UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño de una aplicación móvil para mejorar la gestión del tiempo

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Frasquet Mascarell, Adria

Tutor/a: Vidal Oriola, Germán Francisco

CURSO ACADÉMICO: 2023/2024

Resum

Un problema habitual en estos temps i, especialment, per a les persones que pateixen algun tipus de trastorn d'atenció i/o hiperactivitat (TEA, TDAH, etc.) és la dificultat per organitzar les seues rutines i horaris de manera autònoma. Sovint, es recorre a solucions ad hoc per millorar la gestió del temps: alarmes, notes, la pròpia memòria, etc., amb l'objectiu d'identificar les tasques o rutines del dia de manera que es realitzen totes en el temps previst. En aquest treball, es proposa el desenvolupament d'una aplicació mòbil que ajude a gestionar el temps mitjançant una agenda en la qual es puguen organitzar les activitats del dia, emprant notificacions per als recordatoris, alarmes quan es compleix el temps previst, imatges i pop-ups. Aquest tipus d'aplicació pot resultar útil per a tot el món i, especialment, per a aquelles persones que tinguen algun trastorn d'atenció.

Paraules clau: Organitzar, temps, gestió, aplicació, recordatoris, alarmes, trastorn d'atenció.

Resumen

Un problema habitual en estos tiempos y, especialmente, para las personas que sufren algún tipo de trastorno de atención y/o hiperactividad (TEA, TDAH, etc) es la dificultad para organizar sus rutinas y horarios de forma autónoma. A menudo, se recurre a soluciones ad-hoc para mejorar la gestión del tiempo: alarmas, notas, la propia memoria, etc, con el objeto de identificar las tareas o rutinas del día de forma que se realicen todas en el tiempo previsto. En este trabajo, se propone el desarrollo de una aplicación móvil que ayude a gestionar el tiempo mediante una agenda en la que se puedan organizar las actividades del día, empleando notificaciones para los recordatorios, alarmas cuando se cumple el tiempo previsto, imágenes y pop-ups. Este tipo de aplicación puede resultar útil para todo el mundo y, especialmente, para aquellas personas que tengan algún trastorno de atención.

Palabras clave: Organizar, tiempo, gestión, aplicación, recordatorios, alarmas, trastorno de atención.

Abstract

A common problem these days, especially for people who suffer from some type of attention disorder and/or hyperactivity (ASD, ADHD, etc.), is the difficulty in organizing their routines and schedules autonomously. Often, ad-hoc solutions are used to improve time management: alarms, notes, memory, etc., with the aim of identifying the day's tasks or routines so that they are all completed within the planned time. In this work, the development of a mobile application is proposed to help manage time through a schedule in which daily activities can be organized, using notifications for reminders, alarms when the planned time is met, images, and pop-ups. This type of application can be useful for everyone and, especially, for those who have some attention disorder.

Keywords: Organize, time, management, application, reminders, alarms, attention disorder.

Tabla de contenidos

Índice general

1. Introducción **8**
1.1. Prefacio 9
1.2. Motivación 9
1.3. Objetivos 10
1.4. Estructura 10
2. Estado del arte **12**
2.1. Contexto de las aplicaciones de gestión de tiempo 13
2.1.1. Routinery 13
2.1.2. TimeTune 15
2.1.3. Todoist 17
2.1.4. Me+ 18
2.2. Análisis del estado del arte 20
2.2.1. Conclusiones del Análisis 20
2.2.2. Propuesta 21
2.2.2. Tabla comparativa 22
3. Análisis del problema **24**
3.1. Especificación de requisitos 24
3.1.1 Propósito 24
3.1.2 Ámbito del sistema 24
3.1.3 Propósito del sistema 24
3.1.4 Actores 24
3.1.4 Características 24
3.1.5 Requisitos no funcionales 25
3.1.5 Requisitos funcionales 26
3.2. Diagrama UML 32
4. Diseño de la solución **35**
4.1. Diseño de las interfaces 36
4.2. Arquitectura Modelo-Vista-Controlador 43
4.3. Lenguaje de programación 44
5. Desarrollo de la solución **46**
5.1. Creación de la base de datos 47
5.2. Creación del repositorio del proyecto 48
5.3. Creación del backlog 51
5.4. Implementación de la vista 51
5.4.1 Expo Router 51
5.4.2 React Navigation 52

5.4.4 Pantallas 52
5.4.5 componentes 54
5.5. Implementación del controlador 55
5.5.1 Context 55
5.5.1 Controllers 56
5.6. Implementación del modelo 56
5.7. Resultado final 57
5.8. Análisis del trabajo 72
6. Pruebas **75**
6.1. Herramientas utilizadas 75
6.2. Pruebas unitarias 76
6.3. Pruebas end to end 77
7. Conclusiones y trabajos futuros **79**
Bibliografía **81**
Objetivos de desarrollo sostenible, apéndice A **85**

Índice de figuras

Figura 2.1 Imágenes de Routinery 14
Figura 2.2 Análisis de rutinas, Routinery 14
Figura 2.3 Imágenes de Routinery 15
Figura 2.4 Imágenes TimeTune 16
Figura 2.5 Imágenes TimeTune 16
Figura 2.6 Imágenes Todoist 17
Figura 2.7 Imágenes Todoist 18
Figura 2.8 Imágenes Me+ 19
Figura 2.9 Imágenes Me+ 19
Figura 2.10 Imágenes Me+ 20
Figura 3.1 Diagrama UML de la BD 33
Figura 4.1 Pantalla de TimeSphere, rutinas 37
Figura 4.2 Pantalla de TimeSphere, calendario 38
Figura 4.3 Pantalla de TimeSphere, Diario 39
Figura 4.4 Pantalla de TimeSphere, Supermercados 40
Figura 4.5 Pantalla de TimeSphere, configuración 41
Figura 4.6 Pantalla de TimeSphere, Rutina activa 42
Figura 5.1 Imagen de la BD de TimeSphere 48
Figura 5.2 Estructura inicial de carpetas 48
Figura 5.3 Primer commit 49
Figura 5.4 Estructura de carpetas para el MVC 50
Figura 5.5 worki 51
Figura 5.6 Expo Router 52
Figura 5.8 Pantallas TimeSphere 53
Figura 5.9 Pantallas TimeSphere 54
Figura 5.10 controladores TimeSphere 56
Figura 5.11 modelos TimeSphere 57
Figura 5.12 Pantalla final de rutinas, TimeSphere 58
Figura 5.13 Pantalla final de Calendario, TimeSphere 59
Figura 5.14 Pantalla final de Supermercados, TimeSphere 60
Figura 5.15 Pantalla final de Opciones, TimeSphere 61
Figura 5.16 Pantalla final de crear rutina, TimeSphere 62
Figura 5.17 Pantalla final de crear rutina (selector de hora), TimeSphere 63
Figura 5.18 Pantalla final de editar notificación, TimeSphere 64
Figura 5.19 Pantalla final de editar rutina, TimeSphere 65
Figura 5.20 Pantalla final de editar actividad, TimeSphere 66
Figura 5.21 Pantalla final de editar actividad (Picker), TimeSphere 67
Figura 5.22 Pantalla final de rutina activa, TimeSphere 68
Figura 5.23 Pantalla final de diario, TimeSphere 69
Figura 5.24 Pantalla final de Filtros, TimeSphere 70
Figura 5.25 Pantalla final de editar listas de la compra, TimeSphere 71

Figura 5.26 Pantalla final de listas de la compra, TimeSphere **72**
Figura 5.27 Dashboard con gráficas del desarrollo, Worki **73**
Figura 6.1 pruebas unitarias, TimeSphere **76**
Figura A.1: Objetivos de Desarrollo Sostenible **85**

Índice de tablas

Tabla 2.1 Tabla comparativa, entre las aplicaciones del mercado y TimeSphere. 22
Tabla 3.1 Requisito no funcional, RNF1 26
Tabla 3.2 Requisito no funcional, RNF2 26
Tabla 3.3 Requisito no funcional, RNF3 26
Tabla 3.4 Requisito no funcional, RNF4 26
Tabla 3.5 Requisito funcional 1, configurar detalles de la Interfaz de Usuario 27
Tabla 3.6 Requisito funcional 2, Crear una rutina 27
Tabla 3.7 Requisito funcional 3, Editar una rutina existente 28
Tabla 3.8 Requisito funcional 4, Editar una actividad 28
Tabla 3.9 Requisito funcional 5, Iniciar una rutina5 29
Tabla 3.10 Requisito funcional 6, Acceder a rutinas por el calendario 29
Tabla 3.11 Requisito funcional 7, Configurar las notificaciones 30
Tabla 3.12 Requisito funcional 8, Acceder a las rutinas a través de las notificaciones 30
Tabla 3.13 Requisito funcional 9, Crear una lista de la compra 31
Tabla 3.14 Requisito funcional 10, Crear una lista de la compra 31
Tabla 3.15 Requisito funcional 11, Registrar estado de ánimo 32
Tabla 3.16 Requisito funcional 12, Registrar entrada en el diario 32
Tabla A.1: nivel de relación de nuestro proyecto con las diferentes ODS 86

1. Introducción

1.1. Prefacio

Este proyecto comenzó a desarrollarse conceptualmente en 2023. En ese entonces, nos encontrábamos desarrollando una aplicación para la feria de proyectos, dentro del marco de la asignatura PIN. Quiero agradecer a nuestro amigo Jordi, quien ayudó a que surgiera la idea que terminó convirtiéndose en SPHERE, un gran proyecto creado gracias a los esfuerzos de cinco de los más valientes compañeros que he conocido. En ese proyecto, se quiso crear una aplicación que fuese un lugar de encuentro para padres de niños con TEA, en la cual se pretendía cubrir todas las necesidades que podrían surgir a cualquiera de estos padres. Desde un chat general donde los padres podían publicar fotos, expresar sus problemas y pedir o dar consejo, pasando por herramientas de apoyo a la gestión del tiempo en niños con TEA (la parte que dará vida a este TFG), hasta un buscador de asociaciones cercanas.

Con el tiempo, aprendimos mucho sobre cómo desarrollar código, cómo gestionar el tiempo, cómo testear y, sobre todo, cómo unir ideas que parecían muy dispares. Al final, llegó el día de la presentación. Estábamos preparados, la aplicación estaba completa y sabíamos cómo presentarla. Lo que no esperábamos que ocurriera aquel día fue que una persona con autismo se acercara a preguntar por nuestra aplicación. Ella nos sugirió que estaría bien que también existieran aplicaciones que ayudaran directamente a las personas con autismo. Esta reflexión nos llevó a concluir que tenía toda la razón, no solo por centrar el foco en lo verdaderamente importante, sino también por darle validez a las personas neurodivergentes, quienes por lo general también pueden usar aplicaciones como cualquier otra persona.

Tras esto, nos picó la curiosidad e intentamos conocer cuáles eran algunos de los problemas de estas personas. Fue entonces cuando vimos un problema recurrente dentro de las diferentes comunidades: **la gestión del tiempo es complicada**. Incluso nos fijamos en nosotros mismos y nos dimos cuenta que nos ocurría lo mismo. Puede que muchas personas tengan la determinación para realizar todas sus tareas en el momento que sea necesario, sin retrasos y sin pensarlo más de la cuenta, pero ese no es nuestro caso, y sabemos que tampoco es el de muchos. ¿Quién no ha querido alguna vez poner una lavadora y se le ha olvidado? Incluso hay muchas personas a las que les resulta mucho más complicado centrarse en una actividad y no distraerse, perdiendo el tiempo.

Esta es la problemática que motiva este TFG: no solo la creencia de que la organización es una cuestión crucial en el día a día de las personas, sino también la inclusión de aquellas personas a quienes se les puede dificultar más esta cualidad. Esperamos que este humilde trabajo esté a la altura de las necesidades de todas estas personas.

1.2. Motivación

La motivación de este TFG surge de la observación de las dificultades que tienen muchas personas, especialmente aquellas con trastornos de atención y/o hiperactividad, para organizar sus rutinas y horarios de manera autónoma. Durante un proyecto anterior, donde desarrollamos una aplicación para apoyar a padres de niños con TEA, se detectó la necesidad de herramientas que también beneficien directamente a las personas neurodivergentes. Este trabajo busca no solo mejorar la organización personal, sino también fomentar la inclusión de quienes enfrentan estas dificultades, con la esperanza de satisfacer sus necesidades y hacer su día a día más manejable.

1.3. Objetivos

El principal objetivo de este TFG es la creación de una aplicación móvil para la gestión del tiempo. Dentro de este objetivo general, podemos especificar diferentes objetivos particulares, todos los cuales conformarán la totalidad del proyecto:

- Creación de una aplicación móvil con la que los usuarios puedan realizar una mejor gestión de su tiempo
- Investigación de frameworks y lenguajes de programación que permitan el desarrollo de esta aplicación en diferentes dispositivos móviles
- Diseño e implementación de un sistema de persistencia que permita el almacenaje y la recuperación de los diferentes datos utilizados en nuestra aplicación
- Uso de las metodologías ágiles durante la totalidad del desarrollo de nuestra aplicación
- Investigación de diseños de interfaces que mejoren la visibilidad y legibilidad para cualquier persona

1.4. Estructura

El resto de la memoria se ha dividido en los siguientes seis capítulos:

2. Estado del arte: En este apartado se realiza un estudio de soluciones similares a nuestra propuesta, lo que nos permite contextualizar y definir mejor las características de nuestra aplicación en comparación con los productos ya existentes.

3. Análisis del problema: Recoge el análisis de los requisitos que debe cumplir el software que se pretende desarrollar. En este apartado también se presenta un diagrama UML para nuestro sistema de almacenamiento de datos persistentes.

4. Diseño de la solución: En este capítulo se recoge todas las decisiones de diseño que deben tomarse antes de iniciar la implementación del código. En esta parte se aborda el diseño

de los test y preguntas, las pautas para crear nuestra interfaz de usuario, la arquitectura de software que se utiliza y se describen todas las tecnología que se utilizarán durante el proyecto.

5. Desarrollo de la solución: Este capítulo describe cómo se ha realizado la implementación de cada una de las capas de nuestra aplicación y cómo han sido utilizados los servicios externos de los que se hace uso. También se expone el resultado final de la implementación y cómo sería una sesión habitual de uso de la aplicación.

6. Pruebas: Este apartado recoge las diferentes pruebas llevadas a cabo para testear el correcto funcionamiento de la aplicación.

7. Conclusión: En este apartado se realiza un ejercicio de retrospectiva y se determina si los objetivos expuestos al principio de la memoria han sido logrados con éxito. También se relaciona el trabajo realizado con todos los conocimientos obtenidos durante los estudios cursados. Por último, se exponen las diversas propuestas que pretenden llevar a cabo en futuras versiones de la aplicación.

2. Estado del arte

Para determinar la viabilidad de la aplicación final que se plantea desarrollar en este trabajo, primero debemos analizar en qué contexto de mercado se distribuirá, examinando pormenorizadamente las características que podemos encontrar a día de hoy repartidas en diferentes aplicaciones que podrían llegar a ser potenciales competidoras.

2.1. Contexto de las aplicaciones de gestión de tiempo

Hoy en día, el mercado de aplicaciones móviles está abarrotado, lo que provoca dificultades para los desarrolladores a la hora de producir aplicaciones novedosas que realmente puedan contribuir a la calidad de vida de las personas.

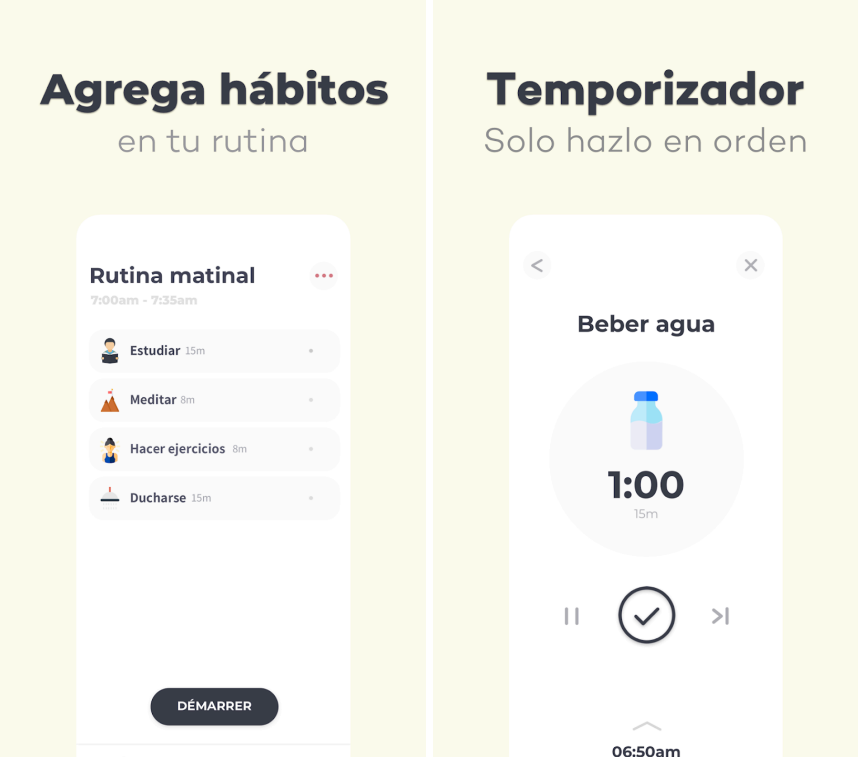
En concreto el mercado de las aplicaciones de gestión del tiempo tiene una competencia intensa, en la que las empresas líderes compiten en ámbitos como la funcionalidad, la usabilidad, y la integración con otras herramientas. Por tanto, vamos a analizar si existe un hueco dentro de este mercado para una aplicación como la que se plantea en este trabajo, en la que se prioriza la accesibilidad.

Las aplicaciones que vamos a analizar han sido seleccionadas siguiendo una prioridad de popularidad y valoración en la Play Store de Google.

2.1.1. Routinery

Routinery [1] es una aplicación diseñada para ayudar a las personas a mejorar su concentración, aumentar sus horas de sueño y evitar la procrastinación, entre otros beneficios. Una de las principales características de Routinery es la posibilidad de crear horarios lineales en forma de rutinas, que pueden ser tanto predefinidas como personalizadas por los usuarios según sus necesidades (Figura 2.1a). Estas rutinas se desarrollan de manera secuencial, guiando al usuario a través de cada actividad programada, y le indican cuánto tiempo debe dedicar a cada una (Figura 2.1b). Además, la aplicación envía notificaciones oportunas para recordar al usuario cuándo es momento de pasar a la siguiente actividad.

Además de estas funciones principales, Routinery también ofrece herramientas de seguimiento de rutinas, que permiten a los usuarios analizar el cumplimiento de sus horarios y evaluar su desempeño a lo largo del tiempo (Figura 2.2). Esto proporciona una visión detallada del progreso personal, ayudando a los usuarios a identificar áreas de mejora y ajustar sus rutinas para optimizar su productividad y bienestar.



A Ejemplo de rutina B. Rutina en ejecución

Figura 2.1 Imágenes de Routinery

***Figura 2.2** Análisis de rutinas, Routinery*

2.1.2. TimeTune

TimeTune [2] es una aplicación que se centra en mejorar la productividad de las personas, aportando herramientas que optimizan la gestión del tiempo dentro de un día. Para ello, esta app ofrece una agenda (Figura 2.3a) en la que los usuarios pueden crear su propio horario para seguir durante el día o crear plantillas (Figura 2.3b) que puedan utilizarse en otros días del mes usando el calendario (Figura 2.4).

Para que estos horarios sean efectivos, TimeTune incluye estadísticas del tiempo para poder analizar globalmente las actividades que se realizan durante el día (Figura 2.5b). Además, esta app utiliza notificaciones para recordar la actividad que se debe realizar; estas notificaciones pueden personalizarse (Figura 2.5a).

A. Agenda de TimeTune B. Plantillas de TimeTune

Figura 2.3 *Imágenes de Routinery*

Figura 2.4 Imágenes TimeTune

A. Estadísticas de tiempo B. Notificaciones personalizadas

Figura 2.5 Imágenes TimeTune

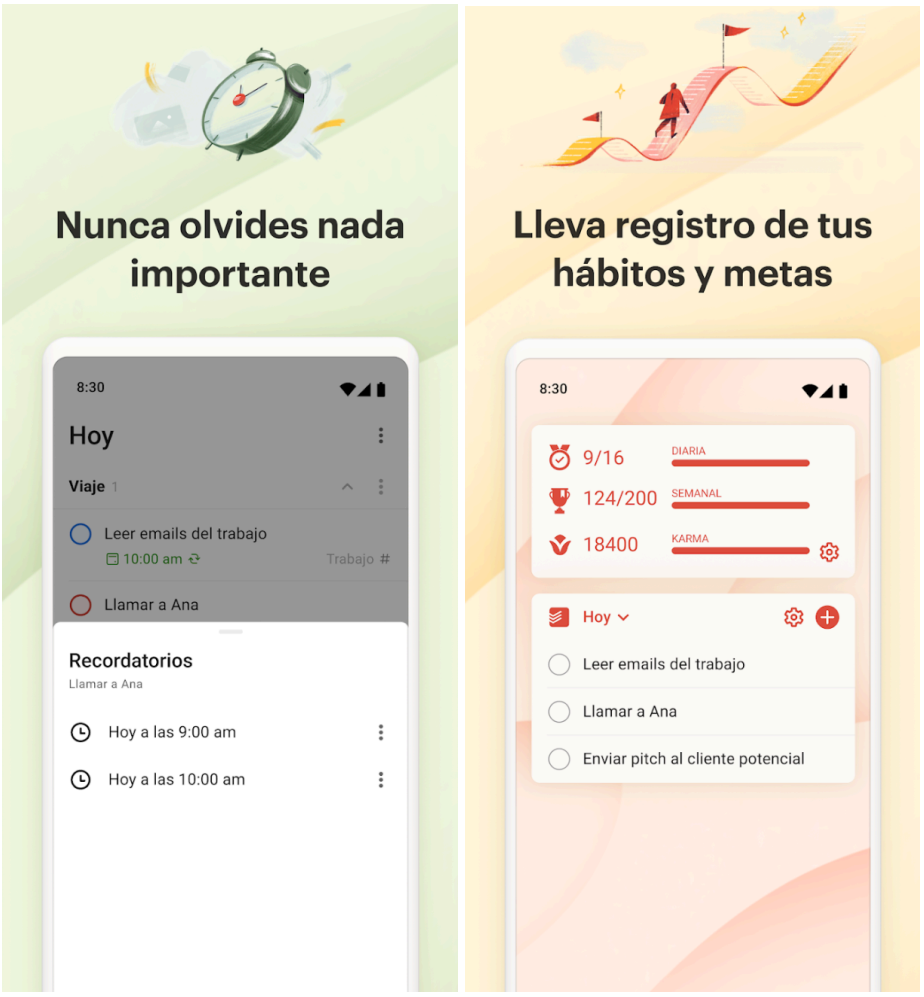
2.1.3. Todoist

Organizar tanto a equipos como a usuarios individuales es el objetivo principal de *Todoist* [3], para ello esta app nos aporta diferentes herramientas, como una checklist con prioridades (Figura 2.6a) o un horario con tareas personalizadas (Figura 2.6b).

A. Checklist con prioridad B. horario semanal

Figura 2.6 Imágenes Todoist

Como en otras aplicaciones mencionadas anteriormente, esta app incluye un sistema de notificaciones personalizadas (Figura 2.7a) y una pantalla de estadísticas en la que podemos analizar nuestro uso del tiempo mientras utilizamos las herramientas de la app (Figura 2.7b).



A. Notificaciones personalizadas B. Estadísticas de tiempo

Figura 2.7 Imágenes Todoist

2.1.4. Me+

Me+ [4] es una app de gestión del tiempo enfocada al tiempo personal. Dentro de sus objetivos principales se encuentra el que sus usuarios creen hábitos diarios y rutinas saludables. Para ello, nos brinda varias herramientas, la principal y que primero se nos presenta es la pantalla de rutinas (Figura 2.8), en la que podemos crear rutinas para ese mismo día o para diferentes días de la semana.

Además, una de las funciones interesantes de Me+ es la inclusión de diferentes recursos en línea que pueden ayudar a los usuarios a motivarse para cumplir sus horarios o encontrar nuevas rutinas que encajen mejor con ellos (Figura 2.9a y 2.9b).

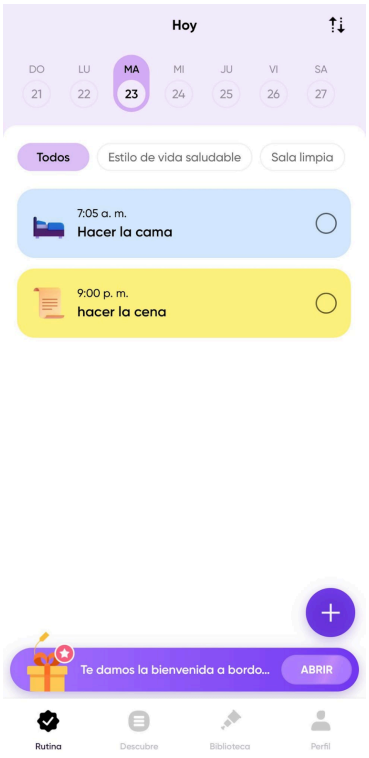
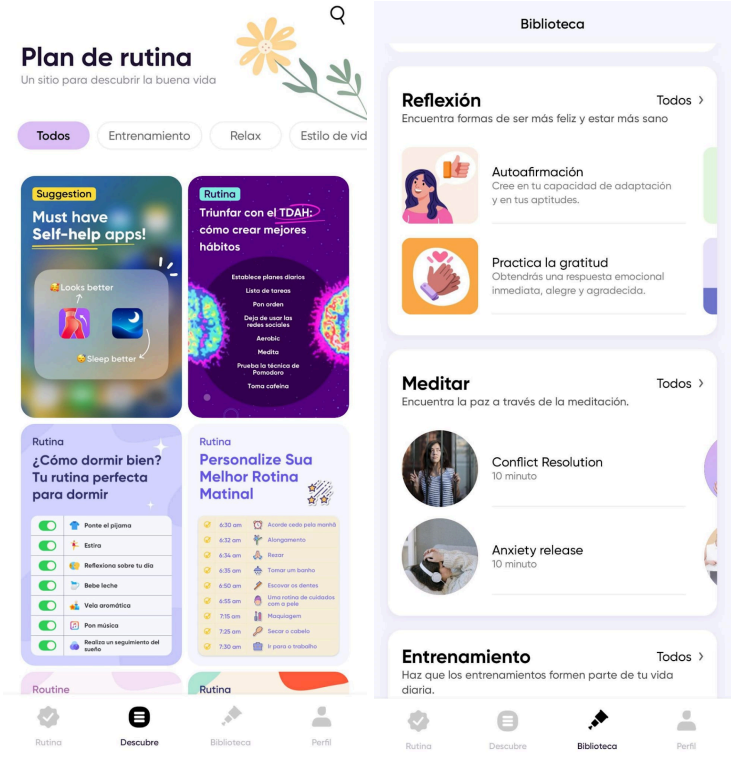


Figura 2.8 Imágenes Me+



A. Recursos para horarios B. Recursos para nuevas rutinas

Figura 2.9 Imágenes Me+

Por último, cabe mencionar otra herramienta interesante dentro de la app: un calendario de seguimiento del estado de ánimo (Figura 2.10), en el que los usuarios pueden registrar entradas diarias y revisarlas en cualquier momento.

Figura 2.10 Imágenes Me+

2.2. Análisis del estado del arte

2.2.1. Conclusiones del Análisis

Tras analizar diferentes aplicaciones que entran dentro del marco de las aplicaciones de gestión del tiempo, podemos sacar algunas conclusiones sobre por qué estas y no otras se encuentran en lo más alto dentro de la Play Store. Primero, destacamos las funcionalidades más compartidas entre las aplicaciones analizadas.

- Uno de los puntos que comparten todas las aplicaciones analizadas, y que parece ser el principal, es la creación de rutinas. Aunque estas no se crean de igual forma en todas las aplicaciones, por ejemplo, en Todoist, las rutinas son simples líneas de texto que describen una actividad que está o no terminada, mientras que en TimeTune son actividades que contienen varios detalles como la hora a la que se deben realizar, su duración y una descripción.
- Otro punto compartido entre las aplicaciones es el de las estadísticas de uso de la app, donde podemos comprobar si estamos realizando las actividades de manera adecuada y puntual o no. Esta funcionalidad puede ayudar a mantener la motivación en los usuarios más activos al reconfortarse viendo su progreso y a dar un toque de atención a los usuarios menos activos.
- De igual manera, una función que comparten la mayoría de las aplicaciones analizadas es la de un sistema de notificaciones, en el que los usuarios reciben recordatorios para realizar las actividades programadas a cierta hora. Estas notificaciones pueden ser personalizadas o automáticas dependiendo de la aplicación.

El resto de las funcionalidades no se han destacado debido a que se encuentran en una sola app o en dos aplicaciones pero implementadas de maneras diferentes, evidenciando que no son funciones principales dentro de las apps de gestión del tiempo.

2.2.2. Propuesta

Tras haber analizado las diferentes funciones presentes en las diversas aplicaciones de gestión del tiempo, podemos realizar nuestra propuesta sabiendo cuáles serán sus puntos fuertes y diferenciadores.

Primeramente, el nombre de nuestra app será *TimeSphere*, nombre que hace referencia tanto a que es una aplicación relacionada con el tiempo, como a la aplicación *Sphere* [5], ya que estas dos apps se encuentran íntimamente relacionadas.

A continuación, presentamos las funciones que hemos decidido implementar tras analizar el estado del arte:

- Una función centrada en la creación de rutinas, similar a las funciones vistas en las diferentes aplicaciones analizadas. Además cuando se estén realizando estas rutinas, se podrá visualizar imágenes que mejoren la accesibilidad además de ayudar a la concentración en la actividad
- Desarrollo de un calendario, que ayude a visualizar las rutinas en cualquier día, y las entradas del seguimiento semanal y el diario
- Implementación de un sistema de alarmas personalizables que notifique al usuario cuándo debe realizar una tarea en particular.
- Desarrollo de una sección para listas de la compra, con imágenes y un localizador de supermercados accesibles.
- Creación de una sección de seguimiento emocional, en la que se pueda anotar visualmente el estado de ánimo diario.

- Diseñar una función de Diario en la que los usuarios puedan tomar nota de las cosas que consideren interesantes día a día.

Destacar que hay funcionalidades como la de estadísticas que a pesar de ser una función útil para las necesidades de nuestros posibles usuarios, hemos decidido no incluir debido a que representan un desafío técnico que podría requerir mucho tiempo y que su implementación podría resultar en una reducción de la calidad del resto de funciones.

2.2.2. Tabla comparativa

Vistas las funciones que vamos a implementar, ya podemos comparar las funciones disponibles en aplicaciones del mercado y las que encontraremos en nuestra app. Para ello crearemos una Tabla comparativa (Tabla 2.1).

	TimeSphere	Routinery	TimeTune	Todoist	Me+
Rutinas	Sí	Sí	Sí	Sí	Sí
Agenda	Sí	No	Sí	Sí	No
Notificaciones	Sí	Sí, pero no personalizables	Sí	Sí	Sí, pero no personalizables
Lista de la compra	Sí	No	No	No	No
Comercios accesibles	Sí	No	No	No	No
Seguimiento emocional	Sí	No	No	No	Sí
Diario	Sí	No	No	No	Sí
Recursos para nuevas rutinas	No	No	No	No	Sí
Estadísticas de uso	No	Sí	Sí	Sí	No
Sugerencias de posibles rutinas	No	No	No	No	Sí
Plataformas disponibles	Android, IOS	Android, IOS	Android	Android, IOS	Android, IOS

Tabla 2.1 Tabla comparativa, entre las aplicaciones del mercado y TimeSphere.

3. Análisis del problema

Habiendo analizado ya el contexto en el que nuestra aplicación se va a desarrollar y en el que deberá competir, nos disponemos a realizar el análisis del problema. En este análisis detallaremos las funciones que deberá realizar la aplicación y plantearemos los requisitos funcionales y no funcionales que deberá cumplir. De igual manera realizaremos un diagrama *UML* [6] con el que definiremos la estructura de nuestra base de datos. Este apartado ayudará a que nuestro desarrollo sea de calidad.

3.1. Especificación de requisitos

3.1.1 Propósito

Para garantizar que la especificación de requisitos de nuestro proyecto se realice de manera correcta, clara y completa, deberemos analizar los siguientes componentes de nuestro problema; El ámbito del sistema, el propósito del sistema, los actores, las características, los requisitos no funcionales y los funcionales.

3.1.2 Ámbito del sistema

El ámbito del sistema se restringe a la gestión del tiempo y sus funciones concretas como la creación de rutinas, agendas o notificaciones personalizadas, con un enfoque en la claridad y la accesibilidad para cualquier persona. Cumpliendo con las metodologías de desarrollo que aseguran un software de calidad.

3.1.3 Propósito del sistema

El propósito de TimeSphere es ser una herramienta para la gestión del tiempo y la organización personal, enfocándose en la accesibilidad y en la mejora del bienestar emocional. Su principal objetivo es proporcionar a los usuarios una herramienta útil y personalizable que no solo les permita organizar sus actividades diarias y semanales, sino también mejorar su calidad de vida mediante la planificación de rutinas, la gestión de tareas, y el seguimiento emocional.

3.1.4 Actores

En nuestra aplicación podremos encontrar un único actor. El usuario final, la persona que podrá interactuar con la aplicación creando rutinas o revisando su diario, por ejemplo. Estas personas necesitarán una forma de modificar las opciones visuales y de sonido.

3.1.4 Características

1. Interfaz de Usuario Intuitiva y Accesible: La aplicación presentará un diseño visual limpio, con uso de colores que contrastan entre sí, así como tipografías legibles para facilitar la navegación y su uso por parte de personas con diferentes capacidades. Además, la app contará con la integración de imágenes que acompañen las rutinas y tareas para facilitar la comprensión y el seguimiento, especialmente para usuarios con dificultades cognitivas o visuales. Por último, los usuarios podrán personalizar la apariencia de la interfaz, incluyendo la opción de elegir entre tema claro o oscuro.

2. Gestión de Rutinas: La gestión de rutinas incluye la posibilidad de crear y editar rutinas diarias o semanales de manera sencilla, con la opción de agregar tareas, asignarles horarios específicos, los cuales contarán con imágenes asociadas para mejorar la concentración. Además, estas rutinas contarán con una funcionalidad para ver el progreso de la rutina que se está realizando, con un indicador visual claro del estado de la tarea.

3. Calendario: Los usuarios podrán utilizar un calendario con el que poder acceder a las rutinas establecidas en días distantes, permitiendo al usuario organizar sus días con mucha antelación y además acceder a las funciones de seguimiento emocional y diario personal.

4. Sistema de Alarmas Personalizables: Los usuarios podrán establecer alarmas y recordatorios para las rutinas, con opciones para personalizar la hora, la vibración y el mensaje de la notificación.

5. Listas de Compras con Localizador: Se permitirá crear, editar y organizar listas de compras, con la posibilidad de agregar imágenes de los productos para una mejor identificación. Además, esta funcionalidad contará con un localizador de supermercados accesibles con el que encontrar supermercados cercanos con accesibilidad mejorada, proporcionando información sobre rampas, horarios con iluminación y música reducida, u otro tipo de sistemas de accesibilidad, esta función sería exclusiva de nuestra aplicación pues no se encuentra en ninguna de las demás aplicaciones que analizamos en el capítulo anterior.

6. Seguimiento Emocional: Los usuarios podrán registrar su estado de ánimo diario utilizando emoticonos, colores u otros elementos visuales que reflejen sus emociones. La aplicación también ofrecerá una visión general del estado emocional del usuario a lo largo del tiempo, estilo calendario.

7. Funcionalidad de Diario Personal: Los usuarios podrán escribir en un diario digital, registrando sus pensamientos, experiencias o cualquier otro dato que consideren relevante. Esta funcionalidad estará situada en la misma pantalla que la del seguimiento emocional.

3.1.5 Requisitos no funcionales

Número de requisito	RNF1
Descripción	La aplicación debe presentar un diseño visual limpio, con uso de colores que contrasten entre sí, y tipografías legibles para facilitar la navegación y uso por parte de personas con diferentes capacidades.
Prioridad	Alta

Tabla 3.1 Requisito no funcional, RNF1

Número de requisito	RNF2
Descripción	La aplicación debe tener una buena usabilidad, para ello deben evitarse funciones que requieran de tres pasos o más.
Prioridad	Alta

Tabla 3.2 Requisito no funcional, RNF2

Número de requisito	RNF3
Descripción	La aplicación debe asegurar la seguridad de los datos guardados en las diferentes funcionalidades
Prioridad	Alta

Tabla 3.3 Requisito no funcional, RNF3

Número de requisito	RNF4
Descripción	La aplicación debe asegurar compatibilidad con las últimas versiones de los sistemas operativos a los que va dirigida. IOS y Android.
Prioridad	Alta

Tabla 3.4 Requisito no funcional, RNF4

3.1.5 Requisitos funcionales

Número de requisito	RF1
Nombre	Configurar detalles de la Interfaz de Usuario
Descripción	Los usuarios finales podrán acceder a una configuración que les permita modificar el tema (claro/oscuro) y activar o desactivar el sonido y/o la vibración.
Características	1. Interfaz de Usuario Intuitiva y Accesible
Requisitos no funcionales	RNF1, RNF2, RNF4
Prioridad	Media
Actor	Usuario final

Tabla 3.5 Requisito funcional 1, configurar detalles de la Interfaz de Usuario

Número de requisito	RF2
Nombre	Crear una rutina
Descripción	Para crear una rutina el usuario deberá acceder a la pantalla de crear rutina, ahí aparecerá un lista de opciones, en el que se podrá elegir el nombre de la rutina, la hora de inicio y activar o desactivar la notificación, o personalizarla (RF6). Luego tendremos que pulsar sobre el botón de Hecho, el cual nos llevará a una pantalla principal.
Características	2. Gestión de Rutinas
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.6 Requisito funcional 2, Crear una rutina

Número de requisito	RF3
Nombre	Editar una rutina existente
Descripción	Para editar una rutina, el usuario deberá seleccionar una rutina existente. Al hacerlo, se mostrará una pantalla con los datos de la rutina en la parte superior y una lista con las actividades de la rutina en el centro. Si la rutina aún no tiene actividades establecidas, en la lista aparecerá un mensaje que dirá 'No hay actividades disponibles', para añadir alguna actividad se deberá pulsar sobre el botón ‘Nueva actividad’.
Características	2. Gestión de Rutinas
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.7 Requisito funcional 3, Editar una rutina existente

Número de requisito	RF4
Nombre	Editar una actividad
Descripción	Para editar una actividad, el usuario deberá introducir los datos de la actividad en la pantalla de edición de actividad, accesible desde la pantalla de editar rutina. Los datos de la actividad son: título, duración y tipo de actividad. Una vez que se hayan completado estos campos, se podrá pulsar un botón para guardar la actividad y volver a la pantalla de editar rutina.
Características	2. Gestión de Rutinas
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.8 Requisito funcional 4, Editar una actividad

Número de requisito	RF5
Nombre	Iniciar una rutina
Descripción	Para iniciar una rutina, el usuario deberá acceder a la pantalla de edición de rutinas. En esta pantalla, deberá pulsar el botón 'Iniciar rutina'. Al hacerlo, se dirigirá a una pantalla en la que se mostrarán las actividades de manera sucesiva, con su título en la parte superior, su imagen en el centro y su duración en un contador. Cada actividad durará el tiempo establecido durante su creación, y se podrá ver el tiempo restante en un componente contador que actuará como reloj. Este componente permitirá pausar, reanudar o saltar a la siguiente actividad. Si la actividad es la última, se regresará a la pantalla inicial.
Características	2. Gestión de Rutinas
Requisitos no funcionales	RNF1, RNF2, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.9 Requisito funcional 5, Iniciar una rutina5

Número de requisito	RF6
Nombre	Acceder a rutinas por el calendario
Descripción	La pantalla de rutinas solo mostrará las rutinas del día actual. Para acceder a las rutinas del día siguiente o a las del anterior, se podrá acceder a través de un pequeño gráfico de la semana en la parte superior de la pantalla, o a través del calendario, seleccionando un día en concreto.
Características	3. Calendario
Requisitos no funcionales	RNF1, RNF2, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.10 Requisito funcional 6, Acceder a rutinas por el calendario

Número de requisito	RF7
Nombre	Configurar las notificaciones
Descripción	Cada Rutina tendrá una opción de activar notificaciones y una de personalizar la notificación. Si pulsamos sobre la última, podremos configurar si queremos sonido, vibración y que texto contendrá esta notificación. Estos componentes no pueden ser nulos y si no se llegase a personalizar la notificación pero sí se activase, se realizaría una notificación generica con el nombre de la rutina.
Características	4. Sistema de Alarmas Personalizables
Requisitos no funcionales	RNF1, RNF2, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.11 Requisito funcional 7, Configurar las notificaciones

Número de requisito	RF8
Nombre	Acceder a las rutinas a través de las notificaciones
Descripción	Si el usuario es notificado, este podrá pulsar sobre la notificación para abrir la aplicación e iniciar directamente la rutina.
Características	4. Sistema de Alarmas Personalizables
Requisitos no funcionales	RNF1, RNF4
Prioridad	Baja
Actor	Usuario final

Tabla 3.12 Requisito funcional 8, Acceder a las rutinas a través de las notificaciones

Número de requisito	RF9
Nombre	Crear una lista de la compra
Descripción	Al acceder a la pantalla de listas de la compra podrá ver una lista con las listas ya creadas. En ella aparecerá un botón en el que podrá crear una lista de la compra en la que el usuario podrá introducir productos con sus respectivas imágenes, mejorando así la accesibilidad y la claridad de la lista
Características	5. Listas de Compras con Localizador
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Media
Actor	Usuario final

Tabla 3.13 Requisito funcional 9, Crear una lista de la compra

Número de requisito	RF10
Nombre	Localizar supermercados accesibles
Descripción	En la misma pantalla que las listas de la compra habrá una opción para acceder al localizador de supermercados accesibles, en el que el usuario podrá ver una lista de supermercados ordenados por distancia y en el que se especificará qué tipo de accesibilidad tienen. Estos se podrán filtrar por un tipo de accesibilidad en concreto.
Características	5. Listas de Compras con Localizador
Requisitos no funcionales	RNF1, RNF2, RNF4
Prioridad	Media
Actor	Usuario final

Tabla 3.14 Requisito funcional 10, Crear una lista de la compra

Número de requisito	RF11
Nombre	Registrar estado de ánimo
Descripción	El usuario podrá acceder al calendario del seguimiento emocional. Sí se pulsa sobre un día, nos moverá a la pantalla correspondiente, en la cual se podrá añadir o modificar el estado de ánimo.
Características	6. Seguimiento Emocional
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.15 Requisito funcional 11, Registrar estado de ánimo

Número de requisito	RF12
Nombre	Registrar entrada en el diario
Descripción	El usuario podrá acceder al calendario del seguimiento emocional. Al pulsar sobre un día, se abrirá la pantalla correspondiente, en la que se puede añadir o modificar la entrada del diario.
Características	7. Funcionalidad de Diario Personal
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta
Actor	Usuario final

Tabla 3.16 Requisito funcional 12, Registrar entrada en el diario

3.2. Diagrama UML

Para almacenar de manera eficiente los datos de los usuarios, como sus rutinas, listas de la compra, entradas en el diario y otros tipos de información personal, es esencial utilizar una herramienta de persistencia que permita gestionar estos datos. En nuestro caso, hemos optado por utilizar una base de datos noSQL implementada en Firebase, una plataforma que se adapta

perfectamente a las necesidades de nuestra aplicación. Firebase nos proporcionará las capacidades necesarias para almacenar y recuperar datos de manera rápida y escalable, garantizando al mismo tiempo la integridad y disponibilidad de la información. La estructura y organización de los datos se seguirán según el diagrama UML presentado en la Figura 3.1, que servirá como guía para la implementación del modelo de datos en nuestra aplicación.

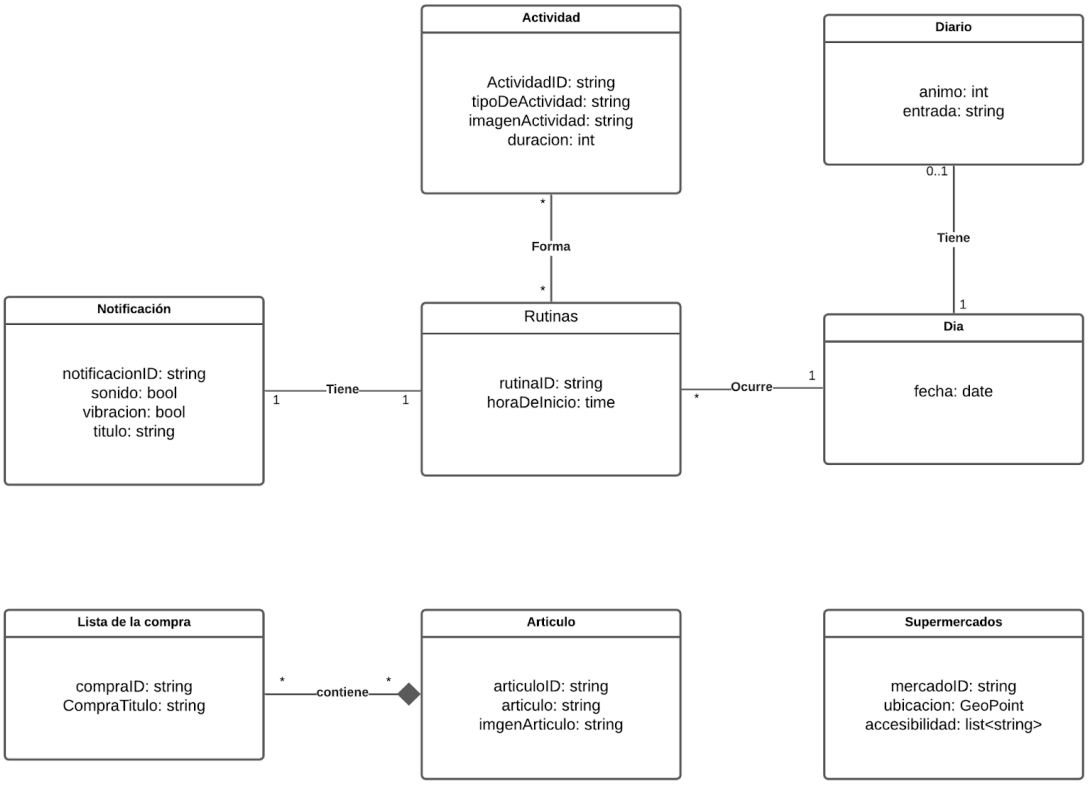


Figura 3.1 Diagrama UML de la BD

Las diferentes clases y atributos que se describen en este diagrama son los siguientes:

- Rutinas: esta clase representará las diferentes rutinas que el usuario cree.
  - rutinaID: cadena que identifica cada rutina de manera única.
  - horaDeInicio: este atributo define a qué hora debe iniciarse la rutina.
- Notificación: esta clase representa las notificaciones que siempre tienen las rutinas.
  - notificacionID: esta cadena identifica las notificaciones de manera única.
  - sonido: este booleano indica si la notificación debe hacer o no sonido.
  - vibración: este booleano indica si la notificación debe vibrar o no.
  - título: esta cadena indica qué texto debe mostrar la notificación.
- Actividad: esta clase representa las diferentes actividades que forman parte de las rutinas.
  - ActividadID: esta cadena identifica cada actividad de manera única.

    - tipoDeActividad: esta cadena identifica el tipo de actividad (higiene, deporte, cocina, etc).
    - imagenActividad: esta cadena representa la url de la imagen que debe aparecer durante la rutina cuando se esté ejecutando esta actividad.
    - duración: número que nos indica cuanto tiempo (en minutos) tardará la actividad en realizarse.
- Diario: esta clase representa el estado de ánimo y la entrada en el diario de un día cualquiera.
    - ánimo: este entero indica el estado de ánimo que se tuvo el día al que pertenece; el 0 representa la falta de estado de ánimo; el resto de enteros empezando por el 1 y terminando en X representan los diferentes estados de ánimo.
    - entrada: esta cadena representa el texto de la entrada del diario correspondiente al día al que pertenece.
- Día: esta clase representa un día cualquiera.
    - fecha: esta fecha en formato Date indica la fecha concreta del día, identificándolo de manera única.
- Lista de la compra: esta clase representa las diferentes listas de la compra que los usuarios pueden crear.
    - compraID: esta cadena identifica la lista de la compra de manera única.
    - compraTitulo: esta cadena representa el título de la lista de la compra.
- Artículo: esta clase representa los diferentes artículos que se pueden introducir en una lista de la compra
    - articuloID: esta cadena identifica cada artículo de manera única.
    - artículo: esta cadena representa el nombre del artículo.
    - imagenArticulo: esta cadena representa la url de la imagen del artículo.
- Supermercados: esta clase representa los diferentes supermercados que podemos encontrar en el buscador.
    - mercadoID: esta cadena identifica cada supermercado de manera única.
    - ubicación: este geoPoint contiene la información de la ubicación de los supermercados con su latitud y su longitud.
    - accesibilidad: esta lista describe los diferentes tipos de accesibilidad que tiene el supermercado.

4. Diseño de la solución

Una vez especificados y analizados los diferentes requisitos de nuestra aplicación, podemos iniciar la etapa de diseño. En esta fase, explicaremos detalladamente cómo planeamos implementar tanto las diversas características como los requisitos, además de realizar los diseños de las interfaces y plantear la arquitectura del código.

4.1. Diseño de las interfaces

En este apartado diseñaremos las diferentes interfaces de la aplicación para conseguir que la aplicación siga las especificaciones, y sea posible realizar todos los casos de uso especificados en el capítulo anterior. Usaremos la página web "*WireframePro*" [7] para crear los mockups que las definan. Para el correcto desarrollo de nuestras interfaces hemos seguido consejos de diferentes fuentes documentales como pueden ser "*Design for Accessibility: 7 Essential Principles for Inclusive UX Designs"* [8], o "*What Are Accessible Mobile Apps and How To Design Them"* [9], *"Master Mobile UI Design: Delight Your Users"* [10], *"Creating Universal Design Experiences: The Power of Intuitive Mobile App Design"* [11] y por último las 10 reglas heurísticas de Nielsen [12], Dentro de todas estas guías se encuentran los consejos que hemos utilizados a la hora de diseñar nuestra app, de entre los que destacamos los siguientes:

"*La interfaz debe ser intuitiva, permitiendo a los usuarios navegar sin esfuerzo por la aplicación. Esto se logra a través de menús bien estructurados, íconos claros y una organización lógica del contenido. Un diseño intuitivo es crucial porque reduce la carga cognitiva y mejora la accesibilidad para todos los usuarios, incluidos aquellos con discapacidades cognitivas o de memoria.*" [10]

"*Utilizar una jerarquía visual clara con el uso adecuado de tipografía, colores y disposición del contenido es clave para guiar la atención del usuario y hacer la información fácilmente digerible. Un diseño visual efectivo no solo es atractivo, sino que también facilita la navegación y comprensión de la aplicación.*" [9].

"*Mantener un diseño consistente en todos los elementos de la interfaz, como botones, colores y tipografías, ayuda a crear una experiencia fluida y reduce la confusión del usuario. La consistencia también facilita el aprendizaje de la aplicación, ya que los usuarios se familiarizan rápidamente con la disposición y funcionalidad de los elementos.*" [8].

"*Permitir a los usuarios personalizar la interfaz, cómo ajustar el tamaño del texto, los colores y el contraste, no solo mejora la usabilidad sino que también hace la app más inclusiva. Esto es importante para adaptarse a diferentes necesidades visuales y preferencias personales, mejorando la experiencia de usuario.*" [8]

"Proveer retroalimentación clara y efectiva (visual, auditiva y háptica) y manejar los errores de manera comprensible es esencial para mantener a los usuarios informados sobre sus acciones y errores, mejorando así la confianza y satisfacción en el uso de la aplicación." [10]

4.1.1 Pantalla principal

Lo primero a destacar es que nuestra app no contará con un sistema de usuarios registrados. Cada usuario contará con su propia información de forma local. Dicho esto, la primera página que veremos al abrir la aplicación será la de las rutinas del día (Figura 4.1).

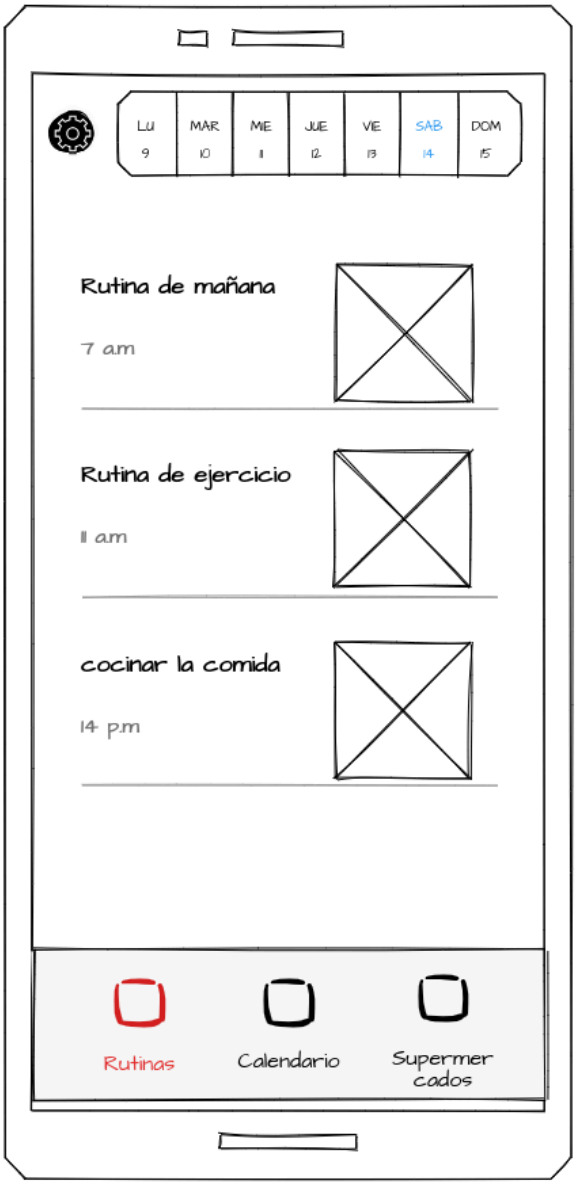


Figura 4.1 Pantalla de TimeSphere, rutinas

4.1.2 Calendario

En esta pantalla (Figura 4.2) podremos movernos entre días de manera más rápida y con mayor rango, pudiendo incluso planificar los meses posteriores. Al pulsar sobre un día volveríamos a la pantalla principal, cambiando el día al seleccionado.

Figura 4.2 Pantalla de TimeSphere, calendario

4.1.3 Diario

En esta pantalla (Figura 4.3) tenemos un calendario similar al del calendario normal. Si se pulsa en alguno de estos días pasaremos a la pantalla de edición del diario, en la que podremos establezer el estado de ánimo del día o la entrada en el diario.

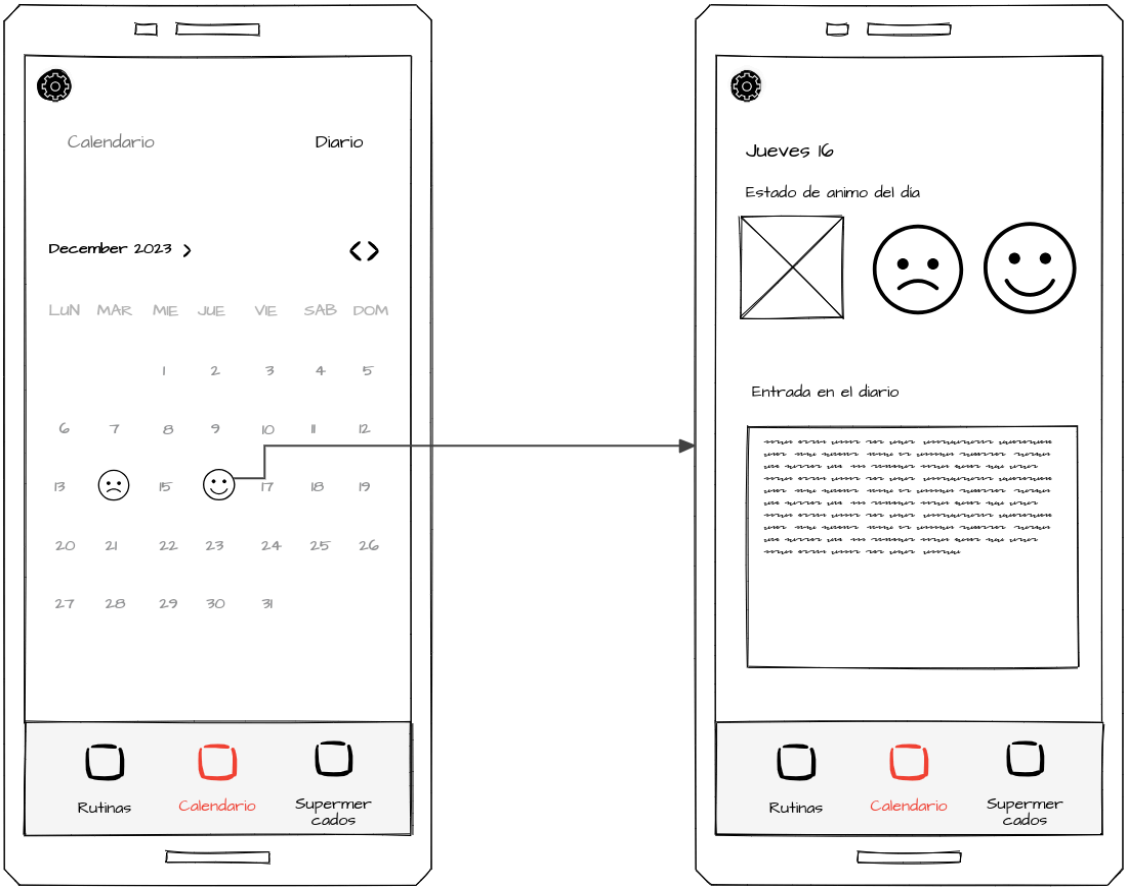


Figura 4.3 Pantalla de TimeSphere, Diario

4.1.4 Supermercados

En esta pantalla (Figura 4.4) nos encontramos con una lista de supermercados, en la que está indicado: el nombre del supermercado, la ubicación y las etiquetas de accesibilidad. Si pulsamos sobre el botón de filtrar podemos ver una lista de etiquetas que se pueden marcar o desmarcar para que los supermercados que nos aparezcan posteriormente contengan esas etiquetas.

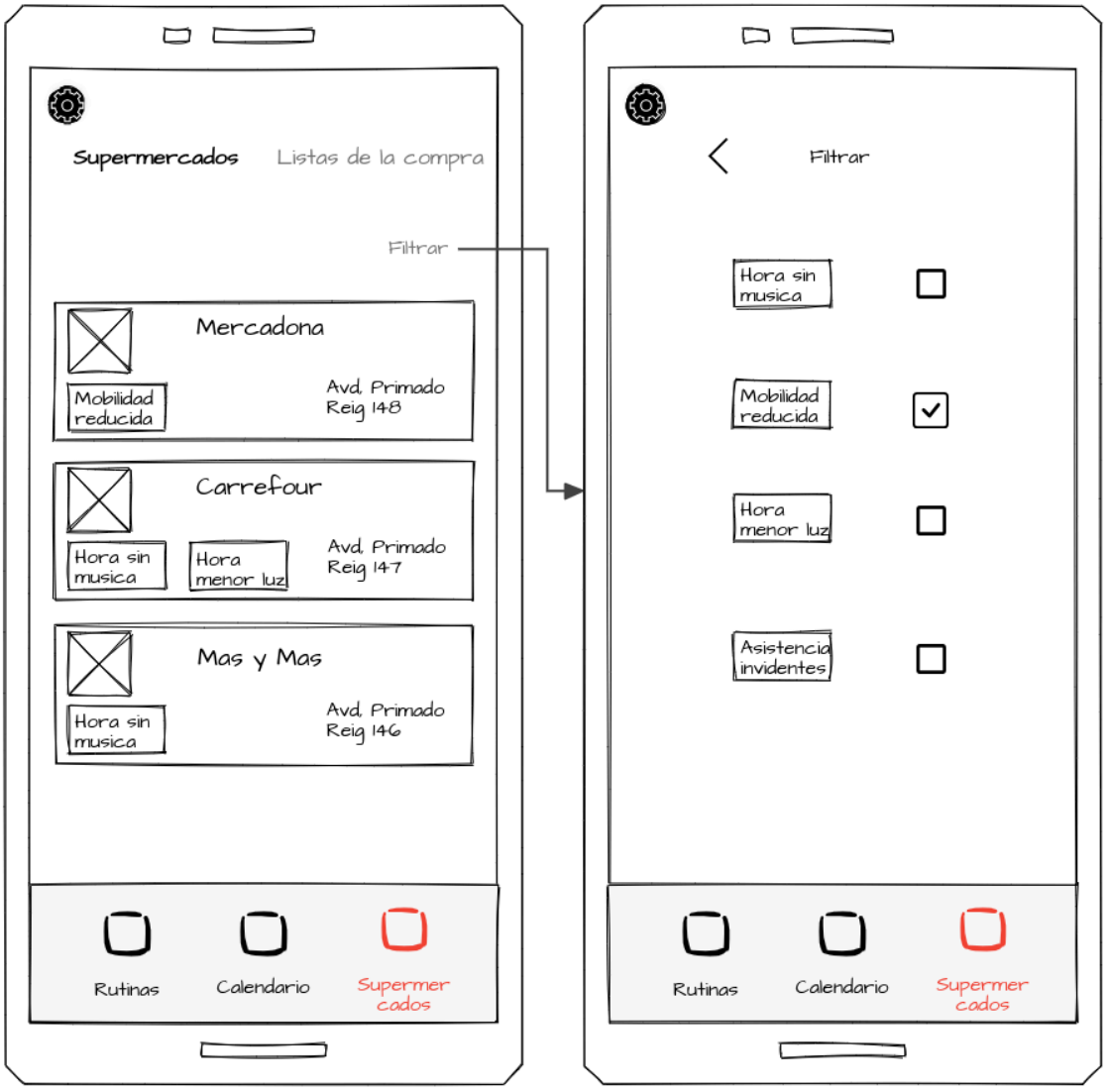


Figura 4.4 Pantalla de TimeSphere, Supermercados

4.1.5 Configuración

En cualquiera de las pantallas podemos encontrar un icono en el que podemos configurar la aplicación. Al pulsar sobre él llegaremos a la pantalla de configuración (Figura 4.5). En esta podremos observar diferentes opciones que pueden ser activadas y desactivadas para modificar el aspecto de la aplicación, el sonido o la vibración.

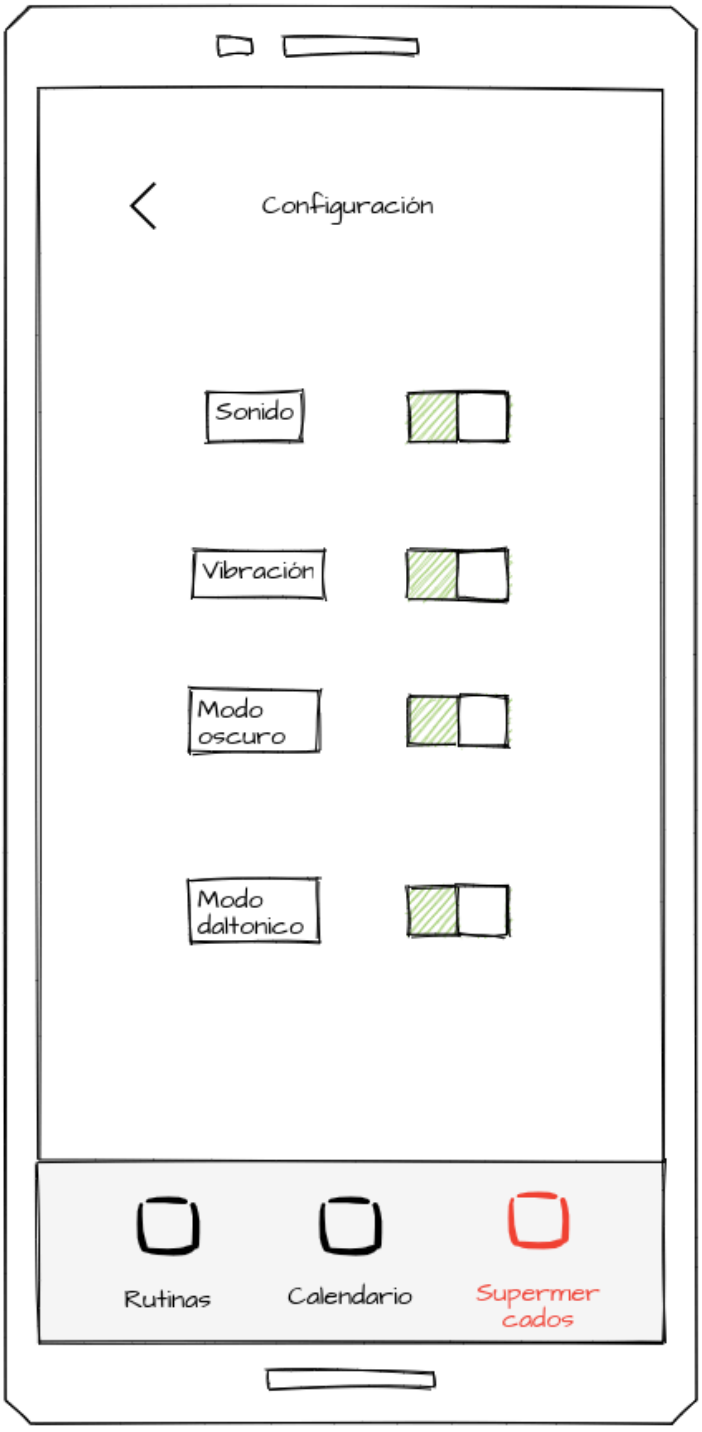


Figura 4.5 Pantalla de TimeSphere, configuración

4.1.6 Rutina activa

Al pulsar sobre una rutina podremos iniciarla. Al hacerlo pasaremos a una pantalla en la que las diferentes actividades de la rutina aparecerán secuencialmente. Cada actividad terminará cuando se termine su tiempo preestablecido, dando paso a la siguiente actividad (Figura 4.6).

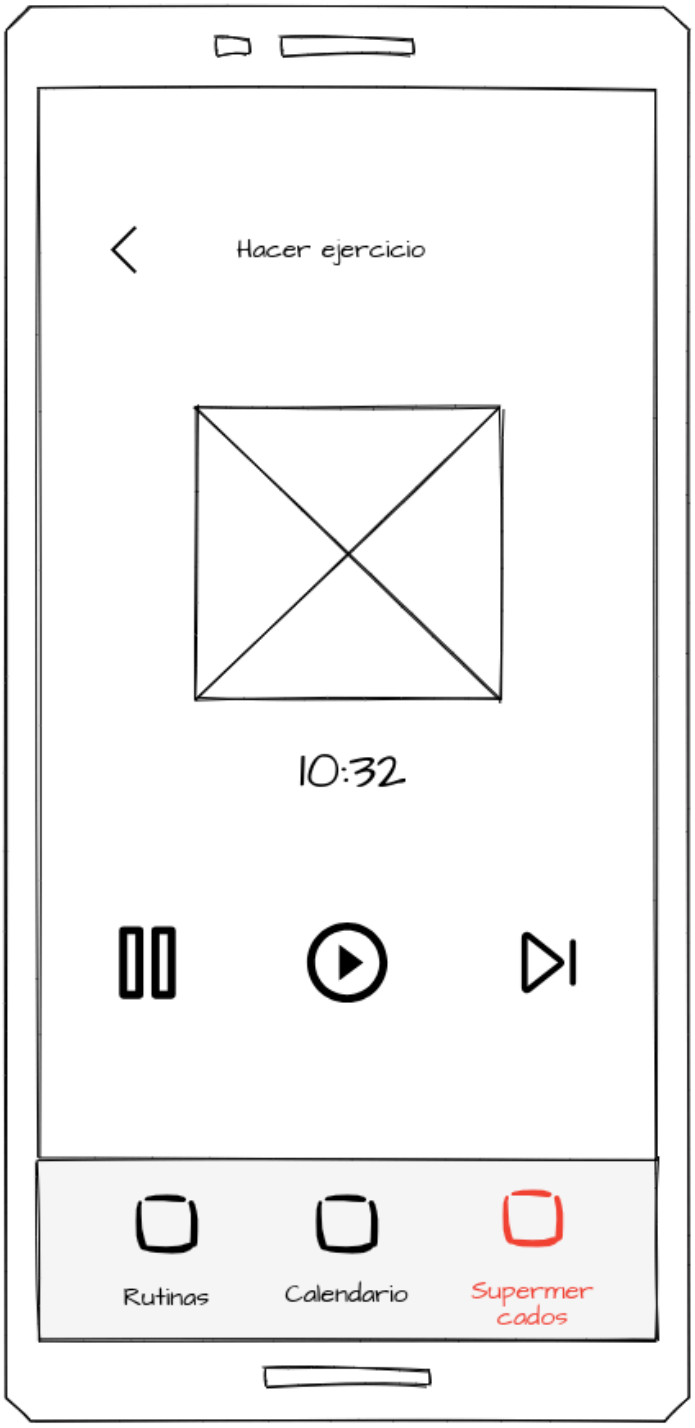


Figura 4.6 Pantalla de TimeSphere, Rutina activa

4.2. Arquitectura Modelo-Vista-Controlador

La elección de un modelo o patrón de arquitectura adecuado es crucial para el éxito de un proyecto, permitiendo que se alinee con los requisitos de coste, tiempo y el equipo disponible. Por ello hemos elegido el patrón Modelo-Vista-Controlador (MVC) [13], que es uno de los más populares debido a su capacidad para organizar el código de manera clara y separada, facilitando el desarrollo, mantenimiento y escalabilidad de las aplicaciones.

El patrón MVC divide una aplicación en tres componentes principales: Modelo, Vista y Controlador. Cada uno tiene responsabilidades claramente definidas, lo que permite una separación de las preocupaciones y facilita la colaboración entre los desarrolladores.

1. Modelo

El **Modelo** es la capa que maneja la lógica de la aplicación relacionada con los datos. Aquí se encuentra la representación de la información sobre la cual opera la aplicación, así como las reglas de negocio asociadas a esta. El Modelo es responsable de:

- **Gestionar los datos**: Almacena, recupera y manipula los datos de la aplicación.
- **Realizar cálculos**: Ejecuta operaciones y reglas de negocio que transforman los datos en información útil.
- **Interactuar con la base de datos**: Realiza operaciones CRUD (*Create, Read, Update, Delete*) en la base de datos y maneja la lógica de persistencia.

2. Vista

La **Vista** es la capa encargada de la presentación visual y de la interacción con el usuario. Es la interfaz que el usuario ve y con la que interactúa. Sus responsabilidades incluyen:

- **Renderizar la interfaz de usuario**: Presenta los datos proporcionados por el Modelo de manera visual.
- **Gestionar la interacción del usuario**: Captura eventos de usuario como clics y entradas de datos, y los envía al Controlador para su procesamiento.
- **Actualizar dinámicamente**: Responde a los cambios en el Modelo, refrescando la UI en consecuencia.

3. Controlador

El **Controlador** actúa como un intermediario entre el Modelo y la Vista. Su función principal es recibir entradas del usuario, procesarlas y coordinar las respuestas apropiadas de la aplicación. Las tareas del Controlador incluyen:

- **Interpretar las entradas del usuario**: Recoge eventos de la Vista, como comandos de usuario, y los traduce en acciones que el Modelo puede entender.
- **Actualizar el Modelo**: Modifica el estado del Modelo en función de las interacciones del usuario.
- **Actualizar la Vista**: Coordina las actualizaciones de la interfaz para reflejar los cambios en el Modelo.

4. Ventajas de la Arquitectura MVC

- **Separación de responsabilidades**: Cada componente del MVC tiene una función específica, lo que facilita el mantenimiento y escalabilidad del sistema.
- **Reutilización de código**: La lógica de negocio (Modelo) es independiente de la interfaz de usuario (Vista), permitiendo reutilizar componentes en diferentes contextos.
- **Facilidad para realizar pruebas**: Gracias a la separación de componentes, es más sencillo testear cada parte del sistema de manera aislada.

Este patrón es ideal para aplicaciones que requieren una clara división entre la lógica de negocio, la interfaz de usuario y el control de flujo, asegurando un código más limpio y modular que facilita tanto el desarrollo inicial como el mantenimiento a largo plazo.

4.3. Lenguaje de programación

Para satisfacer las necesidades de este proyecto, es fundamental seleccionar tanto un lenguaje de programación como un *framework* que permitan crear una aplicación móvil con una interfaz clara, intuitiva y que funcione sin problemas de rendimiento. Además, es esencial que la aplicación pueda ser utilizada en los dos sistemas operativos predominantes en el mundo de los móviles: *iOS* y *Android*. Por estas razones, hemos elegido utilizar *React Native* como *framework* de desarrollo y *JavaScript* como lenguaje de programación principal.

React Native[14] es un *framework* desarrollado por *Facebook* que permite crear aplicaciones móviles nativas utilizando *JavaScript*. Una de las principales ventajas de *React Native* es su capacidad para desarrollar una única base de código que se puede compilar tanto en *iOS* como en *Android*, lo que ahorra tiempo y recursos al evitar la necesidad de desarrollar y mantener dos aplicaciones separadas para cada plataforma.

Además, *React Native* permite el uso de componentes nativos, lo que significa que la aplicación puede aprovechar al máximo las capacidades de los dispositivos móviles, ofreciendo un rendimiento cercano al de las aplicaciones desarrolladas en lenguajes nativos como *Swift* para *iOS* o *Kotlin* para *Android*. Esto asegura una experiencia de usuario fluida y optimizada.

Otra característica destacada de *React Native* es su vasta comunidad de desarrolladores y su amplio ecosistema de bibliotecas y herramientas, lo que facilita la implementación de

funcionalidades adicionales y la resolución de problemas técnicos. Asimismo, su integración con *React* [15], una biblioteca popular de *JavaScript* para construir interfaces de usuario, permite un desarrollo ágil y modular, lo cual es ideal para mantener y escalar la aplicación en el futuro.

JavaScript [16] ha sido elegido como el lenguaje de programación para este proyecto debido a su versatilidad y universalidad, que lo convierten en una opción robusta para el desarrollo de aplicaciones móviles. Es un lenguaje ampliamente soportado en diferentes entornos, desde navegadores web hasta servidores y, gracias a *frameworks* como *React Native*, también en aplicaciones móviles. Esta universalidad permite que el equipo de desarrollo pueda trabajar con un solo lenguaje en diversas áreas del proyecto, facilitando la integración y la colaboración.

Además, *JavaScript* permite un desarrollo ágil y eficiente. Su naturaleza interpretada, junto con la capacidad de trabajar en tiempo real, permite a los desarrolladores ver los cambios de manera inmediata. Esto acelera significativamente los ciclos de desarrollo y prueba, permitiendo iterar rápidamente en el diseño y la funcionalidad de la aplicación.

Finalmente, *JavaScript* cuenta con una de las comunidades más grandes y activas del mundo de la programación. Esto significa que existen abundantes recursos, bibliotecas y herramientas disponibles para resolver prácticamente cualquier desafío técnico que pueda surgir durante el desarrollo. La disponibilidad de una vasta cantidad de documentación y tutoriales facilita la resolución de problemas y contribuye a un desarrollo más fluido y respaldado por las mejores prácticas de la industria.

En resumen, la combinación de *React Native* y *JavaScript* ofrece una solución potente y flexible para desarrollar aplicaciones móviles modernas, eficientes y capaces de funcionar en múltiples plataformas sin sacrificar rendimiento ni calidad.

5. Desarrollo de la solución

5.1. Creación de la base de datos

Como se mencionó en el capítulo 3, nuestra base de datos se implementará usando *Firebase* [17], debido a que es una de las plataformas más robustas y fiables para el desarrollo de aplicaciones móviles y web. Además cuenta con una integración nativa con *React Native*, lo que facilita la implementación de funciones clave como la autenticación, el almacenamiento en tiempo real y la sincronización de datos en la nube.

Por otro lado, *Firebase* es altamente escalable y se adapta a las necesidades del proyecto sin necesidad de gestionar la infraestructura subyacente.

La tecnología concreta de *Firebase* que vamos a utilizar es *Cloud Firestore* [18], una base de datos *NoSQL* que organiza los datos en documentos y colecciones, permitiendo un acceso rápido y eficiente a la información.

Una de las principales ventajas de *Cloud Firestore* es su capacidad para manejar datos de manera escalable y en tiempo real. Esto significa que cualquier cambio realizado en la base de datos se refleja instantáneamente en todas las instancias de la aplicación, ofreciendo una experiencia de usuario fluida y consistente.

Cloud Firestore organiza los datos en una estructura jerárquica de documentos y colecciones. Los documentos son las unidades fundamentales de almacenamiento y contienen pares de clave-valor, que pueden ser de varios tipos de datos, como cadenas de texto, números, mapas, listas y referencias a otros documentos. Cada documento se identifica de forma única dentro de una colección.

Las colecciones son grupos de documentos que comparten una estructura y propósito común. Por ejemplo, en una aplicación de gestión de tareas, podríamos tener una colección llamada "tareas", donde cada documento representa una tarea individual con atributos como el nombre, la descripción y el estado de finalización.

En la Figura 5.1 se muestra nuestra base de datos con sus documentos y colecciones. Esta base de datos almacenará todos los datos de las diferentes entidades creadas en nuestra aplicación en tiempo real. Por ejemplo, al crear una rutina, esta se guardará automáticamente en la base de datos, generando un documento para el día si es necesario. La próxima vez que recuperemos información, podremos encontrar la nueva rutina almacenada.

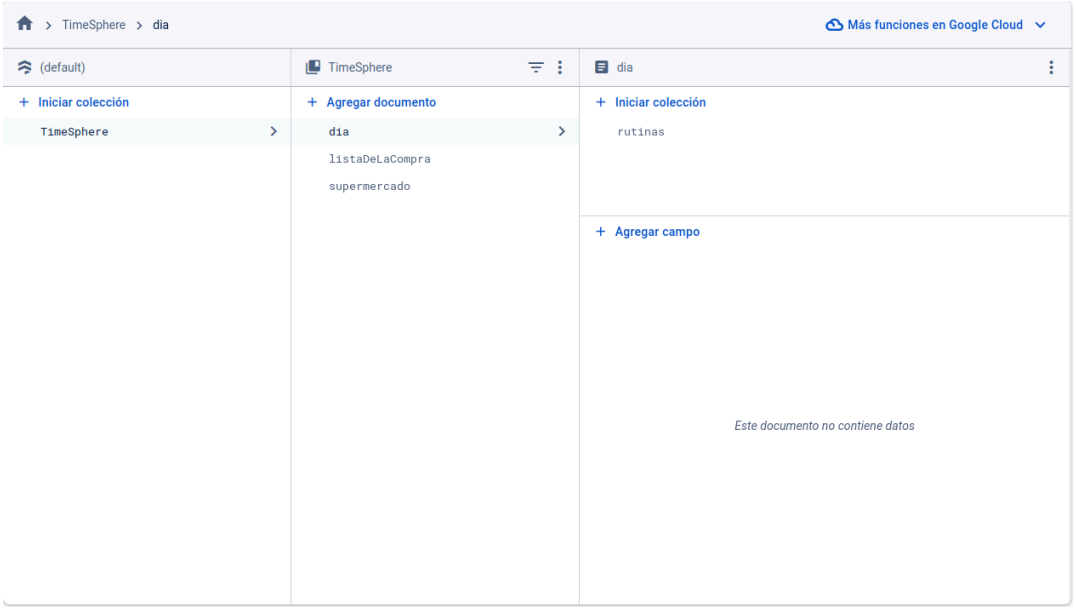


Figura 5.1 Imagen de la BD de TimeSphere

5.2. Creación del repositorio del proyecto

El siguiente paso para desarrollar nuestra aplicación es la creación del repositorio donde trabajaremos en el proyecto, para lo cual utilizaremos *Expo* [19]. *Expo* es una plataforma que simplifica el desarrollo con *React Native*, ofreciendo un entorno de desarrollo con bibliotecas y *APIs* preconfiguradas. Esto permite a los desarrolladores construir aplicaciones móviles sin preocuparse por la configuración nativa específica de *iOS* o *Android*.

Para crear el proyecto con *Expo*, debemos ejecutar el comando “npx create-expo-app@latest”, que generará una estructura de carpetas como la que se muestra en la Figura 5.2.

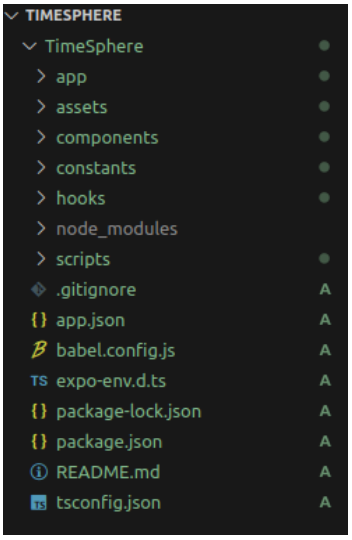


Figura 5.2 Estructura inicial de carpetas

Para poder almacenar nuestros continuos desarrollos en un lugar seguro y accesible desde diferentes dispositivos, vamos a utilizar *git* [20], con el que crearemos un repositorio en la

nube y gestionaremos las versiones de nuestro proyecto. La información del uso de git ha sido recopilada en parte del libro *Pro Git* [21]

Para ello hemos creado un repositorio remoto en *Github* [22] y hemos hecho push con nuestro primer commit, que contiene la creación inicial del proyecto con *Expo* (Figura 5.3).

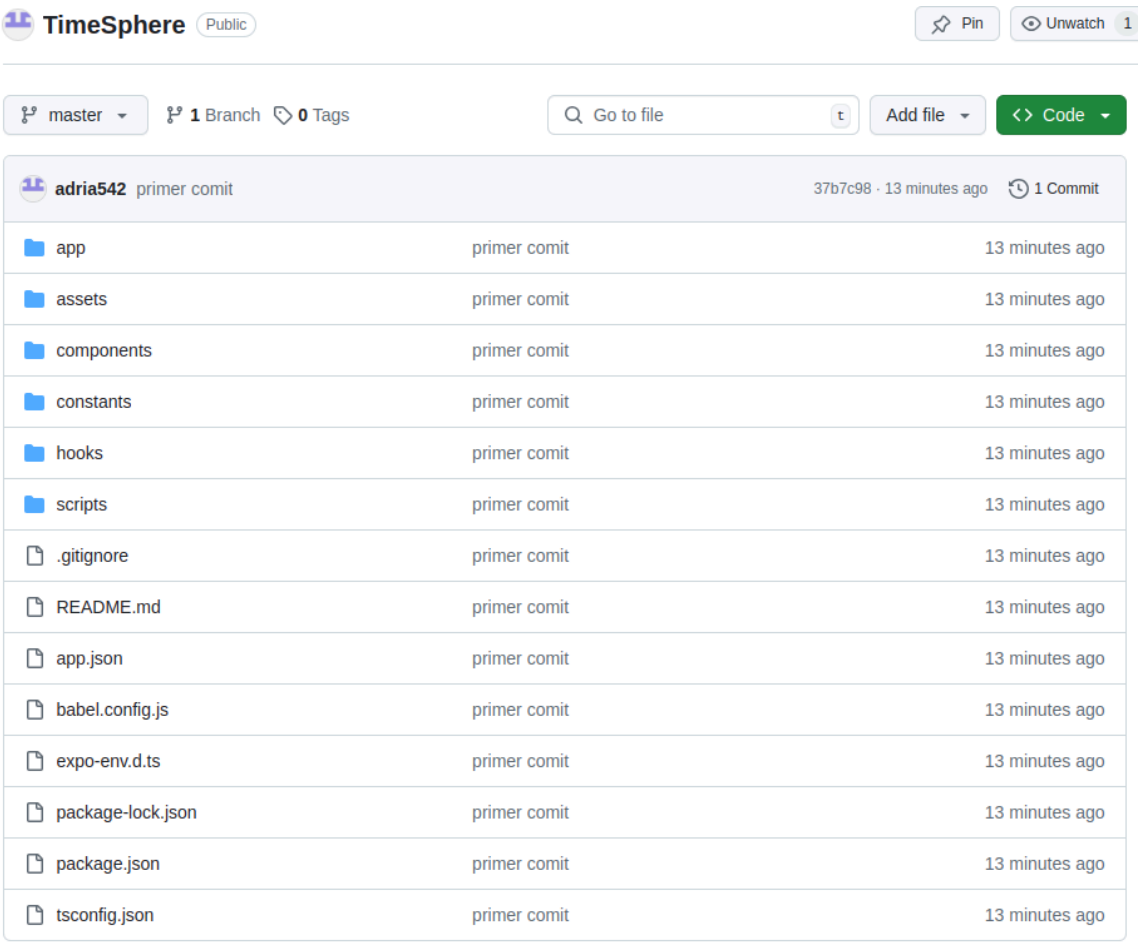


Figura 5.3 Primer commit

Una vez que hayamos creado el repositorio del proyecto, el siguiente paso será ajustar la estructura de las carpetas para que se adapte correctamente a la organización propia de una aplicación desarrollada utilizando la arquitectura modelo-vista-controlador (MVC) (Figura 5.4). Esta reorganización es esencial para mantener un código ordenado y modular, facilitando así la separación de responsabilidades dentro del proyecto. La estructura resultante permitirá gestionar de manera más eficiente los componentes del modelo, la vista y el controlador.

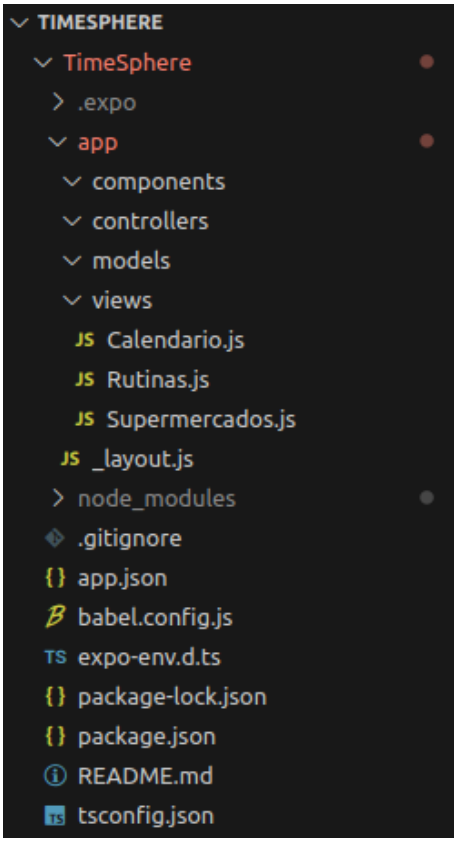


Figura 5.4 Estructura de carpetas para el MVC

/.expo: La carpeta .expo almacena datos de configuración y caché específicos del entorno de desarrollo de Expo.

/app: Contendrá la mayor parte del código fuente de la aplicación.

- **/components**: contendrá componentes reutilizables que pueden ser utilizados en diferentes vistas o controladores.
- **/controllers**: Contendrá la lógica de controladores que maneja la interacción entre el modelo y la vista.
- **/models**: Contendrá los modelos de datos y la lógica relacionada con los datos.
- **/views**: Contendrá las interfaces con las que el usuario interactuará.
- **_layout.js**: este archivo contiene el código de la pantalla de navegación de la app y es la que maneja el movimiento entre las diferentes interfaces en /views.

/node_modules: Contiene todas las dependencias y paquetes de JavaScript instalados para un proyecto Node.js, incluidos los de React Native y Expo.

app.json: contiene la configuración del proyecto, incluyendo metadatos como el nombre, icono, versión y configuraciones.

5.3. Creación del backlog

Una vez tenemos preparado nuestro repositorio, podríamos empezar a desarrollar directamente nuestra app sin un plan organizado, pero gracias a nuestro conocimiento en desarrollo ágil sabemos que esta decisión sería errónea. Para poder desarrollar una app de manera ágil tendremos que utilizar una herramienta en la que podamos organizar todo el trabajo pendiente. Para ello usaremos *worki* [23], una herramienta que ya hemos usado en el pasado y con la que podremos organizar adecuadamente todo nuestro trabajo en un backlog.

Nuestro proyecto se desarrollará en tres sprints diferenciados. En el primero se desarrollará la vista, en el segundo el modelo y en el último el controlador

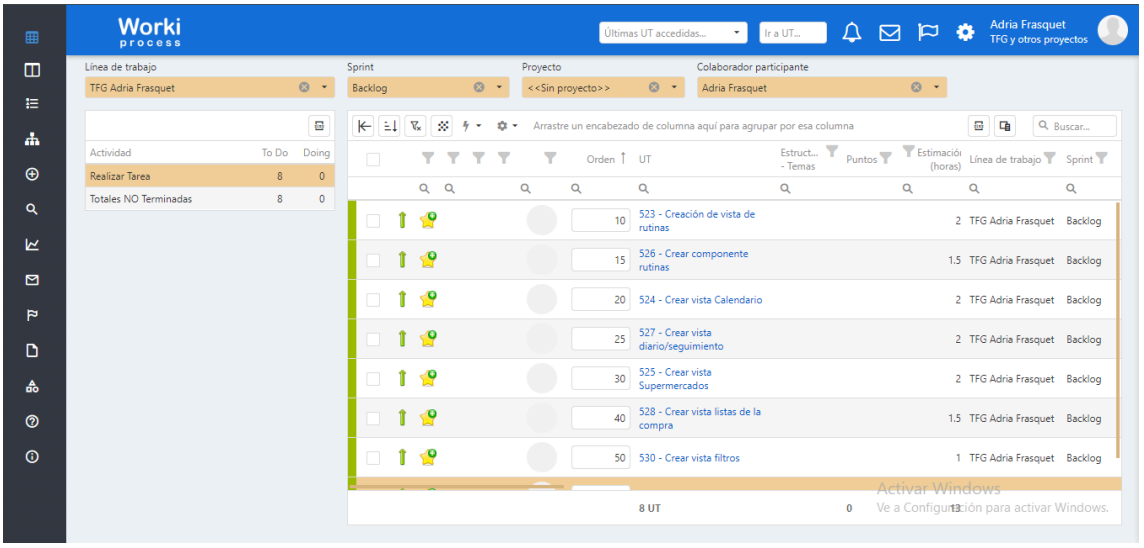


Figura 5.5 worki

Como podemos ver en la Figura 5.5 hemos organizado el trabajo del primer sprint en UTs (unidad de trabajo) ordenadas por prioridad y con una estimación de su duración en horas de programación ideales.

5.4. Implementación de la vista

5.4.1 Expo Router

Una vez organizado el trabajo, podemos empezar a desarrollar nuestro proyecto, empezando por la vista. Para desarrollar la vista utilizaremos una herramienta propia de Expo: "*Expo Router*"[24]. Esta herramienta convierte cada archivo *.js .tsx* o *.jsx* que se encuentre en la carpeta *app* (Figura 5.6), en una ruta que puede ser mostrada si se codifica en el archivo

index.js. Este archivo es el gestor de navegación. Gracias a él, nuestra app contará con la barra de navegación inferior que podemos observar en los mockups descritos en capítulo anterior.

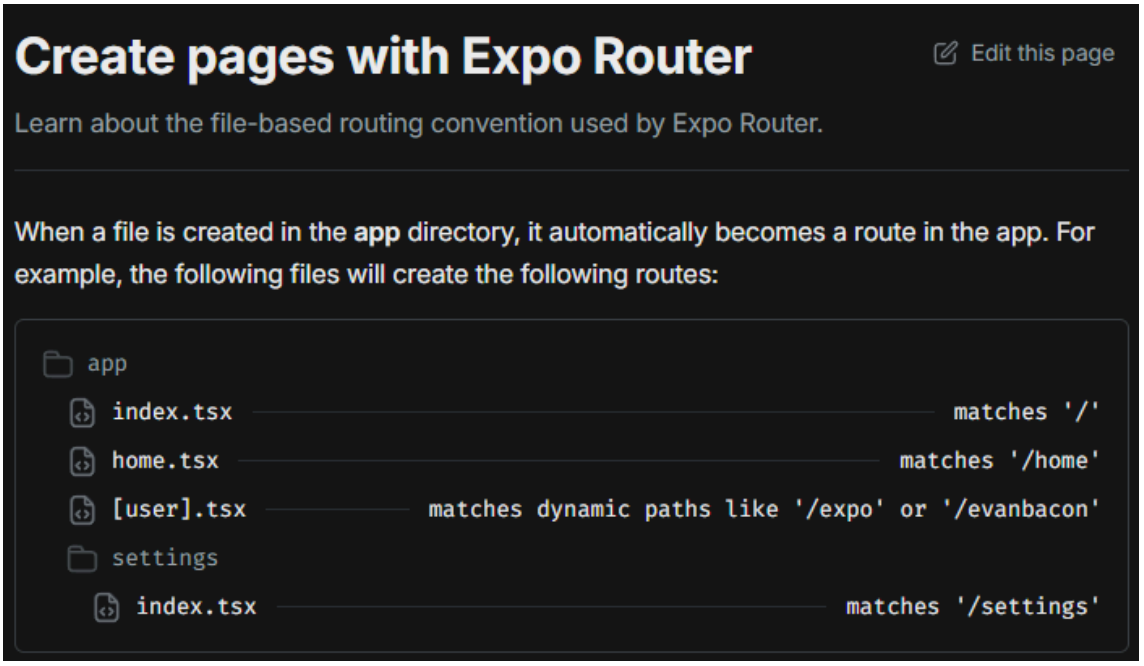


Figura 5.6 Expo Router

5.4.2 React Navigation

Dado que nuestra aplicación requerirá más pantallas además de las tres principales que estarán presentes en la barra de navegación inferior, necesitaremos una herramienta eficaz que nos permita navegar de manera fluida entre diversas interfaces. Para lograr esto, utilizaremos una de las bibliotecas más populares en el desarrollo con React Native: React Navigation [25]. Esta biblioteca nos ofrece la flexibilidad necesaria para crear un flujo de navegación personalizado, permitiendo a los usuarios moverse entre las distintas pantallas de la aplicación de manera intuitiva y coherente, mejorando así la experiencia de usuario en su totalidad. Con React Navigation, podremos configurar y gestionar de forma eficiente la transición entre las diferentes vistas de nuestra aplicación, asegurando que la navegación sea tanto funcional como agradable.

5.4.4 Pantallas

Todas las pantallas que conforman nuestro flujo de navegación se encuentran en la carpeta views de nuestro repositorio (Figura 5.8). En este apartado vamos a explicar brevemente cada una de estas pantallas.

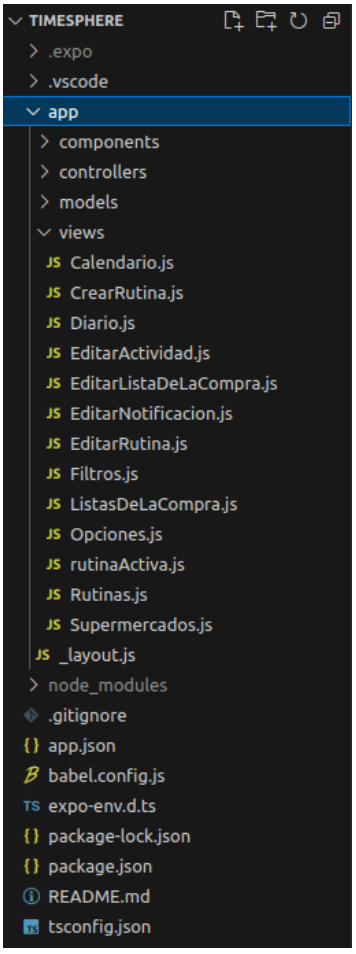


Figura 5.8 Pantallas TimeSphere

- Calendario.js: Pantalla en la que podemos ver el calendario, desde donde se puede acceder a las rutinas o al diario de diferentes días.
- CrearRutina.js: Pantalla en la que podemos crear rutinas, modificando su nombre, hora de inicio o activando/desactivando sus notificaciones.
- Diario.js: Pantalla en la que podemos establecer el estado de ánimo o escribir entradas en el diario del día.
- EditarActividades.js: Pantalla en la que podemos modificar las actividades, cambiando su imagen, título, duración o tipo.
- EditarListasDeLaCompra.js: Pantalla que muestra una lista de artículos seleccionables para la creación de una lista de la compra.
- EditarNotificaciones.js: Pantalla en la que podemos modificar las rutinas, cambiando su texto o activando/desactivando el sonido o la vibración.
- EditarRutinas.js: Pantalla que muestra las actividades que conforman una rutina; aquí podemos añadir o eliminar actividades o iniciar la rutina.
- Filtros.js: Pantalla en la que podemos filtrar los supermercados según sus sistemas de accesibilidad.
- ListasDeLaCompra.js: Pantalla en la que podemos ver una lista de la compra que hemos creado anteriormente. Además, podemos pulsar sobre el botón editar para pasar a la pantalla de EditarListasDeLaCompra.js.

- Opciones.js: Pantalla en la que podemos modificar las opciones de la aplicación, como activar/desactivar el modo oscuro, el sonido o la vibración.
- RutinaActiva.js: Pantalla que muestra la ejecución de una rutina, incluyendo la imagen de la actividad junto con su nombre y tiempo restante. Además, presenta botones para pausar, reanudar o saltar la actividad.
- Rutinas.js: Pantalla inicial de la app, donde se encuentra una barra superior para seleccionar un día de la semana y ver las rutinas correspondientes. Incluye una lista de rutinas del día y un botón para añadir más rutinas.
- Supermercados.js: Pantalla que muestra una lista de supermercados ordenados por distancia al usuario, incluyendo su nombre, ubicación y sistemas de accesibilidad. También dispone de un botón para acceder a la pantalla de filtros y otro para cambiar a la lista de listas de la compra.
- _layout.js: Este archivo no es una pantalla como tal, pero es importante mencionar que este código gestiona la navegación entre las diferentes pantallas mencionadas.

5.4.5 componentes

En esta carpeta encontramos los diferentes archivos de código que definen diferentes componentes que se encuentran en diferentes lugares dentro de las interfaces (Figura 5.9). En esta sección vamos a explicar brevemente de qué trata cada uno.

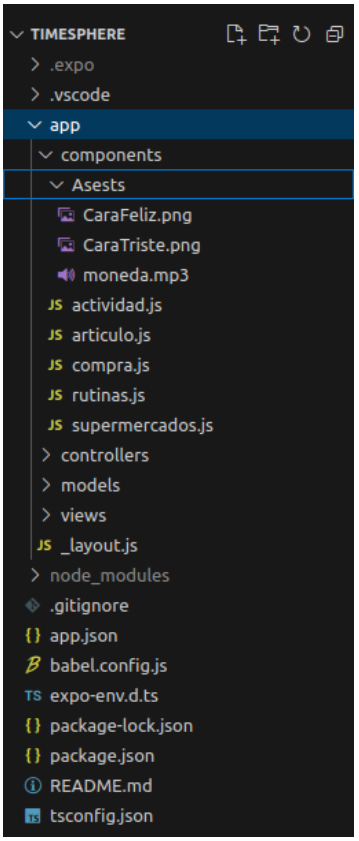


Figura 5.9 Pantallas TimeSphere

- /Assets: En esta carpeta se encuentran los diferentes recursos utilizados en nuestra app, como imágenes y archivos de audio.
- actividad.js: Este componente nos permite visualizar la lista de actividades que conforman una rutina. Estas actividades incluyen una imagen que describe la actividad, un título y un subtítulo que indica la duración de la actividad.
- articulo.js: Este componente nos permite visualizar los diferentes artículos que conforman una lista de la compra. Cada artículo incluye una imagen, su nombre y un checkbox que indica si el artículo ha sido recogido o no.
- compra.js: Este es un componente simple que nos permite visualizar las listas de la compra creadas por el usuario, mostrando únicamente el título de cada lista.
- rutina.js: Este componente nos permite visualizar las diferentes rutinas que el usuario ha creado para un día en particular. Cada rutina incluye una imagen, un título y un subtítulo que indica la hora de inicio de la actividad.
- supermercados.js: Este componente nos permite visualizar la lista de supermercados. Cada supermercado se compone de su nombre, ubicación y sistemas de accesibilidad.

5.5. Implementación del controlador

Para que la vista sea útil, los componentes con los que interactúa el usuario deben modificar el estado de la aplicación. Aquí es donde entra en juego el controlador, que actualiza el modelo y la vista según los diferentes inputs del usuario. Algunas de estas modificaciones deben afectar el estado global de la aplicación. Para hacer esto de manera accesible desde cualquier interfaz, utilizaremos *Context* [26], una herramienta de React .

5.5.1 Context

Context en React es una herramienta que nos permite compartir valores y funciones entre componentes a través del árbol de componentes, sin necesidad de pasar *props* [27] manualmente en cada nivel. Con *Context*, podemos crear un "*Provider*" que envuelva una parte de nuestro árbol de componentes, proporcionando información y funciones que puedan ser accedidas por cualquier componente dentro de ese árbol.

Por ejemplo, podríamos tener un contexto llamado *ThemeContext* que almacena información sobre el tema actual de la aplicación (como el modo claro u oscuro). Esto permitiría que cualquier componente de la aplicación acceda al tema y ajuste su apariencia en consecuencia, sin necesidad de propagar explícitamente esta información a través de cada componente individual. Así, la implementación de un modo oscuro se vuelve más sencilla, ya que los componentes pueden reaccionar automáticamente a los cambios en el tema a medida que navegas entre diferentes interfaces.

En nuestro caso hemos implementado tres context, un “*ThemeProvider*”, un “*DayProvider*” y un “*ShortSpanProvider*”. Los tres se encuentran codificados en “*controladorContexto.js*”.

ThemeProvider se compone por “*isDarkMode*” que es una variable booleana que nos indica si la aplicación se encuentra o no en modo oscuro. Y “*setIsDarkMode*” que es una función que nos permite cambiar el estado de *isDarkMode*, cambiando el tema de la aplicación.

DayProvider se compone por "*selectedDay*" que representa el día en el que nos encontramos y por "*setSelectedDay*" que nos permite modificar el día en el que nos encontramos, este contexto nos permite acceder al diario o a las rutinas de diferentes días.

ShortSpanProvider se compone por varias variables y *setters* de estas variables que necesitan accederse desde varias pantallas pero no constituyen la necesidad de un *provider* propio y global. El nombre de este provider indica que la mayoría de estas variables necesitan aportar información durante poco tiempo y dejan de ser necesarias.

Cada uno de estos tres contextos envuelve el árbol de navegación completamente. Permitiéndonos acceder en cualquier punto de nuestra aplicación a estas variables/funciones.

5.5.1 Controllers

Además de modificar el estado global, las diferentes vistas deben interactuar con el controlador para acceder a variables y métodos que les permitan cumplir con sus funciones. Para ello, hemos creado un *controller* para cada una de las interfaces (Figura 5.10). Estos *controllers* gestionan la navegación, las funciones *OnPress* [28] (aportan la funcionalidad a los botones) y los *UseEffect* [29] (actualizan las interfaces en su creación y los momentos que sea necesario) entre otras funciones.

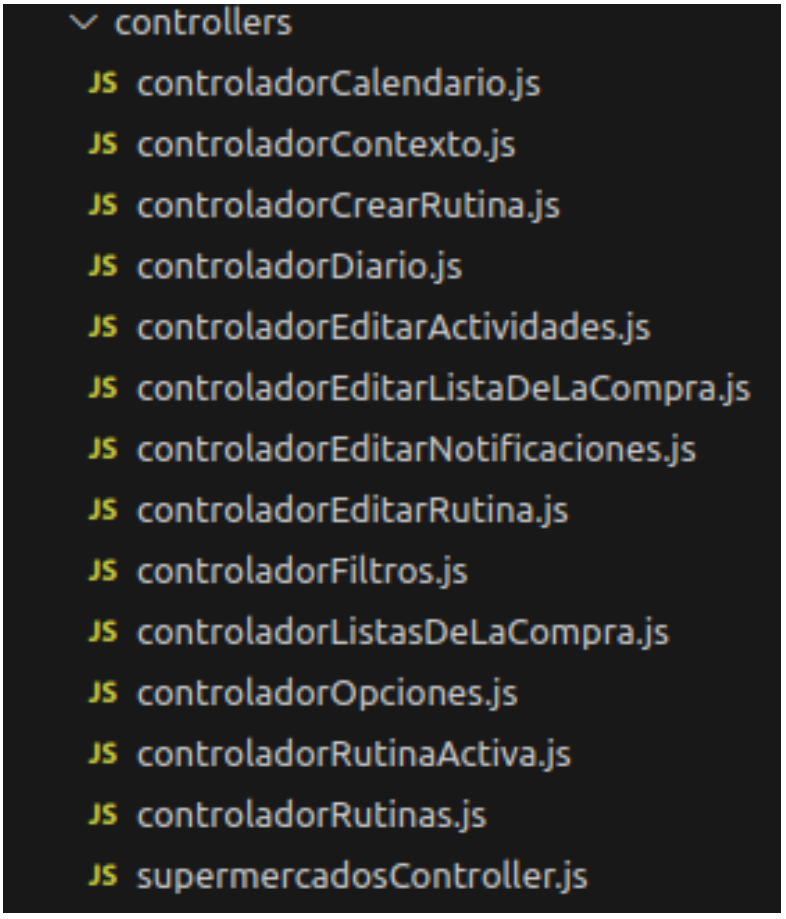


Figura 5.10 controladores TimeSphere

5.6. Implementación del modelo

Como se ha explicado anteriormente, el modelo es responsable de gestionar los datos de la aplicación. Para ello, hemos implementado siete clases (Figura 5.11) que definen las siete entidades que almacenan los diferentes datos que nuestra aplicación necesita gestionar. Estas clases cuentan con un constructor que especifica los atributos de cada uno de estos objetos, así como con funciones que permiten el guardado y la recuperación de estos objetos en la base de datos, además de otras funciones específicas de cada uno de ellos. En este apartado, vamos a comentar estas siete clases, destacando las funciones más importantes de cada una.

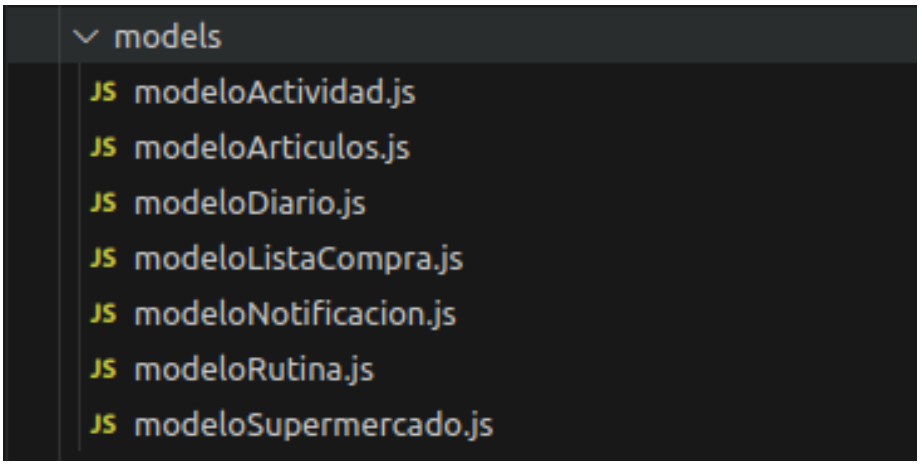


Figura 5.11 modelos TimeSphere

- modeloActividad.js: Esta clase define únicamente el objeto *Actividad*, que será necesario para definir el array de actividades de las rutinas.
- modeloArticulos.js: Esta clase define el objeto *Articulo*, así como dos funciones que permiten recuperar todas las actividades, o recuperar una en concreto.
- modeloDiario.js: Esta clase define el objeto *Diario*, además de dos funciones que permiten guardar y recuperar un diario en concreto en la bd.
- modeloListaCompra.js: Esta clase define el objeto *ListaCompra*, así como tres funciones que permiten la recuperación de todas las listas, de una en concreto o guardar una lista en la bd
- modeloNotificacion.js: Esta clase define el objeto *Notificación* y una función para modificar su atributo *activa*, que indica si la notificación está activa o no, este objeto se guarda como un campo de rutinas.
- modeloRutina.js: Esta clase define el objeto *Rutina*, además de una función que permite recuperar todas las rutinas de un día determinado y una que permite recuperar una rutina en concreto. Dos funciones que convierten actividades y notificaciones en campos de *Rutina*, y por último, una función que permite guardar una rutina en la base de datos.
- modeloSupermercado.js: Esta clase define el objeto *Supermercado*, junto con una función para recuperar todos los supermercados y otra que ordena la lista de supermercados por distancia al usuario utilizando otra función que calcula la distancia entre dos geoPoints.

5.7. Resultado final

En este apartado, presentamos el resultado final del desarrollo de la solución, mostrando el conjunto de pantallas que conforman un flujo de uso típico de nuestra aplicación, tanto en su versión clara como en su versión oscura. Las imágenes son capturas de pantalla tomadas desde el simulador móvil de *Expo*.

La primera pantalla que aparece al abrir la aplicación es la pantalla de rutinas (Figura 5.12), donde se pueden ver las rutinas del día. Además, es posible cambiar de día utilizando la barra superior para visualizar las rutinas de otros días de la semana. Si seleccionamos alguna rutina, accedemos a la pantalla de edición de rutina. Por último, en la parte inferior, hay un botón que permite añadir una nueva rutina. La pantalla de rutinas forma parte del *Tab.Navigator*, que permite moverse entre las tres pantallas principales de la aplicación, siendo esta la primera de ellas.

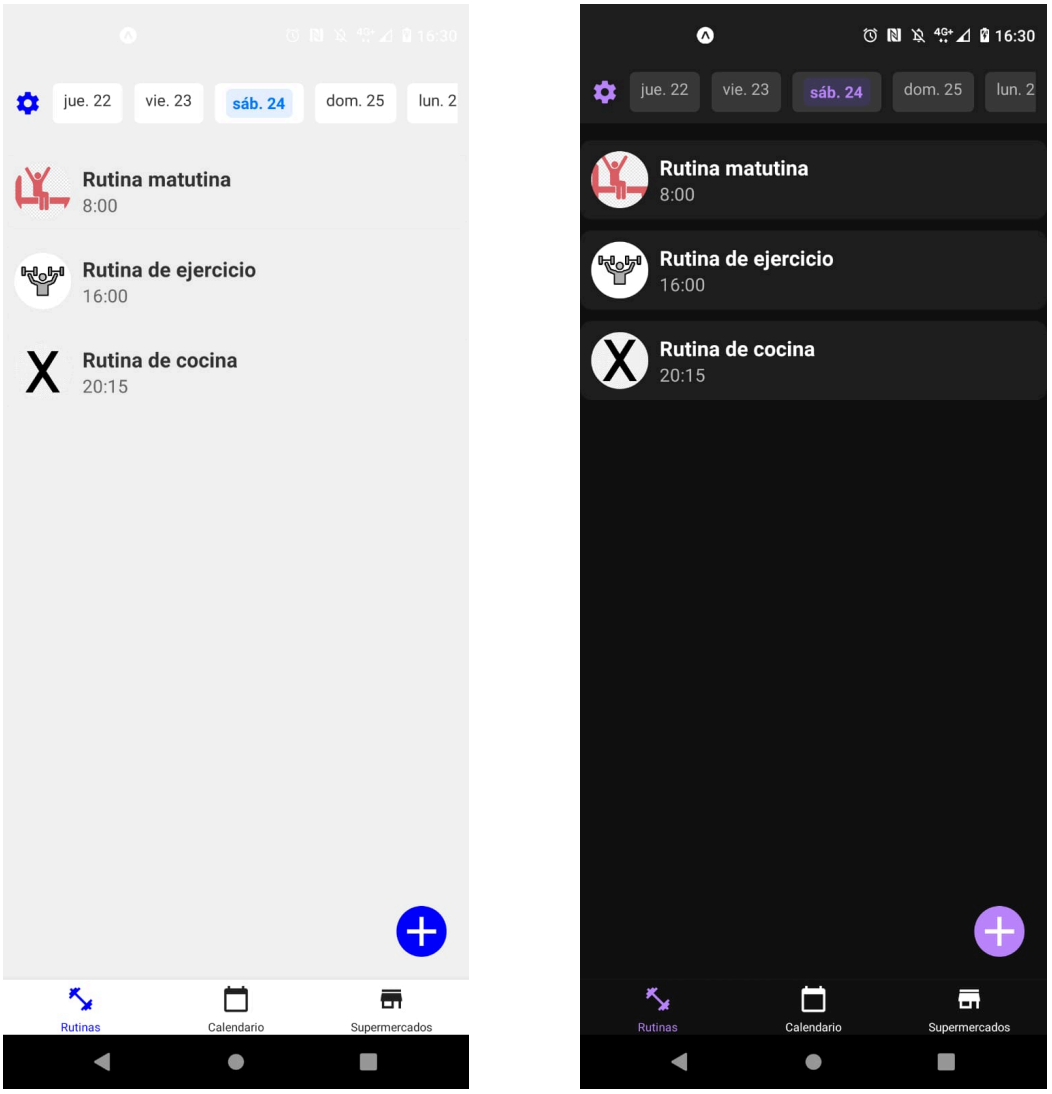


Figura 5.12 Pantalla final de rutinas, TimeSphere

La siguiente pantalla que encontramos en el navegador inferior es la pantalla de calendario (Figura 5.13). Esta pantalla cuenta con dos botones superiores que permiten alternar entre las funciones de calendario y las de diario. Ambas funciones comparten el mismo calendario, el cual muestra todos los días del mes, destacando el día actual. Este calendario permite acceder a las rutinas, así como a los estados de ánimo y entradas del diario de días en distintos meses.

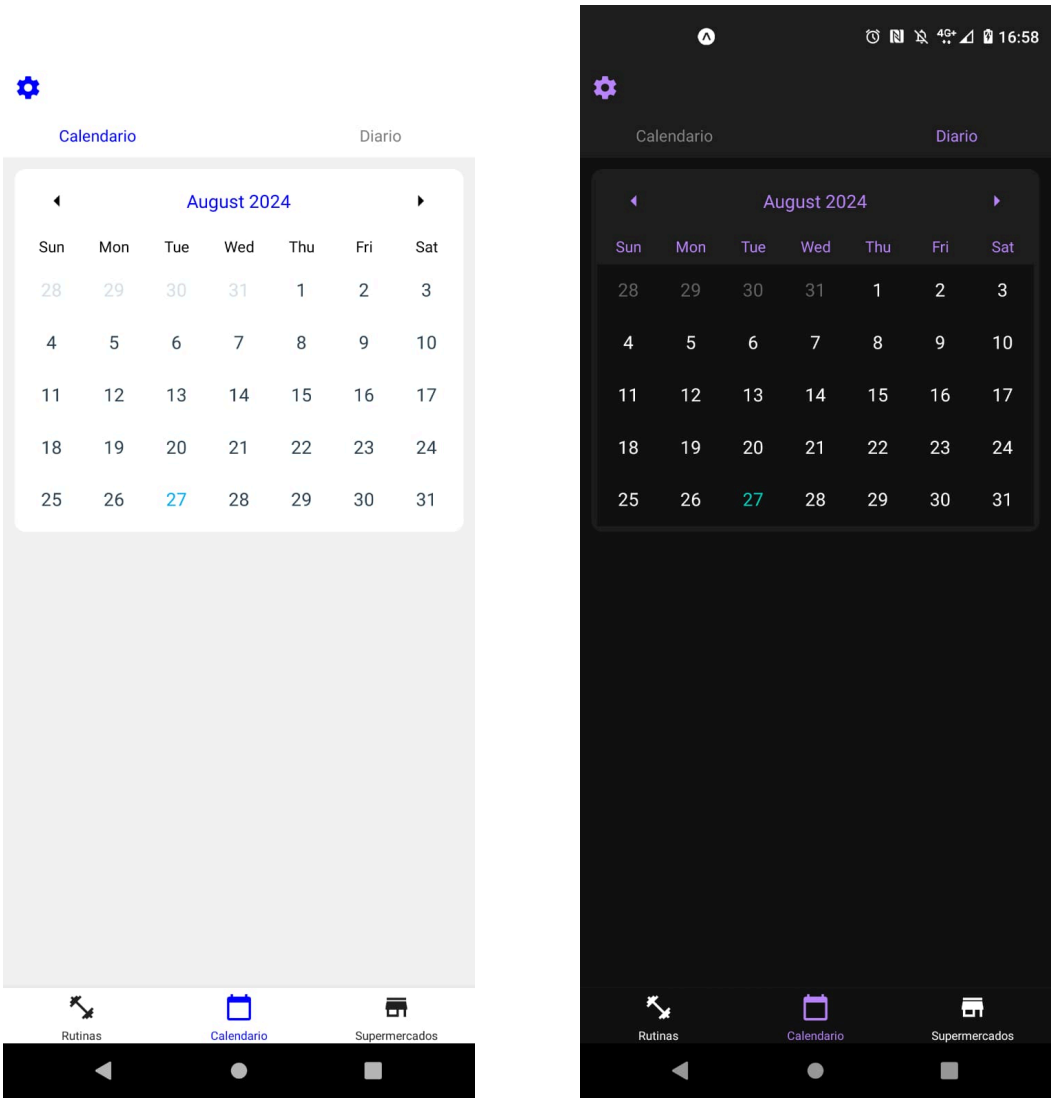


Figura 5.13 Pantalla final de Calendario, TimeSphere

La última pantalla que encontramos en el navegador inferior es la pantalla de Supermercados (Figura 5.14). En esta, podemos ver unos botones superiores similares a los de la pantalla anterior, que permiten alternar entre la lista de supermercados cercanos y las listas de la compra. Además, en la pantalla de Supermercados, hay un botón llamado "Filtros" que permite acceder a la pantalla de filtros. Por otro lado, en la pantalla de Listas de la compra, podemos pulsar sobre alguna de las listas de la compra o sobre el botón inferior para acceder a la pantalla de editar lista de la compra.

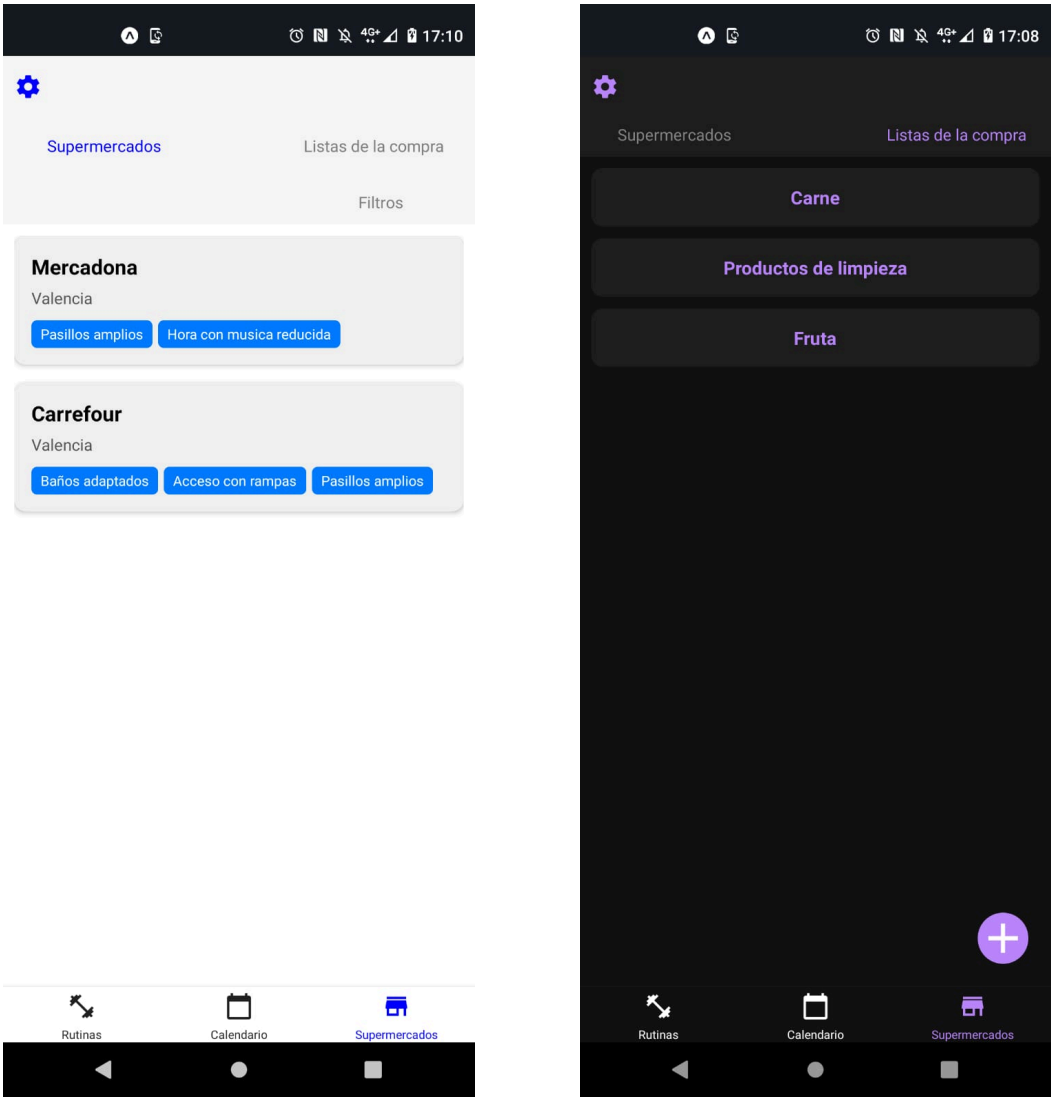


Figura 5.14 Pantalla final de Supermercados, TimeSphere

En todas las pantallas principales mencionadas anteriormente, hemos podido observar un botón superior con forma de tuerca. Este botón permite acceder a la pantalla de opciones (Figura 5.15), donde se pueden modificar configuraciones como el sonido, la vibración o el modo oscuro.

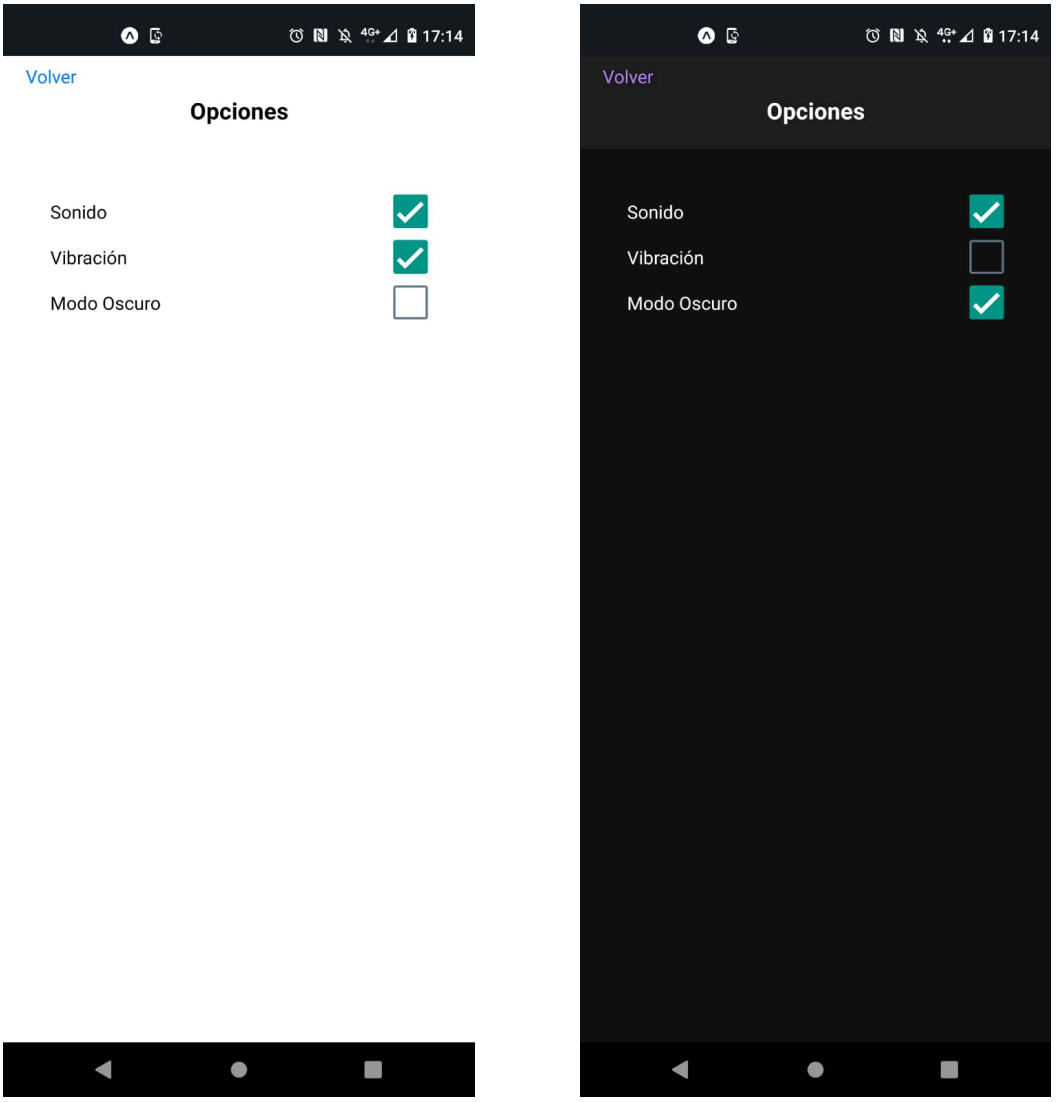


Figura 5.15 Pantalla final de Opciones, TimeSphere

Una vez revisadas todas las pantallas contenidas en el navegador inferior, nos disponemos a exponer las pantallas que se encuentran dentro de ellas. La primera que vamos a ver es la pantalla de crear rutina (Figura 5.16). En esta pantalla es donde se crean las rutinas para el día en el que nos encontramos en la pantalla de rutinas. Como primer elemento, encontramos un botón superior con la etiqueta “Volver”, que sirve para regresar a pantallas anteriores. Este botón es recurrente en todas las subpantallas de la aplicación, por lo que solo se destacará en esta pantalla.

Los siguientes elementos incluyen un campo de texto (*Text Input*) para establecer el título de la rutina, un selector de hora que permite elegir la hora de inicio de la rutina (Figura 5.17), y, como último elemento principal, un interruptor (*switch*) que permite activar o desactivar las notificaciones para esta rutina. Al activar este interruptor, aparece un nuevo botón que permite acceder a la pantalla de editar notificación. El último elemento que encontramos en esta pantalla es el botón “Hecho”, que guarda la rutina y nos devuelve a la pantalla de rutinas.

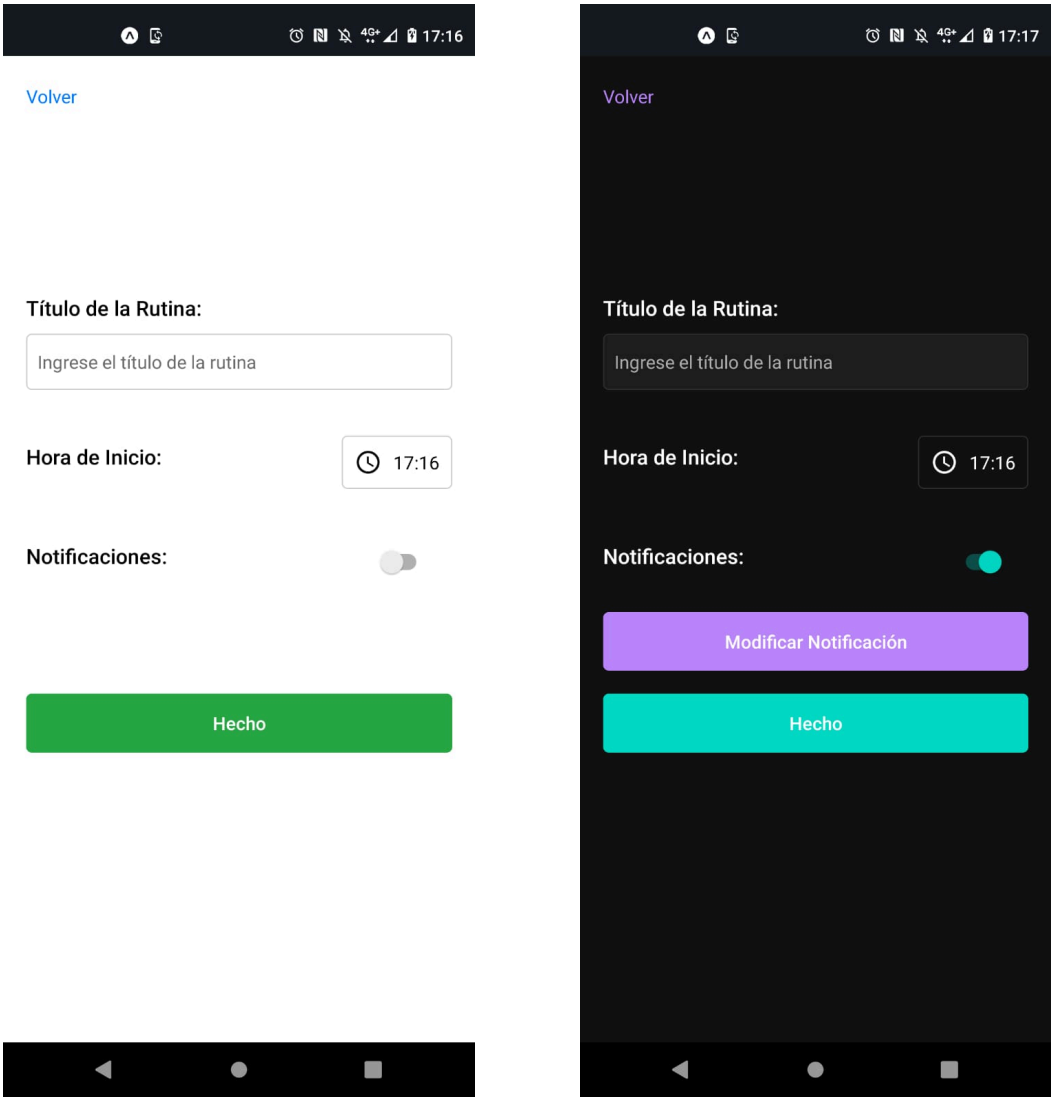


Figura 5.16 Pantalla final de crear rutina, TimeSphere

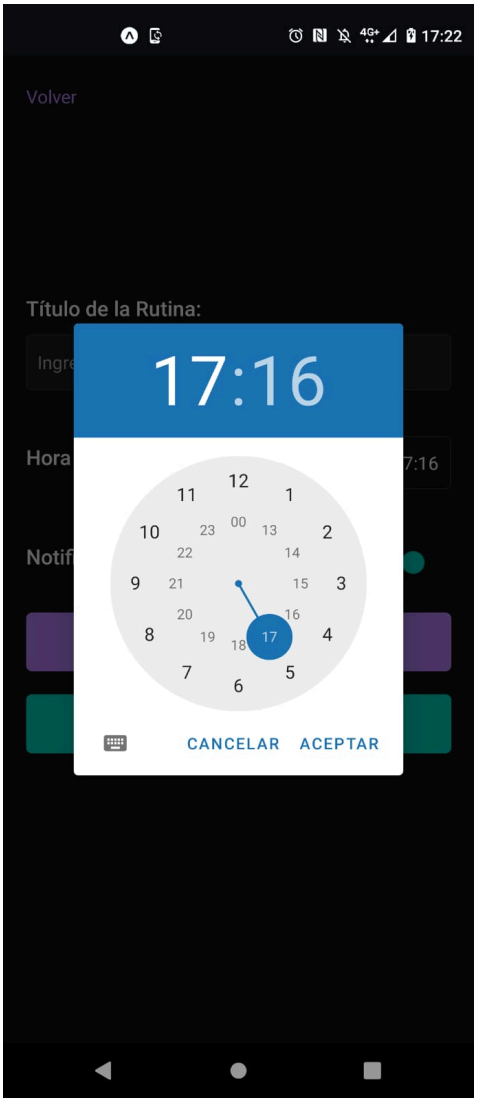


Figura 5.17 Pantalla final de crear rutina (selector de hora), TimeSphere

La siguiente pantalla que vamos a revisar es la de editar notificación (Figura 5.18). Con esta pantalla, podemos editar las notificaciones que la aplicación nos mostrará cuando llegue la hora de realizar la rutina que estamos creando. Esta pantalla incluye un campo de texto para introducir el mensaje de la notificación y dos interruptores (switches) para activar o desactivar el sonido y/o la vibración de la notificación. Por último, encontramos un botón que permite guardar la notificación y volver a la pantalla de editar rutina.

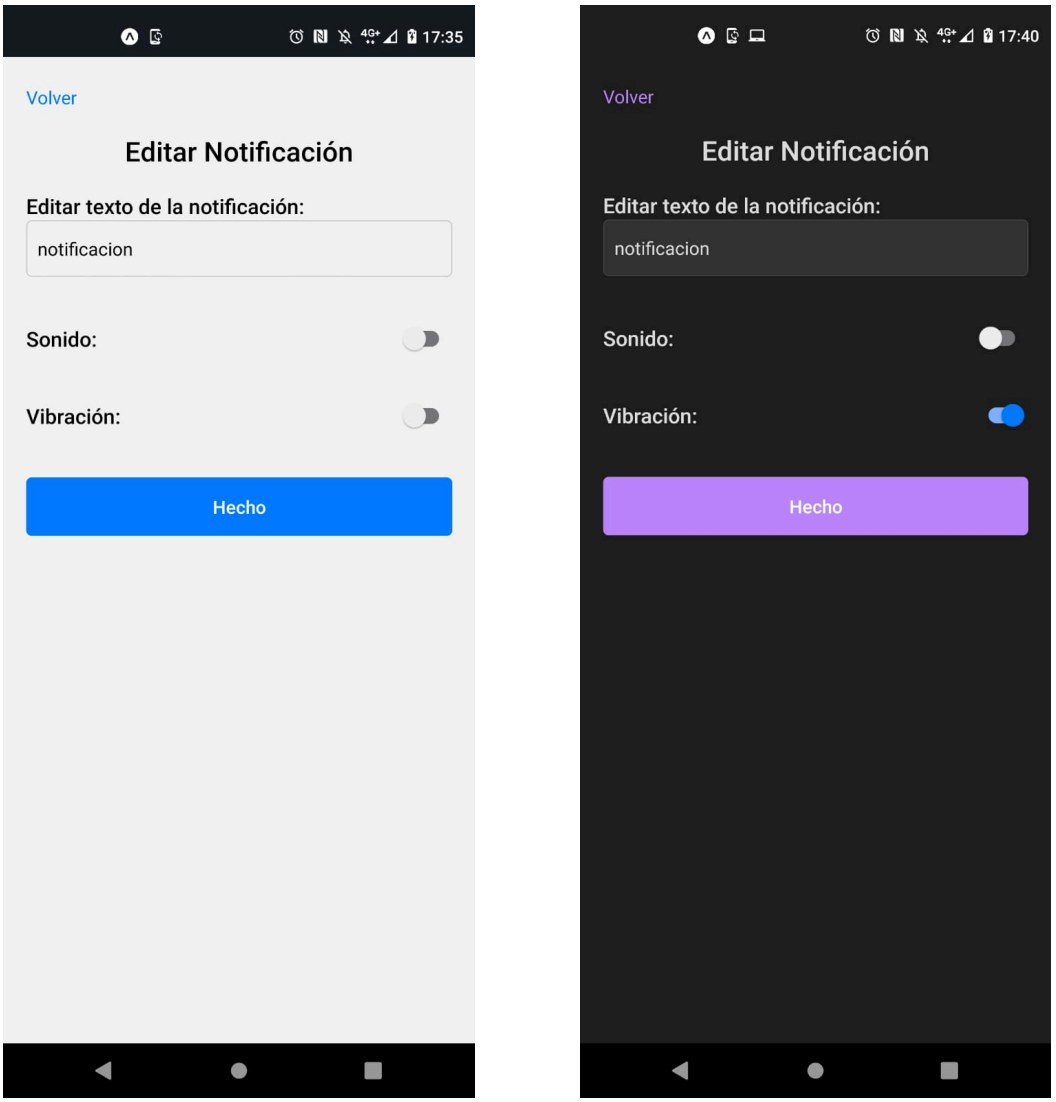


Figura 5.18 Pantalla final de editar notificación, TimeSphere

Para acceder a la siguiente pantalla, debemos pulsar sobre alguna de las rutinas que hayamos creado en la pantalla de rutinas, ya que se trata de la pantalla de editar rutina (Figura 5.19). En esta pantalla, encontramos un título que muestra el nombre de la rutina y un subtítulo que indica la hora de inicio y la hora de finalización de la rutina. La hora final se calcula automáticamente sumando las duraciones de las actividades incluidas en la rutina. Estos dos textos se pueden pulsar para acceder a la pantalla de crear rutina, la cual se rellenará automáticamente con la información actual de la rutina y donde podremos editarla.

A continuación, se muestra una lista con las actividades contenidas en la rutina. Cada actividad aparece con una imagen descriptiva, el título de la actividad, su duración y una etiqueta con el tipo de actividad. Si pulsamos sobre alguna de ellas, accederemos a la pantalla de editar actividad, que se rellenará automáticamente con la información actual de la actividad. Finalmente, en la parte inferior, encontramos dos botones: uno para añadir una nueva actividad y otro para activar o desactivar la rutina.

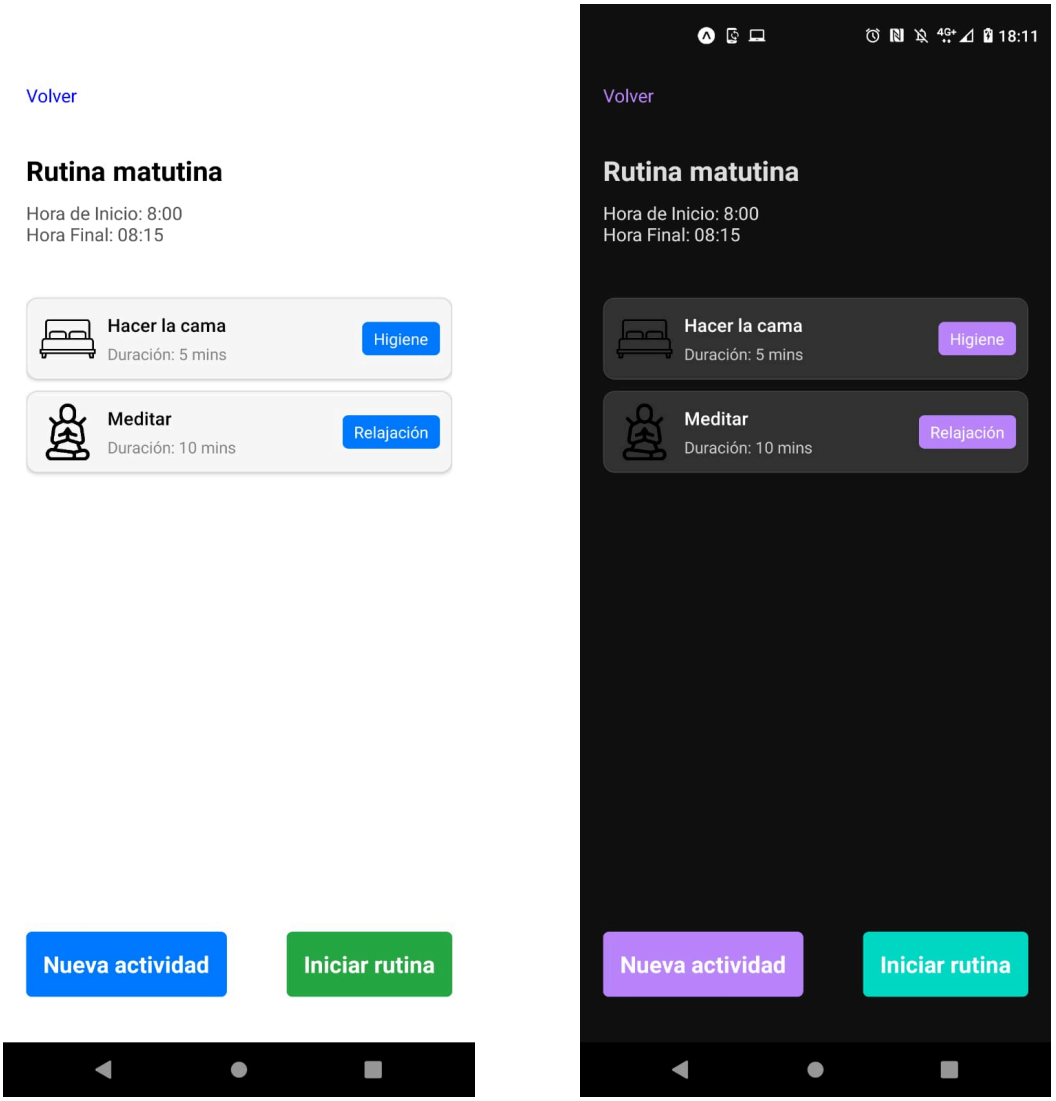


Figura 5.19 Pantalla final de editar rutina, TimeSphere

La siguiente pantalla que vamos a analizar es la pantalla de editar actividad (Figura 5.20). En esta pantalla es donde introducimos los datos de una actividad para añadirla a una rutina o editar una actividad ya existente. Aquí encontramos un campo de texto para introducir el título de la actividad, un campo numérico para especificar la duración en minutos, y un "Picker" que permite seleccionar el tipo de actividad (Figura 5.21). Finalmente, se muestra un botón "Guardar Actividad", que guarda la actividad en la rutina y nos regresa a la pantalla de editar rutina.

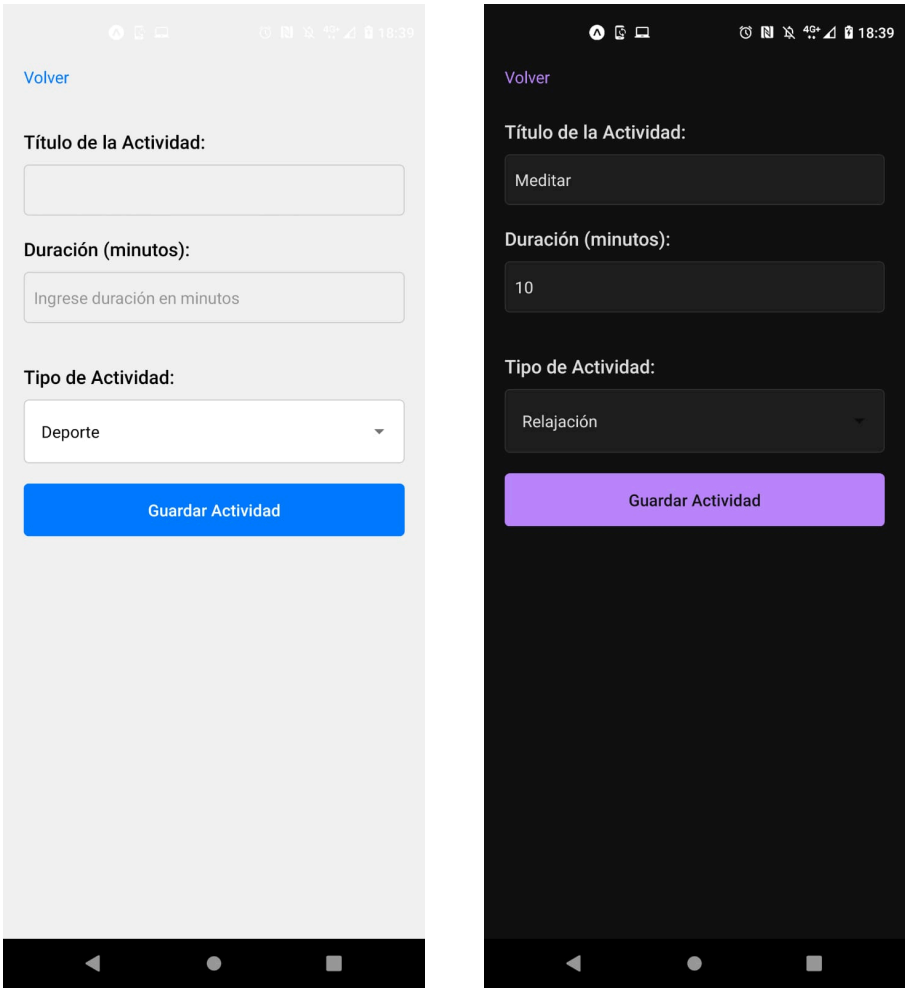


Figura 5.20 Pantalla final de editar actividad, TimeSphere

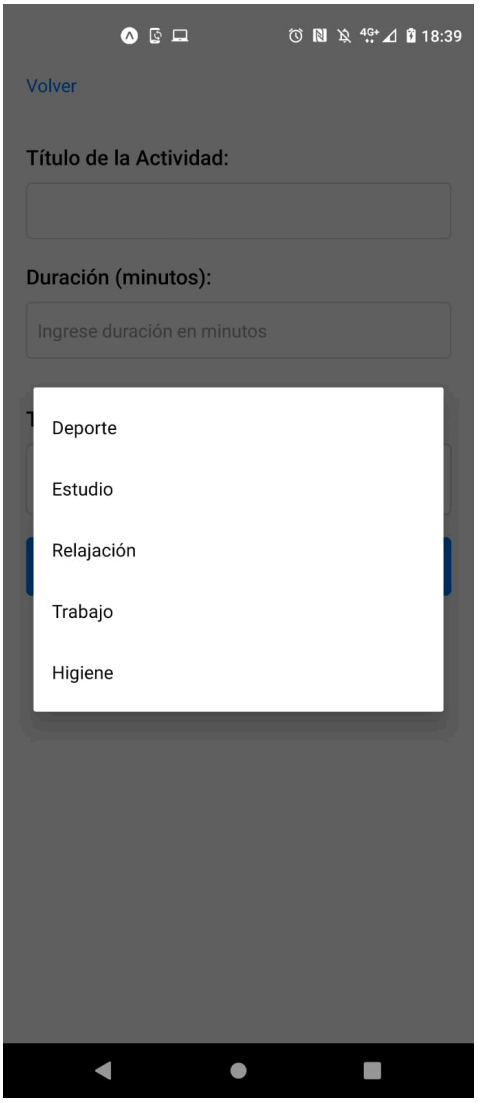


Figura 5.21 Pantalla final de editar actividad (Picker), TimeSphere

La siguiente pantalla que vamos a ver es la última de la sección de rutinas: la pantalla de rutina activa (Figura 5.22). En esta pantalla es donde las rutinas se llevan a cabo. Se muestran sucesivamente todas las actividades asociadas a la rutina que hemos seleccionado anteriormente, con un título que especifica el nombre de la actividad, la imagen de la actividad en grande, y un contador de tiempo. Este contador puede manejarse con los botones situados debajo: un botón para pausar el contador, uno para iniciarlo, y otro para saltar la actividad. Si saltamos la actividad o el contador llega a cero, este se reinicia, estableciendo su nuevo tiempo al asignado para la siguiente actividad, cambiando también el título y la imagen.

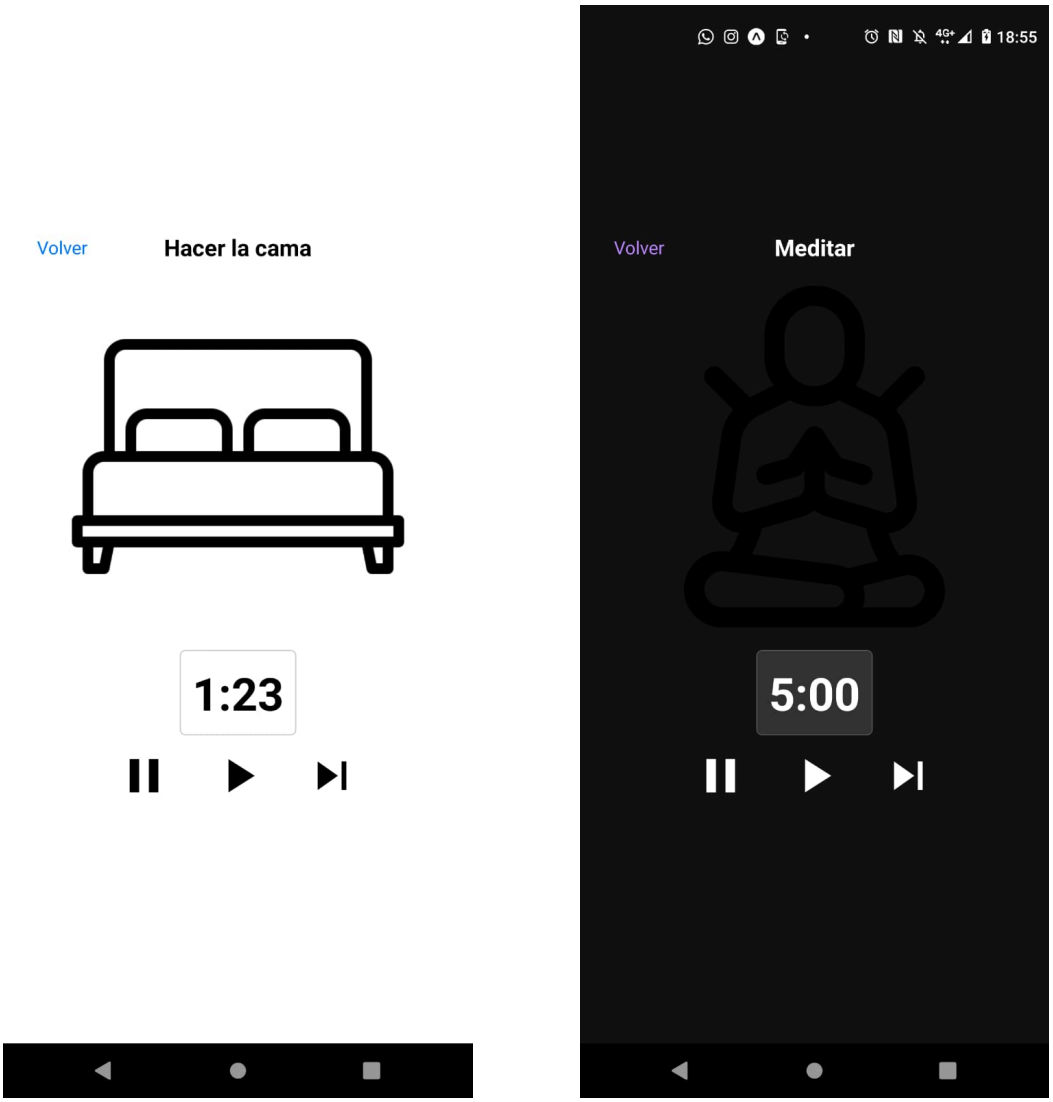


Figura 5.22 Pantalla final de rutina activa, TimeSphere

La siguiente pantalla que vamos a comentar se encuentra en la sección de Calendario y es la pantalla de Diario (Figura 5.23). En esta pantalla, vemos un título que indica el día concreto en el que nos encontramos. A continuación, encontramos la sección de estado emocional, donde podemos modificar el estado de ánimo del día. La última sección está destinada a introducir una entrada en el diario. Por último, hay un botón para guardar los datos del diario para ese día y volver al calendario.

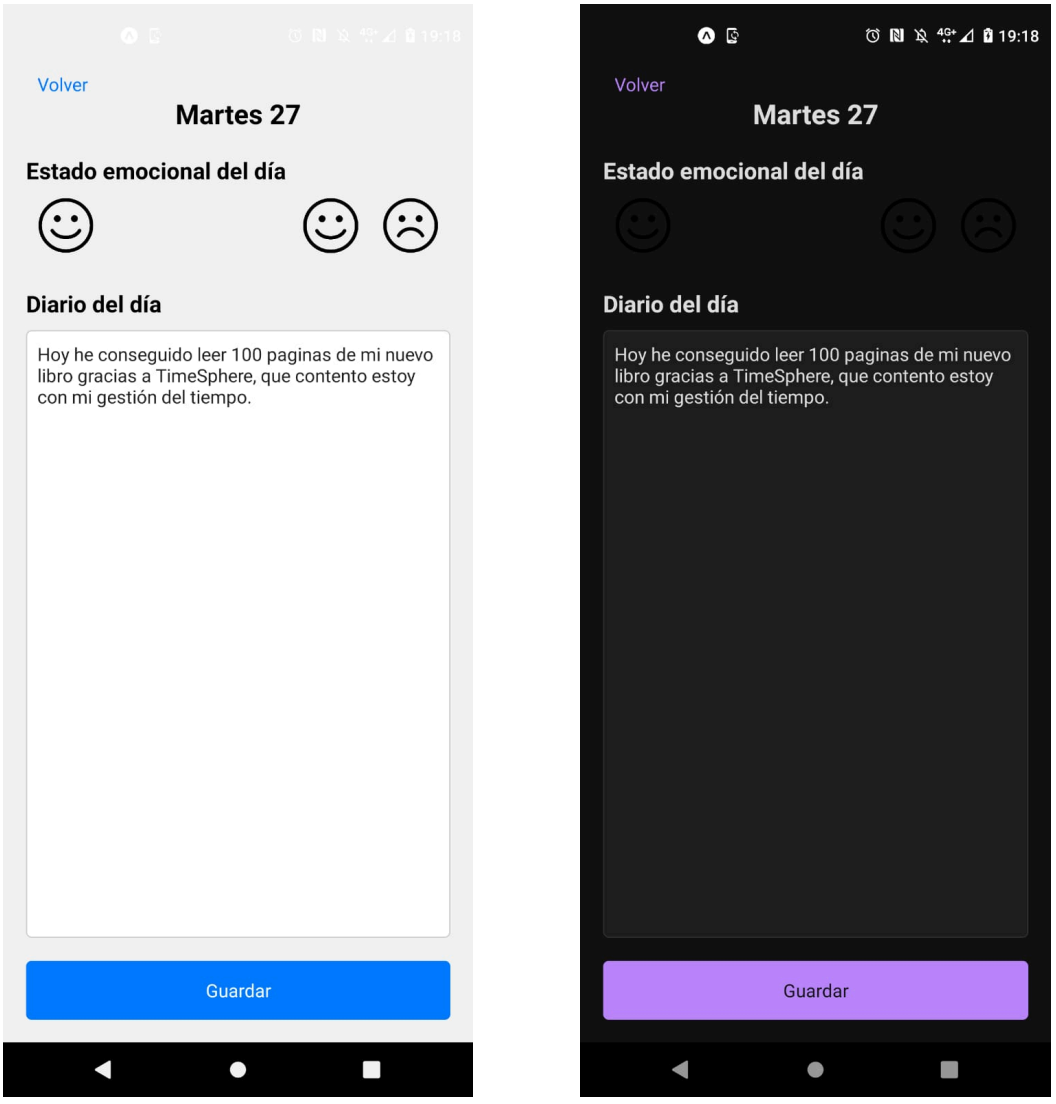


Figura 5.23 Pantalla final de diario, TimeSphere

La siguiente pantalla de la que vamos a ver se encuentra en la sección de supermercados y es la pantalla de filtros (Figura 5.24). En esta pantalla, podemos ver varios tipos de accesibilidad con sus respectivos checklists. Si activamos alguno de estos filtros, se mostrarán únicamente los supermercados que cumplan con los tipos de accesibilidad seleccionados. Si no tenemos ninguno activo, podremos ver todos los supermercados.

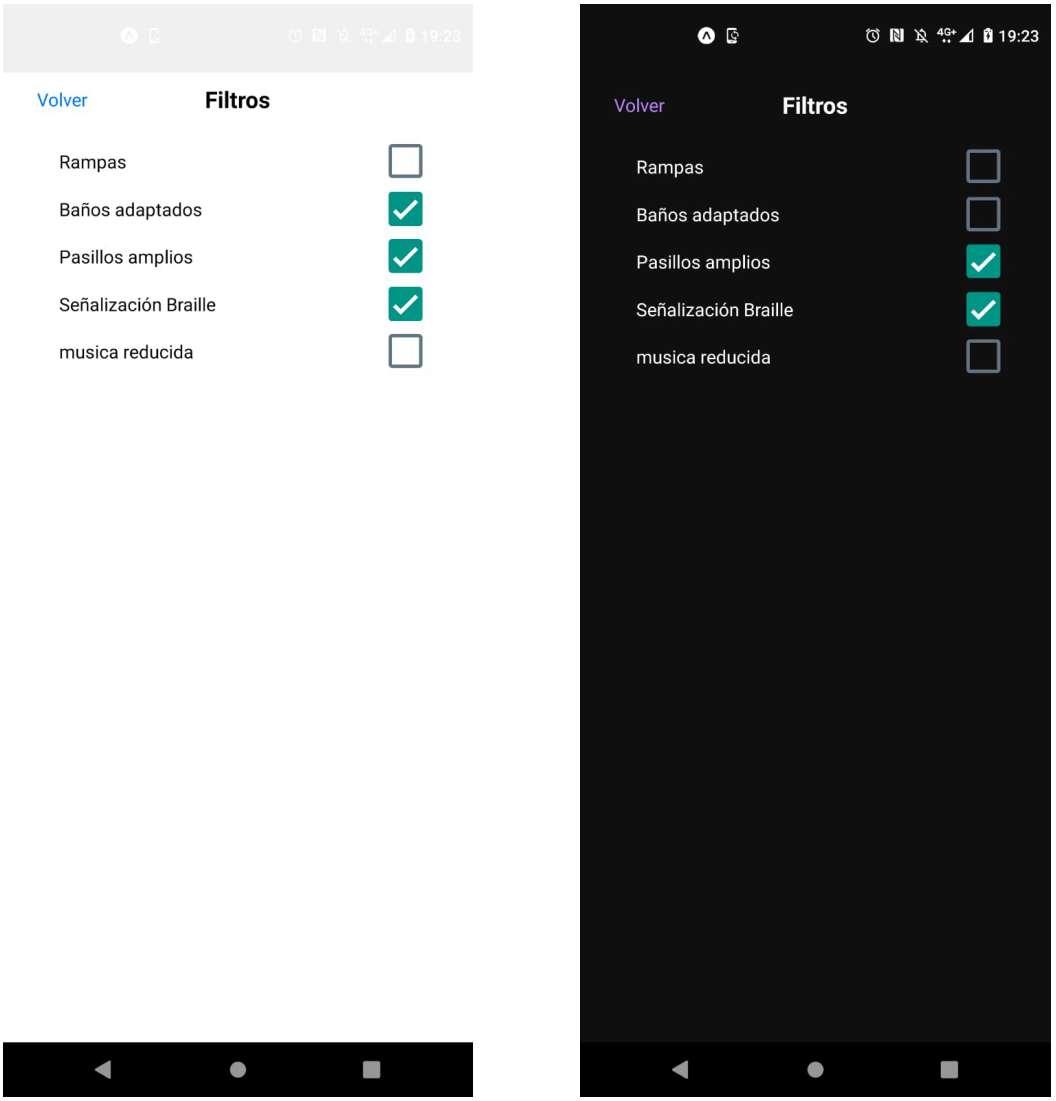


Figura 5.24 Pantalla final de Filtros, TimeSphere

La siguiente pantalla de la que vamos a hablar es la de editar lista de la compra (Figura 5.25). En esta pantalla, tenemos un campo de texto para introducir el título de la lista de la compra, y una lista de productos que podemos marcar para agregar a la lista. Por último, hay un botón que permite guardar la lista de la compra e ir a la pantalla de lista de la compra con los datos de la lista que acabamos de crear.

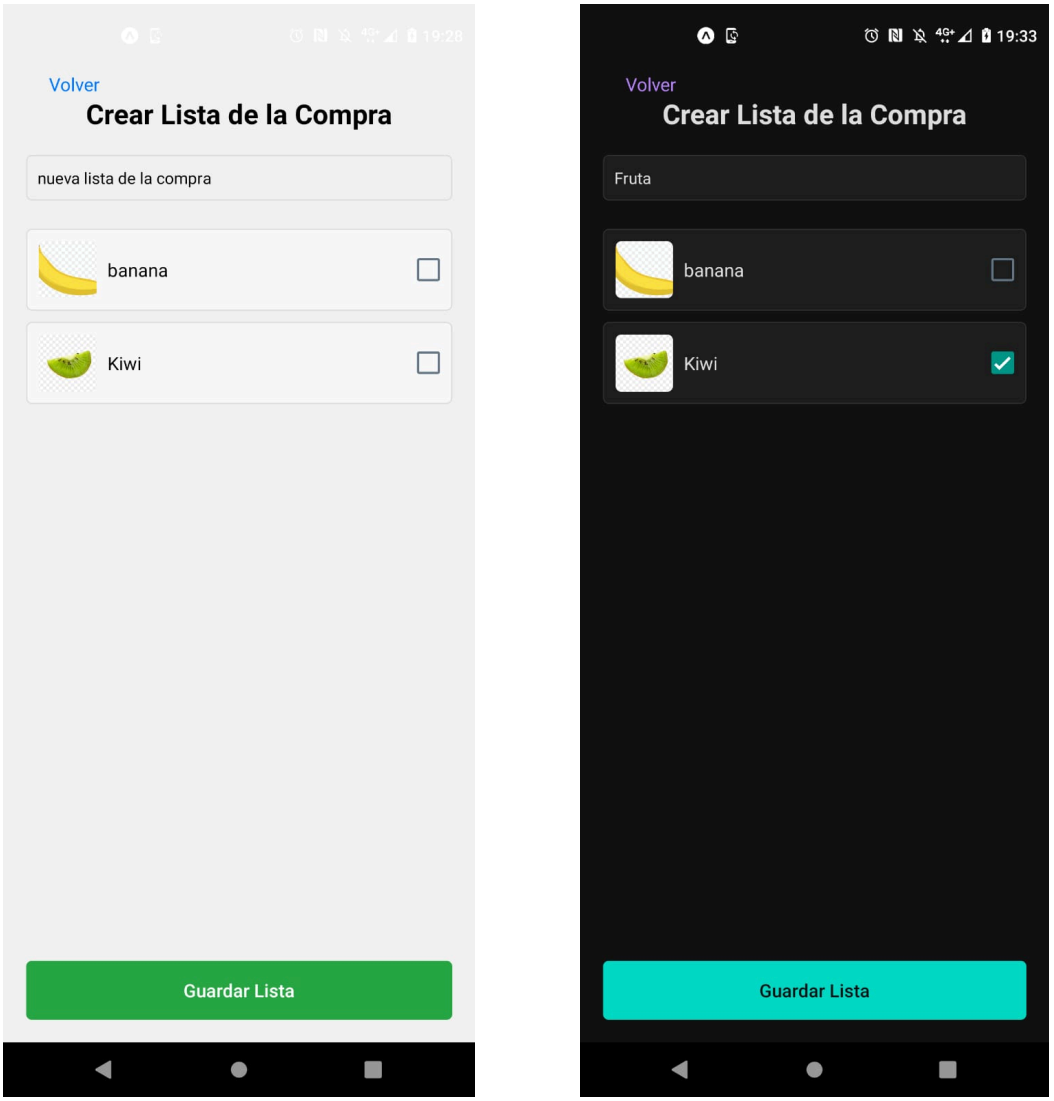


Figura 5.25 Pantalla final de editar listas de la compra, TimeSphere

La última pantalla que vamos a ver es la pantalla de lista de la compra (Figura 5.26). Esta pantalla es muy similar a la anterior, pero en ella podemos ver el nombre de la lista en el título superior y la lista de artículos muestra los artículos que elegimos cuando creamos la lista. En esta pantalla, podemos marcar los productos mientras realizamos la compra para no olvidar ninguno. Por último, el botón inferior cambia a "Editar lista". Con este botón, podemos acceder a la pantalla de editar lista de la compra y modificar el nombre y los productos que deseamos que la lista contenga.

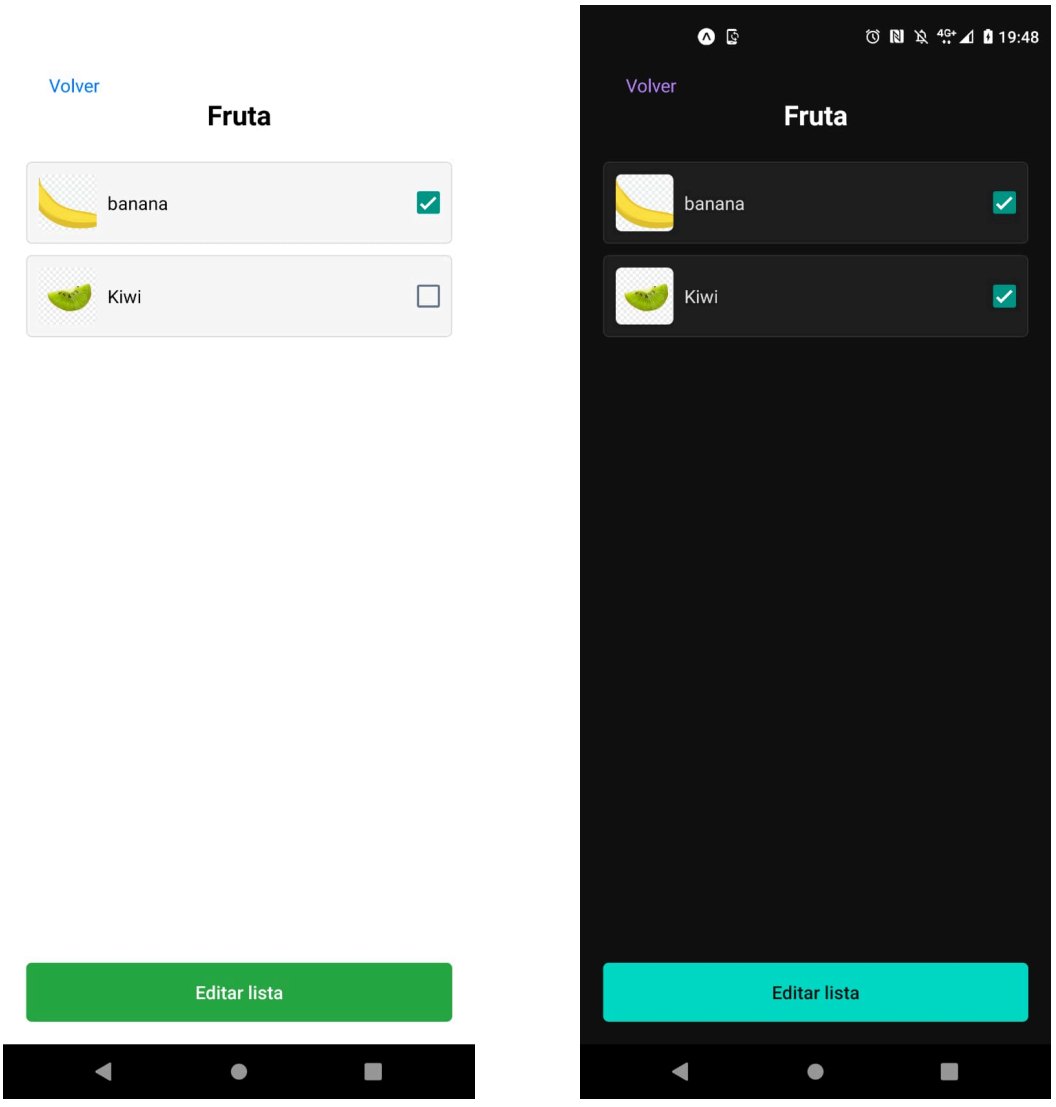


Figura 5.26 Pantalla final de listas de la compra, TimeSphere

5.8. Análisis del trabajo

Gracias a haber registrado las horas de trabajo en la plataforma Worki, ahora podemos realizar un análisis del trabajo realizado durante el desarrollo. Para ello, comentaremos las diferentes gráficas que nos brinda la herramienta Dashboard de Worki (Figura 5.27). Como contexto, este proyecto se ha desarrollado en tres sprints, en los que se abordó una parte diferente de la aplicación: el primero fue la vista, el segundo el modelo y, por último, el controlador.

En estas gráficas podemos ver el estado de las diferentes UT (unidades de trabajo) y cómo han evolucionado a lo largo del tiempo. Como se observa, las tareas se mantuvieron en desarrollo durante un par de semanas. Durante este tiempo, se desarrollaron todas las vistas de la aplicación en paralelo, ya que el diseño de algunas podía cambiar si se encontraba una mejor forma de distribuir los elementos o se querían modificar los estilos, como el modo oscuro, por ejemplo. Al finalizar el primer sprint, todas las UT que se habían desarrollado en paralelo pasaron a estado "terminadas" simultáneamente (la UT rosa se debe a que la UT del modo oscuro se mantuvo en pruebas hasta el final del desarrollo). En el segundo sprint, se desarrollaron todas las UT de los modelos en pocos días, debido a que eran reducidas en cantidad y tamaño. Por último, en el tercer sprint se desarrollaron los controladores, y como se observa, estas UT fueron las que más tiempo consumieron.

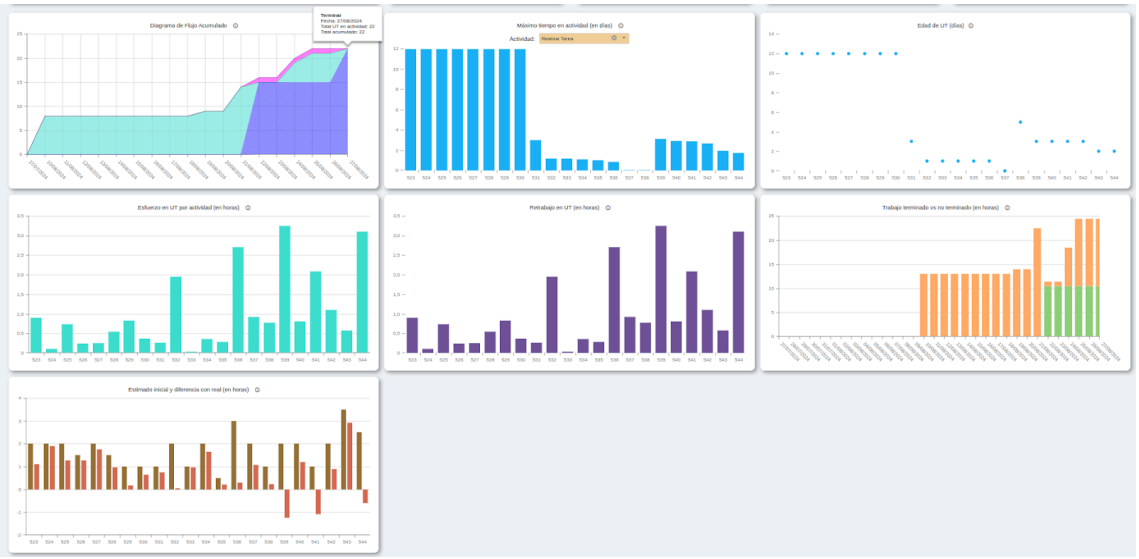


Figura 5.27 Dashboard con gráficas del desarrollo, Worki

6. Pruebas

En el desarrollo de cualquier producto software, las pruebas juegan un papel crucial para asegurar que el producto final sea funcional, fiable y esté libre de errores. Este proyecto no es la excepción y por lo tanto en este apartado explicaremos cuáles han sido las pruebas que hemos realizado durante nuestro desarrollo, y por qué estas nos han permitido determinar que nuestra aplicación no solo cumple con los requisitos establecidos sinó que también estos se han implementado de manera correcta y no producen errores.

Durante nuestro desarrollo se han realizado diferentes tipos de pruebas. Entre ellos, se encuentran las pruebas unitarias para validar la funcionalidad de componentes y funciones individuales, y las pruebas end-to-end para simular flujos de usuario completos y verificar la aplicación en su conjunto.

6.1. Herramientas utilizadas

Para asegurar la calidad del desarrollo y cubrir todos los aspectos del proceso de testing, se han seleccionado varias herramientas especializadas que se integran eficazmente con el entorno de desarrollo de *React Native*. A continuación, se presenta una descripción general de las herramientas utilizadas para las pruebas unitarias y las end-to-end:

Jest [30] es una herramienta de testing ampliamente utilizada en el ecosistema de JavaScript. Está diseñada para realizar pruebas unitarias de manera rápida y sencilla, permitiendo a los desarrolladores escribir y ejecutar tests que verifican el comportamiento de funciones y componentes individuales. Jest se caracteriza por su facilidad de configuración, la capacidad de trabajar con snapshots para comparar el output de componentes, y su compatibilidad nativa con React y React Native. Esto lo convierte en una opción ideal para asegurarse de que cada unidad de código funciona como se espera de manera aislada.

React Native Testing Library [32] es una herramienta que se enfoca en pruebas unitarias y end-to-end, facilitando la verificación del comportamiento individual de los componentes de la interfaz de usuario y su funcionamiento en escenarios completos. Esta librería extiende React Testing Library [33] para soportar aplicaciones móviles desarrolladas con React Native, proporcionando un conjunto de utilidades que permiten simular eventos de usuario y asegurar que los componentes respondan correctamente a estos eventos. La filosofía de esta herramienta se basa en probar el comportamiento de la aplicación desde la perspectiva del usuario, garantizando que tanto los componentes individuales como la aplicación en su totalidad funcionen de manera coherente y predecible.

Detox [34]es una herramienta de testing end-to-end diseñada específicamente para aplicaciones móviles. A diferencia de las pruebas unitarias o de integración, las pruebas end-to-end realizadas con Detox abarcan flujos completos de la aplicación, simulando la interacción del usuario con la interfaz en su totalidad. Esto incluye la navegación entre pantallas, la introducción de datos, y la verificación de que la aplicación se comporta correctamente en un entorno realista. Detox se destaca por su capacidad de ejecutar tests en

dispositivos reales o emuladores, ofreciendo una visión completa del funcionamiento de la aplicación en condiciones reales. Esta herramienta es esencial para garantizar que todos los componentes de la aplicación trabajen de manera coordinada en un entorno de producción.

6.2. Pruebas unitarias

Las pruebas unitarias son un pilar fundamental en el desarrollo de software de calidad. Se centran en validar el comportamiento de componentes individuales o funciones específicas de manera aislada. En el contexto de una aplicación React Native, estas pruebas permiten asegurarse de que cada componente funcione correctamente en distintas situaciones antes de integrarse con otros elementos de la aplicación. Para la realización de estas pruebas, se utilizará tanto Jest como *React Native Testing Library* como librería para crear los tests.

Dentro de las diferentes opciones que tenemos para realizar pruebas unitarias, nos hemos decantado por analizar nuestros componentes (Figura 6.1), ya que como no son componentes predefinidos por react native, existe una mucho mayor posibilidad de encontrar errores en estos, concretamente vamos a crear pruebas que comprueben que los componentes se renderizan de manera adecuada, en modo claro y en modo oscuro, que respondan correctamente a los eventos que puedan tener que manejar como pueden ser eventos *OnTouch* o *checked*, y que respondan correctamente a eventos de tiempo (caso del componente contador).

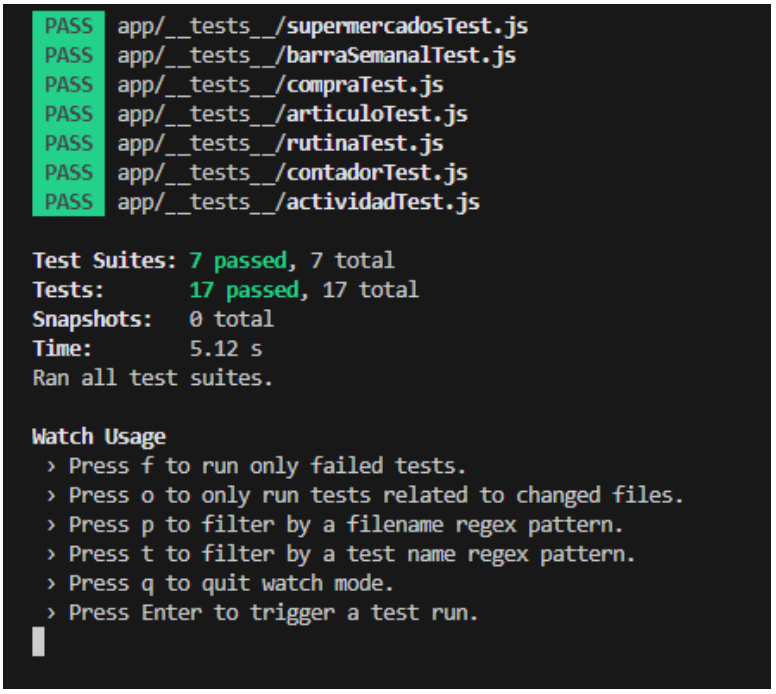


Figura 6.1 pruebas unitarias, TimeSphere

6.3. Pruebas end to end

Las pruebas *End-to-End* (E2E) son un tipo de prueba de software que simula el comportamiento real del usuario al interactuar con una aplicación. A diferencia de las pruebas unitarias o de integración, que verifican partes específicas de la aplicación, las pruebas E2E abarcan todo el flujo de la aplicación desde el principio hasta el fin.

El objetivo principal de las pruebas E2E es asegurarse de que todos los componentes de la aplicación funcionen correctamente juntos en un entorno lo más cercano posible al de producción. Estas pruebas cubren todo el proceso, desde la interfaz de usuario (UI) hasta la comunicación con bases de datos, sistemas de red y otros servicios externos.

Para realizar estas pruebas hemos usado Detox, una herramienta que nos ha permitido definir pruebas en Jest y ejecutarlas en un simulador en tiempo real. Estos tests no se han considerado correctos hasta que el tester haya realizado la prueba en el simulador. Para que esta simulación sea lo más cercana posible a la experiencia real de nuestros usuarios, hemos utilizado un dispositivo móvil físico como simulador para estas pruebas.

7. Conclusiones y trabajos futuros

El desarrollo de la aplicación móvil para la gestión del tiempo ha sido un proceso enriquecedor que ha permitido aplicar y consolidar los conocimientos adquiridos a lo largo de la carrera de Ingeniería Informática. Este proyecto no solo ha sido una oportunidad para poner en práctica conceptos teóricos, sino que también ha fomentado el desarrollo de habilidades prácticas en programación, diseño de interfaces y gestión de proyectos.

A lo largo del trabajo, se han integrado diversas materias cursadas, como programación orientada a objetos, diseño de bases de datos y desarrollo de aplicaciones móviles. La elección de JavaScript y React Native como tecnologías de desarrollo ha sido fundamental, ya que estas herramientas son ampliamente utilizadas en la industria, lo que nos ha permitido familiarizarnos con tecnologías que serán de gran utilidad en futuros proyectos profesionales.

Además, el enfoque en la accesibilidad y la inclusión de personas con trastornos de atención ha sido un aspecto clave del proyecto. Esto no solo refleja una responsabilidad social, sino que también abre la puerta a futuras investigaciones y desarrollos en el ámbito de la tecnología asistiva. La experiencia adquirida en la creación de una aplicación que responde a necesidades específicas de un grupo de usuarios nos ha motivado a considerar la importancia de la empatía y la usabilidad en el diseño de software.

De cara al futuro, este trabajo sienta las bases para la continuación del desarrollo de la aplicación. Se pueden explorar nuevas funcionalidades, como la integración de inteligencia artificial para personalizar aún más la experiencia del usuario, o la implementación de análisis de datos para mejorar la gestión del tiempo. Asimismo, se prevé la posibilidad de colaborar con profesionales de la psicología y la educación para enriquecer el contenido y las herramientas ofrecidas en la aplicación.

En resumen, este proyecto no solo ha sido un ejercicio académico, sino un paso hacia la creación de una herramienta que puede tener un impacto positivo en la vida de muchas personas. La combinación de conocimientos técnicos y la sensibilidad hacia las necesidades de los usuarios ha sido fundamental para el éxito de este proyecto.

Bibliografía

[1] Web oficial de la aplicación Routinery. Consultar a https://routinery.app/. Última consulta 29/7/2024.

[2] Web oficial de la aplicación Timetune. Consultar a https://timetune.app/. Última consulta 29/7/2024.

[3] Web oficial de la aplicación Todoist. Consultar a https://todoist.com/es. Última consulta 29/7/2024.

[4] Web oficial de la desarrolladora de Me+. Consultar a https://enerjoy.life/. Última consulta 29/7/2024.

[5] Repositorio de la aplicación Sphere. Consultar a https://github.com/nuuuuur02/SPHERE-/tree/main. Última consulta 4/8/2024.

6] Introducción al lenguaje de modelado unificado. Universitat Oberta de Catalunya. Disponible en: https://openaccess.uoc.edu/bitstream/10609/9121/1/Intro_UML.pdf. Última consulta: 5/8/2024."

[7] Página web oficial de WireframePro. Consultar a https://wireframepro.mockflow.com/#Wireframe. Última consulta 9/8/2024.

[8] Novac, D. (2023). *UX Design Principles*. UX Media. Disponible en: https://uxmedia.io/blog/ux-design-principles/. Última consulta 10/8/2024.

[9] Guia en la página web de la consultora Open Forge. Consultar a https://openforge.io/mobile-academy/guides/mobile-app-design/achieve-accessibility-in-mobile-application-design/. Última consulta 10/8/2024.

[10] Entrada en la web de la fundación del diseño de la interacción. Consultar a https://www.interaction-design.org/literature/article/lessons-from-ixdf-mobile-ui-design-course?srsltid=AfmBOooI_eJYTJV0gOvfBV3O3qjltpnaVaTPWooULIdN5cDA_W8Sqqw5. Última consulta 10/8/2024.

[11] Entrada en el blog Perficient. Consultar a https://blogs.perficient.com/2024/03/06/creating-universal-design-experiences-the-power-of-intuitive-mobile-app-design/. Última consulta 10/8/2024.

[12] Página web oficial del grupo Nielsen Norman. Consultar a https://www.nngroup.com/. Última consulta 10/8/2024.

[13] Blancarte, O. (s.f.). Patrón de diseño Modelo-Vista-Controlador (MVC). Oscar Blancarte - Software Architecture. Consultar a: https://www.oscarblancarteblog.com/2014/07/21/patron-de-diseno-modelo-vista-controlador-mvc/. Última consulta: 13/8/2024.

[14] Documentación oficial de React Native. Consultar a https://reactnative.dev/docs/getting-started. Última consulta 16/8/2024.

[15] Documentación oficial de React. Consultar a https://es.react.dev/learn. Última consulta 16/8/2024.

[16] Artículo técnico sobre JavaScript. Consultar a https://developer.mozilla.org/es/docs/Web/JavaScript. Última consulta 16/8/2024.

[17] Documentación oficial de FireBase. Consultar a https://firebase.google.com/docs/guides?hl=es. Última consulta 16/8/2024.

[18] Documentación específica de Cloud Firestore. Consultar a https://firebase.google.com/docs/firestore?hl=es. Última consulta 16/8/2024.

[19] Documentación oficial de Expo. Consultar a https://docs.expo.dev/router/introduction/. Última consulta 16/8/2024.

[20] Documentación oficial de Git. Consultar a https://git-scm.com/doc. Última consulta 16/8/2024.

[21] Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress. ISBN 978-1484200773.

[22] Documentación oficial de Github. Consultar a https://docs.github.com/es. Última consulta 16/8/2024.

[23] Documentación oficial de Worki. Consultar a https://cliente.tuneupprocess.com/help/web/Paraempezar.html. Última consulta 16/8/2024.

[24] Documentación específica de Expo Router. Consultar a https://docs.expo.dev/router/introduction/. Última consulta 22/8/2024.

[25] Documentación específica de React Navigation. Consultar a https://reactnavigation.org/docs/getting-started. Última consulta 22/8/2024.

[26] Documentación específica de *Context*. Consultar a https://legacy.reactjs.org/docs/context.html. Última consulta 22/8/2024.

[27] Documentación oficial de React Native. Consultar a https://legacy.reactjs.org/docs/components-and-props.html#gatsby-focus-wrapper. Última consulta 22/8/2024.

[28] Documentación oficial de React Native. Consultar a https://reactnative.dev/docs/handling-touches. Última consulta 22/8/2024.

[29] Documentación oficial de React Native. Consultar a https://legacy.reactjs.org/docs/hooks-effect.html. Última consulta 22/8/2024.

[30] Página web oficial del framework Jest. Consultar a https://jestjs.io/es-ES/. Última consulta 26/8/2024.

[31] Documentación oficial de la biblioteca de React Native Testing Library. Consultar a https://callstack.github.io/react-native-testing-library/docs/start/intro. Última consulta 26/8/2024.

[32] Documentación oficial de la biblioteca de React Testing Library. Consultar a https://testing-library.com/docs/react-testing-library/intro/. Última consulta 26/8/2024.

[33] Documentación oficial de la biblioteca de Detox. Consultar a https://wix.github.io/Detox/. Última consulta 26/8/2024.

Objetivos de desarrollo sostenible, apéndice A

Los ODS, o Objetivos de Desarrollo Sostenible (Figura B.1), son un conjunto de 17 objetivos globales establecidos por la Asamblea General de las Naciones Unidas en 2015, como parte de la Agenda 2030 para el Desarrollo Sostenible. Estos objetivos están diseñados para abordar una amplia gama de problemas globales y promover el desarrollo económico, social y ambiental de manera equitativa y sostenible. Cada ODS tiene metas específicas y un marco para medir el progreso hacia su consecución.

Figura A.1: Objetivos de Desarrollo Sostenible

Objetivos de desarrollo sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X

ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**			X	
ODS 10. **Reducción de las desigualdades.**	X			
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Tabla A.1: nivel de relación de nuestro proyecto con las diferentes ODS

De todas las ODS hay 4 que tienen una relación con nuestro proyecto, En este apéndice vamos a comentar cuales són y porqué tienen relación con nuestro proyecto.

ODS con relación Alta:

- Reducción de desigualdades: La relación de nuestra aplicación con este ODS es evidente, ya que nuestro proyecto se ha enfocado desde el inicio en ser accesible para todos, incluidas las personas con problemas cognitivos. Muchas de nuestras funcionalidades están diseñadas para reducir las desigualdades derivadas de diversas condiciones. Por ejemplo, el localizador de supermercados accesibles ayuda a las personas a encontrar tiendas que se ajusten a sus necesidades, contribuyendo a la reducción de barreras y desigualdades.

ODS con relación media:

- Salud y bienestar: La capacidad de crear y seguir rutinas diarias puede llevar a una vida más organizada y equilibrada, lo cual es crucial para la salud mental y física. Las rutinas ayudan a reducir el estrés y la ansiedad al proporcionar estructura y previsibilidad en la vida diaria. Además, el uso de imágenes para mejorar la accesibilidad y concentración puede hacer que la gestión del tiempo sea más efectiva y accesible para personas con dificultades visuales o de concentración, promoviendo así el bienestar general.
- Educación de calidad: La gestión efectiva del tiempo es una habilidad importante para el aprendizaje y el desarrollo personal. La creación de rutinas ayuda a establecer hábitos de estudio y aprendizaje regulares, lo que puede mejorar el rendimiento académico y el desarrollo personal. La accesibilidad mejorada mediante imágenes también facilita el aprendizaje para usuarios con diferentes estilos de aprendizaje y necesidades.

ODS con relación baja:

- Industria, innovación e infraestructuras: La funcionalidad del localizador de supermercados accesibles puede ayudar a las industrias de distribución de alimentos y otros sectores a mejorar la visibilidad de sus sistemas de accesibilidad. Esto puede fomentar la adopción de prácticas más inclusivas y mejorar el acceso a sus servicios en todos los centros, promoviendo así una infraestructura más equitativa e innovadora.