TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica
Superior de Jaén

Estudio y desarrollo de un prototipo de aplicación web para seguimiento de usuarios con diabetes

Autor: Diego Corral Fernández

Grado: Ingeniería Informática

Director: José Ramón Balsas Almagro
Departamento del director: Lenguajes y Sistemas Informáticos

Fecha: 24/05/2024

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mi tutor, José Ramón Balsas Almagro, por brindarme su invaluable orientación y apoyo durante la realización de este Trabajo Final de Grado.

Agradezco especialmente a mi primo Pedro y a mi abuelo Antonio por su generosa colaboración en la definición de los diversos requisitos que han sido fundamentales para el desarrollo de esta aplicación.

No puedo dejar de mencionar el incondicional apoyo de mis padres y hermano, quienes han estado a mi lado durante todo este proceso de aprendizaje, brindando apoyo en los momentos más difíciles.

Finalmente, deseo expresar mi más sincero agradecimiento a todos aquellos que han formado parte de mi entorno durante mi trayectoria en la Universidad de Jaén. En particular, quiero reconocer a mi grupo de la carrera "Pary", con quienes he compartido gratos momentos durante estos cinco años, a mis compañeros de piso que han hecho más amena esta aventura, y por último, pero no menos importante, a mi gran amiga Sandra, quien ha sido un apoyo incondicional en esta etapa inolvidable de mi vida.

La constante presencia y el apoyo brindado por todos ustedes han sido un pilar fundamental en este camino académico y personal.

¡Simplemente, gracias!

Ficha del Trabajo Fin de Título	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Ingeniería del Software
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	**Diego Corral Fernández**
Fecha de asignación	04/10/2023
Descripción corta	El proyecto se centra en realizar un estudio de las necesidades de las personas que viven con diabetes y presentar un prototipo de aplicación diseñado para brindarles apoyo y autonomía en su vida diaria La aplicación ofrecerá una variedad de funciones destinadas a mejorar la gestión de la alimentación, el nivel de actividad física, la medicación, los recordatorios y la información sobre la enfermedad. Estas características están diseñadas para proporcionar un acompañamiento integral y facilitar la toma de decisiones saludables. Para garantizar la relevancia y efectividad de la aplicación, se ha llevado a cabo un análisis exhaustivo de los requisitos, con la participación activa de personas con diabetes, las cuales han proporcionado valiosa información orientativa. El desarrollo de la aplicación se ha realizado utilizando el framework multiplataforma *Flutter*. Esta elección no solo ha permitido una rápida agilidad en el desarrollo, sino que también sienta las

	bases para una expansión futura.

NORMAS APLICADAS EN ESTE DOCUMENTO

Locales	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
Nacionales e Internacionales	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Índice

Ilustraciones

Tablas

1.-Introducción

1.1.- Motivación

Este Proyecto de Trabajo de Fin de Grado encuentra su razón de ser en la conjunción de la imperante necesidad de optimizar la gestión diaria de la diabetes y una experiencia personal reveladora. La esencia de esta propuesta se nutre de la ambición de proporcionar una solución tangible y práctica a un desafío de salud que no sólo me concierne profesionalmente, sino que también toca fibras personales profundas.

La concepción de desarrollar una aplicación multiplataforma para el seguimiento de personas diabéticas tuvo su génesis en una tarde de verano, inmerso en una conversación reveladora con mi primo, quien enfrenta diariamente los retos que impone la diabetes. Durante nuestro diálogo, compartió las limitaciones y carencias de las aplicaciones de monitoreo disponibles, convirtiendo esa experiencia personal en la llama que aviva la motivación para emprender este proyecto.

Dicha motivación que impulsa este trabajo va más allá de aplicar conocimientos académicos, representa la oportunidad de convertir la empatía inspirada por experiencias reales en una contribución palpable. El propósito fundamental es evidente: proporcionar a las personas diabéticas una herramienta que no solo recopile datos, sino que también responda a sus necesidades específicas y cotidianas.

La accesibilidad y la simplicidad son los fundamentos sobre los cuales se erige este proyecto, guiados por las vivencias compartidas durante la charla con mi primo. Este Trabajo de Fin de Grado no es meramente una respuesta a los requisitos académicos, es un proyecto imbuido de una conexión personal que aspira a generar un impacto positivo en la vida de las personas afectadas por la diabetes.

1.2.- Objetivos

Estudiaremos el día a día de las personas que padecen diabetes para comprender mejor sus necesidades y ofrecerles soluciones que mejoren su calidad de vida.

Para ello vamos a implementar un sistema informático orientado a la utilización de las arquitecturas, principios y metodologías de desarrollo web, donde seleccionaremos un entorno de desarrollo basado en tecnologías web que utilizaremos para desarrollar un prototipado de dicha aplicación que cumpla con las necesidades que se hayan determinado.

1.3.- Metodología

La metodología hace referencia al conjunto de procedimientos racionales utilizados para alcanzar el objetivo o la gama de objetivos que rige una investigación científica, una exposición doctrinal o tareas que requieran habilidades, conocimientos o cuidados específicos. **[1]**

Las metodologías de desarrollo de *software* son el conjunto de técnicas y métodos que se utilizan para diseñar una solución de *software* informático. **[2]**

Es importante establecer una metodología previamente al inicio de un proyecto, con el fin de evitar futuros retrasos en entregas, errores que pueden no ser reparados o un consumo de recursos excesivo. De esta forma, conseguiremos obtener mejores resultados, resolver problemas organizativos a tiempo y obtener sencillez a la hora del desarrollo.

Actualmente existen dos tipos de metodologías para el desarrollo de *software* (tradicionales y ágiles), las cuales serán analizadas a continuación.

1.3.1.- Metodologías tradicionales

Las metodologías tradicionales son las primeras en su tipo, que surgen como guía para garantizar la creación de un producto con un nivel de calidad. Este método se realiza de forma lineal, es decir, que cada etapa está ligada a la etapa anterior. **[3]**

Se caracteriza porque todas las fases se realizan de forma secuencial, es decir, que las etapas se llevan a cabo una detrás de otra, pero teniendo en cuenta que cada una debe de estar finalizada antes de que se empiece una nueva.

En la **ilustración 1**, se presenta las fases del modelo

Ilustración 1: Metodología tradicionales (Diego Calvo)

Es útil escoger este tipo de metodología cuando se conocen desde un inicio los detalles del producto, ya que se puede estimar la información. Además, su uso es ideal cuando los requisitos iniciales no cambian y su entorno es estable.

1.3.2.- Metodologías ágiles

Las metodologías ágiles surgen como una alternativa a las metodologías tradicionales, se caracterizan por ser rígidas, mecánicas, secuenciales y puntuales.

Más que una metodología, lo que establecieron fue un marco teórico que permitiera acortar los tiempos de desarrollo, eliminar la incertidumbre, mejorar la eficiencia del equipo y obtener un resultado de calidad, con el propósito de brindar la mayor satisfacción posible al cliente, a través de la interacción con los avances obtenidos en cada etapa y la retroalimentación durante la construcción del producto.

Se caracteriza porque los objetivos se acuerdan desde el inicio con el cliente, pero se acepta que durante el desarrollo del proyecto aparezcan cambios. Dichos cambios se utilizan como método de aprendizaje para futuros proyectos.

Poseen un enfoque eficaz para el desarrollo de *software* ya que permite a los equipos proporcionar a los clientes resultados de manera rápida y continua, adaptarse de forma rápida a los cambios, colaborar estrechamente y mejorar continuamente el proceso de desarrollo.

En la **ilustración 2**, se presenta las fases que contiene

Ilustración 2: Ciclo de la vida general de las metodologías ágiles (*Jucaripo*)

Según *Braintrust Consulting Group* **[4]**, la mayoría de las empresas emplean la metodología ágil SCRUM, seguida de Kanban, SAFe y de otras metodologías, como lo podemos ver en la **Ilustración 3**.

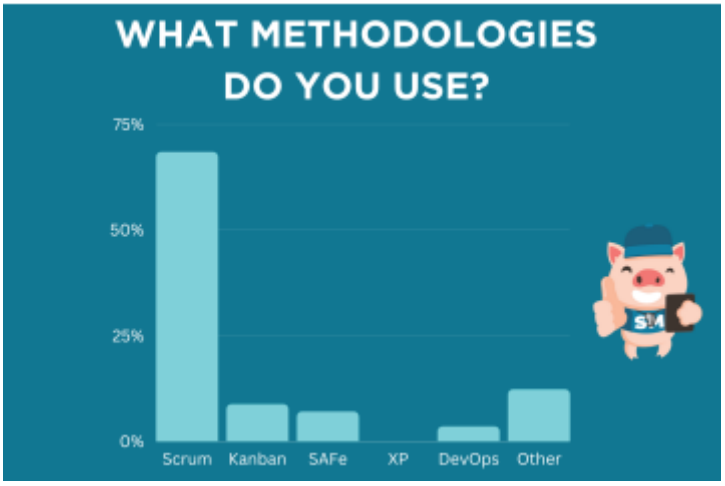


Ilustración 3: Métodos ágiles más usadas en 2023 (Braintrust Consulting Group)

1.3.3.- Metodologías tradicionales vs ágiles

Según *State of Agile*, los equipos de trabajo de la mayoría de las organizaciones utilizan una metodología ágil para el desarrollo de sus proyectos, aunque también hay que destacar que podemos encontrar un pequeño porcentaje de empresas que usan metodologías tradicionales e incluso híbridas, que son una mezcla de metodologías ágiles y tradicionales.

En la **ilustración 4**, se muestra el porcentaje con el que se están utilizando las metodologías más destacadas actualmente.

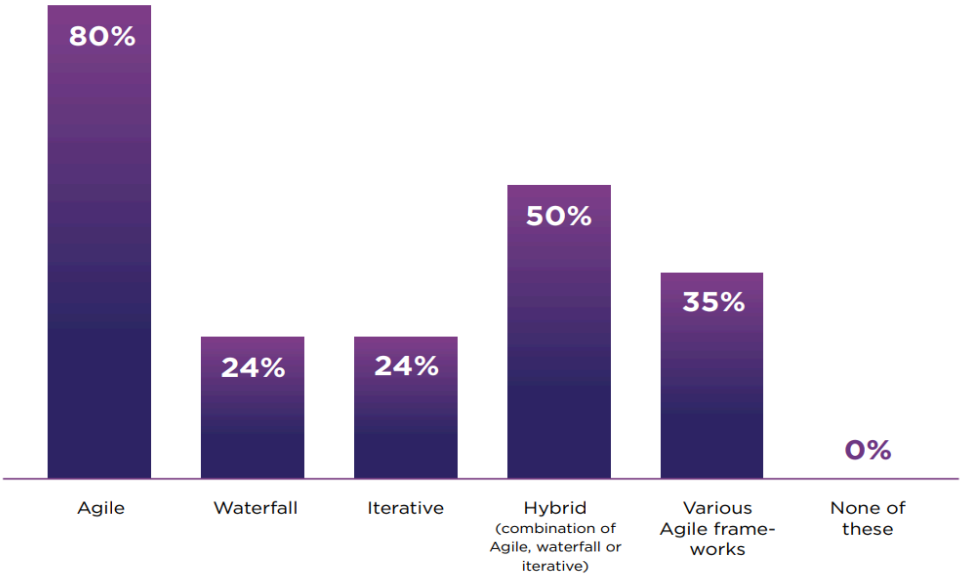


Ilustración 4: Metodologías utilizadas actualmente (State of Agile)

1.3.4.- SCRUM vs Kanban

SCRUM es una metodología ágil concebida para abordar proyectos complejos que requieren adaptación constante. Se fundamenta en ciclos breves de desarrollo, conocidos como *sprints*, que suelen tener una duración de aproximadamente cuatro semanas en lugar de entregar el proyecto en una única fase. Durante estos *sprints*, el equipo de desarrollo se enfoca en elementos específicos del backlog del producto. Esta estructura facilita la flexibilidad necesaria para ajustarse a los cambios que puedan surgir a lo largo del proceso de desarrollo.

Se basa en tres conceptos básicos, adaptación, transparencia e inspección.

Kanban, por su parte, se enfoca en optimizar los procesos y garantizar una entrega constante de valor al cliente. Pone en énfasis la visualización clara del flujo de trabajo, la restricción de la cantidad de trabajo en curso y la búsqueda constante de mejoras en el proceso.

Esta metodología se fundamenta en cinco principios esenciales que abarcan la visualización, la priorización, la mejora continua, el liderazgo en todos los niveles y la garantía de calidad.

Las labores se ejecutan según una lista de tareas o elementos que fluyen a través de un tablero visual. Cada tarea se asigna a una etapa específica del flujo de trabajo y se prioriza en base a las necesidades del cliente. Conforme se finalizan las tareas, se desplazan hacia la siguiente fase del proceso.

Desde un enfoque práctico, estos son tableros virtuales que desglosan el flujo de trabajo en columnas. Comúnmente, estas columnas representan las fases secuenciales de tareas pendientes, tareas en progreso y tareas completadas, en ese orden. Puedes ver un ejemplo visual en la **ilustración 5**.

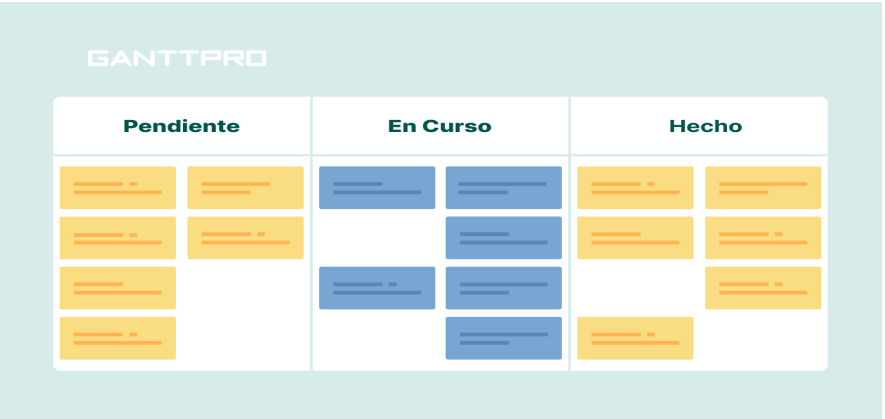


Ilustración 5: Flujo de trabajo en Kanban (Ganttpro)

A diferencia de SCRUM, en el marco de Kanban, el trabajo no se organiza en *sprints* y carece de reuniones prescritas. La ejecución se adapta conforme a la capacidad del equipo y se ajusta en consonancia con las lecciones aprendidas a lo largo del proceso. El equipo tiene la flexibilidad de reunirse según la necesidad para revisar el progreso y abordar cualquier problemática o identificar oportunidades de mejora.

Entre las ventajas más destacables de Kanban podemos hablar de:

- Es una metodología visual y sencilla
- Optimiza proceso
- Aumenta la productividad
- Evita la sobrecarga de trabajo
- Incrementa la sostenibilidad del negocio

Al tratarse de una metodología apta para proyectos individuales, como es el presente caso, se valida que este modelo se adapta de manera óptima, buscando alcanzar un equilibrio coherente entre los distintos elementos de la aplicación a desarrollar.

2.- Análisis

2.1.- Análisis preliminar

Hoy en día, las personas diabéticas a menudo realizan el seguimiento de su condición mediante métodos tradicionales **[5]**, cómo llevar un registro manual en papel o depender exclusivamente de las recomendaciones de sus profesionales de la salud. Sin embargo, estas prácticas pueden resultar limitadas en cuanto a la precisión, la accesibilidad y la capacidad de proporcionar información en tiempo real.

Al estudiar otras soluciones afines al proyecto, como *Diabetes:M*, *One Drop* y *FreeStyle*, *Diabetes Control Insul Lock*, las cuales son algunas de las aplicaciones móviles más punteras hoy en día en este campo (disponibles tanto para *Android* como para *iOS*), se ha descubierto conceptos que pueden ser buenos para perfeccionar mi propuesta:

2.1.1.- Diabetes:M [1]

Tras un breve análisis y recopilación de opiniones de usuarios sobre Diabetes:M, se ha destacado que su formulario de registro es uno de los más completos, abarcando casi todos los parámetros necesarios para un funcionamiento eficiente, como podemos observar en las **ilustraciones 6-10**, donde se muestra los diferentes campos para poder registrar los datos personales y sanitarios. No obstante, se observa una carencia de orden lógico en la disposición de la interfaz. Comenzado con el registro de los datos fundamentales para después abordar la información personal del usuario y finalmente regresar a datos esenciales.

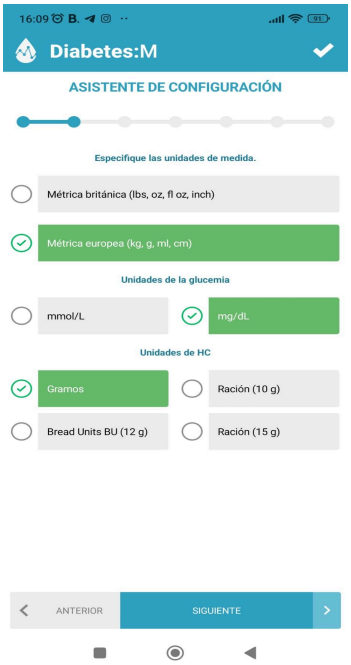
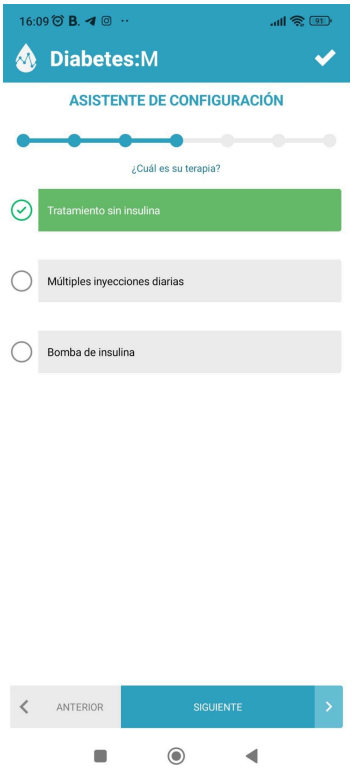


Ilustración 6: Registro de unidades de medida en Diabetes:M

Ilustración 7: Registro de datos personales en Diabetes:M

1. https://play.google.com/store/apps/details?id=com.mydiabetes&hl=es&gl=US (Mayo, 2024)



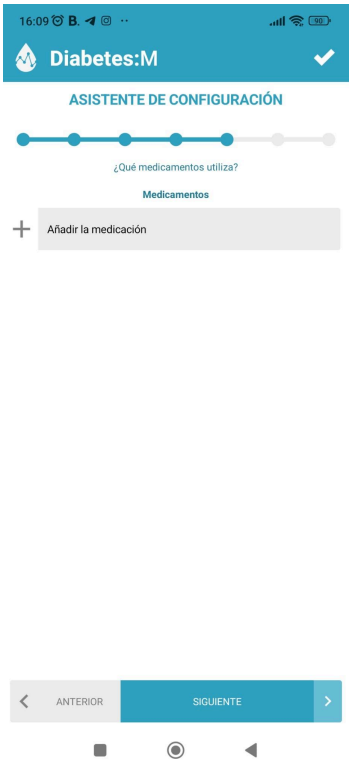


Ilustración 8: Registro de tratamiento en Diabetes:M

Ilustración 9: Registro de medicamentos en Diabetes:M

Ilustración 10: Registro de glucemia en Diabetes:M

2.1.2.- FreeStyle [2]

FreeStyle ha servido de guía para desarrollar la visualización de lecturas e informes, puesto que ofrece una personalización del gráfico bastante eficiente para poder representar los diferentes datos. Esta capacidad se ha considerado como un punto fuerte, ya que si el usuario que hace uso del producto no está familiarizado con la diabetes, le va a costar más comprender la gráfica de la **ilustración 12** que la **ilustración 11**, que es más intuitiva.

Otro aspecto que se ha encontrado bastante interesante es la capacidad de comunicarse con otros dispositivos médicos como puede ser el sensor de *FreeStyle*, que es utilizado para realizar las diferentes lecturas.

Por otro lado, se ha identificado un aspecto desfavorable en el uso de la aplicación. Después de una semana de uso, se ha notado un nivel significativo

de complejidad. Es relevante destacar que esta aplicación es desarrollada por una empresa líder en el sector y es ampliamente utilizada por usuarios con diabetes.

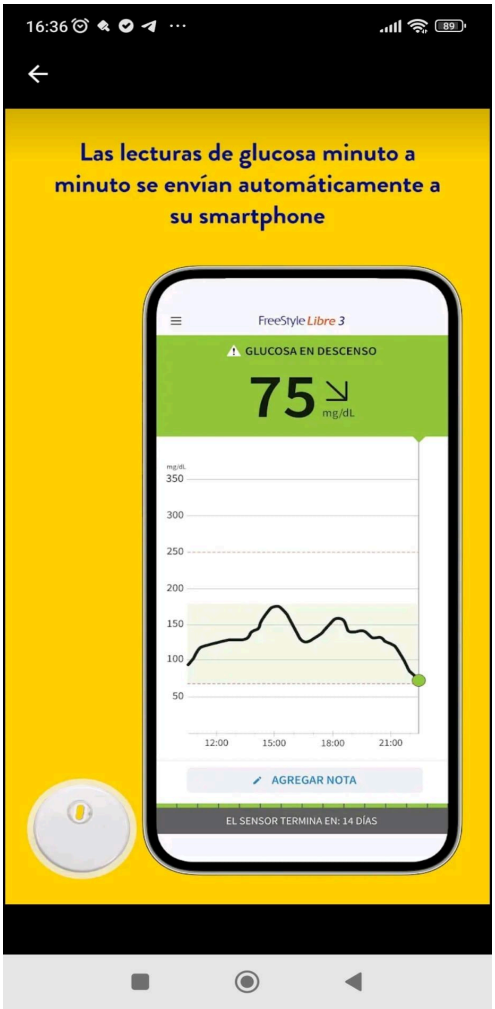


Ilustración 11: Lectura de glucosa

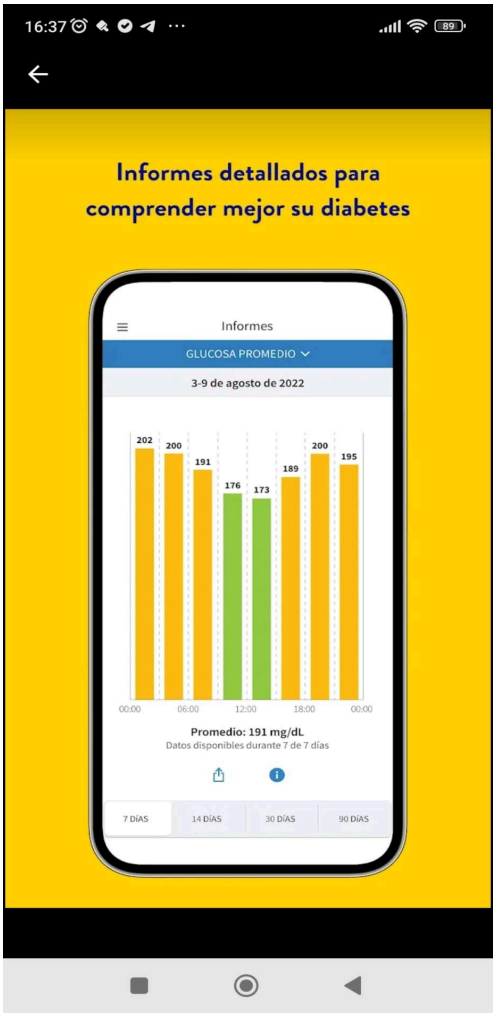


Ilustración 12: Informes detallados

2. https://play.google.com/store/apps/details?id=com.freestylelibre.app.us&hl=es&gl=US (Mayo, 2024)

2.1.3.- One Drop [3]

Tras el análisis de la aplicación *One Drop* se ha encontrado realmente útil la idea de mostrar al usuario artículos o consejos sobre cómo mejorar su calidad de vida, como se puede ver en la **ilustración 13**. Aunque esta funcionalidad se considere importante en aplicaciones de este estilo, no se ha terminado de percibir como totalmente necesaria, puesto que, la presentación de esta

información adicional puede desviar la atención de los datos esenciales, como las lecturas.

Ilustración 13: Consejos en One Drop

2.1.4.- Diabetes Control Insulclock [4]

Se ha encontrado la configuración de recordatorios y el registro de las comidas, como se puede observar en las **ilustraciones 14 y 15**. Estás funcionalidades son realmente imprescindibles en las aplicaciones de este estilo, puesto que son datos que pueden afectar a los niveles de azúcar del usuario, como son las comidas y las unidades de insulinas. Sin embargo, se ha pensado que la interfaz de registro de la comida debería de ser más

3. https://play.google.com/store/apps/details?id=today.onedrop.android&hl=es&gl=US (Mayo, 2024)

sencilla e intuitiva para que cualquier usuario pueda realizar los registros correctamente.

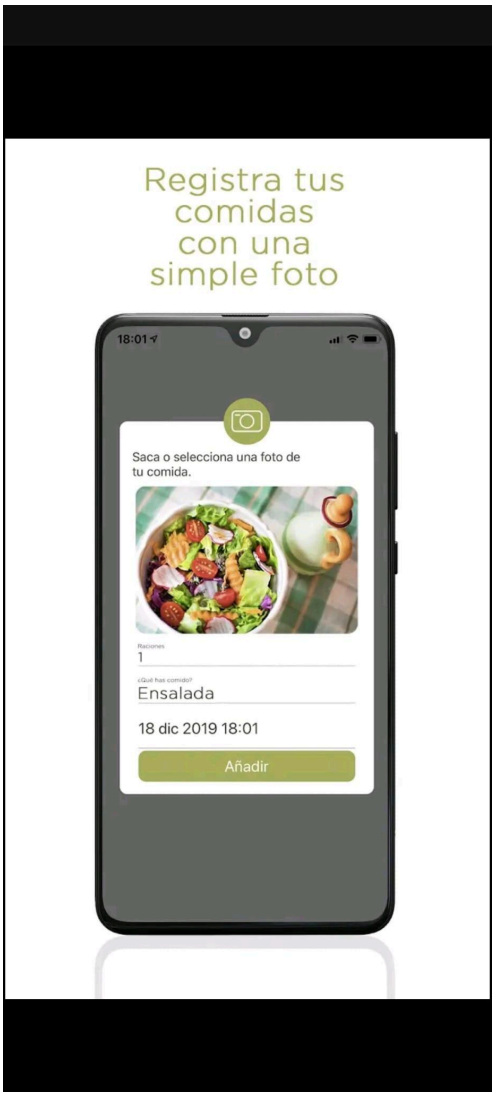


Ilustración 14: Registro de alimentos en Insulclock

Ilustración 15: Configuración de alarmas en Insulclock

Después de realizar un exhaustivo análisis, recopilar opiniones de usuarios y realizar pruebas, se han identificado algunas características esenciales para mejorar nuestro prototipo. Se considera crucial un formulario de registro detallado, ya que la eficiencia de la aplicación depende en gran medida de estos datos recopilados. Además, se busca una visualización de datos atractiva e intuitiva para facilitar la correcta interpretación de la información.

Por otro lado, se planea incluir consejos que ayuden a los usuarios en su día a día como hemos visto en One Cloud, así como un sistema de registro de actividades físicas y alimenticias para un funcionamiento óptimo. También se quiere destacar la importancia de un sistema de notificaciones para recordar a los usuarios la toma de medicamentos o proporcionar alertas.

4. https://play.google.com/store/apps/details?id=com.insulclock.insulclockAndroid&hl=es_419&gl=US (Mayo, 2024)

En la investigación, se ha notado la ausencia de funciones como el cálculo automático de unidades de insulina y avisos de emergencia en muchas aplicaciones existentes. Considero que estas características son de gran utilidad y, por lo tanto, decidimos incluirlas en nuestro prototipo. Creemos que estas adiciones pueden proporcionar una ventaja significativa sobre otras soluciones disponibles en el mercado.

Además de todo lo comentado, se ha decidido desarrollar la aplicación para las plataformas móviles más utilizadas hoy en día (*Android* e *iOS*), ya que actualmente son las que cuentan con más usuarios activos.

2.1.5.- Estudio de tecnologías

Para el desarrollo de la aplicación multiplataforma, se ha optado por elegir una solución basada en tecnologías web en lugar de otras opciones como aplicaciones nativas. Esta elección se basa en la accesibilidad multiplataforma que ofrece, permitiéndonos acceder de manera uniforme desde cualquier dispositivo. Además, la implementación de actualizaciones y mejoras se puede realizar de manera centralizada, evitando la necesidad de realizar actualizaciones individuales para cada dispositivo. Esto nos brinda consistencia en las funcionalidades y facilita la descarga e instalación en los dispositivos donde se utilizará la aplicación. **[6]**

2.1.5.1.- Front - End

Para el desarrollo de la parte front-end podemos encontrar diversas opciones que nos permiten desarrollar aplicaciones multiplataforma, como son:

React Native nos ofrece una solución rentable para el desarrollo de aplicaciones multiplataforma (*Android*, *iOS, web*…) al proporcionar un código único. Al hacer uso del lenguaje de programación JavaScript, atrae a muchos desarrolladores, lo que provoca que tenga una comunidad muy activa. Otro fuerte de las aplicaciones desarrolladas con *React Native* es que exhiben un alto rendimiento que se puede comparar con el de una aplicación nativa.

Sin embargo, posee algunas desventajas que hace que no sea el framework más utilizado por los desarrolladores. A pesar de su sólida comunidad, el hecho de que se encuentre en una fase beta puede implicar limitaciones en ciertos aspectos como puede ser en la depuración. Además, la dependencia exclusiva de Facebook, el creador, plantea inquietudes sobre su soporte. Por otro lado, tenemos el tema de la seguridad, que es un aspecto que también preocupa, puesto que al ser un framework que hace uso JavaScript deja mucho que desear. **[7]**

Flutter, otro framework para realizar aplicaciones multiplataforma. *Flutter* nos muestra actualizaciones de nuestro

proyecto en tiempo real, lo que nos permite encontrar y corregir errores mientras se está ejecutando el código. Un rendimiento nativo gracias al motor gráfico Skia, que nos permite un desarrollo rápido y bien optimizado. Ofrece un proceso de aprendizaje rápido y sencillo, ya que hace uso de un nuevo lenguaje llamado Dart que está basado en Javascript.

Al igual que *Flutter* se considera como una de las mejores opciones para el desarrollo, también contiene ciertos inconvenientes que se debe tener en cuenta a la hora de realizar los proyectos. A la hora de manejar archivos pesados los desarrolladores pueden enfrentarse a grandes dificultades. Puede presentar problemas con el sistema operativo *iOS*, puesto que este framework fue desarrollado por la compañía *Google*, lo que les da una gran ventaja a los dispositivos *Android*. Por último hay que destacar que al tratarse de un lenguaje relativamente reciente no posee una gran comunidad como lo posee *React Native*. **[8]**

Otra herramienta que podemos destacar para el desarrollo de aplicaciones multiplataforma es *Angular*, el cual nos ofrece un alto rendimiento ya que solo carga los componentes necesarios en cada momento. Un gran mantenimiento gracias a su arquitectura basada en componentes, lo que nos permite una estructura modular y fácilmente mantenible. Una alta productividad debido al uso del CLI (*Command Line Interface*) que nos permite generar rápidamente componentes, servicios y otros elementos necesarios. También hay que destacar que cuenta con una gran comunidad de usuarios.

Sin embargo cabe destacar que *Angular* requiere de una curva de aprendizaje que necesita una gran cantidad de tiempo y dedicación. Una flexibilidad limitada con otros lenguajes de programación y un alto peso de aplicación debido a su estructura modular y a todas las características que ofrece. **[9]**

Para terminar la parte del front-end, como última herramienta encontramos *Swift*, que es una tecnología que nos ofrece un desarrollo rápido y eficiente gracias a su sintaxis clara y sencilla, junto con la interferencia de tipos, el flujo expresivo y la prevención de errores mejorada. Nos brinda una experiencia de usuario mejorada ya que nos permite la creación de productos más impactantes e innovadores.

Pero también nos encontramos ante una herramienta bastante joven (fundada en 2014) que provoca que la comunidad no sea muy grande, con una curva de aprendizaje difícil puesto que *Swift* está creciendo de forma exponencial y cada vez son más los recursos que deben aprender los desarrolladores para hacer uso de framework. Pero sin embargo la gran desventaja que posee *Swift* es que es una herramienta que está pensada únicamente para el desarrollo de entornos *iOS* y *macOS*, lo que provoca ciertas limitaciones en el desarrollo de aplicaciones multiplataforma. **[10]**

2.1.5.2.- Back-end

Dejando de lado la parte front-end, nos adentramos en la back-end en la que encontramos herramientas como:

La herramienta *Firebase,* la cuál posee una licencia gratuita que hace uso de una base centralizada que es capaz de manejar las actualizaciones de datos en tiempo real entre los dispositivos sin ningún tipo de bloqueo. Posee un almacenamiento en la nube, un escalamiento automático y una *API* multiplataforma lo que nos permite realizar una instalación sencilla en nuestros proyectos. Otro punto importante de *Firebase* es que ofrece métodos ya implementados que nos puede servir de mucha utilidad, como por ejemplo el proceso de autenticación para los usuarios.

También encontramos algunas desventajas como el uso del lenguaje JSON en vez de SQL para el almacenamiento de los datos, tiene establecida una limitación de 100 conexiones y 1 GB de almacenamiento para aquellos usuarios que hacen uso de la versión gratuita, lo que es un gran inconveniente si nuestra aplicación recoge una gran cantidad de datos. **[11]** Sin embargo, estos inconvenientes se pueden solucionar pagando la licencia Plan *Blaze*. Donde podremos obtener un mayor espacio de almacenamiento y un mejor almacenamiento, cosa que en el plan *Spark* está muy limitado.

Otra herramienta bastante utilizada es *Django*, que es un *framework* que nos ofrece una seguridad bastante robusta debido a que tapona automáticamente las vulnerabilidades que los desarrolladores podrían pasar por alto de forma involuntaria. Una buena adaptación a proyectos con un alto tráfico de datos, debido a que gestiona de forma rápida todos los datos, reduciendo el tiempo de carga. También nos facilita el mapeo de sitios web y nos da la posibilidad de generar enlaces y URL dinámicos basados en palabras clave específicas.

Django está pensado para ser implementado en proyectos de gran tamaño lo que provoca que su implementación en aplicaciones que no tengan un alto nivel de tráfico de datos no esté muy recomendado. Posee un entorno de trabajo muy monolítico, no permite gestionar simultáneamente varias solicitudes lo que podría provocar un colapso de la base de datos. Por último posee una curva de aprendizaje demasiado pronunciada lo que le hace muy difícil de ser manejable. **[12]**

Por último tenemos Spring Boot como *framework* para desarrollar el *back-end*. Ofrece un desarrollo fácil debido a que todas sus características tienen como objetivo este tipo de desarrollo. No necesita la implementación de archivos .war, ya que puede empaquetar aplicaciones ejecutables .jar. Posee un marco que viene con muchas características útiles para realizar testing, para soportar proyectos Maven o para ofrecer servidores web integrados en cada aplicación.

Pero por otro lado, encontramos que carece de control ya que crea muchas dependencias que no se terminan de utilizar que termina creando un archivo de implementación demasiado grande. El uso de *Spring Boot* no está

recomendado para proyectos de gran tamaño, si no que está pensado para aplicaciones pequeñas ya que muchos especialistas han argumentado que presenta un mejor rendimiento en aplicaciones pequeñas. Finalmente en ocasiones puede ser inflexible puesto que está enfocado a una solución única para cualquier problema planteado. **[13]**

2.2.- Propuesta de solución

La propuesta de solución consiste en desarrollar una aplicación multiplataforma destinada al seguimiento y gestión de la diabetes, con el objetivo de mejorar la calidad de vida de las personas afectadas por esta condición. La aplicación ofrecerá diversas funcionalidades clave para un manejo integral de la diabetes, abordando aspectos como el monitoreo de glucosa, registro de alimentos, seguimiento de actividad física, recordatorios de medicamentos, y más.

Para la parte *front-end* finalmente nos hemos decidido por el *framework Flutter* por los siguientes motivos:

1. Nos permite un desarrollo multiplataforma para *iOS*, *Android*, web y aplicaciones de escritorio desde un solo código base, lo que provoca un ahorro de tiempo y recursos bastante elevado.
1. Posee un alto rendimiento nativo ya que las aplicaciones se compilan a código nativo, lo que resulta en un rendimiento similar al de aplicaciones nativas.
2. Contiene una amplia gama de personalización de la interfaz de usuario gracias a su gran número de *widget* que los usa para construir las interfaces, lo que facilita la creación de diseños atractivos.
3. Nos permite realizar *Hot reload*, con esta función podemos ver los cambios en tiempo real durante el desarrollo, acelerando así el proceso de iteración y prueba.
4. Posee una gran comunidad y soporte de *Google*. *Flutter* cuenta con un gran número de desarrolladores activos y un respaldo sólido por parte del propio *Google*, lo que garantiza actualizaciones regulares, un soporte técnico y una gran cantidad de recursos educativos.
5. Tener una experiencia agradable y una sensación de comodidad tras haber realizado otro tipo de proyectos en ocasiones previas.

Para la parte del back-end nos hemos decidido por la tecnología de *Firebase* ya que está diseñada para ser de fácil uso, con una interfaz de usuario intuitiva, que nos facilita la configuración y gestión de los diferentes servicios. Nos permite el desarrollo de aplicaciones al proporcionar servicios listos para usar, como por ejemplo bases de datos en tiempo real, autenticación o almacenamiento.

Firebase se integra bien con diversas plataformas, incluyendo el *framework Flutter* para el desarrollo multiplataforma, lo que facilita la creación de aplicaciones que funcionan en diferentes sistemas operativos.

2.3.- Historias de Usuario

> Las historias de usuario son descripciones cortas y simples de una característica contada desde la perspectiva del usuario. Son herramientas ágiles que ayudan a comunicar y entender los requisitos del *software* de manera precisa y concisa. Son fundamentales en metodologías ágiles para planificar y priorizar el trabajo del equipo de desarrollo.

La elección de historias de usuario sobre casos de uso en metodologías ágiles se debe a su enfoque centrado en el usuario, flexibilidad para la entrega incremental y capacidad de iteración continua. Las historias de usuario, al redactarse desde la perspectiva del usuario, facilitan la comprensión y empatía con las necesidades reales. Además, su brevedad y adaptabilidad permiten ajustes rápidos a medida que se obtiene retroalimentación.

Las historias de usuario reciben los llamados *story points* o puntos de historia **(PH)**, que representa una estimación de la complejidad y esfuerzo que requiere para completarlas. En este caso se ha considerado una escala desde un punto de historia hasta diez. Donde uno es el mínimo, es decir, la historia de usuario que requiere menos esfuerzo y diez es el máximo.

Por otro lado se ha usado la metodología *MoSCoW* para realizar una aproximación de la priorización de las tareas. Dado que no contamos con un periodo de tiempo indefinido, deberemos descartar aquellas historias de usuario que no se consideren realmente relevantes para nuestro prototipado.

El método MoSCoW es una herramienta para definir la priorización de tareas, y que tiene como objetivo lograr el entendimiento de los integrantes del equipo sobre la importancia que le asignan a cada requisito.

Este método de priorización tiene un papel primordial en las metodologías ágiles. En un proyecto ágil, es esencial comprender la importancia de las diferentes cosas como, requisitos, tareas, productos, etc., debido a que el tiempo es un recurso fijo.

El acrónimo en inglés *MoSCoW*, está formado por cuatro categorías las cuales se derivan de la primera letra que conforman dicha palabra, mientras que las letras “o” facilitan la pronunciación del acrónimo, que significa **[14]**:

- *Must-have* (Debe tener)
- *Should-have* (Debería tener)
- *Could-have*(Podría haber)
- *Won’t-have* (No lo haré)

Para elaborar las siguientes historias de usuario, nos hemos reunido con varios usuarios para identificar los requisitos que consideran esenciales para que una aplicación sea completa.

Hemos recopilado estas necesidades para asegurarnos de que la aplicación satisfaga adecuadamente las expectativas de los usuarios.

ID	1
TÍTULO	Registro
DESCRIPCIÓN	Como usuario nuevo, quiero poder crear una cuenta fácilmente para empezar a utilizar la aplicación.
ESTIMACIÓN	10
PRIORIDAD/VALOR	*Must-have*
DEPENDENCIAS	Ninguna
CONDICIÓN DE COMPLETITUD	● Cuando el usuario haga clic en el botón de registrar, se abrirá el formulario de registro. ● Debe realizarse en menos de 5 minutos.

ID	2
TÍTULO	Inicio de sesión
DESCRIPCIÓN	Como usuario, quiero tener la capacidad de iniciar sesión de manera segura para que la ley de protección de datos se aplique.
ESTIMACIÓN	10
PRIORIDAD/VALOR	*Must-have*
DEPENDENCIAS	1
CONDICIÓN DE COMPLETITUD	● Cuando el usuario introduzca sus credenciales o pulse los botones habilitados, podrá acceder a la aplicación. ● Se le deberá notificar en caso de que algún dato sea erróneo. ● Ofrecer diferentes formas de autenticación.

ID	3
TÍTULO	Visualización de datos
DESCRIPCIÓN	Como usuario, quiero realizar una visualización gráfica de mis estadísticas, para poder identificar patrones que me ayuden a ajustar mi estilo de vida y tratamiento, con opciones de personalización en la presentación de datos.
ESTIMACIÓN	8
PRIORIDAD/VALOR	*Must-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	● Debe proporcionar gráficos claros y comprensibles. ● Selección por fechas. ● Manejar grandes cantidades de datos.

ID	4
TÍTULO	Registro de lecturas de glucosa
DESCRIPCIÓN	Como usuario, quiero poder ingresar de forma rápida y fácil mis lecturas de glucosa. Para mantener un registro preciso de mis niveles de glucosa y visualizar los cambios a lo largo del tiempo
ESTIMACIÓN	10
PRIORIDAD/VALOR	*Must-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	● El registro debe ser rápido y sencillo. ● Debe ser visualmente intuitivo. ● Fácil de comprender.

ID	5
TÍTULO	Registro de alimentos
DESCRIPCIÓN	Como usuario, quiero poder registrar los diferentes tipos de alimentos que consumo a lo largo del día.
ESTIMACIÓN	7
PRIORIDAD/VALOR	*Must-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	● Contar con una base de datos extensa de alimentos. ● Obtener valores nutricionales. ● Registro rápido y sencillo. ● Visualmente intuitivo.

ID	6
TÍTULO	Funcionalidades de emergencia
DESCRIPCIÓN	Como usuario, quiero contar con funcionalidades de emergencia accesibles, para acceder rápidamente a ayuda médica en casos de que hiciera falta, con información personalizada según mis necesidades y contactos de emergencia predefinidos.
ESTIMACIÓN	5
PRIORIDAD/VALOR	*Could-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	● Deberá poseer un acceso fácil y rápido. ● Debe permitir la modificación de datos. ● Establecer contactos.

ID	7
TÍTULO	Recurso educativos
DESCRIPCIÓN	Como usuario, quiero acceder a recursos educativos o consejos sobre la diabetes de manera personalizada, para estar informado en todo momento.
ESTIMACIÓN	4
PRIORIDAD/VALOR	*Would-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	● Deberá ofrecer información actual.

ID	8
TÍTULO	Notificaciones
DESCRIPCIÓN	Como usuario, quiero poder recibir notificaciones personalizadas para recordar mi medicación, citas médicas u otros recordatorios.
ESTIMACIÓN	5
PRIORIDAD/VALOR	*Would-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	● El usuario podrá personalizar el periodo de notificación. ● El usuario podrá crear, editar o borrar notificaciones. ● La notificación deberá ser llamativa.

ID	9
TÍTULO	Sincronización de datos
DESCRIPCIÓN	Como usuario, quiero poder sincronizar mis datos entre diferentes dispositivos, para acceder a mi información desde cualquier dispositivo y tener una experiencia personalizada en cada uno, con opciones de sincronización adaptadas a mis preferencias.
ESTIMACIÓN	10
PRIORIDAD/VALOR	*Must-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	• Cuando el usuario acceda desde un dispositivo, deberá obtener los datos de su cuenta.

ID	10
TÍTULO	Registro de actividades físicas
DESCRIPCIÓN	Como usuario, quiero poder registrar las actividades físicas que realizó de manera personalizada, para visualizar el total de kilocalorías quemadas y comprender cómo afecta a mis niveles de glucosa, adaptado a mis preferencias y objetivos de actividad física.
ESTIMACIÓN	7
PRIORIDAD/VALOR	*Must-have*
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	• Contar con una base de datos extensa de actividades. • Obtener valores nutricionales. • Registro rápido y sencillo. • Visualmente intuitivo.

ID	11
TÍTULO	Modificación de datos
DESCRIPCIÓN	Como usuario, quiero poder realizar modificaciones en mis datos personales para poder obtener resultados ajustados a las modificaciones
ESTIMACIÓN	9
PRIORIDAD/VALOR	Must-have
DEPENDENCIAS	2
CONDICIÓN DE COMPLETITUD	• Si el usuario realiza una modificación en sus datos, esta debe ser almacenada. • Se debe de realizar de forma rápida y sencilla. • Notificación al usuario en el caso de editar su correo o contraseña.

2.4.- Planificación inicial de tareas

Tras un estudio de las diferentes historias de usuario definidas anteriormente, se ha establecido como primera historias de usuario épicas el sistema de registro y logueo del usuario junto con el sistema de lecturas de los niveles de glucosa.
Una vez que se complete esta historia de usuario épica se pasará a la siguiente que abordará el sistema de registro de alimentos, de actividades físicas y el sistema de cálculo automatizado de unidades de insulina. Por último, se ha agregado una última épica que incluirá la implementación de notificaciones, emergencias y los recursos educativos.

- **Iteración 1: Registro e Inicio de Sesión (4 semanas)**
  - **Historias de Usuario:** (PH 29)
    - Historia de usuario 1
    - Historia de usuario 2
    - Historia de usuario 11
  - **Tareas:**
    - Implementación de la creación de cuentas y registro inicial (Programador)
    - Desarrollo de la funcionalidad de inicio de sesión seguro (Programador)
    - Desarrollo de la edición y actualización de los datos (Programador)
    - Revisión y pruebas de la iteración (Testeador)

- **Iteración 2: Lecturas de glucosa y Sincronización de datos (4 semanas)**
  - **Historias de Usuario:** (PH 28)
    - Historia de usuario 3
    - Historia de usuario 4
    - Historia de usuario 9
  - **Tareas:**
    - Implementación de la entrada de lecturas básicas de glucosa (Programador)
    - Desarrollo de la sincronización de datos (Programador)
    - Revisión y pruebas de la iteración (Testeador)

- **Iteración 3: Registro de alimentos, actividades físicas y Visualización de datos (3 semanas)**
  - **Historias de Usuario:** (PH 14)
    - Historia de usuario 5
    - Historia de usuario 10
  - **Tareas:**
    - Implementación del registro de alimentos y actividades físicas (Programador)
    - Desarrollo de la visualización gráfica de estadísticas básicas (Diseñador/Programador)
    - Revisión y pruebas de la iteración (Testeador)

- **Iteración 4: Notificaciones (2 semanas)**
  - **Historias de Usuario:** (PH 5)
    - Historia de usuario 8
  - **Tareas:**
    - Implementación de las notificaciones(Diseñador/Programador)
    - Revisión y pruebas de la iteración (Testeador)

- **Iteración 5: Recursos educativos (2 semanas)**
  - **Historias de Usuario:** (PH 4)
    - Historia de usuario 7
  - **Tareas:**
    - Implementación de la visualización de recursos educativos. (Programador)
    - Revisión y pruebas de la iteración (Testeador)

- **Iteración 6: Funcionalidades de Emergencia (2 semanas)**
  - **Historias de Usuario:** (PH 5)
    - Historia de usuario 6
  - **Tareas:**

        - Desarrollo de las funciones de emergencia (Programador)
        - Revisión y pruebas de la iteración (Testeador)

- **Iteración 7: Pruebas y Ajustes Finales (2 semanas)**
    - **Tareas:**
        - Realización de pruebas exhaustivas (Analista)
        - Ajustes finales según los comentarios de prueba (Programador/Diseñador)

- **Iteración 8: Despliegue y Documentación (1 semana)**
    - **Tareas:**
        - Preparación de la aplicación para el despliegue (Programador)
        - Documentación del proceso de desarrollo y funcionamiento (Testeador)

Después de distribuir las tareas del proyecto, se ha observado que las dos primeras interacciones demandan una mayor cantidad de trabajo y atención en comparación con las demás. Esto se debe a que en estas etapas se están implementando las funcionalidades esenciales de la aplicación, marcando así un punto crítico en el desarrollo que requiere una dedicación más intensiva.

Para realizar los diferentes *sprints* en los que se ha dividido el proyecto, contaremos con los roles que se indicarán a continuación:

- El programador que va a ser la persona encargada de implementar las funcionalidades que componen la aplicación y del diseño que desarrolle el diseñador de interfaces.
- El diseñador será el responsable de realizar el boceto y diseño final de la interfaz de la aplicación.
- El testeador será la persona encargada de realizar las pruebas para comprobar que el funcionamiento y el desarrollo está yendo de forma correcta.

Sabiendo que solo se cuenta con una única persona en el equipo de desarrollo, las 300 horas que ocupa dicho proyecto se han repartido en las siguientes fases:

- **Definición del proyecto:** se dedicarán 10 horas, debido a que el autor del trabajo disponía ya de una definición bastante sólida. Esta fase fue la primera que se realizó.
- **Análisis y diseño inicial:** se dedicarán 80 horas, debido a que la temática del proyecto es un amplio campo y había que realizar un análisis bastante extenso y profundo
- I**mplementación del prototipo:** se dedicarán 180 horas. Dicha estimación ha sido considerada en base a una dedicación por parte del autor de entre 5 y 7 horas al día.

- **Elaboración de la memoria:** se dedicarán 30 horas. Dicha fase se ha ido realizando a lo largo de todo el desarrollo.

A continuación, se mostrará un diagrama de *Gantt* donde se ha reflejado la planificación inicial que se ha establecido para dicho desarrollo.

Ilustración 16: Planificación inicial

2.4.1.- Evolución de la planificación inicial

Tras comenzar con la planificación inicial, hemos experimentado ciertas variaciones debido a nuestra falta de experiencia en la organización de proyectos. En concreto, tuvimos dificultades al implementar algunas funcionalidades, como el registro de los niveles de glucosa. Esto nos obligó a modificar nuestro enfoque y dedicar más tiempo a encontrar soluciones que cubriera dicha necesidad. Lo que provocó que esta parte del desarrollo se alargue otras cuatro semanas más de lo previsto.

Por otro lado, la implementación de la *API* de *Google Gemini* y el sistema de gestión de notificaciones se completó una semana antes de lo planeado, gracias a que su guía de instalación era muy sencilla de seguir. Esto ayudó a compensar el tiempo extra que tomó desarrollar la funcionalidad de los niveles de glucosa

A continuación, se mostrará el diagrama de *Gantt* que refleja la planificación final del proyecto, donde las fechas establecidas se han intentado ajustar con la mayor exactitud posible a la realidad.

Como conclusión, podemos decir que la planificación inicialmente establecida se ha respetado, ya que en cada iteración se ha seguido el orden de las fases pero no el periodo de tiempo que se le había establecido a cada una.

Además, gracias a la metodología Kanban, hemos podido establecer prioridades adecuadas para cada tarea según su importancia en el proyecto, lo que nos ha ayudado a organizar mejor la planificación y obtener versiones estables para los clientes lo antes posible.

Nombre actividad	Fecha inicio	Duración en días	Fecha fin
Iteración 1	01-feb	28	29-feb
Registro	01-feb	10	11-feb
Inicio de sesión	11-feb	7	18-feb
Modificación de datos	18-feb	11	29-feb
Iteración 2	29-feb	30	30-mar
Registro de lecturas de glucosa	29-feb	61	30-abr
Visualización de datos	10-mar	10	20-mar
Sincronización de datos	20-mar	10	30-mar
Iteración 3	30-mar	21	20-abr
Registro de alimentos	30-mar	10	09-abr
Registro de actividades físicas	09-abr	11	20-abr
Iteración 4	20-abr	14	04-may
Notificaciones	20-abr	7	27-abr
Iteración 5	20-abr	14	04-may
Recurso educativos	20-abr	7	27-abr
Iteració 6	04-may	14	18-may
Funcionalidades de emergencia	04-may	14	18-may
Iteración 7	04-may	14	18-may
Pruebas	04-may	14	18-may
Ajustes finales	04-may	14	18-may
Iteración 8	18-may	6	24-may
Despliegue	01-feb	113	24-may
Documentación	01-feb	113	24-may

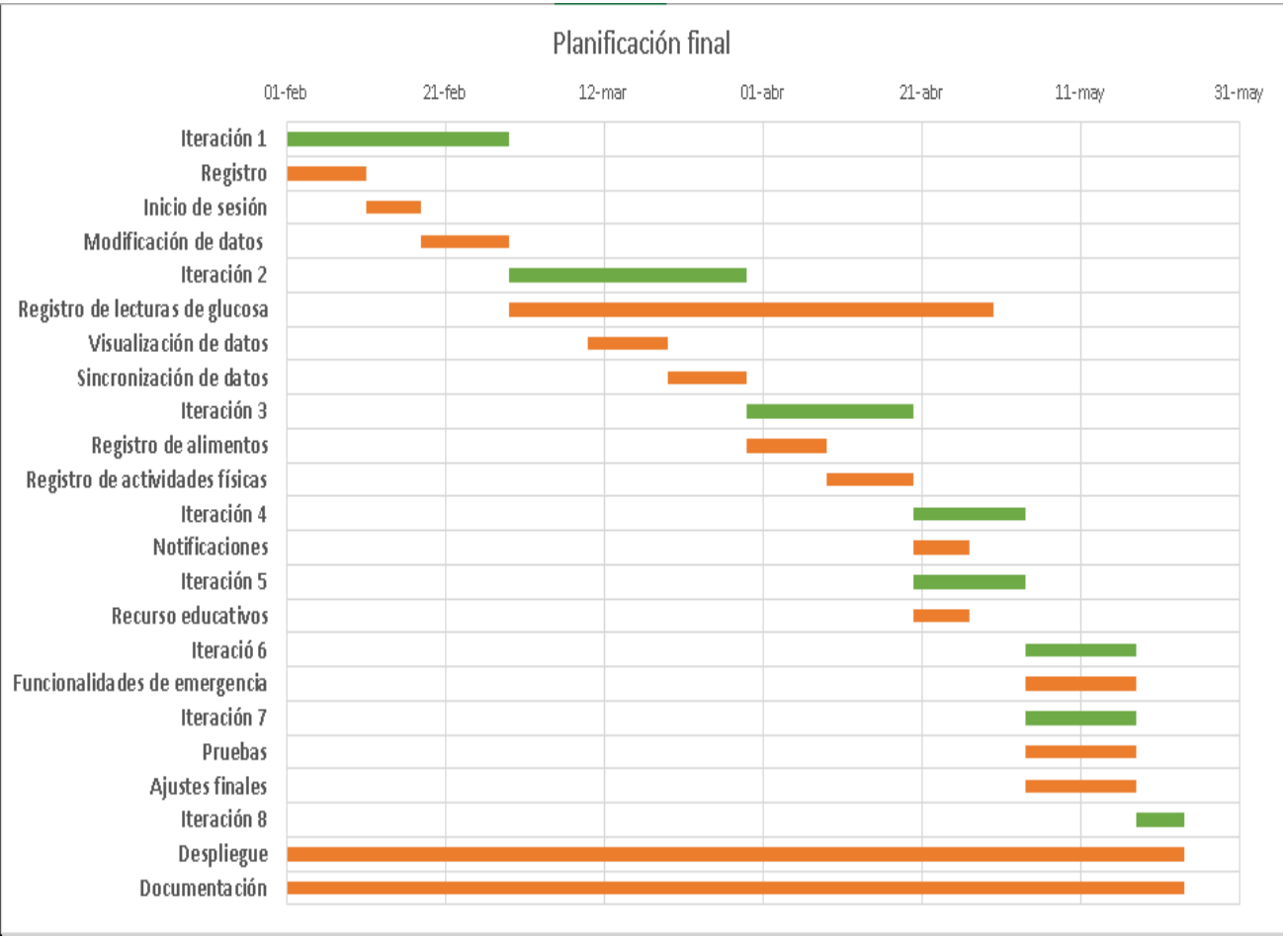


Ilustración 17: Planificación final

2.5.- Estudio de viabilidad

Teniendo en cuenta que en el año 2024, el salario neto de un ingeniero de *software* promedio en España es de 28.000€ al año a 47.000€ para profesionales más experimentados **[15]**, sabiendo que solo ha habido solamente una persona a cargo del proyecto, y considerando que el plan de estudios establece 300 horas para la realización del proyecto, podemos hacer una estimación cercana a los costes reales.

Desglose de costes del personal humano			
Coste por año	**Coste por mes (12 meses)**	**Coste por día (suponiendo 22 días laborables al mes)**	**Coste por hora (suponiendo 4 horas diarias)**
28.000	2.333	106,06	26.515
Total	26,515*300 horas = 7.954,54€		

Tabla 1: Desglose de costes del personal

Teniendo en cuenta los siguientes resultados, se puede sacar la conclusión que en costes del personal supondría en torno 7.954,54€.

Para la parte del *hardware* se ha utilizado los siguientes elementos:

- Análisis, diseño e implementación: MSI Pulse GL66 (2021, i7) adquirido en diciembre de 2021, cuyo coste es de unos 1.400€. Se estima que su vida útil sea de unos 5 años.
- Pruebas de desarrollo y calidad: Xiaomi Poco X4 Pro 5G, adquirido en mayo de 2022, cuyo coste es de unos 350€. Se estima que su vida útil sea de unos 4 años.

Teniendo en cuenta esto datos y suponiendo que para la parte de análisis, implementación y diseño se ha realizado 20 días (90 horas), para la implementación 30 días (180 horas) y para realizar el testeo de la aplicación 25 días (30 horas), se puede estipular los siguientes costes:

HARDWARE					
Dispositivo	**Precio**	**Vida útil**	**Amortización**	**Coste durante desarrollo**	**Coste de análisis y diseño**
MSI PULSE GL66 (desarrollo)	1.400€	5 años	1.400/(5*36)= 0.76€	0.76*20= 15.2€	0.76*30= 22.8€
Xiaomi Poco X4 Pro 5G (pruebas)	350€	4 años	350/(4*365)= 0.23€	0.23*25=5.75€	
Total	15.2€ + 5.75€ + 22.8€ = 43.75€				

Tabla 2: Desglose del coste del hardware

Esta estimación ha sido basada según el apartado **2.4.1.- Evolución de la planificación inicial.**

En cuanto a la parte del *software*, se puede indicar los siguientes programas, algunos bajo licencia gratuita por ser estudiante universitario y otros son gratuitos

- *Visual Studio Code*
- *Visual Paradigm 17.1* (**licencia de estudiante**)
- *Android Studio*
- *Visibily*
- *GitHub*

SOFTWARE		
Programas	**Coste**	**Coste de desarrollo**
Visual Code Studio	Gratis	0€
Android Studio		
GitHub		
Visual Paradigm 17.1		
Visibily		
Total	0€	

Tabla 3: Desglose del coste del software

Tras realizar todos los cálculos de los costes que supondría dicho el desarrollo del proyecto, se puede sacar el siguiente resumen:

Tabla resumen	
Coste del personal	7.954,54€ * 1 persona = 7.954,54 €
Coste del *hardware*	43.75 €
Coste del *software*	0 €
Coste total del proyecto	7.998,29 €

Tabla 4: Resumen del coste total del desarrollo

2.6.- Modelo de dominio

Un modelo de dominio es una representación de las clases conceptuales del mundo real, no de componentes *software*. No se trata de un conjunto de diagramas que describen clases *software*, u objetos *software* con responsabilidad. **[16]**

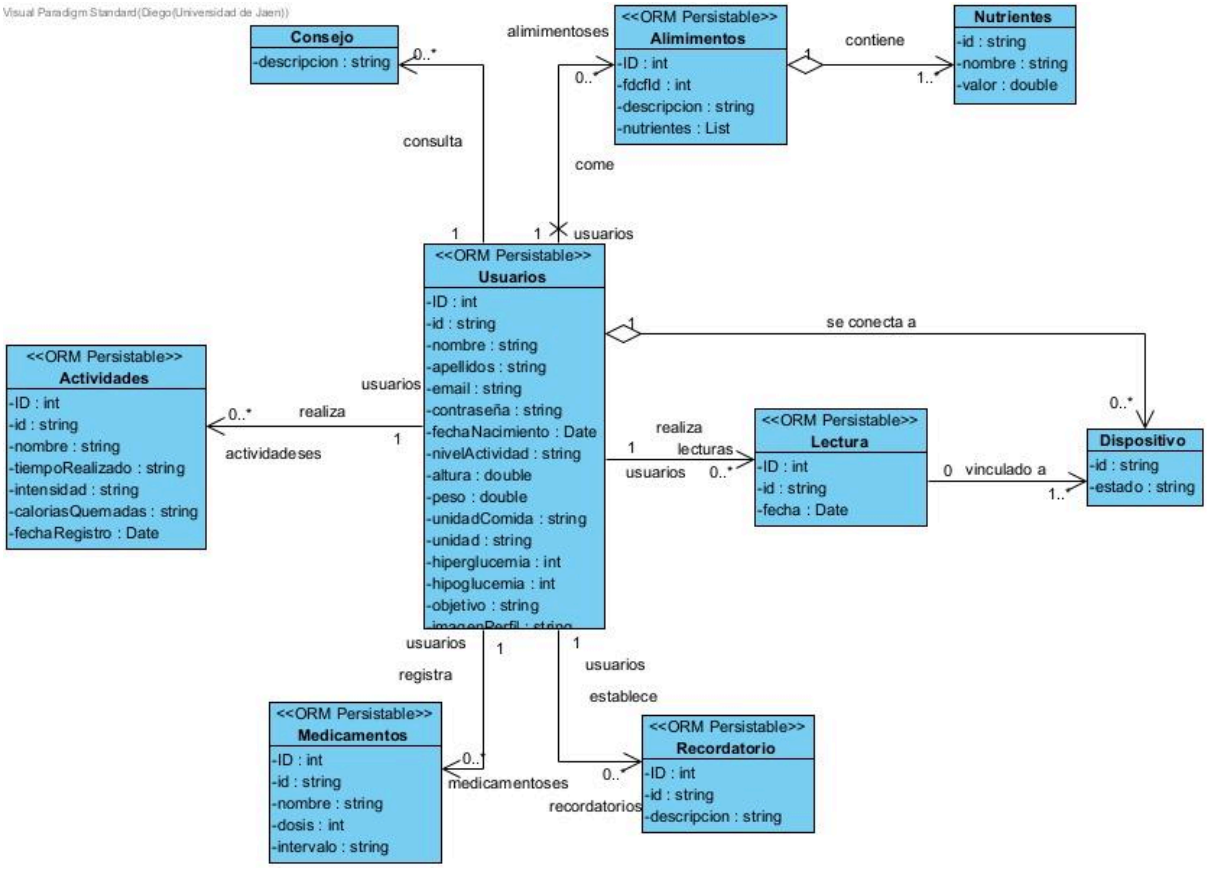


Ilustración 18: Modelo de dominio

En la siguiente ilustración, se muestra el esquema del diagrama de dominio de la aplicación. El cual se encuentra formado por las siguientes entidades:

- **Usuario**: representa la persona que interactúa con la aplicación. Este perfil incluye atributos que permiten almacenar información personal relevante, como el nombre, altura, peso, nivel de actividad física, valores de

hiperglucemia e hipoglucemia, entre otros datos pertinentes para la gestión de la salud y el bienestar. Estos datos son fundamentales para personalizar la experiencia del usuario y proporcionar recomendaciones y consejos adaptados a sus necesidades específicas.

- **Lectura**: registra las distintas mediciones de azúcar en sangre realizadas por el usuario. Este registro incluye atributos como un identificador único, fecha y hora de la toma, así como el nivel de glucosa medido. Los niveles de glucosa se encuentran dentro de tres rangos de referencia para la monitorización de la diabetes, cada uno con un papel crucial en el bienestar del usuario.

  El nivel de hipoglucemia representa el límite mínimo de azúcar en sangre, indicando una condición en la que el usuario corre riesgo y necesita atención inmediata para evitar complicaciones. Por otro lado, el nivel de hiperglucemia establece el límite máximo de azúcar en sangre, señalando un estado que requiere intervención y gestión para evitar complicaciones a largo plazo.

  El nivel objetivo de glucosa representa el valor recomendado para mantener un estado de salud óptimo. Al mantenerse dentro de este rango, el usuario puede minimizar el riesgo de complicaciones relacionadas con la diabetes y mantener un estilo de vida saludable.

- **Dispositivo**: hace referencia a los dispositivos utilizados para realizar las lecturas de glucosa. Su principal atributo es el estado, que indica si el dispositivo está conectado o no en un momento dado.

- **Actividad**: representa las actividades físicas realizadas por el usuario, donde tendrá atributos como la fecha de realización, la duración que indicará cuánto ha durado el ejercicio, la intensidad que hará referencia al esfuerzo que el usuario ha hecho durante la actividad y un identificador.

- **Alimentos**: registra los alimentos ingeridos por el usuario. Al igual que las actividades contendrá un atributo para indicar la hora en que se ha tomado, otro para almacenar su nombre y uno para guardar las calorías que posee.

- **Medicamentos**: registra las distintas medicinas que el usuario puede tomar. Cuenta con atributos como el nombre, un identificador y una variable de dosis para indicar la cantidad que el usuario debe de tomarse. Por ejemplo, el usuario podrá ingresar que se toma dos pastillas de ibuprofeno al día.

- **Recordatorio**: registra los diferentes avisos que el usuario registra en la aplicación, para ello cuenta con los atributos de fecha, hora y descripción.

- **Consejo**: representa los consejos que ofrece *Google Gemini* al usuario, contando solamente con el texto que genera.

Aparte de hablar de las clases que lo componen es importante tratar las relaciones que han sido establecidas entre ellas.
En el diseño del sistema, se han establecido relaciones entre las distintas entidades mediante el uso principalmente de agregaciones. Esto se debe a que la existencia de ciertos elementos depende de la existencia de otros.

Por ejemplo, el usuario es el componente central de la aplicación. Sin un usuario registrado, no pueden existir otros elementos en el sistema. Por lo tanto, se ha establecido una relación de agregación entre el usuario y las demás entidades, lo que significa que el usuario "posee" o "contiene" las lecturas de glucosa, las actividades físicas realizadas y los alimentos consumidos.

En el caso de los alimentos, se han considerado como entidades que contienen nutrientes. Esto se debe a que un alimento está compuesto por una combinación de nutrientes como proteínas, carbohidratos, grasas, vitaminas, etc. Por lo tanto, si no existe un alimento registrado en el sistema, los nutrientes asociados a ese alimento tampoco existirán. Esto se ha implementado mediante una relación de agregación entre la entidad "Alimento" y "Nutriente".

En cuanto a la relación entre la entidad "Lectura" y "Dispositivo", se ha establecido una asociación. Esto significa que un dispositivo puede existir independientemente de las lecturas de glucosa realizadas, pero las lecturas de glucosa sólo pueden existir en el contexto de un dispositivo. Es decir, un dispositivo puede tener múltiples lecturas asociadas, pero una lectura específica siempre estará vinculada a un único dispositivo. Esta relación de composición asegura que las lecturas de glucosa estén asociadas correctamente con el dispositivo que las realizó.

Un aspecto que hemos querido destacar, es el uso abundante de las relaciones con cardinalidad de uno a ninguno/muchos. Esto se debe a que se ha considerado que los usuarios pueden o no usar las diferentes funcionalidades de la aplicación.

3.- Diseño

3.1.- Diagrama Entidad-Relación

Un diagrama entidad-relación, también conocido como modelo entidad relación o ERD, es un tipo de diagrama de flujo que ilustra cómo las "entidades", como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema. **[17]**

Para generar el diagrama mostrado en la siguiente ilustración, se empleó la herramienta Visual Paradigm for Visual Studio, que automatiza la creación del diagrama de modelos basándose en una definición previa. Este conocimiento se adquirió en la asignatura "Diseño de Software" del cuarto año de la rama de Ingeniería del Software.

Gracias a los conocimientos que se han adquirido en las asignaturas “Fundamentos de Bases de Datos” y “Gestión y Administración de Bases de Datos”, podemos decir, que se encuentra en tercera forma normal. Esto se debe a que cada entidad tiene su clave primaria, y cualquier dependencia transitiva se elimina mediante el uso de claves foráneas.

A pesar de que el sistema se adaptaría bien a utilizar un sistema de bases de datos relacional, para la implementación de la base de datos se ha optado por utilizar *Firebase*, un *framework* que ofrece una base de datos *NoSQL*. *Firebase* permite almacenar datos en forma de colecciones, lo que facilita la organización y gestión de la información. Cada registro en las colecciones, como "Actividad", "Alimentos", etc., posee una clave primaria para su identificación y una clave foránea asociada al usuario correspondiente, asegurando así la integridad referencial y la asignación adecuada de los datos.

Firebase utiliza un mapeo automático de las entidades y sus atributos, simplificando el desarrollo y permitiendo una adaptación rápida a los cambios. Esto reduce el código repetitivo, minimiza los errores y mejora la productividad al centrarse en la lógica de negocio de la aplicación.

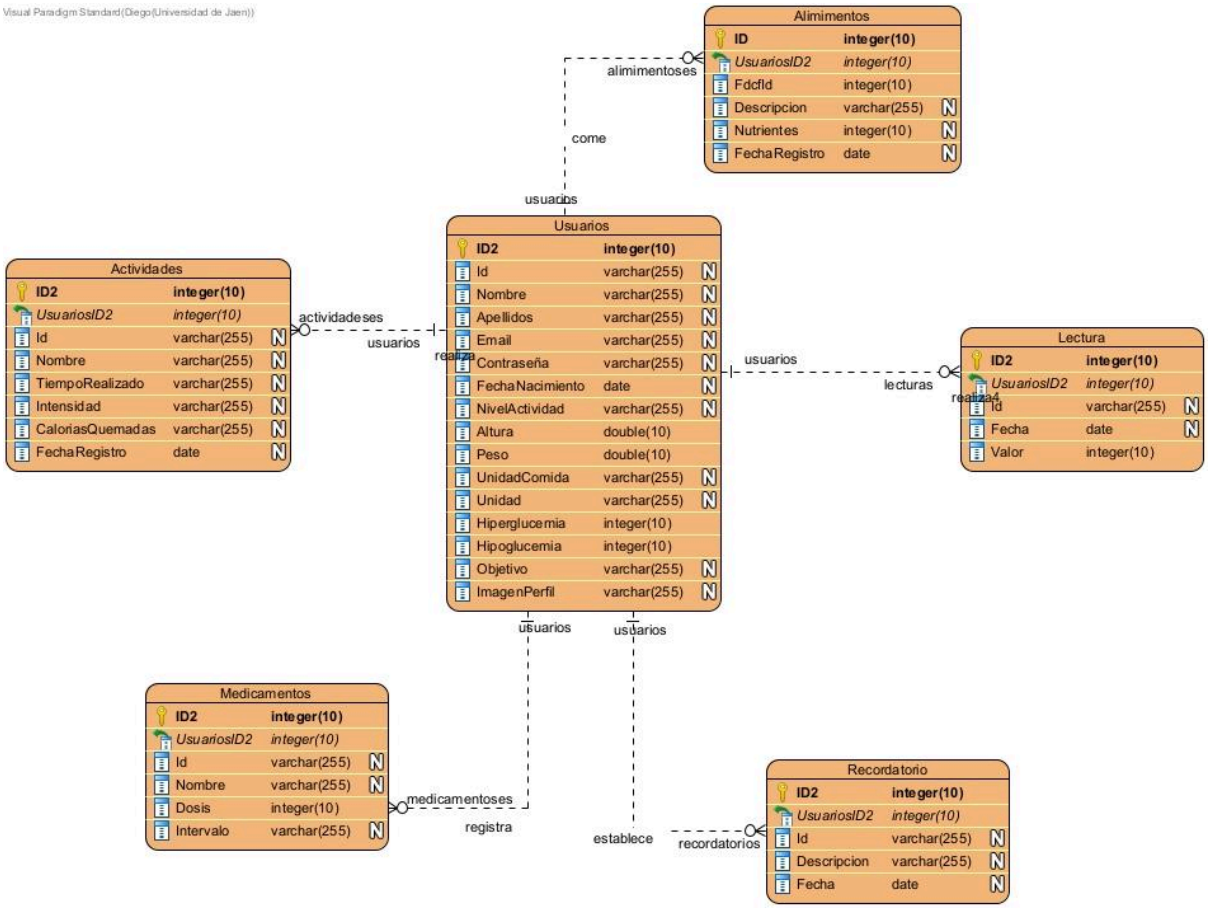


Ilustración 19: Diagrama Entidad-Relación

3.2.- Diagrama de Clases

Para el diseño de la aplicación se ha utilizado el patrón arquitectónico de *software Model-View-View Model* (MVVM). Facilitan¡ndo la separación de la interfaz de usuario, de la gestión de datos y de la lógica funcional de la aplicación.

Este patrón se comprende de tres componentes principales:

- **Modelo**: Representa la capa de datos y/o la lógica de negocio, también denominado como el objeto del dominio. El modelo contiene la información, pero nunca las acciones o servicios que la manipulan. En ningún caso tiene dependencia alguna con la vista.

- **Vista**: La misión de la vista es representar la información a través de los elementos visuales que la componen. Las vistas en MVVM son activas, contienen comportamientos, eventos y enlaces a datos que, en cierta manera, necesitan tener conocimiento del modelo subyacente.

- **Modelo de vista**: El modelo de vista es un actor intermediario entre el modelo y la vista, contiene toda la lógica de presentación y se comporta

como una abstracción de la interfaz. La comunicación entre la vista y el modelo de vista se realiza por medio de los enlaces de datos. **[18]**

En la siguiente ilustración, hemos querido ofrecer un esquema básico donde se representan las interacciones entre los diferentes elementos que componen este patrón.

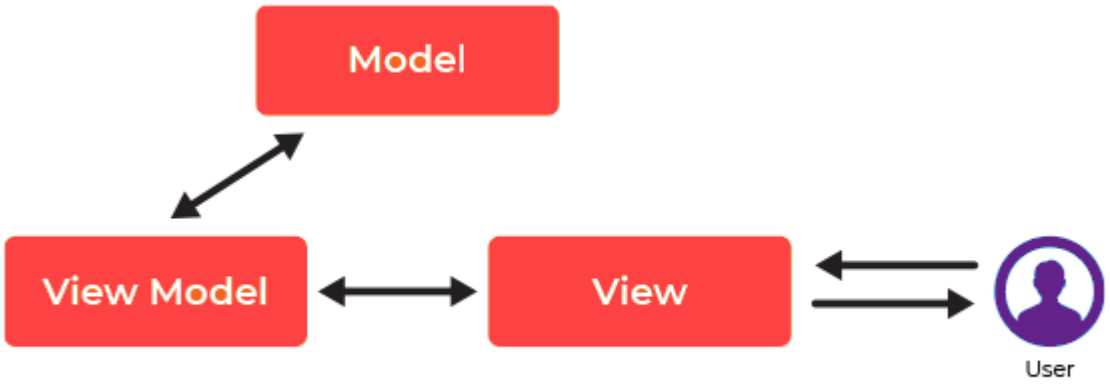


Ilustración 20: Diagrama de clases aplicando el patrón MVVM

Aplicando este modelo, nuestra aplicación ha quedado organizada de la siguiente forma. Esta distribución refleja todo el trabajo realizado durante las diferentes iteraciones comentadas anteriormente.

3.2.1.- Modelos/Servicios

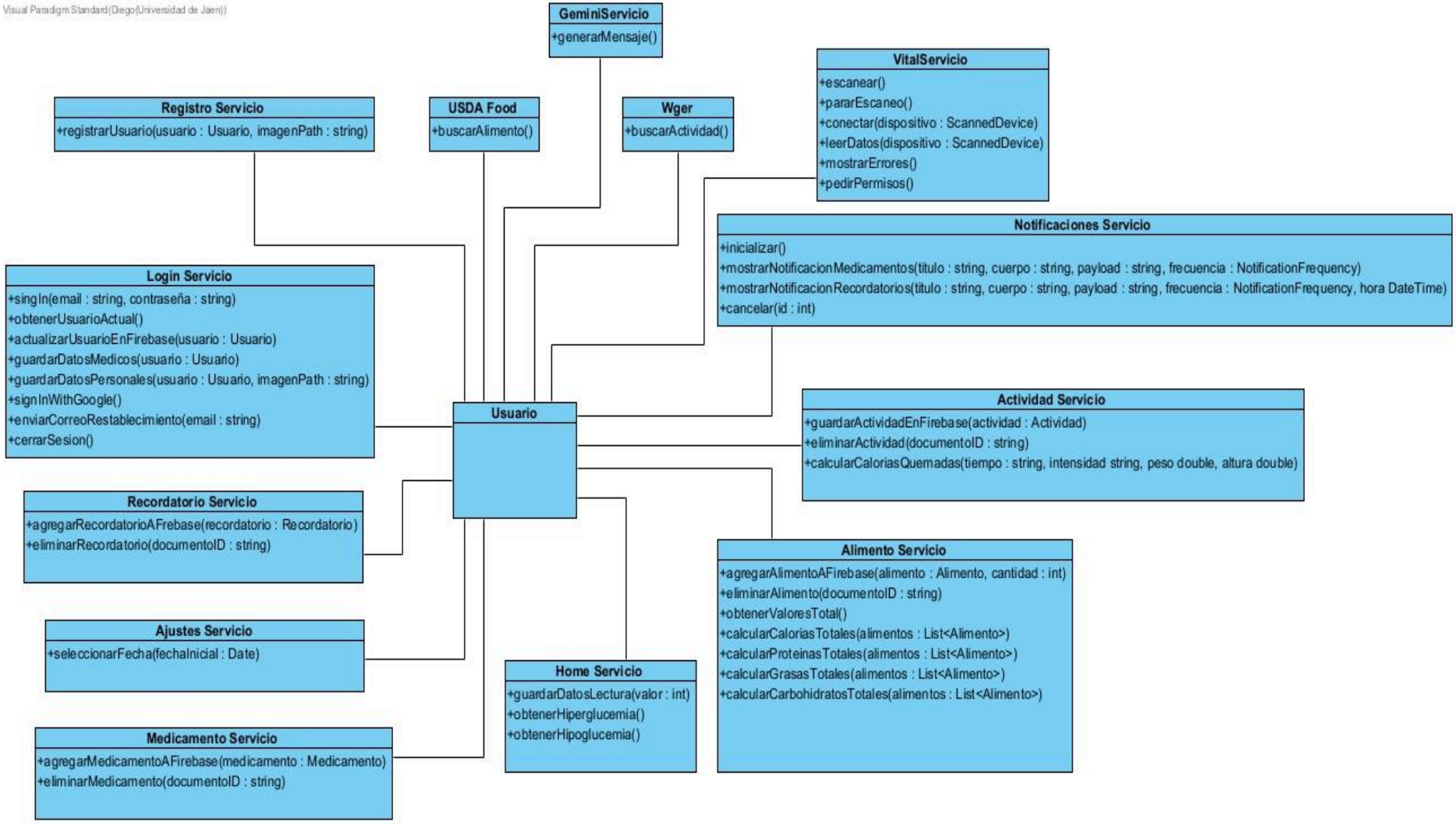


Ilustración 21: Relaciones modelo-servicio

En la ilustración podemos observar las diferentes interrelaciones entre los servicios y los modelos que componen el proyecto.

En la aplicación, los usuarios tienen la oportunidad de interactuar con una variedad de servicios que ofrecen diferentes funcionalidades para mejorar su experiencia. Los servicios que usan en nuestra aplicación:

- **ActividadServicio:** permite a los usuarios realizar diversas acciones relacionadas con la gestión de actividades físicas, desde obtener información sobre los ejercicios ofrecidos por *Wger*, hasta registrar y consultar aquellas actividades que han sido realizadas por el usuario, así como calcular el gasto calórico asociado.

- **AjustesServicio:** ofrece las funcionalidades necesarias para que el usuario pueda cambiar su foto de perfil como su fecha de nacimiento.

- **AlimentoServicio:** permite una gestión completa de los alimentos, desde la obtención de información detallada ofrecida por *USDA Food* hasta el registro, consulta y eliminación de alimentos en la base de datos, así como la capacidad de filtrar y obtener valores nutricionales específicos.

- **HomeServicio:** proporciona funcionalidades que permiten a los usuarios realizar sus lecturas de sangre como obtener sus niveles de hiperglucemia e hipoglucemia.

- **LoginServicio:** proporciona una gestión integral del proceso de inicio de sesión de los usuarios, ofreciendo varias opciones de autenticación y funcionalidades relacionadas con la administración de datos personales y médicos, así como opciones para cerrar sesión y restablecer la contraseña.

- **MedicamentoServicio:** proporciona una gestión completa de la medicación para los usuarios, permitiéndoles almacenar, consultar y eliminar información sobre sus medicamentos.

- **NotificacionServicio:** permite a los usuarios gestionar las notificaciones de la aplicación, pudiendo inicializarlas, mostrarlas y cancelarlas.

- **RecordatorioServicio:** proporciona una gestión completa de los recordatorios para que los usuarios puedan almacenar, consultar y eliminar sus citas, consultas, etc.

- **RegistroServicio:** ofrece la gestión integral para el almacenamiento de los usuarios en la aplicación.

- **USDAFoodServicio:** ofrece la obtención de los datos más relevantes de los alimentos que el usuario consulta.

- **WgerServicio:** ofrece la obtención de los datos más relevantes de los alimentos que el usuario consulta.

- **VitalServicio**: ofrece los métodos necesarios para poder conectar dispositivos de monitorización de glucosa con nuestra aplicación y poder realizar lecturas. Debido a que no hemos podido contar con una versión completamente funcional de la *API*, hemos implementado a forma de *mock* un sistema que nos permitirá simular las lecturas que se realizan realmente al usar este servicio. Esta implementación consiste en generar una serie de valores basados en una fórmula estadística. No obstante, en el apartado **4.2.6.4** se detallarán más a fondo los diferentes métodos que nos ofrece la *API* de *Vital* para poder realizar las lecturas.

- **GeminiServicio**: ofrece a los usuarios la posibilidad de mantenerse al tanto de las últimas noticias relacionadas con su enfermedad, así como de obtener consejos para mejorar su día a día. Además, les brinda la oportunidad de crear un plan alimenticio o una rutina de ejercicios para mejorar su estado de salud. En general, este servicio se ha implementado con la idea de ayudar en la calidad de vida de los usuarios día a día.

3.2.2.- Vistas

Nuestra aplicación está compuesta por una gran cantidad de vistas.

3.2.2.1.- Inicio

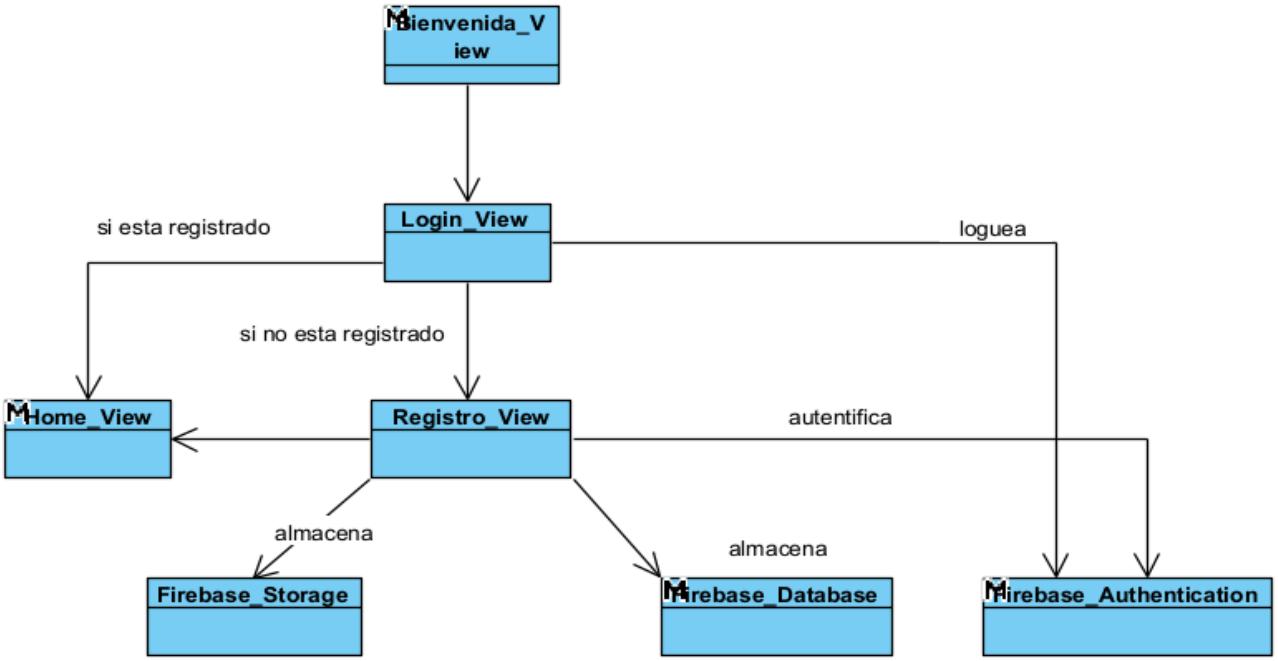


Ilustración 22: Vistas relacionadas con el inicio y registro

Como se puede observar en la ilustración, el apartado inicial de la aplicación que está compuesto por tres vistas (*Bienvenida_View*, *Login_View* y *Registro_View*), las cuales usan los servicios de *Firebase Database* para almacenar aquellos usuarios que se registran, *Firebase Storage* para almacenar su imagen de perfil y *Firebase Authentication* para poder loguerlos tanto cuando se registran como cuando ya existen.

Tras finalizar el logueo o el registro el usuario accede a la pantalla inicial de la aplicación (*Home_View*))

3.2.2.2.- Home

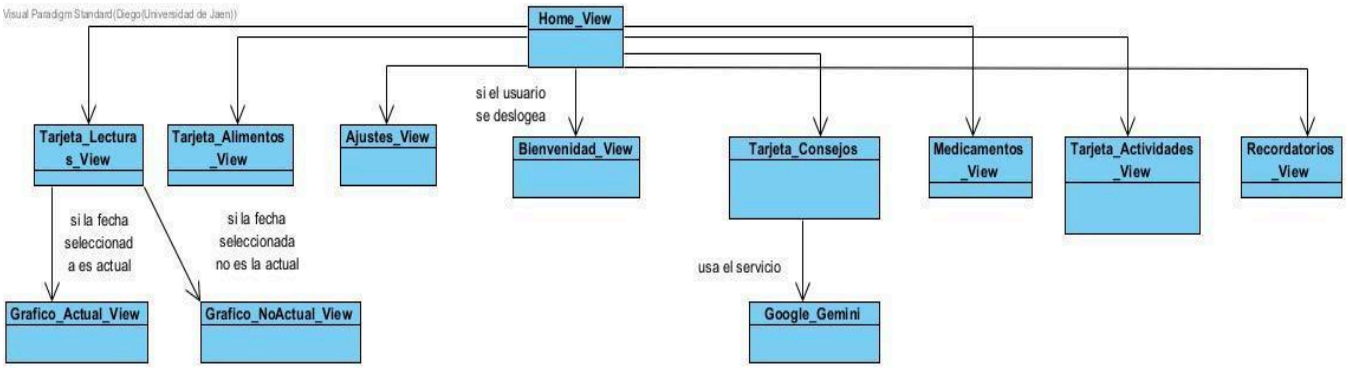


Ilustración 23: Vistas relacionadas con la pantalla principal

Una vez que el usuario se ha logueado accede a la vista principal de la aplicación (*Home_View*). Esta pantalla está compuesta por los siguientes componentes:

- **Tarjeta_Lectura_View**: es el componente que nos muestra los datos de las diferentes lecturas de glucosa. En este caso podemos diferenciar dos tipos de gráficos, aquel que se visualiza cuando nos encontramos en el día actual (*Grafico_Actual_View)* y otro para el caso contrario, él cual nos muestra un resumen de las lecturas del día (*Grafico_NoActual_View*).

- **Tarjeta_Alimentos_View**: es el componente que engloba todo lo relacionado con la alimentación.

- **Tarjeta_Actividades_View**: engloba todo lo relacionado con las actividades deportivas.

- **Tarjeta_Consejos_View**: utiliza el servicio de *Google Gemini* para que se pueda ver la respuesta a la pregunta que el usuario realiza a la vista.

- **Medicamentos_View**: es el componente encargado de dirigirnos a la sección de medicamentos.

- **Recordatorios_View**: es el componente encargado de dirigirnos a la sección de recordatorios.

- **Ajustes_View**: es el componente encargado de dirigirnos a la sección de ajustes.

Además de estos componentes mencionados, tenemos la vista de *Bienvenido_View*, la cual se muestra cuando el usuario se desloguea.

3.2.2.3.- Alimentación

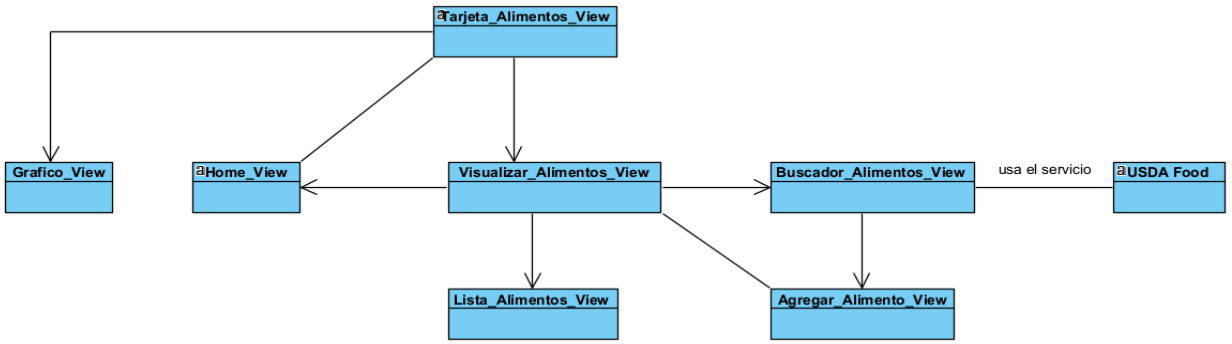


Ilustración 24: Vistas relacionadas para almacenar datos

Se debe destacar que este esquema de componentes es usado tanto para esta sección como para la de actividades.

Como podemos ver en la **ilustración 24**, estas secciones se componen de las siguientes vistas:

- **Visualizar_Aliementos_View**: representa la página donde se mostrará el listado de alimentos que posee el usuario en cada día.

- **Lista_Alimentos_View**: es una tarjeta en la que se visualizan los diferentes datos de cada alimento almacenado.

- **Buscador_Alimentos_View**: es la vista que se encarga de representar el buscador de alimentos, la cuál hace uso del servicio USDA Food para poder obtener la información de los alimentos.

- **Agregar_Alimentos_View**: representa la pantalla donde se visualizan los datos del alimento que se ha buscado en *Buscador_Alimentos_View*.

- **Gráfico**: esta vista ofrece un gráfico donde se representan las calorías consumidas en cada día.

A la vista *Home_View* se accede en el caso de que desde *Visualizar_Alimentos_View* se retroceda.

3.2.2.4.- Ajustes

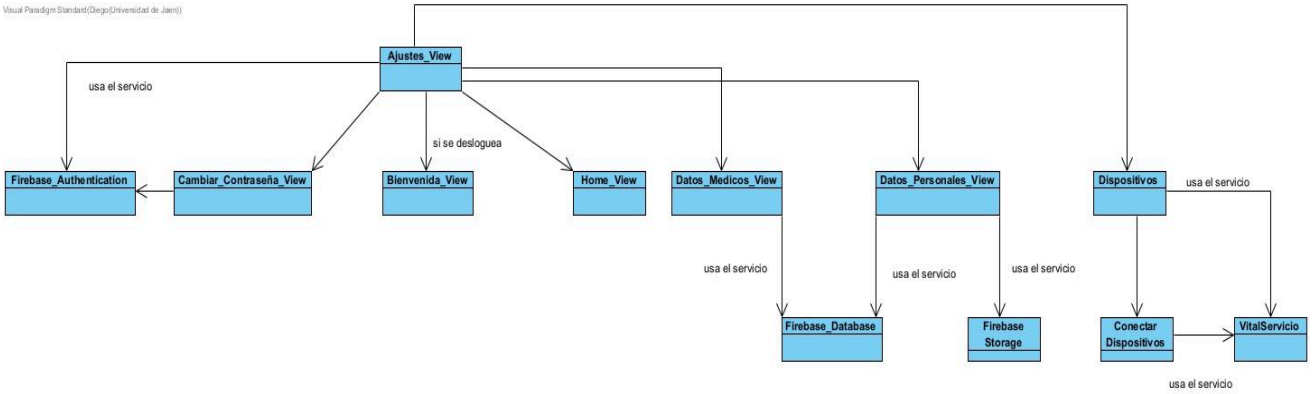


Ilustración 25: Vistas relacionadas con los ajustes

El apartado de los ajustes, se encuentra formado por las vistas Datos_Personales_View, Datos_Medicos_View y Cambiar_Contraseña_View, las cuales hacen referencia a las diferentes pestañas que el usuario se puede encontrar.
En ellas se hacen uso de los servicios Firebase Database para poder almacenar los cambios de datos, Firebase Storage para almacenar la imagen de perfil en caso de que el usuario la modifique y del servicio Firebase Authentication para volverse a loguear al usuario cuando modifique su contraseña o cierre sesión.
Como se ha indicado en los apartados anteriores, el usuario puede retroceder a Home_View.

Aparte de estas secciones, la aplicación cuenta con más secciones como son la de los medicamentos y recordatorios. Que poseen una estructura muy parecida que la sección de las actividades y alimentos, pero quitando los componentes de Grafo_View y Agregar_Alimentos/Actividades_View.

3.2.3.- Modelos de vistas

Al igual que la aplicación se compone de numerosas vistas, también contiene un gran número de modelos de vistas que se encargan de comunicar los servicios con las propias vistas.

- **Actividad_viewmodel:** es el intermediario entre las vistas relacionadas con las actividades y el servicio correspondiente. No solo solicita datos al servicio y los procesa para visualización, sino que también recibe datos introducidos por el usuario en las vistas. Además, gestiona los eventos y acciones del usuario relacionados con las actividades, como agregar, consultar o eliminar actividades físicas seleccionadas en diferentes partes de la aplicación.

- **Ajustes_viewmodel:** actúa de intermediario entre las vistas relacionadas con la sección de ajustes y los servicios que se encargan de ella y con el logueo del usuario. Es el responsable de manejar los eventos y acciones del usuario relacionados con los ajustes, como es el funcionamiento para enviar el correo de restablecimiento de la contraseña, para cerrar la sesión actual, seleccionar la imagen nueva de perfíl del usuario o guardar los datos del usuario.

- **Alimentos_viewmodel:** gestiona la interacción del usuario con la sección de alimentación de la aplicación. Facilita la comunicación entre las vistas y el servicio de alimentos, permitiendo la visualización, adición, eliminación y cálculo de valores nutricionales. Además de recibir datos del servicio, también recibe datos introducidos por el usuario en las vistas, como la selección de alimentos para su registro.

- **Home_viewmodel:** se encarga de comunicar la vista home con los servicios *HomeServicio* y *LoginServicio*, permitiendo su interacción para poder visualizar los datos que el usuario desea obtener por vista.

- **Login_viewmodel:** es el responsable de manejar las interacciones del usuario relacionadas con el proceso de inicio de sesión. Solicita datos al servicio de inicio de sesión, los procesa y los proporciona a las vistas para su visualización adecuada. Además, recibe datos introducidos por el usuario en las vistas de inicio de sesión, como nombre de usuario y contraseña.

- **Medicamentos_viewmodel:** actúa como intermediario entre las vistas y el servicio de medicamentos. Además de solicitar datos al servicio y procesarlos para su visualización, también recibe datos introducidos por el usuario en las vistas de gestión de medicamentos, como la adición de nuevos medicamentos o cambios en la dosis.

- **Notificaciones_viewmodel:** gestiona la configuración de notificaciones en la aplicación. Sirve como intermediario entre las vistas y el servicio de notificaciones, asegurando que la configuración de notificaciones se comunique adecuadamente a todas las partes de la aplicación que lo requieran.

- **Recordatorios_viewmodel:** responsable de la comunicación entre las vistas y el servicio de recordatorios. Además de solicitar datos al servicio y procesarlos para su visualización, también recibe datos introducidos por el usuario en las vistas de gestión de medicamentos, como la adición de nuevos medicamentos o cambios en la dosis.

- **Registro_viewmodel:** es el intermediario entre el formulario de registro y el servicio correspondiente. Se encarga de comunicar datos al servicio, procesarlos según sea necesario y proporcionarlos a las vistas para su visualización, facilitando así el proceso de registro de usuarios en la aplicación.

3.3.- Diagramas de secuencia

Muestran una interacción entre objetos organizados en una secuencia de tiempo. Los diagramas de secuencia pueden dibujarse con distintos niveles de detalle y para satisfacer distintos objetivos en las diversas etapas del ciclo de vida del desarrollo del sistema. Normalmente se utilizan para representar la interacción entre objetos que se produce en un caso de uso o para una operación. Cuando se utiliza para modelar el comportamiento dinámico de un caso de uso puede considerarse como una especificación detallada del caso de uso. **[19]**

3.3.1.- Inicio de sesión/Registro

En las **ilustraciones 26 y 27**, podemos ver las diferentes acciones que hay que realizar para que un usuario pueda iniciar sesión o registrarse. También se puede observar que en cuanto se inserta un dato incorrecto se lanza un aviso de dicho error en la misma vista.

Este procedimiento deberá ser realizado primeramente por los usuarios para poder hacer el resto de procedimientos.

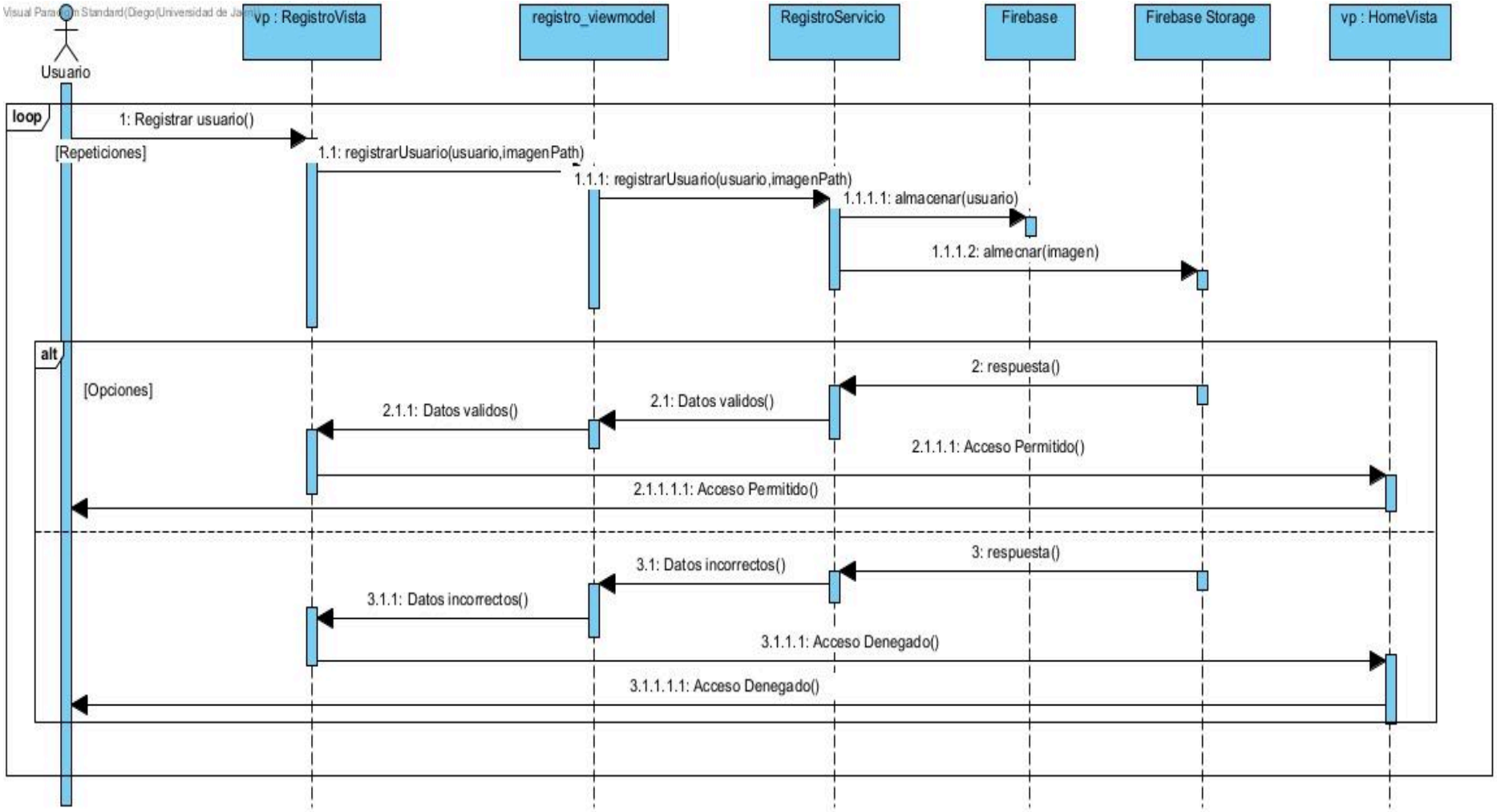


Ilustración 26: Diagrama de secuencia para el registro

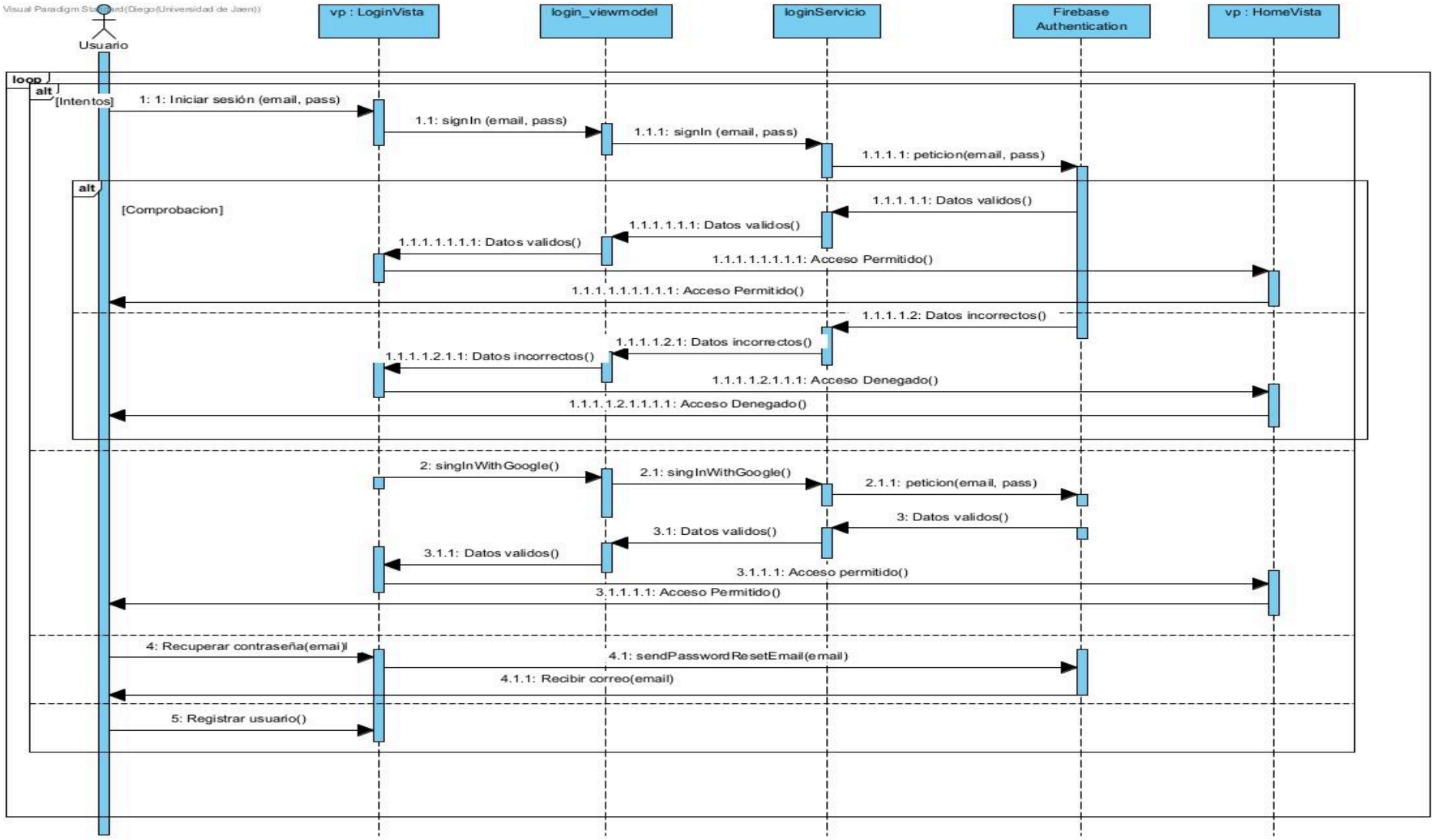


Ilustración 27: Diagrama de secuencia para el inicio de sesión

3.3.2.- Lecturas

En la siguiente imagen, se observa la secuencia que el usuario debe de seguir para poder realizar un registro de una lectura del nivel de glucosa.

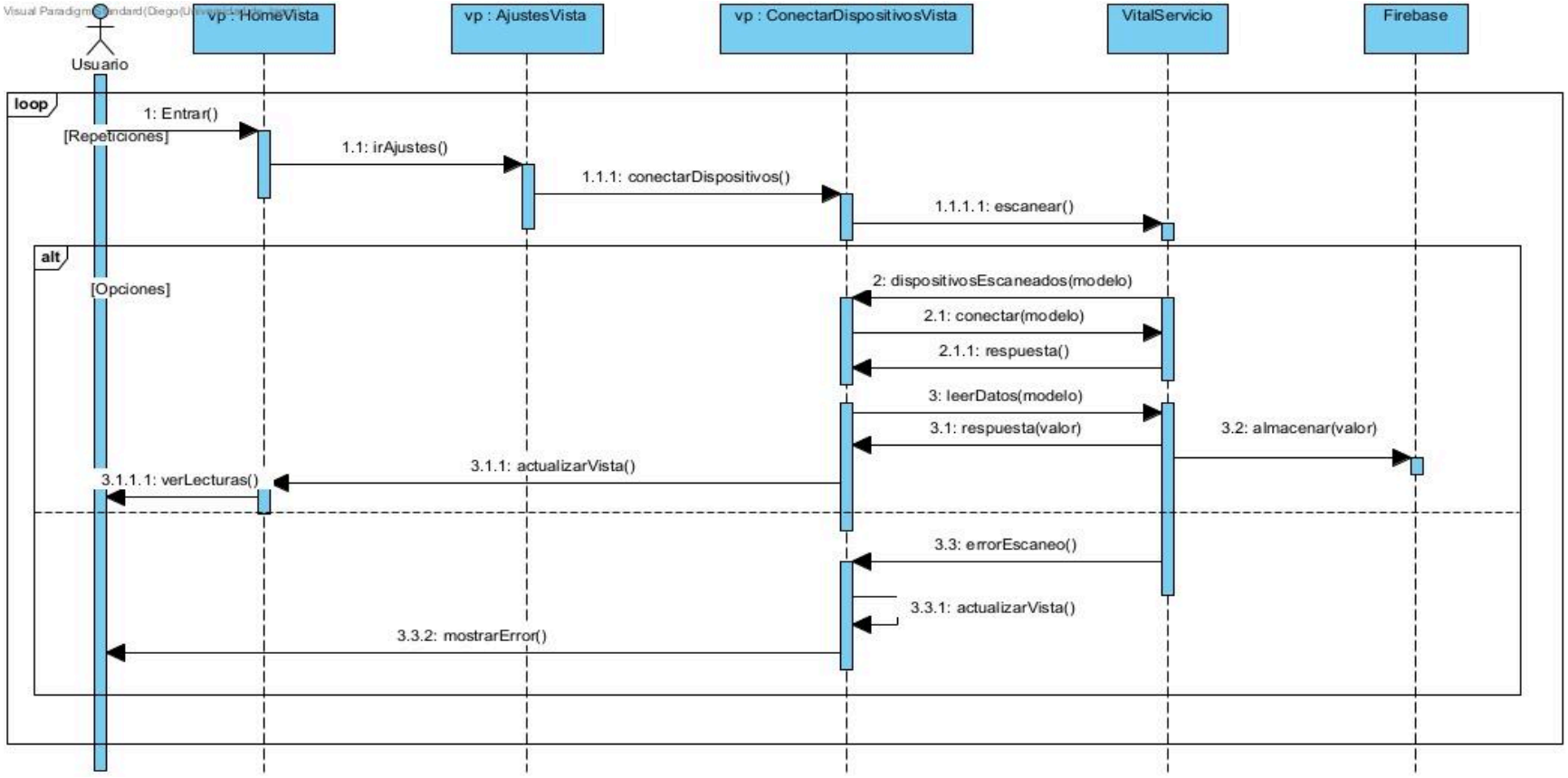


Ilustración 28: Diagrama de secuencia para el registro de lecturas de glucosa

3.3.3.- Registro de alimentos y actividades físicas

En la **ilustración 29** se observa la secuencia que se debe seguir para registrar tanto un alimento como una actividad física.

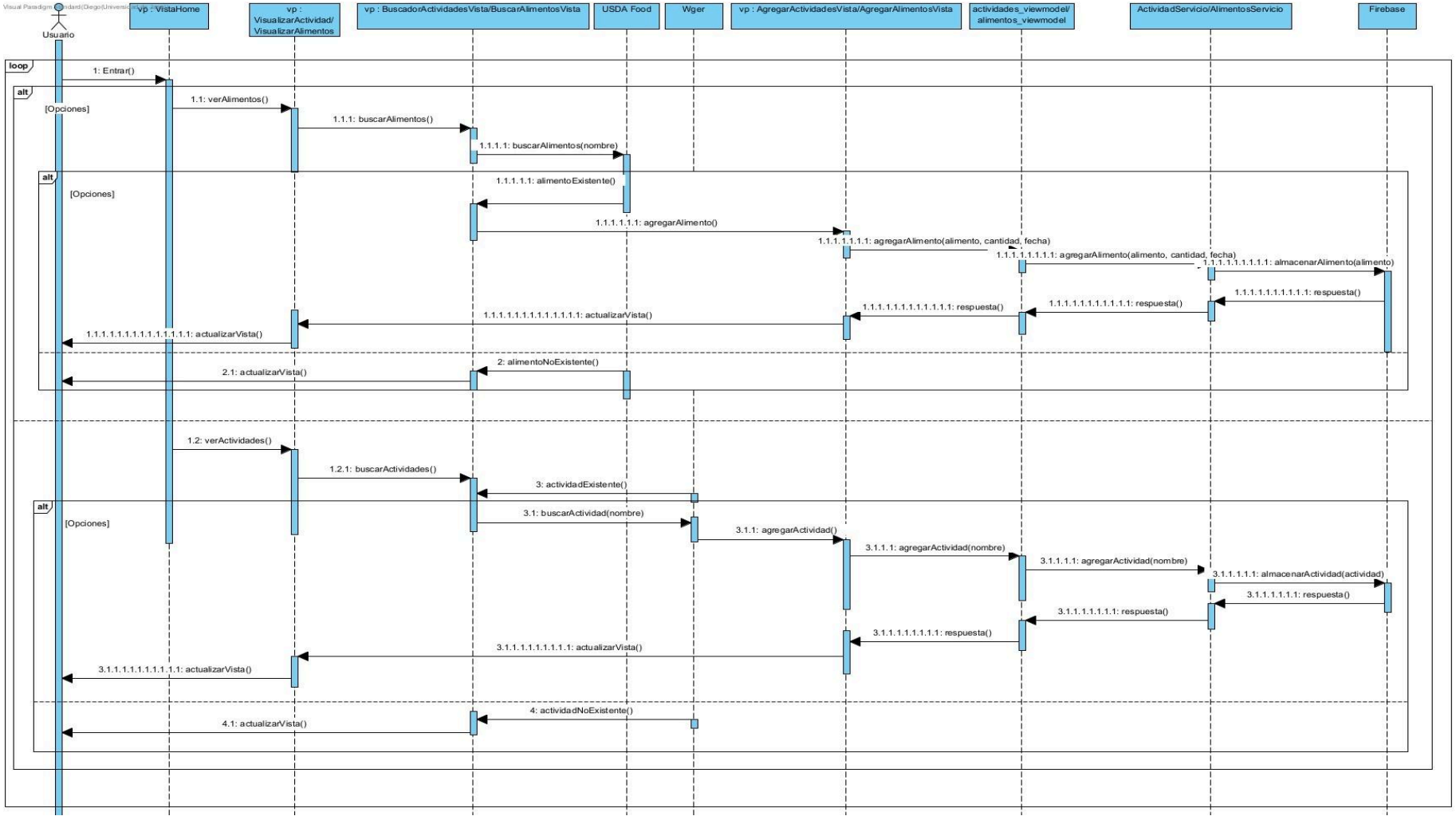


Ilustración 29: Diagrama de secuencia para el registro de alimentos o ejercicios

3.3.4.- Ajustes

En esta ilustración se muestra las diferentes acciones que el usuario podrá realizar en la sección de ajustes, como por ejemplo modificar tanto sus datos personales como los médicos o establecer conexión con los diferentes glucómetros disponibles

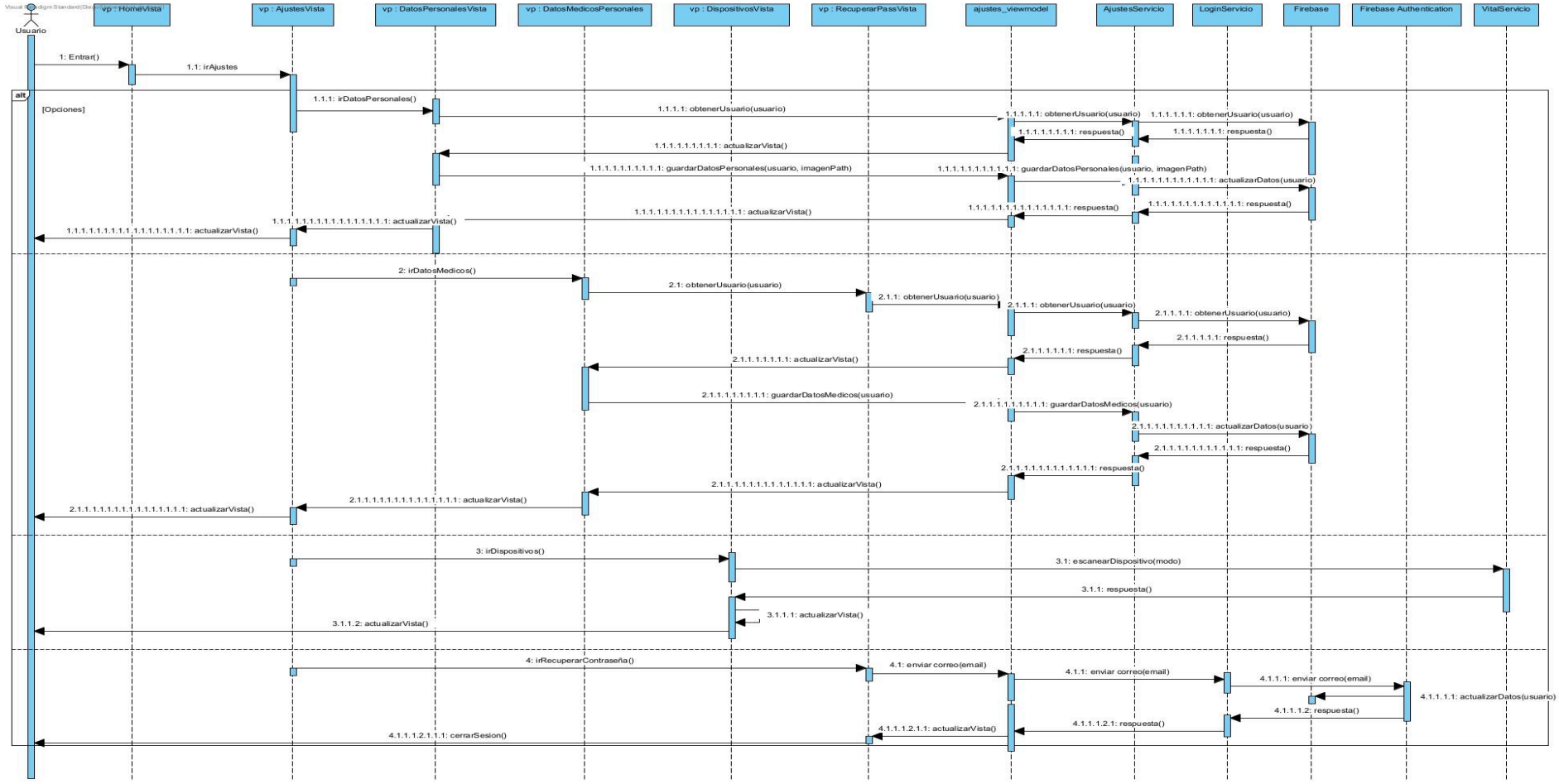


Ilustración 30: Diagrama de secuencia para realizar la configuración o modificación de la aplicación

3.3.5.- Registro de medicamentos y recordatorios

En la **ilustración 31** se observa el procedimiento que el usuario deberá realizar para poder agregar sus medicamentos o alguna cita importante que no quiera olvidar.

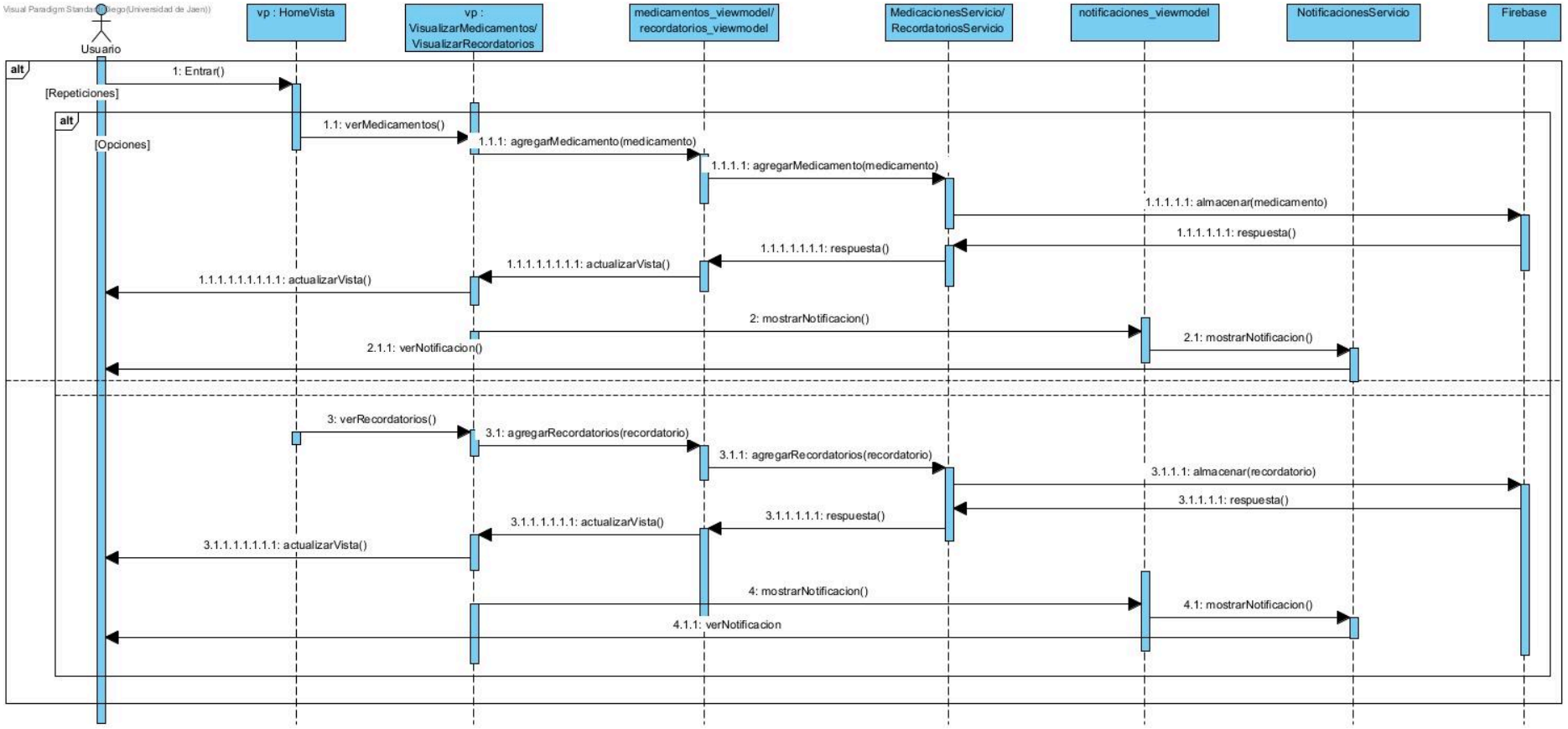


Ilustración 31: Diagrama de secuencias para registrar recordatorios y medicamentos

3.3.6.- Consejos

Ilustración que resume los pasos que el usuario puede realizar para poder resolver cualquier duda que se le presente sobre las diabetes.

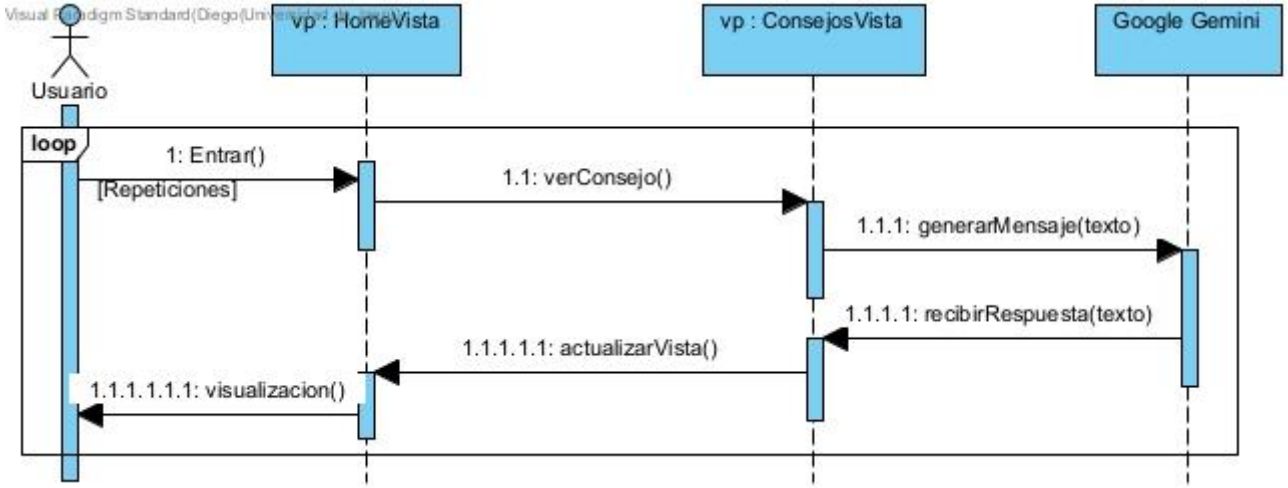


Ilustración 32: Diagrama de secuencia para obtener los recursos educativos

3.4.- Diseño de la Interfaz

A continuación las siguientes ilustraciones mostrarán los prototipos de las diferentes ventanas que compondrán la aplicación.

Visily es una herramienta de *wireframes* impulsada por inteligencia artificial diseñada para permitir que equipos de todas las habilidades y tamaños creen fácilmente *wireframes* de aplicaciones. Esta herramienta es ideal tanto para diseñadores como para no diseñadores. **[21]**

Esta plataforma ofrece las siguientes características:

- *Wireframing* e ideación rápida
- *Mockups* de alta fidelidad
- Prototipado y presentación
- Colaboración en equipo
- Asistente de inteligencia artificial

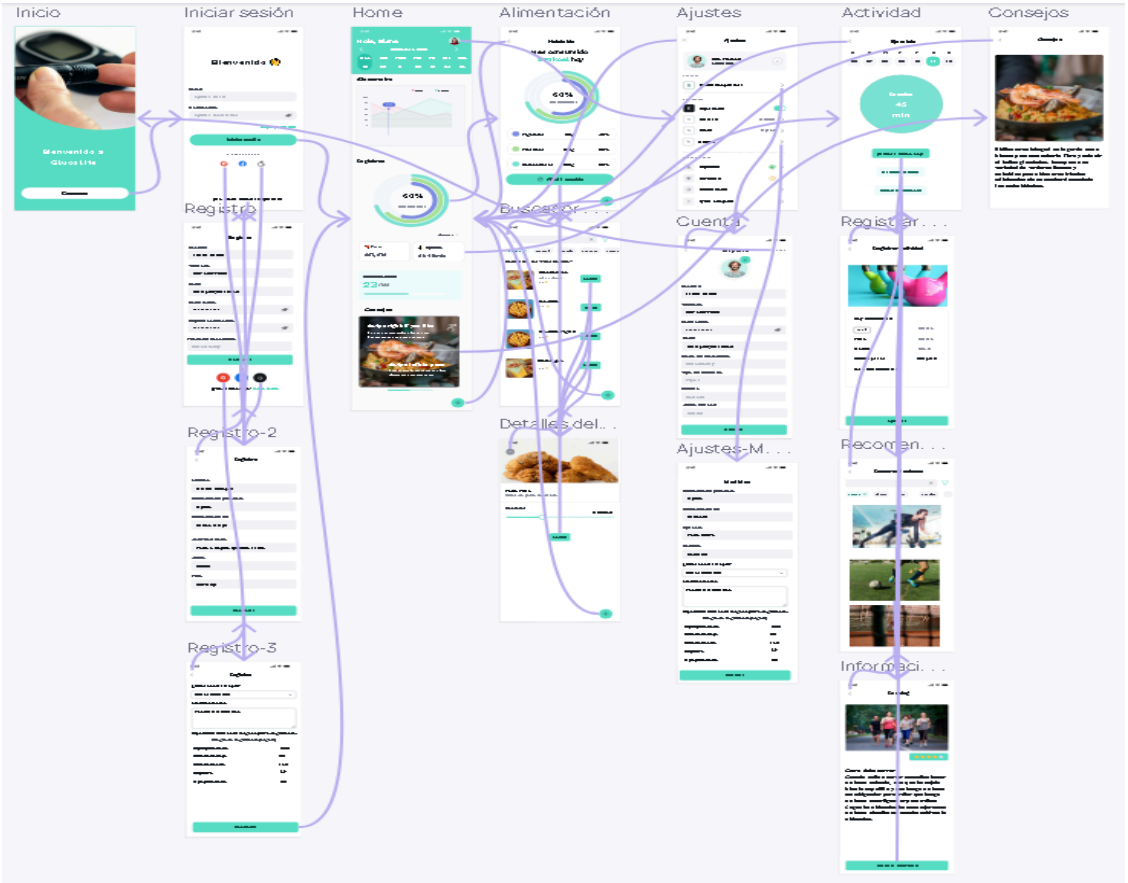


Ilustración 33: Mapa web obtenido con Visily

En el siguiente enlace [5] se ofrece el acceso a la plataforma, donde se podrá observar con mayor determinación las diferentes relaciones entre las ventanas que componen la aplicación.

3.4.1.- Inicio de sesión y registro

Las siguientes ilustraciones muestran las vistas que el usuario recibirá cuando proceda a iniciar sesión o registrarse por primera vez en la aplicación.

Relacionado con el registro se puede observar las distintas páginas que componen el formulario.

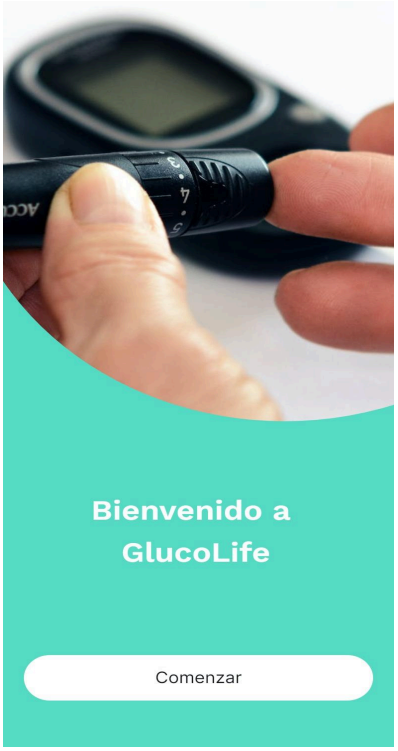


Ilustración 34: Pantalla de inicio de aplicación

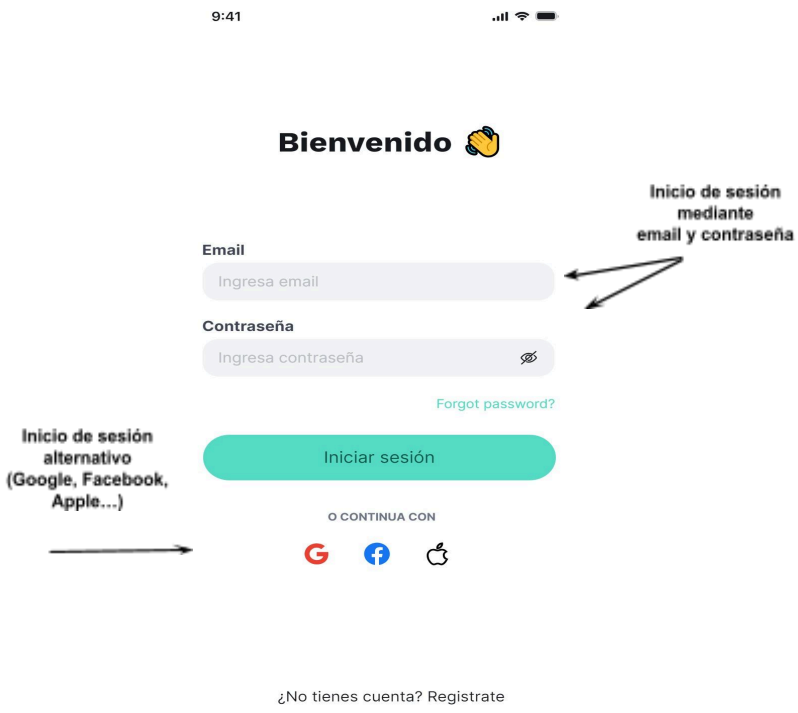


Ilustración 35: Pantalla de inicio de sesión

5. https://app.visily.ai/projects/b800b3cc-29dd-4899-97f7-fe64bb0e386e/boards/555335 (Marzo, 2024)

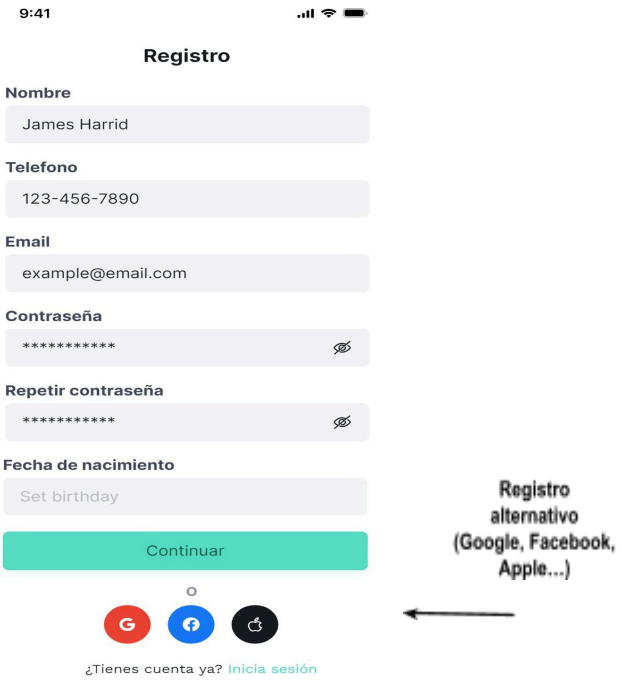


Ilustración 36: Primera pantalla de registro

Ilustración 37: Segunda pantalla de registro

Ilustración 38: Tercera pantalla de registro

3.4.2.- Pantalla principal

En la **ilustración 39**, se ha querido mostrar el aspecto que tendrá la pantalla principal (*home*) de la aplicación. En ella se puede encontrar las diferentes secciones que corresponden a las funcionalidades que ofrecerá la aplicación.

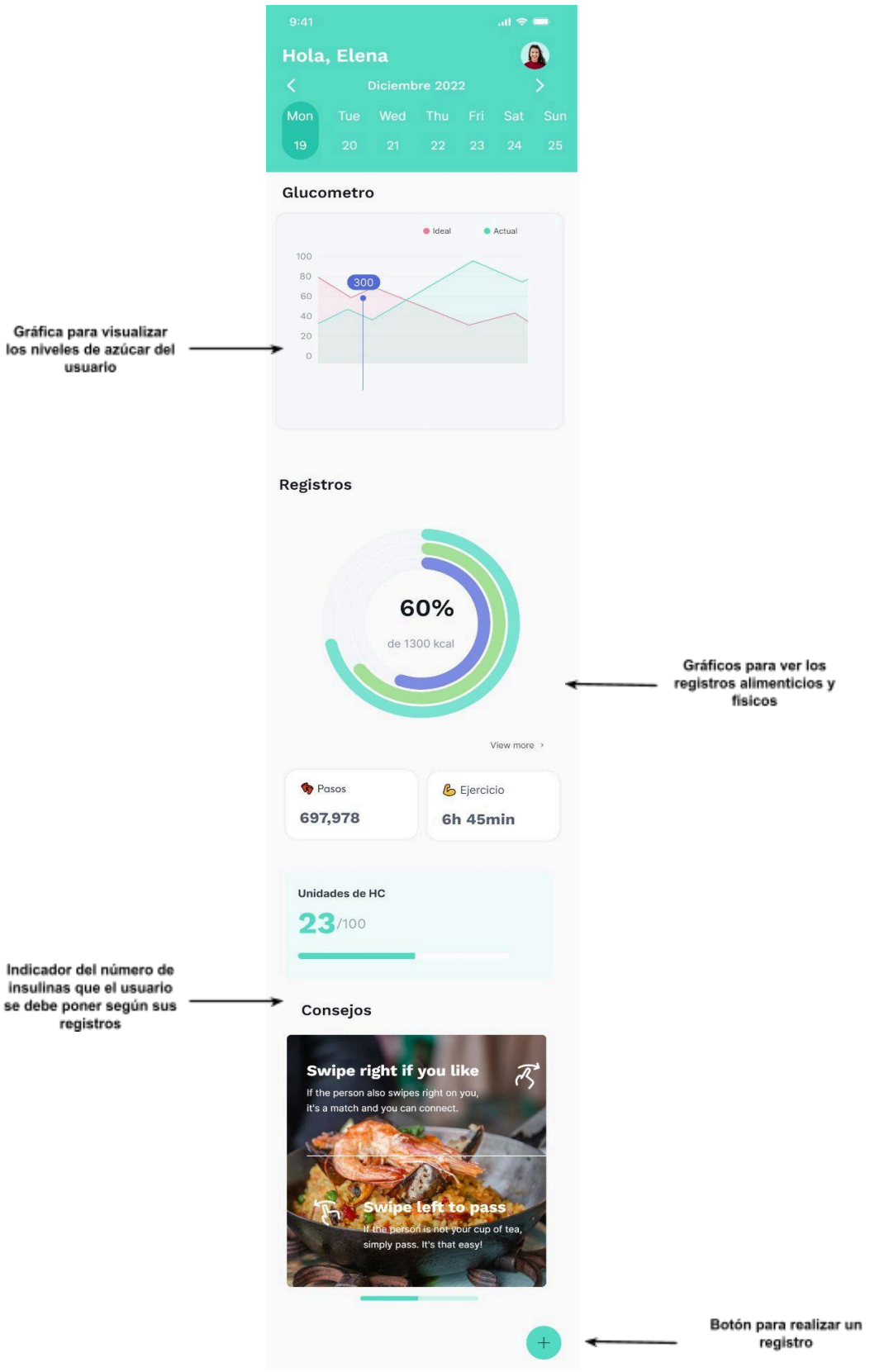


Ilustración 39: Pantalla de principal de la aplicación

3.4.3.- Pantalla de ajustes

La siguiente ilustración muestra las diferentes vistas que componen la sección de ajustes. En esta sección, los usuarios podrán modificar o consultar tanto sus datos personales como sus datos médicos, pudiendo también cambiarlos.

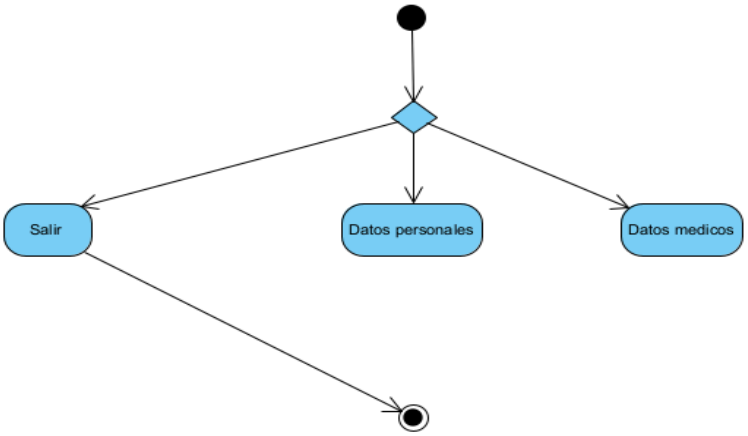


Ilustración 40: Diagrama de actividad de la ventana de ajustes

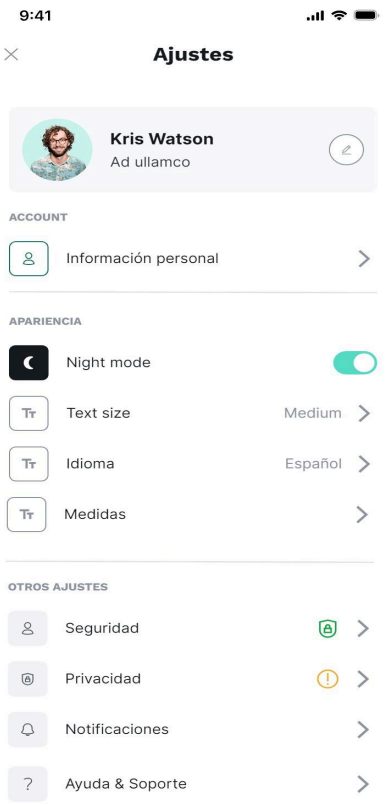


Ilustración 41: Pantalla de ajustes

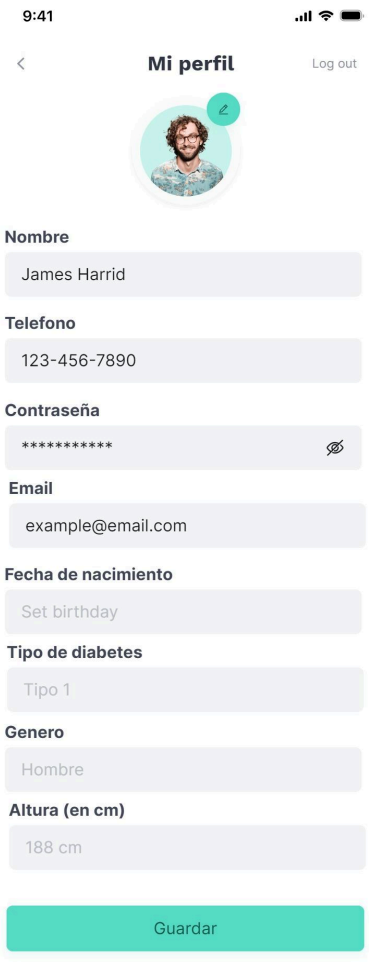


Ilustración 42: Pantalla de datos personales

Ilustración 43: Pantalla de datos médicos

3.4.4.- Actividades físicas

La **ilustración 44** muestra las diferentes vistas que componen el apartado relacionado con la actividad física del usuario. En esta sección, se podrá consultar sus datos, así como agregar aquellas actividades que el usuario haya realizado.

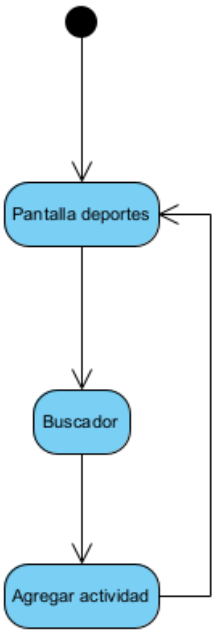


Ilustración 44: Diagrama de actividad del apartado de actividades

Ilustración 45: Vista general de actividades físicas

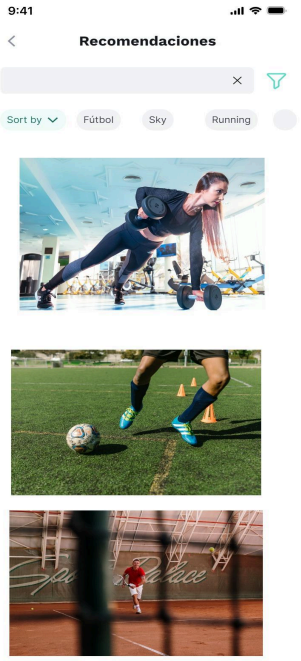


Ilustración 46: Buscador

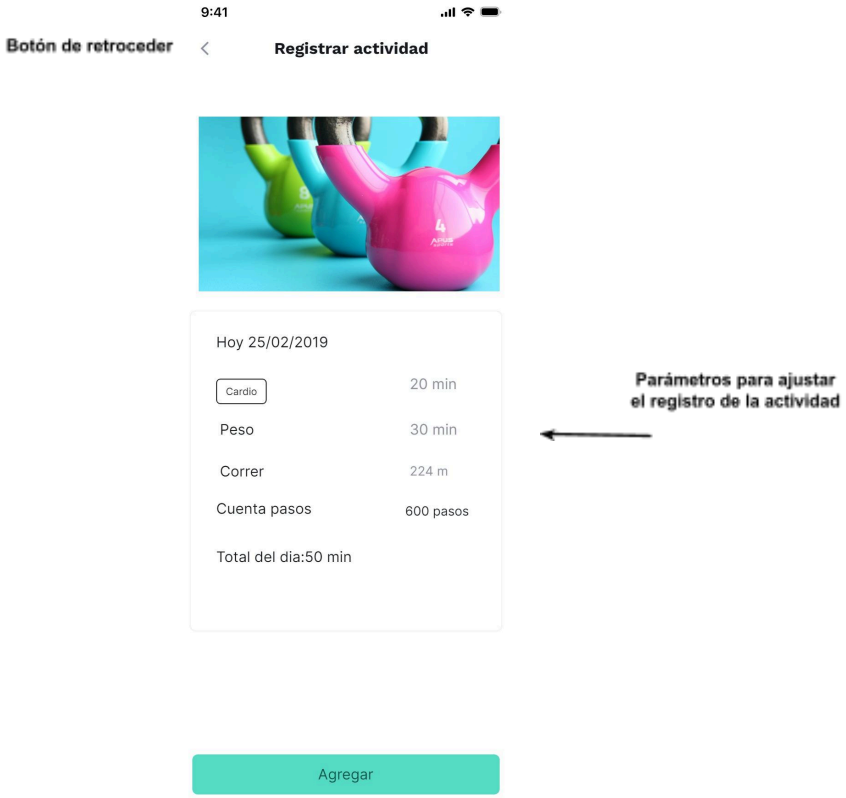


Ilustración 47: Pantalla del registro de actividad

3.4.5.- Alimentación

La **ilustración 48** muestra las diferentes vistas que componen el apartado relacionado con la alimentación. En esta sección, se podrá consultar sus datos, así como agregar aquellos alimentos ingeridos.

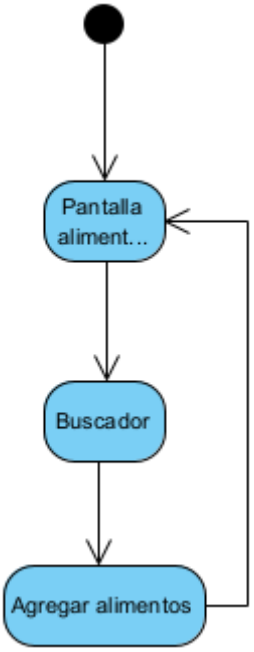


Ilustración 48: Diagrama de actividades del apartado de alimentación

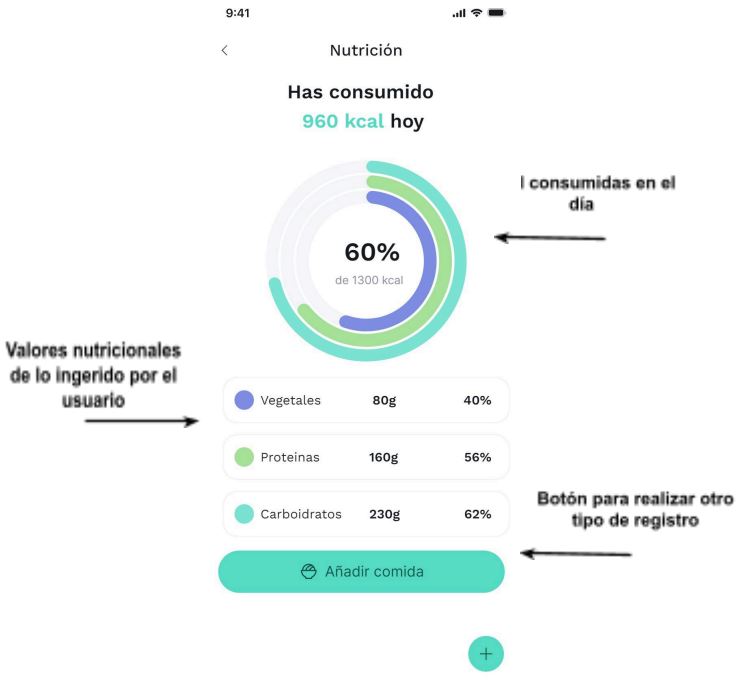


Ilustración 49: Pantalla del apartado de alimentación

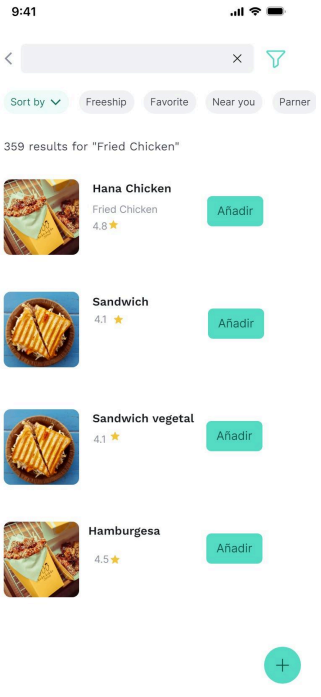


Ilustración 50: Buscador

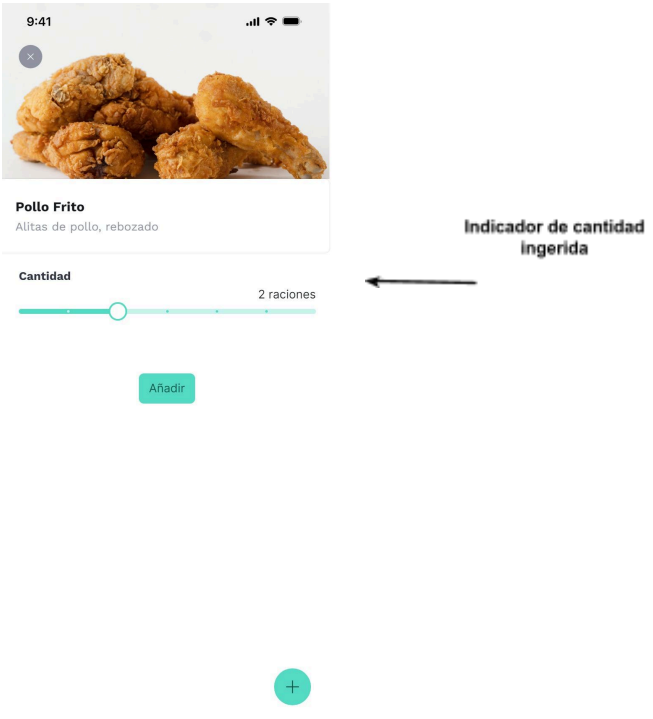


Ilustración 51: Pantalla del registro de alimentos

3.4.6.- Consejos

En la siguiente imagen, se muestra un ejemplo de un recurso educativo/consejo que los usuarios pueden obtener gracias a una inteligencia artificial implementada. Esta tecnología les permite acceder a esta información en cuestión de segundos. En nuestro caso tendremos un asistente personal (*Samantha)* que hará uso de *Gemini* para poder generar dichos recursos.

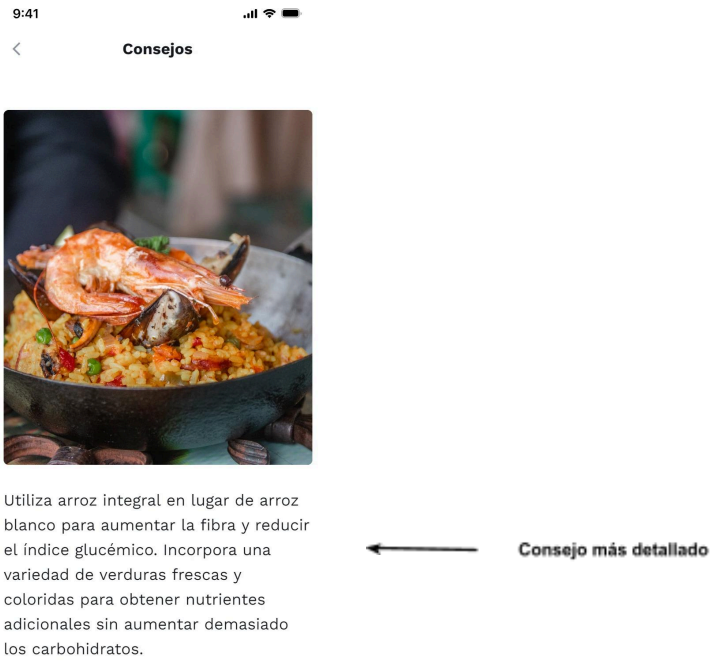


Ilustración 52: Pantalla del apartado de recursos educativos

3.5.- Plan de pruebas

Como parte del proceso de desarrollo del proyecto, se ha establecido un plan de pruebas para verificar el correcto funcionamiento de la aplicación. Siguiendo una metodología ágil/iterativa, se han definido varias pruebas de aceptación que se ejecutarán al final de cada iteración. Este enfoque nos permitirá validar que se cumplen todos los requisitos establecidos en cada etapa del desarrollo.

Cada versión de la aplicación será entregada a un grupo de usuarios diversos, que consta de tres tipos distintos. En primer lugar, contaremos con la retroalimentación de un diabético con 10 años de experiencia, quien se encargará de verificar que la aplicación cubra todas sus necesidades diarias. Además, un padre participará en el proceso, brindando asistencia en aquellos aspectos que puedan presentar dificultades, especialmente desde la perspectiva de cuidar a un hijo diabético. Por último, una persona mayor probará la aplicación para evaluar su impacto en su vida diaria, ayudándonos a identificar posibles fallos o áreas de mejora relacionadas con la accesibilidad y la facilidad de uso para este grupo demográfico. Esta retroalimentación directa nos permitirá detectar y abordar los problemas de manera oportuna, asegurando así que la aplicación sea verdaderamente útil y efectiva para todos los usuarios previstos.

Nuestro plan de pruebas estará compuesto por los siguientes tipos de pruebas:

- **Pruebas de aceptación**: hemos ido estableciendo una serie de pruebas que han sido realizadas por los diferentes usuarios finales, los cuales han sido los encargados de validar que la aplicación cumple con las necesidades y expectativas establecidas en el **apartado 2.3**, mediante los criterios de aceptación de cada historia de usuario. Además, estas pruebas contribuyeron a mejorar la navegación de los usuarios en la aplicación, asegurando una experiencia óptima.

- **Pruebas de caja negra**: hemos querido centrarnos especialmente en garantizar la correcta gestión del acceso a bases de datos externas y en evitar cualquier posible problema asociado con las estructuras de datos al almacenar la información manejada por el usuario en la aplicación.

- **Pruebas de compatibilidad:** durante las diferentes interacciones, se les proporcionó a los usuarios un archivo *APK* de la aplicación para que realizarán las pruebas en sus dispositivos. Esto permitió comprobar la diferencia de rendimiento entre diversos dispositivos y asegurarse de que la aplicación funcionará de manera óptima en una variedad de plataformas y configuraciones.

Este enfoque de pruebas nos permitirá verificar que cada componente y característica de la aplicación cumple con su funcionamiento esperado para los

diferentes tipos de usuarios previstos, y nos ayudará a identificar cualquier problema de integración con el resto del sistema. Esta validación exhaustiva garantizará que la aplicación cumpla con los más altos estándares de calidad y satisfaga las necesidades de todos los usuarios de manera efectiva y eficiente.

A continuación, en el apartado **5.-Pruebas** detallaremos las diferentes pruebas que se han realizado a lo largo del desarrollo, destacando los hallazgos y las acciones tomadas para abordar cualquier problema identificado.

4.- Implementación

4.1.- Arquitectura

La arquitectura de *software* de un sistema muestra la estructura del sistema y explica el comportamiento esperado. Apoya y proporciona una base sólida sobre la que se puede construir el *software*.

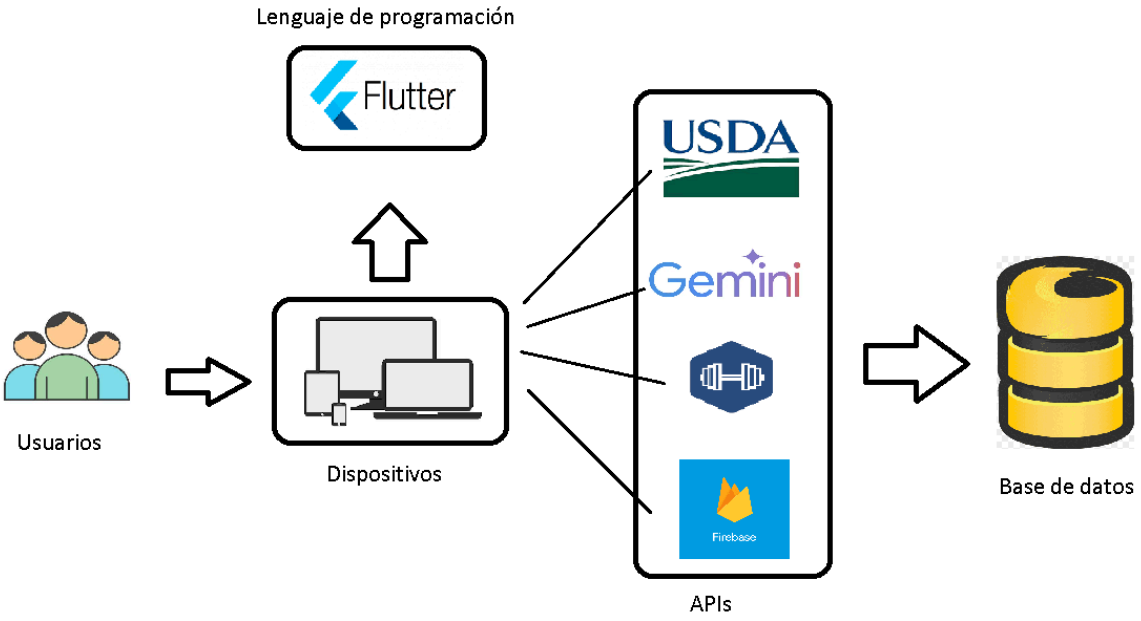


Ilustración 53: Diagrama de arquitectura de *software*

Según se observa en la **ilustración 53**, los usuarios tendrán la capacidad de utilizar la aplicación desarrollada mediante el lenguaje *Flutter*, desde cualquier dispositivo móvil u ordenador. Estos dispositivos se conectarán a través de las distintas *APIs* (*Gemini, USDA Food, Wger…*) implementadas para acceder a las diversas funcionalidades, mientras que la gestión de los datos estará a cargo de *Firebase Database*.

4.2.- Detalles sobre implementación

En esta sección, comentaremos algunos de los aspectos más destacables de la fase de desarrollo de nuestra aplicación. Donde hablaremos sobre las herramientas que se han ido utilizando para poder llevar a cabo un seguimiento del desarrollo, la tecnología que se ha empleado para la implementación del prototipado, la generación de su correspondiente documentación y por último de las diferentes bibliotecas usadas.

4.2.1.- Metodología y herramientas de seguimiento

Como se ha comentado anteriormente en el apartado **1.3.4.- SCRUM vs Kanban** la metodología ha sido implementada mediante la herramienta Trello, donde se ha creado un tablero con cuatro columnas: "Por hacer", "En progreso", "Revisión" y "Listo para lanzar".

Cada tarea del proyecto se ha registrado como una tarjeta en el tablero. Cada una contiene una breve descripción de la tarea, así como dos etiquetas: una indicando la iteración correspondiente y otra señalando la prioridad de la tarea.

Dado el enfoque adoptado, no se han programado reuniones diarias breves (*stand-ups*) o semanales. En su lugar, las reuniones se han llevado a cabo cuando se ha completado una gran parte de las tareas de cada iteración, lo que ha permitido realizar el feedback correspondiente.

A medida que el desarrollo ha progresado, las tarjetas de las tareas se han movido de una columna a otra en el tablero de Trello. Este enfoque ha proporcionado un seguimiento actualizado del progreso del desarrollo de la aplicación de manera clara y efectiva.

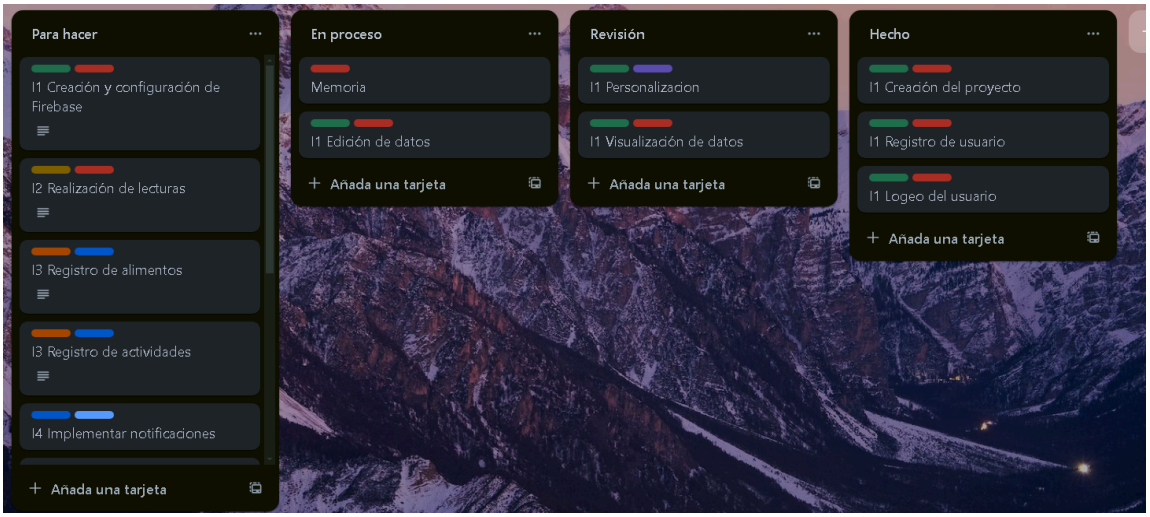


Ilustración 54: Tablón de Trello

Ilustración 55: Ejemplo de tarjeta

4.2.2.- Entorno de desarrollo

En este apartado se proporcionará una descripción detallada de los recursos físicos y lógicos utilizados a lo largo del proyecto. Esto incluye tanto el *hardware* utilizado, como los ordenadores y los dispositivos móviles, así

como el *software* empleado. Esta información es esencial para comprender el entorno de desarrollo del proyecto y garantizar su replicabilidad y mantenimiento a lo largo del tiempo.

- **Hardware**
  Para realizar todo el desarrollo del proyecto se ha utilizado un ordenador *MSI Pulse GL66* (2022), con un procesador Intel Core i7, 16 GB de RAM y 1TB de almacenamiento SSD. Para la parte de las pruebas, se ha usado un dispositivo móvil *Xiaomi Poco X4 Pro 5G* con la versión MIUI Global 14.0.7 y *Android* 13.

- **Software**
  Con respecto a los recursos lógicos usados, podemos destacar el uso del Windows 10 Pro como sistema operativo, las versiones estables de los IDEs con los plugins necesarios para poder trabajar en el desarrollo de una aplicación multiplataforma, *Visual Studio Code* 1.83.1 y *Android Studio Giraffe* | 2022.3.1 Patch 1, *Firebase* 12.7.0 para gestionar las bases de datos y *GitHub* para llevar un control de versiones

4.2.3.- Tecnologías utilizadas

Esta sección proporciona una visión general de las herramientas tecnológicas utilizadas, incluyendo sus versiones específicas, lo que permite comprender mejor el entorno de desarrollo y garantizar la coherencia y compatibilidad del proyecto.

El proyecto se ha desarrollado utilizando el lenguaje de programación *Dart* en su versión estable 3.2.6. Además, para aprovechar las capacidades del lenguaje de manera óptima, se ha empleado el *framework Flutter* en su versión estable 3.19.6. *Flutter* ha simplificado significativamente el desarrollo de las interfaces de usuario, proporcionando una experiencia de desarrollo más eficiente.

Además de estas tecnologías fundamentales, se ha integrado *Firebase* en su versión 12.7.0 para gestionar los servidores y las bases de datos del proyecto, así como para implementar métodos de autenticación que permitan la creación de una aplicación multiplataforma.

Otro componente clave utilizado en el proyecto son los *Provider* que ofrece *Flutter*. Estos han sido esenciales para gestionar la actualización de estados de la aplicación de manera eficiente. Por ejemplo, se han utilizado para garantizar que la aplicación se actualice automáticamente después de realizar cambios en los datos del usuario.

Esta combinación de tecnologías ha proporcionado una base sólida para el desarrollo del proyecto, permitiendo la creación de una aplicación robusta y fácil de mantener.

```
runApp(
  /// Inicializacion de los Provider
  MultiProvider(
    providers: [
      ChangeNotifierProvider(
        create: (context) => UsuarioProvider(),
      ),  // ChangeNotifierProvider
      ChangeNotifierProvider(
        create: (context) => ActividadViewModel(
        ),  // ActividadViewModel
      ),  // ChangeNotifierProvider
      ChangeNotifierProvider(
        create: (context) => SeleccionarFechaProvider(),
        child: MyApp(),
      ),  // ChangeNotifierProvider
      ChangeNotifierProvider(
        create: (_) => DevicesBloc(deviceManager),
        child: const ConectarDispositivos(),
      ),  // ChangeNotifierProvider
    ],
    child: MyApp(),
  ),  // MultiProvider
);
```

Ilustración 56: Providers usados

4.2.4.- Documentación de código

La documentación del código y los procesos del proyecto desempeñan un papel crucial en la comprensión, mantenimiento y colaboración efectiva en el desarrollo de *software*.

En este proyecto, hemos seguido el estándar *DartDoc*, que es el recomendado para *Dart* y *Flutter*, para nuestra documentación. Antes de cada clase, método y función, hemos incluido ciertos comentarios que proporcionan descripciones claras del comportamiento y propósito de cada elemento del código.

```dart
/// Clase que representa un usuario en el sistema.
class Usuario {
  /// Nombre del usuario.
  String nombre;
  /// Apellidos del usuario.
  String apellidos;
  /// Dirección de correo electrónico del usuario.
  String email;
  /// Contraseña del usuario.
  String password;
  /// Fecha de nacimiento del usuario.
  DateTime fechaNacimiento;
  /// Nivel de actividad del usuario.
  String nivelActividad;
  /// Altura del usuario en metros.
  double altura;
  /// Peso del usuario en kilogramos.
  double peso;
  /// Unidad utilizada para las mediciones de comida.
  String unidadComida;
  /// Unidad utilizada para las mediciones de glucosa.
  String unidadMedida;
  /// Valor de hiperglucemia del usuario.
  double hiperglucemia;
  /// Valor de hipoglucemia del usuario.
  double hipoglucemia;
  /// Nivel de glucosa a obtener del usuario.
  double nivel_objetivo;
  /// URL de la imagen de perfil del usuario.
  String imagenUrl;
```

Ilustración 57: Ejemplo de uso de DartDoc

Este proceso está integrado en nuestro flujo de trabajo de desarrollo. Cada vez que realizamos cambios significativos en el código, actualizamos la documentación correspondiente para reflejar esos cambios y garantizar que su coherencia con el código actualizado.

Para generar la documentación del proyecto, se ha utilizado la herramienta *DartDoc*, que permite generar documentación de forma automática a partir del código fuente. Esto ha facilitado enormemente la creación y actualización de la documentación, asegurando que esté siempre sincronizada con el código base y proporcionando una referencia clara y precisa para los desarrolladores que trabajan en el proyecto.

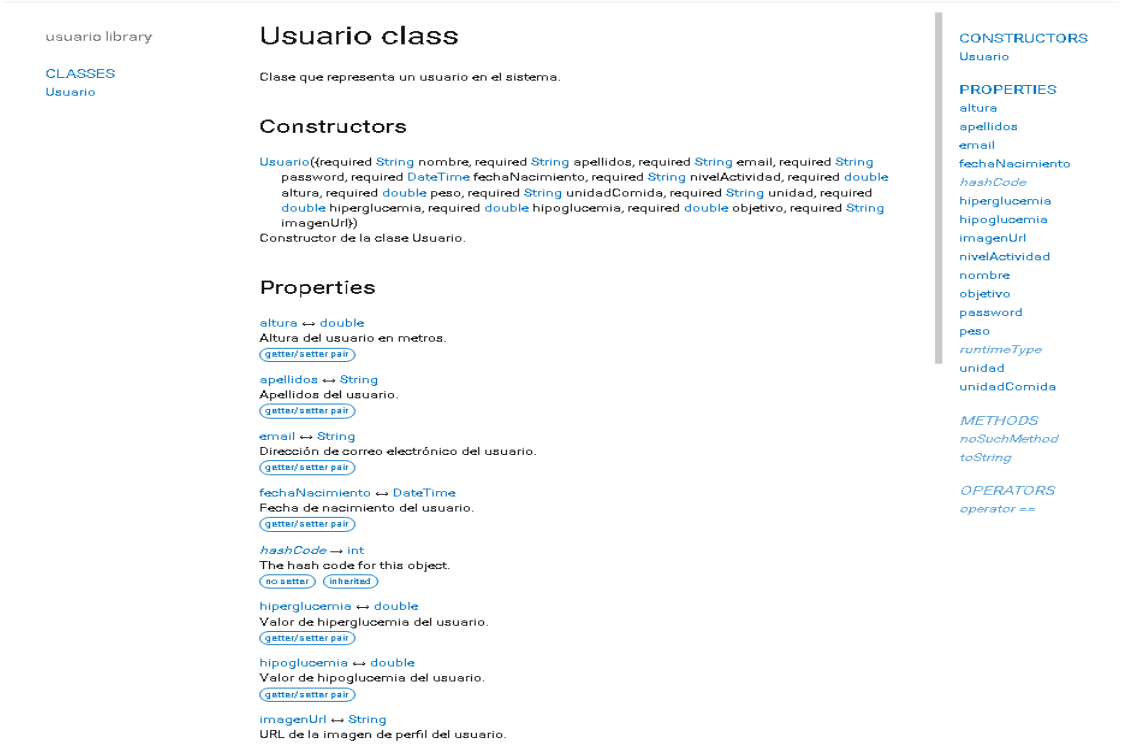


Ilustración 58: Ejemplo de la documentación generada por DartDoc

Para generar esta información, simplemente ejecutamos en nuestro entorno de desarrollo el comando **“dart doc .”**. Esto nos permitirá generar una carpeta llamada **“doc/api”** en nuestro proyecto, que contiene una carpeta para cada biblioteca documentada de nuestro proyecto con dicho formato. En ellas, se podrá encontrar los diferentes HTML que componen nuestra documentación.

En nuestro caso, se han documentado 66 librerías.

Dentro del comando **“dart doc .”**, encontramos diferentes opciones que nos pueden ser de gran utilidad.

- **-no-code**: esto no meterá código fuente en nuestra documentación.
- **-m, -mode**: nos permite definir cómo se generan las páginas *HTML*, dentro de esta opción encontramos:
  - **live-nav**: es la opción por defecto, que genera un fichero llamado nav.json que contiene toda la información necesaria.
  - **static**: genera los ficheros *HTML* completamente estáticos.
- **-v,-verbose**: muestra toda la información durante la generación, dentro de ella encontramos:
  - **--include-api**: agrega las librerías SDK utilizadas
  - **–link-api**: proporciona un enlace con la documentación online de *Dart*.
  - **–out**: indica el directorio donde se generá la documentación, que por defecto es **“doc/api”**.

Dicha documentación se encuentra añadida en el **apéndice de ejemplo de documentación.**

4.2.5.- Arquitectura de una aplicación Flutter

Los *frameworks* juegan un papel fundamental en el desarrollo de *software*, proporcionando estructuras y herramientas que simplifican y aceleran el proceso de creación de aplicaciones. En este apartado, se presentarán aquellos que han sido utilizados en el proyecto, destacando su función, importancia y contribución al desarrollo de la aplicación. Comprenderlos es esencial para apreciar la arquitectura y las funcionalidades implementadas en el proyecto, así como para garantizar su mantenimiento y escalabilidad a lo largo del tiempo.

Como ya se ha mencionado en apartados anteriores, se ha utilizado el *framework Flutter* para la creación de la aplicación. El cuál sigue un enfoque de desarrollo basado en widgets, lo que significa que la interfaz de usuario se construye mediante la composición de dichos elementos, como pueden ser botones, campos de texto, listas, etc. Estos *widgets* son altamente personalizables y se pueden combinar de diversas maneras para crear interfaces complejas y ricas en funcionalidades.
Además, *Flutter* ofrece un conjunto completo de herramientas y bibliotecas para facilitar el desarrollo, incluyendo soporte para la gestión de estados, acceso a servicios en la nube, animaciones o pruebas automatizadas.

4.2.5.1 Clase main

El archivo *main* de un proyecto *Flutter* sirve como punto de entrada principal de la aplicación. En él, se inicializan y configuran elementos cruciales como *Firebase*, proveedores de estado, servicios externos y rutas de navegación. Esencialmente, el archivo main actúa como el núcleo de la aplicación *Flutter*, coordinando todas las partes y asegurando su correcto funcionamiento.

En las siguientes imágenes, se ilustra la inicialización de la conexión del proyecto con *Firebase*, la configuración de las notificaciones locales, así como la inicialización de los servicios externos de *Wger, USDA Food*, *Firebase Authentication*… y los diferentes proveedores.

```dart
/// Función principal de la aplicación.
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await NotificacionServicio.init();

  WidgetsFlutterBinding.ensureInitialized();
  Fimber.plantTree(DebugTree());

  final vitalClient = vital_core.VitalClient()
    ..init(
      region: Region.eu,
      environment: Environment.sandbox,
      apiKey: Claves.VitalKey,
    );

  final DeviceManager deviceManager = DeviceManager();

  var initialNotification =
  await flutterLocalNotificationsPlugin.getNotificationAppLaunchDetails();
  if (initialNotification?.didNotificationLaunchApp == true) {
    Future.delayed(Duration(seconds: 1), () {
      navigatorKey.currentState!.pushNamed('/another',
          arguments: initialNotification?.notificationResponse?.payload);
    });  // Future.delayed
  }

  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
```

Ilustración 59: Inicialización de *Firebase* y servicios externos

```
runApp(
  /// Inicializacion de los Provider
  MultiProvider(
    providers: [
      ChangeNotifierProvider(
        create: (context) => UsuarioProvider(),
      ),  // ChangeNotifierProvider
      ChangeNotifierProvider(
        create: (context) => ActividadViewModel(
        ),  // ActividadViewModel
      ),  // ChangeNotifierProvider
      ChangeNotifierProvider(
        create: (context) => SeleccionarFechaProvider(),
        child: MyApp(),
      ),  // ChangeNotifierProvider
      ChangeNotifierProvider(
        create: (_) => DevicesBloc(deviceManager),
        child: const ConectarDispositivos(),
      ),  // ChangeNotifierProvider
    ],
    child: MyApp(),
  ),  // MultiProvider
);
```

Ilustración 60: Configuración de los proveedores

En la **ilustración 61**, se muestra el fragmento donde se establece la configuración de temas, rutas y del título de la aplicación.

```
/// Clase principal de la aplicación.
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'GlucoLife',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),  // ThemeData
      initialRoute: '/',
      routes: {
        '/': (context) => BienvenidaVista(),
        '/home': (context) => HomeVista(),
      },
    );  // MaterialApp
  }
}
```

Ilustración 61: Clase MyApp

4.2.5.2 Widget

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Registrar Medicamento'),
      backgroundColor: Colors.green,
    ),  // AppBar
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          TextFormField(
            decoration: InputDecoration(labelText: 'Nombre del medicamento'),
            controller: _nombreMedicamentoController,
          ),  // TextFormField
          TextFormField(
            decoration: InputDecoration(labelText: 'Dosis'),
            keyboardType: TextInputType.number,
            controller: _dosisController,
          ),  // TextFormField
```

Ilustración 62: Ejemplo de un Widget

En la siguiente ilustración podemos observar un ejemplo de cómo se definen los widgets en el *framework Flutter*.
Dichos elementos contienen la siguiente estructura:

- **Clase del Widget**: Pueden ser de dos tipos principales: *StatelessWidget* para aquellos que no necesitan mantener un estado interno o *StatefulWidget* para aquellos que necesiten cambiar su estado a lo largo del tiempo.
  En este caso se trata de un *StatefulWidget.*

- **Método *build***: este método retorna la estructura de la interfaz de usuario del widget.

- **Estructura de la interfaz de usuario**: Dentro del método build, se retorna la estructura de la interfaz. En este caso se devuelve un Scaffold, el cual es un widget de nivel superior que proporciona una estructura de página básica para una aplicación. Dicho tipo contiene un *AppBar* que es una barra que se localiza en la parte superior de la pantalla. En este caso, se establece el título de la vista. El *body*, que es donde se define el contenido principal de la pantalla. En este caso, el cuerpo tiene un widget *Padding* que se utiliza para agregar espacio de relleno entre los elementos, que en este ejemplo es una columna de *TextFormField.*

- **Propiedades**: Cada widget puede tener propiedades específicas y estilos aplicados a ellos. En este ejemplo, se ha establecido una decoración en los *TextFormField.*

En el fragmento que se muestra en la ilustración anterior corresponde a un *widget* del apartado de medicamentos de la aplicación. En este caso posee:

- ***appBar***: para poder establecer el título de la sección.

- ***body***: donde se ha definido una serie de *TextFormField* que nos servirán para poder establecer los diferentes campos para introducir los datos necesarios que se requieren para registrar cualquier tipo de medicamento.

A continuación, se van a mostrar más ejemplos de algunos widgets que se han implementado en la aplicación.

```
DropdownButtonFormField(
  decoration: InputDecoration(labelText: 'Intervalo'),
  value: _selectedIntervalo,
  items: intervalos.map((String intervalo) {
    return DropdownMenuItem(
      value: intervalo,
      child: Text(intervalo),
    );  // DropdownMenuItem
  }).toList(),
  onChanged: (value) {
    setState(() {
      _selectedIntervalo = value as String?;
      if (_selectedIntervalo == 'Personalizado') {
        _intervaloPersonalizado = null;
      }
    });
  },
),  // DropdownButtonFormField
```

Ilustración 63: DropDownButton

En este ejemplo, se puede observar un ejemplo sobre un *DropDownButton* (lista desplegable), el cuál nos permitirá seleccionar el periodo de repetición de nuestras alarmas establecidas para cada medicamento. Este *widget* se compone de :

- ***decoration***: permite definir la decoración visual del widget.

- *value*: especifica el valor que actualmente está seleccionado en la lista
- ***items***: es una lista desplegable que mostrará las diferentes opciones establecidas en la lista "intervalos". Cuando el usuario seleccione una de las opciones, el valor de *value* se actualiza automáticamente con el valor correspondiente.
- ***onChanged***: es una función nos permite actualizar el *DropDownButton,* esto se ha usado para que el usuario pueda establecer sus propios periodos de alarmas cuando marca la opción de "Personalizado".

```
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => LoginVista()),
    );
  },
  style: ElevatedButton.styleFrom(
    primary: Colors.white,
    onPrimary: Colors.green,
    padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(30.0),
    ),  // RoundedRectangleBorder
  ),
  child: Text(
    'Comenzar',
    style: TextStyle(fontSize: 16),
  ),  // Text
),  // ElevatedButton
```

Ilustración 64: ElevatedButton

En la **ilustración 64** podemos ver un ejemplo de un *ElevatedButtom* (botón con elevación), el cual permitirá al usuario en esta ocasión avanzar hasta la pantalla de logueo. Este widget contiene:

- ***onPressed***: especifica la función que se acciona cuando el botón es pulsado, en este caso se nos redirige a la vista de logueo.

- ***style***: nos permite definir la apariencia del botón.

- ***child***: especifica el texto que se mostrará dentro del botón, que en este ejemplo es "Comenzar".

4.2.5.3 Ejemplo

```
class BuscadorActividadesVista extends StatefulWidget {
  @override
  _BuscadorActividadesVistaState createState() =>
      _BuscadorActividadesVistaState();
}

class _BuscadorActividadesVistaState extends State<BuscadorActividadesVista> {
  final TextEditingController _searchController = TextEditingController();

  @override
  void initState() {
    super.initState();

    /// Llamada al provider para poder mantener la vista actualizada
    Provider.of<ActividadViewModel>(context, listen: false)
        .obtenerActividades();
  }

  @override
  Widget build(BuildContext context) {
    /// Llamada al provider para poder mantener la vista actualizada
    final actividadViewModel = Provider.of<ActividadViewModel>(context);

    return Scaffold(
      appBar: AppBar(
        title: Text('Buscador'),
        backgroundColor: Colors.green,
        centerTitle: true,
      ),  // AppBar
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: TextField(
```

Ilustración 65 : Ejemplo de vista

4.2.6.- Bibliotecas y servicios externos

En el desarrollo de aplicaciones de *software*, es una práctica común utilizar bibliotecas y servicios externos para aprovechar funcionalidades preexistentes y agilizar el proceso de desarrollo. Estos recursos externos ofrecen una amplia gama de herramientas y características que pueden integrarse fácilmente en el proyecto. La comprensión detallada de estas bibliotecas y servicios externos es esencial para entender la arquitectura y las funcionalidades implementadas en el proyecto. Además, garantiza el mantenimiento y la escalabilidad a largo plazo del *software* desarrollado. Este apartado proporciona una visión general de las bibliotecas y servicios

externos utilizados en el proyecto, junto con una descripción concisa de su función y contribución al desarrollo.

Software	Versión	Descripción
Flutter SDK	3.16.9	Utilizado para el desarrollo de interfaces de usuario multiplataforma.
FlutterFire	0.2.7	Vincula *Firebase* con el proyecto para acceder a sus servicios.
Provider	6.1.1	Gestiona de forma simplificada el estado de la aplicación.
Firebase_core	2.24.2	Esencial para configurar y acceder a los servicios de *Firebase*.
Firebase_auth	4.16.0	Biblioteca de *Firebase* para la autenticación de usuarios.
Cloud_firestore	4.14.0	Proporciona acceso a la base de datos de *Cloud Firestore* de *Firebase*.
Http	1.1.0	Ofrece capacidades de cliente HTTP para realizar solicitudes.
Horizontal_week_calendar	0.0.11	Proporciona un calendario semanal de forma horizontal.
Fl_chart	0.65.0	Permite crear gráficos animados y personalizables en aplicaciones *Flutter*.
Circular_profile_avatar	2.0.5	Facilita la creación de avatares circulares.
Timezone	6.2.1	Proporciona funcionalidades para trabajar en diferentes zonas horarias.
Google_sign_in	1.5.1	Facilita la autenticación de usuarios con *Google*.
Google_Gemini	11.6.9	Ofrece la posibilidad de trabajar con la IA de *Google*.
Rxdart	11.6.9	Proporciona extensiones reactivas para *Dart*.

Flutter_datetime_picker	1.0.7	Proporciona la selección de fechas y horas personalizables.
Alarm	17.0.0	Ofrece funcionalidades para programar alarmas.
Firebase_storage	11.6.9	Permite el almacenamiento de archivos en la nube utilizando *Firebase Storage*.
Image_picker	1.0.7	Permite a los usuarios seleccionar imágenes de la galería o tomar fotos con su cámara.
Flutter_local_notifications	17.0.0	Facilita la programación y visualización de notificaciones locales de la aplicación.
Vital_devices	3.1.8	Es un SDK que nos permite conectarnos a una serie de dispositivos.
Vital_core	3.1.8	Nos permite enviar datos desde un dispositivo a un servidor.
Disposebag	1.5.0	Este paquete ayuda a cancelar *StreamSubscriptions* y cerrar *Sinks.*

Tabla 5: Bibliotecas utilizadas

Todas estas dependencias se han agregado en el archivo *pubspec.yaml* de nuestro proyecto, ya que es el responsable de gestionar la configuración de ellas en la aplicación.

```
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  firebase_core: ^2.24.2
  horizontal_calendar: ^1.1.2
  firebase_auth: ^4.16.0
  cloud_firestore: ^4.14.0
  http: ^1.1.0
  provider: ^6.1.1
  firebase_storage: ^11.6.9
  fl_chart: ^0.65.0
  circular_profile_avatar: ^2.0.5
  animated_floating_buttons:
  timezone: ^0.9.2
  image_picker: ^1.0.7
  google_sign_in: ^6.2.1
  google_gemini:
  flutter_local_notifications: ^17.0.0
  rxdart: ^0.27.7
  flutter_datetime_picker: ^1.5.1
  alarm: ^3.0.14
  vital_devices: ^3.1.8
  vital_core: ^3.1.8
  disposebag: ^1.5.0
```

Ilustración 66: Dependencias usadas

Aparte de estas bibliotecas mencionadas anteriormente, hemos usado los siguientes servicios externos.

A continuación, se mostrarán diferentes ilustraciones de cómo la aplicación interactúa con aquellas que se han considerado más relevantes.

4.2.6.1.- Wger

Wger es una *API* pública que ofrece una biblioteca para que los usuarios agreguen ejercicios o información nutricional con el fin de crear una aplicación fitness completa.

Para la funcionalidad de registrar las actividades físicas que se ha implementado, se ha utilizado el siguiente *endpoint* GET /api/v2/exerciseinfo [6], de la cual hemos podido obtener información importante como el nombre y una breve descripción de cada ejercicio.

6. https://wger.de/api/v2/schema/ui (Febrero, 2024)

```dart
/// Servicio para interactuar con la API de Wger.
class WgerService {
  final String apiUrl = 'https://wger.de/api/v2/exerciseinfo/';

  /// Obtiene la lista de ejercicios desde la API de Wger.
  ///
  /// Retorna una lista de objetos de tipo [Actividad].
  ///
  /// Lanza una excepción si no se pueden cargar los ejercicios.
  Future<List<Actividad>> buscarActividad() async {
    final response = await http.get(Uri.parse(apiUrl));
    DateTime fechaActual = DateTime.now();
    if (response.statusCode == 200) {
      final List<dynamic> data = json.decode(response.body)['results'];
      return data
          .map((item) => Actividad(
        nombre: item['name'],
        intensidad: "",
        tiempoRealizado: "",
        caloriasQuemadas: 0.0,
        fechaRegistro: fechaActual,
      ))
          .toList();
    } else {
      throw Exception('Error al cargar las actividades');
    }
  }
}
```

Ilustración 67: Acceso a *API* pública *Wger* (WgerServicio)

4.2.6.2.- USDA Food

USDA Food es una base de datos con datos nutricionales de una gran variedad de alimentos.

Al igual que con *Wger*, el *USDA* proporciona un punto de acceso desde el cual hemos extraído una cantidad significativa de información crucial para la funcionalidad del registro de alimentos, incluyendo nombres y valores nutricionales. Para realizar las consultas correspondientes, hemos tenido que generar una clave de *API* privada que nos permitirá acceder a los datos. Esto se debe a que, como se puede observar en el sitio web oficial del *FoodData Center* [7,8], la estructura de los endpoints requiere dicha clave.

```dart
/// Clase que realiza búsquedas de alimentos utilizando la API de USDA.
class USDAFood {
  /// Realiza una búsqueda de alimentos utilizando la API de USDA.
  ///
  /// [query]: La cadena de búsqueda para la consulta de alimentos.
  ///
  /// Devuelve una excepcion cuando no se puede encontrar el alimento ingresado
  Future<List<Alimentos>> buscarAlimento(String query) async {
    final response = await http.get(
      Uri.parse(
        'https://api.nal.usda.gov/fdc/v1/foods/search?query=$query&api_key='+Claves.USDAFoodKey,
      ),
    );

    if (response.statusCode == 200) {
      final Map<String, dynamic> data = json.decode(response.body);
      if (data.containsKey('foods')) {
        List<dynamic> foods = data['foods'];

        DateTime fechaActual = DateTime.now();

        List<Alimentos> results = foods.map((food) {
          List<Nutriente> nutrientes = [];


          if (food['foodNutrients'] != null &&
              food['foodNutrients'].isNotEmpty) {
            nutrientes = food['foodNutrients']
                .where((nutrient) =>
                    nutrient['nutrientName'] == 'Protein' ||
                    nutrient['nutrientName'] == 'Carbohydrate, by difference' ||
```

Ilustración 68: Utilización del *API* pública *USDA Food* (USDAFoodServicio)

7. https://fdc.nal.usda.gov/api-guide.html#bkmk-6 (Febrero, 2024)
8. https://app.swaggerhub.com/apis/fdcnal/food-data_central_api/1.0.1#/FDC/postFoodsSearch (Febrero, 2024)

4.2.6.3.- Firebase

Como se ha mencionado previamente, *Firebase* ofrece una amplia gama de servicios que pueden potenciar diversos aspectos de nuestra aplicación. Entre los servicios implementados se incluyen:

- **Authentication**
  En varias ocasiones anteriores hemos destacado las múltiples opciones de autenticación de usuarios que *Firebase* ofrece para nuestras aplicaciones. Hasta ahora, hemos predominado el uso de la autenticación básica, tal y como se muestra en la **ilustración 69**, la cual consiste en que los usuarios ingresen su correo electrónico y contraseña. Al ingresar estos datos, *Firebase* los valida y, si son correctos, permite el acceso a la aplicación.

  Además de esta opción, hemos dado los primeros pasos para implementar la autenticación mediante cuentas de *Gmail*. Esta función aprovecha las capacidades proporcionadas por *Google*, permitiendo a los usuarios acceder a la aplicación utilizando sus cuentas de Gmail, como se muestra en la **imagen 70**.

```
await _auth.signInWithEmailAndPassword(email: email, password: password);
```

Ilustración 69: Uso de la autenticación básica de *Firebase* (LoginServicio)

```
GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();
GoogleSignInAuthentication? googleAuth = await googleUser?.authentication;
```

Ilustración 70: Uso de la autenticación de *Google* de *Firebase* (LoginServicio)

- **Firestore/ Database**
  En la siguiente imagen se ilustra cómo se gestiona el almacenamiento de usuarios en *Firebase Database o Firestore*. Cuando se registra el primer usuario en la aplicación, se crea una colección llamada "usuarios" en la base de datos. En esta colección, se guarda el usuario con todos sus datos asociados, como nombre, correo electrónico, ID único, entre otros.

  Este mismo proceso se repite para otros tipos de datos gestionados en la aplicación, como alimentos, actividades, y demás. Por ejemplo, si la aplicación gestiona información sobre alimentos, se crea una colección dedicada a ellos en la base de datos, donde se almacenan todos los alimentos con sus propiedades correspondientes.

```
if (firebaseUser != null && usuario != null) {
  await _firestore.collection('usuarios').doc(firebaseUser.uid).update({
    'altura': usuario.altura,
    'peso': usuario.peso,
    'hiperglucemia': usuario.hiperglucemia,
    'hipoglucemia': usuario.hipoglucemia,
    'nivelActividad': usuario.nivelActividad,
    'objetivo': usuario.nivel_objetivo,
    'unidad': usuario.unidadMedida,
    'unidadComida': usuario.unidadComida,
  });
}
```

Ilustración 71: Almacenamiento de los usuarios en la base de datos (RegistroServicio)

- **Storage**

  Como se mencionó previamente, *Firebase Storage* ofrece la capacidad de almacenar contenido multimedia de las aplicaciones.

  Al igual que con la base de datos, *Firebase* nos permite organizar el almacenamiento de contenido mediante el uso de carpetas. En nuestro caso, hemos decidido crear una carpeta llamada "usuarios" donde se almacenarán las fotos de perfil de cada persona.

  Para lograr esto, creamos una ruta de almacenamiento denominada *storagePath* que indica la ubicación dentro de *Firebase Storage* donde se guardarán las imágenes de perfil de los usuarios. Luego, utilizamos *storageReference* para poder almacenar correctamente la imagen correspondiente de cada usuario en *Firebase Database.*

```
if (imagePath != null) {
  String storagePath = 'usuarios/${firebaseUser.uid}/perfil.jpg';
  Reference storageReference = _storage.ref().child(storagePath);
  await storageReference.putFile(File(imagePath));
  usuario.imagenUrl = await storageReference.getDownloadURL();
}
```

Ilustración 72: Almacenamiento de imágenes en *Storage* (RegistroServicio)

```
return Row(
  children: [
    CircleAvatar(
      radius: 20.0,
      backgroundImage:
      NetworkImage(usuarioModel.obtenerUsuario?.imagenUrl ?? "https://cdn-icons-png.flaticon.com/512/3135/3135768.png")
    ), // CircleAvatar
    SizedBox(width: 8),
    Text('Bienvenido, ${usuarioModel.obtenerUsuario?.nombre ?? "Usuario"}'),
  ],
); // Row
```

Ilustración 73: Obtención de datos de *Firebase* (Home)

En la **ilustración 73**, observamos cómo se obtiene la foto de perfil y el nombre del usuario logueado para su correspondiente visualización en la pantalla *Home*.

4.2.6.4.- Vital

Como mencionamos previamente, "Vital" es un SDK (*Software Development Kit*) que facilita la conexión de dispositivos relacionados con la salud. Esto incluye dispositivos como smartwatches de marcas reconocidas como *Xiaomi*, *Google*, entre otros, así como lectores de glucosa como los de la marca *FreeStyle*.

Ha sido implementado en nuestra aplicación de la siguiente forma

- **Buscador de dispositivos**

```
/// Escanea dispositivos disponibles.
///
/// [context]: El contexto de la aplicación.
void escanear(BuildContext context) {
  dispositivosAdministradro.getConnectedDevices(deviceModel).then((devices) {
    connectedDevices = devices;
    notifyListeners();
  });

  scanSubscription =
      dispositivosAdministradro.scanForDevice(deviceModel).listen((newDevice) {
        if (!scannedDevices.contains(newDevice)) {
          scannedDevices.add(newDevice);
          notifyListeners();
        }
      }, onError: (error) {
        Fimber.i('Error al escanear los dispositivos: $error');
      });

  notifyListeners();
}
```

Ilustración 74: Buscador de dispositivos (VitalServicio)

El método presentado en la ilustración anterior detalla el proceso de vinculación con nuestro dispositivo

1. **Obtener dispositivos conectados**: Se realiza una consulta inicial para identificar los dispositivos que ya están conectados al dispositivo principal.

2. **Escaneo de dispositivos disponibles**: Se inicia un escaneo para detectar los dispositivos disponibles para conexión, incluyendo aquellos que aún no están vinculados pero están dentro del rango de detección del dispositivo principal.

3. **Vinculación de dispositivos**: Después de recopilar la lista de dispositivos disponibles, se procede a la acción de vinculación, estableciendo la conexión entre el dispositivo principal y los dispositivos seleccionados.

- **Emparejar dispositivos**

```
/// Empareja el dispositivo seleccionado.
///
/// [context]: El contexto de la aplicación.
/// [scannedDevice]: El dispositivo escaneado que se va a emparejar.
void conectar(BuildContext context, ScannedDevice scannedDevice) {
  Fimber.i('Request to pair device: ${scannedDevice.deviceModel.name}');

  dispositivosAdministradro.pair(scannedDevice).then((event) {
    ScaffoldMessenger.of(context)
        .showSnackBar(const SnackBar(content: Text("Vinculación correcta")));
    Fimber.i('Vinculación correcta con el dispositivo: ${scannedDevice.deviceModel.name}');

    notifyListeners();
  }, onError: (error, stackTrace) {
    Fimber.i(error.toString(), stacktrace: stackTrace);
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text("Error al vincular: $error")));
  });
}
```

Ilustración 75: Emparejamiento de dispositivos (VitalServicio)

Este método nos permite establecer la vinculación con nuestros dispositivos, mostrándonos avisos de si todo ha salido exitosamente o se ha producido algún error durante el emparejamiento.

- **Lectura de datos**

```dart
/// Lee los datos del dispositivo seleccionado.
///
/// [context]: El contexto de la aplicación.
/// [scannedDevice]: El dispositivo escaneado del cual se leerán los datos.
void leerDatos(BuildContext context, ScannedDevice scannedDevice) {
  Fimber.i(
      'Request to read data from device: ${scannedDevice.deviceModel.name}');

  switch (scannedDevice.deviceModel.kind) {
    case DeviceKind.glucoseMeter:
      dispositivosAdministradro.readGlucoseMeterData(scannedDevice).then(
            (List<QuantitySample> newReadings) {
          Fimber.i(
              'Received ${newReadings.length} samples from device: ${scannedDevice.deviceModel.name}');

          for (var newReading in newReadings) {
            if (!glucometroResultados
                .any((e) => e.startDate == newReading.startDate)) {
              glucometroResultados.add(newReading);
            }
          }

          glucometroResultados
              .sort((a, b) => b.startDate.compareTo(a.startDate));

          notifyListeners();
        },
        onError: (error, stackTrace) =>
```

Ilustración 76: Lectura de datos desde el dispositivo (VitalServicio)

Este método maneja la lectura de datos del dispositivo vinculado con nuestro móvil.

1. **Identifica el dispositivo**: verifica si se trata de un lector de glucosa.

2. **Lectura de datos**: se procede a la lectura de datos.

3. **Procesamiento de los datos**: tras realizar las lecturas, se procede a ordenar los datos registrados en orden descendiente filtrados por la fecha de registro.

4. **Manejo de errores**: en caso de que ocurra algún error durante el proceso de lectura de datos, se invoca una función de manejo de errores, la cual puede proporcionar información sobre el error al usuario o al sistema para su gestión.

Como se mencionó anteriormente, ésta funcionalidad ha sido diseñada como un *mock*. En este enfoque, hemos querido reflejar únicamente el método de la *API* de *Vital* “Lectura de datos”. Dicha implementación

genera valores de manera periódica, imitando así el escaneo continuo de un sensor real. Estos valores son sometidos a una fórmula estadística, lo que nos permite representar con mayor exactitud una lectura real.

```
addDataTimer = Timer.periodic(const Duration(seconds: 2), (timer) {
  setState(() {
    double actual = math.Random().nextDouble() * 100 + 50;
    double ideal = actual + (math.Random().nextDouble() * 20 - 10);
    /// Llamada al metodo para almacenar los datos en Firebase
    _homeViewModel.guardarDatosGlucosa(actual);
    sinPoints.add(FlSpot(xValue, actual));
    cosPoints.add(FlSpot(xValue, ideal));
  });
  xValue += step;
});
```

Ilustración 77: Implementación del *mock* (GraficaActual)

Para realizar esta implementación, hemos utilizado la guía que ofrecía la página oficial de *Flutter* [9] y su ejemplo [10] correspondiente.

4.2.6.5.- Google Gemini

Este paquete ofrece una sólida conexión entre tu aplicación y la innovadora IA *Gemini* de *Google*. Facilita la integración fluida de las capacidades de *Gemini* en tu aplicación, abriendo un amplio abanico de posibilidades para crear experiencias inteligentes, innovadoras y atractivas que transformen la interacción de los usuarios.

En nuestro caso, *Gemini* se ha utilizado para ofrecer una amplia variedad de recursos educativos, permitiéndo al usuario mantenerse informado sobre las últimas novedades de su enfermedad o simplemente mejorar su día a día. Dicha información es generada mediante una mensaje de texto que ingresarán en la tarjeta “Consejos del día” de la página principal de la aplicación, como se puede observar en la **ilustración 93**. Esto permitirá que la inteligencia artificial ofrezca una respuesta acorde a la pregunta realizada.

A continuación se mostrará un ejemplo de cómo *Gemini* ofrece una respuesta en base a una pregunta realizada.

5. https://pub.dev/packages/vital_devices (Abril, 2024)
6. https://github.com/tryVital/vital-flutter (Abril, 2024)
7. https://pub.dev/packages/google_gemini (Febrero, 2024)
8. https://pub.dev/packages/google_gemini/example (Febrero, 2024)
9. https://ai.google.dev/ (Marzo , 2024)

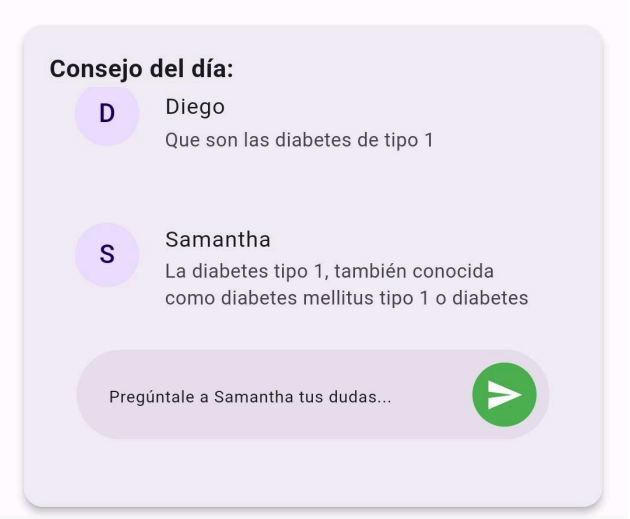


Ilustración 78: Generación de respuesta de *Gemini*

Para realizar esta implementación, hemos utilizado la guía que ofrecía la página oficial de *Flutter* [11] y su ejemplo [12] correspondiente.

En primer lugar, necesitaremos configurar la *API key* que nos permitirá acceder al asistente. Esta clave puede obtenerse en la página oficial de *Gemini* [13]. Es importante tener en cuenta que, actualmente, la región de España no está disponible. Por lo tanto, para poder generar la *API key* y utilizar el asistente, será necesario utilizar una *VPN* hasta que nuestra región esté disponible.

```
final gemini = GoogleGemini(apiKey: Claves.GeminiKey);
```

Ilustración 79: Crear instancia de *Gemini* (TarjetaConsejoVista)

En segundo lugar, tras declarar la clave, el usuario podrá formular sus dudas o cualquier consulta que quiera realizar. El código que se ve en la siguiente imagen se encarga de gestionarlo.

5. https://pub.dev/packages/vital_devices (Abril, 2024)
6. https://github.com/tryVital/vital-flutter (Abril, 2024)
7. https://pub.dev/packages/google_gemini (Febrero, 2024)
8. https://pub.dev/packages/google_gemini/example (Febrero, 2024)
9. https://ai.google.dev/ (Marzo , 2024)

```
final usuarioModel = Provider.of<UsuarioProvider>(context, listen: false);
setState(() {
  loading = true;
  textChat.add({
    "role": usuarioModel.obtenerUsuario?.nombre ?? "Usuario",
    "text": query,
  });
  _textController.clear();
});
scrollToTheEnd();
```

Ilustración 80: Envío de consulta por parte del usuario (Consejos)

Al igual que se debe tratar la parte del usuario, también se debe gestionar la generación de respuestas por parte de la inteligencia artificial. En la ilustración, se puede observar como llama al método "***generateFromText***", que pertenece a la biblioteca de *google_gemini*.

```
gemini.generateFromText(query).then((value) {
  setState(() {
    loading = false;
    textChat.add({"role": "Samantha", "text": value.text});
  });
  scrollToTheEnd();
}).onError((error, stackTrace) {
  setState(() {
    loading = false;
    textChat.add({"role": "Samantha", "text": error.toString()});
  });
  scrollToTheEnd();
});
```

Ilustración 81: Generación de respuesta por parte de Gemini (Consejos)

4.2.6.6.- HorizontalCalendar

```
HorizontalCalendar(
  date: seleccionaFecha,
  initialDate: DateTime(2000),
  textColor: Colors.black,
  backgroundColor: Colors.white,
  selectedColor: Colors.green,
  showMonth: true,
  locale: Localizations.localeOf(context),
  lastDate: DateTime.now(),
  onDateSelected: (date) {
    setState(() {
      seleccionaFecha = date;
      // Check if the selected date is the current date
      showChart = seleccionaFecha.year == DateTime.now().year &&
          seleccionaFecha.month == DateTime.now().month &&
          seleccionaFecha.day == DateTime.now().day;
    });
    seleccionaFechaModel.actualizarFecha(date);
  },
),  // HorizontalCalendar
```

Ilustración 82: Uso de la biblioteca HorizontalCalendar (Home)

Esta biblioteca nos permite visualizar un calendario horizontal a nuestra vista, pudiendo establecer la fecha inicial con la que empieza, la fecha actual, el color del calendario, etc…

5.- Pruebas

El testeo es una pieza fundamental en el desarrollo de proyectos, independientemente de su naturaleza. Como hemos aprendido en la asignatura de Desarrollo Ágil y como se ha discutido previamente, su importancia radica en garantizar la calidad y el éxito del producto final. Para ello, mostraremos algunos de los resultados obtenidos a la hora de aplicar nuestro plan de pruebas establecido en el apartado **3.5.- Plan de pruebas**

A la hora de realizar el testeo, y según se estableció anteriormente en el plan de pruebas, hemos dividido las pruebas en las siguientes secciones:

5.1.- Pruebas de aceptación

Entre las diferentes funcionalidades que han sido sometidas a éste tipo de pruebas, podemos destacar las siguientes:

- **Logueo del usuario**: hemos realizado pruebas exhaustivas del proceso de inicio de sesión del usuario en numerosas ocasiones, cada vez que los usuarios intentaban acceder introduciendo sus credenciales.
  En los casos de éxito, los usuarios proporcionaban las credenciales válidas que estaban registradas en nuestra base de datos, como por ejemplo, "corralfernandezdiego@gmail.com" con la contraseña "123456". Esto resultaba en un acceso exitoso a la pantalla principal de la aplicación, acompañado de la visualización de los datos correspondientes al usuario autenticado.
  Por otro lado, en los casos en los que se introducían datos incorrectos, como por ejemplo, "corralfernandezdiego@gmail.com" con la contraseña "654321", se desencadenaba la visualización de un mensaje de error indicando que las credenciales eran incorrectas.

- **Registro del usuario**: se realizaron 6 intentos de prueba para esta funcionalidad, con el objetivo de detectar posibles errores y garantizar su correcto funcionamiento.

  Durante estas pruebas, los usuarios tuvieron la oportunidad de explorar diversas formas de completar el formulario de registro de la aplicación.

  Entre las pruebas realizadas, se incluyeron casos como dejar campos en blanco y proporcionar tipos de datos incorrectos, como cadenas de texto en campos que solo aceptaban valores numéricos.

  También se evaluó qué ocurría si se dejaba el campo del correo electrónico en blanco, observando las respuestas del sistema ante los valores por defecto que aparecían.

En los casos en los que el formulario se completaba de forma correcta, el inicio de sesión se llevaba a cabo automáticamente, lo que permitía el acceso a la pantalla principal con los datos correspondientes del usuario.

Por otro lado, si se detectaban errores en los datos introducidos en el formulario, se mostraba un mensaje de error indicando que había datos incorrectos, y se especificaron los campos que requerían corrección.

- **Registro de alimentos**: durante las pruebas para verificar el funcionamiento relacionado con los alimentos, los usuarios realizaron diversas interacciones con la aplicación. Se evaluaron escenarios donde intentaban agregar alimentos que estaban presentes en la base de datos de *USDA Food*, así como aquellos que no lo estaban.

  Cuando un alimento no estaba en la base de datos, los usuarios no podían acceder a la vista que mostraba la información de dicho alimento. Por ejemplo, al intentar agregar "Lubina a la plancha", la aplicación no permitía el acceso a la vista detallada del alimento.

  Por otro lado, si el alimento estaba en la base de datos, los usuarios podían acceder a la vista correspondiente y agregarlo. Una funcionalidad adicional permitía al usuario especificar la cantidad de unidades consumidas, lo que afectaba los valores nutricionales mostrados. Por ejemplo, si el usuario indicaba que tomaba 2 unidades de pizza, los valores nutricionales se multiplicaban por dos.

  Una vez registrado el alimento, el usuario era redirigido a la vista que mostraba el listado de alimentos que había registrado. Si el registro se realizaba correctamente, se mostraba una tarjeta con el alimento correspondiente en esta vista. En caso contrario, si había algún error en el registro, no se mostraba ningún elemento.

- **Registro de actividades**: al igual que con los alimentos, los usuarios tenían la posibilidad de registrar actividades que estuvieran previamente listadas en la base de datos de *Wger*, así como otras actividades que no estuvieran incluidas.

  Cuando seleccionaban una actividad existente, podían registrarla y verificar que la fórmula para calcular las calorías quemadas según la duración e intensidad de la actividad estuviera correctamente ajustada.

  Una vez completado el registro del ejercicio, el usuario era redirigido automáticamente a una vista que mostraba un listado detallado de todas las actividades que había realizado.

Por otro lado, si la actividad no estaba disponible en la base de datos, los usuarios no tenían la posibilidad de avanzar hacia la vista correspondiente para almacenarla.

- **Configuración de recordatorios y medicamentos**: para comprobar las notificaciones, los usuarios han realizado diferentes pruebas creando varios recordatorios.

  Para ello, se ha revisado que los permisos para poder utilizar las notificaciones estuviera activado.

  En caso de que los usuarios crearan las citas correctamente, la aplicación desplegaba el mensaje de notificación correspondiente y el dispositivo vibraba según la configuración. Por otro lado, si hubiera algún problema en la creación de la cita, la aplicación no emitía ninguna notificación ni vibración.

- **Monitorización de glucosa**: se ha estado perfeccionando en cada interacción con los usuarios, si la visibilidad de los datos de las diferentes lecturas y la interfaz para establecer la conexión con los glucómetros era realmente fácil e intuitiva.

- **Edición de datos**: Durante estas pruebas, se verificó que los usuarios tuvieran acceso adecuado a las funciones de edición y que pudieran modificar los datos según fuera necesario.

  En caso de que la edición de datos se realizará correctamente, se confirmó que los cambios se reflejaran correctamente en la aplicación y que los usuarios recibirán confirmación visual de la actualización.

  Por el contrario, si se presentaban problemas durante la edición, se garantizó que la aplicación manejara la situación de manera adecuada, proporcionando mensajes de error informativos y permitiendo al usuario corregir los datos.

  A continuación se mostrará un resumen de las diferentes pruebas realizadas, donde se indicará el número de veces que se han realizado y sus correspondientes resultados obtenidos.

Nombre de las pruebas	***Repeticiones***	***Resultados***
Logueo del usuario	10	- Se ha estado mejorando la interfaz del inicio de sesión de usuarios para hacerla cada vez más intuitiva y sencilla. - Se han corregido los errores encontrados relacionados con las funcionalidades de cerrar sesión y recuperar la contraseña. - Se ha verificado que el usuario solo puede acceder a la aplicación introduciendo sus credenciales. - Se han mejorado los avisos cuando las credenciales son incorrectas. - Se ha logrado implementar dos métodos de inicio de sesión.
Registro	5	- Se ha estado mejorando la interfaz del formulario de registro para hacerla cada vez más intuitiva y sencilla. - Se han ido agregando los campos que los sujetos de prueba nos indican en cada reunión, ya que eran importantes para el registro de usuarios. - Se han mejorado las funcionalidades para cambiar la foto de perfil y la fecha de nacimiento. - Se ha podido perfeccionar para que dure menos de 5 minutos.
Registro de alimentos	15	- La interfaz ha sido mejorada para que los usuarios puedan registrar sus comidas de forma sencilla, haciéndola cada vez más intuitiva. - Se ha corregido el cálculo de las calorías totales consumidas para asegurar la precisión de la información nutricional proporcionada. - Se ha podido perfeccionar la visualización por fecha, lo que facilita a los usuarios hacer un seguimiento más efectivo de su ingesta diaria. - Se ha mejorado la visualización de los datos.

Tabla 6: Resumen de las pruebas de aceptación (I)

Registro de actividades	10	- Se ha estado mejorando la interfaz para que los usuarios puedan registrar sus actividades de forma sencilla, haciéndola cada vez más intuitiva. - Se ha corregido el cálculo de calorías totales quemadas durante la sesión. - Se ha podido perfeccionar la visualización por fecha, lo que facilita a los usuarios hacer un seguimiento más efectivo de su actividad. - Se ha mejorado la visualización de los datos.
Configuración de recordatorios y medicamentos	5	- Se ha estado mejorando la interfaz para hacerla cada vez más intuitiva y sencilla. - Se ha podido encontrar el fallo que causaba que las notificaciones no se activaran. - La visibilidad de los avisos se ha ido mejorando para que sean lo más llamativos posibles.
Lecturas de glucosa	10	- Se ha estado mejorando la interfaz para que los usuarios puedan registrar sus dispositivos y puedan realizar sus mediciones de glucosa de forma sencilla, haciéndola cada vez más intuitiva. - Se ha podido mejorar la visualización de los datos obtenidos de las lecturas para que sea más fácil comprenderlas.
Edición de datos	10	- Se ha estado mejorando la interfaz para hacerla cada vez más intuitiva y sencilla. - Se han mejorado las funcionalidades para cambiar la foto de perfil y la fecha de nacimiento. - Se ha perfeccionado la función de notificación al usuario en caso de que su contraseña se cambie.

Tabla 7: Resumen de las pruebas de aceptación (II)

5.2.- Pruebas de caja negra

Como se discutió en la asignatura de "Calidad del Software", las pruebas de caja negra están diseñadas para evaluar todos los requisitos funcionales del *software*.

Para llevar a cabo las pruebas, hemos agrupado la aplicación en diferentes clases de equivalencia. Estás agrupan conjuntos de estados válidos o inválidos basados en ciertas condiciones de entrada, donde le hemos aplicado la estrategia de prueba del array ortogonal (OATS) que aprendimos en la asignatura mencionada anteriormente.

Estas pruebas nos permiten detectar funcionalidades incorrectas o ausentes, errores en la interfaz, problemas en las estructuras de datos o en el acceso a bases de datos externas, así como errores de comportamiento.

A continuación, se presentan ilustraciones que muestran cómo la aplicación responde correctamente cuando se ingresan condiciones de entrada válidas.

- **Monitorización de glucosa**

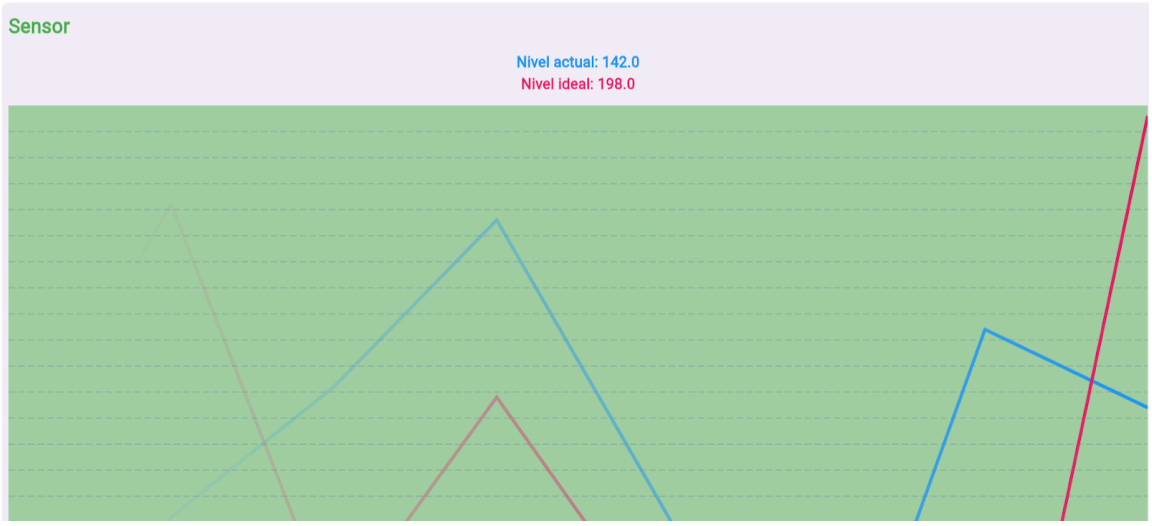


Ilustración 83: Lectura de glucosa en la aplicación

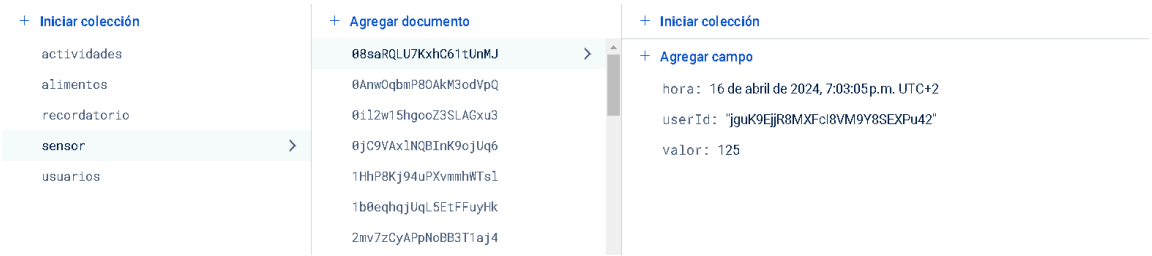


Ilustración 84: Almacenamiento de lecturas en *Firebase*

- **Calorías consumidas**

Ilustración 85: Registro de alimentos en la aplicación

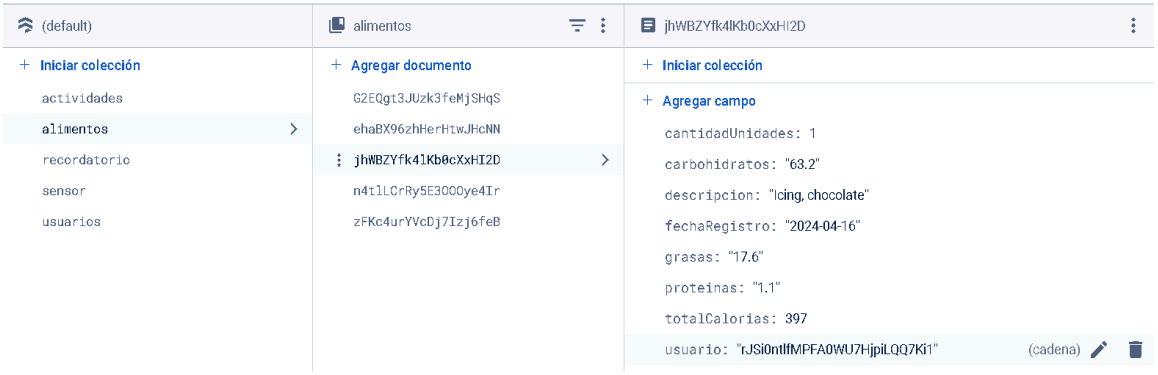


Ilustración 86: Almacenamiento de alimentos en la *Firebase*

- **Calorías quemadas**

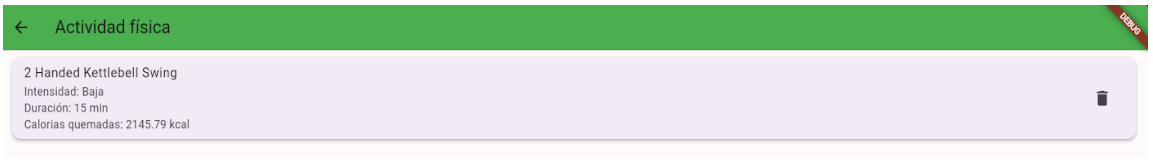


Ilustración 87: Registro de actividades en la aplicación

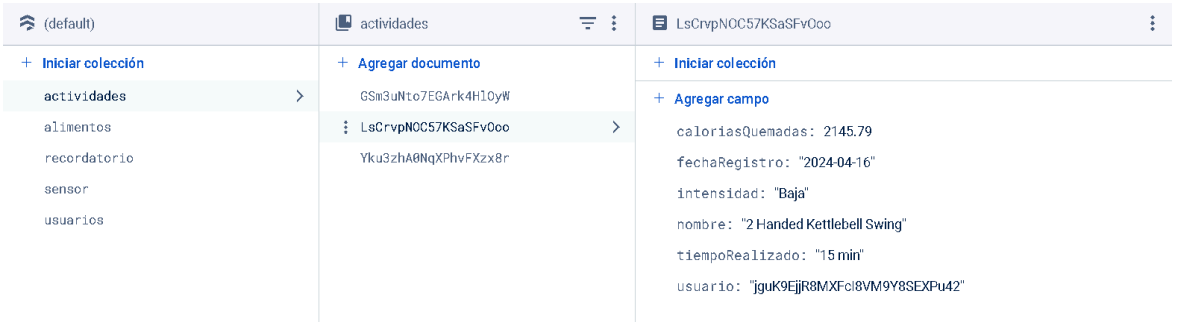


Ilustración 88: Almacenamiento de actividades en la *Firebase*

- **Logueo de usuario**

Ilustración 89: Registro de los correos en *Firebase Authentication*

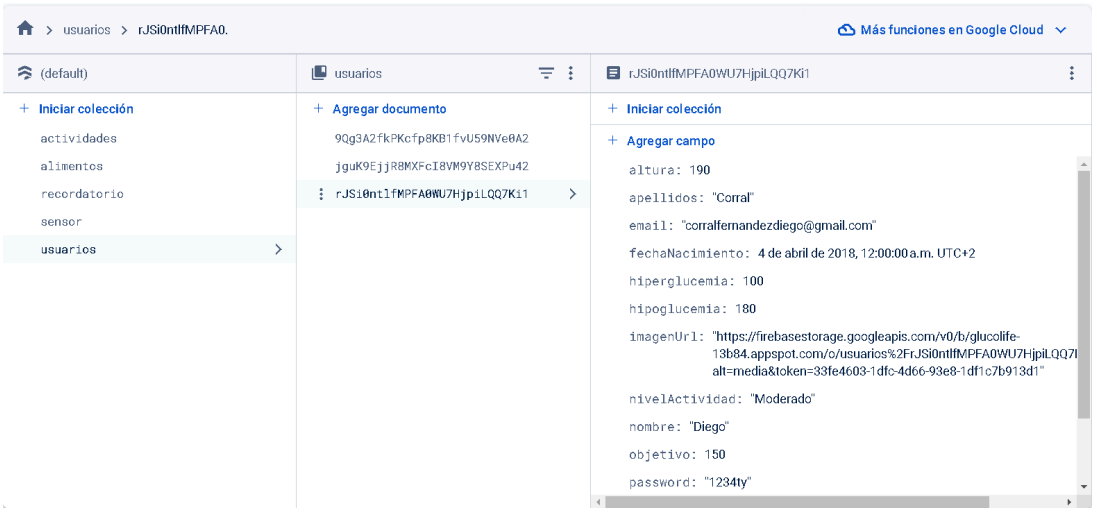


Ilustración 90: Almacenamiento de los usuarios en *Firebase Database*

A continuación, se presentará un tabla resumen donde se mostrarán los resultados obtenidos durantes las pruebas.

Nombre de las pruebas	Repeticiones	Resultados
Monitorización de glucosa	5	- Se han corregido los diferentes errores relacionados con el almacenamiento de los valores en *Firebase*.
Calorías consumidas	10	- Se ha podido corregir aquellos fallos que se han encontrado relacionados con el almacenamientos en la base de datos. - Se ha ido perfeccionando la obtención de datos de *USDA Food.*
Calorías quemadas	15	- Se ha podido corregir aquellos fallos que se han encontrado relacionados con el almacenamientos en la base de datos. - Se ha ido perfeccionando la obtención de datos de *Wger.*

Tabla 8: Resumen de las pruebas de caja negra (I)

Nombre de las pruebas	Repeticiones	Resultados
Registro de usuarios	20	- Se ha podido comprobar que los usuarios se almacenaban tanto en la base de datos como en *Firebase Authentication*. - Se ha podido corregir el error de que los datos no se actualizaban cuando eran modificados en la aplicación.

Tabla 9: Resumen de las pruebas de caja negra (II)

5.3.- Pruebas de compatibilidad

Gracias a estas pruebas, hemos podido verificar la completa funcionalidad de la aplicación en una variedad de dispositivos.

En cada iteración, se les ofrecía a los usuarios un archivo *APK,* el cuál es obtenido mediante la explicación que se da en el apéndice de **Instalación**. Al principio, durante la instalación les ofrecemos apoyo para resolver aquellos problemas que pudieran tener. Finalmente, durante las últimas iteraciones los usuarios realizaban la instalación en sus dispositivos por sí solos.

Inicialmente, a la hora de realizar dichas pruebas las realizamos nosotros para enseñarles de forma generar los diferentes aspectos de la aplicación. Pero una vez que los usuarios empezaron a coger experiencia con la aplicación les dejamos varios días e incluso semanas para que probaran el funcionamiento de la aplicación en sus dispositivos.

Tabla resumen		
Dispositivos	**Versión del S.O**	**Conclusiones**
Xiaomi X3 PRO	MIUI Global 13.0.2	- Instalación rápida y sencilla.
Xiaomi Redmi 9	MIUI Global 13.0.2	- Comprobación del comportamiento de la aplicación en dispositivos con pantallas grandes. - Rendimiento irregular.

Tabla 10: Dispositivos utilizados (I)

Tabla resumen		
Dispositivos	**Versión del S.O**	**Conclusiones**
Oppo A18	ColorOS V12	- Fantástico rendimiento, fue el mejor dispositivo donde la aplicación funcionó.
Xiaomi X4 PRO 5G	MIUI 14.0.7	- Instalación complicada por el sistema de seguridad incorporado.

Tabla 11: Dispositivos utilizados (II)

6.- Conclusiones

El presente Trabajo de Fin de Grado ha consistido en el análisis, diseño y desarrollo de un prototipo de aplicación asistente para personas que padecen diabetes. La principal motivación detrás de este proyecto fue la intención de brindar apoyo a mi primo, quien fue mi inspiración para emprender este proyecto. Además, me propuse desarrollar una aplicación completa, ya que observé que las opciones disponibles en el mercado no satisfacían por completo las necesidades de las personas con esta condición.

Como se detalló en el apartado **1.2.- Objetivos** y en documento de la propuesta, considero que dichos puntos se han cumplido satisfactoriamente por los siguientes motivos:

- Se ha realizado un estudio sobre las necesidades que padece una persona con diabetes mediante reuniones. Dichas necesidades se han conseguido cubrir mediante las siguientes funcionalidades:
    - **Registro y logueo de usuarios**: hemos desarrollado un sistema de autenticación robusto utilizando el *framework Firebase*. Nuestro formulario de registro recopila todos los datos necesarios para los usuarios con diabetes, proporcionando una experiencia satisfactoria y segura.

    - **Registro de alimentos y actividades**: integramos dos bases de datos confiables, *Wger* y *USDA Food*, para ofrecer información relevante sobre alimentos. Además de esto, hemos agregado funciones adicionales, como el cálculo de calorías consumidas y quemadas, enriqueciendo completamente estas secciones.

    - **Registro de lecturas de glucosa**: utilizando el paquete *Flutter vital_devices*, hemos logrado conectar nuestra aplicación con varios dispositivos para realizar lecturas de glucosa de manera efectiva. Esta integración garantiza un alto nivel de satisfacción al cumplir con este objetivo crucial.

    - **Edición de datos**: en la sección de ajustes, hemos implementado opciones que permiten a los usuarios modificar sus datos de manera intuitiva. Toda la información editada se registra tanto en la aplicación como en la base de datos, garantizando la coherencia de los datos.
    - **Registro de recordatorios**: hemos cumplido con el objetivo de establecer recordatorios implementando una función que permite a los usuarios configurar notificaciones en momentos específicos. Estas notificaciones incluyen datos importantes relacionados con el recordatorio, mejorando la gestión de la diabetes.

    - **Registro de medicamentos**: desarrollamos un formulario simple e intuitivo que permite a los usuarios registrar información relevante sobre sus medicamentos, incluidos los períodos de toma. Esta funcionalidad se integra perfectamente en la aplicación, facilitando el seguimiento de la medicación.

- Se ha elaborado un prototipo de aplicación web, en la que se ha ido utilizando la arquitectura MVVM (*Model - View - View Model)*, principios de diseño y *Kanban* como metodología de trabajo. Además se ha utilizado *Flutter* como entorno de desarrollo, él cual nos ha permitido realizar el prototipado de forma sencilla y rápida.

El uso de la metodología ágil *Kanban* ha sido fundamental en la gestión del proyecto, permitiéndonos dividirlo en diferentes fases funcionales que se entregaban al cliente de manera incremental. Esto facilitó el desarrollo del proyecto al permitir una mayor flexibilidad y adaptabilidad a medida que avanzábamos.

Sin embargo, a lo largo del desarrollo, he encontrado algunos inconvenientes que podrían resolverse con otras metodologías ágiles como *SCRUM*. Por ejemplo, al no establecer una duración predeterminada para cada fase, algunas podrían durar más que otras, lo que a veces resultaba molesto para el cliente. Con *SCRUM*, la implementación de *sprints* con una duración fija podría ayudar a mantener un ritmo de entrega más constante y predecible, lo que podría ser beneficioso en este sentido.

En cuanto a la elección de la tecnología, *Flutter* demostró ser una decisión acertada para el desarrollo del proyecto. Su versatilidad nos permite generar ejecutables para las tres principales plataformas: escritorio, dispositivos móviles y web. Esta flexibilidad abre la posibilidad de considerar la publicación de la solución en múltiples plataformas, dependiendo de las decisiones futuras sobre el desarrollo del proyecto.

Durante el desarrollo de la aplicación, nos enfrentamos a varios desafíos, siendo la implementación de la funcionalidad para realizar lecturas de los niveles de glucosa el aspecto más relevante. Esta tarea se logró gracias al paquete *Flutter vital_devices*, el cual nos permitió vincular nuestra aplicación con una variedad de dispositivos de monitorización de la salud, como los sensores de glucosa. Consideramos que este paquete es indispensable para cualquier proyecto que tenga como finalidad el monitoreo de la salud.

Para concluir esta sección, es importante destacar que el desarrollo de este Trabajo Fin de Grado (TFG) ha sido un recorrido a lo largo de los cuatro años del grado. Desde los conceptos básicos de programación, elaboración de diagramas y la gestión de las bases de datos hasta los aspectos más específicos de mi área, como

la aplicación de metodologías ágiles, la gestión de proyectos de *software* y conocimientos necesarios para gestionar las aplicaciones. Estos conocimientos, han sido obtenidos gracias a asignaturas como Programación Orientada a Objetos, Fundamentos de Bases de Datos, Fundamentos del Software, Desarrollo Ágil, Calidad del Software, Gestión de Proyectos de Software, Ingeniería del Software, Desarrollo de Aplicaciones Web y Desarrollo de Aplicaciones Empresariales, fueron fundamentales para el éxito de este proyecto.

6.1.- Mejoras y trabajos futuros

A partir de lo desarrollado, se abre un gran abanico de opciones, que pasan por la expansión de prestaciones de la aplicación, ya sea con ideas descartadas o con nuevas. La aplicación siempre se ha ido desarrollando con la idea de que en un futuro se pueda ir expandiendo y llegar a publicarla, bien como aplicación de pago o como solución gratuita, o aplicando un modelo mixto en el que el código fuente esté alojado en un repositorio de código online y la aplicación final se consiga, previo pago, en las tiendas de aplicaciones. Esta cuestión merecerá una mayor instrucción en materia de licencias de software por parte del autor del presente Trabajo.

Algunos de los aspectos que tenemos en mente para un futuro son los siguientes:

Mejora	Motivo
Aplicar machine learning	Para poder obtener con mayor exactitud futuros niveles de glucosa.
Personalización de interfaz	Darle mayor libertad al usuario para la personalización de la aplicación.
Mejorar el asistente	Implementar una versión mejor que nos permita obtener imágenes, mejores respuestas, etc.
Implementación de emergencias	Esta historia de usuario **(ID: 6)** finalmente no ha podido ser implementada debido a la falta de tiempo. Se ha priorizado perfeccionar otras funcionalidades que poseen una mayor importancia para garantizar un funcionamiento óptimo del sistema.
Aplicar *Test-Driven-Development*	Sería de gran ayuda para proseguir con el desarrollo de la aplicación en un futuro y conseguir que obtenga buenos resultados ante la competencia del mercado.

Tabla 12: Mejoras (I)

Mejorar la conexión con glucómetros	Aunque se ha dejado los métodos implementados para poder vincular la aplicación *Vital*, en próximas versiones se podría mejorar dicha funcionalidad, ya sea mejorando la conectividad con dicha *API* o buscando otra solución más efectiva.
Implementación del cálculo de unidades de insulina	En las etapas finales del desarrollo, un usuario de prueba sugirió aprovechar la inteligencia artificial para implementar el cálculo de unidades de insulina necesarias. Esta característica podría ser una ventaja competitiva, por lo que será desarrollada en futuras versiones.

Tabla 13: Mejoras (II)

7.- Bibliografía

[1] Wikipedia (2023). Metodología. [Online] Disponible en: https://es.wikipedia.org/wiki/Metodolog%C3%ADa. Acceso: Recuperado en Dic 13, 2023

[2] Universitat Carlemany (2021). Metodologías de desarrollo de software. [Online] Disponible en: https://www.universitatcarlemany.com/actualidad/blog/metodologias-de-desarrollo-de-software/. Acceso: Recuperado en Dic 13, 2023

[3] Liñan Salinas, E. (2023). Metodologías Tradicionales vs Ágiles. [Online] Disponible en:https://www.linkedin.com/pulse/metodolog%C3%ADas-tradicionales-vs-%C3%A1giles-efrain-li%C3%B1an-salinas/?originalSubdomain=es. Acceso: Recuperado en Nov. 21, 2023.

[4] The braintrust consulting group. (2023). The State of Agile. [Online] Disponible en: https://www.braintrustgroup.com/wp-content/uploads/2023/06/Braintrust-2023-State-of-Agile-Report.pdf. Acceso: Recuperado en Dic 13, 2023.

[5] Gear People (2021). Mediciones de glucosa sin pinchazos. [Online] Disponible en: https://www.solucionesparaladiabetes.com/magazine-diabetes/medidores-de-glucosa-sin-pinchazos/. Acceso: Recuperado en Dic 4, 2023.

[6] Openinnova App Nativa vs Web App. Cual es la mejor elección?. [Online] Disponible en: https://www.openinnova.es/app-nativa-vs-web-app-la-mejor-eleccion. Acceso: Recuperado en Nov. 21, 2023

[7] Mati Presta. React Native: Ventajas y desventajas. [Online] Disponible en: https://blog.back4app.com/es/react-native-ventajas-y-desventajas-reveladas/#Diez_ventajas_principales_de_React_Native. Acceso: Recuperado en Nov. 22, 2023

[8] Grupo EBIM (2023). Conoce las ventajas y desventajas del desarrollo de aplicaciones con *Flutter*. [Online] Disponible en: https://www.grupoebim.com/blog/ventajas-desventajas-flutter/. Acceso: Recuperado en Nov. 22, 2023.

[9] Marc Bolufer Gil (2023). Análisis de las ventajas y desventajas de Angular: Una mirada en profundidad. [Online] Disponible en: https://ventajasydesventajastop.com/angular-ventajas-y-desventajas/. Acceso: Recuperado en Nov 22. 2023

[10] Alba Naveira (18 julio 2023). Ventajas y desventajas de Swift [Online] Disponible en: https://www.campustraining.es/noticias/ventajas-desventajas-swift/. Acceso: Recuperado en Nov 22, 2023

[11] Roberto Aguilar (21 enero 2022). ¿Cuales son las ventajas y desventajas de usar Firebase para una base de datos? [Online] Disponible en: https://www.linkedin.com/pulse/cu%C3%A1les-son-las-ventajas-y-desventajas-de-usar-para-aguilar-bernabe/?originalSubdomain=es. Acceso: Recuperado en Nov 22, 2023

[12] Ahmed Bahgat (25 agosto 2023). Flask vs Django: Elijamos tu proximo framework python. [Online] Disponible en: https://kinsta.com/es/blog/flask-vs-django/. Acceso: Recuperado en Nov 22, 2023

[13] Anastasia Kushnir (4 julio 2021). Pros y contras del uso de Spring Boot. [Online] Disponible en: https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot/. Acceso: Recuperado en Nov 22, 2023

[14] Wincaptor Método MoSCoW. [Online]
Disponible en: https://www.wincaptor.com/metodo-moscow.html
Acceso: Recuperado en Mar 25, 2024

[15] Glassdor (2023). Sueldos para el puesto de Ingeniero de software en España. [Online] Disponible en: https://www.glassdoor.es/Sueldos/ingeniero-de-software-sueldo-SRCH_KO0,21.htm. Acceso: Recuperado en Dic 13, 2023

[16] Universidad de Salamanca (2018). Ingeniería de software I: Modelo de dominio [Online] Disponible en : https://repositorio.grial.eu/bitstream/grial/1153/1/8.%20Modelo%20de%20dominio.pdf Acceso: Recuperado en Mar 26, 2024

[17] Lucidchart (2024) ¿Qué es un modelo entidad relación? [Online] Disponible en: https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion
Acceso: Recuperado en Abr 20, 2024

[18] Wikipedia (2024) Modelo–vista–modelo de vista [Online] Disponible en: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93modelo_de_vista Acceso: Recuperado en Mar 26, 2024

[19] L. Alfonso Ureña López y Ángel Aguilera García. “Modelación de la Dinámica”, Fundamentos de Ingeniería del Software, Jaén, 2019, Tema 6

[20] YESWELAB. Visily - Herramienta de wireframe impulsada por IA para crear app wireframes [Online] Disponible en: https://yeswelab.com/blogs/aplicaciones-de-la-inteligencia-artificial/visily-herramienta-wireframe-impulsada-por-ia Acceso: Recuperado en Mar 25, 2024

[21] Bailey, Thomas, Biessek, Alessandro. *Flutter for beginners : an introductory guide to building cross-platform mobile applications with flutter 2. 5 and dart.* Packt Publishing, Limited, 2021 [Online]. Disponible en Recursos Electrónicos de la Universidad de Jaén

Apéndice I. Descripción de contenidos suministrados

El material complementario será proporcionado mediante un archivo .zip, donde se encontrará la siguiente información:

Archivo .ZIP	
Capturas	Contiene todas imágenes correspondientes al prototipado.
Diagramas	Dentro de esta carpeta se encuentran una serie de subcarpetas, que se corresponden a los diferentes diagramas comentados.
Video explicativo	Ejemplo de funcionamiento de la aplicación en tiempo real.
Código fuente	Contiene el código fuente de la aplicación desarrollada.
APK	Ejecutable que podrá ser instalado en cualquier dispositivo *Android 13*.
Wireframe	Incluirá las diferentes imágenes del boceto que se realizó en el estudio.
Documentación	Dentro de esta carpeta se encuentran los diferentes archivos que componen la documentación del proyecto.
Excel	Contiene el archivo excel donde se han realizado los diagramas de *Gantt*.

Tabla 14: Contenidos de la carpeta .zip

Apéndice II. Manual de Instalación del sistema

El código fuente de la aplicación se encuentra en el siguiente repositorio de *GitHub* [14]

Si contamos con un sistema donde se encuentre *Flutter* instalado y configurado bastará con clonar el repositorio y compilar dicho proyecto mediante el comando "**flutter build apk - -release**".
Dado que nuestro proyecto está principalmente orientado a dispositivos móviles *Android*, utilizamos la opción "apk" en el comando de compilación. Sin embargo, si en el futuro deseamos ejecutar la aplicación en otros dispositivos, debemos considerar las siguientes opciones:

- ***"appbundle"***: construye una aplicación de *Android* en formato *AppBundle*, una alternativa al *APK* más reciente.
- ***"ios"***: construye los archivos necesarios para ejecutar la versión de *iOS* (solo disponible en *Flutter* para *macOS*).
- ***"ipa"***: construye la versión de *iOS* preparada para su distribución.
- ***"macos"***: construye la versión de *macOS*.
- "***web***": genera los archivos necesarios para la versión web (que será ejecutada por nuestro navegador predefinido).
- ***"windows"***: construye archivos ejecutables para *Windows*.
- ***"linux"***: construye binarios para sistemas *GNU/Linux*

Si no dispones de un dispositivo con *Flutter* instalado, puedes consultar la página oficial de *Flutter* [15] para obtener instrucciones sobre cómo configurarlo.

En la siguiente ilustración, se muestra un ejemplo de una ejecución de dicho comando.

```
PS C:\Users\corra\Documents\GitHub\glucolife\glucolife_app> flutter build apk --release

Running Gradle task 'assembleRelease'...                          42,5s
√  Built build\app\outputs\flutter-apk\app-release.apk (52.4MB).
```

Ilustración 91: Generación del APK

14. https://github.com/diiegoo26/glucolife (Mayo, 2024)
15. https://docs.flutter.dev/get-started/install (Mayo, 2024)

Apéndice III. Manual de Usuario

Tras la instalación, al usuario se le mostrará la pantalla de bienvenida de la aplicación, donde podrá loguearse en caso de que cuente con una cuenta o realizar el formulario de registro en caso contrario.

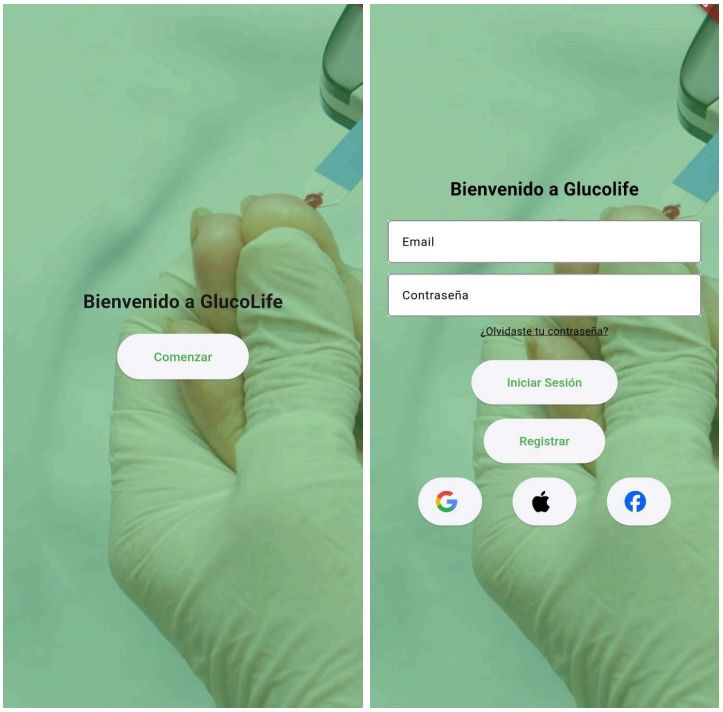


Ilustración 92: Inicio de aplicación

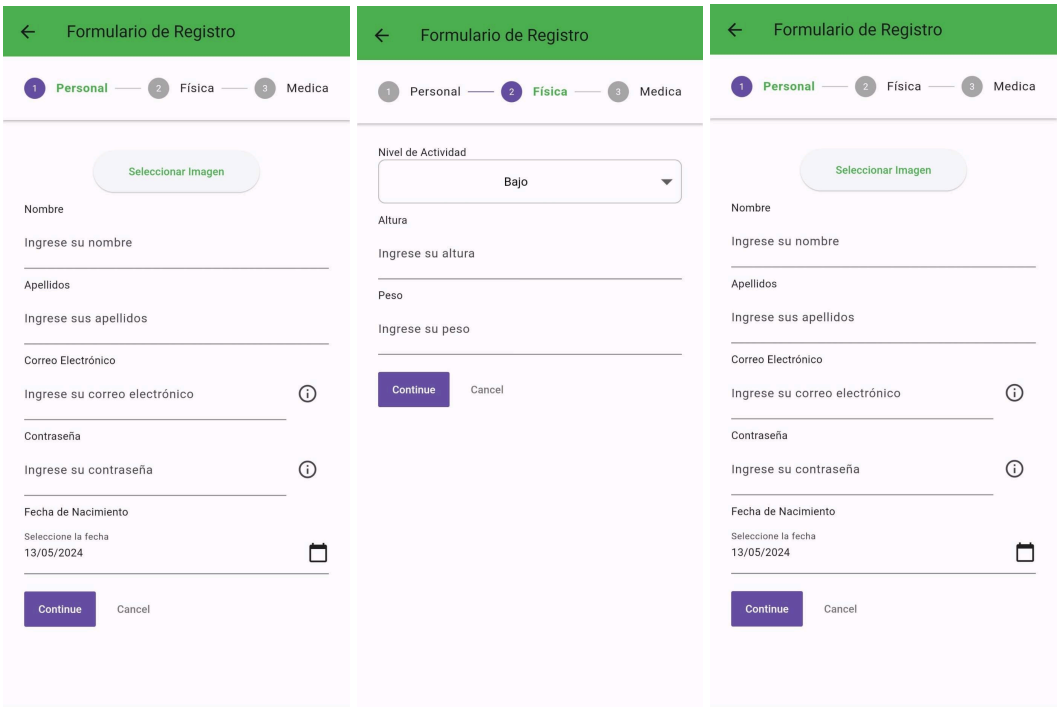


Ilustración 93: Formulario de registro

En la vista de logueo, los usuarios podrán disponer de la funcionalidad para recuperar la contraseña en caso de olvido, así como de iniciar sesión mediante diversos métodos.

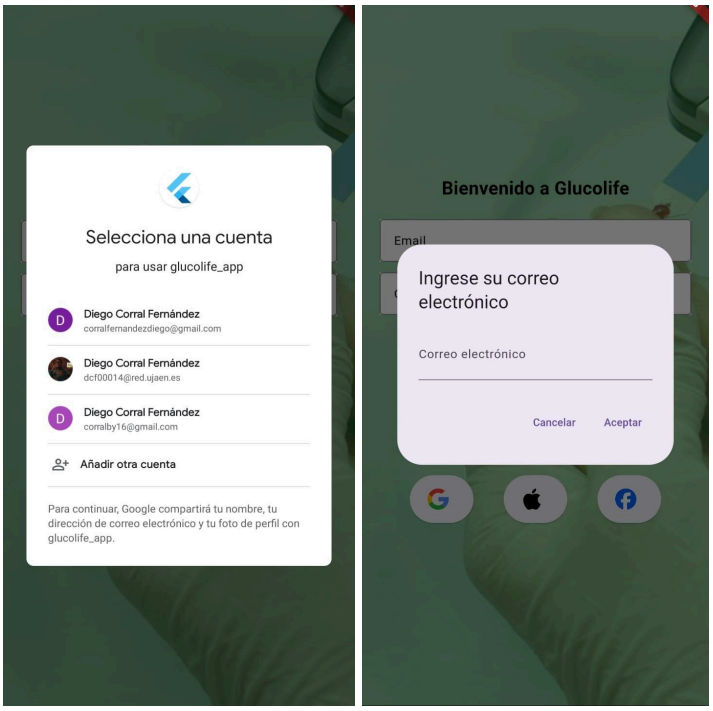


Ilustración 94: Funciones en la pantalla de logueo

Después de completar el proceso de logueo o registro, los usuarios serán dirigidos a la pantalla *Home*. Aquí, podrán ver un resumen de su monitorización, incluyendo las calorías quemadas y consumidas. Además, tendrán acceso a diversas funcionalidades a través de los distintos accesos disponibles.

Adicionalmente a lo mencionado anteriormente, los usuarios tendrán la capacidad de seleccionar, mediante un calendario, la visualización de sus datos diarios. También podrán interactuar con la asistente virtual "*Samantha*".

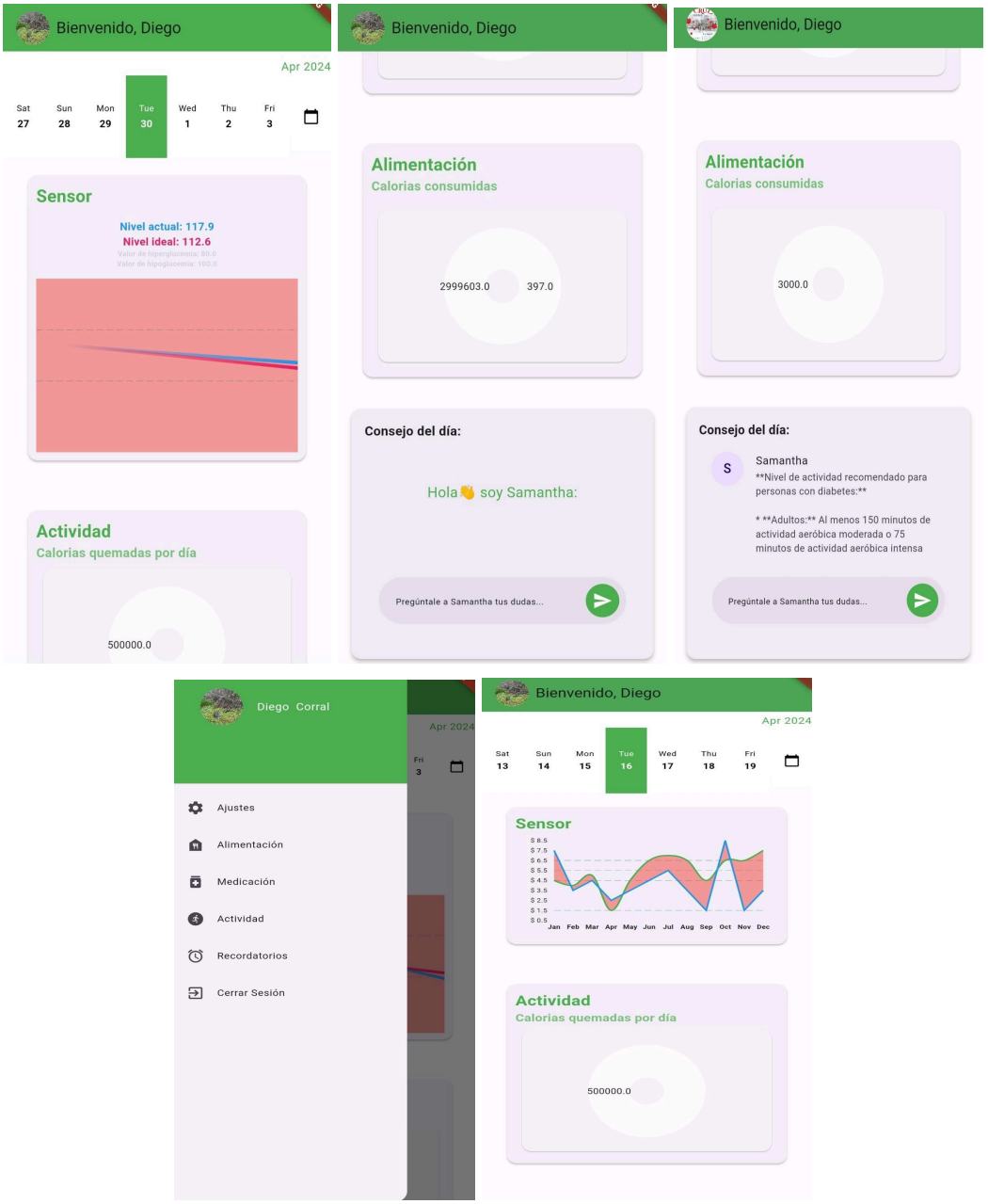


Ilustración 95: Pantalla Home

Dentro de la sección de ajustes, los usuarios encontrarán opciones para modificar tanto su información personal como la médica. También tendrán acceso a funciones para cambiar la contraseña y vincular dispositivos.

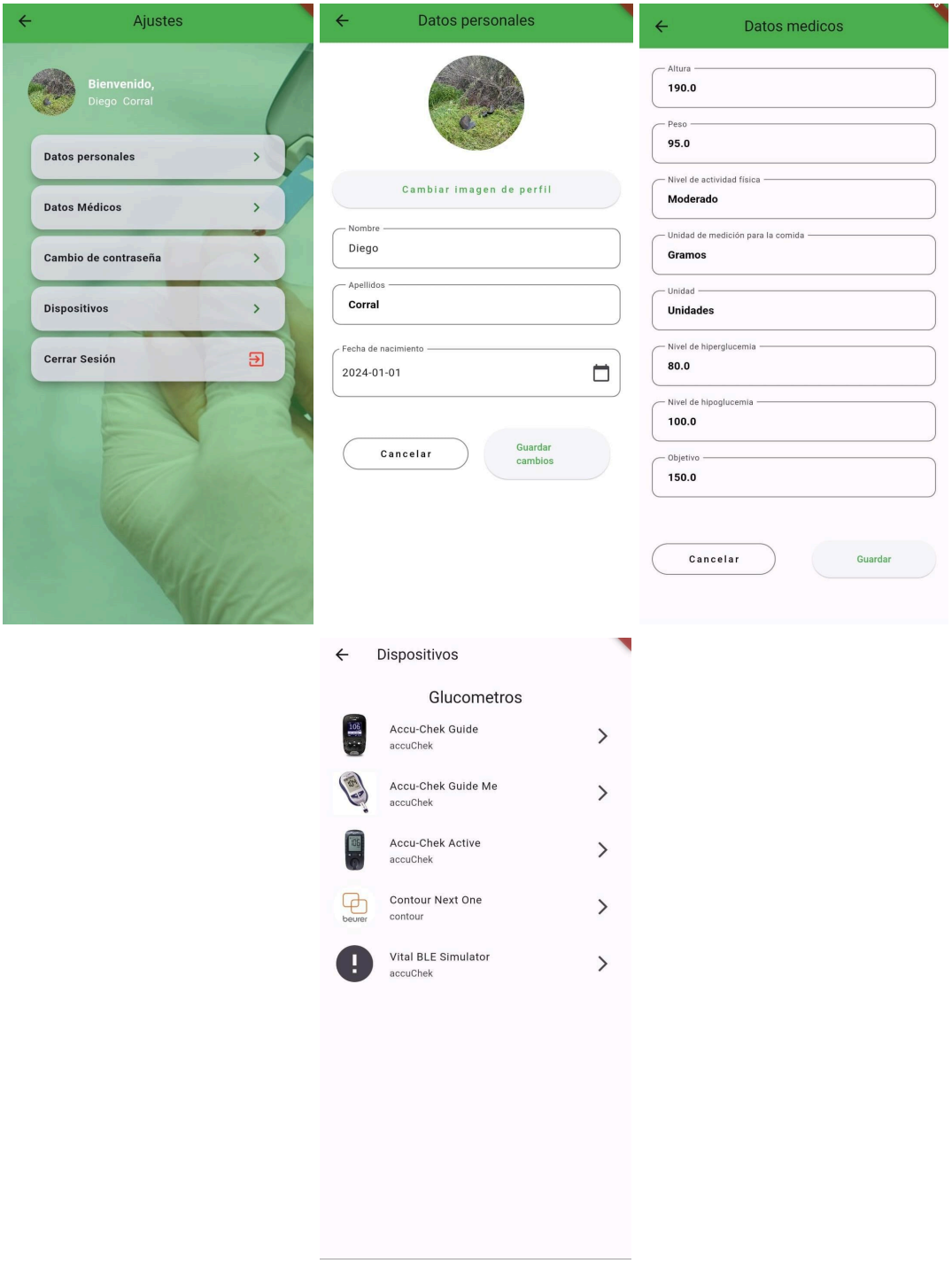


Ilustración 96: Sección de ajustes

En la sección de alimentación, los usuarios encontrarán inicialmente una vista que muestra los alimentos consumidos en el día, donde tendrán la opción de eliminarlos. También tendrán un botón que les dará acceso al buscador para registrar alimentos adicionales. Después de realizar la búsqueda, podrán ver los datos nutricionales de los alimentos encontrados, y tendrán la opción de agregar o eliminar unidades según sea necesario.

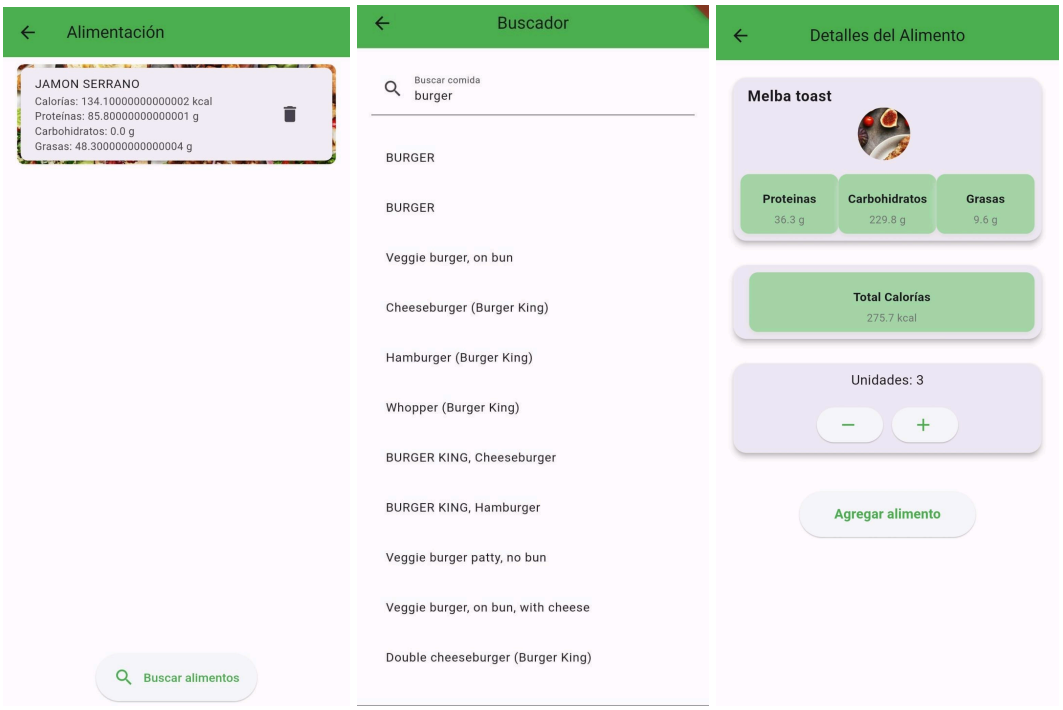


Ilustración 97: Sección de alimentación

En la sección de actividades físicas, los usuarios encontrarán inicialmente una vista que muestra las actividades en el día, donde tendrán la opción de eliminarlas. También tendrán un botón que les dará acceso al buscador para registrar actividades adicionales. Después de realizar la búsqueda, podrán modificar los parámetros del entrenamiento y así obtener el total de calorías consumidas.

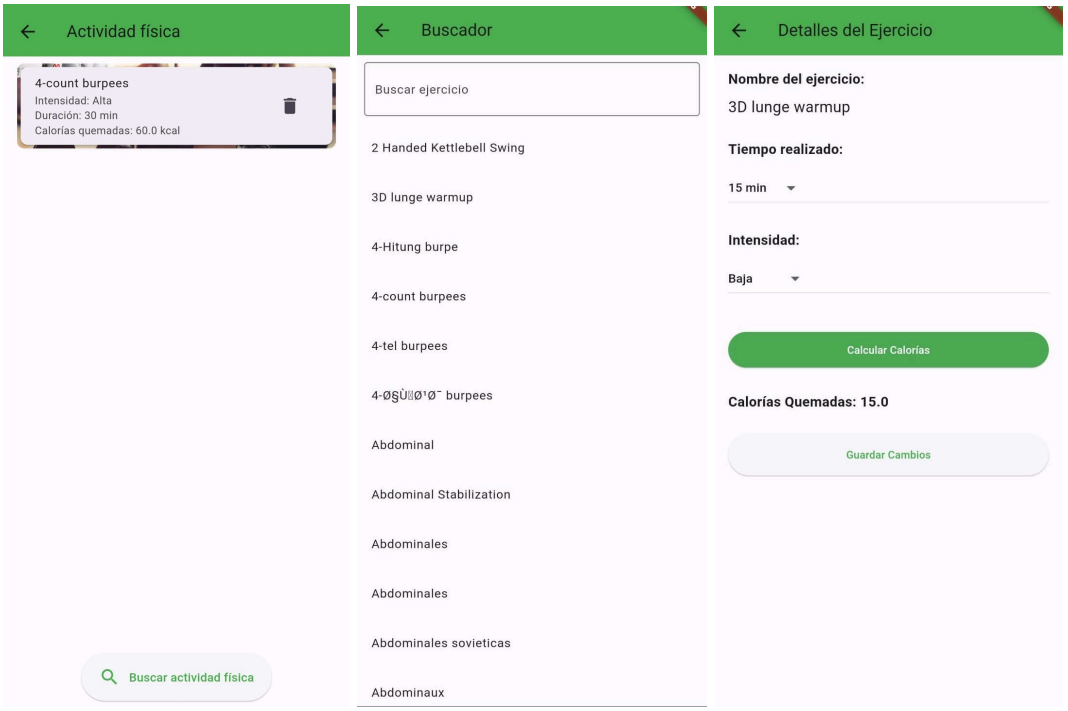


Ilustración 98: Sección de actividad física

Dentro de la sección de medicamentos, los usuarios serán recibidos por una vista que enumera los medicamentos que deben tomar, con la opción de eliminarlos si es necesario. Además, encontrarán un botón para registrar nuevos medicamentos, donde podrán ingresar el número de dosis y la frecuencia de administración requerida.

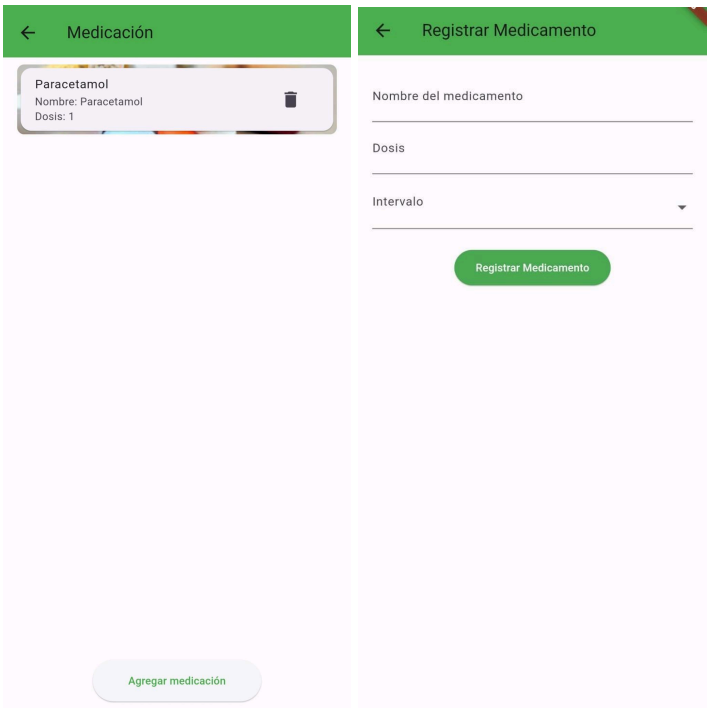


Ilustración 99: Sección de medicamentos

En la sección de recordatorios, los usuarios serán recibidos por una vista que enumera aquellas citas que no quiere olvidar, donde también dispondrá de la opción de poderlas eliminar en caso de que sea necesario. Además encontrarán un botón para agregar nuevos recordatorios, debiendo establecer la fecha y hora de notificación.

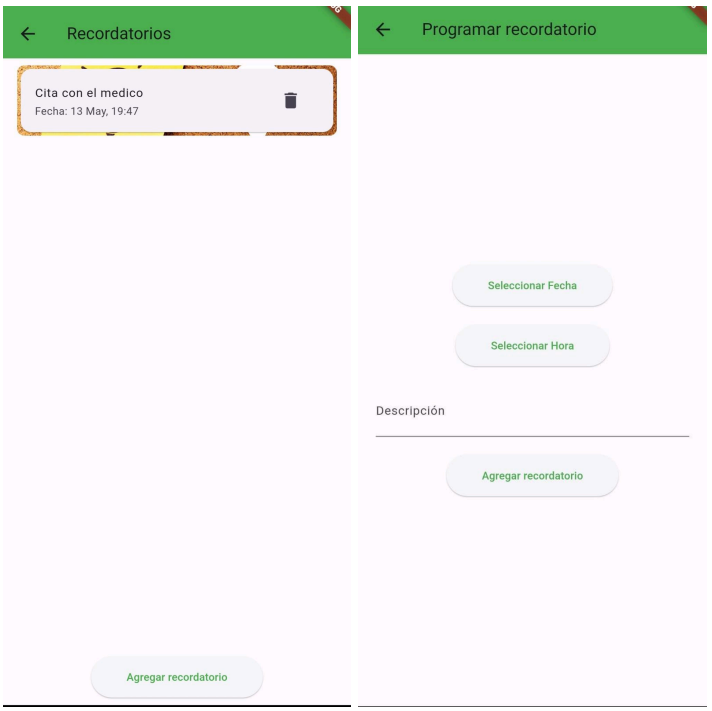


Ilustración 100: Sección de recordatorios

Tanto para los medicamentos como para los recordatorios, se enviará una notificación en caso de que se cumpla la condición.

Apéndice IV Ejemplo de documentación

Como ya se ha mencionado anteriormente, la documentación se compone de un archivo *index.html* y una serie de subcarpetas correspondientes a las diferentes clases que se han documentado.

Si abrimos el archivo principal (*index.html*), nos encontraremos con la siguiente página.

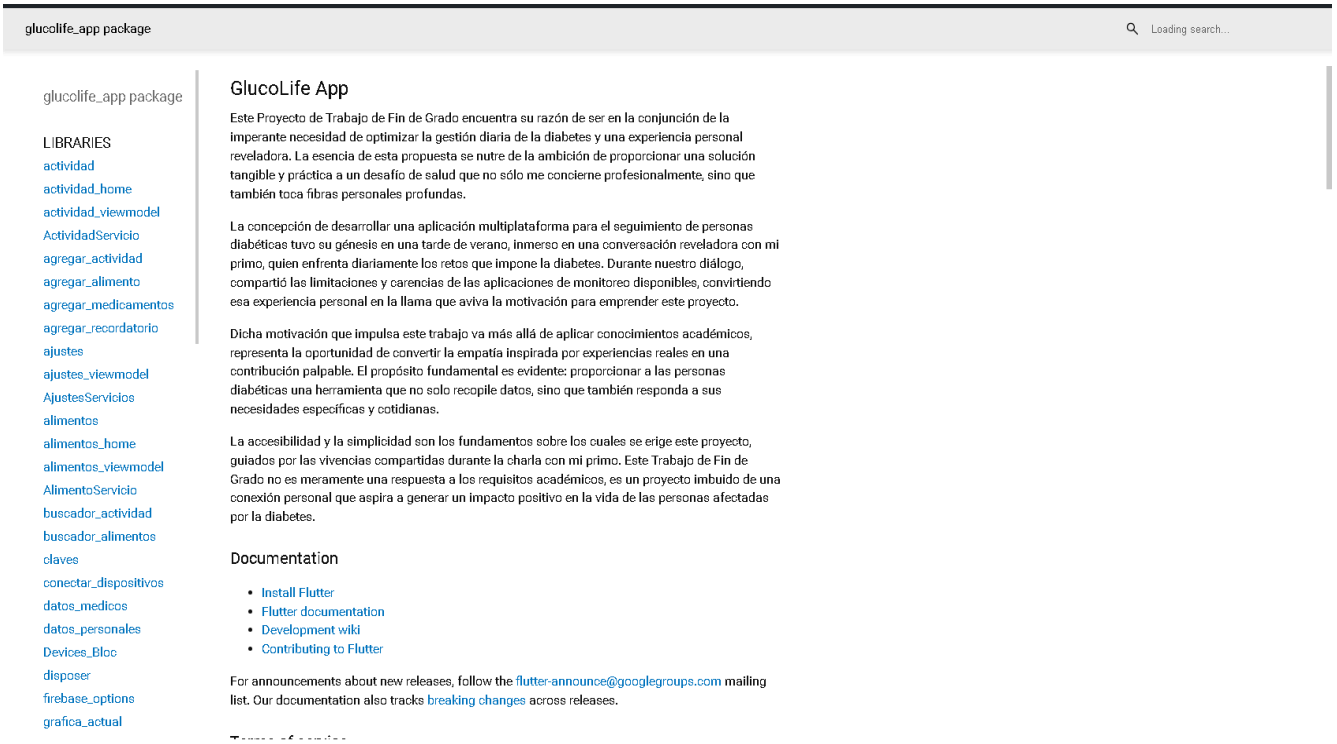


Ilustración 101: Página principal (*index.html*)

En ella encontraremos el contenido del archivo *README* del proyecto, donde podemos ver una breve explicación sobre el proyecto e información sobre *Flutter.* Aparte de esto, también observamos las diferentes librerías que poseen una documentación.

En la siguiente ilustración, se muestra un ejemplo del contenido que se puede encontrar en las diferentes secciones de la aplicación. Dentro de cada uno, se tallan los métodos, operaciones y otras funcionalidades que se utilizan en cada clase.

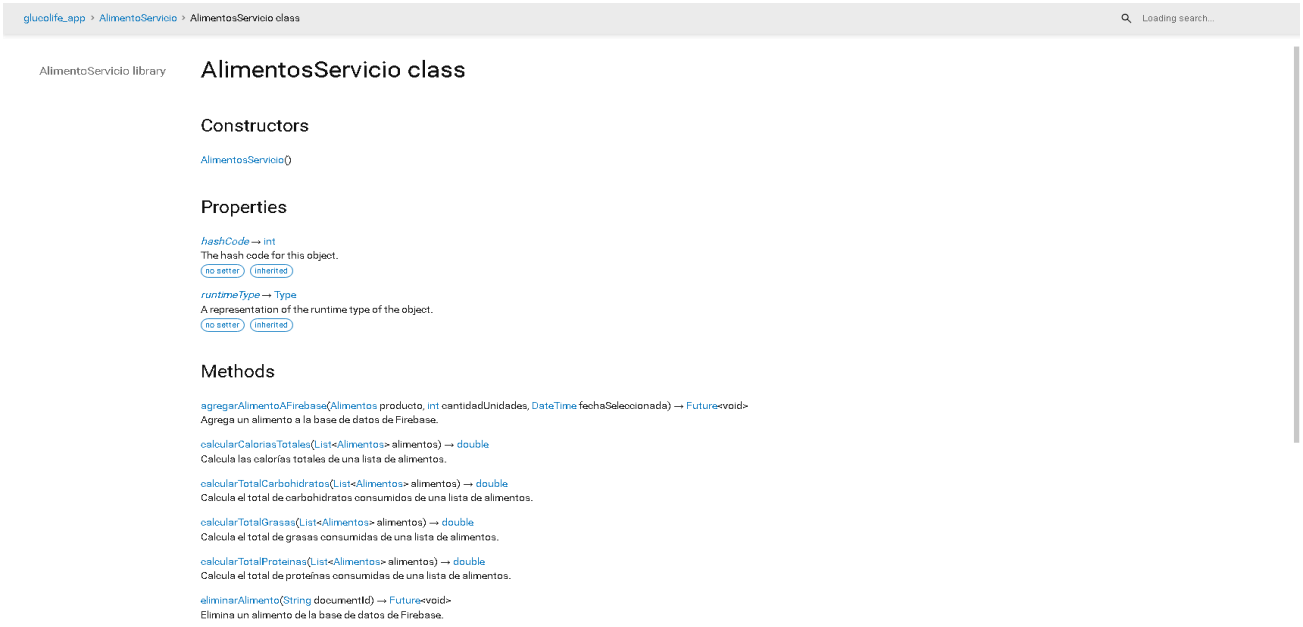


Ilustración 102: Contenido de una clase

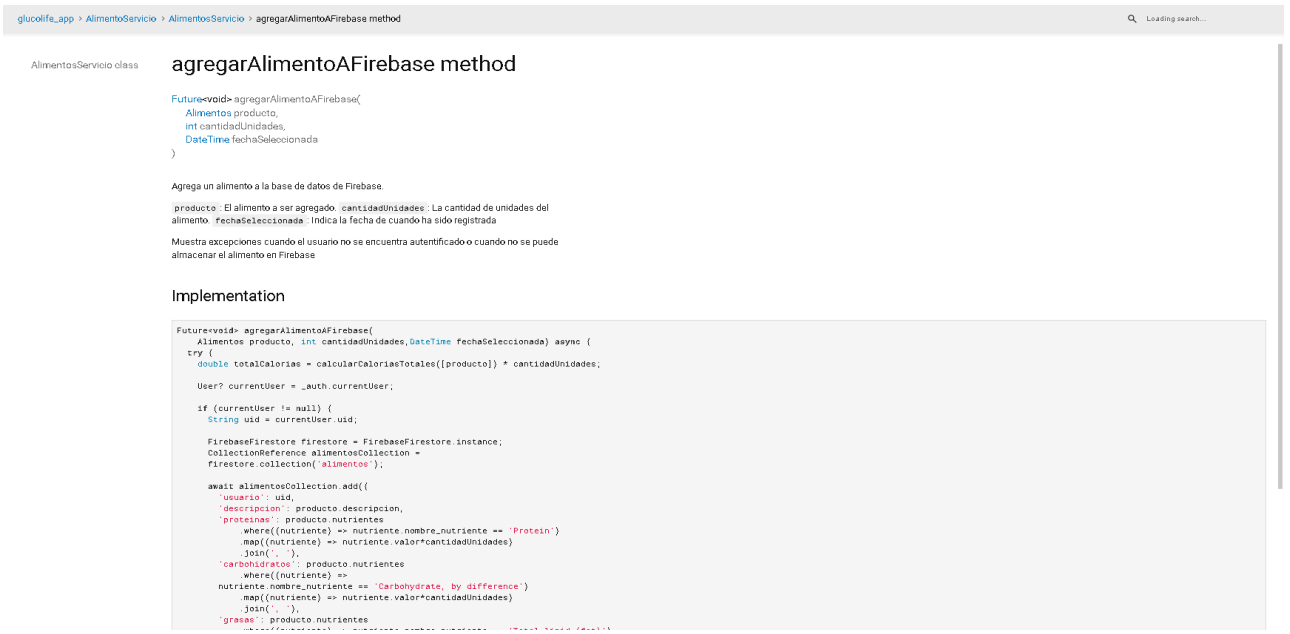


Ilustración 103: Visualización de un método

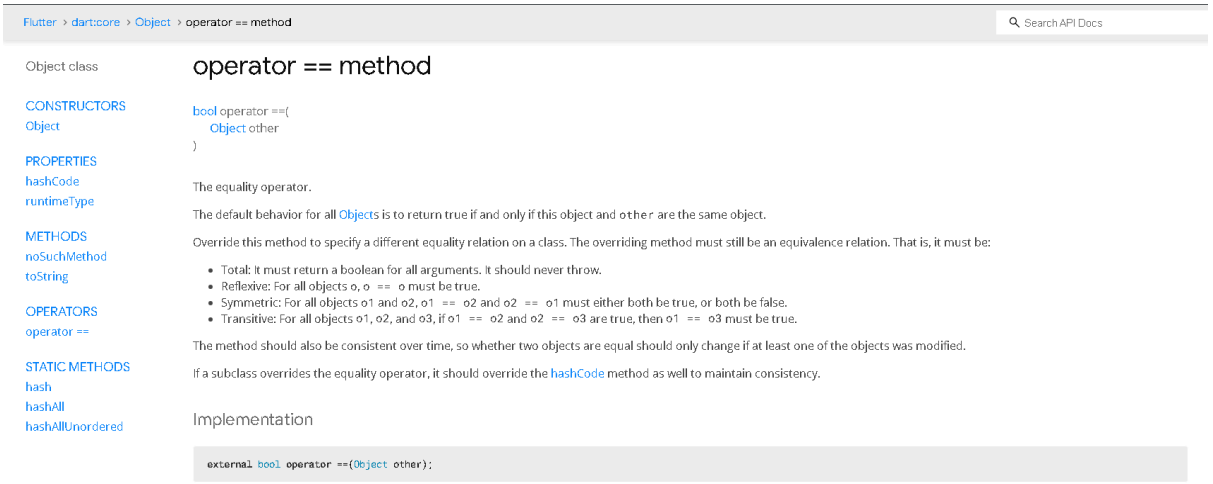


Ilustración 104: Visualización de una operación