Diseño e implementación de un launcher Android enfocado a combatir la procrastinación por el uso excesivo del smartphone en estudiantes de pregrado de la Universidad del Valle

Juan Sebastián Ruiz Aguilar

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Tuluá, Valle del Cauca
2025

Diseño e implementación de un launcher Android enfocado a combatir la procrastinación por el uso excesivo del smartphone en estudiantes de pregrado de la Universidad del Valle

Juan Sebastián Ruiz Aguilar
Código: 2059898
juan.ruiz.aguilar@correounivalle.edu.co

Director
Ing. Héctor Fabio Ocampo Arbeláez
Magister en analítica e inteligencia de negocios
hector.ocampo@correounivalle.edu.co

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Tuluá, Valle del Cauca
2025

Tabla de contenido

Índice de cuadros

Índice de figuras

Resumen

Los estudiantes universitarios que hacen un uso excesivo y/o inadecuado del smartphone tienen un menor rendimiento académico e insatisfacción al no gestionar adecuadamente su tiempo e incumplir los plazos de sus actividades pendientes. Dado que el ecosistema de los teléfonos móviles y sus aplicaciones incentiva a pasar el mayor tiempo posible usándolos, se propone desarrollar una pantalla de inicio (launcher) para teléfonos Android que apunte a reducir la procrastinación mediante una interfaz mínimamente llamativa, a mitigar las distracciones y a gestionar mejor el tiempo invertido dentro de las aplicaciones, principalmente en horarios donde se requiera total disposición a una actividad concreta.

Palabras clave: Procrastinación, Uso Excesivo del Smartphone, Gestión del Tiempo, Launcher Android, Productividad.

Introducción.

En un mundo conectado digitalmente, donde la tecnología tiene cada vez más alcance y uso en todas las actividades de la vida diaria, es muy común emplear un smartphone para realizarlas o complementarlas. Su amplio abanico de herramientas lo hace apropiado para muchos tipos de tareas y esto hace que, en ocasiones, se emplee gran cantidad de tiempo en ellos, generando distracciones casi inevitables. Esto es perjudicial para las personas que necesitan concentrarse lo mejor posible en su jornada laboral o académica y está afectando, en gran medida, a los estudiantes universitarios.

Los dispositivos móviles y las aplicaciones hoy en día (en especial las empresas detrás de ellos) diseñan sus productos con base en ciertos principios y teorías que fomentan un uso adictivo, teniendo como objetivo retener a los usuarios el mayor tiempo posible dentro de las plataformas [3] y así maximizar sus ganancias dependiendo del modelo de negocio de cada una. Esto es lo que Santiago Giraldo y Cristina Fernández denominan la *economía de la atención* [4].

Un estudio realizado a 536 estudiantes de educación superior en Estados Unidos revela que aquellos que pasan una cantidad de tiempo prolongada usando el smartphone ven disminuido su rendimiento académico y su nivel de aprendizaje por las constantes distracciones que genera, además de afectar las variables como la motivación y la autorregulación, aspectos clave en la formación de los individuos [5]. Además, muchos de los estudiantes no son conscientes de cuánto tiempo pasan en estos dispositivos: su uso se ha vuelto parte de su rutina desde el primer momento del día y se ven en la necesidad de permanecer conectados [4]. Estos factores propician que la situación se vuelva repetitiva y pueda catalogarse como procrastinación, haciendo que los universitarios acumulen y posterguen sus actividades, trayendo consigo dificultades no solo en las aulas de clase, sino también a nivel personal [6], físico [7, 8], emocional y social [9].

Para minimizar estos comportamientos y analizar a detalle el uso del smartphone, teniendo en cuenta que aproximadamente el 70 % de los usuarios poseedores de un smartphone a nivel mundial tienen instalado el sistema operativo Android [10], se propone desarrollar un launcher para esta plataforma que ayude a los universitarios a gestionar de mejor manera el tiempo que pasan en su smartphone enfocado a disminuir la procrastinación, especialmente en lo académico, y aumentar sus niveles de productividad y bienestar general.

1. Formulación del problema.

1.1. Descripción del problema.

La forma en la que los estudiantes universitarios llevan a cabo su proceso de formación tiene un gran componente autodidacta que requiere de concentración, autorregulación, autoorganización y autoevaluación tanto en el aula de clase como en otros espacios y tiempos destinados al desarrollo de sus actividades académicas [11]. Su proceso fluye normalmente cuando están presentes varios de estos factores, pero la productividad disminuye al tener distractores constantes en el entorno como pueden ser las notificaciones provenientes del smartphone, un ambiente ruidoso en casa o en espacios fuera del aula de clase. De hecho, el aula de clase también puede convertirse en un espacio propicio a distracciones si la temática de la clase es confusa y/o monótona, conduciendo al uso ineludible del smartphone para desconectar del momento.

Estos dispositivos se convierten en un escape, una salida rápida de los momentos difíciles, la soledad, las responsabilidades, entre otros. Además, se expande hasta abarcar una gran porción de tiempo en la vida diaria, reduciendo o eliminando los momentos que se deberían usar para el crecimiento personal y profesional, subestimando los plazos de entrega de sus pendientes pensando que aún hay tiempo suficiente y que se cuenta con la capacidad de elaborarlos, sin pensar en consecuencias futuras [12].

A pesar de que muchos de los estudiantes son conscientes del uso excesivo que hacen de las plataformas móviles [4, 9], se crea una bola de nieve de insatisfacción, frustración, estrés y ansiedad por postergar sus actividades, no cumplir sus objetivos, no entregar a tiempo sus trabajos y no interiorizar adecuadamente el conocimiento adquirido, haciendo que el rendimiento en su proceso formativo disminuya y su estabilidad emocional se vea afectada. Todos estos sentimientos solo acentúan más el problema de la procrastinación por el uso del smartphone porque se sigue recurriendo a él buscando dopamina que genera, dando paso a una mala gestión del tiempo y deteriorando la calidad de los resultados esperados en sus actividades.

El uso prolongado de plataformas móviles como redes sociales y servicios de streaming afecta negativamente varios aspectos de la vida de los estudiantes incluyendo el rendimiento académico, la salud mental y el bienestar, el desarrollo social y la adicción a los dispositivos. Las empresas que dirigen este sector utilizan estrategias de diseño psicológico, como el uso del color, el texto y la tipografía, para mantener a los usuarios más tiempo en sus aplicaciones, creando un sentido de urgencia y pertenencia a través de notificaciones constantes y actualizaciones frecuentes de contenido y características nuevas, aprovechándose de aspectos primitivos de la misma psicología humana [13].

Preocupa aún más si se tiene en cuenta que, según estadísticas del Ministerio de Educación de Perú, la mayoría de los estudiantes de educación superior, concretamente el 65 %, se encuentra en el rango entre los 18 y 25 años [14], mismo rango en el cual se determinó que los universitarios tienen un alto grado de procrastinación y que es significativamente mayor que aquellos por encima de los 25 años [15].

Por todos los problemas mencionados surge la necesidad de, a través del mismo dispositivo en el que los estudiantes universitarios reinciden en un comportamiento adictivo,

regular el uso de las aplicaciones y ayudar a recuperar la excesiva cantidad de tiempo invertido en el smartphone que le corresponde a actividades de mayor importancia, sobre todo a nivel académico y personal.

1.2. Definición del problema.

¿Cómo puede una aplicación móvil diseñada para regular el tiempo de uso del smartphone y optimizar la gestión de las asignaciones ayudar a los estudiantes universitarios a minimizar la procrastinación vinculada al uso excesivo de estas plataformas y a mejorar su concentración, autorregulación y productividad en el ámbito académico y personal?

2. Marco referencial.

2.1. Marco teórico.

Procrastinación. Es el hábito de posponer tareas o responsabilidades cruciales en favor de actividades más inmediatas o gratificantes, aunque menos relevantes a largo plazo. Esta tendencia puede estar influenciada por una variedad de factores psicológicos y ambientales, como la ansiedad ante el fracaso, la falta de motivación intrínseca para completar las tareas asignadas y la presencia constante de distracciones, entre las cuales destaca el uso excesivo del smartphone. La combinación de estos elementos puede llevar a un ciclo perjudicial de aplazamiento continuo, afectando negativamente tanto el rendimiento académico como el bienestar emocional de los estudiantes [9].

Tecnología y adicción. La creciente adicción a la tecnología, particularmente a los dispositivos móviles y sus aplicaciones, ha captado la atención de investigadores y profesionales en salud mental. Estos dispositivos están diseñados con características específicas destinadas a maximizar la participación del usuario, desde las notificaciones constantes hasta la gamificación de las experiencias de usuario. Estas estrategias pueden desencadenar comportamientos adictivos al generar una gratificación instantánea y dificultar la capacidad de autorregulación del tiempo de uso. Como resultado, los usuarios pueden encontrarse atrapados en un ciclo de consumo compulsivo de contenido digital, lo que puede tener repercusiones negativas en su bienestar psicológico y en su capacidad para concentrarse en actividades importantes, como el estudio académico [3].

Teoría del diseño centrado en el usuario. Es una metodología orientada a crear productos y servicios que se adapten de manera óptima a las necesidades, preferencias y habilidades de los usuarios finales. El diseño centrado en el usuario implica iteraciones continuas y pruebas de usabilidad para garantizar que el producto final sea fácil de usar y cumpla con las expectativas de los usuarios. Esto implica la creación de prototipos y la realización de pruebas con estudiantes para evaluar la funcionalidad, la accesibilidad y la eficacia del launcher en la mejora de la gestión del tiempo y la reducción de la procrastinación.

Psicología del color. La Psicología del Color resalta cómo los colores influyen en las emociones y el comportamiento humano. En el diseño del launcher, la elección cuidadosa de colores es crucial, ya que puede impactar la motivación, enfoque y compromiso del usuario con las tareas académicas. Los tonos cálidos como el rojo y el naranja pueden estimular la energía y la acción, fomentando la productividad. Mientras tanto, colores suaves como el azul y el verde pueden crear un ambiente tranquilo, ideal para momentos de concentración y estudio. Mantener consistencia en la paleta de colores y su integración con la interfaz del launcher puede mejorar la experiencia del usuario, aumentando su disposición a utilizar la aplicación de manera efectiva para gestionar su tiempo y combatir la procrastinación.

Gestión del tiempo. Es el proceso de planificar, organizar y controlar cómo se utiliza el tiempo disponible para lograr objetivos específicos, tanto personales como profesionales. Implica identificar las tareas prioritarias, asignarles el tiempo adecuado y utilizar técnicas y herramientas para maximizar la eficiencia y la productividad. El launcher podría ofrecer funciones como recordatorios personalizados para tareas importantes, integración

de calendario para una visualización clara de horarios y compromisos, y la capacidad de establecer objetivos académicos con seguimiento de progreso. Estas características permitirían a los estudiantes planificar y organizar sus actividades de manera efectiva, evitando conflictos y garantizando el cumplimiento de plazos, mientras identifican áreas de mejora para optimizar su rendimiento académico [16].

Productividad. Es la capacidad de producir más resultados con la misma cantidad de recursos, o producir los mismos resultados con menos recursos, en un período de tiempo determinado. Esto es esencial para el éxito tanto académico como personal, implicando la eficiencia en la realización de tareas y la obtención de resultados satisfactorios. En el contexto del launcher, se busca potenciar la productividad de los estudiantes mediante la minimización de distracciones y el logro de objetivos de manera efectiva. Esto se logra a través de funciones que permiten enfocarse en tareas prioritarias y limitar las interrupciones, organizar aplicaciones de manera intuitiva para acceder rápidamente a recursos de estudio, y proporcionar herramientas de seguimiento del tiempo para analizar y mejorar la eficiencia en el uso del dispositivo.

2.2. Estado del arte.

El estudio ."Adicción a las Redes Sociales y Procrastinación Académica en estudiantes Universitarios"[17] encontró una correlación positiva y significativa entre la adicción a las redes sociales y la procrastinación académica, lo que significa que niveles más altos de adicción a las redes sociales corresponden a niveles más altos de procrastinación académica. El estudio también destaca la prevalencia de ambos problemas entre los estudiantes universitarios de todo el mundo, ya que sólo el 15 % de los estudiantes no muestra ningún nivel de adicción a las redes sociales. Los resultados se basan en una muestra de estudiantes de Lima y son relevantes para comprender el impacto de las redes sociales en el rendimiento académico. Este estudio proporciona evidencia de la relación significativa entre la adicción a las redes sociales y la procrastinación académica en estudiantes universitarios, lo cual es una preocupación creciente en la educación superior. Los hallazgos sugieren que abordar la adicción a las redes sociales puede ser una estrategia eficaz para reducir la procrastinación académica y mejorar los resultados de los estudiantes.

El artículo "La Generación Zombie. El uso excesivo de teléfonos celulares en las aulas universitarias peruanas"[18]. Analiza el uso excesivo de celulares por parte de estudiantes universitarios en las aulas de clase y sus efectos negativos en el aprendizaje y la salud. Se destaca que dos tercios de los estudiantes encuestados reconocen que el uso del celular en el aula afecta negativamente su aprendizaje al distraerlos y alejarlos de prestar atención a las actividades académicas. De igual manera resalta que casi el 70 % de los estudiantes está consciente de que el uso excesivo del celular es nocivo para la salud física y mental, generando adicción, problemas de concentración, afectando la interacción humana, etc. Por último, sugieren que las universidades deberían implementar políticas y estrategias para regular el uso de teléfonos móviles en las aulas, como horarios designados para el uso del teléfono y promover el uso responsable.

Adiba Orzikulova destaca los impactos negativos del uso excesivo de teléfonos inteligentes, particularmente en aplicaciones de redes sociales, en los estudiantes universi-

tarios [19]. El estudio sugiere que las intervenciones de autoayuda basadas en aplicaciones móviles pueden ser efectivas para prevenir el uso excesivo de teléfonos inteligentes y la adicción a los teléfonos inteligentes, pero se necesita más investigación para comprender los mecanismos subyacentes de la adicción a los teléfonos inteligentes y el impacto de estas intervenciones en el comportamiento de los estudiantes. El estudio también enfatiza la importancia de la autorregulación y sugiere que las barreras físicas pueden ser más efectivas para prevenir la adicción a los teléfonos inteligentes y promover la autorregulación que las intervenciones basadas únicamente en software.

2.3. Marco conceptual.

Aprendizaje autodirigido. Es un proceso de aprendizaje en el que el estudiante lleva las riendas de su propio proceso educativo. Implica que el alumno identifica de forma autónoma sus necesidades y objetivos de aprendizaje, busca y selecciona los recursos y estrategias más adecuados, y evalúa por sí mismo sus logros y resultados. Requiere que el estudiante emplee habilidades de autorregulación, como estrategias cognitivas, metacognitivas y de motivación personal, ya sea trabajando de manera independiente o con cierta guía externa. En esencia, el aprendiz toma un papel activo y autorresponsable en la conducción de su aprendizaje [20].

Launcher Android. Un launcher en Android es una aplicación que permite personalizar la interfaz de usuario y la apariencia de la pantalla de inicio de un dispositivo móvil. Funciona como una capa de personalización sobre el sistema operativo Android, permitiendo al usuario modificar la disposición de iconos, widgets, fondos de pantalla y otros elementos visuales en la pantalla de inicio. Los launchers ofrecen opciones de personalización avanzadas, como cambiar el diseño de la pantalla de inicio, agregar efectos de transición, modificar los iconos de las aplicaciones y ajustar la organización de las aplicaciones [21].

Autocontrol. Capacidad de modular y controlar las propias acciones de una forma apropiada a la edad de la persona. Es considerado uno de los componentes clave de la inteligencia emocional que debe ser reeducado en los estudiantes. El autocontrol implica tener una sensación de control interno sobre el propio cuerpo, conducta y entorno, permitiendo regular los impulsos y actuar de manera intencionada para lograr los objetivos deseados. Se plantea que el desarrollo del autocontrol desde la infancia constituye una facultad fundamental en el ser humano para tener una voluntad sólida y capacidad de autogobernarse [22].

Rendimiento académico. El rendimiento académico se refiere a la evaluación y medición de los conocimientos y habilidades adquiridos por un estudiante durante su formación académica, ya sea en niveles escolares, terciarios o universitarios. Se considera que un alumno tiene un buen rendimiento académico cuando obtiene calificaciones positivas y aprobatorias en los exámenes y evaluaciones que rinde a lo largo de los ciclos o períodos académicos correspondientes [23].

Concentración. La concentración es el enfoque voluntario de la mente hacia un objetivo específico, tarea o actividad, excluyendo cualquier distracción o interferencia que pueda surgir. Es un proceso psíquico que se lleva a cabo mediante el razonamiento, donde

se dirige toda la atención hacia un punto determinado, ya sea una tarea en curso o algo en lo que se está pensando [24].

Frustración. La frustración es un estado emocional caracterizado por sentimientos de decepción, irritabilidad o insatisfacción que surgen cuando una persona experimenta obstáculos o dificultades para alcanzar sus metas o satisfacer sus necesidades. Se manifiesta como una respuesta emocional negativa frente a la percepción de que los esfuerzos realizados no han dado los resultados deseados. La frustración puede surgir en diversas situaciones de la vida cotidiana, como problemas laborales, académicos, personales o sociales, y puede tener efectos adversos en el bienestar emocional y el comportamiento de la persona afectada [25].

3. Alcance.

3.1. Declaración del alcance.

El alcance del presente trabajo de grado consiste en el desarrollo de un launcher Android personalizado para estudiantes de la Universidad del Valle en Tuluá que incorpore las siguientes funcionalidades:

1. **Diseño minimalista:** Contará con una interfaz sin íconos de aplicaciones para evitar desviar la atención, en tonos neutrales y con atajos útiles para gestionar las demás características presentes en el launcher.

2. **Seguimiento del tiempo de uso de las aplicaciones:** El tiempo que se pasa en ciertas aplicaciones seleccionadas por el estudiante se podrá regular y monitorear con el fin de registrar progreso o áreas de mejora.

3. **Gestión de tareas y hábitos:** El launcher permitirá consignar las tareas que se deseen realizar y crear hábitos en función de tareas, tiempo límite o chequeo regular del hábito en cuestión.

4. **Herramienta(s) de productividad:** Se integrará al menos una herramienta de productividad para apoyar la realización de las tareas o los hábitos, como técnicas de distribución del tiempo de sesiones de trabajo, priorización de tareas, entre otras.

3.2. Supuestos.

1. Correcto funcionamiento de los equipos donde se llevará a cabo el desarrollo y las pruebas del proyecto.

2. Normal desarrollo de los periodos académicos.

3. Continuidad laboral del director de trabajo de grado.

3.3. Restricciones.

1. El proyecto se debe llevar a cabo en un plazo máximo de ocho (8) meses.

2. Solo estará disponible para el sistema operativo Android.

3. Será diseñado para el tamaño y la interfaz de un smartphone, no para tablets u otros dispositivos similares.

4. El proyecto y anteproyecto deben ser aprobados.

4. Objetivos.

4.1. Objetivo general.

Desarrollar un launcher Android personalizado para regular el uso del Smartphone y aumentar la productividad de los estudiantes de la Universidad del Valle en Tuluá.

4.2. Objetivos específicos.

1. Definir los requisitos funcionales de una pantalla de inicio para Android.
2. Diseñar la arquitectura interna e interfaces gráficas de la aplicación.
3. Integrar el diseño y las funcionalidades en la aplicación de acuerdo con los requisitos establecidos.
4. Implementar pruebas unitarias y de usabilidad del launcher en un sistema Android.

4.3. Resultados esperados.

Tabla 1: Resultados esperados

Objetivo específico	Resultados esperados
1. Definir los requisitos funcionales de una pantalla de inicio para Android.	Documento donde se describan las secciones y herramientas que incorporará, lenguajes y tecnologías a emplear. Ver capítulo 5.
2. Diseñar la arquitectura interna e interfaces gráficas de la aplicación.	Diseño de las vistas de la aplicación con sus elementos e interacciones como un prototipo; especificación de la arquitectura que se usará para la estructuración del código y los componentes del launcher. Ver capítulo 6.
3. Integrar el diseño y las funcionalidades en la aplicación de acuerdo con los requisitos establecidos.	Código fuente del launcher con el diseño y las características definidas con anterioridad. Ver capítulo 7.
4. Implementar pruebas unitarias y de usabilidad del launcher en un sistema Android.	Informe de las pruebas a nivel de código y de experiencia de usuario del launcher instalado en un dispositivo Android. Ver capítulo 8.

5. Metodología.

5.1. Tecnologías empleadas.

En el panorama actual del desarrollo móvil, los desarrolladores se enfrentan a una decisión fundamental entre el desarrollo nativo y multiplataforma. Con el crecimiento exponencial del mercado de aplicaciones móviles y la demanda de experiencias de usuario cada vez más sofisticadas, la elección tecnológica se convierte en un factor determinante para el adecuado desarrollo del proyecto. Así pues, se realizó una evaluación de las tecnologías disponibles, considerando las necesidades específicas y las tendencias actuales de la industria.

5.1.1. Desarrollo nativo vs. multiplataforma.

Se optó por el desarrollo nativo de la aplicación ya que ofrece mejor rendimiento, fácil acceso a todos los recursos del smartphone y está enfocado a un sistema operativo en particular (Android). Los frameworks multiplataforma actuales también ofrecen buen rendimiento e integración con las herramientas de desarrollo de Android, pero son más propensos a tener menor rendimiento, usar más recursos del sistema y generar fallos debido a las diferencias entre iOS y Android.

Esta decisión se fundamenta en la naturaleza específica del launcher, que requiere integración profunda con el sistema operativo Android para gestionar aplicaciones, controlar tiempos de uso y proporcionar una experiencia de usuario fluida y responsiva. Los resultados la comparativa se consignaron en la Tabla 2.

5.1.2. Versión objetivo de la API de Android.

La elección de Android 10 (API 29) como versión mínima para el desarrollo de Dino Launcher se fundamenta en la convergencia entre las funcionalidades que introduce esta versión del sistema operativo, los requisitos específicos del launcher desarrollado y la posibilidad de que pueda ser ejecutado en dispositivos no tan recientes de manera satisfactoria. En la figura 1, los datos de distribución de Android 10 alcanzan el 81.2 % de los dispositivos en el mercado, lo que garantiza una amplia compatibilidad con los dispositivos de los usuarios objetivo.

Esta versión introduce características esenciales que hacen posible una de las funcionalidades centrales del launcher, la introducción nativa del **modo oscuro**. Esta permite al launcher mostrar automáticamente los modos claro y oscuro dependiendo de la configuración del tema del sistema, reduciendo significativamente la fatiga visual durante el uso del launcher y mejorando la experiencia de usuario, especialmente durante las horas nocturnas cuando el control del tiempo de pantalla es más crítico.

5.1.3. Selección del lenguaje: Kotlin.

Las dos alternativas principales para el desarrollo nativo en Android son Java y Kotlin. Aunque Java es el lenguaje tradicional para el desarrollo Android, ha perdido terreno

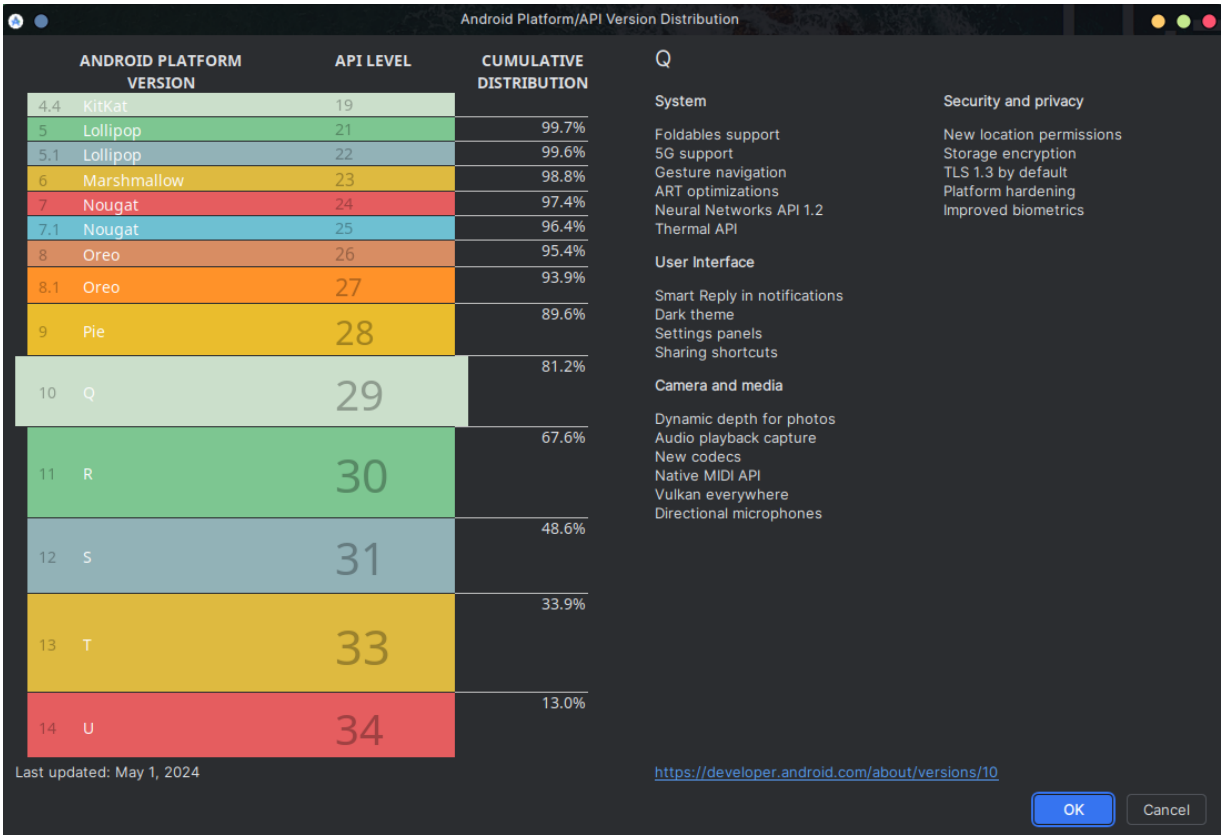
Tabla 2: Comparación entre desarrollo nativo y multiplataforma para Android

Aspecto	**Desarrollo Nativo Android**	**Desarrollo Multiplataforma**
Lenguaje de programación	Kotlin o Java	JavaScript (React Native), Dart (Flutter), C# (Xamarin)
Rendimiento	Óptimo, optimizado específicamente para Android	Bueno, pero generalmente inferior debido a la capa de abstracción
Acceso a funcionalidades	Acceso total a todas las APIs y hardware específicos	Acceso limitado, requiere plugins para funcionalidades específicas
Experiencia de usuario	Adaptación completa a Material Design	Puede no seguir completamente las directrices de Android
Tiempo de desarrollo	Puede ser más largo debido a codificación específica	Más corto, código compartido entre plataformas
Costos de desarrollo	Más altos para Android únicamente, pero justificados	Más bajos si se desarrolla para múltiples plataformas
Mantenimiento	Simplificado, enfoque exclusivo en Android	Más complejo para compatibilidad específica
Actualizaciones de SO	Implementación inmediata con nuevas versiones	Depende de actualizaciones del framework
Reutilización de código	Nula entre plataformas, alta en proyectos Android	Alta reutilización multiplataforma
Compatibilidad	Total, diseño específico para Android	Buena, pero con posibles inconsistencias

gracias a Kotlin y sus mejoras significativas con respecto a Java para el desarrollo móvil, principalmente en cuanto a sintaxis y características modernas. Según la encuesta anual de desarrolladores de Stack Overflow de mayo de 2024 [26], los desarrolladores que trabajan con Kotlin se sienten más cómodos con el lenguaje, al contrario de lo que se observa en Java, donde varios de los encuestados preferirían trabajar con Kotlin. La selección de Kotlin se fundamenta en los siguientes aspectos:

- Google, propietario de Android, recomienda Kotlin para cualquier proyecto nuevo de Android y declaró que construiría sus herramientas de desarrollo con un enfoque Kotlin-first desde la conferencia Google I/O en 2019 [27].
- Es un lenguaje más fácil de entender por su sintaxis simplificada y la reducción de código repetitivo (*boilerplate*), reduciendo el tiempo de aprendizaje.
- Es un lenguaje activamente desarrollado y adaptado a las necesidades actuales de la industria.
- Tiene una gran comunidad y cantidad de recursos disponibles.

Figura 1: Distribución de versiones de Android a nivel mundial. [1]



- Ofrece características modernas como corrutinas, funciones de extensión y clases de datos que facilitan el desarrollo de aplicaciones complejas.

La Tabla 3 resume las principales diferencias entre Kotlin y Java, destacando las ventajas de Kotlin para el desarrollo del launcher.

5.1.4. Librería de UI: Jetpack Compose.

Jetpack Compose es el toolkit de UI más reciente y recomendado por Google para el desarrollo de interfaces en Android desde 2021, marcando un cambio paradigmático hacia la programación declarativa. Compose permite construir interfaces de usuario de manera más intuitiva, eficiente y mantenible en comparación con el sistema tradicional basado en vistas XML. Su adopción se ha consolidado como el estándar para nuevos proyectos Android, facilitando la creación de experiencias visuales modernas y adaptables.

El aspecto más relevante de Jetpack Compose es su integración nativa con Kotlin, lo que permite aprovechar plenamente las características modernas del lenguaje como las funciones lambda, la programación funcional y la sintaxis concisa, mejorando la legibilidad y mantenibilidad del código y reduciendo la cantidad de código necesario para construir interfaces. En la Figura 2, se muestra un ejemplo de código en Jetpack Compose que ilustra su simplicidad y claridad en comparación con el enfoque tradicional de XML para

Tabla 3: Comparación entre Kotlin y Java para desarrollo Android

Aspecto	**Kotlin**	**Java**
Año de lanzamiento	2011	1995
Sintaxis	Concisa, moderna y más legible	Extensa, más detallada y tradicional
Interoperabilidad	Totalmente interoperable con Java	No es nativamente interoperable con Kotlin
Seguridad de tipos nulos	Evita NullPointerExceptions	NullPointerExceptions son comunes
Compatibilidad Android	Totalmente compatible, lenguaje oficial	Compatible pero no recomendado oficialmente
Características modernas	Lambdas, corrutinas, extension functions, data classes	Introducción más lenta de características modernas
Curva de aprendizaje	Relativamente fácil para desarrolladores Java	Relativamente fácil pero más verboso
Productividad	Alta, gracias a sintaxis concisa	Moderada, requiere más código para tareas similares
Soporte y comunidad	Creciente, especialmente en Android	Muy grande y establecida, décadas de documentación
Desempeño	Similar a Java, optimizaciones específicas	Similar a Kotlin, rendimiento comparable
Ecosistema de herramientas	Totalmente soportado en Android Studio	Amplio soporte en diversas IDEs

construir una interfaz que muestre las palabras *"Good" "Morning"* una arriba de la otra.

Figura 2: Ejemplo de código en Jetpack Compose

La Tabla 4 presenta una comparación detallada entre Jetpack Compose y el sistema tradicional de vistas XML, destacando las ventajas técnicas que justifican la adopción de Compose para el desarrollo del launcher.

Tabla 4: Comparación entre Jetpack Compose y vistas XML para desarrollo Android

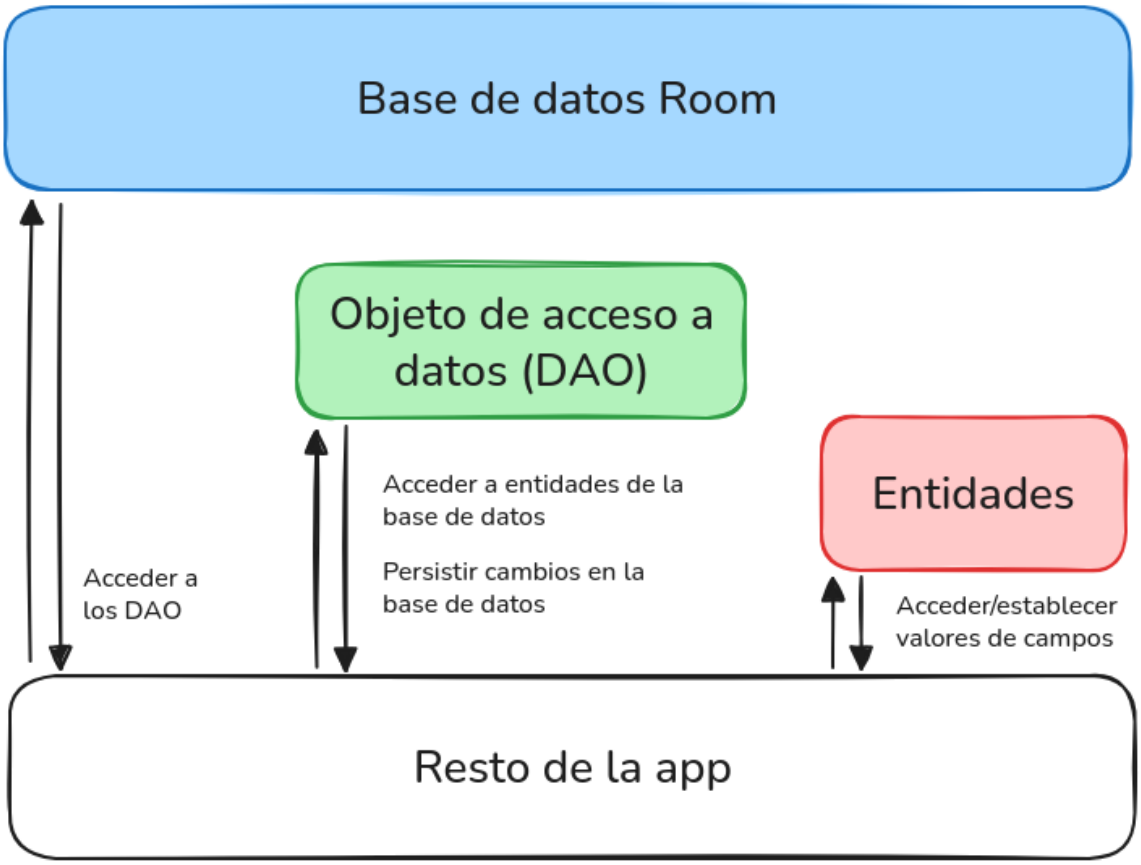
Aspecto	**Jetpack Compose**	**Vistas XML**
Paradigma de programación	Declarativo, describe qué mostrar	Imperativo, describe cómo construir
Lenguaje	100 % Kotlin, fuertemente tipado	XML + Kotlin/Java, tipos débiles
Gestión de estado	Recomposición automática con State	Actualización manual de vistas
Animaciones	APIs nativas integradas, declarativas	Requiere configuración compleja XML/código
Reutilización de código	Composición funcional, alta reutilización	Herencia de vistas, reutilización limitada
Curva de aprendizaje	Requiere conocimiento de programación funcional	Familiar para desarrolladores tradicionales
Rendimiento	Recomposición inteligente, optimizado	Inflado de layouts puede ser costoso
Depuración	Compose Inspector integrado	Layout Inspector tradicional
Tamaño de APK	Comparable o menor	Puede ser mayor con layouts complejos
Interoperabilidad	Completa con vistas XML existentes	No compatible con Compose sin wrappers
Soporte de temas	Material Design 3 nativo	Requiere configuración manual extensa
Testing	Testing declarativo con ComposeTestRule	Testing complejo de interacciones UI
Mantenimiento	Simplificado, lógica en un solo lugar	Complejo, lógica distribuida en XML y código
Adopción empresarial	Creciente, recomendado por Google	Establecido, pero en desuso gradual

5.1.5. Persistencia de datos: Room.

Room es una biblioteca de persistencia que forma parte de Android Jetpack [28] y proporciona una capa de abstracción sobre SQLite. Está diseñada específicamente para simplificar el trabajo con bases de datos en Android, combinando la potencia de SQLite con la comodidad y seguridad de tipos del desarrollo en Kotlin [2]. Entre las principales ventajas de utilizar Room en el desarrollo Android, se encuentran la posibilidad de observar cambios en los datos de manera reactiva con LiveData y Flow, facilitando la actualización automática de la interfaz de usuario; el soporte nativo para corrutinas, que permiten la

ejecución de manera asíncrona utilizando *suspend functions*, evitando bloqueos en el hilo principal y el uso de anotaciones para reducir código repetitivo (como `@Entity`, `@Dao` o `@Database`) para generar automáticamente el código necesario y realizar operaciones en SQLite. En la figura 3, se muestra el flujo de datos de Room entre la base de datos y la aplicación desarrollada.

Figura 3: Flujo de datos de Room. [2]



La selección de Room como solución de persistencia de datos por encima de alternativas basadas en la nube, como Firebase, se fundamenta en las características específicas del launcher y sus requerimientos funcionales en la sección 5.2. El proyecto no requiere que los datos estén disponibles en múltiples dispositivos o se sincronicen en tiempo real, ya que el launcher está diseñado para funcionar de manera individual en cada dispositivo. Además, el tipo de información que maneja la aplicación no demanda características de red en tiempo real ni almacenamiento de datos complejos o de gran tamaño. La naturaleza personal y privada de estos datos hace innecesaria la implementación de sistemas de autenticación de cuentas de usuario o gestión de perfiles en la nube, simplificando la arquitectura del sistema y minimizando el uso de recursos que puedan llegar a consumir las funciones de red.

5.2. Levantamiento de requerimientos.

Se utilizaron múltiples técnicas de recolección de información que representaran de una manera clara las acciones a realizar para cumplir los objetivos, desde la teoría hasta la manera en cómo se interactúa con los dispositivos móviles. A continuación, se detallan las técnicas empleadas y los resultados obtenidos:

Lluvia de ideas (Brainstorming): Se recurrió a la técnica de brainstorming para generar un amplio abanico de ideas sobre las funcionalidades que el launcher podría ofrecer. A través de sesiones creativas, se exploraron diversas posibilidades y se identificaron las características más importantes que podrían contribuir a mejorar la productividad de los estudiantes. Esta técnica permitió establecer el conjunto inicial de funcionalidades como la gestión de tareas, gestión de hábitos y control de tiempo de uso de aplicaciones, en conjunto con sus particularidades a nivel de desarrollo y diseño.

Observación: Al analizar la interacción de los estudiantes con sus dispositivos móviles en entornos académicos, especialmente de la Universidad del Valle en Tuluá, se identificó que muchos de ellos utilizan aplicaciones de mensajería, redes sociales y juegos como una forma de distracción y abren dichas aplicaciones muchas veces por mera memoria muscular. Esta observación llevó a la conclusión de que un launcher que si, a nivel visual, reduce los estímulos que faciliten la detección de dichas aplicaciones, puede contribuir a que sean usadas con menor frecuencia. Es por esto que se definió que la interfaz debía ser minimalista, dejando de lado los íconos que caracterizan a cada aplicación por considerarse un disparador que apunta hacia el inmediato uso de la misma. Este proceso también reafirmó la necesidad de implementar un limite de tiempo a aplicaciones que el usuario identifique que necesita regular.

Análisis de documentación: Se revisaron cuatro artículos más relacionados con la procrastinación académica, los cuales sirvieron de apoyo para entender mejor el contexto en el que se desarrollaría el launcher:

Durante el análisis se identificaron patrones de comportamiento que van más allá del simple uso de dispositivos móviles. Los estudios revisados revelaron que existe una relación negativa entre la procrastinación académica y las intenciones de llevar a cabo conductas saludables, asociada con una menor autoeficacia específica de la salud y bajo control conductual percibido [29]. Se encontró evidencia de una correlación directa de intensidad moderada a fuerte entre la postergación de actividades académicas y la dependencia al dispositivo móvil, siendo especialmente notable en estudiantes universitarios donde el 59,3 % presenta niveles moderados de dependencia [30].

La investigación también demostró que el uso problemático del smartphone va más allá de las aplicaciones de mensajería y redes sociales, extendiéndose a lo que se denomina "procrastinación electrónica", que abarca el uso excesivo de computadoras, videojuegos, televisión, películas y consumo de noticias. Esta forma de procrastinación resulta particularmente seductora debido a la accesibilidad constante que proporciona internet, permitiendo la distracción en cualquier momento del día [31].

Los hallazgos confirman que a mayor uso problemático del smartphone, mayor es la tendencia a procrastinar en los estudiantes, y dado que este uso parece estar relacionado con un bajo autocontrol, se identificó la necesidad de implementar programas de inter-

vención relacionados con la resiliencia para controlar el uso del smartphone y mejorar la gestión del tiempo [32].

5.2.1. Requerimientos funcionales.

Los requerimientos funcionales identificados se organizaron en las siguientes categorías principales:

1. **Lanzador de aplicaciones:** Funcionalidad principal del launcher que mostrará la lista de aplicaciones disponibles para su ejecución. Incluirá gestión básica de aplicaciones mediante pulsación prolongada para desinstalar, acceder a información de la aplicación o fijar aplicaciones en el escritorio.

2. **Accesos rápidos esenciales:** Sistema de accesos directos para aplicaciones indispensables como correo electrónico,cámara o teléfono en una barra lateral siempre visible en la pantalla principal.

3. **Búsqueda de apliciones:** Barra de búsqueda que mostrará aplicaciones que coincidan con el texto ingresado en el menú.

4. **Control de tiempo de uso:** Funcionalidad para establecer límites de tiempo en aplicaciones específicas. Una vez agotado el tiempo permitido, el usuario no podrá abrir la aplicación restringida desde el launcher.

5. **Gestión de tareas:** Sistema tipo to-do que permitirá asignar fechas específicas a las tareas. Las tareas sin fecha asignada aparecerán diariamente hasta su completación. Incluirá sistema de etiquetas para clasificación y organización.

6. **Gestión de hábitos:** Implementación de tareas recurrentes programadas para días específicos de la semana, con fechas de inicio y fin definidas. Permitirá marcado de completitud similar al sistema de tareas.

7. **Integración de pomodoro:** Herramienta de productividad configurable que permitirá establecer número de sesiones, duración de sesiones de trabajo y tiempos de descanso.

5.2.2. Requerimientos no funcionales.

Los requerimientos no funcionales establecidos para el proyecto incluyen:

- **Compatibilidad:** El launcher es compatible exclusivamente con el sistema operativo Android, diseñado específicamente para smartphones.

- **Rendimiento:** Debe ofrecer un rendimiento óptimo aprovechando las ventajas del desarrollo nativo en Android.

- **Usabilidad:** Interfaz minimalista que reduzca los elementos distractores, siguiendo las directrices de Material Design de Android.

- **Mantenibilidad:** Código estructurado, teniendo en cuenta las mejores prácticas de desarrollo que se usan actualmente para facilitar futuras actualizaciones y mejoras.

5.3. Scrumban: Implementación.

La gestión eficiente del proceso de desarrollo constituye un factor determinante en el éxito de cualquier proyecto de software. Para este launcher, se adoptó el framework *Scrumban*, una metodología híbrida que combina la estructura organizacional de Scrum con la flexibilidad visual de Kanban. Esta elección se fundamenta en la necesidad de mantener un desarrollo ágil y adaptativo, característico de Scrum, mientras se aprovecha la organización y visualización coherente de actividades que proporciona el tablero Kanban.

Scrumban resulta particularmente adecuado para proyectos individuales, donde la flexibilidad para ajustar prioridades y la capacidad de visualizar el flujo de trabajo son más importantes que las ceremonias tradicionales de Scrum.

5.3.1. Sprints.

El proceso de desarrollo se estableció en sprints de **dos (2) semanas** de duración, considerándose un ritmo sostenible que permitió la planificación detallada sin comprometer la flexibilidad del proceso. El cronograma de desarrollo se dividió en dos períodos académicos: la fase inicial comprendió desde el 16 de agosto hasta el 19 de diciembre de 2024, seguida de una pausa académica y la posterior reanudación el 21 de febrero de 2025, concluyendo el 29 de mayo de 2025. En total se ejecutaron **dieciseis (16) sprints** que abarcaron desde la conceptualización inicial hasta la implementación completa del launcher.

Los primeros sprints se dedicaron al modelado conceptual de la aplicación y la definición de requerimientos funcionales y no funcionales. Posteriormente, se desarrollaron los mockups y prototipos de interfaz con el objetivo de tener una base visual y de experiencia de usuario sólida para la siguiente etapa. Finalmente, los sprints restantes se enfocaron en la implementación del launcher utilizando Android Studio y las tecnologías previamente seleccionadas.

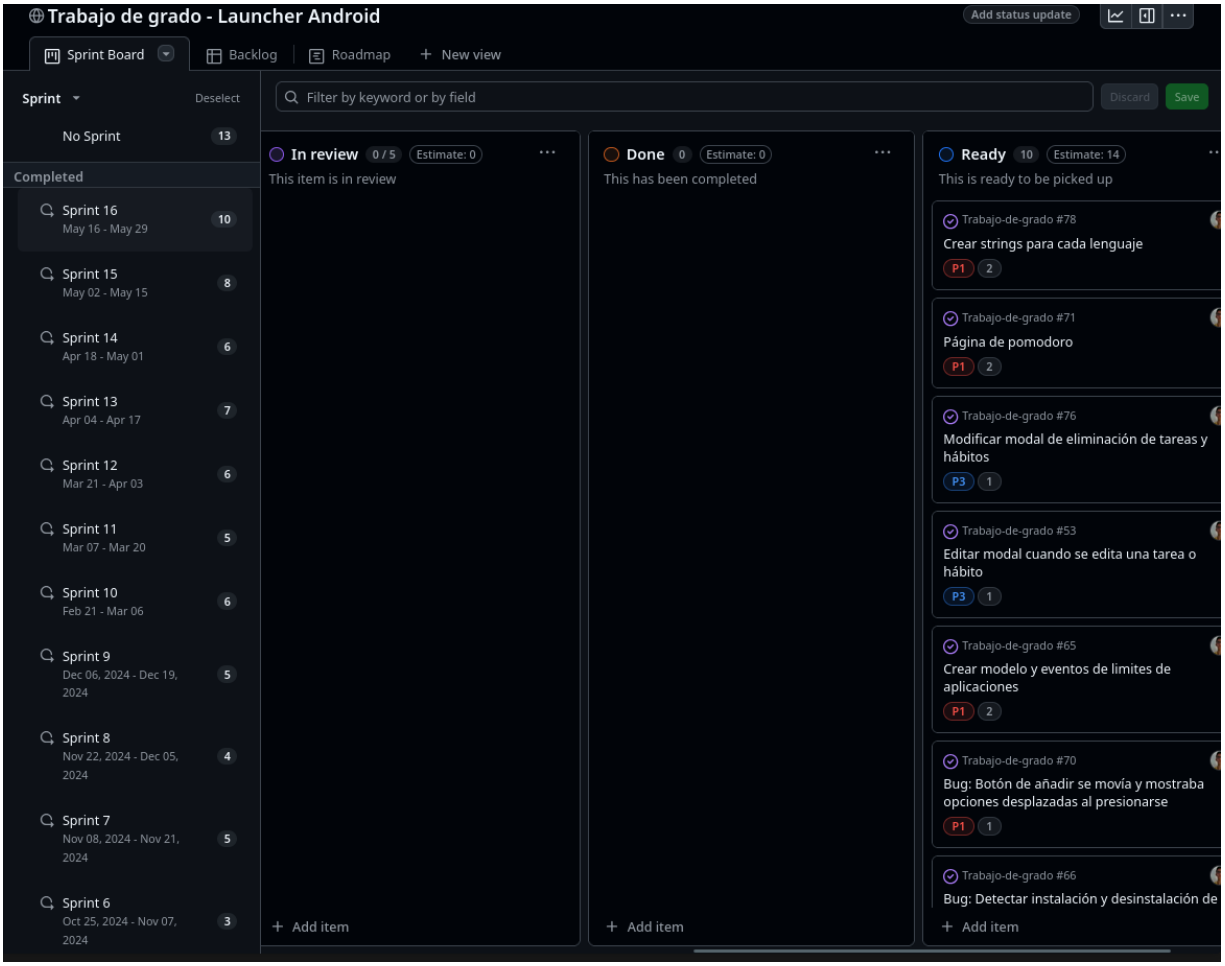
5.3.2. Gestión de actividades: GitHub Projects.

GitHub Projects fue seleccionada como plataforma de gestión de actividades debido a que ofrece una integración y comunicación directa con el repositorio de la aplicación alojado en GitHub y es extremadamente sencilla de utilizar y configurar para diferentes metodologías o tipos de proyectos. La configuración implementada en GitHub Projects incluyó la selección de la plantilla Kanban para reflejar el flujo de trabajo deseado. Se establecieron columnas que representan los estados del trabajo: **Backlog, En progreso, En revisión, Completado y Listo**, como se muestra en la Figura 4.

Las historias de usuario se consignaron en el backlog de GitHub Projects al inicio del proyecto y cada vez que se requería corregir un error o agregar una funcionalidad que no estaba contemplada al inicio del proyecto. Cada historia siguió la estructura estándar de metodologías ágiles:

```
Como [tipo de usuario],
   quiero [acción]
          para [beneficio].
```

Figura 4: Tablero Kanban de GitHub Projects.



Para historias de mayor complejidad o con aspectos técnicos específicos, se incluyeron criterios de aceptación detallados que establecen las condiciones que deben cumplirse para considerar la funcionalidad como completamente implementada. En la Figura 5, Figura 6 y Figura 7 se muestran algunos ejemplos de historias de usuario. Para ver la lista completa de historias de usuario, consultar el proyecto en `https://github.com/users/Juansebas064/projects/3/views/6`.

5.3.3. GitHub Flow.

GitHub Flow es una metodología de control de versiones y colaboración en desarrollo de software que se caracteriza por su simplicidad y enfoque en la entrega continua. Desarrollada por GitHub como una alternativa más ágil y menos compleja que *Git Flow*, esta metodología se fundamenta en un flujo de trabajo lineal que prioriza la rapidez de integración y el despliegue frecuente de cambios. La filosofía central de GitHub Flow se basa en mantener la rama principal (main) siempre en un estado desplegable, lo que significa que cualquier código que se integre a esta rama debe estar completamente funcional y listo

Figura 5: Ejemplo 1: Historia de usuario.

para producción.

Se buscó que la gestión del proyecto fuera simple pero con una estructura bien definida, este flujo de trabajo se adaptó perfectamente a las necesidades de estandarizar la gestión de ramas de *Git* y de integrar de manera continua los cambios con la rama principal. El proceso de GitHub Flow se estructura en seis pasos: **Primero**, se crea una nueva rama a partir de la rama principal para cada nueva funcionalidad, corrección de errores o mejora que se desee implementar. Esta rama debe tener un nombre descriptivo (ejemplo: `apps-limit`) que refleje claramente el propósito del trabajo a realizar.

Segundo, se realizan los commits necesarios en esta rama de trabajo, manteniendo un historial claro y conciso de los cambios implementados.

Tercero, se abre un *Pull Request (PR)* hacia la rama principal, iniciando el proceso de revisión de código. El PR actúa como mecanismo de documentación y discusión sobre los cambios propuestos y protege a la rama principal de ser directamente modificada.

Cuarto, se lleva a cabo la revisión del código en el PR, donde otros miembros del equipo (en el caso de haber varios) evalúan la calidad, funcionalidad y adherencia a los estándares establecidos.

Quinto, una vez que el PR ha sido aprobado y todas las verificaciones automáticas han pasado exitosamente, se procede a la integración del código en la rama principal mediante un *merge*.

Sexto, inmediatamente después de la integración, se procede al despliegue de los cambios en el ambiente de producción, aprovechando que la rama principal se mantiene siempre en estado desplegable (en este caso, la instalación en el dispositivo Android).

En la Figura 8 podemos ver algunas de las ramas que se utilizaron a lo largo del desarrollo del launcher.

Figura 6: Ejemplo 2: Historia de usuario.

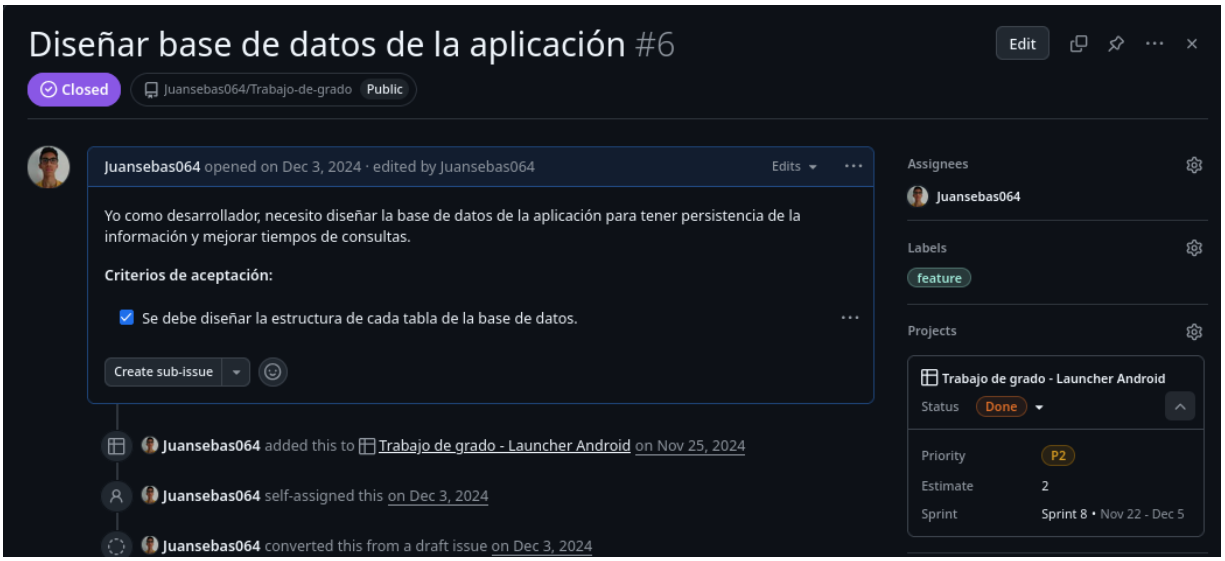


Figura 7: Ejemplo 3: Historia de usuario.

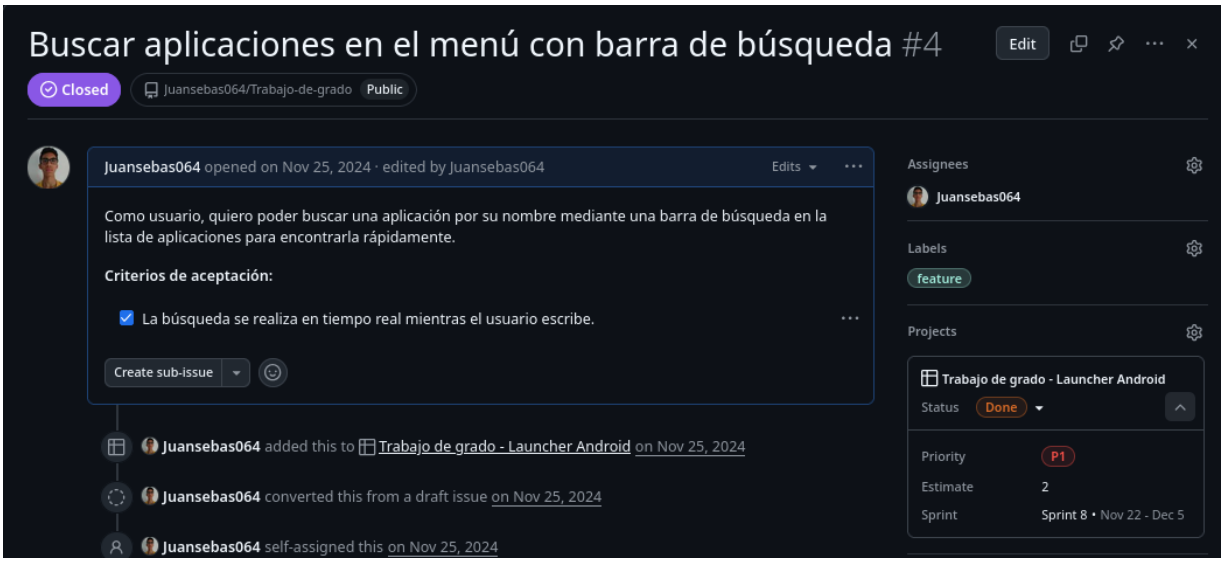
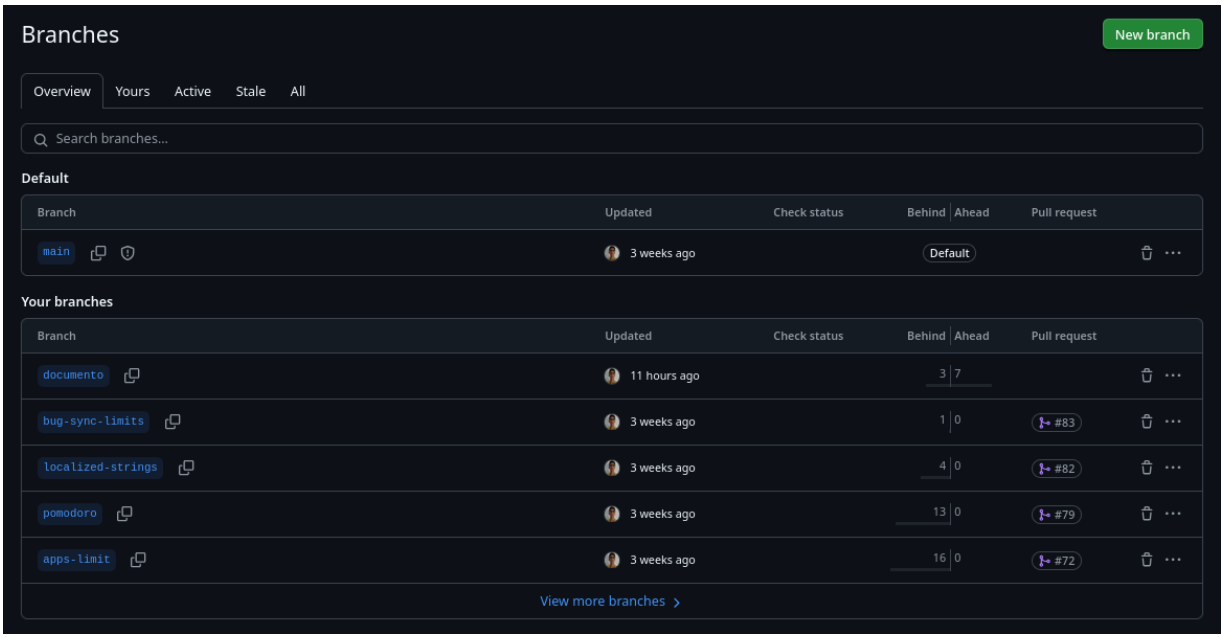


Figura 8: Gestión de ramas con GitHub Flow.



6. Arquitectura y diseño.

6.1. Arquitectura.

6.1.1. Prácticas recomendadas por Google.

El desarrollo de aplicaciones Android modernas requiere de una arquitectura clara, modular y escalable, que facilite el mantenimiento del código, la incorporación de nuevas funcionalidades, así como la mejora continua del producto final. Google establece principios fundamentales para el desarrollo de aplicaciones Android robustas y mantenibles a través de la plataforma Android Jetpack [28], que han evolucionado a partir de años de experiencia en el ecosistema móvil. Estas prácticas se centran en la separación clara de responsabilidades a través de una *arquitectura basada en capas*, donde cada componente del sistema tiene un propósito específico y bien definido [33].

La arquitectura recomendada para aplicaciones Android se basa principalmente en tres capas fundamentales:

- **Capa de Presentación (UI):** Encargada exclusivamente de la interacción visual y la experiencia del usuario. Esta capa se ocupa de mostrar la información, escuchar las acciones del usuario y reaccionar a los cambios en el estado de la aplicación. No contiene lógica de negocio ni interacción directa con fuentes de datos, garantizando así una clara separación de responsabilidades.

- **Capa de Dominio (opcional, pero recomendada en proyectos de mediana o gran escala):** Constituye el núcleo lógico del sistema, implementando casos de uso específicos relacionados directamente con las reglas del negocio. Al aislar esta lógica en una capa independiente, se favorece la reutilización de código, la claridad en la implementación y se simplifican significativamente las pruebas unitarias.

- **Capa de Datos:** Responsable de acceder a las fuentes de información, tanto locales como remotas. Utiliza patrones de diseño como el repositorio, el cual se encarga de gestionar una fuente única de verdad para los datos, manteniendo copias locales y actualizando la información desde fuentes externas según sea necesario.

Dentro de estas prácticas es especialmente importante destacar el patrón conocido como Flujo Unidireccional de Datos *(Unidirectional Data Flow, UDF)*. En el contexto específico de Android, los datos generalmente fluyen desde elementos con un alcance amplio, como los repositorios y fuentes de datos, hacia elementos con un alcance más limitado, como los componentes visuales de la UI. Por otro lado, los eventos generados por el usuario, como pulsaciones de botones o gestos, fluyen en sentido contrario desde la interfaz gráfica hacia la fuente única de verdad, donde los datos se modifican y actualizan posteriormente, siempre a través de estructuras de datos inmutables.

Este enfoque está estrechamente relacionado con la correcta gestión del ciclo de vida de los componentes Android. El conocimiento explícito y el manejo consciente del ciclo de vida permiten que los componentes reaccionen apropiadamente frente a eventos generados por el propio sistema operativo, como la rotación de pantalla o la suspensión temporal

de la aplicación. Herramientas proporcionadas por Android Jetpack, como *ViewModel*, y bibliotecas reactivas como *LiveData* o *Flow*, son clave en este proceso, pues reaccionan de forma automática y segura a los cambios de estado en las actividades y fragmentos.

Google también recomienda emplear bibliotecas especializadas como *Hilt*, para gestionar las dependencias entre componentes. La inyección de dependencias automatiza la creación de objetos en tiempo de compilación sin necesidad de que cada clase los construya. En su lugar, una clase externa se encarga de proveer las dependencias requeridas, lo que simplifica la gestión de dependencias, centraliza las instancias de los objetos y mejora la mantenibilidad del código.

6.1.2. Patrón de arquitectura MVVM.

El patrón de arquitectura elegido para el desarrollo del launcher fue *Model-View-ViewModel* (*MVVM*, por sus siglas en inglés), ya que su principal objetivo es separar la lógica de negocio de la interfaz de usuario, obedeciendo a las prácticas recomendadas por Google para el desarrollo de aplicaciones Android [33]. Aunque originalmente fue diseñado por Microsoft en 2005 para aplicaciones de escritorio basadas en eventos, con el tiempo ha sido cada vez más implementado en el desarrollo de aplicaciones móviles, especialmente con la introducción de *Android Architecture Components* en la conferencia de Google I/O en 2017 [34]. Está inspirado en el patrón *Model-View-Presenter* (*MVP*) y *Model-View-Controller* (*MVC*), manteniendo la separación de responsabilidades e introduciendo un enfoque más reactivo y menos acoplado entre la vista y el modelo. Las principales diferencias entre estos patrones arquitectónicos se presentan en la Tabla 5.

El patrón MVVM se compone de tres componentes principales:

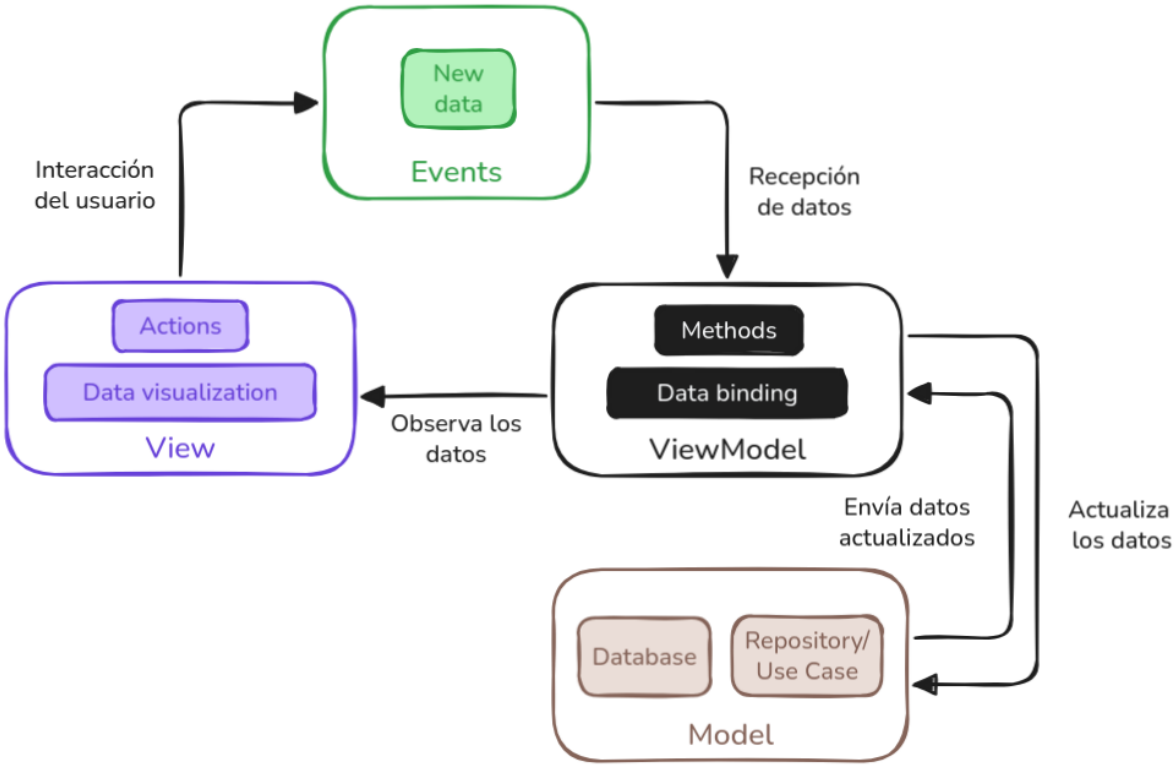
- **Model:** Representa la lógica de negocio y los datos de la aplicación. En el launcher, Model incluye las clases que gestionan las tareas, hábitos, límites y preferencias del usuario.
- **View:** Es la interfaz de usuario que muestra los datos al usuario y recibe sus interacciones. Esta capa está implementada utilizando Jetpack Compose.
- **ViewModel:** Actúa como un intermediario entre Model y View. Contiene la lógica para preparar los datos que View necesita y maneja las interacciones del usuario. En el launcher, cada funcionalidad principal (tareas, hábitos, limites y preferencias) tiene su propio ViewModel.

Tabla 5: Comparación de patrones arquitectónicos: MVC, MVP y MVVM

Aspecto	**MVC (Model-View-Controller)**	**MVP (Model-View-Presenter)**	**MVVM (Model-View-ViewModel)**
Acoplamiento	View y Model están acoplados. Controller gestiona la interacción.	View y Model están completamente desacoplados por el Presenter.	View y ViewModel están débilmente acoplados a través de enlaces de datos (data binding).
Rol de View	Activa. Puede observar directamente a Model y recibir actualizaciones de Controller.	Pasiva. Implementa una interfaz y no contiene lógica. Presenter indica qué mostrar.	Reactiva. Se enlaza a propiedades de ViewModel y se actualiza automáticamente.
Comunicación	Controller manipula el Modelo. View puede observar al Modelo directamente.	View delega los eventos a Presenter. Presenter actualiza a View a través de una interfaz.	View se suscribe a los flujos de datos de ViewModel. Los eventos de View ejecutan funciones en el ViewModel.
Referencia a View	Controller tiene una referencia directa a View.	Presenter tiene una referencia a View, pero a través de una interfaz abstracta.	ViewModel no tiene ninguna referencia a View. La comunicación es unidireccional (View observa a ViewModel).
Testabilidad	Difícil de probar de forma aislada debido al acoplamiento entre View y Controller.	Alta. Presenter es independiente de la UI de Android y se puede probar fácilmente con mocks.	Muy alta. ViewModel es completamente independiente de View, lo que facilita las pruebas unitarias.

El flujo de datos en MVVM es unidireccional, lo que significa que View observa los cambios en el ViewModel y se actualiza automáticamente cuando los datos cambian. Esto se logra mediante el uso de bibliotecas reactivas como *Flow*, que permite a View suscribirse a los cambios en los datos y reaccionar a ellos sin necesidad de acoplarse directamente a Model. La figura 9 ilustra la relación entre estos componentes.

Figura 9: Arquitectura MVVM.



6.2. Diseño.

6.2.1. Aplicaciones similares.

En la fase inicial del proceso de diseño se llevó a cabo una revisión de aplicaciones existentes con enfoques similares al planteado, con el objetivo identificar funcionalidades relevantes y oportunidades de mejora que pudieran ser la base de las decisiones de diseño del proyecto. El análisis se concentró en tres launchers de código abierto que compartían la filosofía de minimalismo y funcionalidad enfocada en la productividad:

Hex Launcher [35] se destacó por su simplicidad visual y la organización de aplicaciones mediante categorías. Su interfaz limpia proporcionó indicadores sobre cómo reducir la sobrecarga visual sin comprometer la funcionalidad.

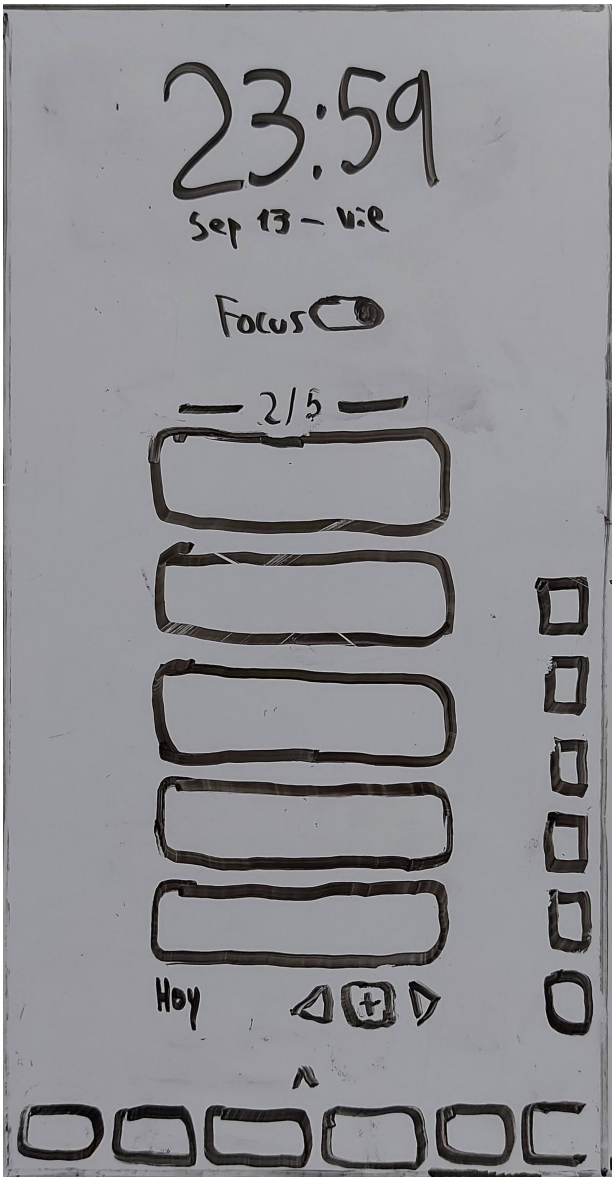
Launcher [36], por su parte, se enfoca en atajos que sean útiles para acceder a las aplicaciones. Su diseño minimalista y la ausencia de elementos decorativos innecesarios sirvieron como inspiración para la creación de una interfaz que prioriza la funcionalidad sobre la estética.

Olauncher [37], contribuyó especialmente en demostrar la efectividad de un diseño extremadamente minimalista centrado en listas textuales en lugar de iconos gráficos para las aplicaciones. Esta aproximación confirmó la viabilidad de eliminar los elementos visuales tradicionalmente asociados con el reconocimiento instantáneo de aplicaciones, reemplazándolos por una interfaz que fomenta el uso consciente e intencional del dispositivo.

6.2.2. Mockups y prototipo.

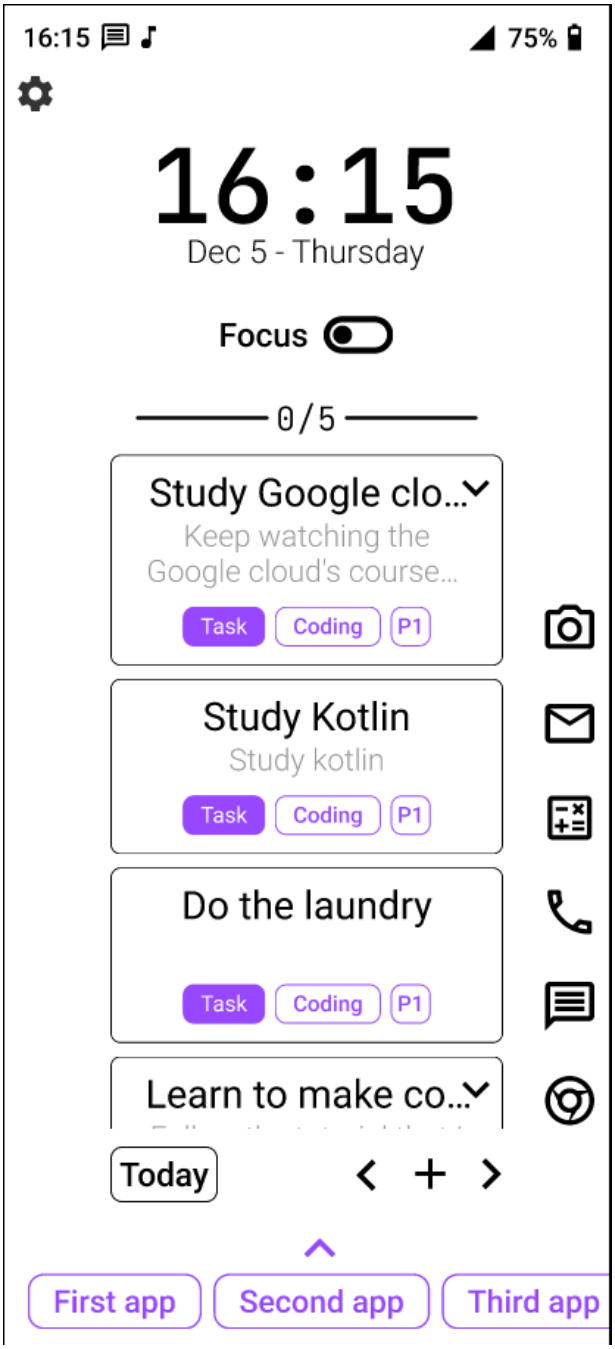
Luego de haber identificado los elementos deseados en la interfaz, se empezaron a diseñar wireframes de baja fidelidad utilizando un tablero físico con marcadores borrables, explorando ideas y cambiando rápidamente los elementos que necesitaran ajustes. Durante estas sesiones, surgieron diversas aproximaciones para la organización de la información, la distribución de elementos en pantalla y los flujos de navegación entre diferentes funcionalidades. Desde el principio se tuvo en cuenta la necesidad de darle protagonismo a las funcionalidades de gestión de tareas y hábitos, ocupando gran parte de la pantalla de inicio con la intención de que el usuario siempre tuviera visible sus pendientes.

Figura 10: Wireframe en tablero.



Una vez definidos los conceptos principales y validadas las ideas en el tablero físico, el proceso evolucionó hacia la creación de wireframes de alta fidelidad utilizando Figma. Esta transición permitió refinar los diseños con mayor precisión y establecer dimensiones específicas para los elementos de interfaz.

Figura 11: Wireframe de alta fidelidad.



La fase final del proceso de diseño consistió en la transformación de los wireframes de alta fidelidad en mockups prototipados que incorporaran todos los aspectos visuales del diseño final. Esta etapa incluyó la definición de la paleta de colores, tipografías, espaciados, proporciones, íconos, acciones y animaciones. Se buscó crear una interfaz que no solo fuera visualmente agradable, sino que también facilitara la interacción del usuario con las funcionalidades clave del launcher, como la gestión de tareas, hábitos y límites. Adicionalmente, se proyectó que la aplicación estuviera disponible en **español e inglés**,

implementación realizada durante el desarrollo del proyecto. Los mockups y su prototipo se encuentran en `https://www.figma.com/design/EbtSyFZ1OGdiSgzbMcXiMW/Mockups?node-id=214-3438&t=4SbJQopzfLf274FE-1`.

Figura 12: Mockup: Pantalla principal.

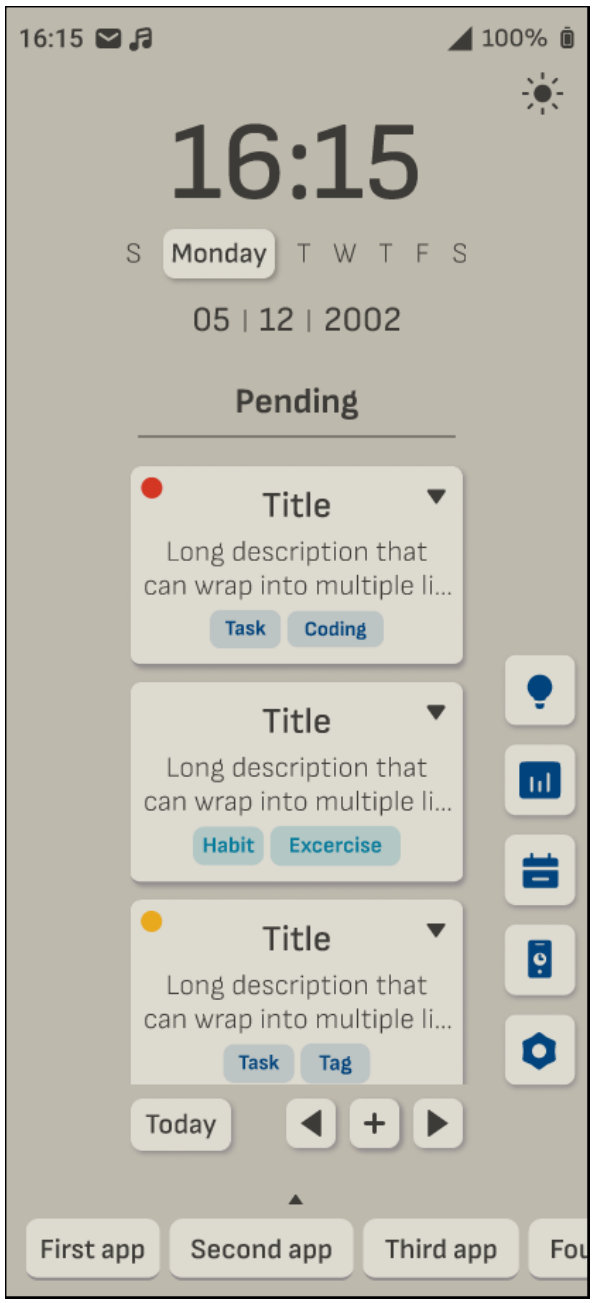


Figura 13: Mockup: Pomodoro.

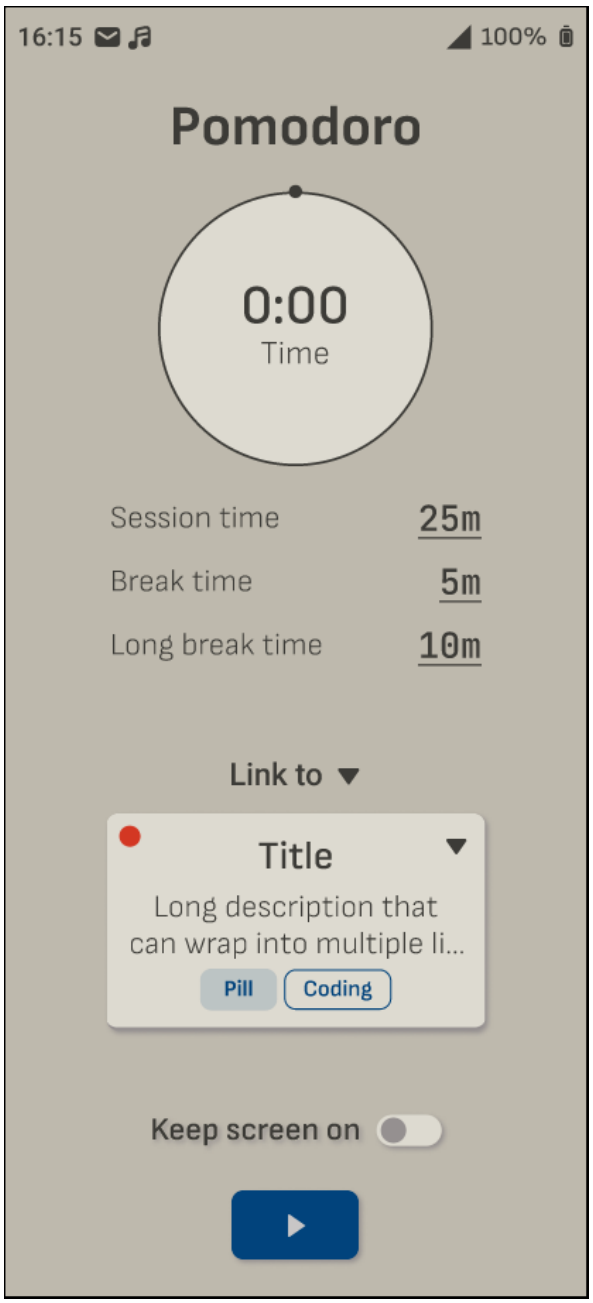


Figura 14: Mockup: Menú de aplicaciones.

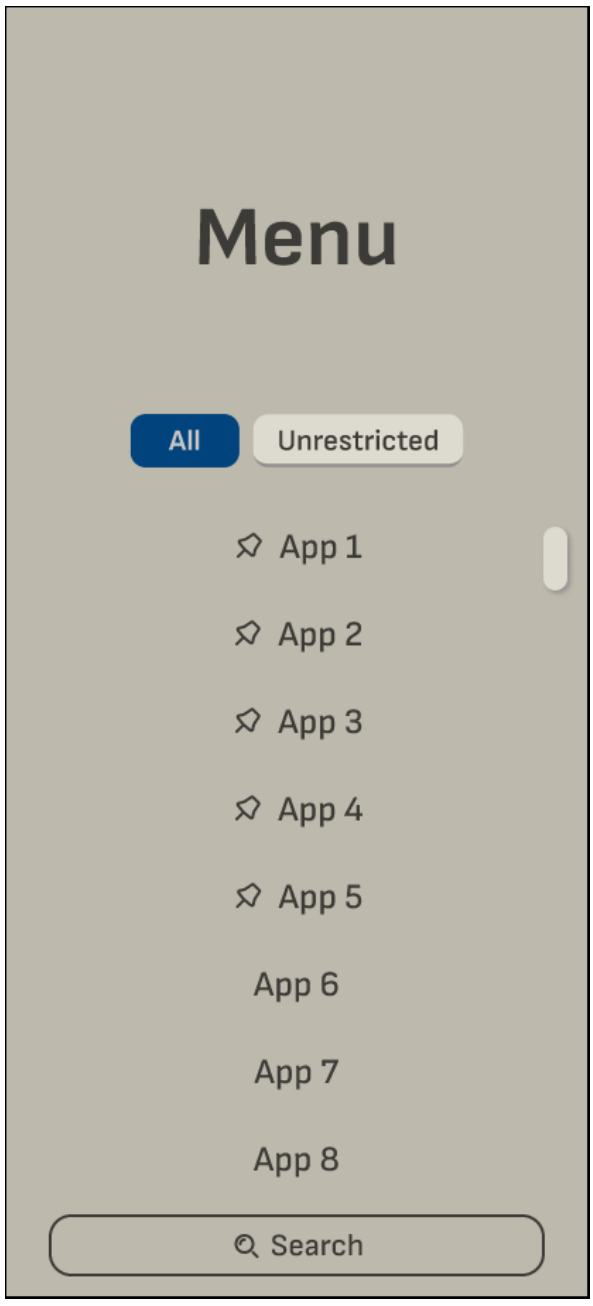
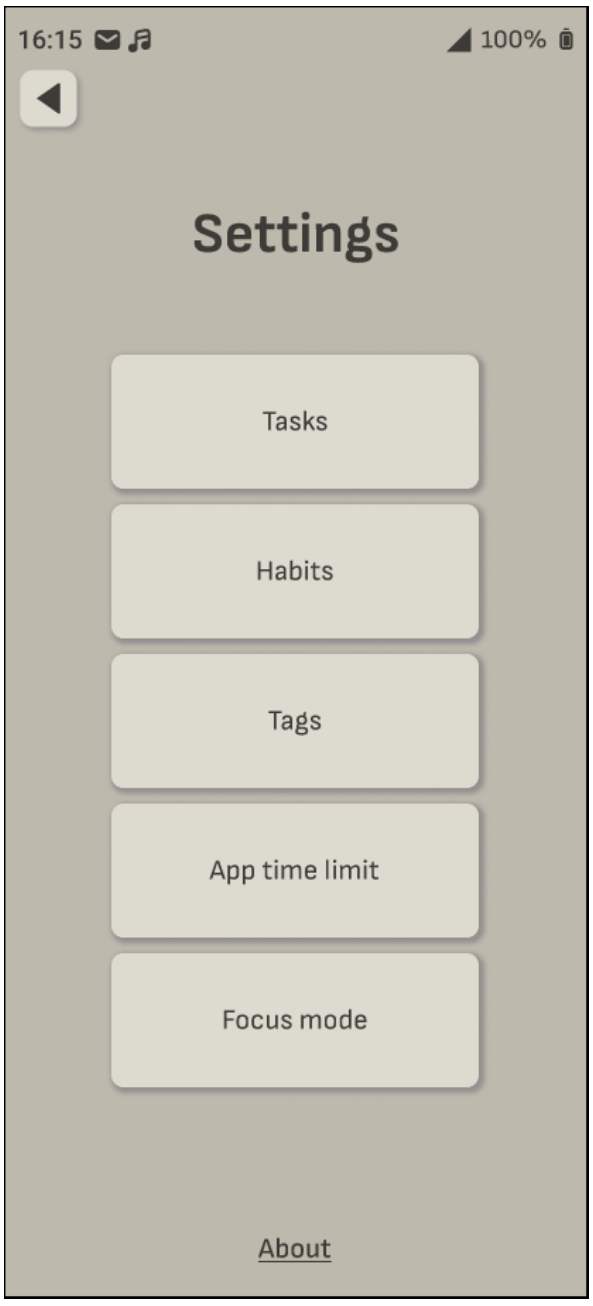


Figura 15: Mockup: Configuraciones.



7. Desarrollo

7.1. Estructura.

7.1.1. Proyecto.

La estructura de la aplicación procura seguir cuidadosamente el patrón MVVM y las mejores prácticas de Android, organizando el código en módulos que facilitan el mantenimiento, la escalabilidad y la comprensión del sistema. La organización del proyecto se define de la siguiente manera:

- **config:** Contiene las configuraciones de la base de datos Room, así como futuros archivos de configuración que pueda necesitar el proyecto.
- **di:** Implementa la inyección de dependencias utilizando Dagger-Hilt, siguiendo las recomendaciones de Google para la gestión de dependencias en Android. Hilt facilita la creación y provisión automática de instancias de UseCases y ViewModels.
- **events:** Define los eventos de la aplicación que permite la coordinación entre la UI, los ViewModels y la gestión de estados.
- **model:** Contiene las entidades de la base de datos que representan la estructura de información del launcher. Incluye modelos para tareas, hábitos, aplicaciones, límites y categorías, cada uno definiendo la estructura de datos específica para su dominio correspondiente mediante anotaciones de Room. Además, incluye clases de estado auxiliares, constantes usadas a lo largo del proyecto y la lógica de negocio encapsulada en UseCases, que representan las operaciones que pueden realizarse sobre los datos, como añadir, eliminar o actualizar tareas y hábitos.
- **navigation:** Gestiona la navegación entre pantallas utilizando *Navigation Compose* de Jetpack, proporcionando una navegación fluida y consistente a través de toda la aplicación. Define las rutas de navegación y maneja las transiciones entre diferentes vistas del launcher.
- **services:** Implementa servicios especializados para límite de tiempo de aplicaciones, gestión de notificaciones del sistema y actualización de estado de tareas y hábitos al inicio de cada día.
- **utils:** Proporciona utilidades y funciones auxiliares reutilizables a lo largo del proyecto.
- **view:** Representa la capa de vista implementada completamente en Jetpack Compose. Esta carpeta se subdivide en módulos específicos que incluyen componentes reutilizables, pantallas principales y vistas especializadas para la gestión de elementos. Cada vista observa los cambios en su ViewModel correspondiente y se recompone automáticamente según el estado de la aplicación.

- **viewmodel:** Constituye el núcleo del patrón MVVM, implementando los ViewModels específicos para cada funcionalidad principal. Cada ViewModel gestiona el estado y la lógica de presentación de su respectiva vista mediante StateFlow y corrutinas de Kotlin.

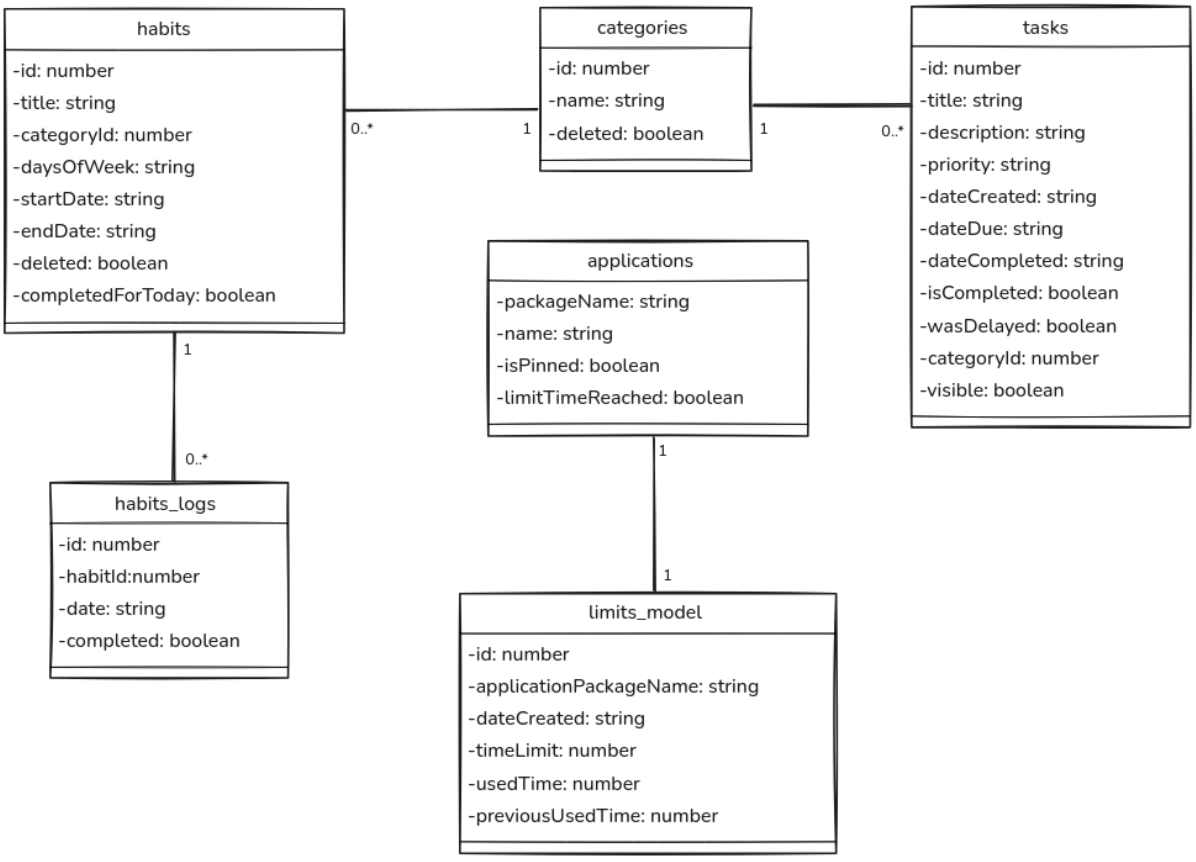
7.1.2. Base de datos.

El modelo de datos del launcher se compone de seis entidades principales, cada una diseñada para encapsular un aspecto específico de la funcionalidad del launcher:

- **ApplicationsModel:** Entidad para gestionar las aplicaciones instaladas en el dispositivo. Esta almacena información esencial sobre cada aplicación que el usuario puede controlar a través del launcher.
- **TasksModel:** Representa las tareas que el usuario puede crear y gestionar dentro del launcher como parte de su sistema de productividad personal.
- **HabitsModel:** Contiene la información relacionada con los hábitos que el usuario desea formar o mantener como parte de su rutina diaria. Esta entidad sigue una estructura similar a las tareas pero está optimizada para el seguimiento a largo plazo.
- **HabitsLogsModel:** Implementa el sistema de seguimiento temporal para los hábitos, permitiendo el registro histórico del cumplimiento de cada hábito a lo largo del tiempo. Esta implementación permite análisis de tendencias y proporciona al usuario retroalimentación sobre su progreso en la formación de hábitos. Aunque fue implementada, no se utilizó en la versión final del launcher. Ver capítulo 10.
- **CategoriesModel:** Este modelo permite al usuario asignar una etiqueta a tareas y hábitos relacionados.
- **LimitsModel:** Gestiona las restricciones de tiempo que el usuario puede establecer para aplicaciones específicas como parte del sistema de control de uso del dispositivo.

La relación entre estas entidades y sus atributos se ilustra en la Figura 16.

Figura 16: Diagrama de la base de datos del launcher.



7.2. Secciones de la aplicación.

7.2.1. Pantalla de inicio (Home).

La primera interfaz que el usuario encuentra al iniciar el launcher es la vista de tareas y hábitos. Desde aquí el usuario podrá crear, editar, completar y eliminar, así como también acceder al histórico de los elementos mencionados. La lista está encabezada por los hábitos creados, que se muestran solo si el día actual está entre los días de la semana seleccionados para su repetición; le siguen las tareas, ordenadas automáticamente según su fecha de creación, vencimiento y prioridad asignadas.

Figura 17: Home: Vista de tareas y hábitos.

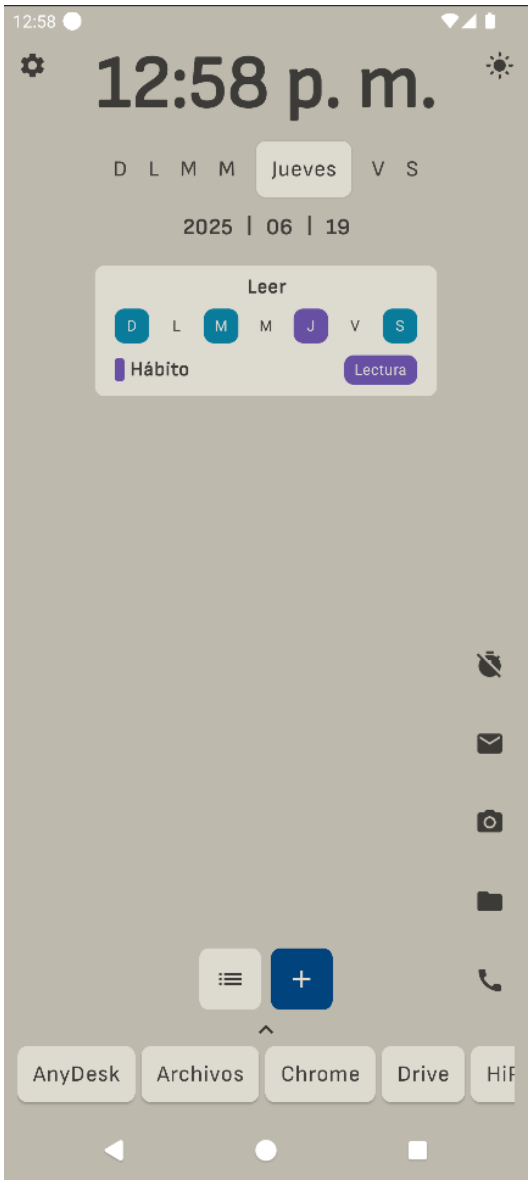


Figura 18: Home: Crear tarea o hábito.

Al crear un hábito (ver Figura 19) o una tarea (ver Figura 20), se mostrará una ventana modal con un pequeño formulario para especificar sus propiedades. Sus diferencias radican en que en el caso del hábito, se podrán seleccionar los días de la semana que se mostrará en la lista y, de asignarse una fecha de fin, se dejará de mostrar posterior a esta. En el caso de la tarea, de asignarse una fecha límite, se mostrará cuántos días faltan y se marcará como atrasada si no se cumple en el plazo establecido. En ambos casos, se podrá crear una categoría directamente en el modal o seleccionar una ya existente. Al guardar el elemento, se notifica al ViewModel correspndiente mediante un *Event*, quien envía la información al UseCase y actualiza la base de datos, reflejando la tarea o hábito recién creado gracias a las funciones de data binding que ofrecen los *Flows* [38] en Kotlin.

Figura 19: Home: Crear hábito.

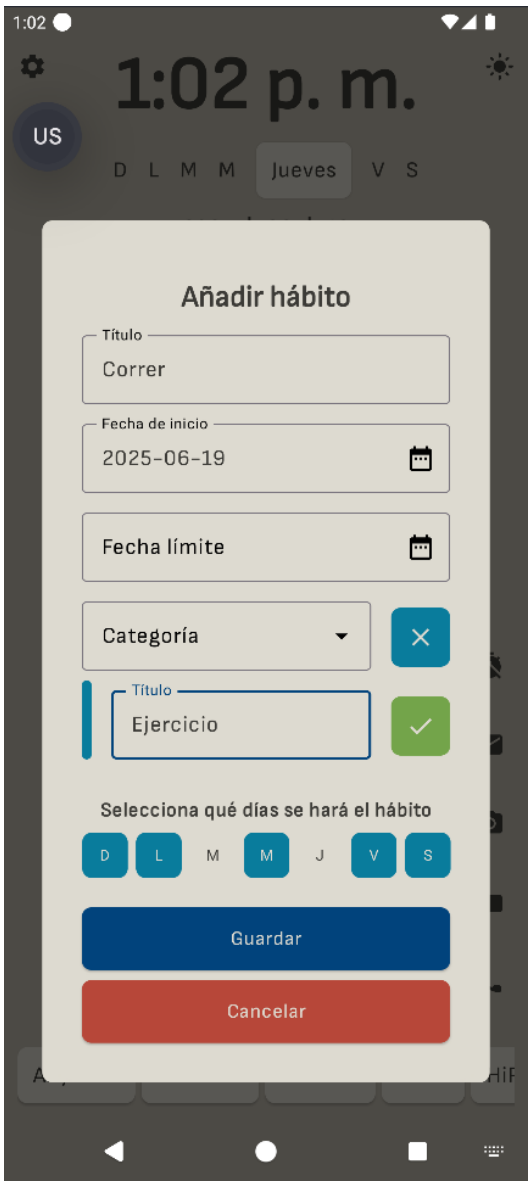


Figura 20: Home: Crear tarea.

Los widgets de hora y fecha también tienen un papel importante en esta pantalla, haciendo consciente al usuario, de manera constante, del momento en el que se encuentra. Al tocar sobre estos widgets, se puede acceder a las aplicaciones de reloj y calendario predeterminadas del smartphone. La interfaz también incorpora una barra de aplicaciones esenciales, en la parte inferior del costado derecho, que el usuario puede configurar para mostrar u ocultar según sus preferencias, proporcionando acceso inmediato a las aplicaciones de **teléfono, archivos, cámara, correo y la función de límite de tiempo de las aplicaciones**, dando un extra de utilidad sin perder su enfoque minimalista.

Del lado izquierdo de la vista de tareas y hábitos, se encuentra la implementación de la técnica de gestión del tiempo más conocida en el mundo de la productividad: Pomodoro. Esta técnica consiste en dividir el tiempo en intervalos de, generalmente, 25 minutos de trabajo seguidos de un descanso de 5 minutos. La idea de implementar una técnica de gestión del tiempo en el launcher es que el usuario pueda concentrarse en una tarea específica durante un período determinado. Al iniciar el temporizador, comenzará a contar el tiempo del ciclo establecido en el tiempo de sesión. Al finalizar, se le notificará al usuario mediante un sonido para que tome un breve descanso. La Figura 21 y la Figura 22 muestran el temporizador de Pomodoro en funcionamiento.

7.2.2. Menú de aplicaciones.

El menú de aplicaciones es una de las secciones más importantes del launcher, ya que permite al usuario acceder a todas las aplicaciones instaladas en su dispositivo. Sin duda es la sección que, a nivel visual, más difiere con respecto a los launchers tradicionales, ya que no muestra los íconos de las aplicaciones, solo una lista con los nombres de las aplicaciones, organizadas alfabéticamente. La intención es que el usuario sepa claramente qué aplicación está a punto de abrir, sin ser influenciado inconscientemente por el diseño llamativo de los íconos de las aplicaciones.

Se divide en dos secciones principales: la **barra de búsqueda** y la **lista de aplicaciones**. En la lista de aplicaciones, el usuario puede desplazarse verticalmente para buscar la que desea usar. Al tocar en una de ellas, se abrirá la aplicación correspondiente. Al dar una pulsación sostenida, se harán visibles tres (3) opciones: **Fijar a la pantalla de inicio, información de la aplicación y desinstalar**. Aquellas aplicaciones fijadas se mostrarán en la parte inferior de la sección de home para un acceso más inmediato. La barra de búsqueda permite al usuario encontrar aplicaciones con más precisión: A medida que el usuario escribe, la lista de aplicaciones se filtra en tiempo real, mostrando solo aquellas que coinciden con el texto ingresado. También se mostrará, en la última posición, un botón en la lista de aplicaciones para buscar el texto ingresado en la Web, abriendo el navegador predeterminado del dispositivo.

Este comportamiento se logra mediante el uso de la API de `PackageManager` en Android, la cual proporciona la información de las aplicaciones instaladas; y la declaración de acciones e intercambio de información a través de `Intents`. Los Intents expresan lo que una aplicación desea enviar o realizar, y permiten al launcher interactuar con el sistema operativo para abrir aplicaciones, diálogos del sistema o realizar búsquedas en la Web. La lista de aplicaciones se guarda en la base de datos de Room al iniciar el launcher por primera

Figura 21: Pomodoro: Tiempo de trabajo.

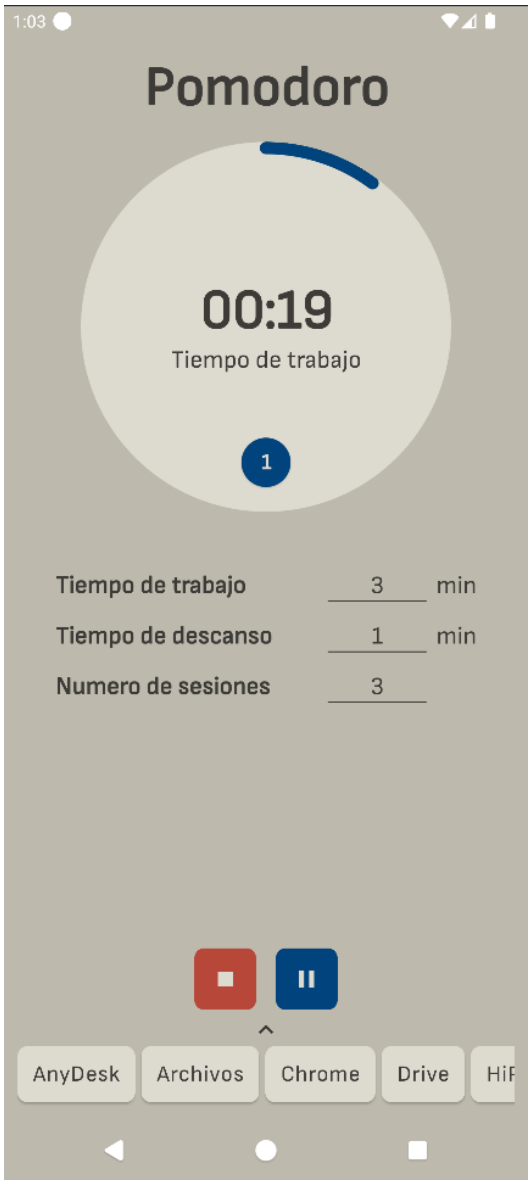
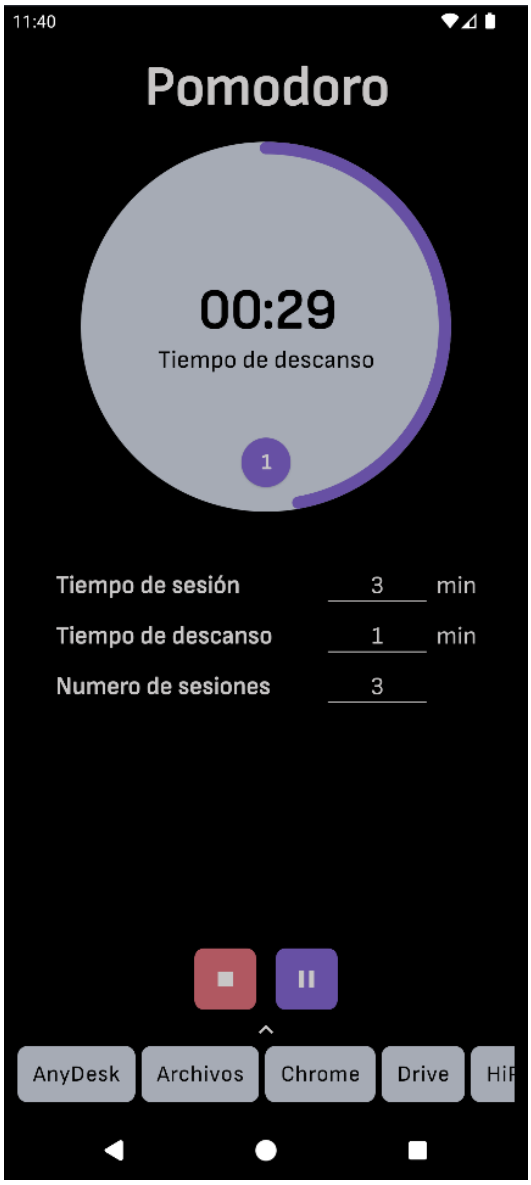


Figura 22: Pomodoro: Tiempo de descanso.



vez, y se actualiza cada vez que el sistema envía un `Broadcast` para notificar un cambio en las aplicaciones instaladas. A su vez, el launcher recibe ese Broadcast mediante un `BroadcastReceiver`, que escucha específicamente Intents de tipo `ACTION_PACKAGE_ADDED` o `ACTION_PACKAGE_REMOVED`. La implementación del menú se muestra en la Figura 23 y la Figura 24.

Figura 23: Menú: Lista de aplicaciones.

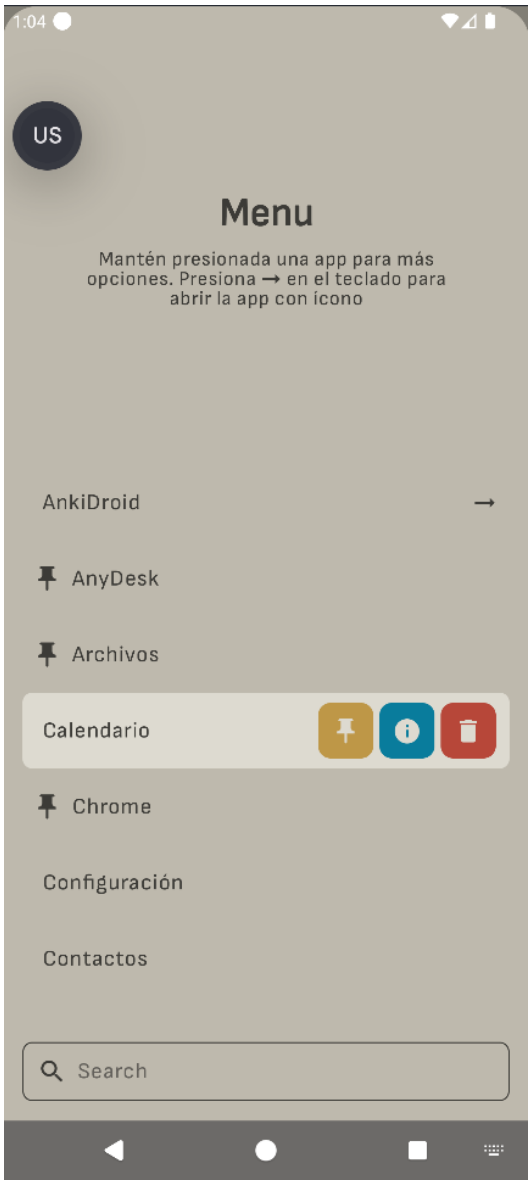
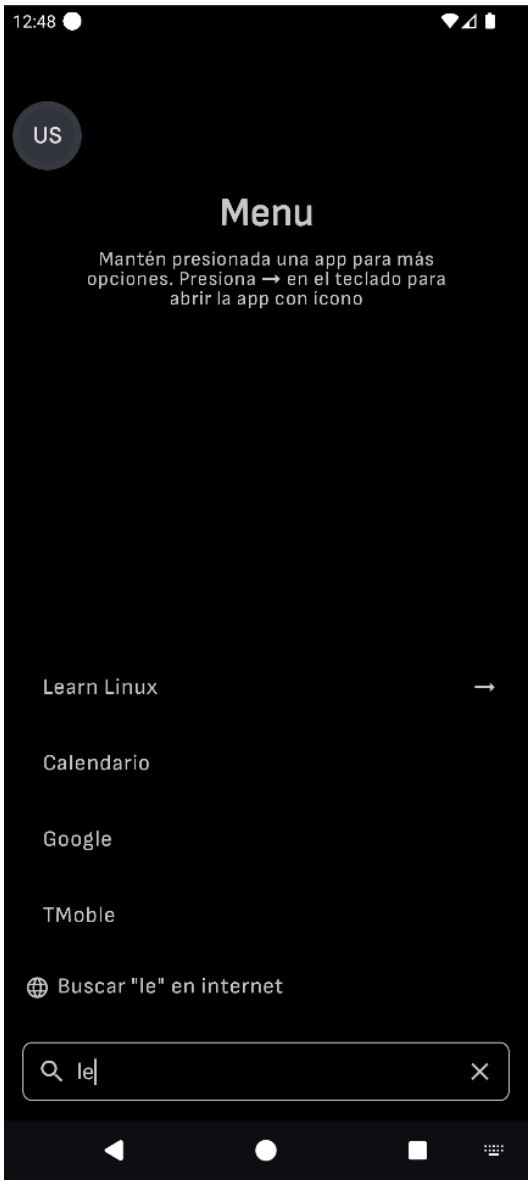


Figura 24: Menú: Búsqueda de aplicaciones.



7.2.3. Ajustes y personalización.

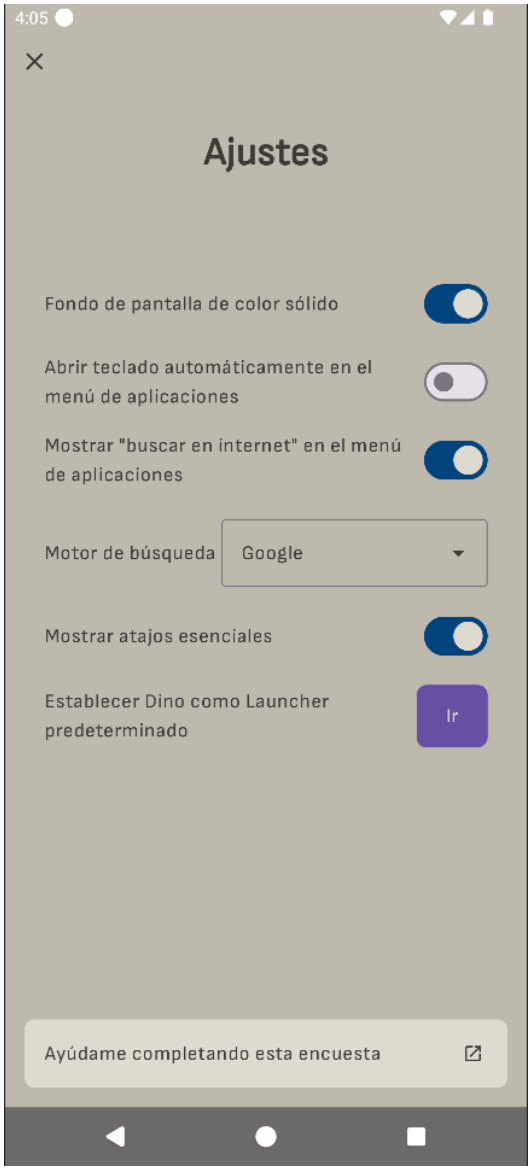
La sección de ajustes, accedida desde el engranaje de la sección de home en la esquina superior izquierda (ver Figura 17 y Figura 18), permite al usuario personalizar ligeramente su experiencia de uso dentro del launcher. Cuenta con las siguientes opciones:

- Fondo de pantalla de color sólido - Opción activada
- Abrir teclado automáticamente en el menú de aplicaciones
- Mostrar "buscar en internet."[en] el menú de aplicaciones

- Motor de búsqueda
- Mostrar atajos esenciales
- Establecer Dino como Launcher predeterminado

Así mismo, en la sección de home, al lado opuesto del engranaje se encuentra el ícono para el cambio de tema, que cuenta con tres (3) modos: **Claro**, **oscuro** y **automático** (predeterminado del sistema). Las variantes del tema ya se han podido evidenciar a lo largo de este capítulo, y se implementaron utilizando el sistema de temas de Jetpack Compose, consignando los colores, tipografías y estilos definidos en el proceso de diseño de los mockups.

Figura 25: Ajustes.



7.2.4. Límite de tiempo de aplicaciones.

Otro pilar fundamental del launcher es la funcionalidad de límite de tiempo de aplicaciones, que permite al usuario establecer restricciones en el uso diario de aplicaciones específicas. Para acceder a la funcionalidad, el usuario debe tocar el ícono de bloqueo de tiempo presente en la barra de aplicaciones esenciales del home, como se puede ver en la Figura 17 y la Figura 18, y aceptar dos (2) permisos adicionales: **Acceso a estadísticas de uso** y **mostrar sobre otras aplicaciones**, requeridos por el launcher para que esta característica funcione correctamente. Al navegar a esta sección, aparecerá una lista de las aplicaciones limitadas y el tiempo en minutos que el usuario las ha utilizado en el día actual, mostrado en la Figura 26. Para limitar una aplicación, basta con ir a seleccionar aplicaciones, mostrado en la Figura 27, y fijar un tiempo máximo de uso.

A través de la API de `UsageStatsManager`, el launcher obtiene las estadísticas de uso de las aplicaciones instaladas y, si hay alguna que se encuentre limitada desde el launcher, muestra su información del tiempo de uso. Aceptar los permisos ya mencionados inicia un tipo de serivicio llamado `Foreground Service`, el cual se encarga de monitorear el uso de las aplicaciones cada minuto y actualizar la interfaz del launcher o los campos que habilitan el bloqueo a las aplicaciones limitadas si el tiempo establecido ya se cumplió. A diferencia de los servicios normales, un Foreground Service muestra una notificación persistente al usuario para hacerle saber de su ejecución (ver Figura 29) y el sistema tiene predilección por mantenerlo activo. Al llegar al límite de tiempo en alguna aplicación limitada, el launcher mostrará una interfaz notificando al usuario e invitándolo a abandonar la aplicación por el día de hoy. Esta interfaz se muestra en la Figura 28.

7.3. Limitaciones.

Es importante señalar que, aunque el servicio de monitoreo implementado en el launcher verifica el uso de aplicaciones cada minuto mediante la API de UsageStatsManager, el sistema operativo Android no actualiza las estadísticas de uso con la misma frecuencia. Esta diferencia temporal puede ocasionar que la interfaz de límite de tiempo alcanzado aparezca varios minutos después de que efectivamente se haya superado el umbral establecido, afectando la efectividad del enfoque restrictivo.

Adicionalmente, debido a las restricciones de seguridad en Android, el launcher no posee la capacidad de controlar directamente la actividad de otras aplicaciones una vez que han sido iniciadas. Por esto, el launcher no puede forzar el cierre automático de aplicaciones cuando se cumple el límite de tiempo establecido, ni tampoco puede impedir que estas sean accedidas a través de mecanismos alternativos del sistema, tales como la pantalla de aplicaciones recientes, notificaciones entrantes, enlaces desde otras aplicaciones o si estas son iniciadas por el propio sistema operativo.

Figura 26: Límite de tiempo: Uso del día actual.

Figura 27: Límite de tiempo: Selección de aplicaciones.

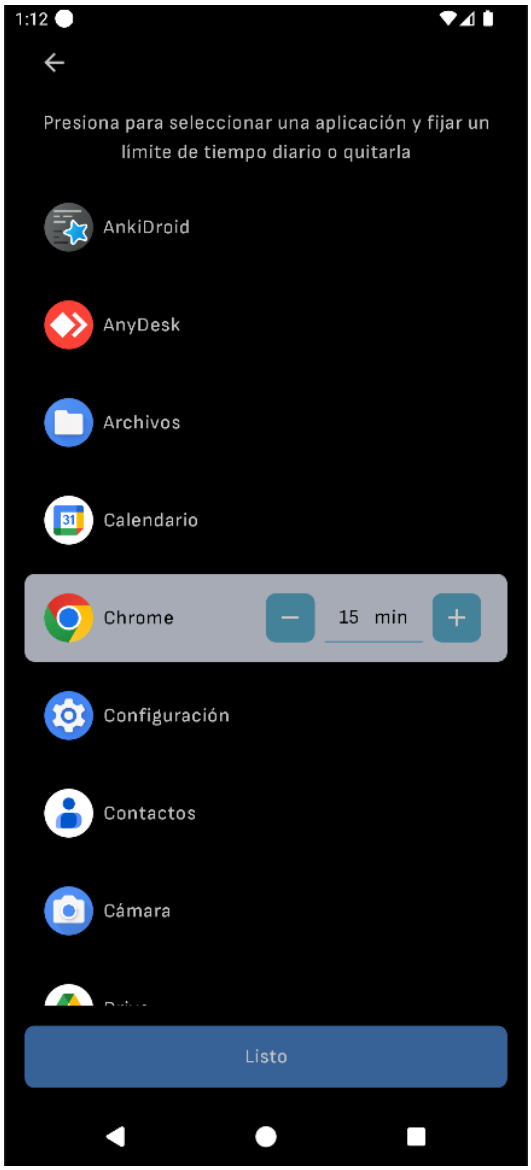


Figura 28: Límite de tiempo alcanzado.

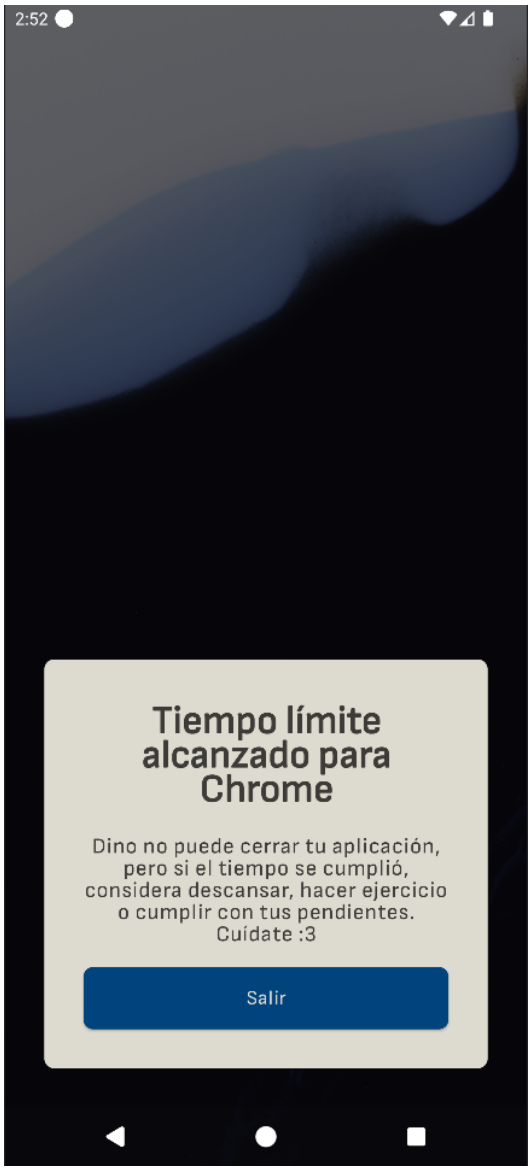
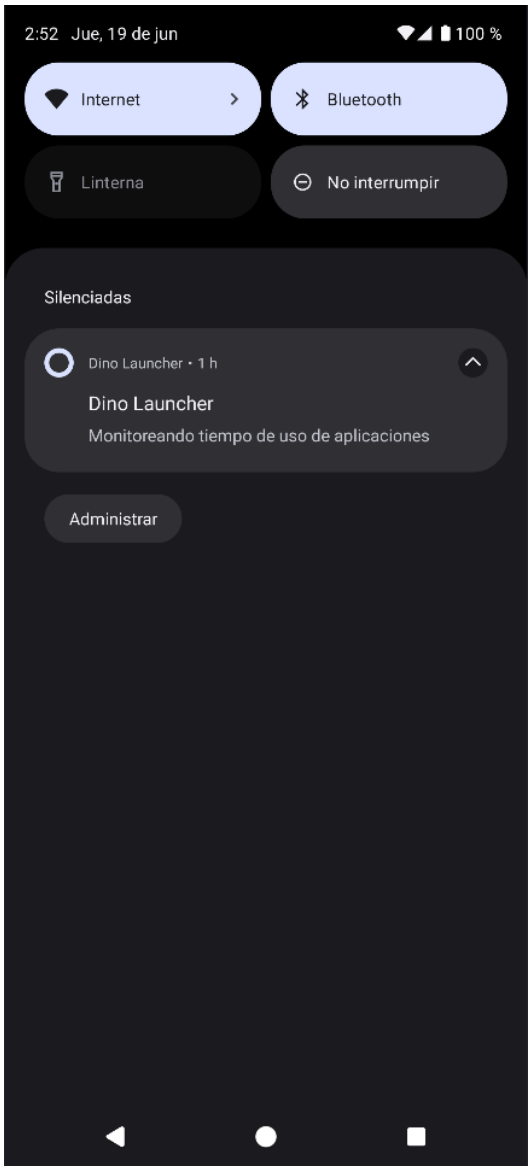


Figura 29: Notificación del Foreground Service.



8. Pruebas.

8.1. Pruebas de usabilidad.

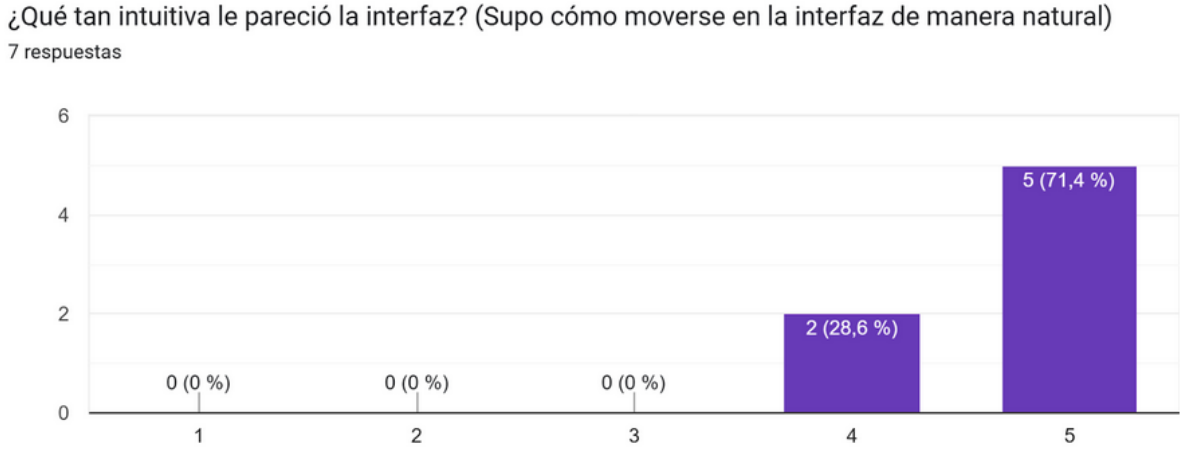
La evaluación de usabilidad del launcher se realizó mediante una encuesta a estudiantes universitarios, grupo objetivo de la aplicación. Los participantes, jóvenes adultos entre 22 y 25 años que usan su smartphone regularmente, reflejan el perfil típico de usuarios que buscan mejorar su concentración y productividad. La mayoría manifestó ser consciente del tiempo excesivo que pasa en su smartphone, especialmente en momentos inapropiados como durante actividades académicas o de descanso, y expresó interés en herramientas que les ayuden a regular este uso.

Cabe destacar que el número de encuestados fue reducido debido a las particularidades técnicas y éticas del proyecto. Dado que se trata de una aplicación para Android que se debe instalar, no es posible su despliegue en un entorno web ni su distribución pública, ya que su instalación requiere permisos especiales relacionados con acceso a información del sistema operativo y el monitoreo del tiempo de uso de las aplicaciones, aspectos que son sensibles en términos de privacidad.

Por estas razones, se decidió aplicar la encuesta únicamente a un grupo selecto de usuarios que aceptaron instalar la aplicación en sus dispositivos personales y participar bajo las condiciones previamente informadas, garantizando así un entorno de prueba controlado, seguro y éticamente responsable.

La Figura 30 muestra que la mayoría de los participantes considera que el launcher es intuitivo y fácil de usar. El 71,4 % de los participantes calificó con la puntuación máxima (5) la intuición de la interfaz del launcher, mientras que el 28,6 % le otorgó una calificación de 4.

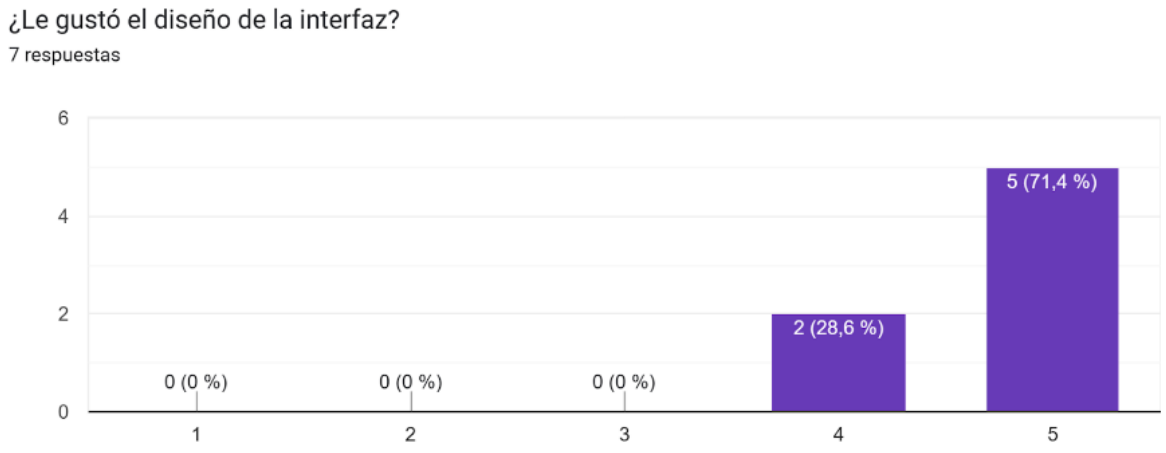
Figura 30: Primera prueba de usabilidad.



En relación con la apariencia de la aplicación, los resultados presentes en la Figura 31 reflejan una alta aceptación del diseño minimalista, funcional y centrado en la productividad por parte de los usuarios. El 71,4 % de los encuestados calificó el diseño de la interfaz

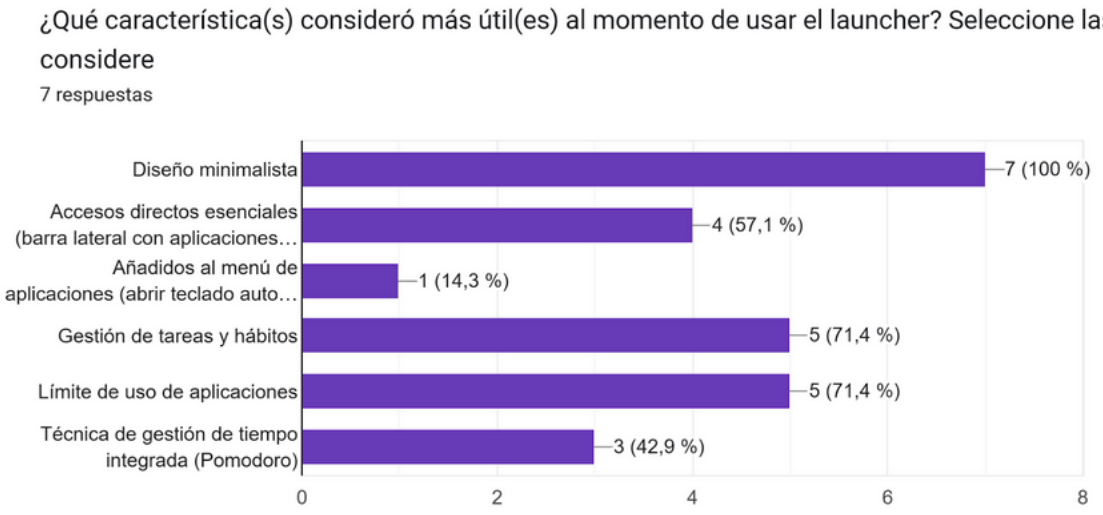
con la puntuación máxima (5), mientras que el 28,6 % otorgó una calificación de 4.

Figura 31: Segunda prueba de usabilidad.



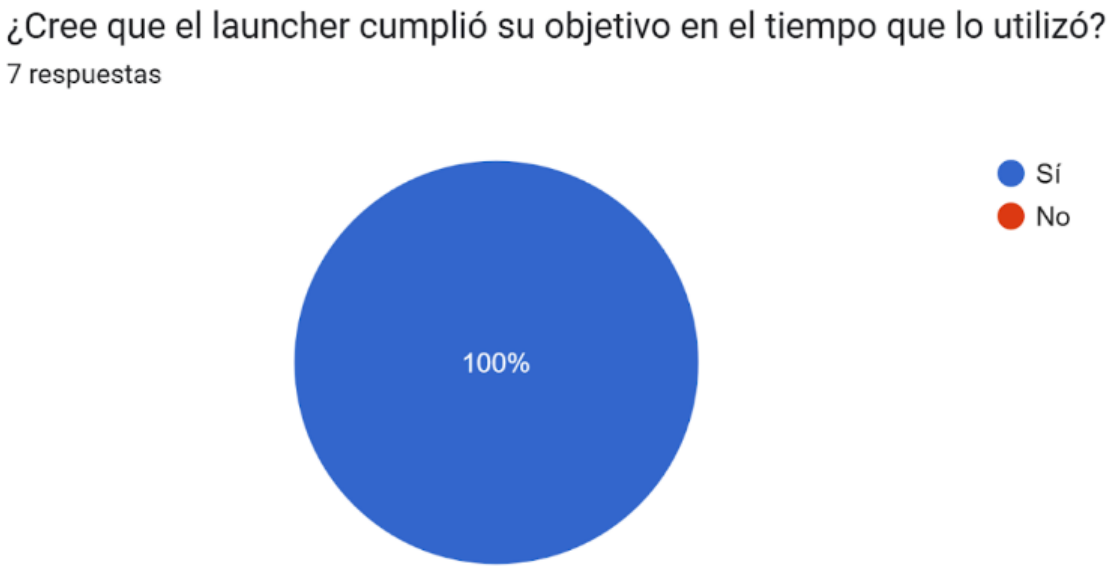
Los resultados de la tercera prueba de usabilidad, mostrados en la Figura 32, indican que el diseño minimalista fue identificado como la característica más útil del launcher por el 100 % de los participantes. En segundo lugar, se destacan la gestión de tareas y hábitos (71,4 %) y el límite de uso de aplicaciones (71,4 %), dos de las principales funcionalidades del launcher.

Figura 32: Tercera prueba de usabilidad.



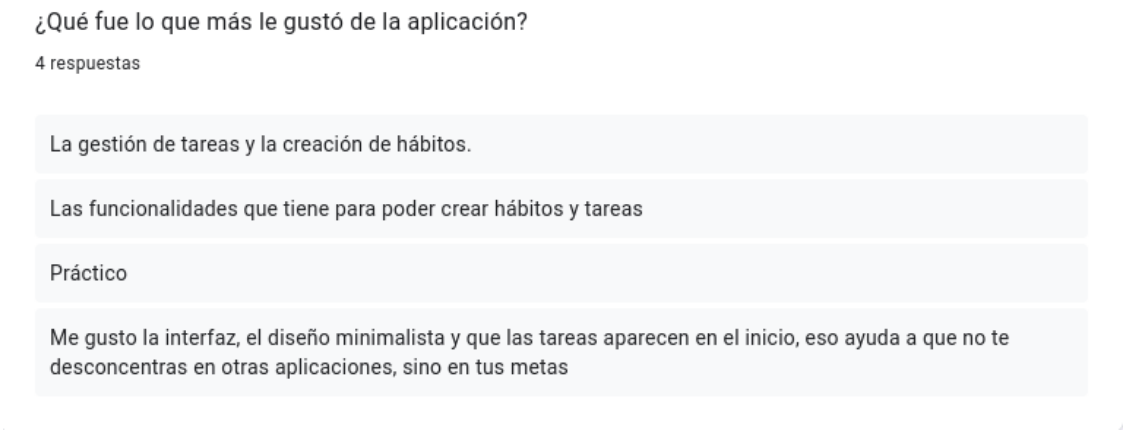
El 100 % de los encuestados, según la Figura 33, coincidió en que el launcher cumplió con su objetivo de proporcionar una interfaz de pantalla de inicio minimalista, funcional y con herramientas que pueden ayudar a evitar las distracciones en el uso del Smartphone y ayudar a mejorar la productividad durante el periodo en que fue utilizado.

Figura 33: Cuarta prueba de usabilidad.



En respuesta a la pregunta: **¿Qué fue lo que más le gustó de la aplicación?**, se destacan la funcionalidad de gestión de tareas y hábitos, la practicidad general de la aplicación y su diseño minimalista. También se resaltó el hecho de que las tareas se visualizan de inmediato al iniciar el launcher para mantener el enfoque en las metas personales.

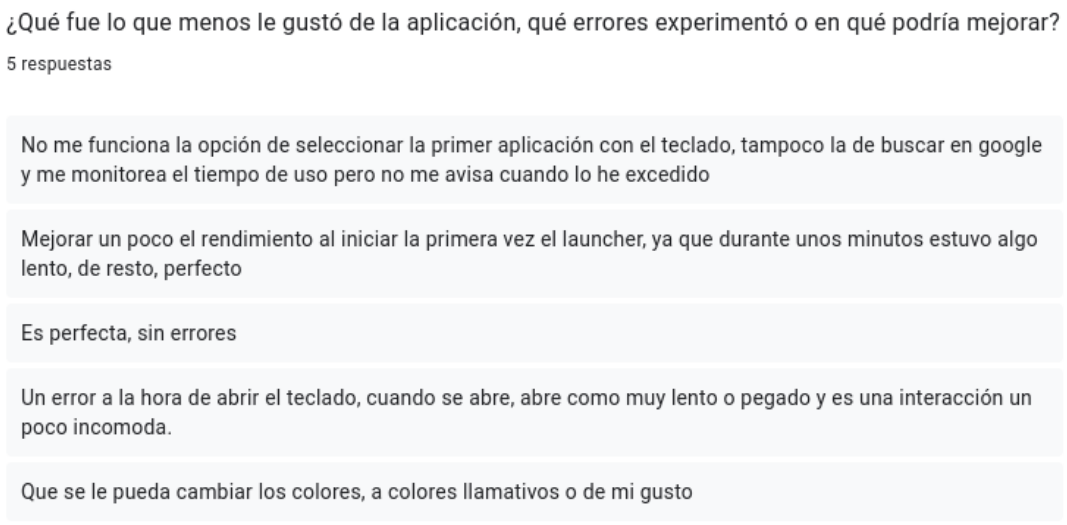
Figura 34: Quinta prueba de usabilidad.



En cuanto a la pregunta: **¿Qué fue lo que menos le gustó de la aplicación, qué errores experimentó o en qué podría mejorar?**, los participantes mencionaron principalmente la falta de opciones de personalización, problemas de rendimiento al iniciar la aplicación o al abrir el teclado al entrar al menú de aplicaciones. Por otro lado, uno de los encuestados destacó que la aplicación funcionó sin inconvenientes, lo cual sugiere que

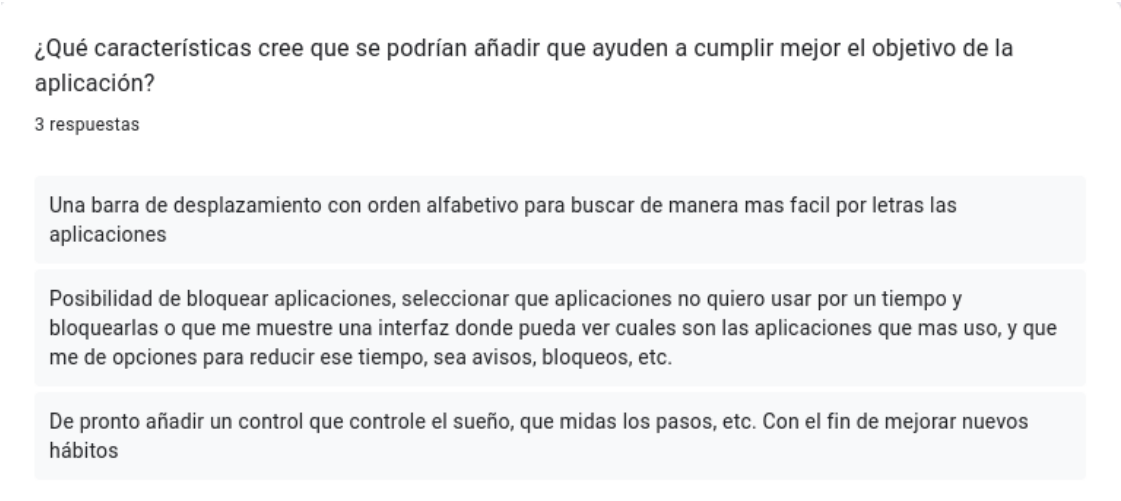
la experiencia puede variar según el dispositivo o la configuración.

Figura 35: Sexta prueba de usabilidad.



En respuesta a: **¿Qué características cree que se podrían añadir que ayuden a cumplir mejor el objetivo de la aplicación?**, los participantes sugirieron una barra de desplazamiento para buscar por orden alfabético las aplicaciones, la posibilidad de bloquear aplicaciones independientemente de si tienen o no límite de tiempo y características de monitoreo de salud, como seguimiento del sueño.

Figura 36: Séptima prueba de usabilidad.



Por último, en la pregunta: **¿Consideraría seguir usando este Launcher o uno con un enfoque similar en su día a día?**, el 71,4 % de los participantes manifestó su intención de seguir utilizando el launcher o una herramienta con un enfoque similar en su

rutina diaria. Por otro lado, el 28,6 % indicó que "tal vez"lo haría, lo cual, aunque no representa una afirmación definitiva, demuestra una disposición positiva hacia la aplicación.

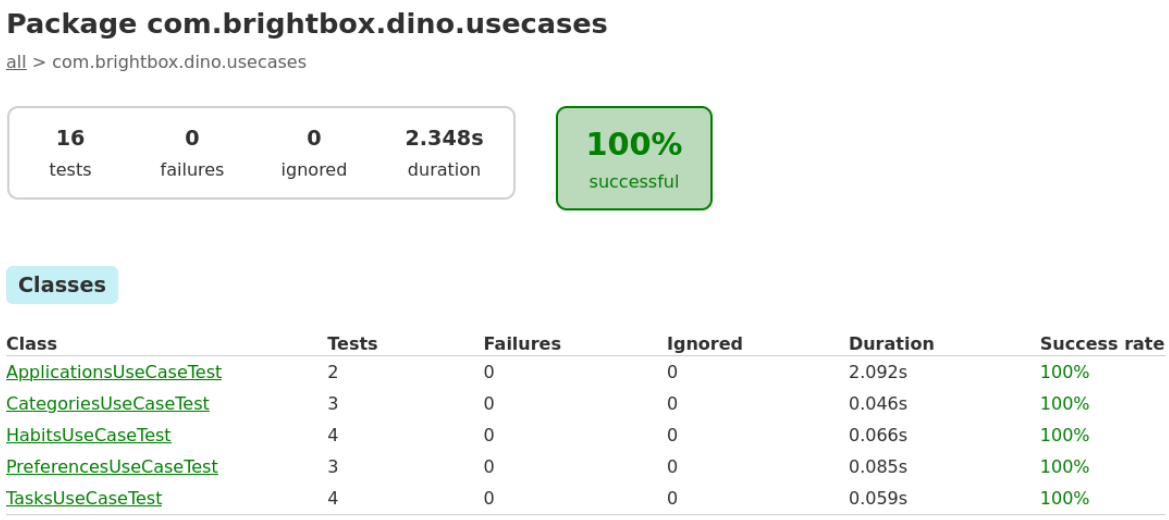
Figura 37: Octava prueba de usabilidad.

8.2. Pruebas unitarias.

Las pruebas unitarias constituyen un componente fundamental en cualquier proceso de desarrollo de software moderno. Son el primer nivel de pruebas que se realizan para validar la funcionalidad de las unidades más pequeñas del código, como funciones o métodos individuales.

Se desarrollaron pruebas unitarias específicas que cubren los casos de uso que encapsulan la lógica de negocio principal (operaciones con los datos, CRUD). La estrategia de testing adoptada utilizó `JUnit 4` como framework base y `mockk` para simluar las entidades usadas en la aplicación real.Los resultados de las pruebas se muestran en la Figura 38.

Figura 38: Resultados pruebas unitarias.



Package com.brightbox.dino.usecases

all > com.brightbox.dino.usecases

16	0	0	2.348s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
ApplicationsUseCaseTest	2	0	0	2.092s	100%
CategoriesUseCaseTest	3	0	0	0.046s	100%
HabitsUseCaseTest	4	0	0	0.066s	100%
PreferencesUseCaseTest	3	0	0	0.085s	100%
TasksUseCaseTest	4	0	0	0.059s	100%

9. Conclusiones.

- Los estudiantes universitarios tienen una tendencia clara a usar su smartphone más tiempo del que deberían y en ocasiones inoportunas, reforzando la idea de implementar controles y hacer seguimiento al uso del smartphone para evitar distracciones, fomentando el cumplimiento de metas y deberes.

- El diseño minimalista, aunque efectivo para reducir distracciones, mostró la necesidad de ofrecer más opciones de personalización a los usuarios, tales como más tipografías, paletas de colores o poder añadir widgets; con el fin de mejorar el nivel de aceptación y adopción como aplicación de uso diario.

- La implementación de las tecnologías más recientes y recomendadas para iniciar nuevos proyectos en Android, concretamente Kotlin y Jetpack Compose, redujeron el tiempo de desarrollo y permitieron una integración completa que aprovecha el conjunto de características del lenguaje con su sintaxis simplificada.

- El patrón de arquitectura MVVM permitió que la aplicación escalara sin problemas a medida que se añadían nuevas funcionalidades, facilitando el mantenimiento, la comprensión y la reutilización del código gracias al principio de separación de responsabilidades.

- Las limitaciones técnicas del sistema operativo Android, como la actualización retardada de estadísticas de uso o el no poder cerrar las aplicaciones limitadas una vez cumplido el tiempo, impiden que este tipo de aplicaciones puedan trabajar acorde a lo que se espera e impactan negativamente la experiencia del usuario.

- Los resultados de las pruebas realizadas evidenciaron la satisfacción de los usuarios durante el periodo en el que interactuaron con la aplicación, expresando que esta fue de su agrado y que cumple con el objetivo de reducir las distracciones y ayudar a mejorar la productividad.

10. Trabajos futuros.

- Implementar una sección dedicada a estadísticas detalladas sobre el tiempo de uso de aplicaciones y el seguimiento de tareas y hábitos completados, proporcionando al usuario información clara sobre sus patrones de uso.
- Desarrollar funcionalidades adicionales que permitan desactivar automáticamente notificaciones y sonidos cuando el temporizador de Pomodoro esté activo, asegurando una mayor concentración del usuario.
- Incorporar nuevas opciones de personalización, como paletas de colores, tipografías, fondos de pantalla y widgets que amplíen las opciones de personalización, mejorando la experiencia visual y adaptabilidad a diferentes preferencias estéticas.
- Mejorar las animaciones para proporcionar transiciones más fluidas y atractivas, incrementando la sensación de calidad percibida por el usuario.
- Agregar opciones avanzadas para reorganizar las aplicaciones fijadas, permitiendo ordenarlas por carpetas definidas por el usuario.
- Implementar recordatorios y notificaciones personalizadas para ayudar a los usuarios a cumplir con sus objetivos y tareas diarias.
- Explorar la posibilidad de integrar la aplicación con servicios de terceros, como calendarios o aplicaciones de notas, para ofrecer una experiencia más completa y conectada.

11. Referencias

[1] Google, "Cómo descargar android studio y app tools - android developers."

[2] Google, "Cómo guardar contenido en una base de datos local con room — app data and files — android developers."

[3] C. Montag, B. Lachmann, M. Herrlich, and K. Zweig, "Addictive features of social media/messenger platforms and freemium games against the background of psychological and economic theories," *International Journal of Environmental Research and Public Health 2019, Vol. 16, Page 2612*, vol. 16, p. 2612, 7 2019.

[4] S. G. Luque and C. F. Rovira, "Vista de redes sociales y consumo digital en jóvenes universitarios: economía de la atención y oligopolios de la comunicación en el siglo xxi," 2020.

[5] A. Lepp, J. E. Barkley, and A. C. Karpinski, "The relationship between cell phone use and academic performance in a sample of u.s. college students," *https://doi.org/10.1177/2158244015573169*, vol. 5, 2 2015.

[6] F. Kus, kaya Mumcu, T. Has, and Y. D. Çevik, "Modelling smartphone addiction: The role of smartphone usage, self-regulation, general self-efficacy and cyberloafing in university students," 2016.

[7] N. Grewal, J. K. Bajaj, and M. Sood, "View of impact of mobile phone usage on academic performance and behaviour of medical students," 2020.

[8] P. A. C. M. Puerto, M. . Puerto, D. . Rivero, L. . Sansores, L. . Gamboa, and L. Sarabia, "Somnolencia, hÁbitos de sueÑo y uso de redes sociales en estudiantes universitarios," *Enseñanza e Investigación en Psicología*, vol. 20, pp. 189–195, 2015.

[9] M. E. Beutel, E. M. Klein, S. Aufenanger, E. Brähler, M. Dreier, K. W. Müller, O. Quiring, L. Reinecke, G. Schmutzer, B. Stark, and K. Wölfling, "Procrastination, distress and life satisfaction across the age range – a german representative community study," *PLoS ONE*, vol. 11, 2 2016.

[10] B. my cell, "How many android users are there? global statistics (2024)," 2024.

[11] E. Rafaila and N. Duta, "Teaching and self-teaching in higher education," *Procedia - Social and Behavioral Sciences*, vol. 197, pp. 1230–1235, 7 2015.

[12] K. Lizbeth, B. Guevara, and V. F. F. Hernández, "Procrastinación y autoeficacia académica en estudiantes universitarios: Procrastination and academic self-efficacy in university students," *LATAM Revista Latinoamericana de Ciencias Sociales y Humanidades*, vol. 4, pp. 2268–2280–2268–2280, 6 2023.

[13] C. Neyman, "A survey of addictive software design," vol. 12, 2017.

[14] M. J. P. Vértiz, Y. M. A. Sullón, D. S. T. M. C. D. G. B. V. J. A. T. F. I. L. C. M. H. Ángel Lozano Bazán Elvin Paolo Ponce Aranguri Juan Melitón Canes Acosta Edición, J. V. P. Diagramación, and F. G. Lara, "La universidad en cifras - mineducación perú," *Calle Del Comercio*, vol. 193, 2023.

[15] A. RODRÍGUEZ, M. CLARIANA, A. RODRÍGUEZ, and M. CLARIANA, "Procrastinación en estudiantes universitarios: su relación con la edad y el curso académico," *Revista Colombiana de Psicología*, vol. 26, pp. 45–60, 1 2017.

[16] A. M. Recuerda, D. J. Varón, M. F. S. Ripoll, and A. R. Villalobos, "La gestión del tiempo como habilidad directiva," *3C Empresa, Investigación y pensamiento crítico*, pp. 6–30, 2012.

[17] R. L. N. Guzmán and B. C. Cisneros-Chavez, "Adicción a redes sociales y procrastinación académica en estudiantes universitarios," 2019.

[18] J. M. Ordoñez, "La generación zombi. el excesivo uso de celulares en las aulas universitarias del perú.," *Revista Científica de Ciencias de la Salud*, vol. 16, pp. 61–72, 12 2023.

[19] A. Orzikulova, "App- and feature-level smartphone interventions to reduce time spent in mobile social media applications," 2022.

[20] C. M. U., E. F. H., C. P. V., J. O. B., P. P. P., L. O. M., O. M. B., and P. I. G., "Relationship between self-directed learning with learning styles and strategies in medical students," *Revista médica de Chile*, vol. 142, pp. 1422–1430, 2014.

[21] Lenovo, "Android™ launcher: What it does & how to use it — lenovo us."

[22] R. E. Navarro, "El rendimiento acadÉmico: Concepto, investigaciÓn y desarrollo," vol. 1, 2003.

[23] S. de educación pública de Hidalgo, "¿cómo mejorar el rendimiento escolar?,"

[24] I. europeo de psicología positiva, "Concentración: La capacidad de mantener la atención - iepp," 2023.

[25] G. V. Kamenetzky, L. Cuenya, A. M. Elgier, F. L. Seal, S. Fosacheca, L. Martin, and A. E. Mustaca, "Respuestas de frustración en humanos," *Terapia psicológica*, vol. 27, pp. 191–201, 12 2009.

[26] StackOverflow, "2024 stack overflow developer survey."

[27] Google, "Android's kotlin-first approach — android developers."

[28] Google, "Recursos para desarrolladores de android jetpack - android developers."

[29] R. Vanguardia, P. . Año, . Volumen, . Numero, D. M. Quant, and A. Sánchez, "Procrastinación, procrastinación académica: Concepto e implicaciones," *Revista Vanguardia Psicológica Clínica Teórica y Práctica, ISSN-e 2216-0701, Vol. 3, Nº. 1, 2012 (Ejemplar dedicado a: Nuevos aportes para una comunicación transdisciplinar), págs. 45-59*, vol. 3, pp. 45–59, 2012.

[30] D. C. V. Vacacela, L. J. M. Jara, D. M. C. Contreras, D. C. V. Vacacela, L. J. M. Jara, and D. M. C. Contreras, "Procrastinación académica y dependencia al dispositivo móvil en estudiantes universitarios," *Revista Eugenio Espejo*, vol. 17, pp. 42–51, 9 2023.

[31] U. Y. Sociedad, C. Alberto, G. Cano, V. S. Castillo, and Y. S. González, "Factores que inciden en la procrastinación académica de los estudiantes de educación superior en colombia," *Revista Universidad y Sociedad*, vol. 15, pp. 421–431, 2023.

[32] A. S. Garcia and M. E. Escalera-Chávez, "Vista de adicción hacia el teléfono móvil en estudiantes de nivel medio superior. ¿cómo es el comportamiento por género?," 11 2020.

[33] Google, "Guía de arquitectura de apps — app architecture — android developers."

[34] Google, "Android developers blog: Announcing architecture components 1.0 stable."

[35] "Mrmannwood/launcher."

[36] "finnmglas/launcher: :rocket: A distraction-free minimal homescreen for android.."

[37] "tanujnotes/olauncher: Minimal af launcher for android. reduce your screen time. daily wallpapers.."

[38] Google, "Kotlin flows on android — android developers."