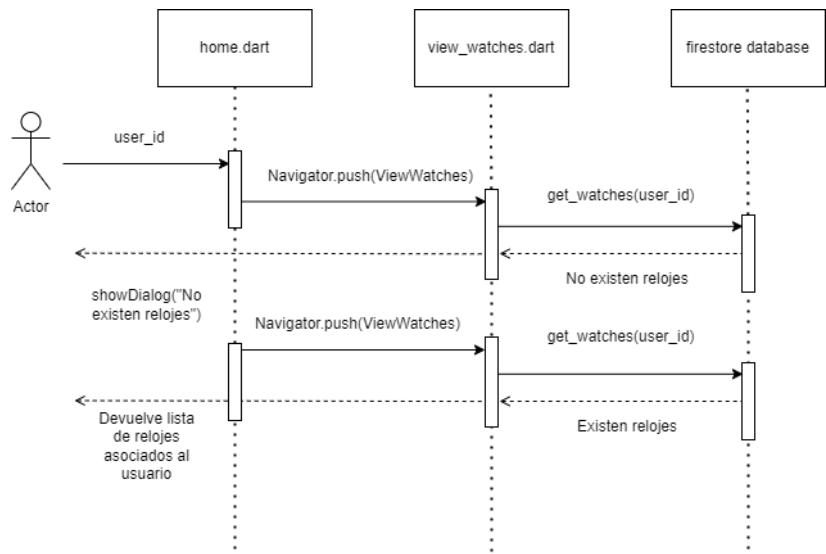


Figura C.10: Ver relojes

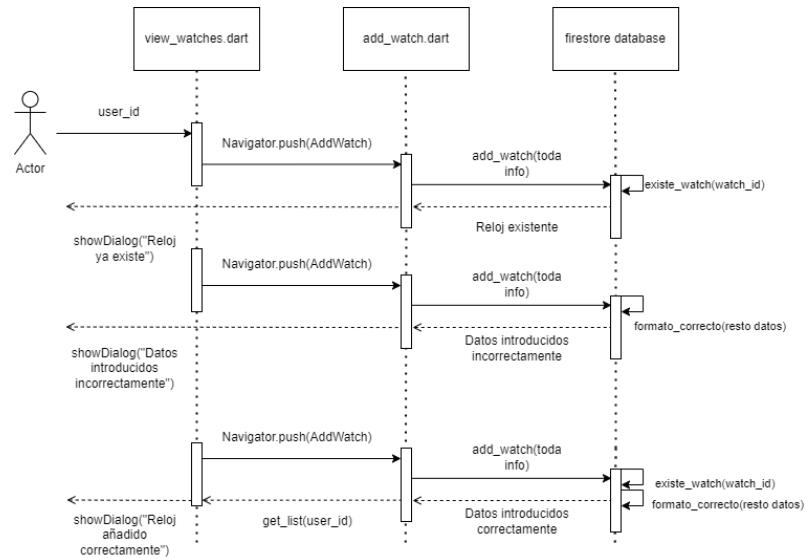


Figura C.11: Añadir reloj

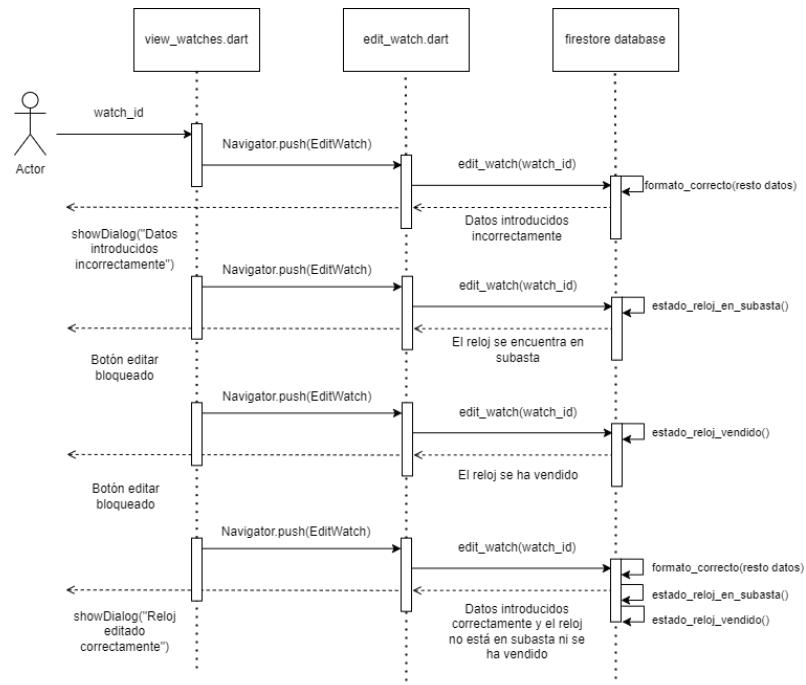


Figura C.12: Editar reloj

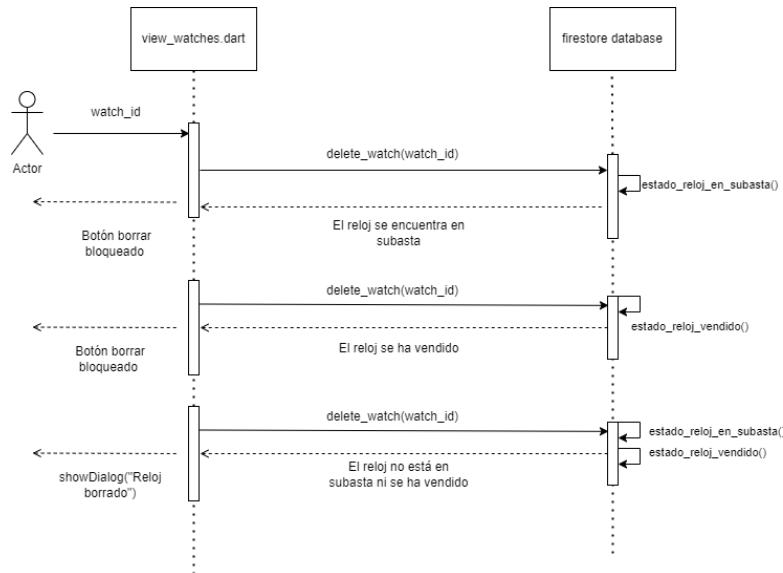


Figura C.13: Eliminar reloj

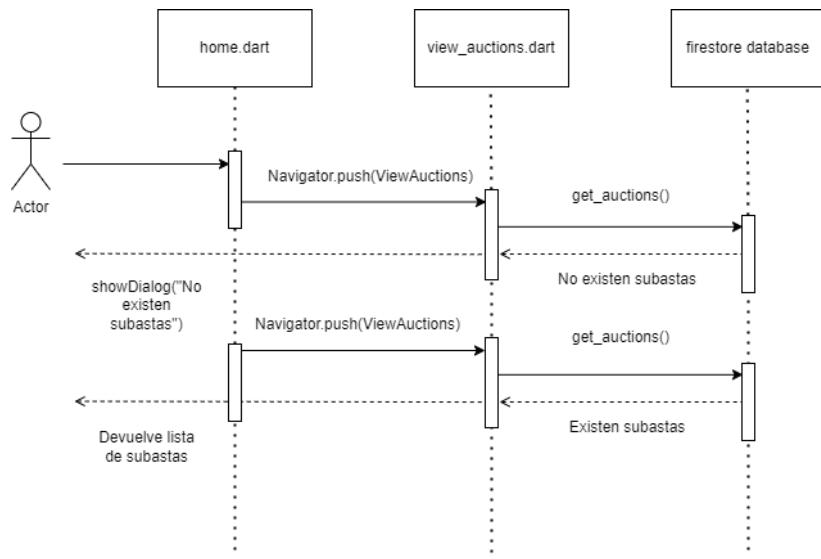


Figura C.14: Ver subastas

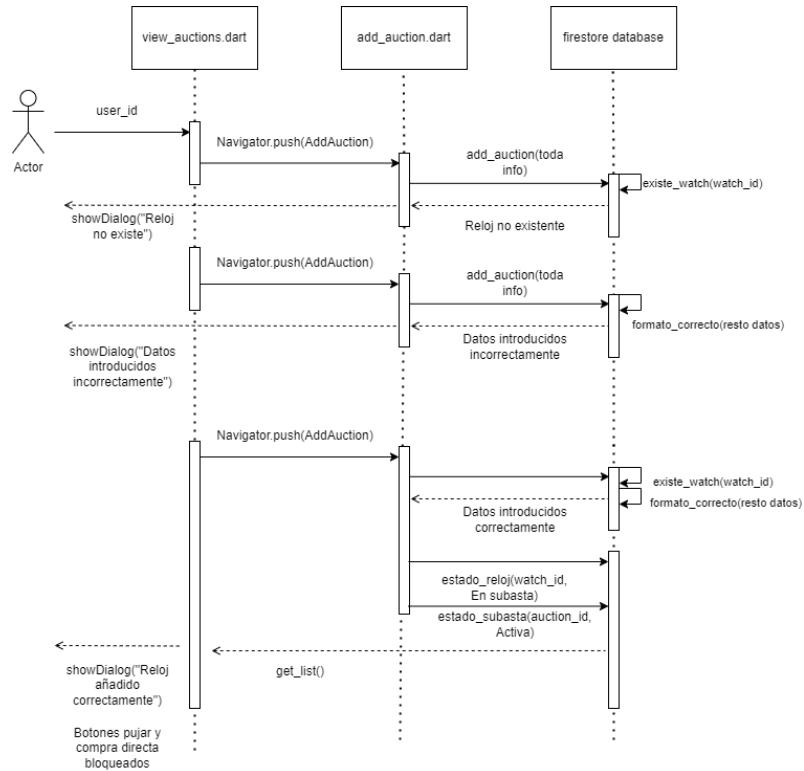


Figura C.15: Crear subasta

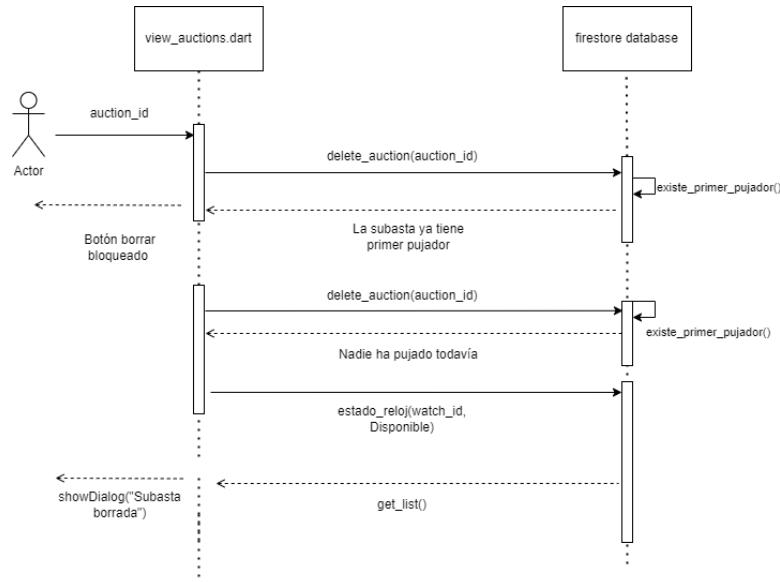


Figura C.16: Eliminar subasta

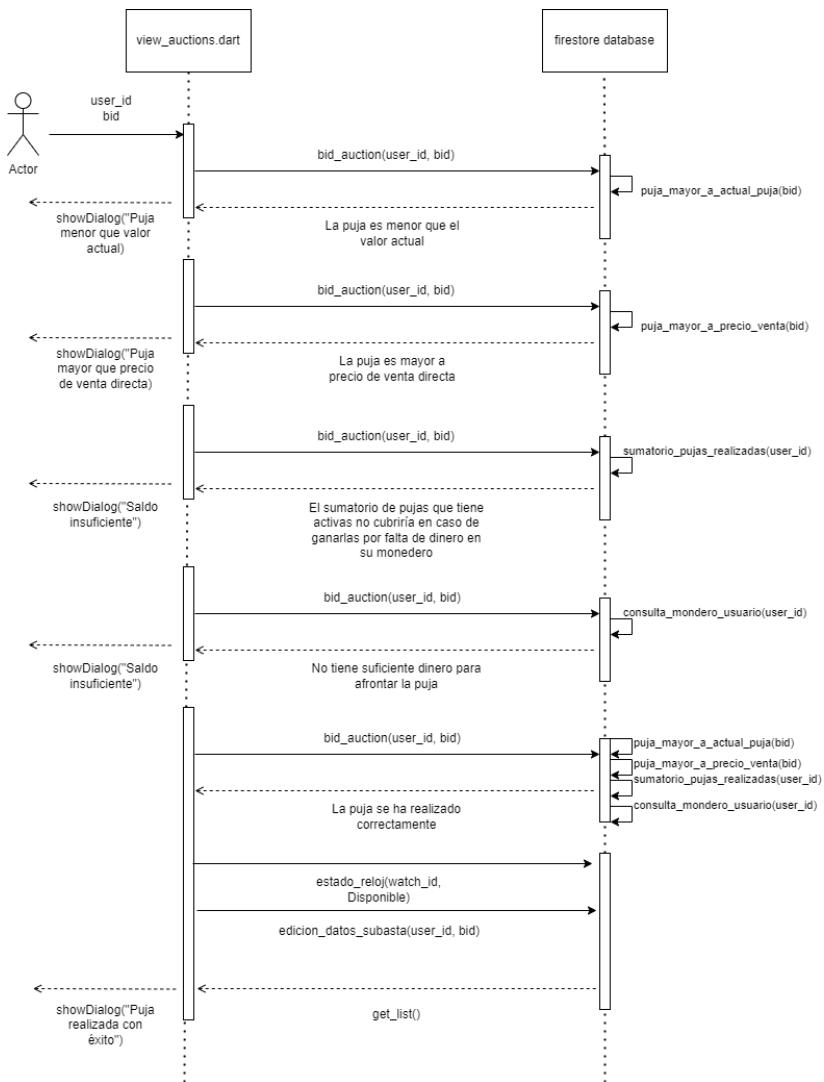


Figura C.17: Pujar en una subasta

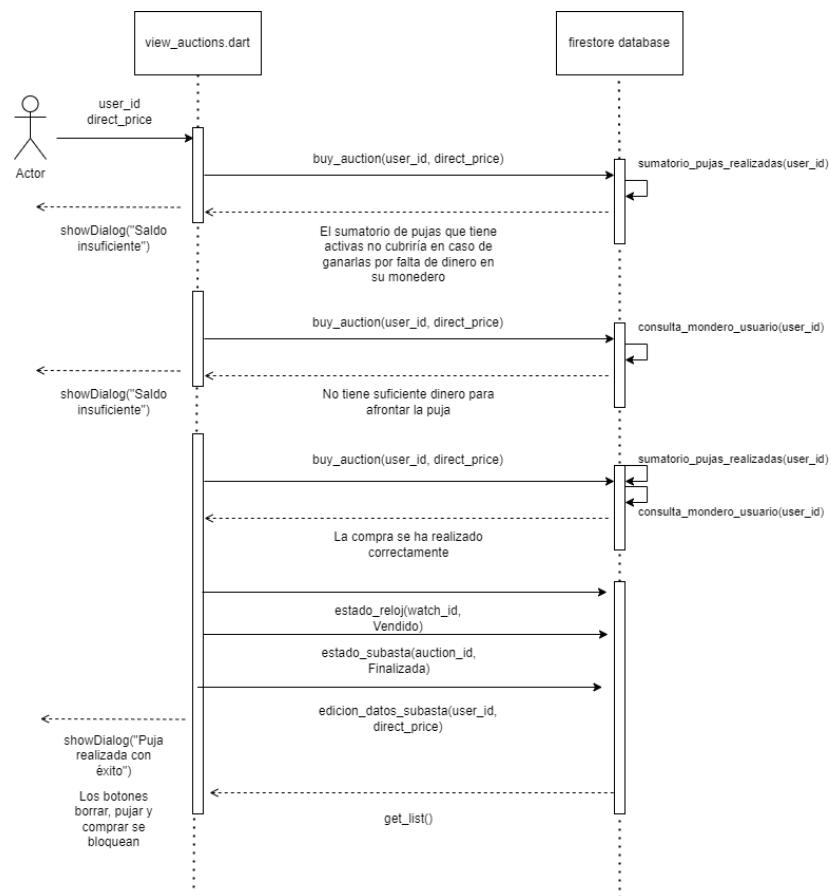


Figura C.18: Comprar reloj directamente en una subasta

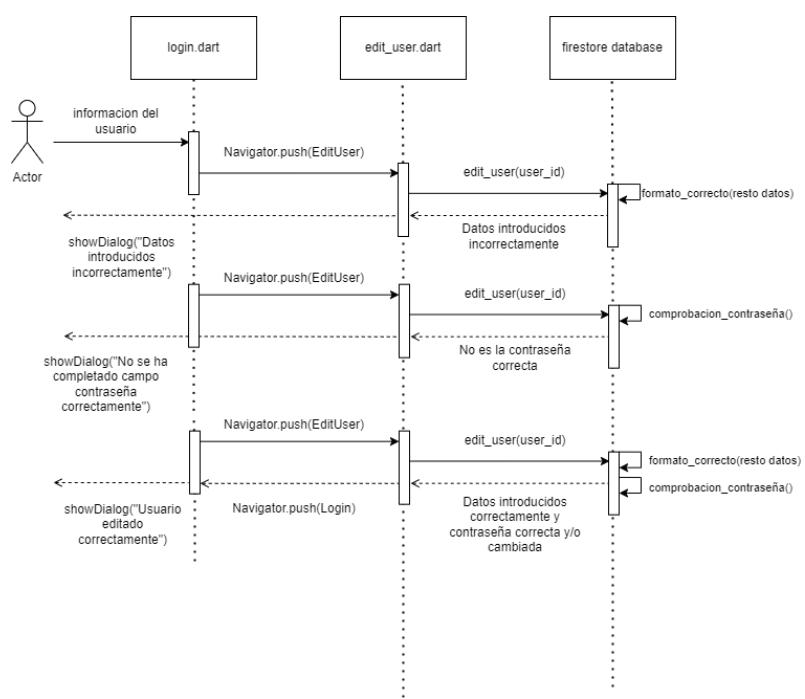


Figura C.19: Editar información personal

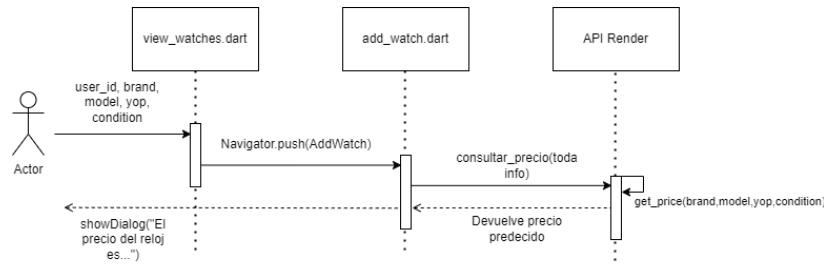


Figura C.20: Consultar precio reloj

C.4. Diseño arquitectónico

Siguiendo un poco lo marcado en la introducción, es importante definir cómo se comunican las distintas partes de este trabajo. En el informe referente a la memoria se establecía que el patrón “Modelo-Vista-Controlador” o MVC iba a ser quien marcaría la arquitectura dentro del proyecto Flutter. Sin embargo, también se debe tener en cuenta cómo se ha conectado la API del modelo de predicción de precios y la base de datos explicada anteriormente. Para ser más preciso, se presenta a través de la Figura C.21 la conexión entre las distintas partes. Para ello, se va a seguir el proceso de una acción como ejemplo.

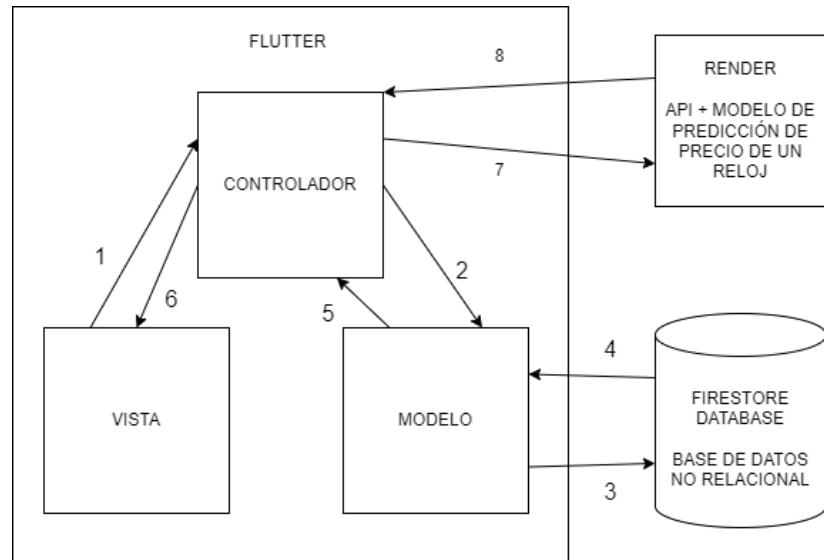


Figura C.21: Esquema conexión de elementos

1. Un usuario completa los campos del registro de un reloj y pulsa en el botón “Añadir reloj”. La vista (en nuestro caso el *widget*) se lo comunica al controlador (definido en el mismo *script* que la vista).
2. El controlador recoge la orden y se la envía al modelo.
3. El modelo se comunica con la base de datos para realizar la orden de añadir el reloj.
4. Si todo correcto, la base de datos realizará la acción sin problema.
5. El modelo comunica al controlador que la acción se realizó correctamente.
6. El controlador procesa esa información y se la comunica a la vista de la manera que sea pertinente.
7. La vista muestra un mensaje de información asegurando la inserción del nuevo reloj.
8. En el caso que se quisiera predecir el precio del reloj a la hora de añadir el reloj, es el controlador quien envía la orden directa a la API alojada en Render.
9. Render devuelve la respuesta a esa consulta y es el controlador quien se lo comunica a la vista para que se muestre una ventana informativa con el resultado de la predicción.

C.5. Diseño visual

Siempre es bueno realizar un pequeño esquema de cómo podrían verse las distintas pantallas de nuestra aplicación. Si es cierto que son bocetos que pueden distar bastante de la realidad, pero es una buena práctica si los realizamos en conjunto con el cliente. Al final lo que buscamos es entendernos con él lo mejor posible y que mejor que a través de un dibujo.

En mi caso, decidí intentar representar las interfaces basándome en todos los requisitos explicados en apartados anteriores. Para ello, utilicé el sitio web NinjaMock, el cual ya había utilizado en anteriores asignaturas del grado.

NinjaMock es una herramienta muy amigable y eficaz si nuestro objetivo es realizar bocetos de cómo podría quedar el *front-end* de nuestra aplicación.

Aunque tiene una versión de pago, este software nos brinda la oportunidad de crear un único proyecto con todas nuestras interfaces de manera gratuita.

A continuación, adjuntamos las distintas interfaces haciendo breves comentarios de ellas donde fuera necesario



Figura C.22: Diseño inicio de sesión y registro de cuenta

Como se puede apreciar en la Figura C.22, en muchas de las ventanas lanzaremos mensajes a modo de advertencia al usuario si este no completa correctamente los campos requeridos. Todas las condiciones de cada campo se marcan en los requisitos expuestos.



Figura C.23: Diseño recuperación de contraseña y página principal

Según lo representado en la Figura C.23, la idea de la recuperación de contraseña es reactivarla a través de un código al correo electrónico del usuario (aunque esto son líneas futuras). La mayoría de los iconos que nos llevan a alguna interfaz es porque se han marcado como iconos a ventanas emergentes para ayudar más al usuario.



Figura C.24: Diseño ver mis relojes y añadir nuevos

Según la Figura C.24, los iconos son bastante explícitos. Destaco lo que se marca como un billete: será la puerta a la puesta en venta del reloj. Lo vemos en la Figura C.25 como “creación de subasta”.



Figura C.25: Diseño creación de subasta y estado de ventas



Figura C.26: Diseño de aplicar a una subasta y predicción de precio

Al igual que en otras interfaces, el usuario podrá aplicar a la subasta directamente desde el botón representado en la Figura C.26. Saldrá una ventana emergente donde marcará el precio a aplicar.



Figura C.27: Diseño configuración de información personal

Hay que destacar sobre lo representado en la Figura C.27 que no debe cumplimentar todos los campos para configurar la información personal. Aun así, habrá campos que dependan unos de otros como los de contraseña.

Apéndice D

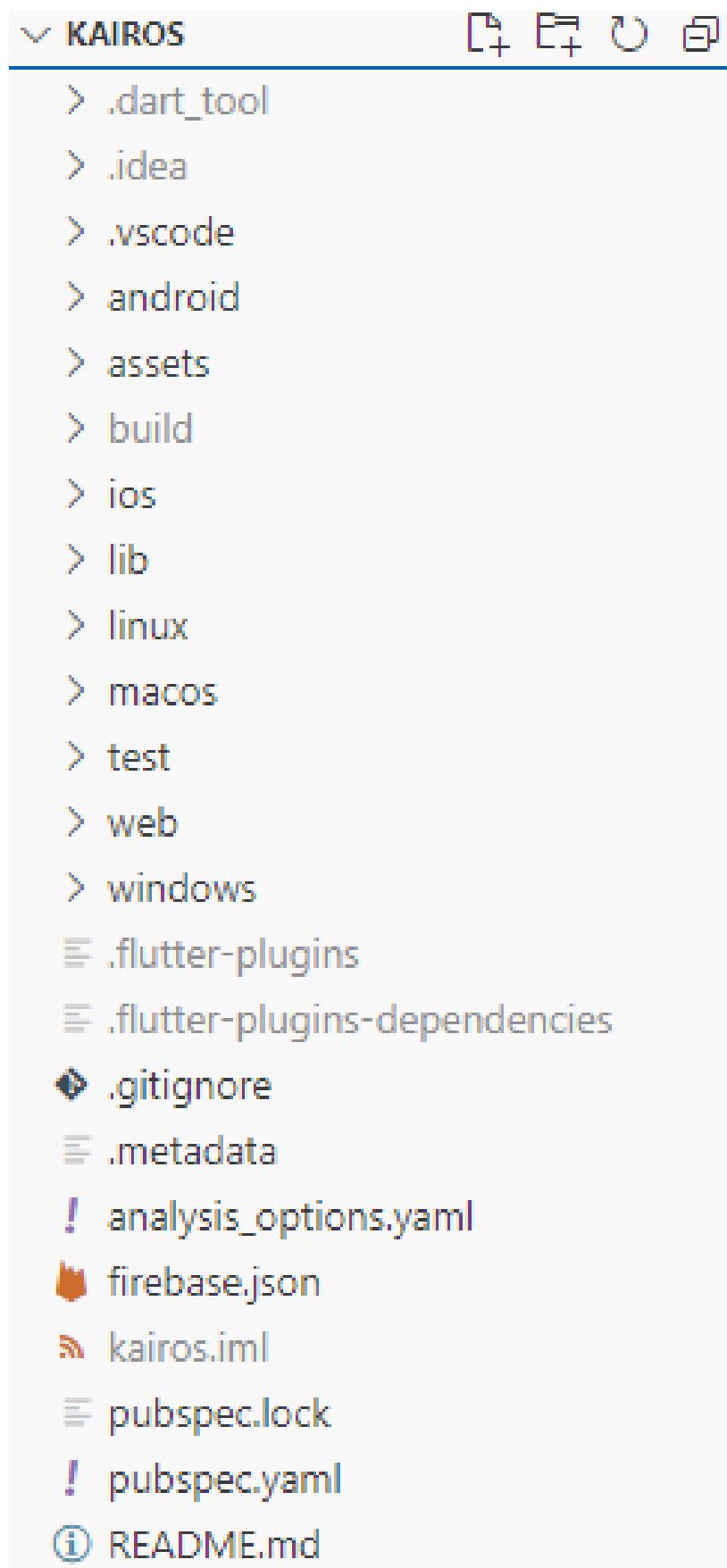
Documentación técnica de programación

D.1. Introducción

La idea de este apartado es marcar los pasos a seguir para que cualquier persona que decidiese escalar este trabajo pudiera ejecutarlo sin problema en su equipo. Es importante aclarar que Flutter establece una serie de requisitos específicos que son de vital importancia para que todo funcione correctamente.

D.2. Estructura de directorios

La estructura de directorios puede verse representada en la Figura D.1. Aunque muchas de las carpetas se crean automáticamente a la hora de crear el proyecto, destacan distintos archivos y directorios creados para la realización del trabajo. Hay que añadir que si no se comentan de manera específica significa que no son archivos donde se hayan realizado cambios y, por consiguiente, no son necesarios tocar si se quisiera escalar la aplicación.



En la carpeta *assets* se encuentran arhivos de diversos tipos que son necesarios para la realización de la aplicación. En mi caso, la carpeta contiene los siguientes archivos:

1. **databasewatches.csv** : archivo de extensión csv utilizado para el entrenamiento del modelo de predicción de precios y para cargar los desplegables de campos como *brand*, *model* y *condition* entre otros. Más adelante veremos como se ha conseguido tal tarea.
2. **kairoswallpaper.png** : imagen utilizada para establecer el fondo decorativo de la aplicación.

En la carpeta *lib* encontramos los scripts que han dado vida a la aplicación. Dentro de ella se han creado subdirectorios para seguir un orden y estructura clara. En este caso, los directorios han sido:

1. **models**: recoge los modelos de la aplicación.
2. **views**: recoge los scripts con las vistas y controladores de la aplicación.
3. **settings**: recoge scripts con funciones necesarias para traer datos de archivos presentes en la carpeta *assets*.

Por último, uno de los archivos más importantes es *pubspec.yaml*, donde se definen todas las dependencias con sus versiones necesarias para compilar el programa.

D.3. Manual del programador

Una vez explicado todo lo que uno debe saber antes de adrentarse en el proyecto, se procede a explicar cómo se ha programado la aplicación.

En la carpeta *models* se encuentran los tres scripts con extensión “.dart” que hacen el papel de modelos. Ya se pudo ver en la Figura C.5 y la Figura C.6 que era necesario entablar la comunicación con la base de datos y cómo debía hacerse.

Con esto visto, solo queda explicar las clases “AuctionRepository”, “UserRepository” y “WatchRepository”. Todas estas clases manejan las operaciones de creación, lectura, actualización y eliminación de objetos según se necesitó para afrontar los requisitos establecidos. Sin embargo, una cosa fundamental que comparten las tres es la instancia de Firebase Firestore dentro

de la clase para poder llevar todas estas a cabo, tal y como se representa en la Figura D.2.

```
class AuctionRepository {
    final FirebaseFirestore _db = FirebaseFirestore.instance;
```

Figura D.2: Definición de atributo Firestore

Ya que todas las funciones están comentadas y explicar todo haría de este informe una novela, me quiero quedar con cómo se han realizado funciones de consulta de datos a la base de datos. En el caso del modelo “user.dart”, se necesitó una función que devolviese el dinero de un usuario pasando como parámetro el correo electrónico de este, tal y como se representa en la Figura D.3.

```
// Returns money held by a user
Future<int> getWalletByEmail(String email) async {
    QuerySnapshot querySnapshot = await _db
        .collection('users')
        .where('email', isEqualTo: email)
        .limit(1)
        .get();

    if (querySnapshot.docs.isNotEmpty) {
        return querySnapshot.docs.first['wallet'];
    } else {
        throw Exception('User not found');
    }
}
```

Figura D.3: Forma de consulta de datos a base de datos

La forma de consultar datos a base de datos nos obliga, en este caso, a definir una función asíncrona que devolverá eventualmente un entero. En cuanto a la parte de la consulta, se debe especificar la colección a la que se quiere acceder (recordemos que lo que siempre hemos conocido como tablas, aquí son colecciones), la condición que debe cumplir, y limitamos a uno el resultado porque solo debemos recibir un único valor (en este caso la cantidad del dinero asociado a ese correo electrónico). El “.get()” será el encargado de devolvernos un “QuerySnapshot” con los resultados de la consulta.

La siguiente comprobación simplemente nos asegura que, si “QuerySnapshot” no llegó vacío, devuelva el valor *wallet* del primer documento. Si algo hubiera salido mal, se lanzaría una excepción especificando que el usuario no fue encontrado.

Con las cosas claras de la parte de los modelos, pasamos a la parte de *settings*. Como se ha explicado antes, esta carpeta se destinó a albergar funciones que pudieran ser comunes a varios *scripts* y/o comunicaciones con otros archivos. En este caso, se destinó a albergar el *script* donde se encuentra la función que permite cargar y procesar los datos del archivo con extensión “.csv” alojado en la carpeta *assets*.

Tal y como se representa en la Figura D.4, la función va a cargar el archivo utilizando una función propia de Flutter y va devolver una lista de mapas con los datos procesados. Es importante tener en cuenta que la primera fila del archivo son los encabezados, por lo que se indica con “*csvTable.skip(1)*”.

```

1 import 'package:flutter/services.dart' show rootBundle;
2 import 'package:csv/csv.dart';
3
4 class CsvService {
5   Future<List<Map<String, String>>> loadCsvData(String path) async {
6     try {
7       final rawData = await rootBundle.loadString(path);
8       List<List<dynamic>> csvTable =
9         const CsvToListConverter().convert(rawData);
10
11     List<Map<String, String>> data = [];
12     for (var row in csvTable.skip(1)) {
13       // disregard the first row
14       if (row.length >= 2) {
15         data.add({
16           'brand': row[0].toString(),
17           'model': row[1].toString(),
18         });
19       }
20     }
21     return data;
22   } catch (e) {
23     return [];
24   }
25 }
26 }
```

Figura D.4: Función para extraer datos de dataset

En cuanto a las vistas, las funciones quedan explicadas en el código con distintos comentarios donde sean necesarios. Sin embargo, existen ciertas

funciones que conviene dejar explciadas en este informe por si surgiese cualquier duda.

Como podemos apreciar en la Figura D.5, todos los textos, botones e imágenes de la aplicación van a compartir semejanzas con ella. Es cierto que podría haberse utilizado funciones *responsive* propias de la herramienta, pero durante la realización vimos que afeaba la interfaz ya que los botones tendían a ocupar el largo de la pantalla y, en monitor, no quedaba bonito. Por esta razón, se ideó ajustar el tamaño de los distintos objetos entendiendo al largo de la pantalla.

```

const SizedBox(height: 10),
SizedBox(
  child: ElevatedButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => const ViewAuctions(),
          settings: RouteSettings(
            arguments: widget.loginUserEmail, // RouteSettings
          ), // MaterialPageRoute
        );
    },
    style: ElevatedButton.styleFrom(
      padding: const EdgeInsets.symmetric(vertical: 15.0),
      textStyle:
        | | TextStyle(fontSize: isLargeScreen ? 18.0 : 14.0),
    ),
    child: const Text('See the auctions'),
  ), // ElevatedButton
), // SizedBox
const SizedBox(height: 10),

```

Figura D.5: Forma de hacer la aplicación responsive

Otro aspecto importante es la actualización de *widgets* cada vez que viajabamos de una vista a otra y/o realizabamos acciones que necesitaran de un *reload* de la ventana para visualizar los nuevos datos. Un caso lo tenemos en la vista “home.dart”, donde queremos que el *widget* que presenta el dinero del usuario, se actualice cada vez que viajamos a la venta home. Esto es lo que se representa en la Figura D.6.

```

@Override
void initState() {
    super.initState();
    _walletFuture = _getWallet();
}

Future<String> _getWallet() async {
    UserRepository userRepository = UserRepository();
    int walletAmount =
        await userRepository.getWalletByEmail(widget.loginUserEmail);
    return walletAmount.toString();
}

```

Figura D.6: Forma de actualizar un widget

Muchas de las cosas predefinidas por Flutter no son intuitivas para el usuario, por lo que se pensó definir una función que controlase la forma de visualizar los mensajes informativos que se den a lo largo de la aplicación. Para ello se utiliza la función presentada en la Figura D.7, la cual consta de dos partes donde definimos el título del error y una descripción detallada de ello.

```

void _showDialog(String title, String content) {
    showDialog(
        context: context,
        builder: (BuildContext context) {
            return AlertDialog(
                title: Text(title),
                content: Text(content),
                actions: <Widget>[
                    TextButton(
                        onPressed: () {
                            Navigator.of(context).pop();
                        },
                        child: const Text('OK'),
                    ), // TextButton
                ], // <Widget>[]
            ); // AlertDialog
        },
    );
}

```

Figura D.7: Funcion para visualización de diálogos

Como hemos indicado en otros apartados, la aplicación cuenta con una llamada a una API, consulta que es necesario definir en el *script* “addwatch.dart”. Para ello, es importante previamente importar el paquete “http”, el cual va a permitir que podamos lanzar la consulta, indicando la dirección de la API en la web (alojada en Render). Si el estado de la consulta es 200, todo habrá ido bien. En cambio, si no lo fuese, debemos tratar las excepciones tal y como se plantea en el código. Código en la Figura D.8.

```

try {
  final response = await http.post(
    Uri.parse('https://tfg-rpu-2024.onrender.com/predict'), // Address of the api loaded in Render.
    headers: {
      'Content-Type': 'application/json',
    },
    body: requestBody,
  );

  if (response.statusCode == 200) {
    var responseData = json.decode(response.body);
    double predictedPrice = responseData['predicted_price'][0];

    int roundedPrice = predictedPrice.round();

    setState(() {
      _priceController = roundedPrice;
    });

    _showDialog('Price Prediction', 'Predicted price: $roundedPrice €');
  } else {
    _showDialog('Prediction Error', 'Failed to predict price.');
  }
} catch (e) {
  _showDialog('Prediction Error', 'Failed to predict price: $e');
}
}

```

Figura D.8: Forma de hacer consulta a la API

En cuanto a las contraseñas, me pareció una buena idea incluir una función muy presente en muchas webs y aplicaciones: ocultar y mostrar la *password*. Para ello, simplemente definí dos variables booleanas que marcaran si la contraseña se debía mostrar o no en función del pulso sobre el botón definido. Tal código se ve representado en la Figura D.9 y la Figura D.10 respectivamente.

```

// whether or not to hide the password
bool _isPasswordVisible = false;
bool _isRepeatPasswordVisible = false;

```

Figura D.9: Variables booleanas para contraseña

```

const SizedBox(height: 20.0),
TextField(
  controller: _passwordController,
  obscureText: !_isPasswordVisible,
  style: TextStyle(fontSize: isLargeScreen ? 18.0 : 14.0),
  decoration: InputDecoration(
    hintText: 'Password',
    hintStyle: TextStyle(fontSize: isLargeScreen ? 18.0 : 14.0),
    fillColor: Colors.white,
    filled: true,
    border: const OutlineInputBorder(),
    suffixIcon: IconButton(
      icon: Icon(
        _isPasswordVisible ? Icons.visibility : Icons.visibility_off,
        color: Colors.grey,
      ), // Icon
      onPressed: () {
        setState(() {
          _isPasswordVisible = !_isPasswordVisible;
        });
      },
    ), // IconButton
  ), // InputDecoration
), // TextField
const SizedBox(height: 20.0),

```

Figura D.10: Lógica de ocultar y mostrar contraseña

Otro concepto que puede no ser entendido es lo definido como “regex”. Simplemente se conocen como expresiones regulares y su objetivo es marcar patrones para la manipulación de texto. Mediante su uso, se ha conseguido marcar las validaciones de campos como contraseña, número de cuenta o nombre del usuario, entre otros. Su programación de puede ver en la Figura D.11.

```

final emailRegex = RegExp(r'^[^\@]+\@\[^@\]+\.\[^@\]+');
final passwordRegex =
  || RegExp(r'^([0-9])([a-z])([A-Z])(\W)+$');
final nameSurnameCountryRegex = RegExp(r'^[^\s]+\$');

```

Figura D.11: Expresiones regulares

Por último, no me quería dejar una función importante a la hora de establecer los primeros pasos con Firestore Database. En el script “main.dart”, en la mayoría de casos, solo se recogen las rutas definidas en la aplicación. Sin embargo, al trabajar con esta base de datos, es necesario definir la función representada en la Figura D.12 para que todo funcione correctamente.

```
Run | Debug | Profile
void main() async {
    WidgetsFlutterBinding.ensureInitialized();
    await Firebase.initializeApp(
        options: DefaultFirebaseOptions.currentPlatform,
    );
    runApp(const MyApp());
}
```

Figura D.12: Función que inicializa Firebase

D.4. Compilación, instalación y ejecución del proyecto

Para poder ejecutar el proyecto en nuestro equipo, es importante cumplir una serie de requisitos previos e instalar algún que otro programa. Para hacerlo de la manera más estructurada posible, se detallan a continuación los que seguí según [?]:

1. Entramos en sitio web oficial de Flutter [?].
2. Pulsamos sobre el botón “Get started”.
3. Pulsamos sobre el icono de nuestro sistema operativo (en mi caso Windows)
4. Pulsamos sobre el botón “Mobile”
5. Instamos Visual Studio Code como

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección del documento, se hará un exhaustivo repaso de cómo el usuario puede realizar cualquier acción disponible en la aplicación. Para ser más precisos, la documentación contará con imágenes y explicaciones relativas a estas.

E.2. Requisitos de usuarios

Como ya se ha indicado a lo largo de este trabajo, la aplicación es multiplataforma, por lo que el usuario podrá acceder desde cualquier dispositivo móvil u ordenador. La aplicación tiene unos caminos lógicos muy claros y que se han pensado para que cualquier persona, ya tenga o no conocimientos informáticos, pueda desempeñar cualquier tarea. Además, toda la aplicación se encuentra en inglés.

E.3. Instalación

La instalación no es necesaria, pues todo se encuentra desplegado en la red. Sin embargo, si quisiera instalarse con el objetivo de escalar la aplicación, toda la información está disponible en la sección 4 del apéndice D de este documento: Compilación, instalación y ejecución del proyecto.

E.4. Manual del usuario

Una vez se sitúe el futuro usuario, su primera toma de contacto será con la ventana representada en la Figura E.1. Esta ventana corresponde con el inicio de sesión. El usuario deberá introducir sus datos personales, concretamente su correo electrónico, su contraseña y pulsar en el botón Log in. Si la información introducida es correcta, significará que el usuario tiene una cuenta en nuestra aplicación y pasará a encontrarse con la ventana representada en la Figura E.3.

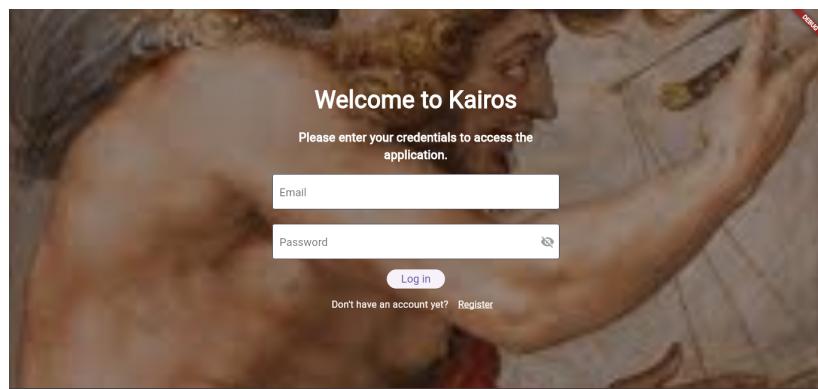


Figura E.1: Inicio de sesión

Si por el contrario el inicio de sesión no fuese correcto, la aplicación lanzará un mensaje de alerta indicando que los datos introducidos no son correctos. Esto puede deberse a dos factores:

1. El correo electrónico y/o la contraseña son incorrectas.
2. El usuario no se encuentra registrado en la base de datos.

La primera opción es simple y solo tendría que introducirse las credenciales tal y como se definieron al principio. Si la opción es la segunda, el futuro usuario podrá llegar a la ventana de registro pulsando en el botón Register. Tras esta acción, el usuario llegará a la venta representada en la figura E.2, donde tendrá que cumplimentar los campos con sus datos personales.

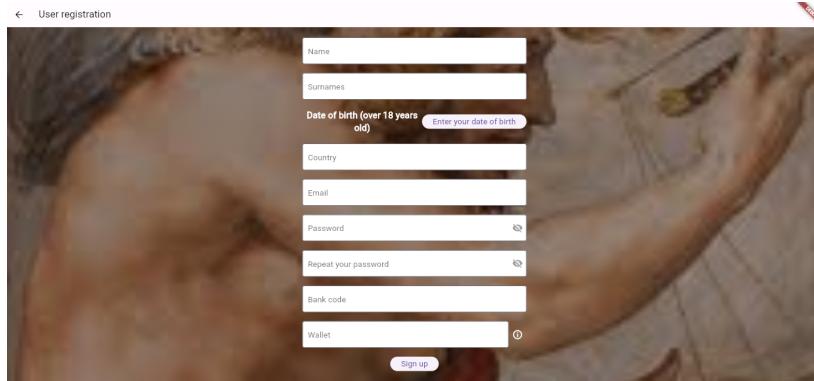


Figura E.2: Registro de usuario

El registro no da lugar a error, pues cuenta con mensajes de alerta estipulando donde se encuentra el fallo a la hora de introducir los datos. Para ser más concretos, se definen las condiciones de cada uno de los campos:

1. El campo *name* no puede contener números.
2. El campo *surnames* no puede contener números.
3. La fecha de nacimiento debe corresponder a una fecha con una diferencia mínima de 18 años respecto a la fecha actual.
4. El campo *country* no puede contener números.
5. El campo *email* debe seguir la forma general de cualquier correo electrónico: no empezar por un número, tener letra antes y después del símbolo @, contener el punto después del texto precedido por el símbolo @ y acabar con una extensión.
6. El campo *password* debe contener al menos 8 caracteres, una mayúscula, una minúscula, un carácter especial y un número.
7. El campo *repeat password* debe formarse con la misma información que el campo *password*.
8. El campo *wallet* solo debe contener números enteros positivos.
9. Todos los campos deben ser completados.

Una vez listo, el futuro usuario debe pulsar en el botón *Sign up* y la aplicación mostrará una ventana confirmando el registro de la cuenta. Para volver a

la ventana de inicio de sesión, solo debe pulsar en la flecha situada en la parte superior izquierda y rehacer los pasos de inicio de sesión explicados antes. Siguiendo estos pasos, el usuario llegará a la venta representada en la Figura E.3 conocida como página principal.

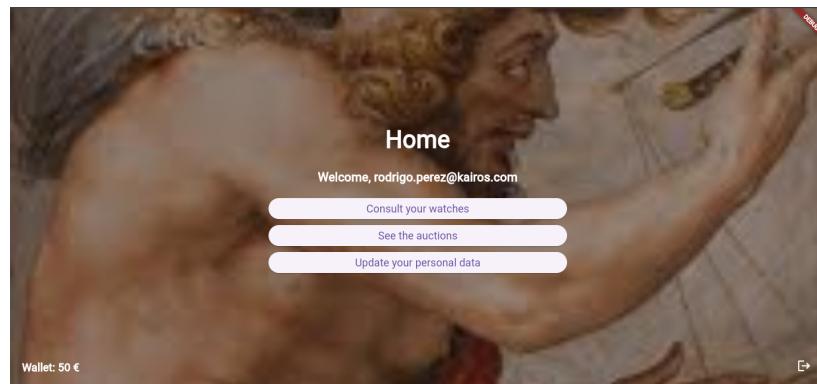


Figura E.3: Página principal

La página principal muestra un mensaje de bienvenida al usuario, dirigiéndose a él por su correo electrónico. Como puede apreciarse, en la parte inferior izquierda se encuentra la cantidad de dinero que el usuario tiene ingresado en su cuenta. En la parte inferior derecha se encuentra un botón que permite al usuario cerrar su sesión.

En la parte central de la página principal se pueden apreciar tres botones en columna que realizan tres acciones totalmente distintas:

1. Edición de la información personal
2. Listado de relojes del usuario
3. Listado de las subastas

Si pulsamos sobre el primer botón empezando desde abajo, viajaremos a la ventana representada en la Figura E.4. o denominada edición de la información del usuario. En ella el usuario podrá editar los datos que considere oportunos, así como recargar y/o sacar dinero de su monedero.

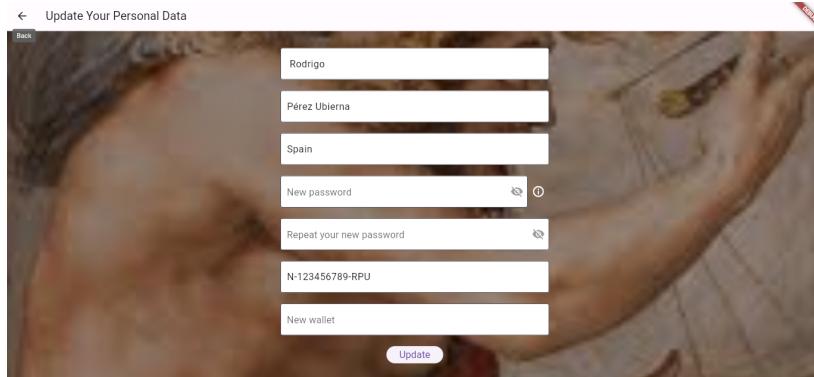


Figura E.4: Edición de la información del usuario

Al igual que en el registro, los campos cuentan con las mismas validaciones para asegurarnos de que no se da lugar al error. Una característica a tener en cuenta es la obligatoriedad a introducir la contraseña de nuevo por motivos de seguridad. Para ser más precisos, la ventana cuenta con un botón de información a la derecha del campo *password* explicando exactamente esta información. Una vez completado los campos correctamente, pulsando en el botón *Update*, la aplicación nos llevará de nuevo a la página principal.

El primer botón de la página principal empezando por el principio, lleva a la ventana representada en la Figura E.5 y denominada listado de relojes del usuario. Aquí el usuario podrá ver los relojes que ha introducido y eliminarlos y/o editarlos si no se encuentran presentes en alguna subasta con estado activo. Por el contrario, si el usuario quisiera añadir un nuevo reloj a su lista, se debe pulsar en el botón con el símbolo + ubicado en la parte inferior derecha. Si se pulsa sobre ello, el usuario viajará a la ventana representada en la Figura E.6 denominada registro de un reloj.



Figura E.5: Página listado de relojes

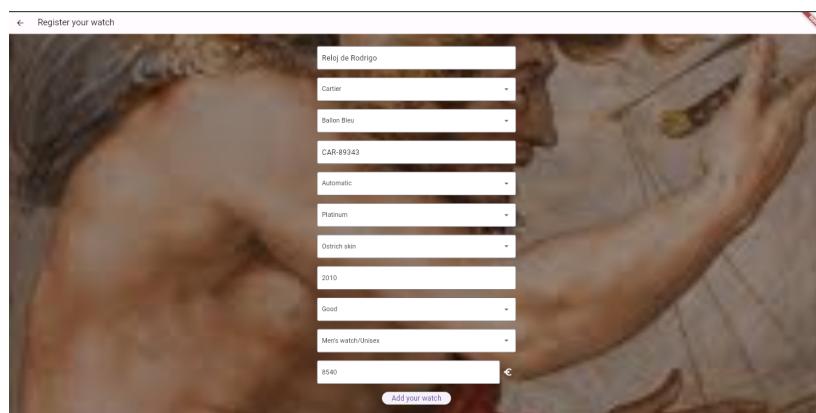


Figura E.6: Registro de un reloj

Para que el usuario registre de manera satisfactoria un reloj, debe completar los campos siguiendo los siguientes criterios:

1. El campo *watch nickname* debe ser completado con un nombre descriptivo a juicio del usuario para el reloj.
2. Los campos de tipo desplegable deben ser completados eligiendo uno de los que se dan como opción.
3. El campo *yop* debe contener solo números enteros positivos.
4. El campo precio debe contener solo números enteros positivos.
5. Los campos que sean acompañados de un asterisco imponen obligatoriedad de ser completados para el registro de un reloj.

A la derecha del campo *price* se aprecia un botón con el símbolo €. Si el usuario pulsa sobre ello y no ha completado los campos *brand*, *model*, *yop* y *condition*, la aplicación lanzará un mensaje marcando que los campos necesarios para la predicción del precio del reloj se encuentran vacíos. Si los campos están completos y se pulsa sobre ello, la aplicación lanzará un mensaje indicando el precio del reloj predicho tras la consulta al modelo creado y explicado en anteriores apartados.

Con todo listo, si pulsamos sobre el botón *Add your watch*, la aplicación lanzará un mensaje de verificación de la creación del reloj y podremos volver a la vista del listado de relojes pulsando sobre la flecha de la parte superior izquierda.

Si nuestra necesidad es editar y/o eliminar un reloj de nuestra lista, pueden darse dos casos en cuanto al bloqueo de botones:

1. Si el reloj no se encuentra en subasta, los botones del lapicero y de la papelera no estarán bloqueados y podrán realizarse las acciones de edición y eliminación respectivamente.
2. Si el reloj se encuentra en subasta, ambos botones estarán deshabilitados y no será posible realizar ninguna de las dos acciones.

Referenciando a la primera situación, si el usuario desease eliminar el reloj, simplemente se lanzaría un mensaje de confirmación y el reloj no aparecería de nuevo en el listado. Si el usuario desease editar la información del reloj, se pulsaría sobre el botón con forma de lapicero y llevaría a la ventana representada en la Figura E.7.

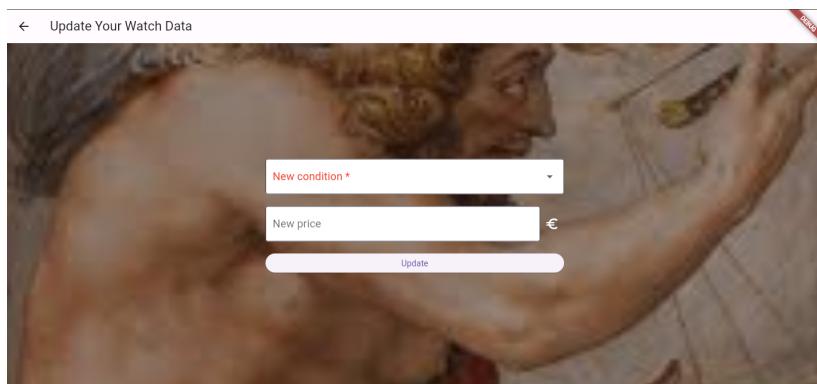


Figura E.7: Edición de un reloj

Hasta aquí toda la información relativa a las acciones relacionadas con un reloj. Volviendo a la página principal, el segundo botón empezando desde arriba nos llevaría a la Figura E.8 denominada listado de subastas.

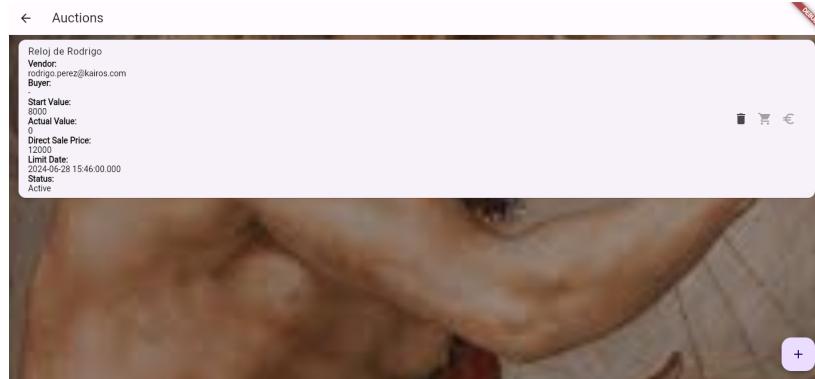


Figura E.8: Página listado de subastas

Una vez pulsado sobre el botón y situados en la página del listado de subastas, el usuario podrá crear una subasta pulsando sobre el botón con símbolo + situado en la parte inferior derecha de la vista. Este llevará a la ventana representada en la Figura E.9 denominada creación de subasta.

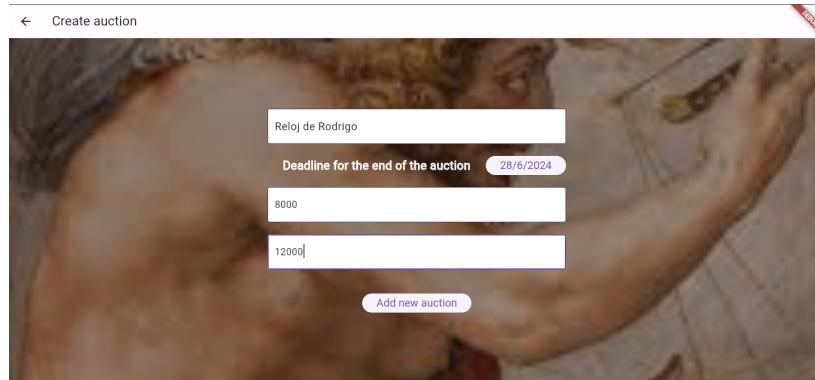


Figura E.9: Creación de una subasta

Para que la creación de la subasta se lleve a cabo de manera satisfactoria, deben cumplirse las siguientes condiciones:

1. El campo *watch nickname* debe completarse con el watch nickname de un reloj existente.
2. Debe introducirse una fecha mayor a la actual.

3. Los campos *minimum value* y *direct sale price* deben contener números enteros positivos.
4. El campo *minimum value* debe ser menor que el *direct sale price*.
5. Todos los campos son obligatorios.

Si no se cumpliese alguna de estas condiciones, la aplicación lanzaría mensajes informativos explicando donde se encuentra el error. Si se ha cumplimentado todo de manera satisfactoria, al pulsar en el botón *Add new auction*, la aplicación no lanzaría un mensaje de confirmación de que la subasta se ha creado correctamente. El usuario podría volver a la vista de listado de subasta pulsando en la flecha situada en la parte superior izquierda de la ventana.

Por último, se puede apreciar en la Figura E.7 como cada una de las tarjetas que engloban las subastas disponen de tres botones:

1. **Papelera:** permite eliminar una subasta. Solo podrá estar disponible para el creador de la subasta y solo podrá borrarla si aun nadie ha aplicado a ella. Si alguien hubiera aplicado, el botón se deshabilitaría. Para los demás usuarios, este botón aparecerá bloqueado.
2. **Carrito de compra:** permite comprar de manera directa el reloj. Estará disponible para cualquier usuario a excepción del creador de la subasta, quien verá el botón deshabilitado. Si la cantidad a pagar es superior al monedero del usuario o a la suma de todas sus pujas, la aplicación respondería con un mensaje informativo.
3. **Símbolo de euro:** permite pujar a la subasta del reloj. Estará disponible para cualquier usuario a excepción del creador de la subasta, quien verá el botón deshabilitado. Para pujar, la aplicación lanzará un modal donde debe introducir la cantidad a pujar. Si la cantidad a pagar es superior al monedero del usuario o a la suma de todas sus pujas, la aplicación respondería con un mensaje informativo. Otras condiciones a tener en cuenta dentro de este punto son:
 - a) No puede pujarse menos que el valor inicial.
 - b) No puede pujarse menos que el valor actual de la subasta.
 - c) No puede pujarse por encima del precio de venta directa de la subasta.

Un ejemplo de cómo otro usuario ve las subastas de otros usuarios se ve reflejado en la Figura E.10

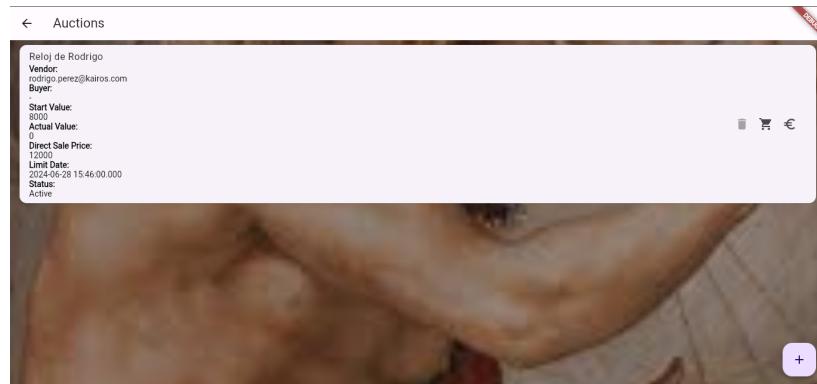


Figura E.10: Vista de subasta respecto a otro usuario

Apéndice F

Anexo de sostenibilización curricular

En la reunión realizada por la Naciones Unidas el 25 de septiembre de 2015 según [?], “los líderes mundiales adoptaron un conjunto de objetivos globales para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos como parte de una nueva agenda de desarrollo sostenible.”

La finalidad de cada objetivo es que toda persona habitante del planeta ponga de su parte y trate de realizar cualquier actividad teniendo en cuenta los objetivos estipulados en la reunión. De esta forma, se prevé que en 15 años se consiga un desarrollo de la sostenibilidad que repercutirá en las generaciones venideras.

Desde Kairos, estamos muy concienciados con la causa y creemos firmemente que si todos aportamos nuestro pequeño grano de arena, podremos conseguir de esta sociedad un mundo mejor. Para ser lo más claro posible, se expone como Kairos vela por alcanzar los objetivos exponiendo ejemplos de parte de ellos.

F.1. Trabajo Decente y Crecimiento Económico

El ODS 8, más conocido como Trabajo Decente y Crecimiento Económico, busca “promover el crecimiento económico inclusivo y sostenible, el empleo y el trabajo decente para todos.”

Kairos sigue la filosofía de dicho objetivo, pues la venta de relojes en la aplicación da lugar a la creación de oportunidades económicas tanto para compradores como para vendedores. Además, si nos centramos en la visión de los vendedores, estos usuarios generan fuentes de ingresos revendiendos los productos adquiridos como nuevos, provocando así el fomento del emprendimiento, y accediendo así a un mercado mucho más amplio y con precios de venta justos (sobre todo gracias a la predicción de precios de los relojes que realiza el modelo de predicción creado).

F.2. Industria, Innovación e Infraestructura

El ODS 9, más conocido como Industria, Innovación e Infraestructura, busca “construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación.”

La implementación de nuestra aplicación es un claro ejemplo de innovación tecnológica que permite acceder a los mercados (en este caso digital) de una manera más eficiente y sostenible. Además, Kairos puede convertirse en un ejemplo para mercados de segunda mano siendo una inspiración para ellos y consiguiendo su modernización.

F.3. Reducción de las Desigualdades

El ODS 10, más conocido como Reducción de las Desigualdades, estipula que “la desigualdad amenaza el desarrollo social y económico a largo plazo, frena la reducción de la pobreza y destruye el sentido de realización y autoestima de las personas”.

Kairos se caracteriza por ser una aplicación de venta de relojes de segunda mano donde cualquier persona de cualquier grupo socioeconómico puede participar en dicho mercado. La idea es llevar el mundo de los relojes a todo el planeta, consiguiendo que sectores de la sociedad que no puedan comprar un reloj nuevo, tengan acceso a productos de calidad con precios mucho más reducidos y cambiantes.

F.4. Promoción y Consumo Responsables

El ODS 12, más conocido como Promoción y Consumo Responsables, busca “garantizar modalidades de consumo y producción sostenibles”.

La aplicación que se presenta en este proyecto fomenta la reutilización y la extensión del ciclo de vida de los productos, en este caso relojes. Además, debido a la facilidad que supone la creación de subastas, la demanda de productos nuevos disminuye, y por tanto, reduce la producción industrial que supone un gran impacto ambiental.

Con todo esto conseguimos reducir el consumo de recursos naturales, generación de residuos electrónicos y la emisión de gases que puedan fomentar el efecto invernadero.

F.5. Alianzas para Lograr los Objetivos

El ODS 17, más conocido como Alianzas para Lograr los Objetivos, busca “revitalizar la alianza mundial para el desarrollo sostenible”.

Desde Kairos sabemos que esto es fundamental para crecer como aplicación. La venta de nuestros productos supone un transporte necesario para llevar el reloj del vendedor al comprador. La idea es colaborar con distintas empresas de logísticas que aseguren realizar sus tareas siguiendo unas prácticas ecológicas.