ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

TRABAJO DE FIN DE GRADO

- Manual del Código -

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 Contratado Doctor 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, junio 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

I. Introducción..... - 1 -

II. Paquete principal ..... - 3 -

2.1. Carpeta “*navigation*” ..... - 4 -

2.2. Carpeta “*ui/theme*” ..... - 12 -

2.3. Carpeta “*usecases*” ..... - 14 -

2.4. Carpeta “*util*”..... - 43 -

2.5. Fichero “*MainActivity*” ..... - 63 -

III. Paquete de recursos..... - 65 -

3.1. Carpetas “*drawable*” ..... - 66 -

3.2. Carpeta “*values*”..... - 66 -

Índice de Figuras

Figura 1. Conjunto de carpetas "drawable". ..... - 66 -

Capítulo
I. 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
II. 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.

2.1. 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.

2.1.1. *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
            )
        }
    }
}
```

2.1.2. *AppNavigationAnimations.kt*

```kotlin
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()
        }
    )
}
```

2.1.3. *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")
}
```

2.1.4. *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
    }
}
```

2.2. 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.

2.2.1. *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)
```

2.2.2. *Theme.kt*

```kotlin
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
    )
}
```

2.3. 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.

2.3.1. 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> {}
```

2.3.2. 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)
                    }
                }
            }
        }
    )
}
```

2.3.3. 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
        }
    }
```

2.3.4. 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
            )
        }
    }
}
```

2.3.5. 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))
            }
        }
    }
}
```

2.4. 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.

2.4.1. 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*

```
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
    }
}
```

2.4.2. 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*

```kotlin
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*

```
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
                )
            }
        }
    }
}
```

2.4.3. 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)
```

2.5. 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
III. 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.

3.1. 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.

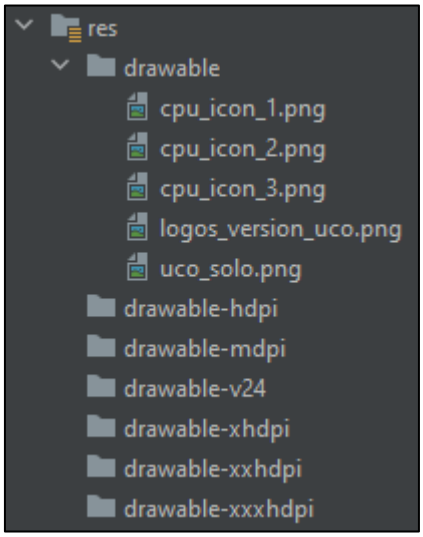

Figura 1. Conjunto de carpetas "drawable".

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.

3.2. 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.

3.2.1. *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>
```