ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

TRABAJO DE FIN DE GRADO

- Memoria Técnica -

Grado en Ingeniería Informática

Mención en Computación

Simulador de algoritmos de planificación de procesos para la asignatura de Sistemas Operativos del Grado de Ingeniería Informática de la Universidad de Córdoba

Autor

Julen Pérez Hernández

Director

Juan Carlos Fernández Caballero

UNIVERSIDAD D CÓRDOBA

D. Juan Carlos Fernández Caballero, Profesor Titular del Departamento de Informática y Análisis Numérico en la Escuela Politécnica Superior de Córdoba de la Universidad de Córdoba, e investigador del grupo AYRNA (Aprendizaje y Redes Neuronales Artificiales).

Informan:

Que el presente Trabajo Fin de Grado de Ingeniería Informática titulado "Simulador de algoritmos de planificación de procesos para la asignatura de Sistemas Operativos del Grado de Ingeniería Informática de la Universidad de Córdoba", constituye la memoria presentada por Julen Pérez Hernández para aspirar al título de Graduado en Ingeniería Informática, y ha sido realizado bajo mi dirección en la Escuela Politécnica Superior de Córdoba de la Universidad de Córdoba, reuniendo, a mi juicio, las condiciones necesarias exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente informe en Córdoba, septiembre de 2023.

- Autor:

Fdo: Julen Pérez Hernández

- Director:

Fdo: Juan Carlos Fernández Caballero

Agradecimientos

Quiero expresar mi más sincero agradecimiento al profesor tutor de este proyecto, Juan Carlos Fernández Caballero, por su dedicación, paciencia y valiosas sugerencias, que me permitieron llevar a cabo este trabajo con éxito.

Asimismo, no puedo dejar de mencionar a mi familia, quienes siempre estuvieron a mi lado brindándome un gran apoyo emocional y económico, lo que me permitió enfocarme completamente en mis estudios universitarios.

También quiero agradecer a mis amigos, quienes me alentaron a seguir adelante cuando las cosas se ponían difíciles, y me dieron su apoyo incondicional en todo momento.

Finalmente, me gustaría agradecerme a mí mismo, por nunca darme por vencido y continuar trabajando duro hasta alcanzar el final de esta importante etapa.

Índice General

Índice de Figuras

Índice de Tablas

Bloque
I. Introducción

Capítulo 1. Introducción

La planificación de procesos es una tarea fundamental del sistema operativo. Los entornos de simulación se utilizan para el estudio del comportamiento de los algoritmos de planificación.

La eficiencia de un algoritmo de planificación de procesos es uno de los apartados más importantes a considerar. El sistema operativo es el encargado de aplicar estos algoritmos en función de las políticas de planificación que se estén llevando a cabo en cada caso, gestionando la entrada y tiempo de procesamiento de cada uno de los procesos dentro de la *CPU*.

Las principales características de estos algoritmos de planificación son el rendimiento *I/O*, el tiempo de espera, la utilización de la *CPU*, el tiempo de respuesta, la prioridad y el tiempo total requerido por cada proceso.

Teniendo lo anterior en cuenta, se puede definir OSSAS (*Operating Systems Scheduling Algorithms Simulator*) como las siglas bajo las que se denomina a las múltiples herramientas para simular distintos escenarios, donde el sistema operativo determina el orden en que se adecúa el procesador a los procesos que lo vayan necesitando y a las políticas que se utilizarán en la eficiencia del tiempo esperado en el sistema [1].

Existen diversos algoritmos para dirigir la ordenación de procesos [2] como, por ejemplo, el algoritmo de planificación FCFS (*First come First served*), el algoritmo de planificación SJF (*Short Job First*), algoritmo de turno rotatorio (*Round Robin*), y algoritmo de prioridades, entre otros [3]. Desde la perspectiva de la enseñanza, estos temas suelen abordarse llevando a cabo numerosos ejercicios en papel o pizarra. Esto deriva,

frecuentemente, en la necesidad de simplificar en exceso los ejercicios o, por el contrario, en tener que desarrollar esquemas de compleja visualización.

Para este trabajo de fin de grado (TFG), surge la idea de la creación de una herramienta de apoyo a la docencia en torno a la simulación de las políticas de planificación del sistema operativo, para la asignatura de Sistemas Operativos de 2º de Grado de Ingeniería Informática de la Escuela Politécnica Superior de Córdoba. De esta manera, tanto alumnos como profesores se beneficiarían de su uso al poder realizar un mayor número de ejercicios en este ámbito, permitiendo agregarles cierta complejidad sin sacrificar su comprensión o sencillez visual. Por otro lado, el desarrollo de una herramienta digital también permitiría al alumnado simular ejercicios en diferentes dispositivos y en cualquier lugar, ya que se realizará un desarrollo dirigido a los dispositivos cuyo sistema operativo se encuentre basado en Android.

Capítulo
2. Definición del problema

En este capítulo, se expondrá de forma concisa el problema principal que impulsa esta investigación, explorando su importancia y alcance. Se analizarán los elementos fundamentales para comprender su naturaleza y aportar posibles soluciones.

2.1. Identificación del problema real

El ámbito educativo se ha beneficiado enormemente de los avances tecnológicos en las últimas décadas, y la integración de herramientas digitales en la enseñanza se ha convertido en un recurso esencial para mejorar la comprensión y el aprendizaje de conceptos complejos. En este contexto, se plantea la cuestión de cómo mejorar la enseñanza y el entendimiento de los algoritmos de planificación de procesos haciendo uso de estas nuevas tecnologías.

En las aulas universitarias, la comprensión de los algoritmos de planificación de procesos es fundamental para formar a futuros profesionales capaces de optimizar el rendimiento de sistemas informáticos. Sin embargo, esta área a menudo presenta desafíos conceptuales que pueden ser difíciles de abordar únicamente a través de métodos tradicionales de enseñanza. Aquí es donde surge la necesidad de una herramienta que proporcione una experiencia interactiva y visual para el aprendizaje de estos algoritmos, a la vez que permitiría al alumnado practicar diferentes ejercicios en cualquier localización.

El objetivo central de este TFG es desarrollar una aplicación móvil para dispositivos Android que simule de manera interactiva diversos algoritmos de planificación de procesos. Esta aplicación se concibe como una herramienta complementaria a la educación en la universidad, diseñada para brindar a los estudiantes una comprensión práctica y tangible de cómo funcionan estos algoritmos en la gestión de la CPU.

La necesidad de esta aplicación se fundamenta en la idea de que la simulación visual y práctica puede simplificar conceptos complejos y mejorar la retención del conocimiento. Al proporcionar a los estudiantes la capacidad de interactuar con diferentes escenarios de planificación de procesos, se espera que la aplicación facilite la asimilación de conceptos teóricos y, en última instancia, mejore la calidad de la educación en este campo.

A lo largo de este apartado, se analizará en mayor profundidad la relevancia de esta problemática, se identificarán los beneficios potenciales de la solución propuesta y se delinearán las metas y objetivos específicos que guiarán el desarrollo de la aplicación.

2.2. Identificación del problema técnico

En esta sección, se abordará el desafío técnico central que enfrenta la creación de una aplicación simuladora de algoritmos de planificación de procesos para dispositivos Android, empleando como guía la PDS (*Product Design Specification*). Se explorarán las complejidades inherentes a la implementación y visualización precisa de estos algoritmos en un entorno móvil, considerando aspectos como eficiencia, interactividad y experiencia del usuario. Este análisis técnico sentará las bases para diseñar soluciones efectivas que permitan superar los obstáculos técnicos y lograr los objetivos de la aplicación educativa propuesta.

2.2.1. Funcionamiento

Respecto al funcionamiento de la aplicación se dispondrá de las siguientes funcionalidades:

1. **Selección de algoritmos**: La aplicación ofrecerá la posibilidad de elegir entre distintos algoritmos de planificación de procesos, permitiendo al usuario especificar la metodología que desea estudiar.

2. **Ingreso de datos precisos**: En cada instancia, se proporcionarán campos requeridos para una entrada exhaustiva de datos pertinentes al funcionamiento del algoritmo seleccionado. Esto asegurará una simulación precisa y detallada.

3. **Flexibilidad en la configuración de procesos**: Se habilitará la modificación de procesos añadidos, posibilitando la ejecución de múltiples simulaciones con diferentes datos o algoritmos, fomentando un análisis comparativo.

4. **Visualización avanzada de resultados**: La aplicación ofrecerá diversas modalidades de presentación de resultados para facilitar el entendimiento del usuario. Esto incluirá tablas de datos y diagramas temporales, contribuyendo a una representación clara de la información.

5. **Múltiples pantallas focalizadas**: Se dispondrá de distintas vistas para permitir al usuario realizar todas las tareas necesarias adecuadas a esta herramienta, siendo estas: la introducción de datos como módulo principal, y la visualización de resultados, la visualización temporal de la simulación y la visualización de las colas de procesos como módulos para el análisis de resultados.

6. **Diagrama de tiempos de la cola de procesos en CPU**: Una de las vistas permitirá la observación detallada del diagrama temporal que ilustra la dinámica de la cola de procesos en la CPU.

2.2.2. Entorno

La aplicación se desplegará en un entorno operativo a partir de la versión 7.0 de Android o posteriores. El acceso a esta herramienta se proporcionará principalmente en formato "*.apk*" accesible desde el propio repositorio de GitHub asociado [23]. Los usuarios también podrán acceder y adquirir la aplicación a través de la plataforma oficial de distribución de aplicaciones, la *Play Store* de Google. Este entorno asegura la compatibilidad con dispositivos que cumplen con los requisitos de sistema establecidos por Android 7.0 y proporciona una amplia accesibilidad a través de una fuente confiable y reconocida para los usuarios.

2.2.3. Vida esperada

El ciclo de vida de la aplicación no está restringido por un límite temporal predeterminado, siempre y cuando se cumplan los requisitos mínimos de compatibilidad con el sistema operativo Android 7.0 o versiones superiores.

Al establecer la compatibilidad con versiones actuales y futuras del sistema operativo Android, junto con la posibilidad de mantenimiento y actualizaciones, la aplicación puede mantener su relevancia y utilidad sin una restricción temporal rígida. La duración del ciclo de vida dependerá en gran medida de su utilidad continuada para los usuarios y su capacidad de adaptación a los cambios tecnológicos y educativos.

2.2.4. Ciclo de mantenimiento

En el alcance de este proyecto, se concibe el producto final como una versión base con la que futuros desarrolladores puedan trabajar y ampliar fácilmente las funcionalidades propuestas. Aunque no se contempla un ciclo de mantenimiento continuo, se implementará un diseño de aplicación altamente modular.

Este enfoque modular permitirá abordar de manera eficiente la corrección de posibles fallos en el futuro y la adición de nuevas funcionalidades. La estructura modular garantizará que cualquier cambio o mejora se pueda llevar a cabo con el menor esfuerzo y perturbación posible en la funcionalidad principal del producto.

2.2.5. Competencia

Dentro del panorama actual, se constata la existencia de diversas alternativas a la presente aplicación, abarcando incluso distintas soluciones, además de hacer la aplicación a medida según se explican los ejercicios de la asignatura de Sistemas Operativos del Grado de Informática de la Universidad de Córdoba. Sin embargo, el enfoque central de este proyecto no radica en competir con estas alternativas, sino en unir las características destacadas identificadas en dichas soluciones. Estas cualidades positivas serán expuestas en detalle en el próximo capítulo de Antecedentes (ver Capítulo 4. Antecedentes).

La visión primordial consiste en concebir una aplicación que capture la esencia distintiva de la Universidad de Córdoba, con un doble propósito: servir como una herramienta complementaria al proceso docente y como un punto de partida modular, permitiendo a futuros alumnos expandir y enriquecer sus funcionalidades como parte de sus propios TFGs.

2.2.6. Aspecto externo

Con relación al aspecto de la aplicación, se buscan destacar los siguientes aspectos:

1. **Diseño amigable e intuitivo**: Se persigue lograr un diseño visual altamente amigable para el usuario, con el objetivo de que la aplicación sea intuitiva en su uso, sin requerir la necesidad de consultar manuales. El diseño se centrará en una experiencia de usuario fluida y natural.

2. **Paleta de colores minimalista y representativa**: Los colores empleados han sido cuidadosamente seleccionados para adoptar un enfoque minimalista, al mismo tiempo que se mantienen en sintonía con la identidad visual característica de la

universidad [5]. Esta elección cromática contribuirá a una coherencia visual y a una identificación inmediata con la institución.

3. **Visualización mediante paginación continua**: La disposición de las vistas de la aplicación se implementará en un formato de paginación continua. Esta elección busca brindar al usuario la sensación de interactuar con una herramienta compacta y organizada, facilitando la navegación y el acceso a las distintas funcionalidades de manera fluida.

En conjunto, estos elementos de diseño contribuirán a la creación de una aplicación visualmente atractiva, fácil de usar y en sintonía con la estética de la universidad, promoviendo una experiencia de usuario positiva y efectiva.

2.2.7. Estandarización

En pro de garantizar la coherencia, eficiencia y calidad en el desarrollo de la aplicación, se han establecido rigurosos estándares y prácticas de estandarización. Estos elementos fundamentales contribuirán a mantener la integridad y la robustez del proyecto, así como a facilitar futuras expansiones y mantenimiento. Entre las principales áreas de estandarización se incluyen:

1. **Nomenclatura uniforme**: Se ha adoptado una nomenclatura coherente y descriptiva para las variables, funciones y clases en el código fuente. Esto permitirá una comprensión más rápida y precisa de la estructura del proyecto, optimizando la posible futura colaboración de nuevos desarrolladores.

2. **Documentación detallada**: Se ha implementado una documentación exhaustiva que abarca la estructura general del código, las funcionalidades clave y los algoritmos empleados. Esta documentación servirá como recurso esencial para futuros desarrolladores y como guía para la expansión y optimización del proyecto.

3. **Comentarios y anotaciones**: El código se encuentra enriquecido con comentarios claros y anotaciones pertinentes que explican procesos, decisiones de diseño y secciones críticas. Estas adiciones permitirán un mayor nivel de legibilidad y comprensión para quienes trabajen en el código en etapas posteriores.

4. **Diseño responsivo**: La interfaz de usuario se ha desarrollado siguiendo principios de diseño responsivo [24], garantizando una experiencia visual y funcional coherente en una variedad de tamaños de pantalla.

5. **Pruebas y validación**: Se han establecido procedimientos de pruebas unitarias y de integración exhaustivas para verificar la funcionalidad y calidad del código (ver Capítulo 11. Pruebas). Estas pruebas se aplicarán tanto a las características existentes como a las nuevas implementaciones, asegurando un alto estándar de rendimiento y eficacia.

6. **Control de versiones**: La gestión de versiones se lleva a cabo utilizando un sistema de control de versiones (*Git*), lo que permite un seguimiento preciso de los cambios realizados en el código con el tiempo. Esto facilita la colaboración y proporciona la capacidad de revertir cambios si fuera necesario.

7. **Compatibilidad**: La aplicación se ha diseñado siguiendo las mejores prácticas de desarrollo y optimización de Android, intentando siempre asegurar la compatibilidad con diferentes versiones de Android (versión 7.0 y posteriores) y dispositivos, lo que garantiza un rendimiento óptimo en una variedad de contextos.

8. **Actualizaciones futuras**: La arquitectura de la aplicación se ha concebido de manera modular, lo que permitirá incorporar nuevas funcionalidades, mejoras y correcciones de errores sin comprometer la estabilidad general de la aplicación (ver Capítulo 13. Futuras Mejoras).

9. **Usabilidad**: El diseño de la interfaz de usuario sigue los principios de usabilidad y diseño centrado en el usuario, garantizando interacciones intuitivas y fluidas que mejoren la experiencia general.

Estos elementos de estandarización se han implementado con el objetivo de asegurar la calidad, la flexibilidad y la continuidad en el desarrollo y evolución de la aplicación, aportando una base sólida para su presente y futuro.

2.2.8. Calidad y fiabilidad

La calidad y fiabilidad de la aplicación son aspectos esenciales para garantizar su funcionalidad, estabilidad y aceptación por parte de los usuarios. En este contexto, se han

adoptado medidas exhaustivas que abarcan desde el proceso de desarrollo hasta las pruebas y validaciones rigurosas. Estos enfoques se han diseñado con el propósito de asegurar una experiencia de usuario superior y una aplicación confiable en todo momento.

1. **Pruebas exhaustivas**: La aplicación ha sido sometida a pruebas extensas para verificar su comportamiento bajo diferentes escenarios y condiciones.

2. **Control de calidad continuo**: Se ha implementado un proceso de control de calidad continuo que incluye revisión y supervisión regular del progreso del desarrollo. Esto garantiza que el código cumpla con los estándares de calidad establecidos y que los problemas se detecten y aborden tempranamente.

3. **Optimización de rendimiento**: Se han realizado esfuerzos para optimizar el rendimiento de la aplicación, y buscar siempre tiempos de respuesta rápidos y correcta eficiencia en la ejecución de procesos. Esto se traduce en una experiencia fluida y ágil para el usuario.

4. **Gestión de errores y excepciones**: Se ha implementado un sistema sólido de manejo de errores y excepciones para anticipar y controlar situaciones inesperadas. Esto contribuye a la estabilidad de la aplicación y minimiza posibles fallas críticas.

5. **Compatibilidad de plataforma**: La aplicación ha sido diseñada para cumplir con los requisitos de Android 7.0 y versiones posteriores, garantizando su funcionamiento en un amplio rango de dispositivos.

2.2.9. Programa de tareas

Se llevará a cabo un proceso de desarrollo riguroso y estructurado que se dividirá en varias fases clave. Estas fases serán de vital importancia para la consecución de los objetivos planteados, y permitirán abordar de manera sistemática y eficiente cada uno de los aspectos que conforman el proyecto en su totalidad.

- **Estudio y análisis del problema**: En la fase de estudio y análisis del problema, se llevará a cabo una evaluación exhaustiva de los requisitos necesarios para la creación de una aplicación Android (ver Capítulo 7. Especificación de requisitos).

En primer lugar, se realizará una revisión detallada de las funcionalidades requeridas para la correcta ejecución de los algoritmos, teniendo en cuenta que la aplicación va destinada al apoyo a la docencia de la asignatura de sistemas operativos, y por tanto de los tipos de ejercicios sobre planificación que en ella se proponen. A su vez, se llevará a cabo una investigación sobre las características más relevantes de los algoritmos de planificación de procesos. En esta fase también se tendrán en cuenta los aspectos técnicos necesarios para el correcto funcionamiento de la aplicación, como el uso de herramientas y lenguajes de programación específicos para Android o la selección de una arquitectura adecuada, entre otros.

- **Diseño**: En la fase de diseño (ver Capítulo 10. Diseño de la interfaz) se llevará a cabo la conceptualización de la aplicación a desarrollar. Se definirán los componentes y elementos necesarios para la creación de una aplicación intuitiva y atractiva que permita a los usuarios comprender y utilizar la aplicación de manera eficiente.

  Asimismo, se establecerán las especificaciones técnicas para la programación de la aplicación, la elección de la arquitectura.

  La fase de diseño permitirá establecer las bases para la programación de una aplicación robusta y eficiente que cumpla con los requerimientos definidos en la fase de estudio y análisis del problema previos.

- **Codificación**: En la fase de codificación se llevará a cabo la implementación de la aplicación de simulación, siguiendo las especificaciones definidas en la fase de diseño. Los lenguajes de programación más comunes para el desarrollo de aplicaciones Android son *Java* y *Kotlin* [7], aunque será este último el seleccionado para este desarrollo debido a que es el lenguaje actual de desarrollo para Google y Android y tiene una gran cantidad de documentación y ejemplos de estudio. Además, se utilizarán herramientas de desarrollo integrado (*IDE*) para agilizar y facilitar el proceso de codificación. *Android Studio* [6] es el IDE oficial de Android y ofrece una amplia variedad de herramientas para el desarrollo de aplicaciones. Este punto se ampliará en secciones posteriores.

- **Pruebas**: En la fase de pruebas (ver Capítulo 11. Pruebas), se llevará a cabo un proceso de evaluación exhaustivo de la aplicación con el objetivo de detectar y corregir posibles errores o fallos en su funcionamiento. Se probará la aplicación en diferentes dispositivos móviles, tanto en móviles como en tabletas, para asegurar que la

aplicación es compatible y funciona correctamente en diferentes tamaños de pantalla y resoluciones. Para llevar a cabo estas pruebas, se utilizarán herramientas de prueba y depuración, como el depurador de Android Studio o herramientas de automatización de pruebas, para facilitar el proceso y garantizar la precisión y calidad de las pruebas.

- **Documentación**: En esta fase, se elaborarán tres manuales esenciales para la comprensión y mantenimiento de este proyecto.

  En primer lugar, se desarrollará un manual técnico que describirá los detalles técnicos del proyecto, incluyendo la arquitectura, diseño y funcionamiento de la aplicación. Este manual deberá ser utilizado por desarrolladores futuros para mantener y mejorar el proyecto.

  En segundo lugar, se creará un manual de usuario que explicará cómo utilizar la aplicación, detallando las diferentes funcionalidades, pantallas y opciones. Este manual será útil para que los usuarios puedan utilizar la aplicación y comprender su funcionamiento.

  Por último, se redactará un manual de código que describirá el código fuente del proyecto. Este manual será utilizado por otros desarrolladores que deseen trabajar con el proyecto en el futuro.

2.2.10. Pruebas

La realización de pruebas exhaustivas juega un papel fundamental en la creación de una aplicación robusta y confiable. En este contexto, se adopta un enfoque integral para garantizar que cada aspecto de la aplicación funcione correctamente y que los problemas se identifiquen y aborden de manera efectiva. Se destacan los siguientes puntos clave en el proceso de pruebas:

1. **Pruebas independientes por módulo**: Para garantizar la integridad de cada módulo individual, se realizan pruebas independientes para evaluar su funcionamiento. Esto se hace con el fin de detectar cualquier problema antes de que pueda propagarse a otras áreas de la aplicación.

2. **Pruebas de integración**: Una vez que se han verificado los módulos de forma individual, se procede a las pruebas de integración. Estas pruebas validan el

funcionamiento conjunto de los módulos, asegurando que la interacción entre ellos sea fluida y coherente.

3. **Herramientas de *Android Studio***: Android Studio ofrece un conjunto de herramientas poderosas para facilitar el proceso de pruebas. La depuración en tiempo real y la simulación de ejecución en diversos dispositivos virtuales permiten identificar y solucionar problemas de manera eficiente.

4. **Pruebas en dispositivos físicos**: Además de las pruebas en dispositivos virtuales, se llevan a cabo pruebas en dispositivos físicos. Esto proporciona una visión más precisa del rendimiento y la experiencia del usuario en condiciones reales.

5. **Verificación de compatibilidad**: Se verifica la compatibilidad de la aplicación en diferentes versiones de Android y dispositivos con diversas configuraciones. Esto garantiza que la aplicación funcione de manera óptima en una variedad de escenarios.

6. **Registro de errores**: Se registra y documenta cualquier error encontrado durante las pruebas, lo que facilita su seguimiento y posterior corrección.

7. **Iteraciones y mejoras**: Las pruebas son un proceso iterativo. Cualquier problema identificado se corrige, se realizan nuevas pruebas y se repite el ciclo hasta que la aplicación alcance un nivel de calidad óptimo.

Capítulo
3. Objetivos

En esta sección, se delinean los objetivos del proyecto, distinguiendo entre los aspectos formales y operacionales. Los objetivos formales definen las metas generales del proyecto, mientras que los operacionales detallan las acciones específicas para lograr esas metas.

3.1. Objetivos formales

En esta sección, se establecen los objetivos formales del proyecto, trazando la dirección y el propósito del proyecto.

1. **Adquisición de conocimientos de desarrollo Android:** Obtener una comprensión profunda de las complejidades intrínsecas a la creación de aplicaciones para los sistemas basados en Android, explorando con detalle cada etapa del proceso de desarrollo.

2. **Manejo de *Android Studio*:** Adquirir experiencia en el manejo de Android Studio, la plataforma de desarrollo principal para la creación de aplicaciones en el ecosistema Android.

3. **Iniciación en el lenguaje de programación *Kotlin*:** Adquirir un nivel adecuado en el lenguaje de programación Kotlin, utilizando su potencial para el desarrollo efectivo de la aplicación.

4. **Aprendizaje en *Jetpack Compose*:** Familiarizarse en el uso de Jetpack Compose [8], la biblioteca de diseño de interfaces de última generación, para desarrollar experiencias de usuario intuitivas y modernas.

5. **Optimización de Rendimiento:** Implementar estrategias avanzadas para optimizar los recursos y lograr un rendimiento adecuado en diversos escenarios de uso.

6. **Generación de Documentación Completa:** Elaborar una documentación completa y detallada que abarque todos los aspectos del proyecto, desde la concepción hasta la implementación, asegurando la transparencia y la replicabilidad del proceso.

3.2. Objetivos operacionales

Los objetivos operacionales detallan las acciones y pasos concretos que se deben emprender para alcanzar los objetivos formales. A continuación, se expondrán estos objetivos operacionales, primeramente, indicando el objetivo principal de la aplicación y posteriormente refinándolo en una serie de objetivos secundarios o subobjetivos.

3.2.1. Objetivo principal

El objetivo primordial del proyecto radica en la concepción y desarrollo integral de una aplicación dirigida a dispositivos móviles basados en el sistema operativo Android. Esta aplicación se concibe con el propósito de desempeñar un papel fundamental como herramienta de respaldo en la enseñanza de los conceptos relacionados con la planificación de procesos, específicamente en el contexto de la asignatura de 2º de Grado de Ingeniería Informática en la Universidad de Córdoba.

Uno de los pilares distintivos de la aplicación es su capacidad para simular los eventos de E/S (entrada/salida) de los procesos. La implementación de uno o dos algoritmos de planificación se sitúa como foco central del desarrollo. Esta selección se deriva de la aspiración de construir una base sólida y funcional, que pueda evolucionar en el tiempo. La aplicación base se concibe como robusta y adaptable, permitiendo la incorporación futura de un repertorio más amplio de características y tipos de algoritmos, de acuerdo con las necesidades y objetivos de la educación en el ámbito de la planificación de procesos.

3.2.2. Objetivos específicos

En esta sección, se detallan los objetivos específicos que guiarán el desarrollo de la aplicación, abarcando desde la usabilidad y simulación de algoritmos de planificación hasta la presentación efectiva de resultados.

OBJ-1. Diseño compacto y accesible: Crear una interfaz de usuario que presente información de manera concisa y accesible. Se asegurará que los elementos clave estén visibles y organizados de manera intuitiva para que los usuarios puedan acceder

a ellos sin necesidad de entrar en submenús adicionales. Se diseñará una navegación fluida y lógica que permita a los usuarios moverse sin problemas entre los diferentes módulos de la aplicación.

En la perspectiva de la navegación, se ha seleccionado un diseño de formato paginado, donde cada módulo o sección se encuentra en una página individual, permitiendo una accesibilidad fluida mediante un desplazamiento lateral.

OBJ-2. Implementación de algoritmos: Implementación de un algoritmo inicial sólido que forme la base de la aplicación y facilitando la incorporación de nuevos algoritmos en el futuro, de forma que los desarrolladores y educadores puedan expandir la gama de algoritmos disponibles sin problemas, enriqueciendo así la experiencia de aprendizaje.

OBJ-3. Gestión de los datos de planificación: Desarrollar herramientas claras y precisas para que los usuarios ingresen los datos necesarios para cada algoritmo. Proporcionar formularios estructurados que guíen al usuario a introducir los valores requeridos de manera eficiente y unívoca.

OBJ-4. Gestión de eventos de E/S: Diseñar un sistema que permita a los usuarios especificar eventos de entrada/salida para cada proceso agregado. Se dispondrá de campos específicos para indicar el momento en el que el proceso a agregar realiza la acción de entrada y la acción de salida del evento.

OBJ-5. Visualización clara de datos ingresados: Desarrollar los sistemas necesarios para que el usuario pueda revisar adecuadamente la información que ha introducido, permitiendo verificar la correcta inserción de los datos.

OBJ-6. Gestión de procesos agregados: Implementar la capacidad de eliminar procesos individualmente o en su totalidad. Se proporcionarán controles intuitivos que permitan a los usuarios administrar los procesos agregados según sus necesidades.

OBJ-7. Visualización numérica de resultados: Desarrollar un módulo que muestre los resultados numéricos obtenidos de la simulación. Presentar los datos (tiempo de inicio, tiempo de espera, tiempo de respuesta, etc.) en un formato tabular para que los usuarios puedan analizar los resultados de manera efectiva.

OBJ-8. Visualización gráfica de tiempos: Crear un módulo que permita a los usuarios ver los tiempos de CPU asociados a cada proceso mediante representaciones gráficas (en un diagrama de Gantt [9]), así como los estados en los que se encuentra en cada momento.

OBJ-9. Visualización del estado de las colas de procesos: Implementar un módulo que muestre el estado de las diferentes colas de procesos en cada etapa de la simulación. Cada cola de procesos representará el orden de los procesos para cada momento de la simulación, permitiendo al usuario comprender la organización seguida por el algoritmo seleccionado.

Capítulo
4. Antecedentes

En el contexto del desarrollo de aplicaciones Android, los antecedentes se refieren a las soluciones previas, herramientas y enfoques utilizados en proyectos similares que han abordado problemáticas análogas.

En el caso específico de esta propuesta, se han explorado diversas opciones y se ha investigado ampliamente en el ámbito de la planificación de procesos, considerando alternativas tanto en el entorno de dispositivos móviles como en otras plataformas (páginas web o computadoras). Esto ha implicado un análisis minucioso de soluciones disponibles en sistemas operativos, *software* educativo y aplicaciones de simulación en general. Esta exploración se ha extendido también a plataformas distintas a los dispositivos móviles, en la búsqueda de enfoques exitosos y transferibles que puedan enriquecer la propuesta.

Este proceso de investigación y exploración de alternativas ha brindado una base sólida para identificar las fortalezas y debilidades de soluciones existentes, destacando así las oportunidades para mejorar y diferenciar la aplicación que se está desarrollando. La amalgama de conocimiento obtenida de diversos antecedentes ha permitido perfilar una propuesta que capitaliza las mejores prácticas y enriquece la propuesta con características distintivas y eficaces.

4.1. Sitios Web Online

Se han examinado plataformas que ofrecen simuladores de planificación de procesos accesibles directamente desde el navegador, eliminando la necesidad de instalar *software* adicional en los dispositivos de los usuarios. A continuación, se presentan algunos ejemplos de sitios web que hospedan tales simuladores, permitiendo una visión más amplia de las opciones disponibles en el mercado y proporcionando una base sólida para la propuesta en desarrollo.

4.1.1. Boonsuen Process Scheduling Solver

Este simulador pertenece al autor "Boonsuen" en GitHub [4], donde se podrá revisar el código utilizado por el simulador para poder comprobar la programación y el funcionamiento de cada uno de los métodos utilizados.

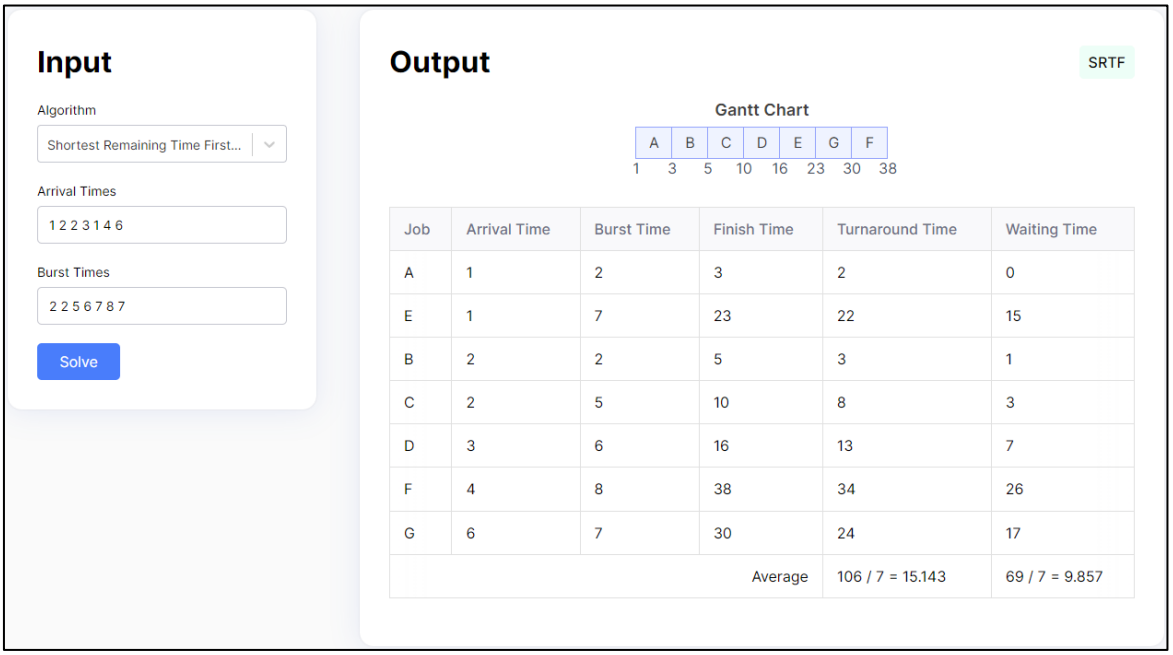

Figura 1. Ejemplo de ejecución del simulador de Boonsuen.

De forma sencilla permite añadir los tiempos pertenecientes a cada uno de los procesos que se desean simular, así como seleccionar el método de planificación deseado (*First Come First Service*, *Shortest Job First*, *Shortest Remaining Time First*, *Round-Robin* o *Priority*, y algunas variantes en versión *preemptive* o "preventiva"), como se puede observar en la figura 1.

A continuación, se muestra una lista de las principales ventajas e inconvenientes o carencias identificados tras un análisis de la aplicación:

Ventajas:

- Buena portabilidad al poder utilizarse y visualizar la información correctamente desde cualquier dispositivo.
- Se destaca su sencillez y facilidad de uso.
- La página contiene enlaces de ayuda y soporte que redirige al usuario a manuales sencillos que explican el uso de la herramienta eficazmente.
- Posee una correcta variedad de algoritmos implementados.

Inconvenientes:

- La representación visual es algo limitada.
- No se observa que se pueda indicar la realización de operaciones de E/S.
- No se permite la modificación de datos ya introducidos.

4.1.2. AnimOS *CPU-Scheduling*

La Universidad Tecnológica de Dortmund ha desarrollado un simulador bastante completo en cuanto a visualización de resultados [11], pudiendo no sólo simular la organización de los procesos, sino que también permite comprobar paso a paso cómo afecta el método seleccionado a dicha organización (ver figura 2).

En cuanto a los algoritmos implementados, éstos únicamente se corresponden con FIFO, SJF y RR, por lo que se puede encontrar cierta limitación en este sentido.

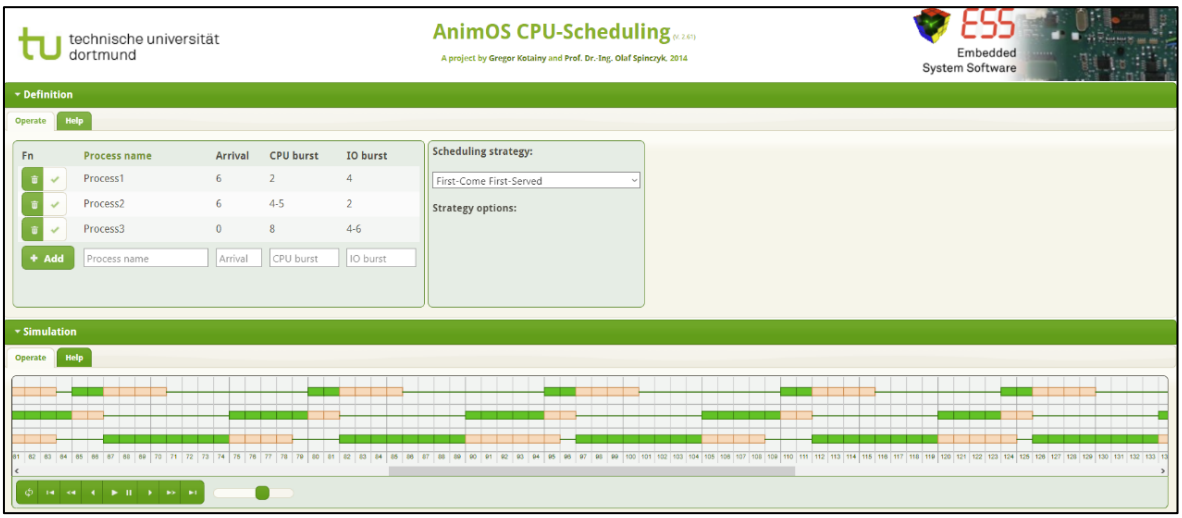

Figura 2. Ejemplo de ejecución del simulador de la Universidad de Dortmund.

La siguiente lista presenta las principales ventajas e inconvenientes de la aplicación identificadas:

Ventajas:

- La simplicidad y claridad en su diseño hacen de este simulador una herramienta muy usable.
- La visualización de resultados es completa y sencilla.

Inconvenientes:

- La legibilidad de los resultados se ve comprometida en dispositivos móviles con pantallas de menor tamaño.
- Carece de cualquier tipo de documentación disponible para el usuario.
- No se observa que se pueda indicar la realización de operaciones de E/S.

4.1.3. *CPU Scheduling Simulator by* Hirusha Cooray

Este sencillo simulador [6], hace uso de los cuatro principales algoritmos de planificación: FIFO (ver figura 3), SJF, SRTF y RR.

Tras su análisis, se identificaron las principales ventajas e inconvenientes de la aplicación que se presentarán a continuación:

Ventajas:

- Uno de los puntos fuertes de este simulador es su usabilidad y accesibilidad.
- Implementa los algoritmos más frecuentemente estudiados.
- La visualización de resultados es responsiva al tamaño de la pantalla.

Inconvenientes:

- La información resultante es precaria y bastante simple.
- No presenta documentación alguna sobre su funcionamiento.
- No se observa que se pueda indicar la realización de operaciones de E/S.
- No se permite la modificación de datos ya introducidos.

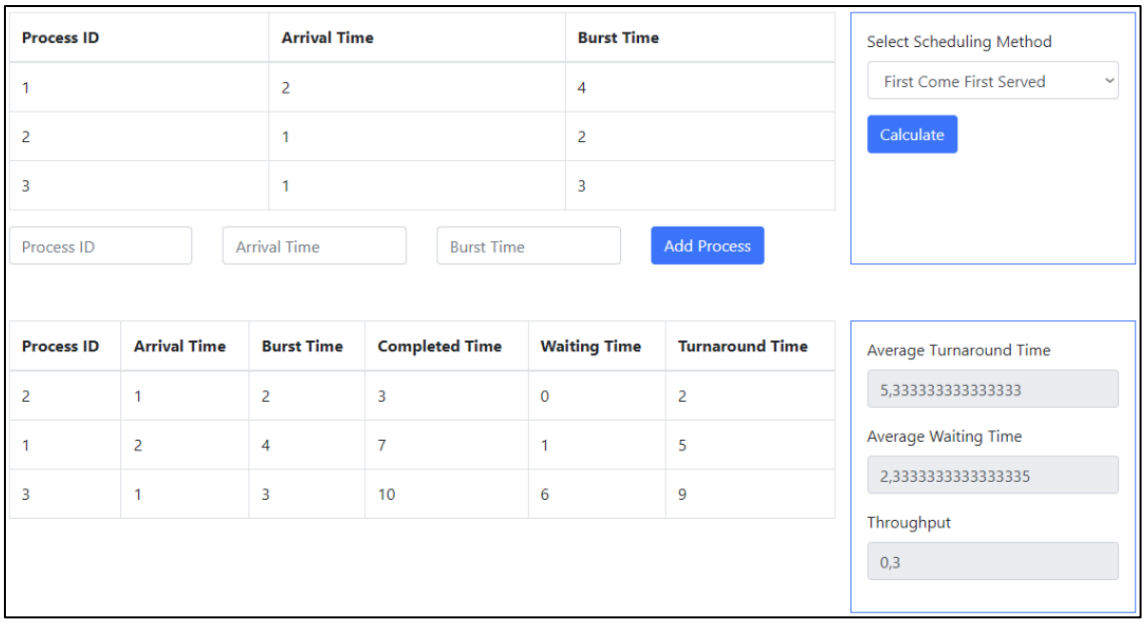

Figura 3. Ejemplo sencillo de ejecución del simulador.

4.2. Aplicación descargable para ordenador personal

A continuación, se ha considerado el *software* creado para ser descargado e instalarlo en cualquier computadora personal. De esta forma podrá ser ejecutado en dicho dispositivo sin necesidad de ningún navegador o una conexión a la red. Cabe destacar que, a la hora de realizar el estudio, se ha tenido en cuenta la independencia de los simuladores respecto al sistema operativo en el que se ejecutan, con el fin de evitar mayores limitaciones.

4.2.1. *Alvareztech*

Aplicación básica y sencilla (ver figura 4), desarrollada por "Alvareztech" [7] en Java, que realiza una pequeña simulación de procesos utilizando dos de los algoritmos mencionados anteriormente, FIFO y RR, además de uno extra el cual resulta ser algo menos frecuente de encontrar implementado, *Highest Response Ratio Next*.

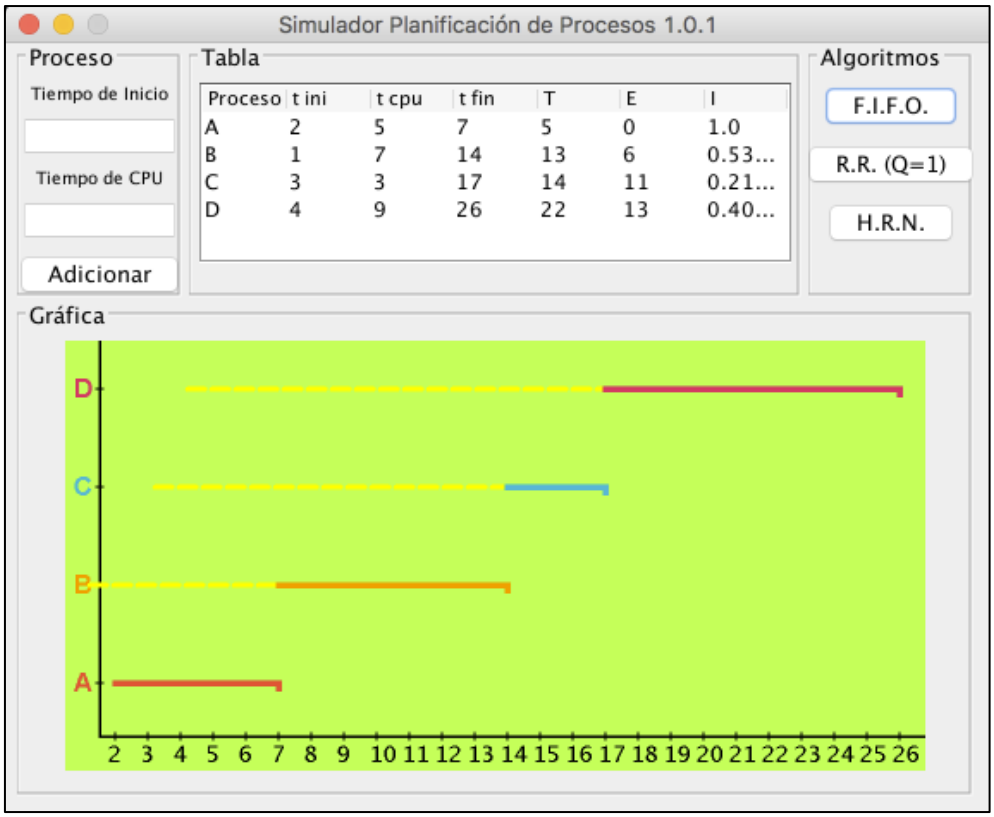

Figura 4. Ejemplo de ejecución del algoritmo FIFO.

Se muestra una lista con las principales ventajas e inconvenientes de la aplicación tras su análisis:

Ventajas:

- Es una herramienta sencilla, usable y fácil de instalar.
- Contiene gráficas que apoyan la visualización de los resultados.

Inconvenientes:

- Los algoritmos implementados son limitados.
- La documentación encontrada es pobre y no contiene desarrollo en el uso de la herramienta, así como en las funcionalidades implementadas.
- No se observa que se pueda indicar la realización de operaciones de E/S.

4.2.2. *IncanatoIT*

Para este caso, el autor en GitHub IncanatoIT [8] ha desarrollado una aplicación de uso sencillo con la particularidad de la visualización de los tiempos de cada proceso en diferentes formatos de tablas, como se observa en la figura 5 y figura 6.

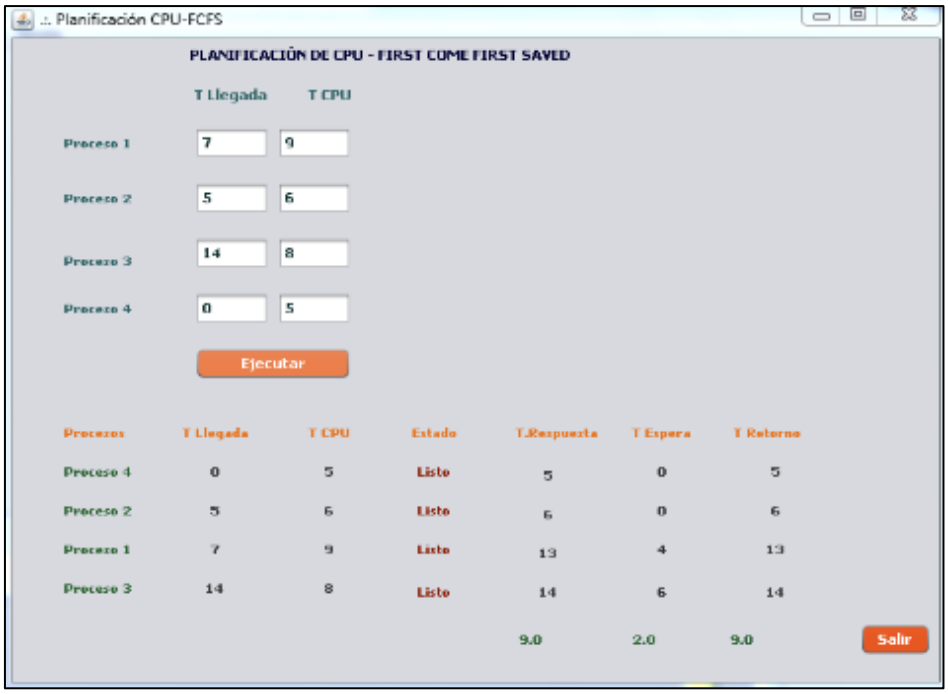

Figura 5. Ingreso de datos para simular los procesos.

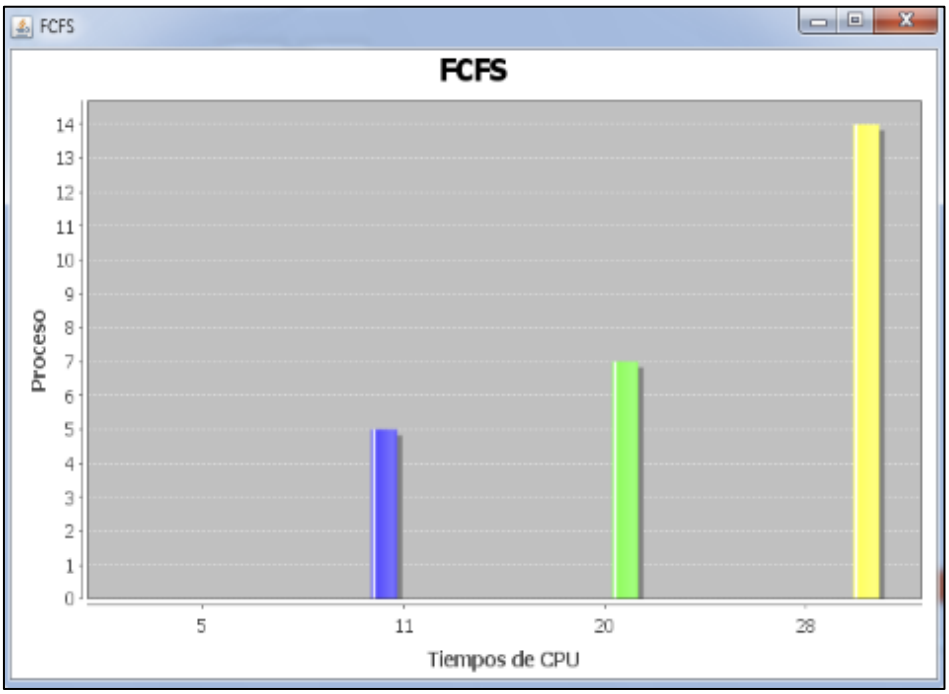

Figura 6. Gráfico resultante de la simulación.

En cuanto a los algoritmos implementados, se puede decir que el simulador hace uso de una variedad interesante de estos, aunque sacrifica la implementación de algunos más básicos en pro de agregar una mayor variedad. Estos se corresponden con los algoritmos FIFO, SJF y RR, así como su variable con prioridades.

A continuación, se detallan las principales ventajas e inconvenientes de la aplicación tras su análisis:

Ventajas:

- Es una herramienta sencilla.
- Se ha encontrado una extensa documentación sobre su funcionamiento junto a unos breves resúmenes de la metodología de cada algoritmo implementado.
- Representación de resultados tanto en tabla como en gráfica.

Inconvenientes:

- La cantidad de algoritmos presentes es algo reducida.
- La interfaz gráfica presenta serias deficiencias estéticas y de usabilidad.
- No se observa que se pueda indicar la realización de operaciones de E/S.

4.2.3. *Ifreddyrondon*

El desarrollador en GitHub "Ifreddyrondon" [9] ha creado uno de los simuladores más completo de los ejemplos correspondientes a este apartado, el cual implementa los cuatro algoritmos básicos: FIFO, SJF, SRTF y RR. Se dispone de una gráfica que muestra el estado de los procesos en tiempo real, así como las diferentes estadísticas relacionadas con los tiempos (ver figura 7).

Figura 7. Ejemplo de simulación y selección del algoritmo FIFO.

A continuación, se expondrá una lista con las principales ventajas e inconvenientes de la aplicación halladas tras su análisis:

Ventajas:

- Facilidad de acceso a la instalación.
- Atractiva y clara representación visual con una extensa información.
- Documentación clara y sencilla respecto al uso de la herramienta.
- Implementación de los algoritmos más frecuentes.

Inconvenientes:

- La usabilidad es algo tosca debido a la interfaz dividida en desplegables.
- No se observa que se pueda indicar la realización de operaciones de E/S.

4.3. Dispositivos Móviles

Como finalización de este capítulo se tratarán algunos ejemplos de simuladores para dispositivos móviles, permitiendo así poder realizar dicha simulación sin necesidad de un ordenador o tener que recurrir a una visualización (probablemente precaria) de simuladores en sitios web mediante el navegador de estos dispositivos. Se han seleccionado las dos aplicaciones más representativas disponibles en *Play Store* de *Google*.

4.3.1. *CPU Simulator (CPU Scheduling)*

La aplicación publicada por *Jeico Games* [10] es una aplicación de interfaz sencilla que permite seleccionar el algoritmo, el número de procesos y sus tiempos, tanto manualmente como directamente importando un fichero con estos datos (ver figura 8), algo ciertamente interesante. Con ello, es capaz de mostrar gráficamente el comportamiento del algoritmo para cada proceso y cómo esto afecta a la CPU en cuestión.

Como característica a destacar, se puede observar que la flexibilidad presente en este simulador rivaliza con el analizado en el apartado anterior, ya que dispone de la implementación de numerosos algoritmos (FIFO, SJF, SRTF, RR, *Priority* y HRRN), considerando esto una clara ventaja respecto a sus iguales, e indicando que no es algo exclusivo de la plataforma anterior.

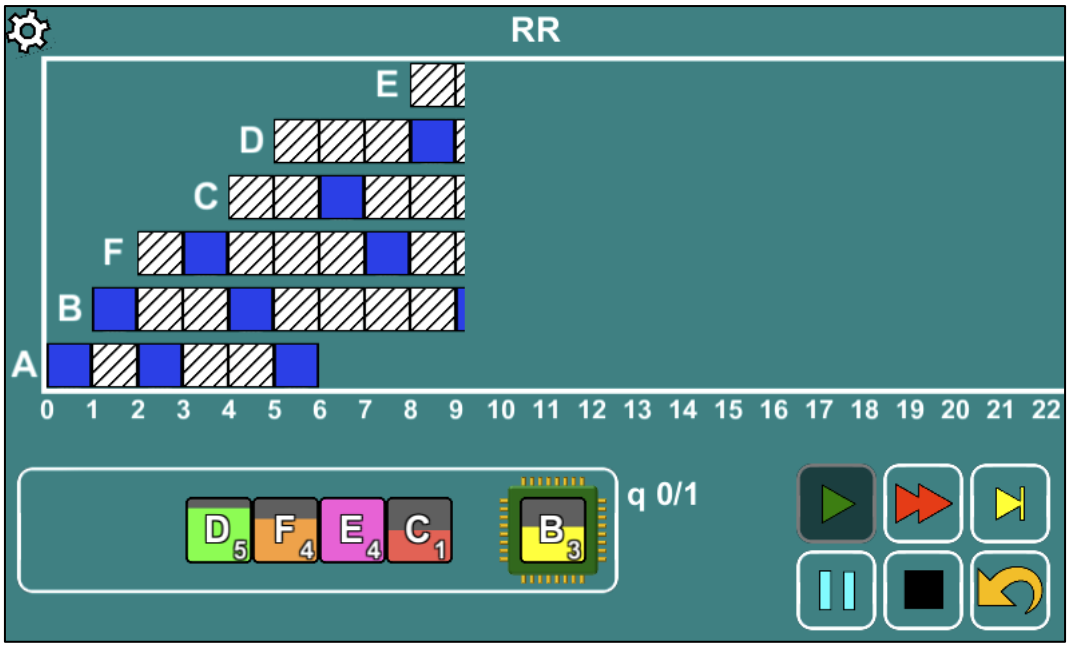

Figura 8. Ejemplo de simulación del algoritmo RR.

A continuación, tras la realización del análisis correspondiente, se detallan las principales ventajas e inconvenientes del simulador:

Ventajas:

- Facilidad de instalación.
- Gran número de algoritmos implementados.
- Exportación / Importación de resultados.

Inconvenientes:

- Representación visual algo confusa debido a la variedad de colores e iconos.
- Falta de documentación o explicación del funcionamiento completo.
- No se observa que se pueda indicar la realización de operaciones de E/S.

4.3.2. *Quantum*

Esta aplicación [11] (todavía en versiones de prueba) muestra una estética mucho más minimalista y elegante a la anterior (ver figura 9 y figura 10), haciendo buen uso de la sencillez para tener un uso intuitivo. Se encuentra un simulador que genera un diagrama de estructuras y un gráfico de ejecución de procesos con los datos introducidos.

Los algoritmos implementados en este caso corresponden con aquellos que más frecuencia se han encontrado en estas herramientas, siendo estos: FIFO, SJF, SRTF y RR.

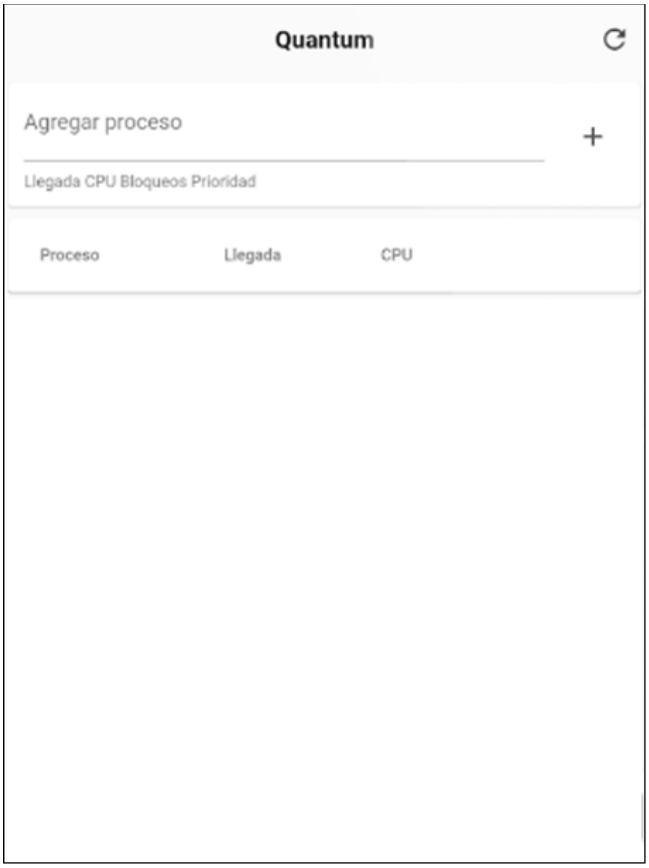

Figura 9. Selección de datos iniciales.

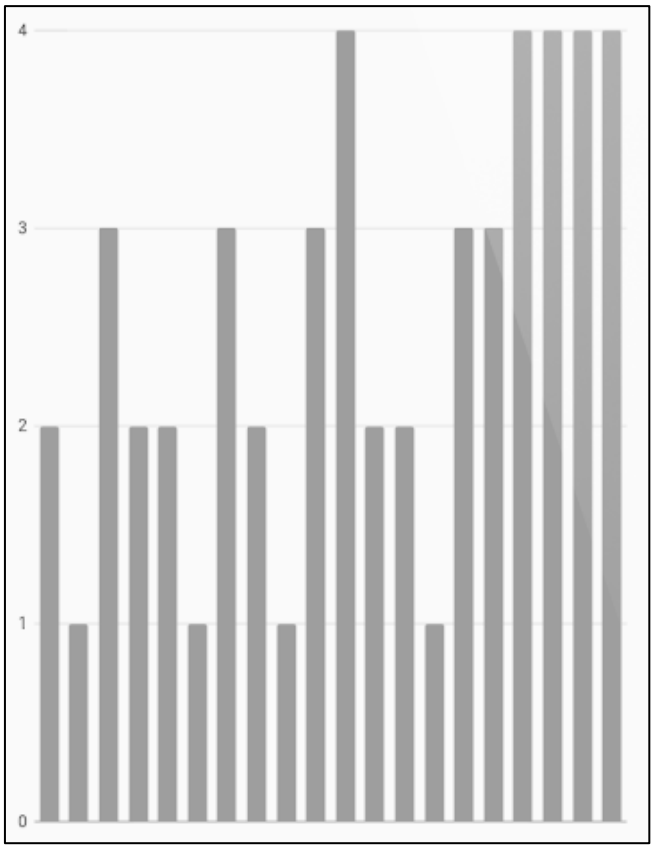

Figura 10. Resultado en gráfico de barras.

La siguiente lista presenta las principales ventajas e inconvenientes de la aplicación identificadas tras un análisis:

Ventajas:

- Descarga e instalaciones sencillas.
- Gran facilidad de uso.
- Representación visual sobria y sencilla.
- El conjunto de algoritmos implementados se conforma por los más frecuentes.
- Visualización de resultados en tabla y gráfica.

Inconvenientes:

- Falta de documentación clara del funcionamiento.
- No se observa que se pueda indicar la realización de operaciones de E/S.
- No permite modificación de datos ya introducidos.

4.4. Conclusiones

Al analizar los antecedentes, se ha destacado la importancia de no solo abordar la implementación técnica de algoritmos de planificación, sino también enfocarse en crear una aplicación atractiva, de alta usabilidad y accesibilidad, con un enfoque didáctico intrínseco. La adaptación de los conceptos teóricos al entorno práctico se ha convertido en un objetivo primordial, permitiendo una interacción significativa y efectiva con los usuarios.

En relación con los rasgos característicos observados en los antecedentes, se ha encontrado una oportunidad clara para marcar una diferencia. La consideración de estados y eventos de entrada/salida en la simulación se alinea directamente con los contenidos de la asignatura de Sistemas Operativos, enriqueciendo el valor educativo de la aplicación. Además, se ha identificado que la adaptabilidad de algunos simuladores web a diferentes tamaños de pantalla es limitada, lo que resalta la importancia de optimizar la presentación en dispositivos móviles.

Una característica adicional que se ha identificado como valiosa y que no todos los simuladores han considerado, es la posibilidad de modificar los datos introducidos sin necesidad de eliminar la totalidad de los datos y comenzar desde cero. Este enfoque más flexible permitirá a los usuarios ajustar y refinar los parámetros de manera eficiente, optimizando la experiencia de simulación.

Por tanto, la materialización de un simulador para la asignatura de Sistemas Operativos que no solo aborde los conceptos teóricos, sino que también incorpore elementos prácticos como estados y eventos de E/S, y facilite la modificación ágil de datos, se convierte en una herramienta de apoyo esencial. La implementación en dispositivos Android, además, abre la puerta a su utilización responsable durante las clases teóricas, generando un entorno interactivo y enriquecedor para el aprendizaje.

Capítulo
5. Restricciones

En el análisis y planificación del proyecto, resulta fundamental identificar y abordar las restricciones que pueden influir en su desarrollo y alcance. Este apartado abordará detalladamente dos aspectos clave: los factores de datos y los factores estratégicos. Cada uno de estos subapartados explorará las limitaciones y consideraciones que guiarán la toma de decisiones en la ejecución del proyecto.

5.1. Factores dato

Esta sección se adentra en las limitaciones fundamentales que influyen directamente en el proyecto en su totalidad. Estas restricciones no modificables tienen un impacto significativo en la planificación y ejecución del proyecto. El análisis exhaustivo de estos factores permitirá un enfoque claro y estratégico para la consecución de los objetivos, considerando cuidadosamente las circunstancias y restricciones inherentes al proyecto en su conjunto.

1. **Limitaciones de personal**:

La disponibilidad y capacidad del equipo humano asignado al proyecto son determinantes en su desarrollo. En este caso, se dispondrá únicamente del apoyo del director del proyecto, Juan Carlos Fernández Caballero.

2. **Limitaciones de tiempo**:

El tiempo disponible para completar el proyecto es un recurso valioso y limitado. La gestión efectiva de los plazos es esencial para mantener el proyecto en curso y garantizar una implementación exitosa.

Este proyecto tiene una cuantía de horas limitadas para su desarrollo (300 horas), según se establece en el reglamento de los TFGs de la Escuela Politécnica Superior de Córdoba, por lo que es un elemento importante a tener en cuenta para el desarrollo.

3. **Limitaciones de *hardware***:

Las restricciones en términos de recursos de *hardware* pueden afectar la eficiencia y el rendimiento del proyecto. Evaluar la capacidad de los dispositivos y equipos utilizados es crucial para optimizar la funcionalidad de la aplicación.

Se dispondrán de los recursos aportados por el proyectista, especificados más adelante en el apartado Recursos *hardware* (ver Capítulo 6. Recursos).

4. **Limitaciones de *software***:

Las limitaciones de las herramientas y plataformas de *software* utilizadas pueden influir en el desarrollo y funcionalidad del proyecto. Debido a ello y considerando la naturaleza principal del proyecto, se hará uso de *software* libre [18]. Este punto también se detallará en el apartado de Recursos *software* (ver Capítulo 6. Recursos).

5. **Limitaciones de rendimiento**:

El rendimiento de la aplicación es esencial para brindar una experiencia de usuario óptima. Considerar las limitaciones de rendimiento, como la capacidad de respuesta y velocidad, es fundamental para la satisfacción del usuario final.

Dado el propósito de desarrollar una herramienta que sea accesible desde una amplia gama de dispositivos Android, es imperativo optimizar la aplicación para garantizar una experiencia de usuario fluida y satisfactoria.

5.2. Factores estratégicos

En esta sección, se exploran las limitaciones y consideraciones estratégicas que pueden afectar la planificación y ejecución del proyecto en su conjunto. Al analizar estos factores, se garantiza una gestión eficaz del proyecto y se minimizan los riesgos potenciales. El entendimiento completo de las limitaciones estratégicas permite una toma de decisiones informada y la implementación exitosa de la aplicación, alineándola con los objetivos y expectativas establecidos.

1. **Sistema Operativo**:

La elección del sistema operativo influye en la compatibilidad y funcionalidad de la aplicación. La elección se ha inclinado hacia el sistema Android, dada su fácil accesibilidad para la mayoría de los estudiantes y docentes. Esta decisión se ha tomado considerando, también, las limitaciones que supone optar por una página web, que conllevaría el

mantenimiento de un dominio y la creación de un diseño adaptable a dispositivos de sobremesa y móviles.

2. **Versión de Android**:

La versión específica de Android determina las características y capacidades disponibles. Se han considerado los porcentajes de uso de las versiones disponibles proporcionadas oficialmente por los desarrolladores, optando por una versión que se encuentre en un término medio entre el porcentaje de uso y su antigüedad, siendo esta la versión 7.0, conocida también como "*Nougat*".

3. **IDE (Entorno Integrado de Desarrollo)**:

El IDE utilizado impacta en la eficiencia y calidad del desarrollo. Para este proyecto, se utilizará el entorno de desarrollo propuesto por los creadores de Android, siendo este Android Studio. Con ello, se obtiene una herramienta de trabajo robusta, eficiente y con un fuerte apoyo de desarrollo y comunidad tras ella.

4. **Lenguaje de programación**:

La elección del lenguaje de programación influye en la estructura y funcionalidad del proyecto. Se optará por un lenguaje que potencie la eficacia del desarrollo y el rendimiento de la aplicación. Dado que se seguirán las nuevas prácticas para el desarrollo de aplicaciones Android, se ha seleccionado el lenguaje Kotlin [7] debido a su gran potencia, soporte y comunidad, lo que implica una excelente ventaja a la hora del desarrollo.

5. **Procesador de textos para el documento**:

El procesador de textos empleado en la documentación es esencial para una presentación clara y profesional.

Se han considerado numerosas opciones que permitan aportar estas características y, finalmente, se ha optado por *Microsoft Word* [25] debido a su potencia y que, pese a ser una herramienta de uso bajo una licencia de pago, la universidad proporciona una licencia oficial a todos sus estudiantes.

6. **Control de versiones con *Git***:

El control de versiones con Git [19] y GitHub [20] es crucial para el desarrollo colaborativo y la gestión del código fuente. Se implementará para asegurar un seguimiento y manejo efectivo de los cambios en el proyecto.

Capítulo
6. Recursos

En este capítulo, se detallan los recursos esenciales que respaldarán la ejecución exitosa del proyecto. Tanto los recursos humanos como los materiales desempeñarán un papel fundamental en la materialización de la propuesta, asegurando la consecución de los objetivos planteados con eficacia y calidad.

6.1. Recursos humanos

- **Autor**: Julen Pérez Hernández. Alumno de Computación en el Grado en Ingeniería Informática de la Universidad de Córdoba.

- **Director**: Juan Carlos Fernández Caballero. Profesor Titular de la Universidad de Córdoba del Dpto. de Informática y Análisis Numérico y miembro investigador del grupo AYRNA.

6.2. Recursos materiales

A continuación, se detallan los recursos materiales esenciales que se requerirán para llevar a cabo el desarrollo de la aplicación propuesta.

6.2.1. Recursos *hardware*

En el caso del proyecto que se presenta, el uso de un ordenador con capacidad suficiente será esencial para la ejecución de las tareas de desarrollo y pruebas de la aplicación, siendo las siguientes las características del ordenador de sobremesa personal a utilizar para el desarrollo:

- **Procesador**: Intel i5-12600, 6 núcleos de alto rendimiento (12 hilos) a 3.30 GHz.

- **Memoria RAM**: 16 GB DDR4 a 3600 MHz.

- **Tarjeta Gráfica**: Nvidia Gforce RTX 3060 12 GB.
- **Almacenamiento**: 1 TB SSD M.2 NVMe a 3500 MB/s + 1 TB HDD.

Además, se hará uso de un dispositivo móvil con sistema operativo Android, ya sea un teléfono o una t*ablet*, para realizar algunas de las pruebas correspondientes en la aplicación.

6.2.2. Recursos *software*

- **Sistema Operativo**: Windows 11 [26].
- **Lenguaje de Programación**: Kotlin [7].
- **Librerías y *frameworks***: Se utilizarán todas aquellas librerías que permitan simplificar, facilitar y modularizar el proyecto todo lo posible. Principalmente *Jetpack Compose* [8] como librería principal en el desarrollo de interfaces.
- **Entorno de desarrollo (IDE) y editores**: *Android Studio* [6].
- **Documentación**: Se utilizará la herramienta Microsoft Word [25].

Bloque
II. Análisis del sistema y especificaciones de requisitos

Capítulo
7. Especificación de requisitos

En este capítulo, se establecerán las especificaciones de requisitos que servirán como la piedra angular del proceso de desarrollo. Se detallarán de manera precisa y clara los requisitos funcionales y no funcionales que la aplicación deberá cumplir. Estas especificaciones no solo guiarán la implementación técnica, sino que también asegurarán que la aplicación final satisfaga las expectativas y necesidades identificadas en las etapas anteriores del proyecto.

7.1. Requisitos de información

Se establecerán los criterios y necesidades relacionadas con la información que la aplicación debe manejar o presentar. Estos requisitos abarcan desde la forma en que se introducen los datos, hasta cómo se presentan al usuario final. En esencia, se trata de definir qué tipo de información se requiere para el correcto funcionamiento de la aplicación y cumplir con los objetivos y funcionalidades previamente establecidos.

En el contexto de este proyecto de desarrollo de una aplicación para simular algoritmos de planificación de procesos en dispositivos Android, se incluirán los siguientes:

1. **Datos de entrada para simulación**: El usuario tendrá todos los campos necesarios para cada algoritmo disponibles con el fin de introducir todos los datos correspondientes para la simulación, siendo estos un identificador o los tiempos de llegada, duración, quantum, entrada en evento, salida del evento, etc.

2. **Resultados de simulación**: Se presentarán los resultados obtenidos derivados de la simulación. El usuario podrá observar los tiempos de inicio, finalización, estancia o espera, entre otros.

3. **Configuración de parámetros**: A la hora de introducir los datos correspondientes a cada proceso, el usuario tendrá siempre disponible la opción de agregar a dicho proceso un evento de E/S, especificando el momento en el que se deberá producir cada una de estas acciones. Adicionalmente siempre se le permitirá eliminar por completo la información introducida o únicamente uno de los procesos, permitiendo así poder modificar el listado de elementos en función a sus necesidades. Cada campo únicamente tendrá permitido agregar el tipo de dato que le corresponde, por ejemplo, los campos de tiempos únicamente tienen permitido un total de dos caracteres numéricos, mientras que el campo del identificador de proceso puede albergar tres caracteres alfanuméricos o símbolos.

7.2. Requisitos funcionales

En esta sección, se establecerán los requisitos funcionales que describen las características esenciales y funcionalidades que la aplicación debe poseer. Estos requisitos detallados definen las acciones específicas que la aplicación debe ser capaz de llevar a cabo para cumplir con su propósito principal. Cada requisito se enfoca en la funcionalidad clave que se espera de la aplicación y se especificará de manera precisa para guiar el desarrollo técnico y asegurar que la aplicación logre sus objetivos de manera efectiva.

1. **Selección de algoritmo**: Los usuarios deben poder elegir entre diferentes algoritmos de planificación para simular.

2. **Ingreso de datos**: La aplicación debe permitir a los usuarios ingresar los datos relevantes para la simulación, como tiempos de llegada y duración de procesos, tiempo de entrada y salida en caso de haberlos, quantum, etc.

3. **Gestión de invocaciones de E/S**: La herramienta debe permitir, además de la introducción de los tiempos de inicio y fin del evento de E/S, considerar dicho evento en la simulación y cómo este afecta a la organización de procesos.

4. **Simulación de procesos**: La aplicación debe ejecutar la simulación de acuerdo con el algoritmo seleccionado y los datos ingresados.

5. **Visualización de resultados**: Los resultados de la simulación, como tiempos de inicio y finalización, deben ser presentados de manera clara y comprensible mediante las tablas o diagramas que se consideren necesarias.

6. **Gestión de procesos agregados**: La aplicación debe permitir la adición y eliminación de procesos para considerar distintos escenarios de simulación.

7. **Visualización de Diagramas de Tiempo**: Los diagramas de Gantt o de tiempo de los procesos deben ser generados y presentados de formas claras y sencillas para una correcta visualización.

8. **Visualización de las colas de procesos**: La herramienta debe considerar, en función al algoritmo empleado, las diferentes colas de procesos y cómo estas son modificadas en cada momento a lo largo de la simulación. Con ello, se debe mostrar esta información al usuario permitiendo así su análisis.

9. **Cambio del tema de la herramienta**: Se deberá permitir al usuario alternar entre los temas claro y oscuro presentes en la aplicación, lo que modificará el esquema de colores utilizado por la herramienta.

7.3. Requisitos no funcionales

En esta sección, se establecerán los requisitos no funcionales que definen los parámetros de calidad y características globales que la aplicación debe cumplir. Estos requisitos se centran en aspectos que no son directamente funcionalidades, pero que son esenciales para garantizar la eficiencia, seguridad, usabilidad y satisfacción general del usuario. Cada requisito no funcional detalla una expectativa específica con respecto a la aplicación y se describirá de manera precisa para orientar su diseño y desarrollo.

1. **Usabilidad**:
   - La aplicación debe ser intuitiva y fácil de usar, con una curva de aprendizaje ágil para usuarios con conocimientos básicos de tecnología.
   - Los elementos de la interfaz deben ser de tamaño adecuado para su interacción en pantallas táctiles, garantizando una experiencia cómoda para usuarios con diferentes tamaños de dedos.

2. **Rendimiento**:
   - La aplicación debe ser capaz de manejar la simulación de algoritmos de planificación de procesos con un tiempo de respuesta promedio aceptable.

- La velocidad de actualización de los resultados y gráficos durante la simulación no debe generar demoras perceptibles por parte del usuario.

3. **Compatibilidad**:
    - La aplicación debe ser compatible con una amplia variedad de dispositivos Android, abarcando diferentes versiones de sistema operativo y tamaños de pantalla.

4. **Accesibilidad**:
    - La aplicación debe cumplir con un mínimo de pautas de accesibilidad para garantizar que la gran mayoría de usuarios puedan interactuar y utilizar la aplicación de manera efectiva.

5. **Mantenibilidad**:
    - El código de la aplicación debe estar bien estructurado y documentado, permitiendo a futuros desarrolladores comprender y mantener el proyecto de manera eficiente.

    - La aplicación debe estar diseñada de manera modular, facilitando la incorporación de nuevos algoritmos de planificación en el futuro sin necesidad de reescribir gran parte del código existente.

6. **Rendimiento en Distintos Dispositivos**:
    - La aplicación debe ofrecer un rendimiento consistente en dispositivos de diferentes capacidades y especificaciones, evitando retrasos significativos en dispositivos con recursos limitados.

7. **Disponibilidad**:
    - La aplicación debe estar disponible para su descarga en formato "*.apk*" y desde la *Play Store* de Google.

8. **Actualizaciones y Mantenimiento**:
    - La aplicación debe permitir la actualización de nuevos algoritmos de planificación y mejoras de manera sencilla, sin requerir reinstalación completa por parte del usuario.

Capítulo
8. Análisis funcional

En este capítulo, se llevará a cabo un análisis funcional utilizando UML [21] (Lenguaje de Modelado Unificado) para definir los diferentes aspectos de la interacción del sistema. Se identificarán los usuarios del sistema, se describirá el caso de uso de contexto que representa las interacciones generales y se detallarán los casos de uso específicos que describen las funcionalidades específicas que la aplicación proporcionará. Esta metodología de modelado permitirá una comprensión más completa de cómo los diferentes elementos del sistema interactúan entre sí, sentando las bases para un diseño y desarrollo coherente y efectivo de la aplicación.

A continuación, se presenta la Tabla 1, diseñada para servir como referencia en la representación de los casos de uso en el marco de este proyecto. Asimismo, se ofrece una descripción detallada de los campos que componen esta tabla y cómo se aplicarán en el análisis funcional del sistema.

Campo	Descripción
Caso de uso	Breve descripción del caso de uso.
Identificador	Identificación única del caso de uso.
Descripción	Descripción informal de los objetivos del caso de uso.
Actores	Actores involucrados en la ejecución del caso de uso.
Precondiciones	Condiciones necesarias para que el caso de uso pueda ocurrir.
Flujo principal	Pasos secuenciales que constituyen el flujo normal.
Flujos alternativos	Pasos alternativos que pueden surgir.
Postcondiciones	Estado en que se encuentra el sistema tras el caso de uso.

Tabla 1. Plantilla de Caso de Uso

Este apartado seguirá una estructura organizada. Se comenzará por la presentación y definición de los actores que interactúan con la aplicación. A continuación, se expondrá el caso de uso de contexto, el cual abarca una visión general del sistema junto con los actores involucrados.

Finalmente, se procederá a describir un refinamiento de este caso de uso para cada uno de los módulos funcionales que componen la aplicación. Este enfoque estructurado facilitará una comprensión clara de las interacciones del sistema y permitirá su análisis detallado.

8.1. Identificación de los usuarios objetivo de la app

Los usuarios a los que se dirige la aplicación son estudiantes del Grado de Ingeniería Informática en la Universidad de Córdoba que cursan la asignatura de Sistemas Operativos. Sin embargo, la aplicación estará accesible para cualquier usuario en general, incluyendo tanto estudiantes como docentes.

8.2. Casos de uso de contexto

A continuación, se expone el caso de uso de contexto, un caso de uso con la funcionalidad más genérica, correspondiente con el inicio de la aplicación, del cual derivan el resto de los casos de uso. Su especificación se muestra en la Tabla 2.

Campo	Descripción
Caso de uso	Caso de uso de contexto.
Identificador	CUC-00.
Descripción	Acceso a la aplicación. Abre la pantalla de inicio.
Actores	Usuario.
Precondiciones	La aplicación se encuentra instalada en el dispositivo.
Flujo principal	El usuario abre la aplicación.
Flujos alternativos	-
Postcondiciones	La pantalla de inicio ha sido correctamente inicializada.

Tabla 2. CUC-00. Caso de uso de contexto.

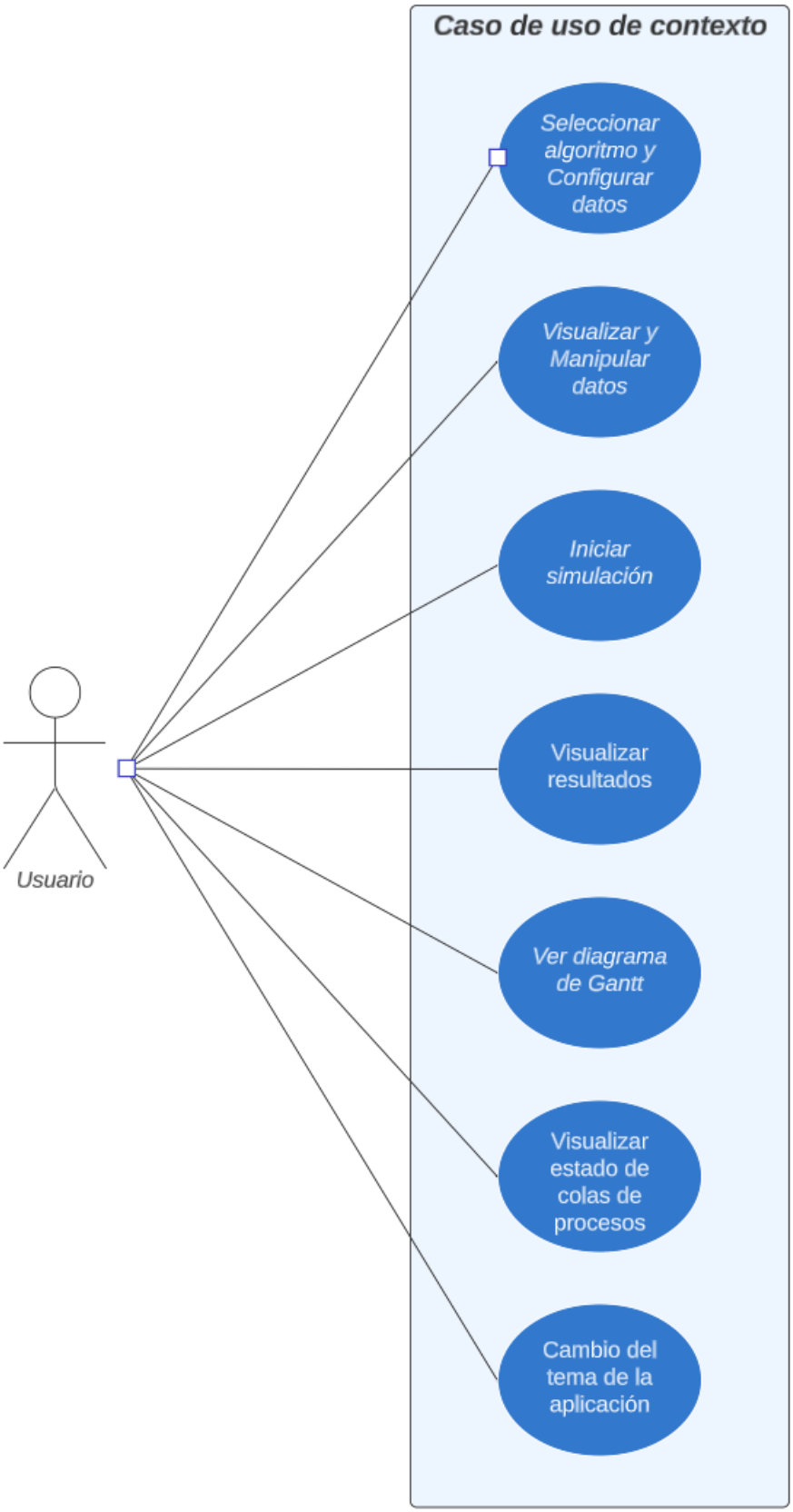

Figura 11. CUC-00.

8.3. Casos de uso

En esta sección, se abordarán los casos de uso específicos que definen las interacciones particulares entre los usuarios y la aplicación. Estos casos de uso detallan las funcionalidades y operaciones específicas que los usuarios pueden llevar a cabo dentro de la aplicación. Cada caso de uso ofrece una perspectiva detallada de cómo los actores interactúan con el sistema para lograr objetivos específicos.

A continuación, se presentarán los casos de uso individuales que componen esta etapa del análisis funcional, brindando una visión detallada de la operación y los resultados esperados en diferentes escenarios de uso.

8.3.1. Seleccionar algoritmo y configurar datos

Campo	Descripción
Caso de uso	Seleccionar algoritmo y configurar datos.
Identificador	CU-01.
Descripción	El usuario inicia la aplicación y accede a la pantalla de inicio. Aquí, puede seleccionar el algoritmo deseado, introducir los datos requeridos para su ejecución y decidir si se desea agregar un evento de E/S.
Actores	Usuario.
Precondiciones	La aplicación se ha iniciado correctamente.
Flujo principal	1. El usuario elige el algoritmo de planificación. 2. El usuario introduce los datos en los campos requeridos. 3. Si es necesario, el usuario agrega un evento de E/S. 4. El usuario agrega el proceso con los datos proporcionados.
Flujos alternativos	El usuario puede cambiar de opinión y seleccionar otro algoritmo o modificar los datos a su gusto.
Postcondiciones	Los datos del proceso se almacenan, se muestran en una tabla y se limpian los campos para poder agregar más procesos.

Tabla 3. CU-01. Selección del algoritmo e Introducción de datos.

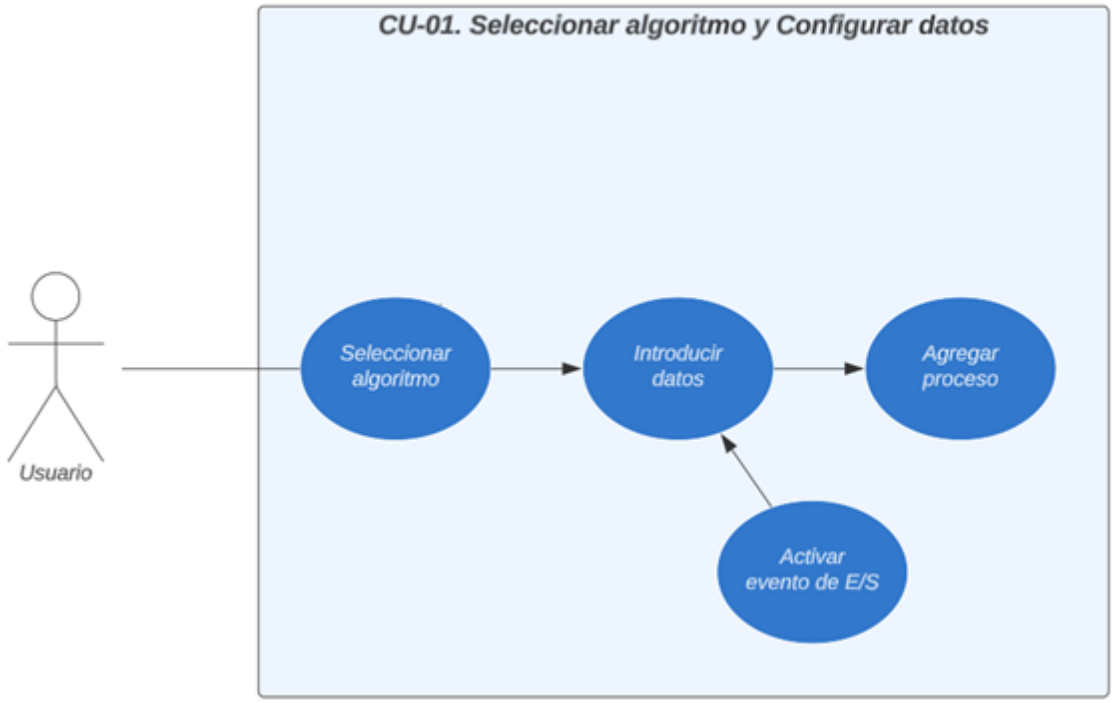

Figura 12. CU-01.

8.3.2. Visualizar y modificar datos

Campo	Descripción
Caso de uso	Visualizar y modificar los datos.
Identificador	CU-02.
Descripción	El usuario puede ver los datos introducidos en forma de tabla y realizar acciones como eliminar procesos individuales o todos los procesos agregados.
Actores	Usuario.
Precondiciones	Se han introducido al menos un proceso en la pantalla de inicio.
Flujo principal	1. El usuario visualiza la tabla de datos. 2. El usuario selecciona un proceso para eliminarlo. a. El usuario confirma la eliminación. b. El usuario cancela la eliminación. 3. El usuario elimina todos los procesos agregados. a. El usuario confirma la eliminación. b. El usuario cancela la eliminación.
Flujos alternativos	El usuario puede optar por no eliminar ningún proceso.
Postcondiciones	Los procesos seleccionados se eliminan y la tabla de datos se actualiza.

Tabla 4. CU-02. Visualizar y Modificar los datos.

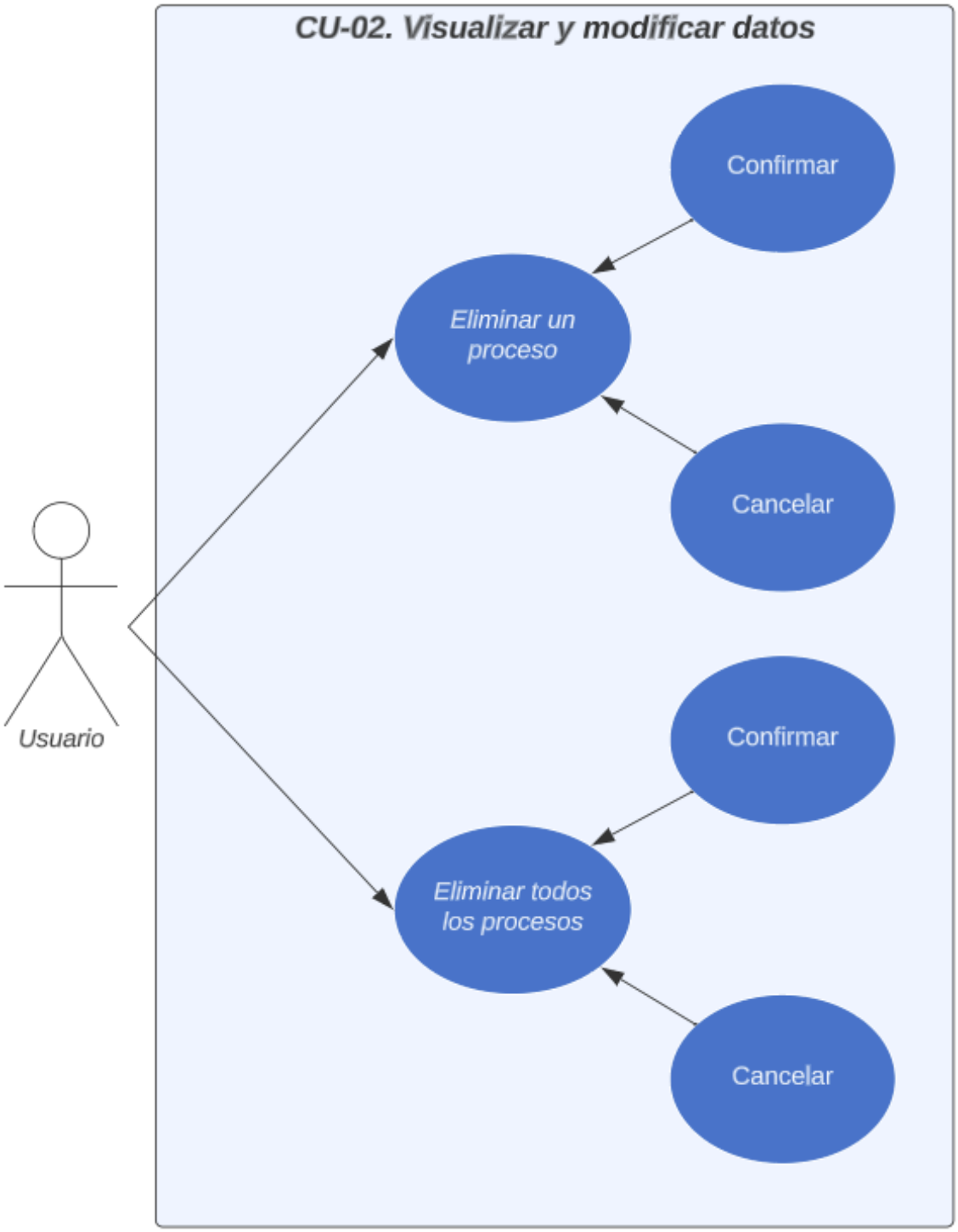

Figura 13. CU-02.

8.3.3. Iniciar simulación

Campo	Descripción
Caso de uso	Iniciar simulación.
Identificador	CU-03.
Descripción	El usuario avanza a la siguiente etapa de la aplicación.
Actores	Usuario.
Precondiciones	Se ha introducido al menos un proceso y se ha avanzado a la siguiente página.
Flujo principal	**1.** El usuario ha introducido al menos un proceso con el que trabajar. **2.** Se interactúa con el elemento correspondiente.
Flujos alternativos	-
Postcondiciones	La simulación se inicia y se avanza a la página de resultados.

Tabla 5. CU-03. Iniciar simulación.

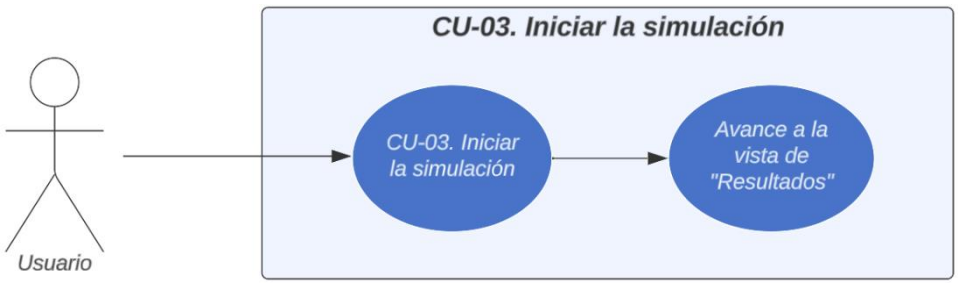

Figura 14. CU-03.

8.3.4. Visualizar resultados

Campo	Descripción
Caso de uso	Visualizar resultados.
Identificador	CU-04.
Descripción	El usuario accede a la página de resultados donde se muestra una tabla con los tiempos obtenidos por la simulación del algoritmo seleccionado.
Actores	Usuario.
Precondiciones	La simulación ha sido iniciada y se ha avanzado a la página de resultados.
Flujo principal	1. El usuario visualiza la tabla de resultados con los tiempos obtenidos por la simulación.
Flujos alternativos	-
Postcondiciones	El usuario obtiene una visión detallada de los resultados de la simulación en la tabla.

Tabla 6. CU-04. Visualizar los resultados.

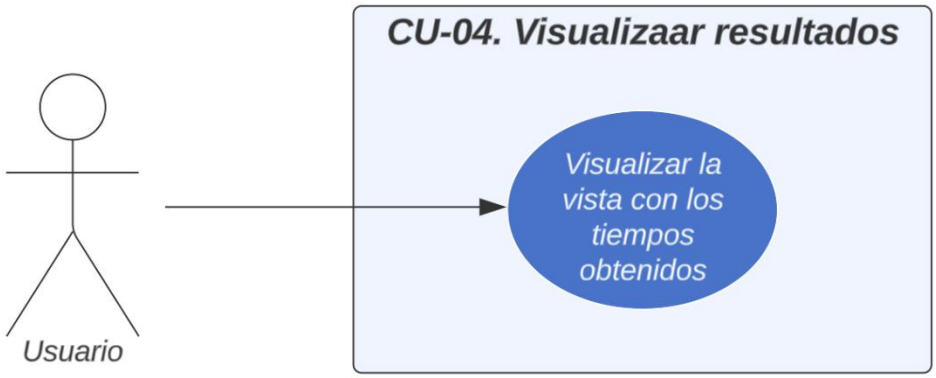

Figura 15. CU-04.

8.3.5. Ver diagrama de Gantt

Campo	Descripción
Caso de uso	Ver diagrama de Gantt.
Identificador	CU-05.
Descripción	El usuario accede a la página que muestra una tabla en forma de diagrama de Gantt que representa los tiempos obtenidos y los estados en los que se encuentra cada proceso en momentos determinados de la simulación.
Actores	Usuario.
Precondiciones	La simulación ha sido iniciada y se ha avanzado a la página del diagrama de Gantt.
Flujo principal	1. El usuario visualiza el diagrama de Gantt con la representación gráfica de los tiempos y estados de los procesos.
Flujos alternativos	-
Postcondiciones	El usuario obtiene una representación visual clara de la evolución de los procesos a lo largo del tiempo.

Tabla 7. CU-05. Visualizar el diagrama de Gantt.

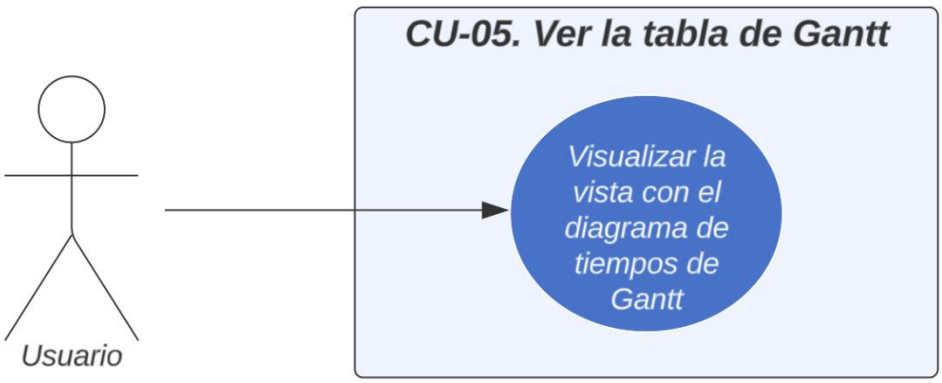

Figura 16. CU-05.

8.3.6. Visualizar estado de colas de procesos

Campo	Descripción
Caso de uso	Visualizar estado de colas de procesos.
Identificador	CU-06.
Descripción	El usuario accede a la página que muestra en tablas el estado de las colas de procesos en diferentes momentos de la simulación.
Actores	Usuario.
Precondiciones	La simulación ha sido iniciada y se ha avanzado a la página del estado de las colas de procesos.
Flujo principal	1. El usuario visualiza las tablas que representan el estado de las colas de procesos en distintos momentos de la simulación.
Flujos alternativos	-
Postcondiciones	El usuario obtiene una visión detallada de cómo los procesos se agrupan y evolucionan en las colas a lo largo de la simulación.

Tabla 8. CU-06. Visualiza las colas de estados de los procesos.

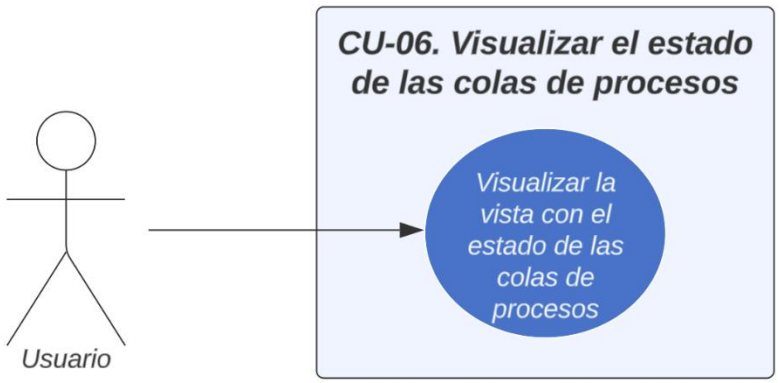

Figura 17. CU-06.

8.3.7. Cambiar el tema de la aplicación

Campo	Descripción
Caso de uso	Cambiar el tema de la aplicación.
Identificador	CU-07.
Descripción	El usuario cambia el tema de la aplicación entre oscuro y claro utilizando el elemento correspondiente en la vista.
Actores	Usuario.
Precondiciones	El usuario se encuentra en cualquier vista de la aplicación.
Flujo principal	1. El usuario pulsa el elemento de cambio de tema. 2. La aplicación alterna su apariencia entre el tema actual y el opuesto.
Flujos alternativos	-
Postcondiciones	La aplicación adopta el tema oscuro o claro según la preferencia del usuario, proporcionando una experiencia visual adaptada a su elección.

Tabla 9. CU-07. Cambio de tema de la aplicación.

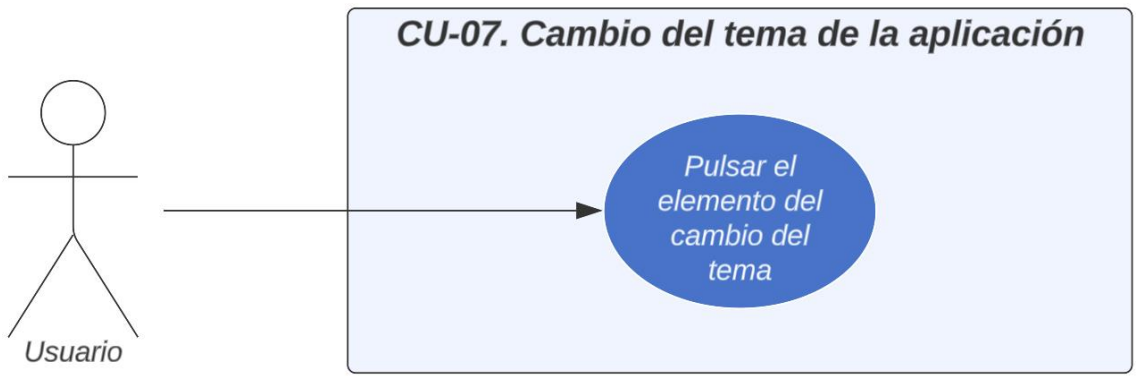

Figura 18. CU-07.

Bloque
III. Diseño del sistema

Capítulo
9. Arquitectura del sistema

El propósito de este capítulo es definir de forma detallada la estructura planteada para este sistema, de forma que se pueda desarrollar la funcionalidad que se especificó en el anterior análisis funcional.

La arquitectura de la aplicación se ha diseñado siguiendo un enfoque modular, lo que facilita la escalabilidad y permite la adición de nuevas funcionalidades en el futuro de manera eficiente. En este proyecto, el desarrollo se ha centrado en el lado cliente, ya que no se requiere un servidor para cumplir con los objetivos de la aplicación.

El lado cliente es la parte visible y con la que interactúa el usuario de la aplicación. Está compuesto por varios tipos de ficheros que trabajan en conjunto para proporcionar la funcionalidad y la interfaz gráfica de la aplicación.

9.1. Organización de paquetes

En esta sección se abordará la estructura seguida a la hora de la creación de los diferentes archivos del proyecto y de la organización establecida para lograr dicha modularización de todos ellos en sus respectivos paquetes. Para este proyecto, es posible dividir esta organización en dos secciones diferenciadas con el fin de destacar los aspectos más relevantes al respecto.

La primera sección se centra en el paquete principal del código, que alberga todas las funciones, clases e interfaces esenciales para el funcionamiento de la aplicación. Aquí, se encontrará información detallada sobre la estructura del código, la lógica detrás de las funciones clave y cómo se relacionan entre sí para lograr el flujo de trabajo de la aplicación.

La segunda sección se dedica al paquete de recursos, que almacena todos los elementos visuales, como íconos, imágenes y textos utilizados en la aplicación. Se considerará cómo se gestionan estos recursos y cómo se integran con el código principal para crear una experiencia de usuario coherente y atractiva.

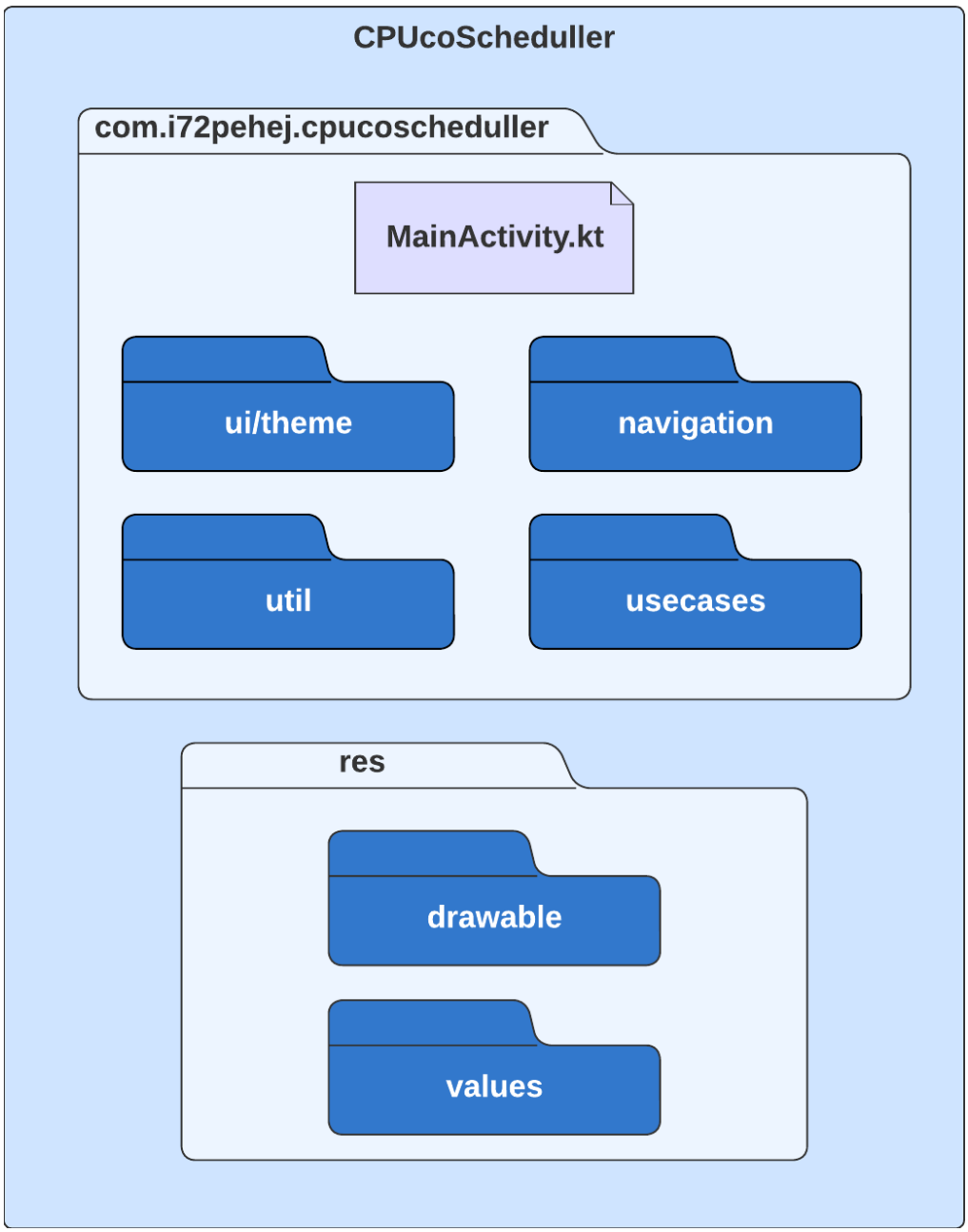

Figura 19. Organización de paquetes.

En la figura 19 se observa la distribución mencionada que, a continuación, se desarrollará más detalladamente para concretar los aspectos más relevantes de cada uno de los paquetes que conforman el proyecto:

- ***com.i72pehej.cpucoscheduller***: Representa el paquete principal que almacena todas las clases y ficheros correspondientes a la funcionalidad de la herramienta. Contiene los siguientes subdirectorios:

    - ***navigation***: Aquí residen todos los archivos dedicados a gestionar la navegación entre las diferentes páginas de la aplicación. Esta estructura modulariza la lógica de la navegación, lo que facilita la incorporación de nuevas vistas y la gestión de las transiciones.

    - ***ui/theme***: En esta carpeta se encuentran los ficheros de configuración que definen los colores generales, formas predefinidas y otros elementos visuales relacionados con los temas de la aplicación. Esto permite un control coherente y sencillo de la apariencia de la aplicación.

    - ***usecases***: Cada fichero contenido en esta carpeta se corresponde con un caso de uso específico de la aplicación. Esta modularización permite que cada funcionalidad de la aplicación esté contenida en archivos individuales, lo que facilita la comprensión y el mantenimiento del código.

    - ***util***: En esta carpeta se asignan todos los ficheros correspondientes a la gestión de todas las utilidades comunes de la aplicación. Esta carpeta se subdivide en dos subcarpetas: **"classes"** y **"extensions"**. En la primera, se encuentran los ficheros destinados a definir nuevas clases que resultan útiles en la aplicación. Mientras que, en la segunda, se albergan funcionalidades generales que se aplican en toda la aplicación, mejorando la experiencia del usuario o modularizando tareas comunes. También se encuentra aquí un fichero que contiene valores accesibles desde cualquier punto de la aplicación, lo que facilita la compartición de estructuras y datos entre diferentes funciones.

    Además, en la raíz del paquete principal, se encuentra el fichero **"MainActivity.kt"**, que actúa como el punto de entrada principal de la aplicación, iniciando la ejecución del resto de elementos.

- ***res***: Almacena todos los archivos relacionados con los recursos visuales y de contenido, como imágenes, iconos, textos y valores necesarios para el funcionamiento correcto de la aplicación. Se divide principalmente en dos subdirectorios:

    - ***drawable*:** En esta subcarpeta, se encuentran todas las imágenes e iconos que la aplicación utiliza en su interfaz. Estos elementos visuales son fundamentales para la representación gráfica de la información y la navegación, aportando un aspecto atractivo y coherente a la aplicación, así como los diferentes iconos que identifican a la aplicación.

    - ***values***: En esta subcarpeta, se albergan ficheros de valores esenciales para el funcionamiento de la aplicación. El fichero destacado es el que contiene las cadenas de texto utilizadas en toda la aplicación. Estas cadenas son vitales para mostrar información al usuario de manera comprensible y, al modularizarlas, se facilita su mantenimiento y su posible traducción a otros idiomas.

Capítulo
10. Diseño de la interfaz

El diseño de la interfaz de usuario es un componente fundamental en el desarrollo de cualquier aplicación móvil exitosa, ya que desempeña un papel crucial en la experiencia del usuario. En este apartado, se detallará minuciosamente la estructura y el aspecto de la interfaz de la aplicación, centrándose en cada uno de los módulos que la componen. Cada módulo ha sido cuidadosamente diseñado con el propósito de ofrecer una experiencia de usuario intuitiva, atractiva y eficaz.

A lo largo de este apartado, se destacarán los subapartados que describen de manera exhaustiva cada uno de los módulos de la interfaz, resaltando sus características distintivas y su integración en la aplicación. Cada módulo ha sido concebido con especial énfasis en la usabilidad, la accesibilidad y la estética, con el objetivo primordial de proporcionar a los usuarios una experiencia fluida y satisfactoria al interactuar con la aplicación.

Cabe mencionar que el esquema de colores empleado en el diseño de la interfaz se ajusta a las directrices específicas proporcionadas por la universidad, garantizando así una coherencia visual con la identidad institucional y contribuyendo a consolidar la imagen de la aplicación como una herramienta de calidad y confiabilidad.

En el proceso de definición y desarrollo de la interfaz de usuario para la aplicación, se han considerado varias metodologías, cada una con sus propias ventajas y desafíos. Estas metodologías incluyen enfoques "*Top-Down*" y "*Bottom-Up*", así como el enfoque "Iterativo e Incremental". A continuación, se discutirá cómo se ha abordado la definición de la interfaz de usuario en este proyecto y por qué se ha optado por un enfoque iterativo e incremental.

- **Metodología *Top-Down* (Arriba-Abajo)**: Inicialmente, se consideró la posibilidad de adoptar un enfoque "*Top-Down*" para definir la estructura general de la interfaz de usuario. Esto habría implicado comenzar con una visión panorámica de la aplicación y luego desglosarla en componentes más específicos. Si bien este enfoque es valioso para establecer una estructura coherente, se consideró que no se adaptaba completamente a la naturaleza modular de la aplicación y a la necesidad de ajustar los detalles de manera flexible a medida que se avanzaba.

- **Metodología *Bottom-Up* (Abajo-Arriba)**: También se consideró el enfoque "*Bottom-Up*", que se centra en la construcción de componentes individuales y su posterior integración en la interfaz completa. Si bien este enfoque es efectivo para abordar componentes aislados y funcionales, se encontró que podría resultar en desafíos en la integración y coherencia general de la interfaz, especialmente cuando se trata de un proyecto complejo con múltiples módulos interdependientes.

- **Metodología iterativa e incremental**: Dado el enfoque modular del proyecto y la necesidad de una interfaz de usuario bien definida y cohesionada, se ha optado por seguir una metodología "Iterativa e incremental". Este enfoque permite dividir el desarrollo en pequeñas iteraciones o fases, cada una de las cuales produce incrementos funcionales y mejoras en la interfaz. Cada iteración se enfoca en un módulo específico de la aplicación, desarrollando y refinando su interfaz y funcionalidad de manera aislada. Al seguir un enfoque iterativo, es posible asegurar que cada parte de la interfaz funcione de manera eficaz antes de avanzar a la siguiente.

  Este enfoque es particularmente adecuado para proyectos de desarrollo de *software* como este, donde la flexibilidad y la capacidad de adaptación son esenciales. También se alinea bien con la naturaleza educativa de esta aplicación, ya que permite incorporar características y mejoras en función de la retroalimentación y las necesidades cambiantes del entorno educativo.

En resumen, se ha adoptado una metodología "Iterativa e incremental" para el diseño de la interfaz de usuario de la aplicación, ya que se considera que es la más adecuada para el proyecto actual y el estilo de trabajo seguido. Este enfoque permite abordar de manera efectiva los desafíos de diseño y desarrollo, asegurando que la interfaz sea coherente, funcional y adaptable a medida que se avanza en el proceso de desarrollo.

Bloque
IV. Pruebas

Capítulo 11. Pruebas

La fase de pruebas desempeña un papel crucial en el ciclo de desarrollo de este proyecto, donde se someten los componentes de la aplicación a diversas ejecuciones y evaluaciones exhaustivas para verificar su cumplimiento de los requisitos establecidos. A diferencia de una fase aislada, las pruebas se integran de manera continua a lo largo del desarrollo, trabajando en paralelo con la codificación de la aplicación. Sin embargo, al concluir la etapa de desarrollo, se dedica un tiempo específico para llevar a cabo pruebas rigurosas tanto a nivel individual de los módulos como a nivel integral del sistema.

Para asegurar la fiabilidad del sistema, se inicia con pruebas unitarias que evalúan minuciosamente cada módulo de funcionalidad, garantizando que cada función se comporte conforme a las expectativas. Estas pruebas unitarias son un paso fundamental antes de avanzar hacia las pruebas del conjunto de módulos que componen la aplicación.

A continuación, se explorarán en detalle los principios y objetivos de las pruebas a las que se someterá el *software*, así como algunas de las pruebas específicas que se han implementado en este proyecto.

11.1. Estrategia de pruebas

Diseñar una estrategia de pruebas sólida es un elemento crucial para garantizar la calidad y confiabilidad de la aplicación, evitando posibles errores y asegurando que sea completamente funcional, segura y robusta. La estrategia de pruebas es una parte esencial de cualquier proyecto de desarrollo de *software*, y en el contexto de este proyecto de TFG, se centra en asegurar que la aplicación cumple con los requisitos y expectativas establecidos.

La estrategia de pruebas se compone de varios tipos de casos de prueba, cada uno con un propósito específico:

1. **Pruebas Unitarias**: Estas pruebas se centran en verificar el funcionamiento individual de cada componente o módulo de la aplicación. Para lograr esto, se dividirán en dos categorías:

    a. **Pruebas de Caja Negra**: Evaluarán la funcionalidad de los componentes sin conocer su implementación interna. Esto garantiza que los resultados sean coherentes con los requisitos funcionales y las expectativas del usuario.

    b. **Pruebas de Caja Blanca**: Examinarán la lógica interna de los componentes, asegurando que los caminos de ejecución, las estructuras de control y las variables cumplan con las especificaciones técnicas y contribuyan al correcto funcionamiento del sistema.

2. **Pruebas de Integración**: Estas pruebas se centran en evaluar la interacción entre diferentes componentes o módulos de la aplicación. El objetivo es garantizar que se comuniquen y colaboren de manera efectiva para lograr la funcionalidad completa de la aplicación.

3. **Pruebas de Sistema**: Se realizan para evaluar el comportamiento y el rendimiento del sistema en su conjunto. Esto incluye probar las funcionalidades globales, la navegación entre pantallas y la respuesta del sistema a situaciones límite. Se verificará que la aplicación se comporte correctamente en diferentes dispositivos Android.

4. **Pruebas de Usabilidad**: Las pruebas de usabilidad implican que los usuarios reales prueben la aplicación y brinden comentarios sobre su facilidad de uso. Esto puede ser valioso para evaluar si la interfaz de usuario es intuitiva y amigable para estudiantes y docentes.

5. **Pruebas de Aceptación**: Estas pruebas implican la validación de la aplicación por parte del usuario final o del docente. Se asegura de que la aplicación cumple con los requisitos iniciales y es satisfactoria para su uso en el contexto educativo.

En este documento, se detallarán los casos de prueba más relevantes de cada una de estas categorías. Dado que la aplicación puede contener múltiples funciones similares, se mostrarán aquellos casos de prueba que mejor ejemplifican el enfoque de la estrategia de pruebas. En caso de que los resultados de las pruebas no cumplan con los estándares mínimos de calidad, se procederá a modificar y optimizar el código correspondiente para garantizar un rendimiento óptimo y una experiencia de usuario de alta calidad.

11.2. Pruebas de caja blanca

Dentro del proceso de desarrollo de *software*, es fundamental garantizar la confiabilidad y la robustez de una aplicación. Las pruebas de caja blanca, también conocidas como pruebas estructurales, son un enfoque esencial para lograr este objetivo. En estas pruebas, se examina y valida la estructura interna del código fuente de la aplicación. En esencia, se trata de abrir la "caja blanca" del *software* y revelar sus secretos internos.

Estas pruebas se basan en el conocimiento de cómo funciona el código, lo que incluye la lógica interna, las rutas de ejecución, las condiciones lógicas y los bucles. Al comprender estas características, los probadores pueden diseñar casos de prueba que evalúen exhaustivamente el código y sus posibles caminos de ejecución. Esto permite identificar posibles errores de programación, garantizar una mayor cobertura del código y mejorar la calidad general del *software*.

En este subapartado, se explorarán en detalle las pruebas de caja blanca realizadas como parte de la estrategia de pruebas de este proyecto. Se describirán los enfoques utilizados, los criterios de prueba, las herramientas empleadas y, lo que es más importante, los resultados obtenidos. Estas pruebas ofrecen una visión profunda del funcionamiento interno de la aplicación, brindando la confianza necesaria en su integridad y capacidad de respuesta.

A continuación, se presentarán las pruebas de caja blanca realizadas como parte de esta estrategia de pruebas y se discutirá cómo estas pruebas han contribuido a garantizar la calidad y la fiabilidad de la aplicación.

11.2.1. CU-01. Seleccionar Algoritmo y Configurar Datos

A continuación, se presenta la tabla 10, que resume las pruebas realizadas para el caso de uso. Cada prueba se ha diseñado para evaluar diferentes aspectos de la funcionalidad y el rendimiento, y se han definido los resultados esperados para cada una.

A través de esta tabla, se documentan los resultados obtenidos y cualquier acción correctiva tomada en caso de que el resultado no cumpla con las expectativas. Esta información proporciona una visión general de la calidad y el estado de la aplicación después de las pruebas.

	Descripción
1	El menú desplegable selecciona correctamente el algoritmo deseado.
2	El formulario actualiza correctamente los campos en función al algoritmo.
3	Se almacenan correctamente la información de los campos.
4	Se consideran adecuadamente los campos correspondientes al evento de E/S (cuando se selecciona).
5	Se agrega correctamente un nuevo proceso al pulsar el botón “+”.

Tabla 10. Pruebas de caja blanca para CU-01.

Las pruebas 1, 2 y 5 arrojaron resultados satisfactorios tras realizar ajustes en el enfoque seguido.

Sin embargo, las pruebas restantes (pruebas 3 y 4) inicialmente presentaron errores de ejecución, los cuales se subsanaron en iteraciones posteriores hasta alcanzar los resultados deseados, detallándose a continuación.

Con relación a la prueba 3, los fallos detectados no eran críticos, sino que generaban resultados no deseados. Precisamente, se observó que la información en los campos se almacenaba correctamente, pero era imperativo restringir la casuística de caracteres válidos para evitar la inserción de datos no permitidos. Esta limitación se logró mediante el empleo de expresiones regulares, que controlaron tanto la cantidad como el tipo de caracteres permitidos en cada campo. Adicionalmente, se adaptó el tipo de teclado (numérico o estándar) en función del campo en el que se encontraba el usuario, y se implementaron notificaciones de error que impiden la ejecución de la simulación en caso de campos vacíos o valores no permitidos.

En cuanto a la prueba 4, se identificó un error por el cual los campos del evento de E/S se consideraban activos en todo momento, independientemente de si se seleccionaban o no. La corrección de este fallo incluyó la incorporación de una variable de confirmación de selección al agregar el proceso correspondiente.

11.2.2. CU-02. Visualizar y Modificar Datos

A continuación, se presenta la tabla 11 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Los procesos que se agregan se añaden correctamente a la tabla.
2	Se visualiza correctamente la información cuando se tiene un evento de E/S.
3	Se elimina correctamente el proceso seleccionado.
4	Se eliminan correctamente todos los procesos.

Tabla 11. Pruebas de caja blanca para CU-02.

Las pruebas 1, 2 y 4 han arrojado resultados esperados sin requerir modificaciones significativas.

No obstante, la prueba número 3 presentó un problema en la eliminación adecuada del proceso deseado. La solución adoptada implicó la necesidad de introducir una variable única como identificador para cada proceso, permitiendo su selección y eliminación precisa del listado.

11.2.3. CU-03. Iniciar Simulación.

A continuación, se presenta la tabla 12 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	El botón “Siguiente” ejecuta la llamada al algoritmo seleccionado.
2	El botón “Siguiente” avanza automáticamente a la ventana de Resultados.

Tabla 12. Pruebas de caja blanca para CU-03.

En este caso, todas las pruebas han sido satisfactorias y no se han encontrado errores de ejecución críticos.

11.2.4. CU-04. Visualizar Resultados.

A continuación, se presenta la tabla 13 con las pruebas realizadas en la aplicación, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	La tabla de resultados muestra correctamente los valores obtenidos.
2	La tabla de resultados muestra adecuadamente los resultados al existir más procesos de los que entran en pantalla.

Tabla 13. Pruebas de caja blanca para CU-04.

La prueba número 1 ha obtenido el resultado esperado.

En contraste, la prueba número 2 no desplegaba los procesos correspondientes cuando la tabla superaba el tamaño de la pantalla del dispositivo. Para abordar esta cuestión, se implementó una solución consistente en emplear un nuevo formato de tabla con funcionalidad de desplazamiento vertical. Con esta modificación, el contenido que excede la capacidad de visualización de la pantalla no se muestra hasta que se encuentra dentro de la pantalla visible.

11.2.5. CU-05. Ver Diagrama de Gantt

A continuación, se presenta la tabla 14 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Se visualiza correctamente la leyenda en la parte superior.
2	Los símbolos se colocan correctamente en las celdas correspondientes.

Tabla 14. Pruebas de caja blanca para CU-05.

Todas las pruebas realizadas han arrojado resultados satisfactorios, y no se han identificado errores de ejecución que impliquen un carácter crítico o que puedan comprometer el funcionamiento adecuado de la aplicación.

11.2.6. CU-06. Visualizar Estados de Colas de Procesos

A continuación, se presenta la tabla 15 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	La tabla correspondiente a la cola de procesos se visualiza correctamente.
2	Las colas de procesos se muestran adecuadamente en cada momento de la ejecución.

Tabla 15. Pruebas de caja blanca para CU-06.

En este caso, es importante destacar que todas las pruebas realizadas han arrojado resultados satisfactorios, y no se han identificado errores de ejecución que revistan carácter crítico o que puedan comprometer el funcionamiento adecuado de la aplicación.

11.2.7. CU-07. Cambiar el Tema de la Aplicación

A continuación, se presenta la tabla 16 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	El menú de ajustes despliega correctamente el *switch* del tema de la app.
2	La animación funciona de manera óptima.
3	Al pulsar el botón, se alterna entre los temas “Claro” y “Oscuro”.

Tabla 16. Pruebas de caja blanca para CU-07.

Las pruebas número 1 y 3 han arrojado resultados adecuados.

En el caso de la prueba número 2, se identificó un problema en la animación de transición entre los temas. Tras la primera pulsación, la animación quedaba bloqueada en dicho estado. Para solventar esta incidencia, se ha implementado una funcionalidad de retorno al estado inicial en la animación cuando el botón es nuevamente pulsado.

11.3. Pruebas de caja negra

A continuación, se presenta el apartado de pruebas de caja negra, una metodología de evaluación que se centra en verificar la funcionalidad de un sistema sin necesidad de conocer su estructura interna o su código fuente. Este enfoque se basa en observar el comportamiento externo de la aplicación y verificar si se cumplen los requisitos especificados previamente.

A lo largo de esta sección, se detallarán las pruebas de caja negra realizadas en el proyecto, evaluando su efectividad y sus resultados.

11.3.1. CU-01. Seleccionar Algoritmo y Configurar Datos

A continuación, se presenta la tabla 17 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Asegurar que los botones de navegación en la pantalla de inicio, como los marcadores de página funcionen correctamente y permitan acceder a las páginas correspondientes.
2	Confirmar que el selector desplegable de selección de algoritmo muestra todas las opciones disponibles y permite elegir un algoritmo sin problemas.
3	Verificar que los campos de entrada de datos acepten datos válidos y rechacen datos inválidos, como caracteres no numéricos en campos numéricos.
4	Probar la funcionalidad de agregar un proceso con datos válidos.
5	Verificar que se muestran notificaciones de error claras cuando el usuario intenta realizar acciones incorrectas, como agregar un proceso sin completar todos los campos obligatorios.
6	Comprobar que el desplazamiento lateral desde la pantalla de inicio a otras páginas se realice sin problemas y que todas las páginas sean accesibles.
7	Comprobar que el botón de navegación de regreso notifica al usuario la salida de la aplicación y la necesidad de realizar una nueva pulsación para confirmar.

Tabla 17. Pruebas de caja negra para CU-01.

El caso de prueba número 5 experimentó un error debido a que no se realizaba una verificación adecuada de todos los campos. Esto ocasionaba que se pudieran agregar procesos incluso cuando ciertos campos estaban vacíos o contenían valores no válidos.

La solución a este problema se encontró mediante la adición de una capa adicional de control y seguridad. Esta comprobación extra ahora se aplica a todos los campos, complementando las propiedades de validación existentes para garantizar un nivel más alto de integridad y control en la aplicación.

11.3.2. CU-02. Visualizar y Modificar Datos

A continuación, se presenta la tabla 18 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Verificar que se muestren correctamente todos los datos previamente ingresados.
2	Comprobar que los procesos sean eliminables y que, al realizar cambios en ellos, los datos se actualicen de manera precisa y reflejen las modificaciones realizadas.
3	Tras realizar cambios, confirmar que los datos se muestren actualizados en la tabla.

Tabla 18. Pruebas de caja negra para CU-02.

Todas las pruebas de este módulo específico han concluido con éxito, sin necesidad de correcciones relevantes. Esto respalda la calidad y el cumplimiento de los requisitos de este componente del proyecto.

11.3.3. CU-03. Iniciar Simulación.

A continuación, se presenta la tabla 19 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Verificar que, al pulsar el botón "Siguiente", el proceso de simulación se inicie correctamente y que los procesos se ejecuten según el algoritmo seleccionado.
2	Antes de iniciar la simulación, asegurarse de que se realice una validación adecuada de los datos ingresados por el usuario en la pantalla principal.
3	Evaluar cómo se manejan situaciones límite, como simular una gran cantidad de procesos, para asegurarse de que la aplicación no presente problemas de rendimiento o errores.

Tabla 19. Pruebas de caja negra para CU-03.

El caso de prueba número 3 arrojó resultados no deseados. En un principio, no se consideraron limitaciones al ejecutar la simulación con numerosos procesos, lo que afectó negativamente a dispositivos con especificaciones más limitadas. Estos dispositivos, en situaciones extremas, experimentaron bloqueos de la aplicación o interrupciones debido a la alta demanda de recursos.

Como solución, dado que el propósito es proporcionar ejemplos didácticos simplificados en lugar de ejecutar simulaciones realistas, se ha establecido un límite en la cantidad máxima de procesos a agregar y en el número máximo de unidades temporales en CPU por proceso, fijándolos en 99 como valores máximos.

11.3.4. CU-04. Visualizar Resultados.

A continuación, se presenta la tabla 20 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Verificar que la página de resultados muestra todos los resultados esperados de la simulación, como tiempos de inicio, finalización y espera de cada proceso.
2	Confirmar que los valores numéricos mostrados en la página de resultados son precisos y coinciden con los cálculos esperados.
3	Asegurarse de que, si la tabla de resultados es larga y no cabe en la pantalla, se pueda desplazar verticalmente para ver todos los datos.
4	Probar la página de resultados en varios dispositivos con diferentes tamaños de pantalla para asegurarse de que se vea correctamente en todos ellos.

Tabla 20. Pruebas de caja negra para CU-04.

Todas las pruebas de este módulo específico han concluido con éxito, sin necesidad de correcciones significativas.

11.3.5. CU-05. Ver Diagrama de Gantt

A continuación, se presenta la tabla 21 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Verificar que, al seleccionar la página correspondiente en la aplicación, se accede a la página del diagrama de tiempos sin problemas.
2	Verificar que los iconos y colores utilizados en la leyenda sean comprensibles y proporcionen una representación clara de los estados.
3	Asegurarse de que el diagrama se muestra correctamente en la pantalla, con una representación visual clara de los procesos y sus estados en el tiempo.

Tabla 21. Pruebas de caja negra para CU-05.

En el caso de prueba número 2, se obtuvieron resultados no deseados en la representación de iconos y colores en la leyenda y el diagrama posterior. Se solucionó al reasignar el esquema de colores personalizado de la app en lugar de utilizar los colores predeterminados del IDE (Android Studio).

11.3.6. CU-06. Visualizar Estados de Colas de Procesos

A continuación, se presenta la tabla 22 con las pruebas realizadas, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Verificar que, al seleccionar la página correspondiente, se accede a la página para la visualización de las colas de procesos.
2	Verificar que las colas muestran la información adecuada correspondiente.

Tabla 22. Pruebas de caja negra para CU-06.

En este apartado, se verificó que el caso de prueba número 2 generaba resultados no deseados en la representación de las colas de procesos, ya que los procesos se dispusieron horizontalmente en lugar de verticalmente, que era la disposición deseada. La solución implicó procesar la lista de procesos de cada momento de manera individual, carácter a carácter, para disponerlos correctamente en cada celda de la tabla.

11.3.7. CU-07. Cambiar el Tema de la Aplicación

A continuación, se presenta la tabla 23 con las pruebas realizadas en la aplicación, sus resultados esperados y las acciones correctivas tomadas en caso de que sea necesario.

	Descripción
1	Verificar que, al seleccionar la opción de cambio de tema en la aplicación, el tema de la interfaz cambie de acuerdo con la elección del usuario, ya sea de modo claro a oscuro o viceversa.
2	Comprobar que el tema seleccionado por el usuario se mantenga constante en todas las pantallas de la aplicación después de haber realizado el cambio.
3	Verificar que la opción de cambio de tema esté disponible y funcione correctamente desde cualquiera de las páginas de la aplicación.
4	Verificar que, al abrir la aplicación por primera vez, se aplique el tema predeterminado del sistema y que el usuario tenga la opción de cambiarlo.

Tabla 23. Pruebas de caja negra para CU-07.

En este caso, todas las pruebas de este módulo específico han concluido con éxito, sin necesidad de correcciones adicionales.

11.4. Pruebas de integración

Las pruebas de integración juegan un papel esencial en el proceso de desarrollo de *software*, ya que permiten verificar que los diferentes módulos y componentes de una aplicación funcionen de manera armoniosa cuando se combinan. En este subapartado, se explorarán en detalle las pruebas de integración que se han llevado a cabo en el contexto de este TFG.

Es importante señalar que, aunque los distintos módulos y páginas de la aplicación están interconectados, la comunicación se centra en la página principal, que actúa como punto central de entrada y control. Recibe y gestiona los datos ingresados por el usuario, y luego el resto de las páginas se encargan de presentar y procesar estos datos.

En este sentido, las pruebas de integración se centran en asegurar que la página principal pueda comunicarse eficazmente con los demás módulos, garantizando que los datos se transmitan y procesen correctamente a medida que avanzan a través de la aplicación.

Esto es esencial para mantener la coherencia y el funcionamiento sin problemas de la aplicación en su conjunto, a pesar de que la comunicación entre las páginas secundarias es relativamente limitada. En las siguientes secciones, se describirán las pruebas específicas diseñadas para evaluar esta integración clave entre la página principal y los demás componentes.

11.4.1. Caso de prueba 1: Paso del listado de procesos

Como se mencionó previamente, la página principal desempeña un papel crucial al solicitar y almacenar los datos del usuario en un listado de procesos, que es un componente esencial para el funcionamiento de la aplicación. Dado que todos los demás módulos dependen de esta información, es vital que tengan acceso a dicho listado.

Esta prueba tuvo un pequeño error en el que el listado de procesos creados era constantemente sobrescrito cada vez que se agregaba un nuevo proceso, por lo que en ningún momento se rellenaba correctamente. Su solución únicamente implicó modificar la forma en la que se agregaban los nuevos procesos, agregándolos a un listado principal y no creando dicho listado al agregar el proceso.

11.4.2. Caso de prueba 2: Paso de los resultados

Después de la ejecución de la simulación, el listado de procesos se actualiza con los resultados obtenidos, lo que incluye la información relevante para cada proceso. Además, se crea un nuevo listado que contiene los estados de los procesos a lo largo de la simulación. En este punto, estos dos listados se vuelven esenciales para los módulos posteriores.

Dado que los próximos módulos se centran en la representación gráfica de estos resultados, es crucial que tengan un acceso adecuado a la información almacenada en ambos listados mencionados.

Esta fase de pruebas tampoco presentó errores significativos.

11.5. Pruebas de sistema

En el contexto de desarrollo de *software*, las pruebas de sistema desempeñan un papel fundamental. Estas pruebas tienen como objetivo evaluar el sistema en su conjunto, verificando que todas las partes individuales funcionen de manera cohesiva y cumplan con los requisitos y especificaciones definidos. Es en esta fase donde se pone a prueba la aplicación en su totalidad, antes de su lanzamiento, para identificar posibles problemas o incompatibilidades que puedan surgir en un entorno de uso real.

En este apartado, se detallarán las pruebas de sistema realizadas en el proyecto, abarcando diversos aspectos de la aplicación. Estas pruebas buscan confirmar que la aplicación funcione según lo previsto (ver Capítulo 7. Especificación de Requisitos) y que los diferentes módulos y componentes interactúen adecuadamente entre sí.

A continuación, se describirán en detalle las pruebas de sistema llevadas a cabo para garantizar un rendimiento óptimo y la total funcionalidad de la aplicación en su conjunto.

11.5.1. Pruebas de Interfaz de Usuario (UI)

Durante esta prueba, se evaluó que todas las páginas de la aplicación mantuvieran su apariencia esperada y que todos los elementos estuvieran correctamente dispuestos en cada una de ellas. Se verificó que esta disposición fuera coherente en diversas resoluciones de pantalla y dispositivos.

Asimismo, se examinó la fluidez observable en la navegación entre las pantallas y se evaluaron las interacciones con la interfaz, asegurando que estuviera dentro de los límites aceptables. También se comprobó que todos los elementos interactivos funcionaran sin problemas. Todo esto se realizó mediante comprobaciones observables, sin mediciones exactas de tiempos ya que no se ha considerado necesario.

11.5.2. Pruebas de navegación

Se llevó a cabo una revisión exhaustiva para confirmar que la navegación entre las diferentes páginas de la aplicación fuera coherente y sin errores. Esto implicó examinar cada transición entre páginas y verificar que todas las interacciones de navegación funcionaran correctamente. El objetivo era garantizar que el flujo de la aplicación fuera intuitivo y que los usuarios pudieran moverse entre las diferentes secciones de manera fluida y sin problemas. Se comprobaron en detalle los enlaces, botones y gestos utilizados para la navegación, y se aseguró que no hubiera discrepancias ni problemas de funcionamiento.

Adicionalmente, se llevaron a cabo pruebas para evaluar la capacidad de retorno a la página principal desde cualquier punto de la aplicación. Esto implicó probar todas las rutas posibles para volver a la pantalla de inicio, independientemente de en qué página se encontrara el usuario en un momento dado. Se confirmó que todas las opciones de retorno funcionaran correctamente y que los usuarios pudieran regresar a la pantalla principal de manera efectiva en cualquier punto de su interacción con la aplicación.

11.5.3. Pruebas de estrés

Se sometió la aplicación a situaciones de carga extrema con el objetivo de evaluar su estabilidad y capacidad de recuperación. Estas pruebas se realizaron para determinar cómo se comportaba la aplicación cuando se le exigía al máximo de su capacidad. Se simularon escenarios en los que un gran número de procesos con eventos de E/S se ejecutaban simultáneamente, lo que podría generar una carga significativa en la aplicación.

Estas pruebas permitieron evaluar la respuesta de la aplicación bajo condiciones extremas y observar que, bajo una carga grande de procesos, la herramienta puede llegar a colapsar la representación de los resultados obtenidos. Aunque se considera como un caso excepcional al cual no se debería de llegar debido al ámbito educativo de la aplicación.

11.6. Pruebas de aceptación

Estas pruebas se centran en asegurar que la aplicación cumpla con las expectativas y requisitos del cliente o usuario final antes de su implementación. Son la última barrera antes del lanzamiento oficial de la aplicación y se realizan para validar que el *software* es apto para su uso en situaciones reales.

En este apartado, se presentarán las pruebas de aceptación llevadas a cabo en el proyecto. Estas pruebas están diseñadas para verificar que la aplicación satisface plenamente los criterios y necesidades del usuario final, asegurando que todas las funcionalidades respondan adecuadamente a las demandas planteadas. Se describirán detalladamente las pruebas de aceptación realizadas, destacando cómo se han llevado a cabo y qué resultados se han obtenido.

11.6.1. Validación por el director del TFG

En este caso, se considera la evaluación por parte del director del TFG de la aplicación con el objetivo de identificar posibles errores y evaluar las mejoras técnicas realizadas durante el desarrollo.

El director del TFG desempeña un papel crucial en la validación y revisión de la aplicación, ya que su experiencia y conocimiento son fundamentales para garantizar la calidad y la coherencia del proyecto. A través de esta evaluación, se busca obtener retroalimentación valiosa que contribuya a perfeccionar la aplicación y asegurar que cumple con los estándares requeridos.

A lo largo de las diferentes reuniones, se han identificado diversas mejoras y cambios que han contribuido a la evolución de la aplicación. Algunas de estas mejoras incluyen:

- Cambio a una disposición horizontal para toda la aplicación, lo que ha mejorado la usabilidad y la experiencia del usuario.

- Implementación de una disposición en formato paginado para las distintas pantallas de la aplicación, lo que facilita la navegación y la visualización de la información.

- Agregación del módulo de las colas de procesos, lo que ha enriquecido la funcionalidad y la capacidad de representación de datos en la aplicación.

Estos cambios y mejoras se han incorporado de acuerdo con las necesidades y los requisitos del proyecto, y su evaluación por parte del director del TFG es esencial para validar su efectividad y relevancia en el contexto de la aplicación.

11.6.2. Pruebas de escenario básico

Durante las pruebas de aceptación, se evaluó exhaustivamente el escenario básico de la aplicación para asegurar su funcionamiento sin inconvenientes. En primer lugar, se confirmó que los usuarios pudieran abrir la aplicación en dispositivos Android sin experimentar ningún error en el proceso de inicio. Esto es esencial para garantizar que la aplicación sea accesible y utilizable en una variedad de dispositivos Android de manera confiable.

En cuanto a la selección de algoritmos de planificación, se comprobó que los usuarios pudieran realizar esta tarea de manera sencilla y sin enfrentar obstáculos. La elección del algoritmo es crucial para la ejecución exitosa de la simulación, y asegurarse de que esta funcionalidad esté disponible sin contratiempos es esencial.

En este caso se detectaron funcionamientos no deseados en los que el algoritmo era seleccionado visualmente en la interfaz, pero no era correspondido con la funcionalidad asociada. Para solucionarlo, se tuvo que considerar agregar una variable adicional, cuya función se basaría en servir de comprobante del algoritmo seleccionado para posteriormente ejecutar la simulación adecuada.

Otro aspecto importante evaluado fue la capacidad de los usuarios para introducir datos con precisión, tanto para agregar procesos como para incluir eventos de E/S. Se verificó que los campos de entrada funcionaran correctamente y que las validaciones impidieran la introducción de datos incorrectos o no válidos.

Además, se confirmó que los usuarios pudieran visualizar los datos ingresados de manera clara y comprensible en la interfaz de usuario. La presentación adecuada de la información es fundamental para que los usuarios puedan revisar y verificar los datos que han introducido, lo que contribuye a una experiencia de usuario positiva.

Finalmente, se verificó que los usuarios pudieran ejecutar la simulación sin problemas y que la aplicación respondiera de manera eficaz a esta acción. Dado que la simulación es la característica central de la aplicación, es esencial que esta función se realice sin errores y que los resultados sean coherentes con los datos de entrada. Estas pruebas de escenario básico aseguraron que la aplicación cumpliera con sus funcionalidades esenciales y que los usuarios pudieran interactuar con ella de manera efectiva.

11.6.3. Pruebas de Interacción con la Interfaz de Usuario

Se prestó especial atención a la evaluación de la facilidad de uso de la aplicación. Esto implicó asegurarse de que los usuarios pudieran navegar de manera intuitiva por las diferentes páginas de la aplicación sin encontrar obstáculos significativos. La navegación es un aspecto crítico para la experiencia del usuario, por lo que se verificó que los elementos de la interfaz de usuario estén dispuestos de manera lógica y que los usuarios pudieran acceder a las funciones clave de manera clara y sencilla.

Además de la navegación, se evaluó la capacidad de los usuarios para realizar acciones importantes, como eliminar procesos o cambiar el tema de la aplicación, de manera eficiente y sin confusiones. Estas acciones son parte integral de la funcionalidad de la aplicación, y es crucial que los usuarios puedan llevarlas a cabo sin dificultad. Se aseguró que los botones y elementos de control estén ubicados de manera que los usuarios puedan identificarlos fácilmente y que las acciones sean intuitivas.

En resumen, estas pruebas se centraron en la experiencia del usuario y la usabilidad de la aplicación. Se garantizó que los usuarios pudieran interactuar con la aplicación de manera fluida y que las acciones comunes sean accesibles y comprensibles, lo que contribuye a una experiencia positiva en general.

11.6.4. Pruebas de Resultados

Se comprobó que los usuarios pudieran ver los resultados de la simulación de manera clara y comprensible. Los resultados de la simulación son un componente esencial de la aplicación, y es fundamental que los usuarios puedan acceder a estos datos de forma sencilla. Se verificó que la presentación de los resultados esté diseñada de manera que sea fácil de entender, con una disposición lógica y una representación visual efectiva.

Además de la claridad en la presentación de los resultados, se confirmó que los tiempos de espera, tiempos de retorno y otros valores relacionados con la simulación sean correctos y coincidan con los resultados esperados. Estos datos son cruciales para comprender el funcionamiento de los algoritmos de planificación y la eficiencia del sistema en general. Se aseguró que los cálculos y representaciones sean precisos, lo que contribuye a la confiabilidad de la aplicación.

11.6.5. Pruebas de Usabilidad

Se llevaron a cabo evaluaciones con usuarios reales para evaluar la facilidad de uso y la experiencia del usuario. Se prestaron especial atención a la interacción con la interfaz de usuario, la navegación entre páginas y la ejecución de acciones como eliminar procesos o cambiar el tema de la aplicación.

El objetivo principal fue identificar posibles áreas de mejora en la usabilidad de la aplicación. Se recopilaron comentarios y retroalimentación de los usuarios para comprender sus necesidades y expectativas. Estos datos ayudaron a refinar la interfaz de usuario y hacer ajustes que mejoren la experiencia del usuario en general.

El cambio más relevante agregado tras realizar estas pruebas corresponde con la consideración de poder avanzar entre los campos del formulario principal pulsando la tecla “intro”, ya que el teclado suponía un problema para la mayoría de usuarios a la hora de visualizar y cumplimentar los campos del formulario, teniendo que cerrarlo manualmente si se quería avanzar entre dichos campos.

Bloque
V. Conclusiones y futuras mejoras

Capítulo 12. Conclusiones

Este TFG representa un hito importante para el alumno, siendo el colofón de su carrera universitaria. Se ha diseñado, desarrollado y probado una aplicación de simulación de algoritmos de planificación de procesos para dispositivos Android.

Se han enfrentado desafíos que han obligaron a repensar enfoques y adoptar soluciones creativas. Se ha aprendido a gestionar el tiempo, los recursos y las expectativas. Pero lo más importante, se ha adquirido una comprensión más profunda de la importancia de la planificación y la organización en el desarrollo de *software*.

En este capítulo, se presentarán las conclusiones clave que emergen de este proyecto. Se hará una reflexión sobre los hitos alcanzados, los obstáculos superados y las lecciones valiosas aprendidas a lo largo de este proceso.

12.1. Objetivos operacionales alcanzados

En el desarrollo de este proyecto, se han logrado cumplir de manera satisfactoria los objetivos operacionales establecidos (ver Capítulo 3. Objetivos). Estos objetivos operacionales, que detallan las acciones y pasos específicos necesarios para alcanzar los objetivos generales del proyecto, han guiado de manera efectiva el proceso de desarrollo de la aplicación.

El objetivo principal del proyecto, que se centraba en la concepción y desarrollo integral de una aplicación dirigida a dispositivos móviles Android para respaldar la enseñanza de conceptos relacionados con la planificación de procesos, ha sido plenamente alcanzado. La aplicación ha sido diseñada con la capacidad de simular eventos de E/S y la

implementación de un algoritmo de planificación, así como una base robusta y modular para futuros desarrolladores que deseen aumentar o pulir todas aquellas funcionalidades que lo requieran.

Además, los objetivos específicos que abordaron aspectos como el diseño de la interfaz de usuario, la navegación entre módulos, la selección de algoritmos, la gestión de datos, la visualización de resultados, la eliminación de procesos y la representación gráfica de tiempos y estados de colas también se han cumplido de manera exitosa.

La aplicación presenta una interfaz compacta y accesible, una navegación clara entre sus módulos, una selección sencilla de algoritmos, una especificación clara de datos, una gestión eficiente de eventos de E/S, una visualización organizada de datos ingresados, una eliminación flexible de procesos y una representación clara de resultados numéricos y gráficos.

En resumen, el proyecto ha logrado cumplir con los objetivos operacionales establecidos, asegurando el desarrollo de una aplicación efectiva y funcional que cumple con su propósito educativo y técnico.

12.2. Objetivos formales alcanzados

Los objetivos formales establecidos para este proyecto han sido alcanzados con éxito, lo que demuestra un sólido logro en varios aspectos clave:

1. **Adquisición de conocimientos de desarrollo**: Durante el desarrollo de esta aplicación, se ha logrado una comprensión profunda y detallada de todas las etapas implicadas en la creación de aplicaciones, lo que ha permitido abordar los desafíos de manera efectiva y respaldar el diseño y desarrollo integrales de la aplicación.

2. **Manejo de Android Studio**: Se ha adquirido una experiencia sólida en el uso de Android Studio, la plataforma principal para el desarrollo de aplicaciones Android.

3. **Iniciación en el lenguaje de programación Kotlin**: Se ha alcanzado un nivel adecuado en el lenguaje de programación Kotlin, lo que ha facilitado la implementación de la lógica de la aplicación y la manipulación de datos.

4. **Aprendizaje en Jetpack Compose**: Se ha logrado familiarizarse con Jetpack Compose, la biblioteca de diseño de interfaces de última generación. Esto ha permitido el desarrollo de una interfaz de usuario intuitiva y moderna que mejora la experiencia del usuario.

5. **Análisis y resolución de problemas**: Se ha perfeccionado la capacidad de analizar y resolver problemas complejos que surgieron durante el desarrollo. Esto ha garantizado la estabilidad del producto final.

6. **Toma de decisiones estratégicas para la experiencia de usuario**: Las decisiones estratégicas tomadas durante el proceso de diseño y desarrollo se basaron en una comprensión profunda de las necesidades del usuario y los principios de diseño. Esto ha mejorado significativamente la experiencia general del usuario.

7. **Generación de documentación completa**: Se ha elaborado una documentación completa y detallada que abarca todas las fases del proyecto. Esto asegura la transparencia y la replicabilidad del proceso, facilitando futuras mejoras y desarrollos.

En conjunto, estos logros demuestran un compromiso sólido con el aprendizaje, la resolución de problemas y la entrega de una aplicación de calidad que cumple con los objetivos formales del proyecto.

Capítulo
13. Futuras mejoras

Un proyecto de *software* nunca es estático, siempre existe margen para la mejora continua. A medida que esta aplicación de simulación de algoritmos de planificación de procesos toma forma, es imperativo considerar cómo evolucionará y se adaptará en el futuro.

En este apartado, se explorarán las áreas donde esta aplicación podría beneficiarse de futuras mejoras y expansiones. A medida que se identifican las limitaciones actuales, también se observan oportunidades para un mayor desarrollo. Se considerará cómo incorporar nuevas características, mejorar la experiencia del usuario y ampliar la utilidad de la aplicación. Con una mirada hacia el futuro, esta sección sienta las bases para que este proyecto siga siendo relevante y útil en un entorno tecnológico en constante cambio. Algunas de las posibles mejoras incluyen:

- **Ampliar los algoritmos implementados**: Se considera la incorporación de un mayor número de algoritmos de planificación, lo que enriquecería la aplicación y proporcionaría a los usuarios una gama más amplia de opciones y recursos para su aprendizaje.

- **Ajustar los campos que requieran los nuevos algoritmos**: Con la incorporación de nuevos algoritmos, es probable que estos requieran introducir nuevos datos adicionales, por lo que el espacio en pantalla para estos campos se vería gravemente afectado. Es esencial ajustar dichos campos de entrada de datos de manera efectiva para que reflejen los requisitos específicos de cada algoritmo. Esto podría lograrse mediante una ventana emergente o un asistente de configuración, lo que simplificaría la experiencia del usuario al proporcionar orientación contextual.

- **Modificar procesos**: Actualmente, la aplicación permite la eliminación de procesos, pero se podría mejorar permitiendo la modificación de procesos existentes. Esto brindaría a los usuarios un mayor control sobre los datos de entrada y permitiría realizar ajustes sin necesidad de eliminar y volver a ingresar información.

- **Señalar el proceso que entra y sale de las colas**: Para una comprensión más clara de la simulación, se podría agregar una función que destaque o señale el proceso que se está moviendo en cada momento dentro de las colas de procesos. Esto ayudaría a los usuarios a seguir el progreso de manera más efectiva.

- **Traducir la aplicación a otros idiomas**: Para ampliar el alcance y la accesibilidad de la aplicación, sería beneficioso traducirla a varios idiomas adicionales. Esto permitiría a usuarios de diferentes regiones y con diferentes idiomas nativos utilizar la aplicación de manera efectiva. Para ello, se debería incorporar un selector de idiomas que permita a los usuarios elegir su preferencia de idioma al iniciar la aplicación. Esto contribuiría a una experiencia de usuario más inclusiva y global.

Estas mejoras potenciales permitirían que la aplicación siga evolucionando y mejorando su utilidad para la enseñanza de la planificación de procesos en el contexto de los sistemas operativos. Además, podrían contribuir a una experiencia de usuario aún más enriquecedora y eficiente.

Bibliografía

[1] Revista Virtual Pro. Virtual Pro. Obtenido de https://www.virtualpro.co/noticias/que-es-la-planificacion-de-procesos-de-un-sistema-operativo (Consultado el 20 de abril de 2023).

[2] Stallings, W. (2005). Sistemas Operativos, Aspectos internos y principios de diseño. Prentice Hall.

[3] A. Mclver, I. M. (2011). Sistemas operativos, 6a edición. Cengage Learning.

[4] Esquemas de colores de la UCO: https://www.uco.es/servicios/actualidad/images/documentos/identidad-corporativa/Manual_UCO_definitivo.pdf (Consultado el 27 de mayo de 2023).

[5] *Clean code*: https://www.freecodecamp.org/news/clean-coding-for-beginners/ (Consultado el 10 de junio de 2023).

[6] Android Studio: https://developer.android.com/studio/intro?hl=es-419 (Consultado el 26 de junio de 2023).

[7] Kotlin: https://developer.android.com/kotlin?hl=es-419 (Consultado el 15 de junio de 2023).

[8] Jetpack Compose: https://developer.android.com/jetpack/compose?hl=es-419 (Consultado el 06 de mayo de 2023).

[9] Diagrama de Gantt: https://asana.com/es/resources/gantt-chart-basics (Consultado el 12 de junio de 2023).

[10] Boonsuen. (s.f.). *Boonsuen (GitHub)*. Obtenido de https://boonsuen.com/process-scheduling-solver (Consultado el 18 de enero de 2023).

[11] Gregor Koytainy, P. D.-I. (2014). *AnimOS CPU-Scheduling*. Obtenido de https://ess.cs.tu-dortmund.de/Software/AnimOS/CPU-Scheduling/ (Consultado el 18 de enero de 2023).

[12] Cooray, H. (2020). *CPU Scheduling Simulator*. Obtenido de https://cpu-scheduling-sim.netlify.app/ (Consultado el 18 de enero de 2023).

[13] Alvareztech. (s.f.). *Alvareztech (GitHub)*. Obtenido de https://github.com/alvareztech/ProcessSimulatorJava (Consultado el 21 de enero de 2023).

[14] IncanatoIT. (2014). *IncanatoIT, Desarrollando Software!* Obtenido de https://www.incanatoit.com/2014/06/simulador-planificacion-de-cpu-memoria.html (Consultado el 20 de enero de 2023).

[15] Ifreddyrondon. (s.f.). *Ifreddyrondon (GitHub)*. Obtenido de https://github.com/ifreddyrondon/ula_so-scheduler (Consultado el 21 de enero de 2023).

[16] Jeico Games. (s.f.). *CPU Simulator (CPU Scheduling)*. Obtenido de https://play.google.com/store/apps/details?id=com.jeicogames.cpusimulator&hl=es_NI&pli=1 (Consultado el 22 de enero de 2023).

[17] Itmarck. (s.f.). *Quantum*. Obtenido de https://play.google.com/store/apps/details?id=software.marcx.quantum (Consultado el 23 de enero de 2023).

[18] Software libre: https://www.gnu.org/philosophy/free-sw.es.html (Consultado del 15 de julio de 2023).

[19] Git: https://git-scm.com/ (Consultado el 12 de junio de 2023).

[20] Github: https://github.com/github (Consultado el 12 de junio de 2023).

[21] UML: https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml (Consultado el 16 de julio de 2023).

[22] Repositorio GitHub del proyecto: https://github.com/Euskindar/TFG.

[23] Diseño responsivo: https://intconsultoria.com/diseno-web-responsive-vs-adaptive-que-diferencia-hay-y-como-afectan-a-mi-web/ (Consultado del 08 de abril de 2023).

[24] Microsoft Word: https://www.microsoft.com/es-es/microsoft-365/word (Consultado el 13 de junio de 2023).

[25] Windows 11: https://www.microsoft.com/es-es/windows/windows-11 (Consultado el 13 de junio de 2023).

Bloque
VI. Apéndices

Manual de Usuario

Capítulo
1. Estructura del manual

El Manual de Usuario es una herramienta esencial diseñada para orientar a los usuarios en la comprensión y el uso efectivo de la aplicación desarrollada como parte de este proyecto. Su propósito principal es servir como un recurso informativo y guía práctica que facilita la interacción con la aplicación y aprovecha al máximo todas sus funcionalidades. En el contexto de esta aplicación específica, el Manual de Usuario es un compañero esencial para estudiantes, profesores y cualquier persona interesada en esta aplicación para la planificación de procesos o en los sistemas operativos.

Este documento proporciona una visión completa de cómo utilizar la aplicación, desde su instalación en un dispositivo Android hasta la realización de simulaciones y la interpretación de los resultados. Los usuarios encontrarán instrucciones detalladas para cada función y módulo, junto con consejos útiles y ejemplos que les ayudarán a sacar el máximo provecho de la aplicación.

A lo largo de este manual, exploraremos la interfaz de usuario, las opciones de configuración, la introducción de datos, la ejecución de simulaciones y la interpretación de los resultados. Además, se explicarán las funcionalidades avanzadas, como la visualización gráfica de tiempos y el manejo de colas de procesos. Este Manual de Usuario está diseñado para garantizar que los usuarios puedan utilizar la aplicación de manera efectiva, independientemente de su nivel de experiencia en planificación de procesos o sistemas operativos.

El Manual de Usuario está estructurado en cuatro secciones fundamentales, cada una diseñada para proporcionar una comprensión clara y detallada de aspectos específicos de la aplicación:

1. **Requisitos mínimos de *software***: En esta sección, se especifican los requisitos mínimos de *hardware* y *software* que su dispositivo debe cumplir para garantizar un rendimiento adecuado de la aplicación. Esto incluye información sobre la versión del sistema operativo Android recomendada y otros aspectos técnicos esenciales para el funcionamiento fluido de la aplicación.

2. **Instalación de la aplicación**: Aquí encontrará instrucciones paso a paso sobre cómo descargar e instalar la aplicación en su dispositivo Android. Se detallan los procesos de descarga desde fuentes seguras, la configuración de permisos necesarios y cómo iniciar la aplicación por primera vez.

3. **Desinstalación de la aplicación**: Esta sección describe cómo eliminar la aplicación de su dispositivo si decide hacerlo en algún momento. Se proporcionan orientaciones claras para garantizar que la desinstalación se realice de manera efectiva y completa, eliminando todos los archivos asociados a la aplicación.

4. **Navegación y uso de la aplicación**: La parte central de este manual se dedica a explorar la interfaz de usuario de la aplicación. Se presentan instrucciones detalladas sobre cómo navegar por las diferentes pantallas y módulos, cómo utilizar las funciones principales, como la selección de algoritmos de planificación y la introducción de datos de procesos, y cómo interpretar los resultados de las simulaciones.

Estas secciones están diseñadas para brindar a los usuarios una experiencia de usuario fluida y garantizar que puedan utilizar la aplicación de manera efectiva, aprovechando al máximo sus características y funcionalidades.

Capítulo
2. Requisitos mínimos de software

Para poder ejecutar la aplicación, solo se requiere un único requisito mínimo de software: su dispositivo móvil debe tener una versión del sistema operativo Android igual o superior a la 7.0, ampliamente conocida como *"Nougat"*.

Este requisito esencial permite que prácticamente cualquier dispositivo Android moderno sea capaz de ejecutar la aplicación sin problemas. Además, es importante destacar que la aplicación está diseñada con un bajo consumo de recursos, lo que significa que no se imponen requisitos mínimos de hardware. Esta característica permite que un amplio rango de dispositivos, desde modelos de gama baja hasta dispositivos más avanzados, pueda disfrutar plenamente de todas las funcionalidades que ofrece la aplicación, sin preocuparse por limitaciones severas de rendimiento.

Capítulo
3. Instalación de la aplicación

La aplicación puede ser instalada mediante la descarga directa desde Play Store o utilizando de forma directa el archivo ".apk" correspondiente. La instalación de la aplicación a partir de un archivo ".apk" es un proceso sencillo. Aquí se detallan los pasos necesarios para llevar a cabo esta instalación:

1. Una vez que haya descargado el archivo de la aplicación en su dispositivo, proceda a seleccionar el archivo ".apk" que representa el instalador de la aplicación.

   **Nota: Tenga en cuenta que, si descargó la aplicación desde la *Google Play Store*, esta se instalará automáticamente y puede omitir este paso.

2. Dado que esta aplicación no se descargó de la Google Play Store, es posible que el sistema emita una alerta indicando que el archivo proviene de una fuente desconocida. Esto es un comportamiento normal de Android para garantizar la seguridad del dispositivo. Puede encontrar dos situaciones similares:

   a. Para continuar con la instalación, simplemente toque la opción "Continuar" en la alerta, como se muestra en la figura 1 que se presenta a continuación.

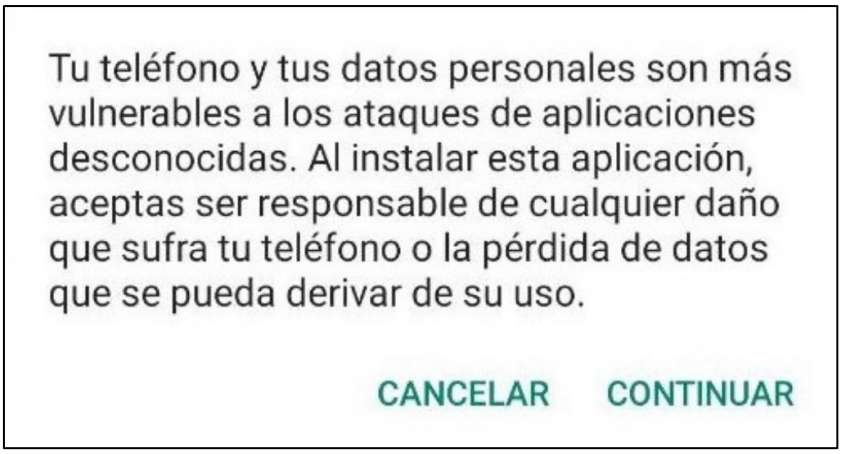

Figura 20. Alerta de instalación de ".apk".

****Nota**: A partir de "Android 8.0", esta confirmación se ha sustituido por una confirmación individual para cada aplicación, ya que *Google* añadió más seguridad al proceso de instalación de aplicaciones externas. En lugar de contar con un único punto en el que dar permiso a todas las apps instaladas en nuestro teléfono para que puedan usar extensiones ".apk", ahora ese permiso debemos darlo aplicación por aplicación.

b. Verá un aviso indicando que "no se pueden instalar aplicaciones de orígenes desconocidos" y se le invitará a entrar en los "Ajustes" o "Configuración" (ver figura 2).

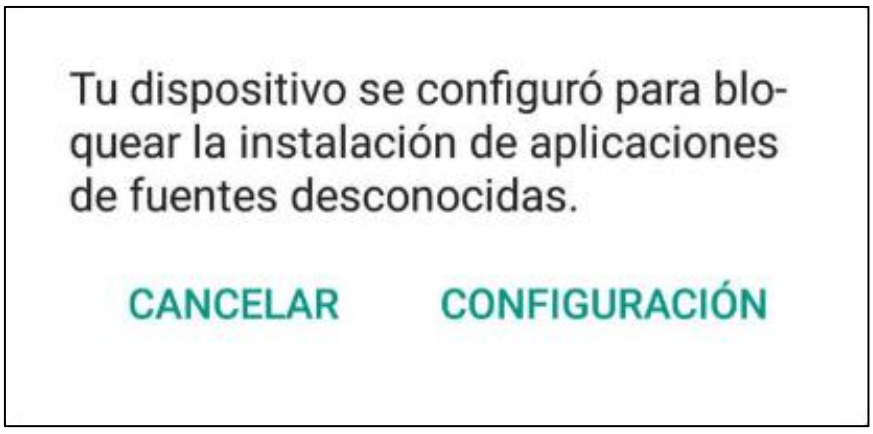

Figura 21. Alerta de instalación de aplicaciones de origen desconocido.

Una vez dentro de los "Ajustes", deberá buscar el apartado "Instalar aplicaciones desconocidas" y activar la opción (ver figura 3). Desde ese momento, la aplicación contará con permisos a la hora de instalar aplicaciones externas.

Figura 22. Permiso de instalación de aplicaciones de origen desconocido.

3. Una vez confirmada la descarga y permitida la instalación, se procederá con la instalación de la aplicación (ver figura 4 y figura 5).

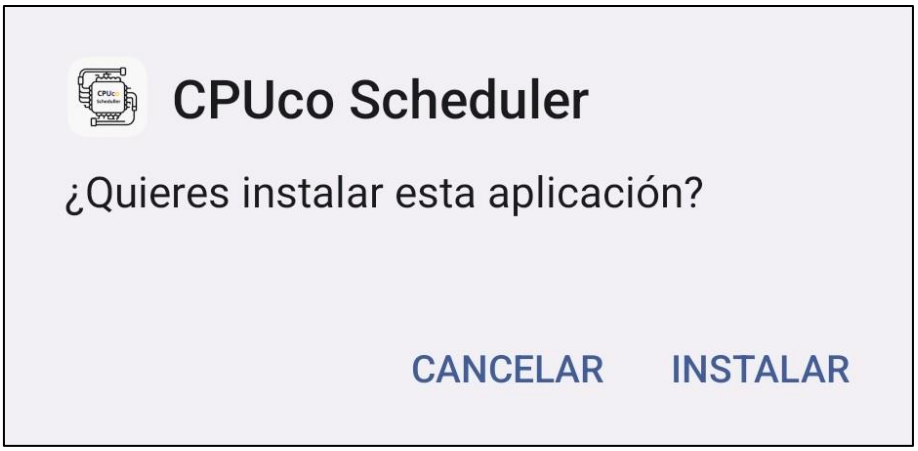

Figura 23. Confirmación para instalación.

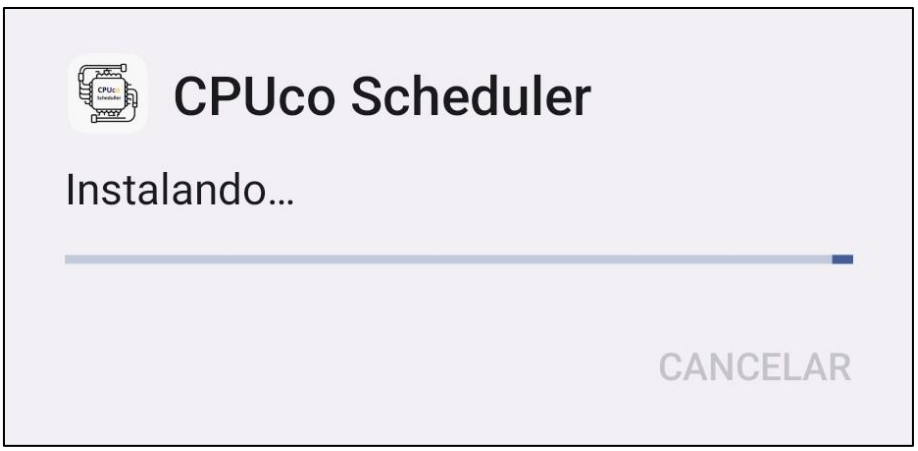

Figura 24. Proceso de instalación.

4. Por último, se mostrará la pantalla de la figura 6, en la que se puede observar que la instalación ha sido exitosa. Puede pulsar la opción “Hecho” para cerrar la ventana y continuar las tareas que estaba realizando o la opción “Abrir” y comenzar la ejecución de la aplicación recién instalada.

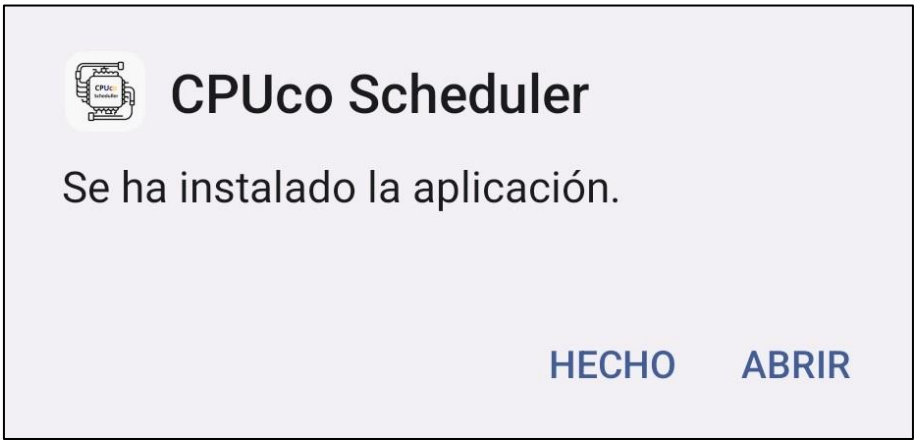

Figura 25. Ventana de instalación exitosa.

Capítulo
4. Desinstalación de la aplicación

Para realizar el proceso de desinstalación de la aplicación, sólo deberá seguir los siguientes pasos. Se le plantearán dos opciones distintas a seguir para lograr este objetivo:

- **Opción 1:**
  1. Acceder al menú principal de aplicaciones.
  2. Buscar el icono correspondiente a la aplicación y mantenerlo pulsado hasta que se desplieguen las opciones correspondientes, tal y como se indica en la figura 7 (estas opciones pueden variar en función a la versión de Android utilizada).

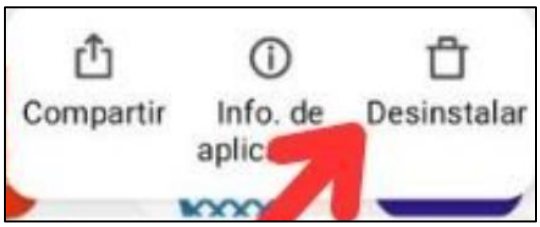

Figura 26. Desinstalación desde el menú principal.

  3. Pulsar la opción de desinstalar y confirmar.

- **Opción 2:**
  1. Acceder al menú de “Ajustes” del dispositivo.
  2. Buscar el apartado de “Aplicaciones” (en algunos dispositivos primero ha de acceder al apartado de “Almacenamiento”).

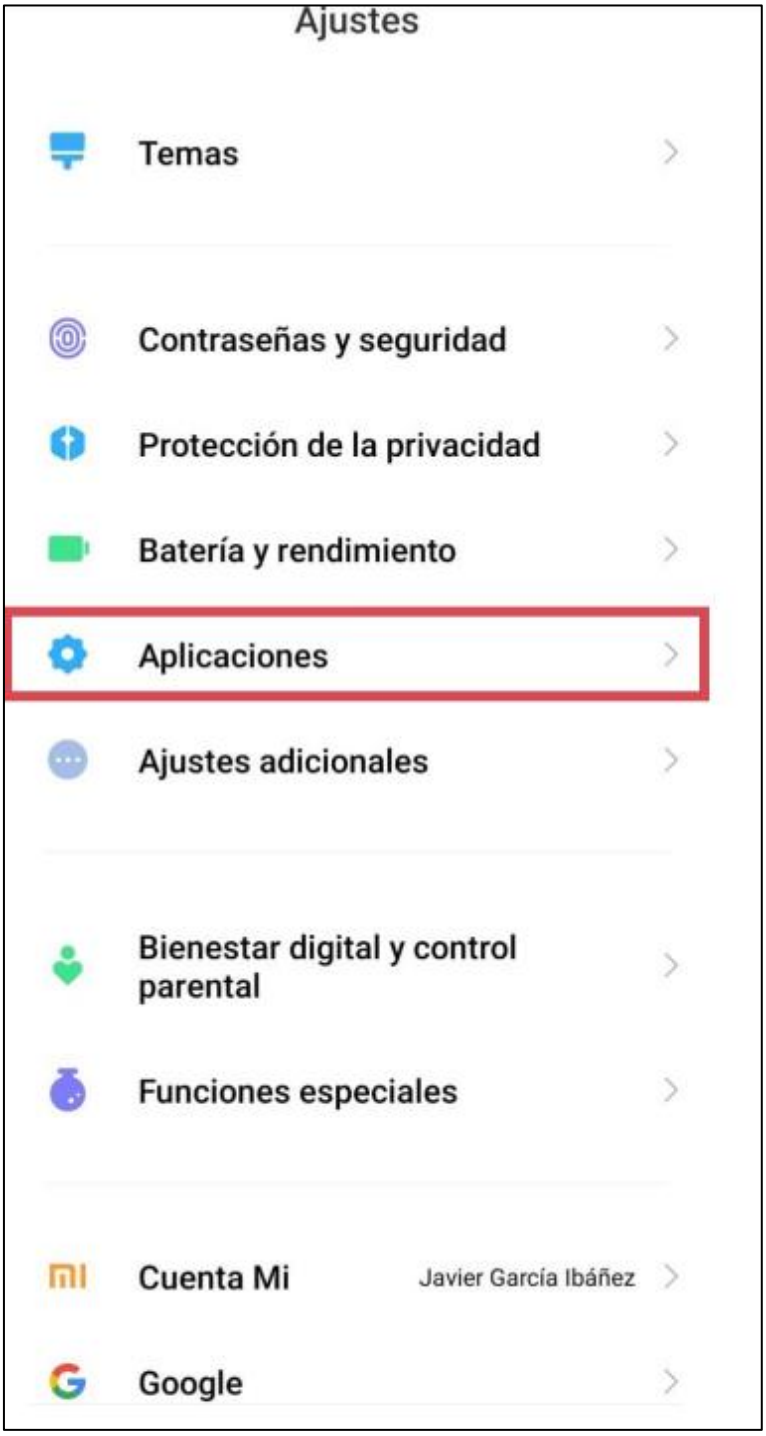

Figura 27. Apartado de "Aplicaciones" en los "Ajustes".

3. Acceder a la opción “Gestionar aplicaciones”.

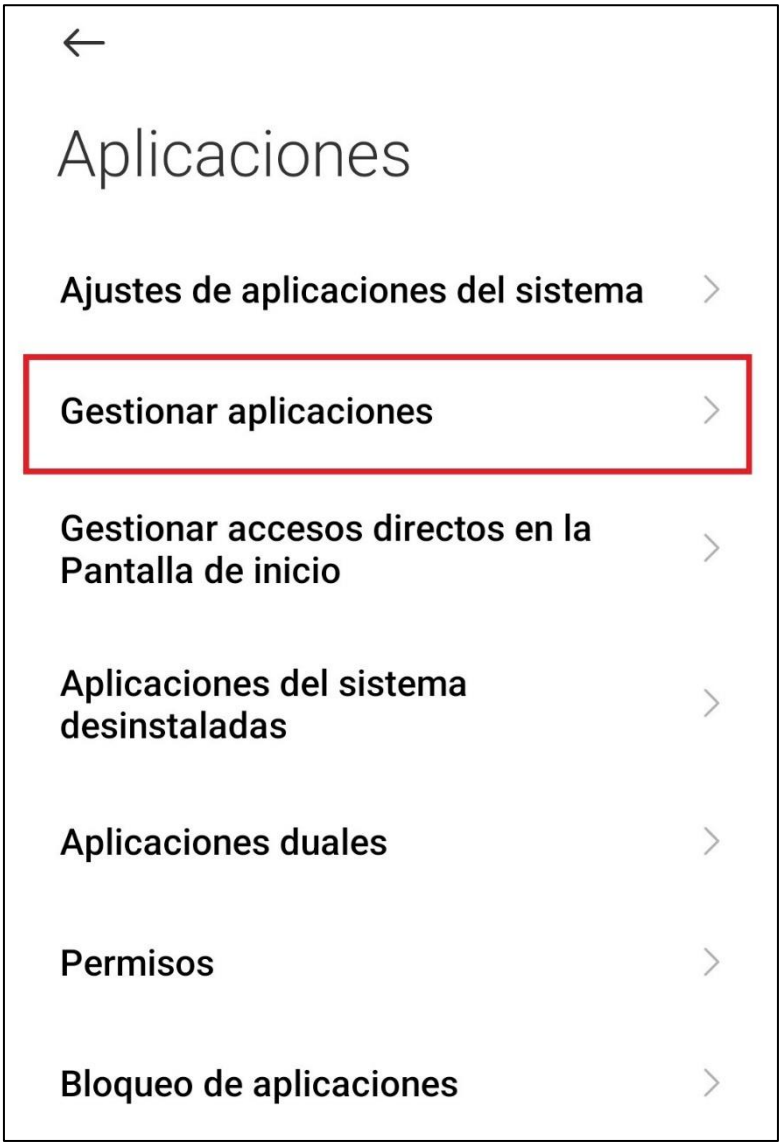

Figura 28. Opción "Gestionar aplicaciones".

4. Buscar la aplicación en el listado de aplicaciones instaladas y pulsar en ella.

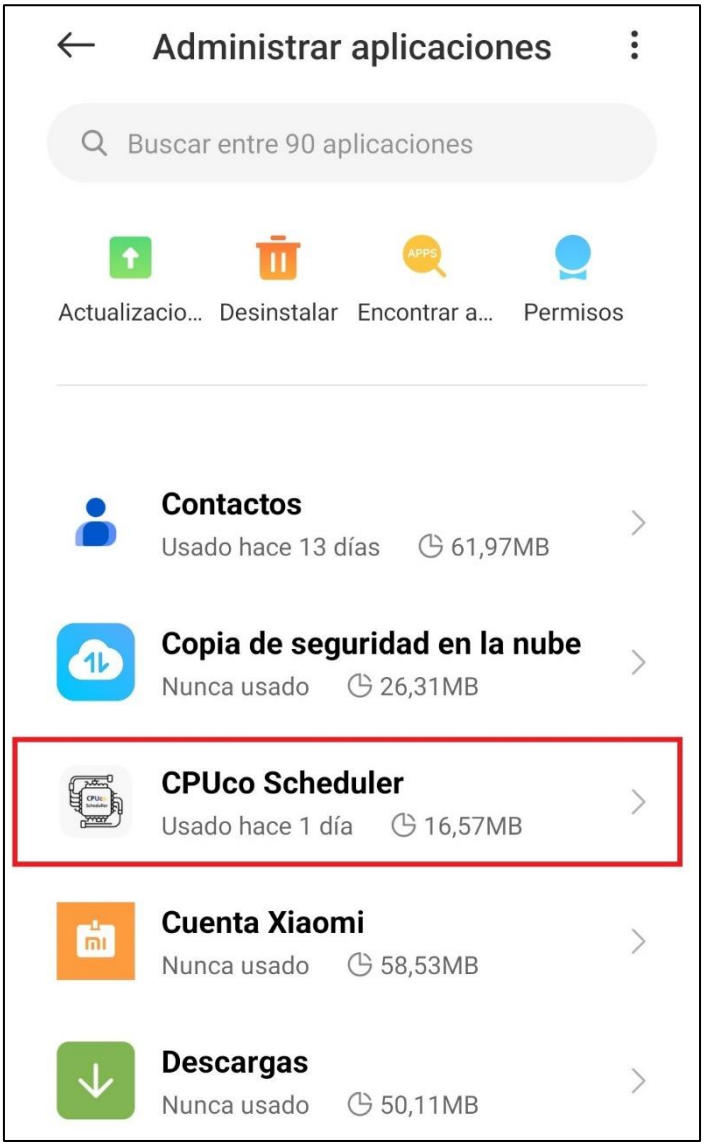

Figura 29. Listado de aplicaciones.

5. Confirmar la desinstalación.

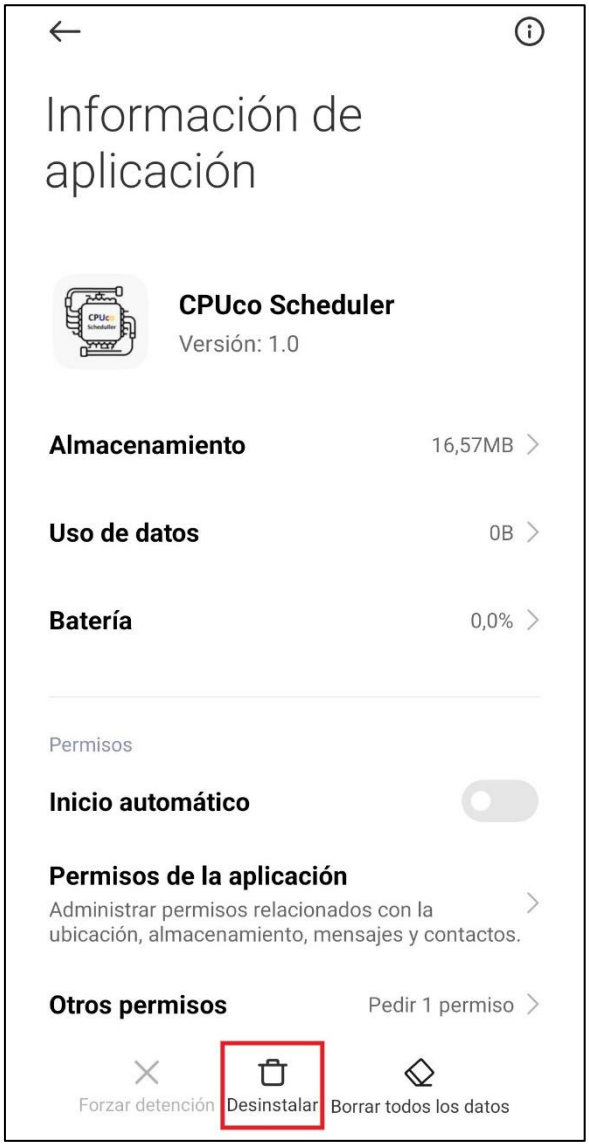

Figura 30. Información de la aplicación.

- **Opción 3:**

Adicionalmente, en caso de haber instalado la aplicación mediante el servicio de Google Play Store, siempre tendrá la opción de buscar la aplicación y desinstalarla directamente desde su página.

Capítulo
5. Navegación y uso de la aplicación

A continuación, se detallará la navegación a través de la aplicación, describiendo los diferentes aspectos presentes en cada pantalla, atendiendo a la interfaz y las distintas opciones disponibles en cada una de ellas. Se utilizarán ejemplos de uso de las distintas funcionalidades disponibles para facilitar el entendimiento de todas ellas.

- **Pantalla principal:**

En la parte superior izquierda de la pantalla principal de la aplicación, encontrará el logotipo simplificado de la universidad. Justo al lado opuesto, verá tres iconos importantes para interactuar con la aplicación.

Figura 31. Iconos superiores de la pantalla principal.

El primero de ellos permite eliminar un proceso específico. Al tocar este icono, se abrirá una ventana emergente que pedirá seleccionar el proceso que desea eliminar a través de un menú desplegable.

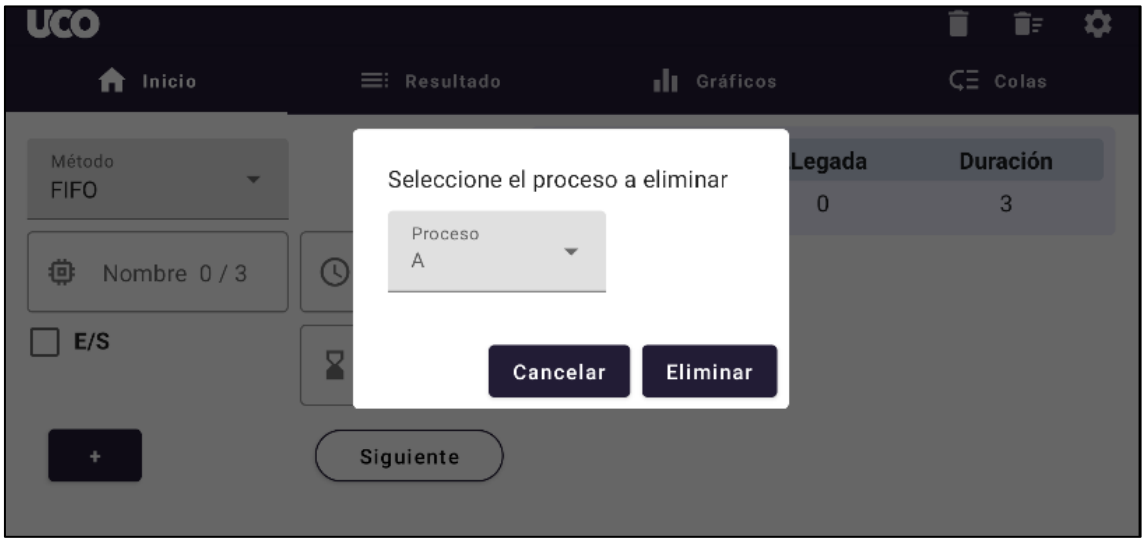

Figura 32. Eliminación de un proceso.

El segundo icono permite eliminar todos los procesos de una vez. Al tocar este icono, aparecerá una ventana emergente que pedirá confirmar la intención de eliminar todos los procesos.

Figura 33. Eliminación de todos los procesos.

El tercer icono es el botón para cambiar el tema de la aplicación. Al tocarlo, la aplicación alternará entre los temas Claro y Oscuro, acompañado de una pequeña animación que indica el tema actual. Esta característica permite personalizar la apariencia de la aplicación según las preferencias.

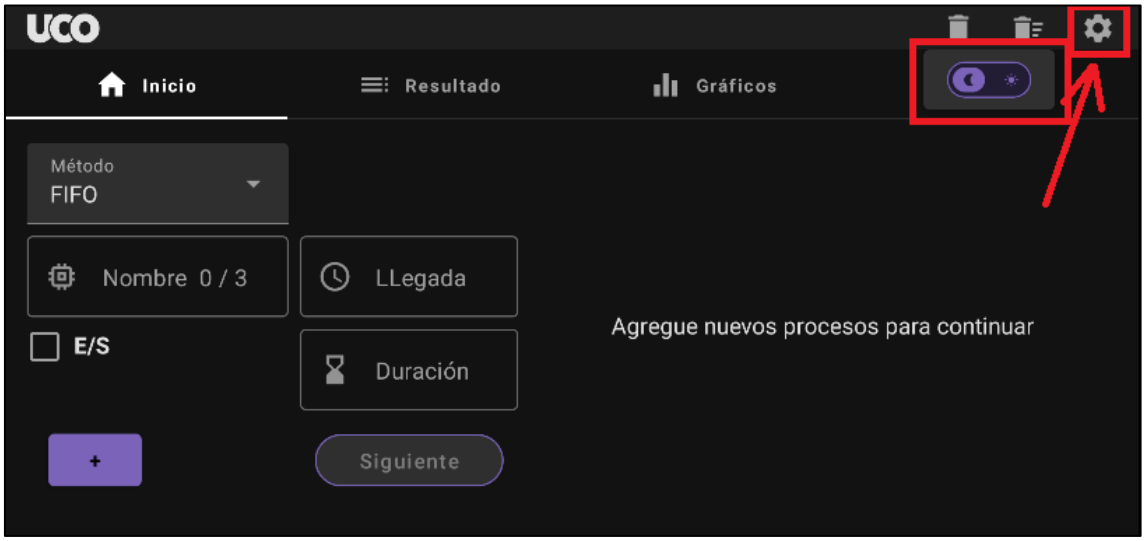

Figura 34. Cambio del tema de la aplicación.

Justo debajo de la barra superior, encontrará una lista de módulos de la aplicación en formato paginado. Puede tocar cualquiera de ellos para acceder a sus respectivas páginas o deslizar la pantalla lateralmente para navegar entre los módulos. Esta barra superior será constante en todas las pantallas de la aplicación, brindando una experiencia de usuario coherente y fácil de usar.

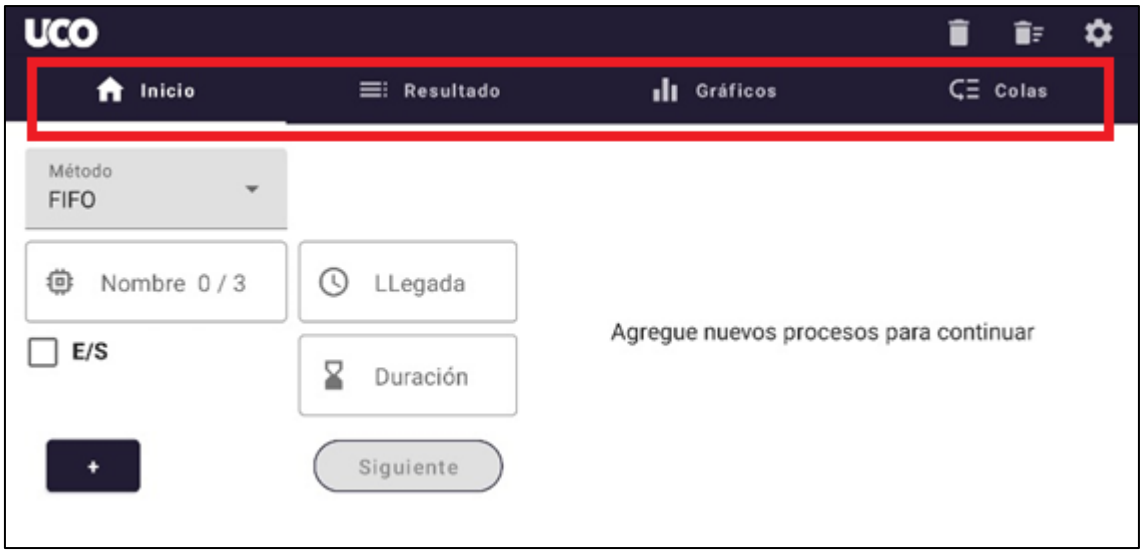

Figura 35. Iconos superiores de la pantalla principal.

La pantalla principal se encuentra en el centro de la interfaz, justo debajo de las barras superiores anteriores. A la izquierda, verá un formulario que permite seleccionar un algoritmo y agregar información sobre cada proceso. Puede elegir el algoritmo deseado en el menú desplegable y los campos necesarios para ese algoritmo se mostrarán automáticamente. Los campos restantes se pueden completar tocándolos y escribiendo la información requerida. Puede cambiar de un campo a otro presionando el botón "*Intro*" del teclado de su dispositivo. Además, los campos para eventos de E/S aparecerán o desaparecerán según seleccione su correspondiente casilla.

Una vez que haya ingresado toda la información del proceso, simplemente toque el botón "+" para agregarlo. Si surge algún error, emergerá un mensaje junto con una indicación sobre el campo afectado.

Figura 36. Agregar proceso.

Después de agregar todos los procesos deseados, toque el botón "Siguiente" para ejecutar el algoritmo y avanzar a la página de resultados.

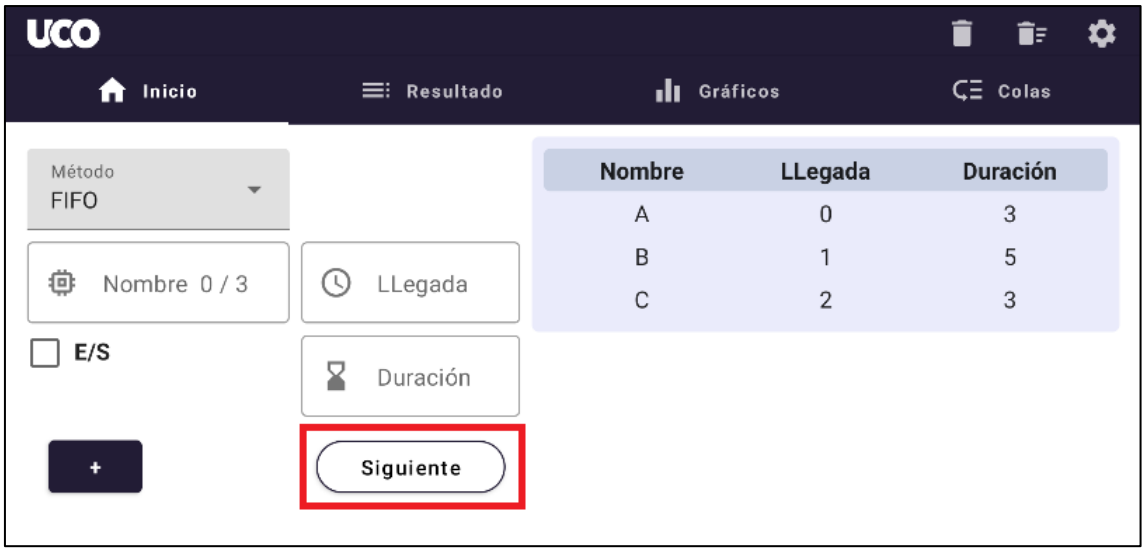

Figura 37. Tabla de los procesos agregados.

En el lado derecho de la pantalla, encontrará una tabla que muestra los procesos que ha agregado, junto con toda su información correspondiente. Si tiene más procesos de los que caben en la pantalla, puede desplazar la tabla verticalmente para ver toda la información.

- **Pantalla de resultados:**

En esta pantalla, que mantiene la consistente parte superior de la interfaz, encontrará una tabla que presenta de manera organizada y clara todos los detalles relacionados con los procesos que ha agregado y que han sido sometidos a la ejecución del algoritmo. Cada fila de esta tabla representa un proceso específico, mientras que las columnas contienen diversos datos temporales resultantes de la simulación.

Figura 38. Página de resultados.

Esta tabla permite visualizar de manera eficiente los resultados de la simulación, facilitando la comprensión de los tiempos de espera, tiempos de estancia y otros valores relevantes para cada proceso. La información se encuentra presentada de manera ordenada, lo que permite realizar un análisis detallado de los datos obtenidos en la simulación.

Desde esta página puede decidir hacia dónde desea desplazarse, pudiendo volver a la pantalla principal para modificar los datos o comenzar una nueva simulación desde cero o, si lo desea, avanzar a las páginas posteriores para visualizar los resultados desde otras perspectivas.

- **Pantalla de gráfica de Gantt:**

La parte superior de esta pantalla se mantiene igual, proporcionando una interfaz coherente y reconocible. A continuación, encontrará una útil leyenda que presenta distintos iconos y colores utilizados en la representación gráfica posterior, junto con una descripción que le ayudará a entender su significado.

Figura 39. Pantalla de la gráfica de Gantt.

Justo debajo de la leyenda, se despliega una tabla que corresponde al diagrama de Gantt. En esta tabla, las filas representan los procesos que ha incluido en la simulación, y las columnas muestran cada uno de los intervalos de tiempo durante la ejecución de la simulación. Cada celda de esta tabla contiene un icono que representa el estado de cada proceso en cada momento específico durante la ejecución de la simulación. Esta representación visual proporciona una clara visión de cómo se desarrolla la ejecución de los procesos a lo largo del tiempo.

- **Pantalla de colas de procesos:**

La parte superior de esta pantalla se mantiene constante, lo que facilita la navegación y la consistencia en toda la aplicación. Aquí, se presentan varias tablas, cada una de ellas correspondiente a cada una de las colas presentes en función al algoritmo empleado para la simulación.

Figura 40. Página de las colas de procesos.

La tabla muestra en su primera columna el momento en el que se encuentra cada una de las colas de procesos, mientras que la segunda columna expone dichas colas. Esto te permite visualizar claramente cómo cambia el orden de los procesos, cuándo entran y salen de las colas y cómo evoluciona su estado a lo largo de la simulación.

Esta representación permite visualizar largas colas de procesos al mismo tiempo que brinda una comprensión precisa de la dinámica de la ejecución de procesos y cómo se gestionan en función del algoritmo elegido.

Manual del Código

Capítulo 1. Introducción

Este documento tiene como objetivo proporcionar una comprensión detallada del código implícito en el desarrollo de la aplicación de planificación de procesos para dispositivos Android. La aplicación se ha concebido como una herramienta educativa y funcional que permite a los usuarios simular la planificación de procesos y comprender los conceptos clave de los sistemas operativos, al mismo tiempo que sirve de proyecto base para que futuros estudiantes puedan ampliar y mejorar las funcionalidades que presenta.

El manual del código se divide en dos partes fundamentales. La primera sección se centra en el paquete principal del código, que alberga todas las funciones, clases e interfaces esenciales para el funcionamiento de la aplicación. Aquí, los desarrolladores encontrarán información detallada sobre la estructura del código, la lógica detrás de las funciones clave y cómo se relacionan entre sí para lograr el flujo de trabajo de la aplicación.

La segunda sección se dedica al paquete de recursos, que almacena todos los elementos visuales, como íconos, imágenes y textos utilizados en la aplicación. Se proporcionará información sobre cómo se gestionan estos recursos y cómo se integran con el código principal para crear una experiencia de usuario coherente y atractiva.

Este manual del código es una valiosa herramienta tanto para aquellos que deseen comprender cómo funciona la aplicación en detalle como para aquellos que buscan realizar modificaciones o mejoras en el proyecto. Proporcionará una visión completa de la estructura y la implementación del código, allanando el camino para un desarrollo continuo y una comprensión profunda de este proyecto.

Capítulo 2. Paquete principal

Este apartado se centra en el "Paquete Principal" de la aplicación. El módulo principal constituye el núcleo esencial de la aplicación, albergando todos los archivos relacionados tanto con su funcionalidad como con el aspecto visual de la interfaz de usuario.

Para garantizar una organización eficiente y facilitar el mantenimiento del código, el módulo principal se divide en varias carpetas distintas, cada una con un propósito específico. Estas carpetas son:

- ***"navigation"***: Aquí residen todos los archivos dedicados a gestionar la navegación entre las diferentes páginas de la aplicación. Esta estructura modulariza la lógica de la navegación, lo que facilita la incorporación de nuevas pantallas y la gestión de las transiciones.

- ***"ui/theme"***: En esta carpeta se encuentran los ficheros de configuración que definen los colores generales, formas predefinidas y otros elementos visuales relacionados con los temas de la aplicación. Esto permite un control coherente y sencillo de la apariencia de la aplicación.

- ***"usecases"***: Cada fichero contenido en esta carpeta se corresponde con un caso de uso específico de la aplicación. Esta modularización permite que cada funcionalidad de la aplicación esté contenida en archivos individuales, lo que facilita la comprensión y el mantenimiento del código.

- ***"util"***: En esta carpeta se asignan todos los ficheros correspondientes a la gestión de todas las utilidades comunes de la aplicación. Esta carpeta se subdivide en dos

subcarpetas: **"*classes*"** y **"*extensions*"**. En la primera, se encuentran los ficheros destinados a definir nuevas clases que resultan útiles en la aplicación. Mientras que, en la segunda, se albergan funcionalidades generales que se aplican en toda la aplicación, mejorando la experiencia del usuario o modularizando tareas comunes. También se encuentra aquí un fichero que contiene valores accesibles desde cualquier punto de la aplicación, lo que facilita la compartición de estructuras y datos entre diferentes funciones.

Además, en la raíz del paquete principal, se encuentra el fichero **"*MainActivity.kt*"**, que actúa como el punto de entrada principal de la aplicación, iniciando la ejecución del resto de elementos.

A continuación, se explorará con más detalle el contenido y el código específico de cada una de estas carpetas y ficheros. Esto permitirá comprender mejor cómo se estructura y opera el núcleo de la aplicación.

- **Carpeta *"navigation"***

A continuación, se presenta el código correspondiente a cada uno de los archivos dedicados a gestionar la navegación entre las diferentes páginas de la aplicación.

  - ***AppNavigation.kt***

```
package com.i72pehej.cpuschedulerapp.navigation

import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.runtime.Composable
import com.google.accompanist.navigation.animation.AnimatedNavHost
import com.google.accompanist.navigation.animation.composable
import
com.google.accompanist.navigation.animation.rememberAnimatedNavController
import com.i72pehej.cpuschedulerapp.usecases.home.HomeScreen
import com.i72pehej.cpuschedulerapp.usecases.launch.SplashScreen

/**
 * @author Julen Perez Hernandez
 *
 * Fichero para establecer la navegacion entre las diferentes pantallas
 */
@OptIn(ExperimentalAnimationApi::class)
```

```
@Composable
fun AppNavigation(temaOscuro: Boolean, onActualizarTema: () -> Unit) {
    val navController = rememberAnimatedNavController()

    AnimatedNavHost(
        navController = navController,
        startDestination = AppScreens.SplashScreen.route
    ) {
        // Elemento composable para la Splash Screen
        composable(
            AppScreens.SplashScreen.route,
            // Llamada a la clase de animaciones personalizada para
modularizar
            exitTransition = AppNavigationAnimations.SplashAnimations.exit
        ) {
            // Llamada a la funcion que maneja el contenido de la pagina
            SplashScreen(navController)
        }

        // Elemento composable para la Home Screen
        composable(
            AppScreens.HomeScreen.route,
            enterTransition =
AppNavigationAnimations.BasicNavigateAnimation.enter,
            exitTransition =
AppNavigationAnimations.BasicNavigateAnimation.exit,
            popEnterTransition =
AppNavigationAnimations.BasicNavigateAnimation.popEnter,
            popExitTransition =
AppNavigationAnimations.BasicNavigateAnimation.popExit
        ) {
            // Llamada a la funcion que maneja el contenido de la pagina
            HomeScreen(
//                navController,
                temaOscuro,
                onActualizarTema
            )
        }
    }
}
```

- *AppNavigationAnimations.kt*

```
package com.i72pehej.cpuschedulerapp.navigation

import androidx.compose.animation.*
import androidx.compose.animation.core.LinearOutSlowInEasing
import androidx.compose.animation.core.tween
import androidx.compose.ui.unit.IntOffset
import androidx.navigation.NavBackStackEntry
import com.i72pehej.cpuschedulerapp.util.fadeAnimationSpeed
import com.i72pehej.cpuschedulerapp.util.slideAnimationSpeed

// Fichero para modularizar las animaciones de transicion de la navegacion
de la app

/**
 * @author Julen Perez Hernandez
 * App navigation animations
 *
 * Clase que modulariza las animaciones de transicion entre paginas de la
app
 *
 * @property enter Animacion de navegacion a la pantalla destino usando
navigate()
 * @property exit Animacion de salida de la pantalla actual cuando se
navega a otro destino
 * @property popEnter Animacion de la pantalla destino cuando se le da al
boton de volver (popBackStack()). Por defecto == enter
 * @property popExit Animacion de la pantalla actual que se va al pulsar el
boton de volver (popBackStack()). Por defecto == exit
 * @constructor Crea los objetos de las animaciones de navegacion que
utilizaran las distintas paginas
 */
sealed class AppNavigationAnimations(
    val enter: (AnimatedContentTransitionScope<NavBackStackEntry>.() ->
EnterTransition?)? = null,
    val exit: (AnimatedContentTransitionScope<NavBackStackEntry>.() ->
ExitTransition?)? = null,
    val popEnter: (AnimatedContentTransitionScope<NavBackStackEntry>.() ->
EnterTransition?)? = null,
    val popExit: (AnimatedContentTransitionScope<NavBackStackEntry>.() ->
ExitTransition?)? = null
```

```
) {
    // Animacion de la SplashScreen
    object SplashAnimations : AppNavigationAnimations(
        exit = {
            slideOut(
                animationSpec = tween(
                    durationMillis = slideAnimationSpeed,
                    easing = LinearOutSlowInEasing
                )
            ) { fullSize ->
                IntOffset(-fullSize.width / 4, -100)
            } + fadeOut()
        }
    )

    // Animaciones base para las transiciones entre pantallas normales
    object BasicNavigateAnimation : AppNavigationAnimations(
        enter = {
            slideInHorizontally(
                animationSpec = tween(
                    durationMillis = slideAnimationSpeed,
                    easing = LinearOutSlowInEasing
                )
            ) { fullWidth ->
                fullWidth / 3
            } + fadeIn(tween(fadeAnimationSpeed))
        },
        exit = { fadeOut(tween(fadeAnimationSpeed)) },
        popEnter = {
            expandHorizontally()
        },
        popExit = {
            fadeOut()
        }
    )
}
```

- *AppScreens.kt*

```
package com.i72pehej.cpuschedulerapp.navigation

// Fichero para definir las diferentes pantallas entre las que podemos
navegar

/**
 * @author Julen Perez Hernandez
 * App screens
 *
 * @property route Nombre de la ruta de la pagina de destino a la que se va
a navegar
 * @constructor Crea los objetos de las rutas de cada una de las paginas a
las que se puede navegar
 */
sealed class AppScreens(val route: String) {
    object SplashScreen : AppScreens("splash_screen")
    object HomeScreen : AppScreens("home_screen")
}
```

- *HomeTabs.kt*

```
package com.i72pehej.cpuschedulerapp.navigation

import androidx.compose.foundation.layout.Column
import androidx.compose.material.Icon
import androidx.compose.material.LeadingIconTab
import androidx.compose.material.TabRow
import androidx.compose.material.TabRowDefaults
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Equalizer
import androidx.compose.material.icons.filled.Home
import androidx.compose.material.icons.filled.LowPriority
import androidx.compose.material.icons.filled.Toc
import androidx.compose.runtime.Composable
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
import com.google.accompanist.pager.ExperimentalPagerApi
```

```
import com.google.accompanist.pager.HorizontalPager
import com.google.accompanist.pager.PagerState
import com.google.accompanist.pager.pagerTabIndicatorOffset
import com.google.accompanist.pager.rememberPagerState
import com.i72pehej.cpuschedulerapp.navigation.HomeTabs.*
import com.i72pehej.cpuschedulerapp.usecases.home.ContenidoHome
import com.i72pehej.cpuschedulerapp.usecases.results.GraphsScreen
import com.i72pehej.cpuschedulerapp.usecases.results.QueuesScreen
import com.i72pehej.cpuschedulerapp.usecases.results.ResultsScreen
import com.i72pehej.cpuschedulerapp.util.infoResultadosGlobal
import com.i72pehej.cpuschedulerapp.util.siguienteSeleccionado
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.launch

/**
 * @author Julen Perez Hernandez
 */

// Alias para la funcion correspondiente a los elementos de cada pantalla
typealias NuevoTab = @Composable () -> Unit

/**
 * Constructor para el control de los tabs de la pantalla Home
 *
 * @constructor Crea una clase para la gestion de los tabs
 */
sealed class HomeTabs(
    var icono: ImageVector,
    var titulo: String,
    var pantalla: NuevoTab
) {
    object TabInicio : HomeTabs(icono = Icons.Default.Home, titulo =
"Inicio", { ContenidoHome() })

    object TabResultados : HomeTabs(icono = Icons.Default.Toc, titulo =
"Resultado", { if (infoResultadosGlobal.isNotEmpty()) ResultsScreen() })

    object TabGraficos : HomeTabs(icono = Icons.Filled.Equalizer, titulo =
"Gráficos", { if (infoResultadosGlobal.isNotEmpty()) GraphsScreen() })

    object TabColas : HomeTabs(icono = Icons.Filled.LowPriority, titulo =
```

```
"Colas", { if (infoResultadosGlobal.isNotEmpty()) QueuesScreen() })
}

/**
 *
================================================================================
================
 */

@OptIn(ExperimentalPagerApi::class)
@Composable
fun CrearTabs() {
    // Lista para almacenar los elementos de los tabs
    val tabs = listOf(TabInicio, TabResultados, TabGraficos, TabColas)

    // Control del paginador que contiene los tabs
    val pagerState = rememberPagerState()

    // Contenido del paginador
    Column {
        Tabs(tabs, pagerState)
        Tabs_content(tabs, pagerState)
    }
}

/**
 *
================================================================================
================
 */

/**
 * Control del cambio de estado del pager state
 *
 * @param tabs Lista de los elementos a colocar en cada tab
 * @param pagerState Controlador del estado de los tabs
 */
@OptIn(ExperimentalPagerApi::class)
@Composable
fun Tabs_content(tabs: List<HomeTabs>, pagerState: PagerState) {
    // Paginador que muestra los tabs horizontalmente y permite desplazarse
```

```
entre ellos
    HorizontalPager(state = pagerState, count = tabs.size) { page: Int ->
        tabs[page].pantalla()
    }
}

/**
 *
================================================================================
================
 */

/**
 * Funcion para crear el contenedor de los tabs
 *
 * @param tabs Lista de los elementos a colocar en cada tab
 * @param pagerState Controlador del estado de los tabs
 */
@OptIn(ExperimentalPagerApi::class)
@Composable
fun Tabs(tabs: List<HomeTabs>, pagerState: PagerState) {
    // Control de contexto
    val scope = rememberCoroutineScope()

    // Fila de los elementos del paginador
    TabRow(selectedTabIndex = pagerState.currentPage, indicator = {
tabPositions ->
        TabRowDefaults.Indicator(
            modifier = Modifier.pagerTabIndicatorOffset(
                pagerState = pagerState,
                tabPositions = tabPositions
            )
        )
    }) {
        // Creacion de cada elemento
        tabs.forEachIndexed { index, homeTabs ->
            LeadingIconTab(
                selected = pagerState.currentPage == index,
                // Control del click para que se anime la transicion a la
pagina indicada
                onClick = { scope.launch {
```

```
pagerState.animateScrollToPage(index) } },
                icon = {
                    Icon(
                        imageVector = homeTabs.icono,
                        contentDescription = "Icono del tab asociado"
                    )
                },
                text = {
                    Text(
                        text = homeTabs.titulo,
                        fontSize = 12.5.sp,
                        fontWeight = FontWeight.Bold
                    )
                }
            )
        }
    }

    // Gestion del cambio de pagina de Home a Results
    val coroutineScope = rememberCoroutineScope()

    fun cambioPagina(coroutineScope: CoroutineScope, pagerState:
PagerState) {
        coroutineScope.launch { pagerState.animateScrollToPage(1) }
    }

    if (siguienteSeleccionado.value) {
        cambioPagina(coroutineScope, pagerState)
        siguienteSeleccionado.value = false
    }
}
```

- **Carpeta "*ui/theme*"**

A continuación, se presenta el código correspondiente a los ficheros más destacables de configuración que definen los colores generales, formas predefinidas y otros elementos visuales relacionados con los temas de la aplicación.

- ***Color.kt***

```
package com.i72pehej.cpuschedulerapp.ui.theme

import androidx.compose.ui.graphics.Color

val Purple200 = Color(0xFFBB86FC)
val Purple500 = Color(0xFF6200EE)
val Purple700 = Color(0xFF3700B3)
val Teal200 = Color(0xFF03DAC5)

// Colores de la guia de estilo web de la UCO

// Colores base de la marca
val Amarillo_base = Color(0xFFDB912F)
val Azul_base = Color(0xFF221C35)
val Rojo_base = Color(0xFFA41E34)

// Colores complementarios al grupo base
val Amarillo_com_1 = Color(0xFFD6A305)
val Amarillo_com_2 = Color(0xFFA37C13)

val Azul_com_1 = Color(0xFF7a63b8)
val Azul_com_4 = Color(0xFFc9d1e4)
val Azul_com_5 = Color(0xFFeaebfb)
val Azul_com_2 = Color(0xFF352663)
val Azul_com_3 = Color(0xFF322b85)

val Rojo_com_1 = Color(0xFFE73057)
val Rojo_com_2 = Color(0xFF691320)

// Colores complementarios no derivados de los base
val Amarillo_deriv_1 = Color(0xFF565220)
val Amarillo_deriv_2 = Color(0xFFA39A39)
val Amarillo_deriv_3 = Color(0xFFE3D553)
```

```
val Verde_deriv_1 = Color(0xFF206F30)
val Verde_deriv_2 = Color(0xFF259624)
val Verde_deriv_3 = Color(0xFF5CB244)
```

- *Theme.kt*

```
package com.i72pehej.cpuschedulerapp.ui.theme

import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material.MaterialTheme
import androidx.compose.material.darkColors
import androidx.compose.material.lightColors
import androidx.compose.runtime.Composable

private val DarkColorPalette = darkColors(
//    primary = Purple200,
//    primaryVariant = Purple700,
//    secondary = Teal200

    primary = Azul_com_1,
    primaryVariant = Azul_base,
    secondary = Azul_com_4,
    secondaryVariant = Azul_com_2,
)

private val LightColorPalette = lightColors(
//    primary = Purple500,
//    primaryVariant = Purple700,
//    secondary = Teal200

    primary = Azul_base,
    primaryVariant = Azul_com_5,
    secondary = Azul_com_2,
    secondaryVariant = Azul_com_4,

    /* Other default colors to override
    onPrimary = Color.White,
    onSecondary = Color.Black,
    */
)

@Composable
```

```
fun CpuSchedulerAppTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit
) {
    val colors = if (darkTheme) {
        DarkColorPalette
    } else {
        LightColorPalette
    }

    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

- **Carpeta "*usecases*"**

A continuación, se presenta el código correspondiente a cada caso de uso específico de la aplicación. Esta carpeta se encuentra dividida por subcarpetas en los diferentes grupos que engloban cada caso de uso o las diversas funcionalidades asociadas.

  - **Subcarpeta "*algorithms*"**

En esta carpeta se engloban todos los ficheros para los algoritmos implementados en la aplicación. En caso de que futuros desarrolladores deseen incrementar las funcionalidades de la herramienta agregando nuevos algoritmos, será en esta carpeta donde se deberá crear el nuevo fichero.

    - ***FifoAlgorithm.kt***

```
package com.i72pehej.cpuschedulerapp.usecases.algorithms

import androidx.compose.runtime.snapshots.SnapshotStateList
import com.i72pehej.cpuschedulerapp.util.classes.InfoGraficoEstados
import com.i72pehej.cpuschedulerapp.util.classes.Proceso
import
com.i72pehej.cpuschedulerapp.util.classes.Proceso.EstadoDeProceso.BLOQUEADO
import
com.i72pehej.cpuschedulerapp.util.classes.Proceso.EstadoDeProceso.COMPLETAD
O
```

```
import
com.i72pehej.cpuschedulerapp.util.classes.Proceso.EstadoDeProceso.EJECUCION
import
com.i72pehej.cpuschedulerapp.util.classes.Proceso.EstadoDeProceso.LISTO
import com.i72pehej.cpuschedulerapp.util.classes.crearProceso
import com.i72pehej.cpuschedulerapp.util.classes.ordenarLlegadaProcesos
import com.i72pehej.cpuschedulerapp.util.listaDeProcesosGlobal

/**
 * @author Julen Perez Hernandez
 */

/**
 * ====================================================================
 */

/**
 * Funcion para implementar el algoritmo FIFO considerando estados
 *
 * @param listaProcesos Listado de prodesos con los que realizar el
algoritmo FIFO
 *
 * @return Devuelve el listado de estados en cada momento para cada proceso
de la lista
 */
fun algoritmoFifo(listaProcesos: SnapshotStateList<Proceso>):
MutableList<InfoGraficoEstados> {
    // Ordenar la lista de procesos por tiempo de llegada
    ordenarLlegadaProcesos(listaProcesos)

    // Funcion para realizar una copia independiente del listado de
procesos
    fun copiarLista(listaProcesosOriginal: SnapshotStateList<Proceso>):
MutableList<Proceso> {
        val nuevaLista = mutableListOf<Proceso>()

        listaProcesosOriginal.forEach { elemento ->
            nuevaLista.add(
                crearProceso(
                    nombre = elemento.getNombre(),
                    tiempoLlegada = elemento.getLlegada(),
```

```
                    duracion = elemento.getDuracion(),
                    estado = elemento.getEstado(),
                    tiempoEntrada = elemento.getTiempoEntrada(),
                    tiempoSalida = elemento.getTiempoSalida()
                )
            )
        }

        return nuevaLista
    }

    // Variable para almacenar el progreso de los ESTADOS de cada proceso
durante el algoritmo
    val infoEstados = mutableListOf<InfoGraficoEstados>()

    // Creacion de la cola de procesos LISTOS
    val colaDeListos = copiarLista(listaProcesos)

    // Variable para almacenar el avance del tiempo con cada proceso
    var momentoActual = colaDeListos.first().getLlegada()

    // Consideramos que se deba completar el ultimo proceso como condicion
de parada del bucle
    while (colaDeListos.isNotEmpty()) {
        // Variable que almacena el primer elemento de la cola
        val cabezaDeCola = colaDeListos.first()

        // Comprobamos que el proceso tenga evento de E/S
        if ((cabezaDeCola.getTiempoEntrada() > 0) &&
(cabezaDeCola.getTiempoEntrada() == momentoActual)) {
            // Bucle para almacenar los tiempos de bloqueo por E/S del
proceso
            for (tiempos in 0 until cabezaDeCola.getTiempoDeEsperaES()) {
                // Guardamos el estado BLOQUEADO
                infoEstados.add(InfoGraficoEstados(nombre =
cabezaDeCola.getNombre(), estado = BLOQUEADO, momento = momentoActual +
tiempos))

                // Incrementamos el tiempo de espera local para el control
de las colas
                val index = listaDeProcesosGlobal.indexOf(cabezaDeCola)
```

```
                if (index > 0)
listaDeProcesosGlobal[index].setTiempoEsperaLocal(cabezaDeCola.getTiempoEsp
eraLocal() + 1)
            }

            // Actualizar la llegada del proceso de vuelta del estado de
BLOQUEO == salida del evento de E/S para retomarlo desde ese punto
            cabezaDeCola.setLlegada(cabezaDeCola.getTiempoSalida())

            // Movemos el proceso al final de la lista de LISTOS
            colaDeListos.add(cabezaDeCola)
            colaDeListos.removeAt(0)

            // Reordenamos la cola por tiempo de llegada para considerar
que el evento E/S termine antes de que haya llegado el siguiente proceso a
la cola
            colaDeListos.sortBy { it.getLlegada() }

            // Actualizamos el momentoActual a la llegada del siguiente
proceso (-1 para considerar el aumento del bucle)
            momentoActual = colaDeListos.first().getLlegada() - 1
        }
        // Si no hay evento de bloqueo de proceso...
        else {
            // Comprobamos que le quede tiempo restante al proceso
            if (cabezaDeCola.getTiempoRestante() > 0) {
                // Buscamos algun proceso que este en ejecucion en este
momento
                val procesoEnEjecucion = infoEstados.any { ((it.getEstado()
== EJECUCION) && (it.getMomento() == momentoActual)) }

                // Si NO hay ningun proceso en EJECUCION...
                if (!procesoEnEjecucion) {
                    // Pasamos a EJECUCION
                    infoEstados.add(InfoGraficoEstados(nombre =
cabezaDeCola.getNombre(), estado = EJECUCION, momento = momentoActual))

                    // Restamos una unidad al tiempoRestante

cabezaDeCola.setTiempoRestante(cabezaDeCola.getTiempoRestante() - 1)
                }
```

```
                // Si hay otro proceso ejecutandose...
                else {
                    // Pasamos a LISTO == ESPERA
                    infoEstados.add(InfoGraficoEstados(nombre =
cabezaDeCola.getNombre(), estado = LISTO, momento = momentoActual))

                    // Incrementamos el tiempo de espera local para el
control de las colas
                    val index = listaDeProcesosGlobal.indexOf(cabezaDeCola)
                    if (index > 0)
listaDeProcesosGlobal[index].setTiempoEsperaLocal(cabezaDeCola.getTiempoEsp
eraLocal() + 1)
                }
            }
            // Si no le queda tiempo restante
            else {
                // Pasamos a COMPLETADO
                infoEstados.add(InfoGraficoEstados(nombre =
cabezaDeCola.getNombre(), estado = COMPLETADO, momento = momentoActual))

                // Actualizamos el estado final del proceso
                cabezaDeCola.setEstado(COMPLETADO)

                // Eliminamos el proceso de la cola
                colaDeListos.removeAt(0)

                // Actualizamos el momentoActual a la llegada del siguiente
proceso (-1 para considerar el aumento del bucle)
                val llegadaCabeza = if (colaDeListos.firstOrNull() == null)
0 else colaDeListos.first().getLlegada() - 1
                momentoActual = llegadaCabeza
            }
        }

        // Avanzamos el tiempo
        momentoActual++
    }

    // Almacenamos la variable de informacion de los tiempos
    return infoEstados
}
```

- ***RoundRobinAlgorithm.kt***

Este algoritmo se encuentra implementado como plantilla ya que únicamente se han implementado ciertas funcionalidades relativas a él, pero no el propio algoritmo en sí.

```
package com.i72pehej.cpuschedulerapp.usecases.algorithms

/**
 * @author Julen Perez Hernandez
 */

/**
 * ====================================================================
 */

// TODO -> FICHERO PLANTILLA PARA EL CORRECTO FUNCIONAMIENTO DE
FUNCIONALIDADES YA AGREGADAS PARA ESTE ALGORITMO

/**
 * Funcion para implementar el algoritmo RoundRobin considerando estados
 *
 * @param quantum Quantum seleccionado para ejecutar el algoritmo de RR
 *
 * @return Devuelve un listado con el historico de cambios de estados de
los procesos
 */
fun roundRobin(quantum: Int): MutableList<InfoGraficoEstados> {}
```

- **Subcarpeta "*common*"**

En esta carpeta se encuentran los ficheros correspondientes a diferentes ampliaciones o modificaciones de funcionalidades de elementos ya implementados por defecto en Kotlin.

- ***CommonButton.kt***

```
package com.i72pehej.cpuschedulerapp.usecases.common

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Button
```

```
import androidx.compose.material.ButtonDefaults
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.i72pehej.cpuschedulerapp.util.buttonBorderWith
import com.i72pehej.cpuschedulerapp.util.buttonCornerRadius
import com.i72pehej.cpuschedulerapp.util.buttonDefaultPadding

/**
 * @author Julen Perez Hernandez
 * Boton base para las acciones basicas de la aplicacion
 *
 * @param text Texto del boton
 * @param onClick Accion al hacer click en el boton
 * @param modifier Modificadores de apariencia
 */
@Composable
fun CommonRoundedButton(
    text: String,
    onClick: () -> Unit,
    isEnabled: Boolean,
    modifier: Modifier = Modifier
) {
    Button(
        onClick = onClick,
        enabled = isEnabled,
        modifier = modifier
            .fillMaxWidth()
            .padding(buttonDefaultPadding.dp),
        shape = RoundedCornerShape(buttonCornerRadius),
        border = BorderStroke(buttonBorderWith.dp,
MaterialTheme.colors.primary),
        colors = ButtonDefaults.buttonColors(
            backgroundColor = MaterialTheme.colors.surface,
            contentColor = MaterialTheme.colors.onSurface
        )
    ) {
        Text(text)
```

```
    }
}
```

- ***CommonScaffold.kt***

```
package com.i72pehej.cpuschedulerapp.usecases.common

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.material.DropdownMenu
import androidx.compose.material.DropdownMenuItem
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.Scaffold
import androidx.compose.material.ScaffoldState
import androidx.compose.material.TopAppBar
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.DeleteSweep
import androidx.compose.material.icons.filled.Settings
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import com.i72pehej.cpuschedulerapp.util.appIconUCOSolo
import com.i72pehej.cpuschedulerapp.util.extensions.EliminarProcesoTabla
import com.i72pehej.cpuschedulerapp.util.extensions.LimpiarProcesos
import com.i72pehej.cpuschedulerapp.util.extensions.ThemeSwitcher
import com.i72pehej.cpuschedulerapp.util.listaDeProcesosGlobal

/**
 * @author Julen Perez Hernandez
```

```
 */

/**
 * ====================================================================
 */

/**
 * Common scaffold para toda la app
 *
 * @param content Contenido que mostrar en pantalla
 */
@Composable
fun CommonScaffold(
    temaOscuro: Boolean,
    onActualizarTema: () -> Unit,
    scaffoldState: ScaffoldState,
    content: @Composable (PaddingValues) -> Unit,
) {
    Scaffold(
        scaffoldState = scaffoldState,
        topBar = {
            CommonTopAppBar(
                temaOscuro,
                onActualizarTema
            )
        },
        content = content,
    )
}

/**
 * ====================================================================
 */

/**
 * Funcion que crea la top bar comun de la app
 *
 * @param temaOscuro Control para el switch del tema de la app
 * @param onActualizarTema Funcion para actualizar el tema
 */
@Composable
```

```
fun CommonTopAppBar(
    temaOscuro: Boolean,
    onActualizarTema: () -> Unit
) {
    // Control del menu de ajustes desplegable
    var verMenuAjustes by remember { mutableStateOf(false) }

    // Control del popup de alerta para editar los procesos
    val mostrarPopupAll = remember { mutableStateOf(false) }
    val mostrarPopupOne = remember { mutableStateOf(false) }

    // Barra superior de la pantalla
    TopAppBar(
        modifier = Modifier.height(30.dp),
        title = {
            Row {
                Icon(
                    painter = painterResource(id = appIconUCOSolo),
                    contentDescription = "Icono principal de la App",
                    modifier = Modifier
                        .align(alignment = Alignment.CenterVertically)
                        .size(50.dp),
                )
            }
        },
        elevation = 1.dp,
        // Menu desplegable para ajustes basicos
        actions = {
            // Icono para limpiar la tabla de procesos
            IconButton(onClick = { mostrarPopupOne.value =
listaDeProcesosGlobal.isNotEmpty() }) { Icon(imageVector =
Icons.Filled.Delete, contentDescription = "Boton de Eliminar un proceso") }

            // Llamada a la funcion para eliminar un proceso
            EliminarProcesoTabla(showDialog = mostrarPopupOne)

            // Icono para limpiar la tabla de procesos
            IconButton(onClick = { mostrarPopupAll.value =
listaDeProcesosGlobal.isNotEmpty() }) { Icon(imageVector =
Icons.Filled.DeleteSweep, contentDescription = "Boton de Limpiar Tabla") }
```

```
            // Llamada a la funcion de limpieza de procesos
            LimpiarProcesos(mostrarPopupAll)

            // Icono de ajustes que cambia el estado del menu
            IconButton(onClick = { verMenuAjustes = !verMenuAjustes }) {
Icon(imageVector = Icons.Filled.Settings, contentDescription = "Boton de
Ajustes") }

            // Menu desplegable
            DropdownMenu(
                expanded = verMenuAjustes,
                onDismissRequest = { verMenuAjustes = false }
            ) {
                DropdownMenuItem(
                    onClick = {},
                    modifier = Modifier
                        .height(25.dp)
                        .width(90.dp)
                ) {
                    Column(
                        modifier = Modifier.fillMaxWidth(),
                        horizontalAlignment = Alignment.CenterHorizontally
                    ) {
                        ThemeSwitcher(darkTheme = temaOscuro, onClick =
onActualizarTema)
                    }
                }
            }
        }
    )
}
```

- **Subcarpeta "*home*"**

En esta subcarpeta se encuentra el fichero correspondiente a la pantalla de inicio. Ha sido creada para considerar cualquier funcionalidad relacionada a esta pantalla.

- *HomeScreen.kt*

```
package com.i72pehej.cpuschedulerapp.usecases.home

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.Button
import androidx.compose.material.Checkbox
import androidx.compose.material.DropdownMenuItem
import androidx.compose.material.ExperimentalMaterialApi
import androidx.compose.material.ExposedDropdownMenuBox
import androidx.compose.material.ExposedDropdownMenuDefaults
import androidx.compose.material.Icon
import
androidx.compose.material.LocalMinimumInteractiveComponentEnforcement
import androidx.compose.material.MaterialTheme
import androidx.compose.material.OutlinedTextField
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.HourglassBottom
import androidx.compose.material.icons.filled.Memory
import androidx.compose.material.icons.filled.Schedule
import androidx.compose.material.icons.filled.Update
import androidx.compose.material.rememberScaffoldState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.CompositionLocalProvider
```

```
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment.Companion.CenterHorizontally
import androidx.compose.ui.ExperimentalComposeUiApi
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.alpha
import androidx.compose.ui.focus.FocusDirection
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.platform.LocalSoftwareKeyboardController
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardCapitalization
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.text.isDigitsOnly
import com.i72pehej.cpuschedulerapp.R
import com.i72pehej.cpuschedulerapp.navigation.CrearTabs
import com.i72pehej.cpuschedulerapp.usecases.algorithms.algoritmoFifo
import com.i72pehej.cpuschedulerapp.usecases.common.CommonRoundedButton
import com.i72pehej.cpuschedulerapp.usecases.common.CommonScaffold
import com.i72pehej.cpuschedulerapp.util.anchuraFormularioNombres
import com.i72pehej.cpuschedulerapp.util.anchuraFormularioTiempos
import com.i72pehej.cpuschedulerapp.util.classes.Proceso
import com.i72pehej.cpuschedulerapp.util.classes.crearProceso
import com.i72pehej.cpuschedulerapp.util.extensions.ConfirmacionBackPress
import com.i72pehej.cpuschedulerapp.util.extensions.TablaProcesos
import com.i72pehej.cpuschedulerapp.util.infoResultadosGlobal
import com.i72pehej.cpuschedulerapp.util.listaDeProcesosGlobal
import com.i72pehej.cpuschedulerapp.util.selectorAlgoritmo
import com.i72pehej.cpuschedulerapp.util.siguienteSeleccionado
import com.i72pehej.cpuschedulerapp.util.tiempoQuantum

/**
 * @author Julen Perez Hernandez
 */

/**
```

```
 *
=============================================================================
================
 */

/**
 * Pantalla inicial en la que comenzar la navegacion por la app
 *
// * @param navController Control de navegacion
 * @param temaOscuro Control para el switch del tema de la app
 * @param onActualizarTema Funcion para actualizar el tema
 */
@Composable
fun HomeScreen(
//    navController: NavHostController,
    temaOscuro: Boolean,
    onActualizarTema: () -> Unit
) {
    // Inicializacion del control de salida por error al pulsar back
    ConfirmacionBackPress()

    // Variable para guardar el estado del contenido
    val scaffoldState = rememberScaffoldState()

    // Disposicion principal de la pantalla
    CommonScaffold(
        temaOscuro = temaOscuro,
        onActualizarTema = onActualizarTema,
        scaffoldState = scaffoldState,
        content = { CrearTabs() }
    )
}

/**
 *
=============================================================================
================
 */

/**
 * Contenido de la pagina para introducir en el scaffold
```

```
 */
@Composable
fun ContenidoHome() {
    // Contenedor padre de los elementos a mostrar en la pagina
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(8.dp)
    ) {
        // Contenedor de los elementos principales
        Row(
            modifier = Modifier
                .fillMaxHeight()
                .padding(start = 8.dp, bottom = 8.dp, end = 8.dp)
        ) {
            // Creamos el formulario de ingreso de procesos, y le pasamos
una función que se llamará cuando se agregue un proceso
            FormularioProceso { proceso ->
listaDeProcesosGlobal.add(proceso) }

            // Separador horizontal para la tabla
            Spacer(modifier = Modifier.width(8.dp))

            // Creamos la tabla de procesos, y le pasamos la lista de
procesos
            TablaProcesos(procesos = listaDeProcesosGlobal)
        }
    }
}

/**
 *
==========================================================================
================
 */

/**
 * Llamada a la ejecucion de cada algoritmo dependiendo de la opcion
seleccionada en el formulario
 */
fun llamarAlgoritmo() {
```

```
    // Limpiar posibles resultados anteriores
    infoResultadosGlobal.clear()

    // Selector de algoritmo
    when (selectorAlgoritmo) {
        // FIFO
        0 -> {
            // Reseteo de los valores basicos de los procesos
            listaDeProcesosGlobal.forEach { it.reset() }

            infoResultadosGlobal = algoritmoFifo(listaDeProcesosGlobal)
        }
        // RoundRobin
        1 -> {
            // TODO -> IMPLEMENTAR CORRECTAMENTE EL ALGORITMO
//              roundRobin(tiempoQuantum.toInt())
        }
    }
}

/**
 *
================================================================================
================
 */

/**
 * Formulario de ingreso de procesos
 *
 * @param onSubmit Se encarga de agregar el proceso a la lista de procesos
y limpiar el formulario.
 */
@OptIn(ExperimentalComposeUiApi::class, ExperimentalMaterialApi::class)
@Composable
fun FormularioProceso(onSubmit: (Proceso) -> Unit) {
    // Variables para el menu para seleccionar el metodo a utilizar
    val algoritmosImplementados = listOf("FIFO", "RoundRobin")
    var expandir by remember { mutableStateOf(value = false) }
    var algoritmoSeleccionado by remember { mutableStateOf(value =
algoritmosImplementados[selectorAlgoritmo]) }
```

```
    // Control de estado para agregar los procesos en agregarProceso()
    var procesoAgregado by remember { mutableStateOf(false) }

    // Control de estado de ejecucion de funcion agregarProceso()
    var quantumSeleccionado by remember { mutableStateOf(false) }

    // Control para ocultar el teclado y perder el foco del formulario al
terminar de agregar cada proceso
    val keyboardController = LocalSoftwareKeyboardController.current
    val focusManager = LocalFocusManager.current

    // Definimos tres variables para almacenar los datos del proceso que
esta siendo ingresado
    var nombre by remember { mutableStateOf("") }
    var tiempoLlegada by remember { mutableStateOf("") }
    var duracion by remember { mutableStateOf("") }
    var quantum by remember { mutableStateOf(tiempoQuantum) }

    var entradaSalidaInicio by remember { mutableStateOf("") }
    var entradaSalidaFin by remember { mutableStateOf("") }

    // Estados para almacenar los errores del formulario
    var errorFormulario by remember { mutableStateOf("") }
    var errorNombre by remember { mutableStateOf(false) }
    var errorLlegada by remember { mutableStateOf(false) }
    var errorDuracion by remember { mutableStateOf(false) }

    var errorFormularioQuantum by remember { mutableStateOf("") }
    var errorQuantum by remember { mutableStateOf(false) }

    var errorFormularioES by remember { mutableStateOf("") }
    var errorESinicio by remember { mutableStateOf(false) }
    var errorESfin by remember { mutableStateOf(false) }

    // Variable para controlar visibilidad del boton Siguiente
    var siguienteEnabled by remember { mutableStateOf(false) }

    // Control de visibilidad de campo de quantum
    val quantumEnabled: Boolean
    val quantumVisibilityAlpha: Float
```

```
    if (selectorAlgoritmo == 1) {
        quantumEnabled = true
        quantumVisibilityAlpha = 1f

        // Comprobar si se cumplen las condiciones para habilitar el boton
        siguienteEnabled = quantum.isNotBlank() &&
listaDeProcesosGlobal.isNotEmpty() && errorFormularioES.isBlank()
    } else {
        siguienteEnabled = listaDeProcesosGlobal.isNotEmpty()
        quantum = ""
        quantumEnabled = false
        quantumVisibilityAlpha = 0f
    }

    // Variables de E/S
    var checkboxMarcado by remember { mutableStateOf(false) }
    var visibleES by remember { mutableStateOf(if (checkboxMarcado) 1f else
0f) }

    // Funcion interna para controlar que se haya seleccionado un quantum
    @Composable
    fun comprobarQuantum() {
        // Control de errores solo para RR
        if (selectorAlgoritmo == 1) {
            errorQuantum = when {
                quantum.isBlank() -> true
                !quantum.isDigitsOnly() -> true
                else -> false
            }

            // Creacion de mensaje de error en el campo del quantum para RR
            errorFormularioQuantum = when {
                quantum.isBlank() ->
stringResource(R.string.error_quantum_blank)
                !quantum.isDigitsOnly() ->
stringResource(R.string.error_quantum_digit)
                else -> {
                    ""
                }
            }
        }
```

```
    }

    // Funcion interna para controlar que se hayan seleccionado los tiempos
de E/S
    @Composable
    fun comprobarEntradaSalida(): String {
        // Control de errores solo cuando se selecciona E/S
        if (checkboxMarcado) {
            errorESinicio = when {
                entradaSalidaInicio.isBlank() -> true
                !entradaSalidaInicio.isDigitsOnly() -> true
                else -> false
            }

            errorESfin = when {
                entradaSalidaFin.isBlank() -> true
                !entradaSalidaFin.isDigitsOnly() -> true
                entradaSalidaInicio.toInt() > entradaSalidaFin.toInt() ->
true
                else -> false
            }

            // Creacion de mensaje de error para los campos de E/S
            errorFormularioES = when {
                entradaSalidaInicio.isBlank() ->
stringResource(R.string.error_E_S)
                !entradaSalidaInicio.isDigitsOnly() ->
stringResource(R.string.error_E_S)

                entradaSalidaFin.isBlank() ->
stringResource(R.string.error_E_S)
                !entradaSalidaFin.isDigitsOnly() ->
stringResource(R.string.error_E_S)
                entradaSalidaInicio.toInt() > entradaSalidaFin.toInt() ->
stringResource(R.string.error_EmenorS)

                else -> {
                    ""
                }
            }
        }
```

```
        return errorFormularioES
    }

    // Función para validar los campos del formulario y agregar un proceso
a la lista de procesos ingresados
    @Composable
    fun agregarProceso() {
        // Validar los campos del formulario (Pone en rojo el campo
visualmente)
        errorNombre = when {
            nombre.isBlank() -> true
            listaDeProcesosGlobal.any { it.getNombre() == nombre } -> true
            else -> false
        }

        errorLlegada = when {
            tiempoLlegada.isBlank() -> true
            !tiempoLlegada.isDigitsOnly() -> true
            else -> false
        }

        errorDuracion = when {
            duracion.isBlank() -> true
            !duracion.isDigitsOnly() -> true
            else -> false
        }

        // Comprobacion de formulario completo correctamente (Agrega el
texto de error)
        errorFormulario = when {
            listaDeProcesosGlobal.any { it.getNombre() == nombre } ->
stringResource(id = R.string.error_nombre_repetido)

            nombre.isBlank() -> stringResource(R.string.error_nombre)

            tiempoLlegada.isBlank() ->
stringResource(R.string.error_llegada_blank)
            !tiempoLlegada.isDigitsOnly() ->
stringResource(R.string.error_llegada_digit)
```

```
            duracion.isBlank() ->
stringResource(R.string.error_duracion_blank)
            !duracion.isDigitsOnly() ->
stringResource(R.string.error_duracion_digit)

            comprobarEntradaSalida() != "" -> errorFormularioES

            // Si los campos son válidos, agregamos un nuevo proceso
            else -> {
                onSubmit(
                    // Crea un nuevo proceso
                    if (checkboxMarcado) {
                        crearProceso(
                            nombre = nombre,
                            tiempoLlegada = tiempoLlegada.toInt(),
                            duracion = duracion.toInt(),
                            estado = Proceso.EstadoDeProceso.LISTO,
                            tiempoEntrada = entradaSalidaInicio.toInt(),
                            tiempoSalida = entradaSalidaFin.toInt()
                        )
                    } else {
                        crearProceso(
                            nombre = nombre,
                            tiempoLlegada = tiempoLlegada.toInt(),
                            duracion = duracion.toInt(),
                            estado = Proceso.EstadoDeProceso.LISTO
                        )
                    }
                )

                // Limpiamos los campos del formulario
                nombre = ""
                tiempoLlegada = ""
                duracion = ""
                entradaSalidaInicio = ""
                entradaSalidaFin = ""

                // Reseteamos las variables de control de E/S
                errorESinicio = false
                errorESfin = false
                checkboxMarcado = false
```

```
                visibleES = 0f

                // Limpiamos el valor de la variable del error
                ""
            }
        }
    }

    // INICIO DEL CONTENIDO VISUAL DEL FORMULARIO

    // Contenedor para la primera columna de campos del formulario
    Column {
        Spacer(modifier = Modifier.height(8.dp))
        // Menu desplegable para seleccion de algoritmo
        ExposedDropdownMenuBox(
            modifier = Modifier.width(anchuraFormularioNombres.dp),
            expanded = expandir,
            onExpandedChange = { expandir = it }
        ) {
            TextField(
                readOnly = true,
                value = algoritmoSeleccionado,
                onValueChange = { },
                label = { Text("Método", fontSize = 12.sp) },
                trailingIcon = {
ExposedDropdownMenuDefaults.TrailingIcon(expanded = expandir) },
                colors = ExposedDropdownMenuDefaults.textFieldColors()
            )
            ExposedDropdownMenu(
                expanded = expandir,
                onDismissRequest = { expandir = false }
            ) {
                algoritmosImplementados.forEachIndexed { posicion,
opcionSeleccionada ->
                    DropdownMenuItem(
                        onClick = {
                            // Se selecciona el algoritmo
                            algoritmoSeleccionado = opcionSeleccionada
                            expandir = false

                            // Guardado de la opcion seleccionada
```

```
                            selectorAlgoritmo = posicion

                            // En caso de no ser necesario, se limpia el
control de errores del quantum
                            if (posicion != 1 /*RR*/)
errorFormularioQuantum = ""
                        }
                    ) { Text(text = opcionSeleccionada) }
                }
            }
        }

        // Creamos el campo de texto para ingresar el nombre del proceso
        OutlinedTextField(
            value = nombre,
            onValueChange = {
                // Control de cantidad de caracteres
                if (it.length <= 3) nombre = it
            },
            label = {
                Text(text = stringResource(id =
R.string.formulario_nombre))
                // Mensaje visual para que el usuario conozca la cantidad
de caracteres permitidos
                Text(
                    text = "${nombre.length} / 3",
                    textAlign = TextAlign.Right,
                    modifier = Modifier.fillMaxWidth()
                )
            },
            keyboardOptions = KeyboardOptions(capitalization =
KeyboardCapitalization.Sentences),   // Primera letra en mayuscula
            keyboardActions = KeyboardActions(onDone = {
focusManager.moveFocus(FocusDirection.Right) }),
            modifier = Modifier.width(anchuraFormularioNombres.dp),
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Memory,
                    contentDescription = "Icono Nombre Proceso"
                )
            },
```

```
            singleLine = true,
            isError = errorNombre
        )
        // Row para agregar un evento de E/S al proceso que se va a agregar
a la lista
        Row(modifier = Modifier.width(anchuraFormularioNombres.dp)) {
            // Checkbox para cargar los campos de E/S
            // Se ha eliminado el padding por defecto para poder
personalizarlo

CompositionLocalProvider(LocalMinimumInteractiveComponentEnforcement
provides false) {
                Checkbox(
                    checked = checkboxMarcado,
                    onCheckedChange = {
                        checkboxMarcado = it
                        visibleES = if (checkboxMarcado) 1f else 0f
                        errorFormularioES = ""
                    },
                    modifier = Modifier.padding(top = 8.dp, end = 8.dp)
                )

                // Leyenda para indicar el funcionamiento del checkbox
                if (!checkboxMarcado) {
                    Text(text = "E/S", modifier = Modifier.padding(top =
8.dp), fontWeight = FontWeight.Bold)
                }
            }

            // Campo para INICIO de bloqueo de proceso por E/S
            OutlinedTextField(
                value = entradaSalidaInicio,
                onValueChange = {
                    // Control de cantidad de caracteres a 2
                    if (it.matches("^([0-9][0-9]?|)$".toRegex()))
entradaSalidaInicio = it
                },
                keyboardOptions = KeyboardOptions.Default.copy(keyboardType
= KeyboardType.Number),
                keyboardActions = KeyboardActions(onDone = {
focusManager.moveFocus(FocusDirection.Right) }),
```

```
                modifier = Modifier
                    .fillMaxWidth(0.46f)
                    .alpha(visibleES),
                label = { Text(text = "In", fontSize = 16.sp) },
                singleLine = true,
                isError = errorESinicio,
                enabled = checkboxMarcado
            )

            Spacer(modifier = Modifier.width(8.dp))

            // Campo para FIN de bloqueo de proceso por E/S
            OutlinedTextField(
                value = entradaSalidaFin,
                onValueChange = {
                    // Control de cantidad de caracteres a 2
                    if (it.matches("^([0-9][0-9]?|)$".toRegex()))
entradaSalidaFin = it
                },
                keyboardOptions = KeyboardOptions.Default.copy(keyboardType
= KeyboardType.Number),
                keyboardActions = KeyboardActions(onDone = {
focusManager.clearFocus() }),
                modifier = Modifier
                    .fillMaxWidth()
                    .alpha(visibleES),
                label = { Text(text = "Out", fontSize = 16.sp) },
                singleLine = true,
                isError = errorESfin,
                enabled = checkboxMarcado
            )
        }

        // Creamos un boton para agregar el proceso ingresado a la lista de
procesos, y llamamos a la funcion onSubmit cuando se hace clic en el boton
        Button(
            modifier = Modifier.padding(start = 15.dp, top = 10.dp),
            onClick =
            {
                procesoAgregado = true
                // Limpiar el foco para ocultar teclado y deseleccionar el
```

```
campo del formulario
                keyboardController?.hide()
                focusManager.clearFocus()
            },
        ) { Text(text = "+", fontWeight = FontWeight.Bold, fontSize =
15.sp) }

        // Mensaje de error para el usuario
        if (errorFormulario.isNotBlank()) {
            Text(
                text = errorFormulario,
                color = MaterialTheme.colors.error,
                style = MaterialTheme.typography.caption,
                textAlign = TextAlign.Center,
                modifier = Modifier.width(anchuraFormularioNombres.dp)
            )
        } else if (errorFormularioQuantum.isNotBlank()) {
            Text(
                text = errorFormularioQuantum,
                color = MaterialTheme.colors.error,
                style = MaterialTheme.typography.caption,
                textAlign = TextAlign.Center,
                modifier = Modifier.width(anchuraFormularioNombres.dp)
            )
        } else if (errorFormularioES.isNotBlank()) {
            Text(
                text = errorFormularioES,
                color = MaterialTheme.colors.error,
                style = MaterialTheme.typography.caption,
                textAlign = TextAlign.Center,
                modifier = Modifier.width(anchuraFormularioNombres.dp)
            )
        } else {
            Spacer(modifier = Modifier.height(16.dp))
        }
    }

    // Separador para los campos de la segunda columna
    Spacer(modifier = Modifier.width(8.dp))

    // Contenedor para la segunda columna de campos del formulario
```

```
    Column {
        Spacer(modifier = Modifier.height((-3).dp))
        // Campo para introducir el quantum para Round Robin
        OutlinedTextField(
            enabled = quantumEnabled,
            value = quantum,
            onValueChange = {
                // Control de cantidad de caracteres
                if (it.length <= 2) {
                    quantum = it
                    tiempoQuantum = it
                }
            },
            label = { Text(stringResource(id =
R.string.formulario_quantum)) },
            keyboardOptions = KeyboardOptions.Default.copy(keyboardType =
KeyboardType.Number),
            keyboardActions = KeyboardActions(onDone = {
focusManager.clearFocus() }),
            modifier = Modifier
                .width(anchuraFormularioTiempos.dp)
                .alpha(quantumVisibilityAlpha),
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Update,
                    contentDescription = "Icono quantum de proceso en RR"
                )
            },
            singleLine = true,
            isError = errorQuantum
        )

        // Campo para introducir el tiempo de llegada
        OutlinedTextField(
            value = tiempoLlegada,
            onValueChange = {
                // Control de cantidad de caracteres a 2
                if (it.matches("^([0-9][0-9]?|)$".toRegex())) tiempoLlegada
= it
            },
            label = { Text(stringResource(id =
```

```
R.string.formulario_llegada)) },
            keyboardOptions = KeyboardOptions.Default.copy(keyboardType =
KeyboardType.Number),
            keyboardActions = KeyboardActions(onDone = {
focusManager.moveFocus(FocusDirection.Next) }),
            modifier = Modifier.width(anchuraFormularioTiempos.dp),
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Schedule,
                    contentDescription = "Icono tiempo de llegada"
                )
            },
            singleLine = true,
            isError = errorLlegada
        )

        // Campo para introducir la duracion del proceso
        OutlinedTextField(
            value = duracion,
            onValueChange = {
                // Control de cantidad de caracteres y no 0 inicial
                if (it.matches("^([1-9][0-9]?|)$".toRegex())) duracion = it
            },
            label = { Text(stringResource(id =
R.string.formulario_duracion)) },
            keyboardOptions = KeyboardOptions.Default.copy(keyboardType =
KeyboardType.Number),
            keyboardActions = KeyboardActions(onDone = {
focusManager.clearFocus() }),
            modifier = Modifier.width(anchuraFormularioTiempos.dp),
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.HourglassBottom,
                    contentDescription = "Icono duracion de proceso"
                )
            },
            singleLine = true,
            isError = errorDuracion
        )

        // Agregamos el botón "Siguiente" que llame a la funcion
```

```
correspondiente al metodo seleccionado
        Column(verticalArrangement = Arrangement.Bottom,
horizontalAlignment = CenterHorizontally) {
            CommonRoundedButton(
                text = stringResource(id = R.string.common_buttonNext),
                isEnabled = siguienteEnabled,
                onClick = {
                    // Llama a la funcion que controla el algoritmo a
ejecutar
                    llamarAlgoritmo()

                    // Control de estado para indicar la comprobacion de
errores en el campo de quantum
                    quantumSeleccionado = true

                    // Cambio de estado para indicar la pulsacion del boton
y cambiar a la pagina siguiente
                    siguienteSeleccionado.value = true
                },
                modifier = Modifier.width(anchuraFormularioTiempos.dp)
            )
        }
    }

    // Si es correcto se agrega el proceso y reinicia estado
    if (procesoAgregado) {
        agregarProceso()
        procesoAgregado = false
    }

    // Si es correcto se pasa a la siguiente pagina y se reinicia el estado
    if (quantumSeleccionado) {
        comprobarQuantum()
        quantumSeleccionado = false
    }
}
```

- **Subcarpeta "*launch*"**

En esta carpeta se considera el fichero correspondiente a la pantalla de lanzamiento de la aplicación, el cual considera la animación de carga para la inicialización del resto de la herramienta.

- ***SplashScreen.kt***

```
package com.i72pehej.cpuschedulerapp.usecases.launch

import android.view.animation.OvershootInterpolator
import androidx.compose.animation.core.Animatable
import androidx.compose.animation.core.AnimationVector1D
import androidx.compose.animation.core.tween
import androidx.compose.foundation.Canvas
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment.Companion.CenterHorizontally
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.scale
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.res.painterResource
import androidx.navigation.NavHostController
import com.i72pehej.cpuschedulerapp.navigation.AppScreens
import com.i72pehej.cpuschedulerapp.util.appIconCPU1
import kotlinx.coroutines.delay

/**
 * @author Julen Perez Hernandez
 */

//
================================================================================
=======
```

```
/**
 * Pantalla de carga para cuando se inicia la aplicacion de forma que de
tiempo a que cargue lo necesario
 *
 * @param navController Elemento para controlar la informacion de
navegacion
 */
@Composable
fun SplashScreen(navController: NavHostController) {
    // Variable para la animacion de salida de circulo
    val scaleBall = remember {
        Animatable(1.0f)
    }

    // Variable para la animacion de entrada de Icono de App
    val scaleUco = remember {
        Animatable(0.0f)
    }

    // Elementos
    Splash(scaleUco, scaleBall)

    // Launch screen
    LaunchScreen(navController, scaleBall, scaleUco)
}

//
================================================================================
=======

/**
 * Creacion del efecto de bote del icono de la pantalla de carga
 *
 * @param navController Elemento para controlar la informacion de
navegacion
 * @param scaleBallIcon Escala inicial del circulo
 * @param scaleUcoIcon Escala inicial del icono de la app
 */
@Composable
fun LaunchScreen(
    navController: NavHostController,
```

```
    scaleBallIcon: Animatable<Float, AnimationVector1D>,
    scaleUcoIcon: Animatable<Float, AnimationVector1D>
) {
    // Llamada a animacion de transicion
    TransitionCircleExit(scaleBallIcon)

    // Llamada a animacion de icono
    AppIconAnimationEnter(navController, scaleUcoIcon)
}

//
================================================================================
=======

/**
 * Animacion del circulo de salida
 *
 * @param scaleBallIcon Escala inicial del circulo
 */
@Composable
fun TransitionCircleExit(scaleBallIcon: Animatable<Float,
AnimationVector1D>) {
    // Efecto para el circulo que sale
    LaunchedEffect(key1 = true) {
        scaleBallIcon.animateTo(
            targetValue = 0.0f,
            animationSpec = tween(900, easing = {
                OvershootInterpolator(0f).getInterpolation(it)
            })
        )

        // Espera para coordinar efectos
        delay(100)
    }
}

//
================================================================================
=======

/**
```

```
 * Animacion de icono de app
 *
 * @param navController Controlador de la navegacion
 * @param scaleUcoIcon Escala inicial del icono de la app
 */
@Composable
fun AppIconAnimationEnter(
    navController: NavHostController,
    scaleUcoIcon: Animatable<Float, AnimationVector1D>
) {
    // Efecto para icono que entra
    LaunchedEffect(key1 = true) {
        scaleUcoIcon.animateTo(
            targetValue = 1.1f,
            animationSpec = tween(1000, easing = {
                OvershootInterpolator(6f).getInterpolation(it)
            })
        )
        // Para mantener el logo en pantalla por estetica
        delay(400)

        // Se llama primero a esta funcion para que el Home sea la primera
pantalla
        // y en caso de darle hacia atras no volvamos a la splashscreen
        navController.popBackStack()
        navController.navigate(AppScreens.HomeScreen.route)
    }
}

//
================================================================================
=======

/**
 * Visualizacion de los elementos de la pantalla de carga
 *
 * @param scaleUco Escala inicial del icono de la app
 * @param scaleBall Escala inicial del circulo
 */
@Composable
fun Splash(
```

```
    scaleUco: Animatable<Float, AnimationVector1D>,
    scaleBall: Animatable<Float, AnimationVector1D>
) {
    // Pantalla de logo para la carga de la app
    Column(
        modifier = Modifier.fillMaxSize(),
        Arrangement.Center,
        CenterHorizontally
    ) {
        // Imagen de Icono de la App
        Image(
            painter = painterResource(id = appIconCPU1),
            contentDescription = "Logo de la app",
            modifier = Modifier
                .fillMaxWidth()
                .scale(scaleUco.value)
        )
    }

    val circleColor = MaterialTheme.colors.background

    // Transicion
    Column(
        modifier = Modifier.fillMaxSize(),
        Arrangement.Center,
        CenterHorizontally
    ) {
        // Imagen en blanco para desaparecer y que salga el icono de la App
        Canvas(
            modifier = Modifier
                .fillMaxSize()
                .scale(scaleBall.value)
        ) {
            val canvasWidth = size.width
            val canvasHeight = size.height

            drawCircle(
                color = circleColor,
                center = Offset(x = canvasWidth / 2, y = canvasHeight / 2),
                radius = size.minDimension
            )
```

```
        }
    }
}
```

- **Subcarpeta "*results*"**

Esta carpeta engloba los ficheros para las diferentes representaciones visuales de los resultados obtenidos por la simulación.

- ***ResultsScreen.kt***

```
package com.i72pehej.cpuschedulerapp.usecases.results

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.i72pehej.cpuschedulerapp.util.extensions.TablaTiemposResultados
import com.i72pehej.cpuschedulerapp.util.listaDeProcesosGlobal

/**
 * @author Julen Perez Hernandez
 */

/**
 * Pantalla de resultados en la que visualizar los tiempos obtenidos
 */
@Composable
fun ResultsScreen() {
    // Disposicion principal de la pantalla
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(8.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        // Generar tabla de resultados
```

```
        TablaTiemposResultados(procesos = listaDeProcesosGlobal)
    }
}
```

- ***GraphsScreen.kt***

```
package com.i72pehej.cpuschedulerapp.usecases.results

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Alignment.Companion.CenterVertically
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.i72pehej.cpuschedulerapp.ui.theme.Azul_com_3
import com.i72pehej.cpuschedulerapp.ui.theme.Rojo_com_2
import com.i72pehej.cpuschedulerapp.ui.theme.Verde_deriv_1
import com.i72pehej.cpuschedulerapp.util.extensions.TablaResultadosGraficos
import com.i72pehej.cpuschedulerapp.util.infoResultadosGlobal

/**
 * @author Julen Perez Hernandez
 */

/**
 * Pantalla de graficos en la que poder visualizar los resultados obtenidos
```

```
de manera grafica
 */
@Composable
//fun GraphsScreen(navController: NavHostController) {
fun GraphsScreen() {
    // Disposicion principal de la pantalla
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(8.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Top
    ) {
        // Tabla de leyenda para la simbologia de la tabla grafica de
resultados
        TablaLeyendaGraficos()

        Spacer(modifier = Modifier.height(8.dp))

        TablaResultadosGraficos(infoRes = infoResultadosGlobal)
    }
}

/**
 *
==========================================================================
================
 */

/**
 * Tabla para considerar la leyenda de los simbolos colocados en la grafica
 */
@Composable
fun TablaLeyendaGraficos() {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .background(MaterialTheme.colors.primaryVariant,
RoundedCornerShape(5.dp))
            .padding(8.dp)
    ) { FilaLeyenda() }
```

```
}

/**
 * Fila en la que visualizan todos los elementos de la leyenda
 */
@Composable
private fun FilaLeyenda() {
    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        SimboloDeLeyenda("E", "Espera", Color.Black.copy(alpha = 0.7f))
        SimboloDeLeyenda("X", "Ejecución", Azul_com_3.copy(alpha = 0.7f))
        SimboloDeLeyenda("B", "Bloqueado", Rojo_com_2.copy(alpha = 0.7f))
        SimboloDeLeyenda("C", "Completado", Verde_deriv_1.copy(alpha =
0.7f))
    }
}

/**
 * Funcion para crear cada uno de los elementos que se colocaran en la
leyenda
 *
 * @param simbolo Simbolo que se colocara en la tabla grafica
 * @param etiqueta Significado al que hace referencia el simbolo
 * @param color Color de fondo del simbolo
 */
@Composable
private fun SimboloDeLeyenda(simbolo: String, etiqueta: String, color:
Color) {
    Row(verticalAlignment = CenterVertically) {
        Surface(
            modifier = Modifier.size(30.dp),
            shape = MaterialTheme.shapes.small,
            color = color
        ) {
            Text(
                text = simbolo,
                fontSize = 20.sp,
                color = Color.White,
                modifier = Modifier.align(CenterVertically),
```

```
                textAlign = TextAlign.Center
            )
        }

        Spacer(modifier = Modifier.size(4.dp))

        Text(
            text = etiqueta,
            fontSize = 12.sp,
            color = MaterialTheme.colors.onSurface.copy(alpha = 0.8f),
            modifier = Modifier.align(CenterVertically)
        )
    }
}
```

- ***QueuesScreen.kt***

```
package com.i72pehej.cpuschedulerapp.usecases.results

import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment.Companion.CenterHorizontally
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import com.i72pehej.cpuschedulerapp.util.classes.Proceso
import com.i72pehej.cpuschedulerapp.util.infoResultadosGlobal
```

```
/**
 * @author Julen Perez Hernandez
 */



/**
 * Pantalla de colas en la que ver las diferentes colas que se crean cuando
los procesos estan listos
 */
@Composable
fun QueuesScreen() {
    // Disposicion principal de la pantalla
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(8.dp),
        horizontalAlignment = CenterHorizontally,
        verticalArrangement = Arrangement.Top
    ) {
        // Creacion del listado de colas
        crearListaColas()

        // Disposicion de la tabla resultante
        TablaColasDeProcesos()
    }
}

/**
 *
================================================================================
================
 */

// Listado de las colas en cada momento
val listaDeColas: MutableList<List<String>> = mutableListOf()

/**
 * Funcion que crea una lista con el orden de los procesos en la cola en
cada momento
 */
fun crearListaColas() {
```

```
    // Limpiamos el listado para evitar sobrecarga de la lista
    listaDeColas.clear()

    // Obtenemos los indices de inicio y fin del bucle
    val primerIndice = infoResultadosGlobal.first().getMomento()
    val ultimoIndice = infoResultadosGlobal.last().getMomento()

    // Crearmos una copia de los estados para evitar trabajar con la
original
    val listaEstados = infoResultadosGlobal.toMutableList()

    // Iteramos por cada columna (== momento) para obtener la lista de
procesos en cola en ese momento
    for (columna in primerIndice until ultimoIndice) {
        // Sublista para obtener solo los elementos del momento actual
        var subListaEstados = listaEstados.filter { it.getMomento() ==
columna }

        // Eliminamos los procesos en estados que no corresponden a la cola
de LISTOS
        subListaEstados = subListaEstados.filter { (it.getEstado() !=
Proceso.EstadoDeProceso.BLOQUEADO) && (it.getEstado() !=
Proceso.EstadoDeProceso.COMPLETADO) }

        // Sublista para obtener los nombres de los procesos del momento
actual
        val subListaNombres = subListaEstados.map { it.getNombre() }

        // Agregamos a la lista de colas la lista de nombres de procesos
        listaDeColas.add(subListaNombres)
    }
}

/**
 *
==========================================================================
================
 */
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun TablaColasDeProcesos() {
```

```
    // Creamos una tabla utilizando LazyColumn
    LazyColumn(
        modifier = Modifier
            .fillMaxWidth()
            .background(MaterialTheme.colors.primaryVariant,
RoundedCornerShape(5.dp))
            .padding(8.dp)
    ) {
        // Agregamos una fila para el encabezado de la tabla
        stickyHeader {
            Row(
                modifier = Modifier
                    .background(MaterialTheme.colors.secondaryVariant,
RoundedCornerShape(5.dp))
                    .padding(4.dp)
            ) {
                // Agregamos una celda inicial en blanco para considerar la
columna
                Text(
                    text = "",
                    modifier = Modifier.width(35.dp),
                    textAlign = TextAlign.Center,
                    fontWeight = FontWeight.Bold
                )

                // Agregamos la columna para el listado de colas
                Text(
                    text = "Cola de Procesos",
                    modifier = Modifier.weight(1f),
                    textAlign = TextAlign.Center,
                    fontWeight = FontWeight.Bold
                )
            }
        }

        // Agregamos una fila para cada cola en cada momento
        itemsIndexed(listaDeColas) { index, item ->
            // Fila de colas
            Row(modifier = Modifier.padding(4.dp)) {
                // Agregamos el momento de la cola
                Text(
```

```
                    text = "$index |",
                    modifier = Modifier.width(35.dp),
                    textAlign = TextAlign.Center
                )

                // Agregamos la cola
                Text(text = item.toString(), modifier =
Modifier.weight(1f))
            }
        }
    }
}
```

- **Carpeta *"util"***

A continuación, se presenta el código correspondiente a todos los ficheros correspondientes a la gestión de todas las utilidades comunes de la aplicación.

- **Subcarpeta *"classes"***

Esta carpeta almacena los ficheros correspondientes a las diferentes clases personalizadas creadas para una mejor gestión de los datos utilizados a lo largo de toda la herramienta por todas las funciones implementadas.

- ***ProcessClass.kt***

```
package com.i72pehej.cpuschedulerapp.util.classes

import com.i72pehej.cpuschedulerapp.util.infoResultadosGlobal

/**
 * @author Julen Perez Hernandez
 */

/**
 * ====================================================================
 */

/**
 * Clase que representa cada uno de los procesos
 *
 * @property nombre Nombre asignado al proceso
 * @property tiempoLlegada Momento en el que el proceso entra en la cola de
procesos listos
 * @property duracion Duracion estimada del proceso
 * @constructor Crea la representacion de un Proceso de CPU
 */
data class Proceso(
    private var nombre: String,
    private var tiempoLlegada: Int,
    private var duracion: Int,
    private var estado: EstadoDeProceso = EstadoDeProceso.LISTO
) {
    // Listado de estados asociados a los procesos
    enum class EstadoDeProceso {
        LISTO,      // Cuando el proceso se encuentra a la espera de entrar
```

```
en la CPU
        EJECUCION,  // Cuando el proceso se encuentra en la CPU
        BLOQUEADO,  // Cuando el proceso queda temporalmente detenido por
otras tareas
        COMPLETADO  // Cuando el proceso ha terminado su ejecucion
    }

    // Variables para considerar tiempos de Inicio y Fin de un evento de
E/S del proceso
    private var tiempoEntrada = -1
    private var tiempoSalida = -1

    // Setters y getters de los parametros
    fun getNombre(): String {
        return this.nombre
    }

//    fun setNombre(nombre: String) {
//        this.nombre = nombre
//    }

    fun getLlegada(): Int {
        return this.tiempoLlegada
    }

    fun setLlegada(tiempo: Int) {
        this.tiempoLlegada = tiempo
    }

    fun getEstado(): EstadoDeProceso {
        return this.estado
    }

    fun setEstado(estado: EstadoDeProceso) {
        this.estado = estado
    }

    fun getDuracion(): Int {
        return this.duracion
    }
```

```
//    fun setDuracion(tiempo: Int) {
//        this.duracion = tiempo
//    }

    fun getTiempoEntrada(): Int {
        return this.tiempoEntrada
    }

    fun setTiempoEntrada(tiempo: Int) {
        this.tiempoEntrada = tiempo
    }

    fun getTiempoSalida(): Int {
        return this.tiempoSalida
    }

    fun setTiempoSalida(tiempo: Int) {
        this.tiempoSalida = tiempo
    }

    fun getTiempoDeEsperaES(): Int {
        return this.getTiempoSalida() - this.getTiempoEntrada()
    }

    // Control de los tiempos del proceso

//    private var tiempoRespuesta: Int = 0
//    fun getTiempoRespuesta(): Int {
//        return this.tiempoRespuesta
//    }

//    fun setTiempoRespuesta(tiempo: Int) {
//        this.tiempoRespuesta = tiempo
//    }

    private var tiempoRestante: Int = this.getDuracion()
    fun getTiempoRestante(): Int {
        return this.tiempoRestante
    }

    fun setTiempoRestante(tiempo: Int) {
```

```
        this.tiempoRestante = tiempo
    }

    fun getTiempoEstancia() = this.tiempoFin() - this.getLlegada()

    fun getTiempoEspera(): Int {
        return (this.getTiempoEstancia() - this.getDuracion())
    }

    // Variable para operar con el tiempo de espera del proceso
    private var tiempoEsperaLocal = 0
    fun getTiempoEsperaLocal(): Int {
        return this.tiempoEsperaLocal
    }

    fun setTiempoEsperaLocal(tiempo: Int) {
        this.tiempoEsperaLocal = tiempo
    }

    // Tiempo que ha tardado el proceso en completarse desde su llegada
    fun tiempoFin(): Int {
        // Se busca en la lista de estados la primera aparicion del
proceso, correspondiente con el estado de COMPLETADO, sino devuelve -1 como
"error"
        return infoResultadosGlobal.find { (it.getNombre() ==
this.getNombre()) && (it.getEstado() == EstadoDeProceso.COMPLETADO)
}?.getMomento() ?: -1
    }

    // Tiempo en el que el proceso inicia su ejecucion
    fun tiempoInicio(): Int {
        // Se busca en la lista de estados la primera aparicion del
proceso, correspondiente con el estado de EJECUCION, sino devuelve -1 como
"error"
        return infoResultadosGlobal.find { (it.getNombre() ==
this.getNombre()) && (it.getEstado() == EstadoDeProceso.EJECUCION)
}?.getMomento() ?: -1
    }

    // Limpieza de tiempos de control
    fun reset() {
```

```
            setEstado(EstadoDeProceso.LISTO)
            setTiempoRestante(this.getDuracion())
        }
    }
/**
 * ========================================================================
 */

/**
 * Funcion para crear un proceso nuevo a la lista de procesos
 *
 * @param nombre Nombre asignado al proceso
 * @param tiempoLlegada Momento en el que el proceso entra en la cola de
procesos listos
 * @param duracion Duracion estimada del proceso
 *
 * @return Devuelve un proceso con los valores correspondientes
 */
fun crearProceso(
    nombre: String,
    tiempoLlegada: Int,
    duracion: Int,
    estado: Proceso.EstadoDeProceso
): Proceso {
    // Se crea un proceso y se devuelve
    return Proceso(nombre, tiempoLlegada, duracion, estado)
}

/**
 * Sobrecarga de la funcion para crear un proceso junto a los tiempos de
bloqueo por E/S
 *
 * @param nombre Nombre asignado al proceso
 * @param tiempoLlegada Momento en el que el proceso entra en la cola de
procesos listos
 * @param duracion Duracion estimada del proceso
 * @param tiempoEntrada Momento en el que el proceso se bloquea por una
accion de E/S
 * @param tiempoSalida Momento en el que el proceso sale del estado de
bloqueo por E/S
 *
```

```
 * @return Devuelve un proceso con los valores correspondientes y tiempos
de E/S
 */
fun crearProceso(
    nombre: String,
    tiempoLlegada: Int,
    duracion: Int,
    estado: Proceso.EstadoDeProceso,
    tiempoEntrada: Int,
    tiempoSalida: Int
): Proceso {
    // Crea un proceso
    val proceso = Proceso(nombre, tiempoLlegada, duracion, estado)

    // Agrega los tiempos de E/S al proceso creado
    proceso.setTiempoEntrada(tiempoEntrada)
    proceso.setTiempoSalida(tiempoSalida)

    return proceso
}

/**
 * ========================================================================
 */

/**
 * Funcion que ordena la lista de procesos por orden de llegada de forma
ascendente
 *
 * @param listaDeProcesos Listado de procesos que se van a ordenar por
orden de llegada
 *
 * @return Devuelve el listado de procesos ordenado
 */
fun ordenarLlegadaProcesos(listaDeProcesos: MutableList<Proceso>):
List<Proceso> {
    listaDeProcesos.sortBy { proceso: Proceso -> proceso.getLlegada() }

    return listaDeProcesos
}
```

- ***StatesGraphsClass.kt***

```kotlin
package com.i72pehej.cpuschedulerapp.util.classes

/**
 * @author Julen Perez Hernandez
 */

/**
 *
==========================================================================
================
 */

/**
 * Informacion del estado de un proceso en un momento determinado
 *
 * @property nombre El nombre del proceso asociado al momento del cambio de
estado
 * @property estado Estado que pasa a tener el proceso en el momento actual
 * @property momento Tiempo en el que el proceso adquiere un estado
distinto al anterior
 */
data class InfoGraficoEstados(
    private var nombre: String,
    private var estado: Proceso.EstadoDeProceso,
    private var momento: Int
) {
    // Getters
    fun getNombre(): String {
        return this.nombre
    }

    fun getEstado(): Proceso.EstadoDeProceso {
        return this.estado
    }

    fun getMomento(): Int {
        return this.momento
    }
}
```

- **Subcarpeta "*extensions*"**

En esta subcarpeta se encuentran los archivos correspondientes a diferentes extensiones de funcionalidades básicas o algunas implementaciones de funcionalidades de realización de tareas sencillas.

- ***BackPressControl.kt***

```
package com.i72pehej.cpuschedulerapp.util.extensions

import android.widget.Toast
import androidx.activity.compose.BackHandler
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import com.i72pehej.cpuschedulerapp.R
import kotlinx.coroutines.delay

/**
 * @author Julen Perez Hernandez
 */

open class BackPressControl {
    object Idle : BackPressControl()
    object InitialTouch : BackPressControl()
}

/**
 * Funcion para controlar que la app no se cierre por accidente al pulsar
el boton de back
 */
@Composable
fun ConfirmacionBackPress() {
    // Variables de control de estados
    var showToast by remember { mutableStateOf(false) }
    var backPressState by remember {
mutableStateOf<BackPressControl>(BackPressControl.Idle) }
```

```
    val context = LocalContext.current

    // Creacion del mensaje
    if (showToast) {
        Toast.makeText(
            context,
            stringResource(id = R.string.back_press_confirmacion),
            Toast.LENGTH_SHORT
        ).show()
        showToast = false
    }

    // Control de la pulsacion del boton y tiempo de espera para la segunda
pulsacion
    LaunchedEffect(backPressState) {
        if (backPressState == BackPressControl.InitialTouch) {
            delay(1500)
            backPressState = BackPressControl.Idle
        }
    }

    // Control del boton de back para cambiar los estados
    BackHandler(backPressState == BackPressControl.Idle) {
        backPressState = BackPressControl.InitialTouch
        showToast = true
    }
}
```

- ***CleanTables.kt***

```
package com.i72pehej.cpuschedulerapp.util.extensions

import androidx.compose.foundation.layout.width
import androidx.compose.material.AlertDialog
import androidx.compose.material.Button
import androidx.compose.material.DropdownMenuItem
import androidx.compose.material.ExperimentalMaterialApi
import androidx.compose.material.ExposedDropdownMenuBox
import androidx.compose.material.ExposedDropdownMenuDefaults
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.Composable
```

```
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.i72pehej.cpuschedulerapp.util.anchuraFormularioTiempos
import com.i72pehej.cpuschedulerapp.util.infoResultadosGlobal
import com.i72pehej.cpuschedulerapp.util.listaDeProcesosGlobal

/**\
 * @author Julen Perez Hernandez
 */

/**
 * ========================================================================
 */

/**
 * Funcion que limpia las listas de procesos y estados para resetear las
tablas
 *
 * @param showDialog Booleano para mostrar o no el dialogo de confirmacion
 */
@Composable
fun LimpiarProcesos(showDialog: MutableState<Boolean>) {
    // Si se ha pulsado el boton de confirmacion, se limpian los procesos
    if (showDialog.value) {
        AlertDialog(
            onDismissRequest = { showDialog.value = false },
            title = { Text(text = "Confirmacion") },
            text = { Text(text = "Se van a borrar los procesos agregados.
¿Quiere seguir?") },
            confirmButton = {
                Button(onClick = {
                    showDialog.value = false
                    listaDeProcesosGlobal.clear()
                    infoResultadosGlobal.clear()
                }) { Text(text = "Si") }
```

```
            },
            dismissButton = { Button(onClick = { showDialog.value = false
}) { Text(text = "No") } },
            modifier = Modifier.width(300.dp)
        )
    }
}

/**
 * ========================================================================
 */

/**
 * Funcion para seleccionar un proceso para eliminarlo de la lista
 *
 * @param showDialog Booleano para mostrar o no el cuadro de dialogo
 */
@OptIn(ExperimentalMaterialApi::class)
@Composable
fun EliminarProcesoTabla(showDialog: MutableState<Boolean>) {
    if (showDialog.value) {
        // Varibales para gestionar la alerta
        var expandir by remember { mutableStateOf(value = false) }
        var procesoSelect by remember {
mutableStateOf(listaDeProcesosGlobal.first().getNombre()) }

        // Cuadro de dialogo
        AlertDialog(
            onDismissRequest = { showDialog.value = false },
            title = { Text(text = "Seleccione el proceso a eliminar") },
            text = {
                // Menu desplegable para seleccion del proceso a eliminar
                ExposedDropdownMenuBox(
                    modifier = Modifier.width(anchuraFormularioTiempos.dp),
                    expanded = expandir,
                    onExpandedChange = { expandir = it }
                ) {
                    TextField(
                        readOnly = true,
                        value = procesoSelect,
                        onValueChange = { },
```

```
                        label = { Text("Proceso", fontSize = 12.sp) },
                        trailingIcon = {
ExposedDropdownMenuDefaults.TrailingIcon(expanded = expandir) },
                        colors =
ExposedDropdownMenuDefaults.textFieldColors()
                    )
                    ExposedDropdownMenu(
                        expanded = expandir,
                        onDismissRequest = { expandir = false }
                    ) {
                        listaDeProcesosGlobal.forEach { procesoSeleccionado
->
                            DropdownMenuItem(
                                onClick = {
                                    // Se selecciona el proceso
                                    procesoSelect =
procesoSeleccionado.getNombre()
                                    expandir = false
                                }
                            ) { Text(text =
procesoSeleccionado.getNombre()) }
                        }
                    }
                }
            },
            confirmButton = {
                Button(onClick = {
                    showDialog.value = false

                    // Eliminar el proceso seleccionado

listaDeProcesosGlobal.removeAt(listaDeProcesosGlobal.indexOf(listaDeProceso
sGlobal.find { it.getNombre() == procesoSelect }))
                }) { Text(text = "Eliminar") }
            },
            dismissButton = { Button(onClick = { showDialog.value = false
}) { Text(text = "Cancelar") } },
            modifier = Modifier.width(300.dp)
        )
    }
}
```

- ***DataTables.kt***

```kotlin
package com.i72pehej.cpuschedulerapp.util.extensions

import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Divider
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.i72pehej.cpuschedulerapp.R
import com.i72pehej.cpuschedulerapp.ui.theme.Azul_com_3
import com.i72pehej.cpuschedulerapp.ui.theme.Rojo_com_2
import com.i72pehej.cpuschedulerapp.ui.theme.Verde_deriv_1
import com.i72pehej.cpuschedulerapp.util.classes.InfoGraficoEstados
import com.i72pehej.cpuschedulerapp.util.classes.Proceso
import com.i72pehej.cpuschedulerapp.util.infoResultadosGlobal

/**\
 * @author Julen Perez Hernandez
 */

/**
```

```
 *
==========================================================================
================
 */

/**
 * Creacion de la tabla de procesos agregados
 *
 * @param procesos Lista de los procesos agregados
 */
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun TablaProcesos(procesos: List<Proceso>) {
    // Si la lista de procesos no está vacía
    if (procesos.isNotEmpty()) {
        // Comprobar si tiene E/S
        val entradaSalida = procesos.any { it.getTiempoEntrada() > 0 }

        // Creamos una tabla utilizando LazyColumn
        LazyColumn(
            modifier = Modifier
                .fillMaxWidth()
                .background(MaterialTheme.colors.primaryVariant,
RoundedCornerShape(5.dp))
                .padding(8.dp)
        ) {
            // Agregamos una fila para el encabezado de la tabla
            stickyHeader {
                Row(
                    modifier = Modifier
                        .background(MaterialTheme.colors.secondaryVariant,
RoundedCornerShape(5.dp))
                        .padding(4.dp)
                ) {
                    // Agregamos cada columna a la fila de encabezado con
un peso de 1f para que tengan el mismo ancho
                    Text(
                        stringResource(id = R.string.formulario_nombre),
                        modifier = Modifier.weight(1f), // Le damos un peso
de 1f para que tenga el mismo ancho que las otras columnas.
                        textAlign = TextAlign.Center,
```

```
                    fontWeight = FontWeight.Bold
                )
                Text(
                    stringResource(id = R.string.formulario_llegada),
                    modifier = Modifier.weight(1f),
                    textAlign = TextAlign.Center,
                    fontWeight = FontWeight.Bold
                )
                Text(
                    stringResource(id = R.string.formulario_duracion),
                    modifier = Modifier.weight(1f),
                    textAlign = TextAlign.Center,
                    fontWeight = FontWeight.Bold
                )

                // En caso de tener E/S se coloca tambien en la tabla
                if (entradaSalida) {
                    Text(
                        text = "E",
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                    Text(
                        text = "S",
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                }
            }
        }

        // Agregamos una fila para cada proceso en la lista de procesos
        items(procesos) { proceso ->
            Row(modifier = Modifier.padding(4.dp)) {
                Text(
                    proceso.getNombre(),
                    modifier = Modifier.weight(1f),
                    textAlign = TextAlign.Center
                )
```

```
                    Text(
                        proceso.getLlegada().toString(),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )
                    Text(
                        proceso.getDuracion().toString(),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )

                    // Si tiene E/S
                    if (entradaSalida) {
                        Text(
                            if (proceso.getTiempoEntrada() > 0)
proceso.getTiempoEntrada().toString() else "-",
                            modifier = Modifier.weight(1f),
                            textAlign = TextAlign.Center
                        )
                        Text(
                            if (proceso.getTiempoSalida() > 0)
proceso.getTiempoSalida().toString() else "-",
                            modifier = Modifier.weight(1f),
                            textAlign = TextAlign.Center
                        )
                    }
                }
            }
        }
    } else {
        // Si la lista de procesos está vacía, mostramos un mensaje
indicando que no hay procesos
        Column(modifier = Modifier.fillMaxSize(), verticalArrangement =
Arrangement.Center, horizontalAlignment = Alignment.CenterHorizontally) {
            Text(
                stringResource(id = R.string.tabla_vacia),
                modifier = Modifier.padding(8.dp),
            )
        }
    }
}
```

```
/**
 *
================================================================================
================
 */

/**
 * Creacion de la tabla de los tiempos resultantes obtenidos
 *
 * @param procesos Lista de los procesos
 */
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun TablaTiemposResultados(procesos: List<Proceso>) {
    // Si la lista de estados no está vacía
    if (infoResultadosGlobal.isNotEmpty()) {
        // Comprobar si tiene E/S
        val entradaSalida = procesos.any { it.getTiempoEntrada() > 0 }

        // Creamos una tabla utilizando LazyColumn
        LazyColumn(
            modifier = Modifier
                .fillMaxWidth()
                .background(MaterialTheme.colors.primaryVariant,
RoundedCornerShape(5.dp))
                .padding(8.dp)
        ) {
            // Agregamos una fila para el encabezado de la tabla
            stickyHeader {
                Row(
                    modifier = Modifier
                        .background(MaterialTheme.colors.secondaryVariant,
RoundedCornerShape(5.dp))
                        .padding(4.dp)
                ) {
                    // Agregamos cada columna a la fila de encabezado con
un peso de 1f para que tengan el mismo ancho
                    Text(
                        stringResource(id = R.string.formulario_nombre),
                        modifier = Modifier.weight(1f), // Le damos un peso
```

```
de 1f para que tenga el mismo ancho que las otras columnas.
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                    Text(
                        stringResource(id = R.string.formulario_llegada),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                    Text(
                        stringResource(id = R.string.formulario_duracion),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )

                    // En caso de tener E/S se coloca tambien en la tabla
                    if (entradaSalida) {
                        Text(
                            text = "E",
                            modifier = Modifier.weight(1f),
                            textAlign = TextAlign.Center,
                            fontWeight = FontWeight.Bold
                        )
                        Text(
                            text = "S",
                            modifier = Modifier.weight(1f),
                            textAlign = TextAlign.Center,
                            fontWeight = FontWeight.Bold
                        )
                    }

                    Text(
                        stringResource(id = R.string.nombre_tiempo_inicio),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                    Text(
                        stringResource(id = R.string.nombre_tiempo_fin),
```

```
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                    Text(
                        stringResource(id =
R.string.nombre_tiempo_estancia),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                    Text(
                        stringResource(id = R.string.nombre_tiempo_espera),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )
                }
            }

            // Agregamos una fila para cada proceso en la lista de procesos
            items(procesos) { proceso ->
                Row(modifier = Modifier.padding(4.dp)) {
                    Text(
                        proceso.getNombre(),                    // Nombre
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )
                    Text(
                        proceso.getLlegada().toString(),        // Llegada
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )
                    Text(
                        proceso.getDuracion().toString(),       // Duracion
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )

                    // Si tiene E/S
                    if (entradaSalida) {
```

```
                        Text(
                            if (proceso.getTiempoEntrada() > 0)
proceso.getTiempoEntrada().toString() else "-",
                            modifier = Modifier.weight(1f),
                            textAlign = TextAlign.Center
                        )
                        Text(
                            if (proceso.getTiempoSalida() > 0)
proceso.getTiempoSalida().toString() else "-",
                            modifier = Modifier.weight(1f),
                            textAlign = TextAlign.Center
                        )
                    }

                    Text(
                        proceso.tiempoInicio().toString(),           //
Inicio
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )
                    Text(
                        proceso.tiempoFin().toString(),              // Fin
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )
                    Text(
//                          "${proceso.getTiempoEstancia()} =
${proceso.tiempoFin()} - ${proceso.getLlegada()}",    // Estancia
                        proceso.getTiempoEstancia().toString(),    //
Estancia
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )
                    Text(
//                          "${proceso.getTiempoEspera()} =
${proceso.getTiempoEstancia()} - ${proceso.getDuracion()}",       // Espera
                        proceso.getTiempoEspera().toString(),        //
Espera
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )
```

```
                }
            }
        }
    }
}

/**
 *
================================================================================
================
 */

/**
 * Funcion de extension para obtener el ultimo elemento que cumple con una
condicion
 *
 * @param T Tipo generico
 * @param predicate Condicion que se debe cumplir
 * @return Devuleve el ultimo elemento que cumple la condicion o NULL en
caso de no encontrar ninguno
 */
fun <T> List<T>.filterLast(predicate: (T) -> Boolean): T? {
    return this.reversed().firstOrNull(predicate)
}

/**
 *
================================================================================
================
 */

/**
 * Creacion de la tabla de procesos agregados
 *
 * @param infoRes Lista de los estados adquiridos por los procesos
 */
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun TablaResultadosGraficos(infoRes: List<InfoGraficoEstados>) {
    // Si la lista de procesos no está vacía
    if (infoRes.isNotEmpty()) {
```

```
        // Variable que almacena el valor maximo que tendra la linea de
tiempos
        val maxMomento = infoRes.last().getMomento() + 1

        // Creamos una tabla utilizando LazyColumn
        LazyColumn(
            modifier = Modifier
                .fillMaxWidth()
                .background(MaterialTheme.colors.primaryVariant,
RoundedCornerShape(5.dp))
                .padding(8.dp)
        ) {
            // Agregamos una fila para el encabezado de la tabla
            stickyHeader {
                Row(
                    modifier = Modifier
                        .background(MaterialTheme.colors.secondaryVariant,
RoundedCornerShape(5.dp))
                        .padding(4.dp)
                ) {
                    // Agregamos una columna inicial para los procesos
                    Text(
                        text = "",
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center,
                        fontWeight = FontWeight.Bold
                    )

                    // Agregamos el numero de columnas correspondientes a
cada momento de la linea de tiempos
                    for (nCols in infoRes.first().getMomento() until
maxMomento) {
                        Text(
                            text = "$nCols",
                            modifier = Modifier.weight(1f),
                            textAlign = TextAlign.Center,
                            fontWeight = FontWeight.Bold
                        )
                    }
                }
            }
```

```
            // Sublista con la primera aparicion de cada uno de los
procesos con nombres distintos para crear las filas
            val listaNombres = infoRes.distinctBy { it.getNombre() }

            // Agregamos una fila para cada proceso en la lista de procesos
            items(listaNombres) { nombreActual ->
                // Filas
                Row(modifier = Modifier.padding(4.dp)) {
                    // Agregamos los nombres de los procesos
                    Text(
                        text = nombreActual.getNombre(),
                        modifier = Modifier.weight(1f),
                        textAlign = TextAlign.Center
                    )

                    // Separador vertical
                    Divider(
                        modifier = Modifier
                            .height(21.dp)
                            .width(1.dp), color =
MaterialTheme.colors.secondary
                    )

                    // Filtramos la lista para obtener los estados de la
fila actual
                    val listaFilaActual = infoRes.filter { it.getNombre()
== nombreActual.getNombre() }

                    // Recorremos las columnas de tiempos
                    for (cols in infoRes.first().getMomento() until
maxMomento) {
                        var simbolo = ""
                        var color = Color.Transparent

                        // Para cada momento, se compara si el proceso
tiene evento y se coloca el simbolo correspondiente
                        if (listaFilaActual.any { it.getMomento() == cols
}) {

                            // EL WHEN TIENE QUE BUSCAR EL ESTADO DE LA
```

```
ULTIMA APARICION DE UN ESTADO CON EL MOMENTO QUE TENEMOS AHORA
                            simbolo = when (listaFilaActual.filterLast {
it.getMomento() == cols }?.getEstado()) {
                                Proceso.EstadoDeProceso.LISTO -> {
                                    color = Color.Black.copy(alpha = 0.8f)
                                    "E"
                                }

                                Proceso.EstadoDeProceso.EJECUCION -> {
                                    color = Azul_com_3.copy(alpha = 0.8f)
                                    "X"
                                }

                                Proceso.EstadoDeProceso.BLOQUEADO -> {
                                    color = Rojo_com_2.copy(alpha = 0.8f)
                                    "B"
                                }

                                Proceso.EstadoDeProceso.COMPLETADO -> {
                                    color = Verde_deriv_1.copy(alpha =
0.8f)
                                    "C"
                                }

                                else -> {
                                    color = Color.Transparent
                                    ""
                                }
                            }
                        }

                        // Texto a poner en la celda
                        Text(
                            text = simbolo,
                            fontSize = 15.sp,
                            color = Color.White,
                            modifier = Modifier
                                .align(Alignment.CenterVertically)
                                .background(color)
                                .weight(1f),
                            textAlign = TextAlign.Center
```

```
                            )
                        }
                    }
                }
            }
        }
    }
```

- *ThemeSwitcher.kt*

```
package com.i72pehej.cpuschedulerapp.util.extensions

import androidx.compose.animation.core.AnimationSpec
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.tween
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Icon
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.LightMode
import androidx.compose.material.icons.filled.Nightlight
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import com.i72pehej.cpuschedulerapp.ui.theme.Azul_base
import com.i72pehej.cpuschedulerapp.ui.theme.Azul_com_1
```

```
/**
 * @author Julen Perez Hernandez
 */

/**
 * Boton que visualiza el cambio de tema de la app
 *
 * @param darkTheme Booleano para controlar el cambio de tema
 * @param size Tamaño del boton que cambia el tema (el contenedor padre
tiene el doble = size*2)
 * @param iconSize Tamaño de los iconos
 * @param padding Padding entre el contenedor base y el boton
 * @param borderWidth Anchura del borde del contenedor padre
 * @param parentShape Forma del contenedor padre
 * @param toggleShape Forma del boton
 * @param animationSpec Animacion que realiza el boton para intercambiar
entre temas
 * @param onClick Accion que realiza el boton
 */
@Composable
fun ThemeSwitcher(
    darkTheme: Boolean = false,
    size: Dp = 30.dp,
    iconSize: Dp = size / 3,
    padding: Dp = 4.dp,
    borderWidth: Dp = 1.dp,
    parentShape: RoundedCornerShape = CircleShape,
    toggleShape: RoundedCornerShape = CircleShape,
    animationSpec: AnimationSpec<Dp> = tween(durationMillis = 300),
    onClick: () -> Unit
) {
    // Control de la posicion del boton que indica el tema en el que
estamos
    val offset by animateDpAsState(
        targetValue = if (darkTheme) 0.dp else size,
        animationSpec = animationSpec, label = "animacion tema"
    )

    // Contenedor padre que da forma al contenedor exterior
    Box(
        modifier = Modifier
```

```
            .width(size * 2)
            .height(size)
            .clip(shape = parentShape)
            .clickable { onClick() }
            .background(Azul_base)
    ) {
        // Contenedor hijo que da forma al boton circular que cambia entre
temas
        Box(
            modifier = Modifier
                .size(size)
                .offset(x = offset)
                .padding(all = padding)
                .clip(shape = toggleShape)
                .background(Azul_com_1)
        )

        // Fila para los iconos de los temas claro y oscuro
        Row(
            modifier = Modifier.border(
                border = BorderStroke(
                    width = borderWidth,
                    color = Azul_com_1
                ),
                shape = parentShape
            )
        ) {
            // Contenedor del icono del tema oscuro
            Box(
                modifier = Modifier.size(size),
                contentAlignment = Alignment.Center
            ) {
                Icon(
                    modifier = Modifier.size(iconSize),
                    imageVector = Icons.Default.Nightlight,
                    contentDescription = "Icono de tema oscuro",
                    // Cambio de tonos dependiendo del tema
                    tint = if (darkTheme) Azul_base else Azul_com_1
                )
            }
```

```
            // Contenedor del icono del tema claro
            Box(
                modifier = Modifier.size(size),
                contentAlignment = Alignment.Center
            ) {
                Icon(
                    modifier = Modifier.size(iconSize),
                    imageVector = Icons.Default.LightMode,
                    contentDescription = "Icono de tema claro",
                    tint = if (darkTheme) Azul_com_1 else Azul_base
                )
            }
        }
    }
}
```

- **Fichero *"GlobalValues.kt"***

En este fichero se encuentran los diversos elementos que son utilizados de forma generalizada por los distintos módulos de la herramienta para poder realizar sus tareas y compartir la información entre ellos.

```
package com.i72pehej.cpuschedulerapp.util

import androidx.compose.runtime.mutableStateListOf
import androidx.compose.runtime.mutableStateOf
import com.i72pehej.cpuschedulerapp.R
import com.i72pehej.cpuschedulerapp.util.classes.InfoGraficoEstados
import com.i72pehej.cpuschedulerapp.util.classes.Proceso

/**
 * @author Julen Perez Hernandez
 */


// ============================================================

// Valores constantes para las animaciones
const val slideAnimationSpeed = 400
const val fadeAnimationSpeed = 300
```

```
// ================================================================

// Valores para los botones basicos comunes de la app
const val buttonCornerRadius = 50
const val buttonBorderWith = 1
const val buttonDefaultPadding = 10

// ================================================================

// Valores para la anchura de los campos del formulario
const val anchuraFormularioNombres = 180
const val anchuraFormularioTiempos = 150

// Variable para almacenar el indice del algoritmo seleccionado de la lista
de algoritmos en el formulario de Home
var selectorAlgoritmo = 0

// Variable que almacena el valor del quantum para evitar que se limpie al
cambiar de pagina
var tiempoQuantum = ""

// ================================================================

// Iconos comunes
val appIconUCOSolo = R.drawable.uco_solo
val appIconCPU1 = R.drawable.cpu_icon_1
//val appIconUCOColor = R.drawable.logos_version_uco
//val appIconCPU2 = R.drawable.cpu_icon_2
//val appIconCPU3 = R.drawable.cpu_icon_3

// ================================================================

// Lista de procesos global con la que trabajar entre pantallas
var listaDeProcesosGlobal = mutableStateListOf<Proceso>()

// Contenedor de informacion de resultados
var infoResultadosGlobal = mutableListOf<InfoGraficoEstados>()

// ================================================================

var siguienteSeleccionado = mutableStateOf(false)
```

- **Fichero “MainActivity”**

A continuación, se presenta el código correspondiente al fichero que actúa como el punto de entrada principal de la aplicación, iniciando la ejecución del resto de elementos.

```
package com.i72pehej.cpuschedulerapp

import android.content.res.Configuration
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.Surface
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import com.i72pehej.cpuschedulerapp.navigation.AppNavigation
import com.i72pehej.cpuschedulerapp.ui.theme.CpuSchedulerAppTheme

/**
 * @author Julen Perez Hernandez
 * Main activity
 *
 * @constructor Crea la actividad Main para la llamada a las funciones
iniciales
 */
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            val systemTheme =
LocalContext.current.resources.configuration.uiMode and
Configuration.UI_MODE_NIGHT_MASK
            var temaOscuro by remember { mutableStateOf(systemTheme ==
Configuration.UI_MODE_NIGHT_YES) }
```

```
            CpuSchedulerAppTheme(darkTheme = temaOscuro) {
                // A surface container using the 'background' color from
the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = Color.White
                ) {
                    // Llamamos al gestor de navegacion, que se encargara
de visualizar las pantallas en el orden seleccionado
                    AppNavigation(temaOscuro, onActualizarTema = {
temaOscuro = !temaOscuro })
                }
            }
        }
    }
}
```

Capítulo
3. Paquete de recursos

En este apartado, se explorará el "Paquete de Recursos" de la aplicación. Este componente es esencial para proporcionar una experiencia de usuario rica y atractiva, ya que almacena todos los archivos relacionados con los recursos visuales y de contenido, como imágenes, iconos, textos y valores necesarios para el funcionamiento correcto de la aplicación. Se encuentra organizado de manera eficiente y se divide principalmente en dos subcarpetas clave:

- ***"drawable"***: En esta subcarpeta, se encuentran todas las imágenes e iconos que la aplicación utiliza en su interfaz. Estos elementos visuales son fundamentales para la representación gráfica de la información y la navegación, aportando un aspecto atractivo y coherente a la aplicación, así como los diferentes iconos que identifican a la aplicación.

- ***"values"***: Dentro de esta subcarpeta, se albergan ficheros de valores esenciales para el funcionamiento de la aplicación. El fichero destacado es el que contiene las cadenas de texto utilizadas en toda la aplicación. Estas cadenas son vitales para mostrar información al usuario de manera comprensible y, al modularizarlas, se facilita su mantenimiento y su posible traducción a otros idiomas.

Se explorará en detalle el contenido y la estructura de estas subcarpetas, lo que permitirá comprender cómo se gestionan los recursos visuales y de contenido.

- **Carpetas "*drawable*"**

Tal y como se ha presentado anteriormente, en esta carpeta se encuentran todas aquellas imágenes, iconos y recursos gráficos utilizados en la aplicación, tanto para la interfaz de usuario como para identificación de la aplicación.

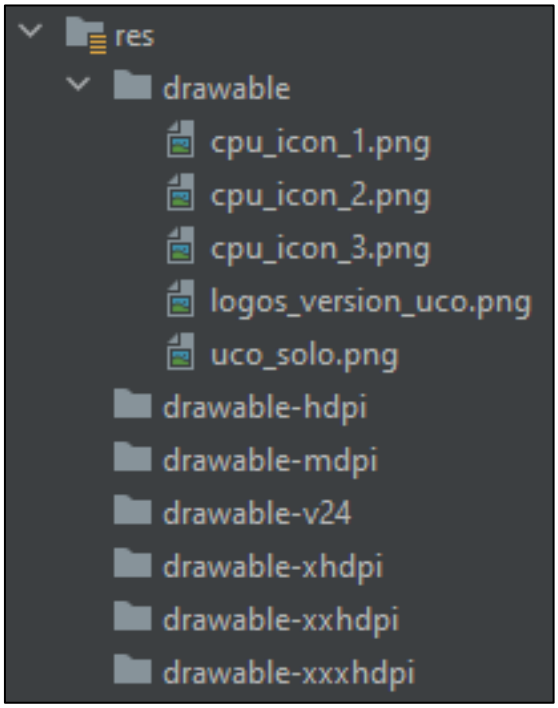

Figura 41. Conjunto de carpetas "drawable".

Todos los iconos e imágenes utilizados en este proyecto han sido obtenidos desde las páginas de distribución oficiales de la universidad o creados desde cero considerando el propósito del concepto del proyecto.

- **Carpeta "*values*"**

En este apartado se considerarán, tal y como se ha mencionado previamente, distintos ficheros de valores esenciales para el funcionamiento de la aplicación. Concretamente, se le prestará una especial atención al archivo correspondiente a los textos utilizados en la aplicación, cuyo propósito radica en la agrupación, en un mismo archivo, de todos los elementos de texto que vayan a ser requeridos. De esta manera, no sólo se consigue un incremento considerable en la facilidad de modificación de estos elementos, sino que también se agrega una enorme ventaja para poder considerar distintos idiomas evitando tener que modificar directamente las cadenas de texto en los respectivos archivos en los que son utilizados.

- ***strings.xml***

```
<resources>
    <!--    Nombres de las paginas -->
    <string name="app_name">CPUco Scheduler</string>
    <string name="results_name">Estamos en la pantalla de
RESULTADOS</string>
    <string name="graphs_name">Estamos en la pantalla de GRÁFICOS</string>
    <string name="queues_name">Estamos en la pantalla de COLAS</string>

    <!--    Strings para los elementos comunes de common -->
    <string name="common_buttonNext">Siguiente</string>

    <!--    Mensajes de formulario -->
    <string name="nuevo_proceso">Nuevo proceso</string>

    <string name="formulario_nombre">Nombre</string>
    <string name="error_nombre">Ingrese un nombre válido</string>
    <string name="error_nombre_repetido">Nombre repetido</string>

    <string name="formulario_llegada">LLegada</string>
    <string name="error_llegada_blank">Ingrese un tiempo de
llegada</string>
    <string name="error_llegada_digit">Ingrese un número entero para el
tiempo de llegada</string>

    <string name="formulario_duracion">Duración</string>
    <string name="error_duracion_blank">Ingrese una duración</string>
    <string name="error_duracion_digit">Ingrese un número entero para la
duración</string>

    <string name="formulario_quantum">Quantum</string>
    <string name="error_quantum_blank">Ingrese un quantum</string>
    <string name="error_quantum_digit">Ingrese un número entero para el
quantum</string>

    <string name="error_E_S">Ingrese un tiempo</string>
    <string name="error_EmenorS">Out debe ser mayor que In</string>

    <string name="tabla_vacia">Agregue nuevos procesos para
continuar</string>
```

```
    <!--    Mensaje de confirmacion de salida -->
    <string name="back_press_confirmacion">Pulsa de nuevo para
salir</string>

    <!--    Titulo de las columnas de la tabla de resultados -->
    <string name="nombre_tiempo_inicio">T. Inicio</string>
    <string name="nombre_tiempo_fin">T. Fin</string>
    <string name="nombre_tiempo_estancia">T. Estancia</string>
    <string name="nombre_tiempo_espera">T. Espera</string>
</resources>
```