

Estrategia de Pruebas

La presente estrategia de pruebas corresponde a la Iteración 1 del desarrollo de la aplicación de Vinilos. Para efectos de este desarrollo se cuenta con el back desplegado en [Heroku](#) el cual no estará sujeto a modificaciones. El proceso de desarrollo se centra en el Front de la aplicación.

1. Aplicación Bajo Pruebas

1.1. Nombre de la aplicación: Vinilos

1.2. Versión:1.0

1.3. Descripción:

Se quiere desarrollar una aplicación que proporcione al coleccionista aficionado la posibilidad de adquirirlos nuevos o usados, ya sea mediante compra o intercambio.

Se espera que por medio de este sistema los usuarios visitantes y los coleccionistas puedan conocer los vinilos disponibles. Por cada uno, la información de los músicos, las canciones y, para los artistas, los premios que el artista ha recibido (si es que tiene alguno). Adicionalmente, un coleccionista debe poder comprar, vender o intercambiar álbumes.

Un coleccionista puede ser comprador, vendedor o ambos. La información del coleccionista debe incluir su nombre, información de contacto (teléfono y correo electrónico) la cual es privada para el sistema, y sus artistas favoritos

La funcionalidad principal de la aplicación es permitir que un coleccionista, que cumpla el rol de vendedor, pueda registrar los álbumes que tiene para vender o hacer trueque; y quien cumpla el rol de comprador, pueda realizar la compra o el intercambio.

La aplicación debe ser muy llamativa para los usuarios e incluir una galería en donde sea fácil filtrar y buscar por distintos criterios como el nombre del album, del artista, de la casa discográfica o del género.

Cada álbum debe caracterizarse por un nombre, la imagen de la carátula, la fecha de salida al mercado, una descripción, el género, la casa discográfica, el artista (o lista de artistas) y el listado de tracks. Los usuarios registrados en la aplicación pueden hacer comentarios sobre los álbumes.

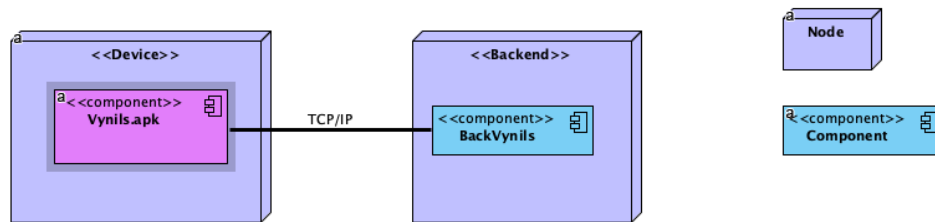
Del artista (que puede ser un músico o una banda) se requiere información como su nombre, una fotografía y un texto con una breve descripción. Si es una banda se debe conocer la fecha de formación, mientras que si es un músico se requiere su fecha de nacimiento.

También es importante conocer los premios que ha recibido el artista y la organización que otorga los premios.

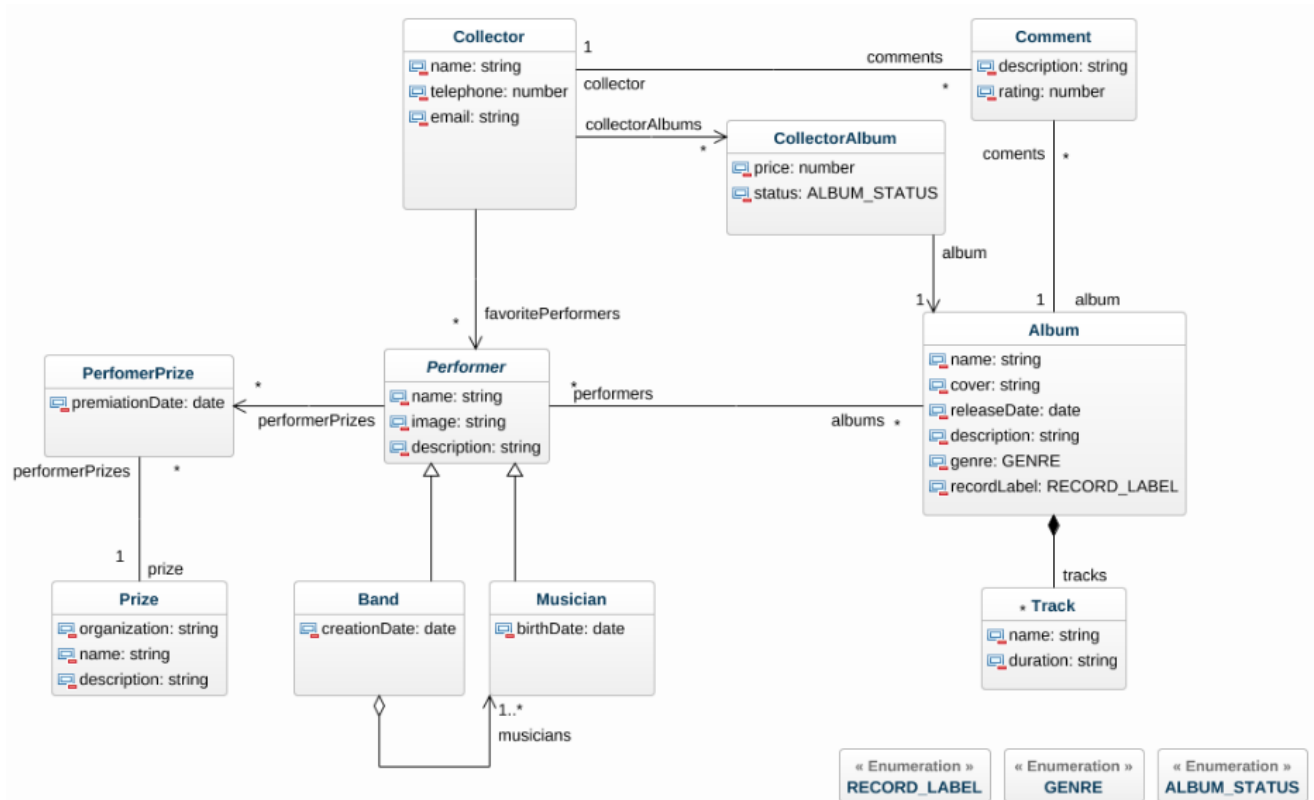
1.4. Funcionalidades Core:

- Albums
- Bands
- Collectors
- Comments
- Musicians
- Prices
- Tracks

1.5. Diagrama de Arquitectura (despliegue):



1.6. Diagrama de Datos:



1.7. Diseño UI/UX:

A continuación, se presenta el link donde se puede consultar y visualizar el [Diseño UI/UX](#)

2. Contexto de la estrategia de pruebas

2.1. Objetivos:

Para la presente estrategia de definieron los siguientes objetivos de acuerdo con las historias de usuario incluidas en el alcance de esta iteración:

Historias de Usuario:

- Consultar Catálogo de Álbumes
- Consultar Información Detalla de Álbum
- Consultar listado de Artistas
- Consultar información detalla de Artistas
- Consultar listado de coleccionistas
- Crear Premio

Objetivos:

- Validar que cada uno de los requerimientos de los usuarios son tenidos en cuenta y desarrollados de la mejor forma (validación interna).
- Validar que cada una de las historias de usuario desarrolladas cumple con la tarea definida por los usuarios (validación externa).
- Garantizar la calidad de cada una de las funcionalidades del sistema implementando herramientas para mitigar los errores en desarrollo.
- Validar que se está construyendo el software a la medida de los requisitos de los usuarios y que cada una de las funcionalidades cumple con lo estipulado (validación interna).
- Verificar que el software construido cumple con las especificaciones de diseño y sigue estándares internos aplicando las buenas prácticas adoptadas.
- Analizar de forma estática cada uno de los componentes implementados mediante revisiones de código por parte de revisores pares del equipo de desarrollo.
- Analizar los escenarios de las pruebas manuales basados en los requerimientos y criterios de aceptación iniciales, garantizando que el sistema cumple con lo establecido.
- Implementar pruebas manuales en cada una de las funcionalidades basándose en los criterios de aceptación (AAT).
- Verificar el correcto funcionamiento de las historias de usuario desarrolladas mediante pruebas automatizadas creadas en Espresso.
- Estimar el desempeño de la aplicación haciendo uso de la funcionalidad *Profiler* disponible en Android Studio.
- Identificar y corregir problemas de calidad del código mediante un análisis estático del código realizado con la herramienta *Lint* disponible en Android Studio.

2.2. Duración de la iteración de pruebas:

En la primer semana se establecen tiempos para el diseño y las pruebas automatizadas en Espresso para las historias: HU01, HU02, HU03, HU04 y HU05.

	Mon, May 1 00:00:00	Tue, May 2 02:00:00	Wed, May 3 02:00:00	Thu, May 4 02:00:00	Fri, May 5 02:00:00	Sat, May 6 02:00:00	Sun, May 7 00:00:00
15:00							
16:00							
17:00							
18:00		HU01:Diseño Prueba Automatizada (Without project) 01:00:00	HU02: Diseño Pruebas Automatizadas (Without project) 01:00:00	HU03: Diseño Pruebas Automatizadas (Without project) 01:00:00	HU04: Diseño Pruebas Automatizadas (Without project) 01:00:00	HU05: Diseño Pruebas Automatizadas (Without project) 01:00:00	
19:00		HU01:Ejecución Pruebas Automatizadas 01:00:00	HU02: Ejecución Pruebas Automatizadas 01:00:00	HU03: Ejecución Pruebas Automatizadas 01:00:00	HU04:Ejecución Pruebas Automatizadas 01:00:00	HU05:Ejecución Pruebas Automatizadas 01:00:00	
20:00							

En la segunda semana se realiza el diseño y la ejecución de las pruebas automatizadas de la historia HU06. Posteriormente, se realizan pruebas de desempeño para el código base haciendo uso de la herramienta Profiler. Se realiza un Análisis Estático haciendo uso de Lint y con base en los resultados obtenidos en estas pruebas se realizan micro-optimizaciones en el código. Una vez se realicen las micro-optimizaciones en el código se procede a ejecutar nuevamente las pruebas de desempeño, esta vez en la versión optimizada de la aplicación.

	Mon, May 8 01:58:00	Tue, May 9 01:00:00	Wed, May 10 02:00:00	Thu, May 11 01:00:00	Fri, May 12 00:00:00	Sat, May 13 00:00:00	Sun, May 14 00:00:00
14:00							
15:00							
16:00							
17:00							
18:00	HU06:Diseño Pruebas Automatizadas (Without project) 00:58:00	Pruebas de desempeño base (Profiler) (Without project) 01:00:00	Análisis estático (Lint) (Without project) 01:00:00	Pruebas de desempeño post Optimización (Without project) 01:00:00			
19:00	HU06:Ejecución Pruebas Automatizadas 01:00:00		Micro-Optimizaciones (Without project) 01:00:00				
20:00							

2.3. Presupuesto de pruebas:

2.3.1. Recursos Humanos

Role	Experiencia Previa	Tiempo disponible
Desarrollador	– Experiencia en desarrollo de aplicaciones para backend en lenguajes Java, C#, Delphi y para frontend en Angular y Typescript, manejo de bases de datos relacionales SQL y Oracle, y experiencia en ambientes de nube con AWS.	4H
Desarrollador	– Experiencia en desarrollo Backend en lenguajes como Java, C#, .Net Core y Python, manejo de bases de datos como MySQL, PL/SQL, DB2 y PostgreSQL	4H
Desarrollador	– Experiencia en desarrollo de software con Typescript y Java, y del ciclo del software en general.	4H
Desarrollador	– Sin experiencia previa	4H

2.3.2. Recursos Computacionales

Dispositivos	Objetivo	Capacidad Computacional	Tiempo computacional
Laptop Personal	Ejecución de Pruebas Manuales	– Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, 2304 Mhz, 4 procesadores principales, 8 procesadores lógicos	4 horas
Laptop Personal	Ejecución de Pruebas Manuales	– Processor 2,6 GHz 6-Core Intel Core i7 – Graphics Intel UHD Graphics 630 1536 MB – Memory 16 GB 2667 MHz DDR4 – MscOS Ventura 13.3.1	4 horas
Laptop Personal	Ejecución de Pruebas Manuales	– Processor 2,6 GHz 6-Core Intel Core i7 de seis núcleos – Graphics AMD Radeon Pro 5300M 4 – Intel UHD Graphics 630 1536 MB – Memory 16 GB 2667 MHz DDR4 – MscOS Ventura 13.3.1 –	4 horas
Laptop Personal	Ejecución de Pruebas Manuales	– Processor 2,6 GHz 6-Core Intel Core i7 – Graphics Intel UHD Graphics 630 1536 MB – Memory 16 GB 2667 MHz DDR4 – MscOS Ventura 13.3.1	4 horas

2.4. TNT (Técnicas, Niveles y Tipos) de pruebas:

A continuación, se especifican las técnicas, niveles y tipos de pruebas que serán implementadas en el presente presupuesto de pruebas:

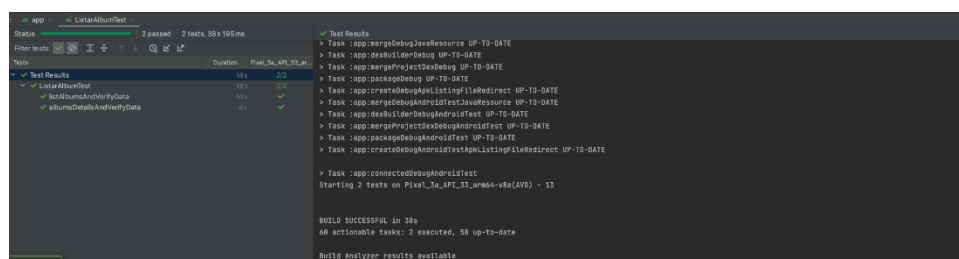
Técnica	Nivel	Tipo	Objetivo
Pruebas Automatizadas	Sistema	Funcional/Positiva	Validar que cada uno de los requerimientos de los usuarios son tenidos en cuenta y desarrollados de la mejor forma (validación interna).
Profiler	Sistema	No Funcional	Estimar el desempeño de la aplicación haciendo uso de la funcionalidad <i>Profiler</i> disponible en Android Studio.
Lint	Sistema	No Funcional	Identificar y corregir problemas de calidad del código mediante un análisis estático del código realizado con la herramienta <i>Lint</i> disponible en Android Studio.

3. Resultados

3.1. Resultados Pruebas Espresso

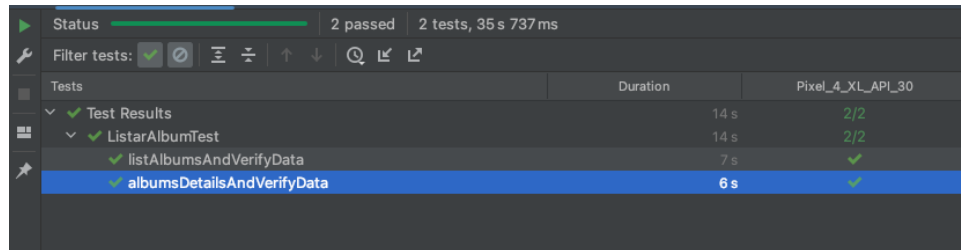
HU01- Consultar catálogo de álbumes

Se implementa prueba para listar los álbumes, se realiza E2E que navega a la lista de álbumes y verifica que se encuentre el total de álbumes y se evalúan atributos específicos de los álbumes presentes en la lista.



HU02 – Detalle del álbum

Se implementan las pruebas sobre la lista de los detalles del álbum. Las pruebas se realizan E2E de tal forma que la prueba inicia en la pantalla principal y navega hasta el detalle del álbum seleccionado, una vez en esta pantalla valida que los datos del álbum corresponden a los que se parametrizaron en la prueba.

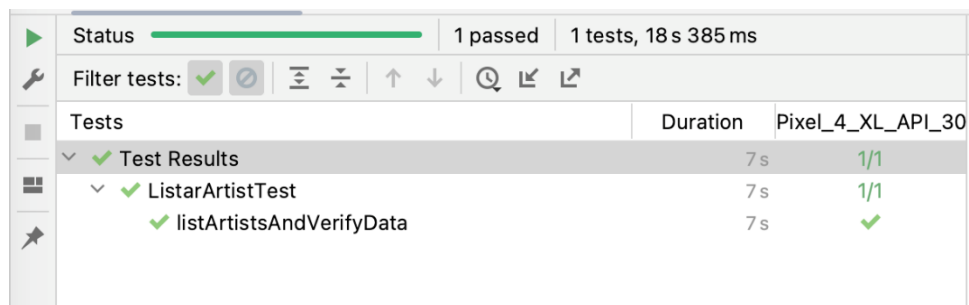


A screenshot of a test runner interface showing the results for HU02. The status bar at the top indicates '2 passed' and '2 tests, 35 s 737 ms'. The test list on the left shows a tree structure with 'Test Results' expanded, containing 'ListarAlbumTest', which in turn contains 'listAlbumsAndVerifyData' and 'albumsDetailsAndVerifyData'. The 'albumsDetailsAndVerifyData' test is highlighted in blue. The table on the right shows the duration and pixel density for each test.

Tests	Duration	Pixel_4_XL_API_30
Test Results	14 s	2/2
ListarAlbumTest	14 s	2/2
listAlbumsAndVerifyData	7 s	✓
albumsDetailsAndVerifyData	6 s	✓

HU03- Consultar el listado de artistas

Se implementa las pruebas para la funcionalidad del listado de artistas. Las pruebas E2E se realizan de tal forma que permita consultar el listado de los artistas que llegan de la consulta al API del backend, y según la cantidad de artistas que lleguen en la consulta se procede a pintar los ítems de los artistas. Esta prueba valida el contenido de la consulta y valida que sea consistente con los parámetros indicados en la prueba.

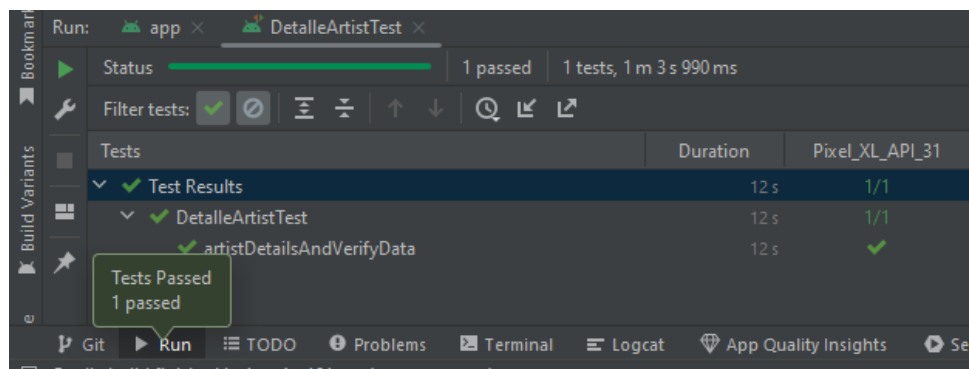


A screenshot of a test runner interface showing the results for HU03. The status bar at the top indicates '1 passed' and '1 tests, 18 s 385 ms'. The test list on the left shows a tree structure with 'Test Results' expanded, containing 'ListarArtistTest', which in turn contains 'listArtistsAndVerifyData'. The 'listArtistsAndVerifyData' test is highlighted. The table on the right shows the duration and pixel density for each test.

Tests	Duration	Pixel_4_XL_API_30
Test Results	7 s	1/1
ListarArtistTest	7 s	1/1
listArtistsAndVerifyData	7 s	✓

HU04- Consultar la información detallada de un artista

Se diseñan y ejecutan pruebas sobre el detalle de los artistas. Las pruebas se realizan E2E de tal forma que la prueba inicia en la pantalla principal y navega hasta el detalle del artista seleccionado. En la pantalla de detalle de artista se valida que el contenido de dicha pantalla sea consistente con los parámetros indicados en la prueba.

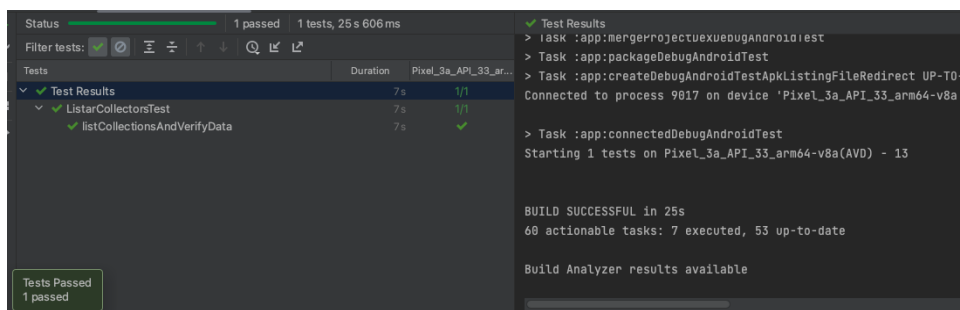


A screenshot of a test runner interface showing the results for HU04. The status bar at the top indicates '1 passed' and '1 tests, 1 m 3 s 990 ms'. The test list on the left shows a tree structure with 'Test Results' expanded, containing 'DetalleArtistTest', which in turn contains 'artistDetailsAndVerifyData'. The 'artistDetailsAndVerifyData' test is highlighted. The table on the right shows the duration and pixel density for each test. A tooltip 'Tests Passed 1 passed' is visible over the 'Run' button.

Tests	Duration	Pixel_XL_API_31
Test Results	12 s	1/1
DetalleArtistTest	12 s	1/1
artistDetailsAndVerifyData	12 s	✓

HU05- Consultar listado de coleccionistas

Se implementa prueba para listar los coleccionistas, se realiza E2E que navega a la lista de coleccionistas y verifica que se encuentre el total de coleccionistas y se evalúan atributos específicos de los coleccionistas presentes en la lista.



HU06- Crear Premio

Se implementa prueba automatizada para crear premio en la aplicación. Esta funcionalidad hace parte del desarrollo de la historia de usuario HU06. En la prueba, se recorre la página o actividad y se encuentra coincidencia con los elementos que en ella se encuentran, tales como nombres de los campos de texto que ingresa el usuario como también el nombre de los botones de interacción para crear premio o cancelar el proceso.

Status	1 passed	1 tests, 18 s 579 ms
Filter tests:		
Tests	Duration	Pixel_4_XL_API_30
Test Results	7 s	1/1
CrearPremioTest	7 s	1/1
createPrizeAndVerifyData	7 s	✓

Total - Resultados de todas las pruebas hasta el momento

A continuación, se muestra el resultado de todas las pruebas que se han implementado hasta el momento en toda la aplicación.

Tests	Duration	Pixel_4_XL_API_30
Test Results	50 s	15/15
CrearPremioTest	8 s	1/1
createPrizeAndVerifyData	8 s	✓
DetalleArtistTest	6 s	1/1
artistDetailsAndVerifyData	6 s	✓
ExampleInstrumentedTest	55 ms	1/1
useAppContext	55 ms	✓
LayoutTest	9 s	8/8
checkAlbumDescriptionDisplayed	1 s	✓
checkArtistNameDisplayed	1 s	✓
checkArtistCoverIsDisplayed	1 s	✓
checkAlbumCoverIsDisplayed	1 s	✓
checkCollectorNameDisplayed	1 s	✓
checkAlbumNameDisplayed	1 s	✓
checkArtistDescriptionDisplayed	1 s	✓
createPrizesDisplayed	1 s	✓
ListarAlbumTest	12 s	2/2
listAlbumsAndVerifyData	6 s	✓
albumsDetailsAndVerifyData	6 s	✓
ListarArtistTest	6 s	1/1
listArtistsAndVerifyData	6 s	✓
ListarCollectorsTest	6 s	1/1
listCollectionsAndVerifyData	6 s	✓

3.2. Implementación de Corrutinas

De acuerdo con la estrategia formulada por el equipo se determinó que la implementación de corrutinas se realizaría en las tareas encargadas de consumo de datos del API. A continuación, se presentan algunos fragmentos de código donde se evidencia esta implementación.

```
suspend fun fetchAlbums(): String = suspendCancellableCoroutine { continuation ->
    val request = ApiService.getRequest( path: "albums",
        { response -> continuation.resume(response) },
        { error -> continuation.resumeWithException(error) }
    )
    apiService.instance.add(request)

    continuation.invokeOnCancellation { it: Throwable?
        request.cancel()
    }
}
```

```
suspend fun fetchArtists(): String = suspendCancellableCoroutine { continuation ->
    val request = ApiService.getRequest( path: "musicians",
        { response -> continuation.resume(response) },
        { error -> continuation.resumeWithException(error) }
    )
    apiService.instance.add(request)

    continuation.invokeOnCancellation { it: Throwable?
        request.cancel()
    }
}
```

```
suspend fun fetchCollectors(): String = suspendCancellableCoroutine { continuation ->
    Log.d( tag: "API", msg: "Making API call to fetch collectors")
    val request = ApiService.getRequest( path: "collectors",
        { response ->
            Log.d( tag: "API", msg: "API call successful $response")
            continuation.resume(response)
        },
        { error ->
            Log.e( tag: "API", msg: "API call failed $error", error)
            continuation.resumeWithException(error)
        }
    )
    apiService.instance.add(request)

    continuation.invokeOnCancellation { it: Throwable?
        request.cancel()
    }
}
```

```

suspend fun postPrize(prize: JSONObject): String = suspendCancellableCoroutine { continuation ->
    val request = ApiService.postRequest( path: "prizes", prize,
        { response -> continuation.resume(response.toString()) },
        { error -> continuation.resumeWithException(error) }
    )
    apiService.instance.add(request)

    continuation.invokeOnCancellation { it: Throwable?
        request.cancel()
    }
}

```

3.3. Resultados de Micro-optimización

3.3.1. Implementación de patrón singleton, correcciones en la formación de las URLs y mejor manejo de errores

Patrón Singleton: La clase ApiService actualmente crea una nueva RequestQueue cada vez que es instanciada. Es una buena práctica seguir el patrón Singleton para este tipo de clase de servicio. De esta manera, sólo se crea una instancia de RequestQueue que se reutiliza en toda la aplicación. Esto puede ahorrar memoria y permitir una gestión centralizada de las peticiones.

Construcción de URL: Es mejor utilizar la clase Uri.Builder para construir URLs. Reduce el riesgo de URLs malformadas y hace el proceso de construcción de URLs más flexible y legible.

Gestión de errores: El ErrorListener de Volley sólo proporciona un VolleyError, pero no indica qué petición ha fallado. Para mejorar la depuración, podemos envolver el error listener en otra función lambda para proporcionar más contexto sobre el error.

3.3.2. Lint y Refactoring

Se realizaron cambios sobre el código una vez que se hizo la inspección del código a través de la herramienta que ofrece Android Studio. La inspección del código arrojó warnings sobre archivos que contenían fragmentos de código sin utilizar y archivos incompatibles con la versión de la aplicación.

```

▼ Performance 21 warnings
  > Invalidating All RecyclerView Data 5 warnings
  > Obsolete SDK_INT Version Check 1 warning
  > Unused resources 15 warnings

```

Adicionalmente se hizo una inspección de forma manual en todos los archivos del proyecto y se eliminaron imports sin usar, fragmentos de código comentado y funciones que no tenían ningún llamado. El detalle de el refactor se puede consultar en el siguiente link, correspondiente al Pull Request

https://github.com/Helena-Pat29/IngSoft_Appmoviles_CSHS/pull/48/files

3.4. Buenas prácticas consumo de memoria

Cache Manager

Con la nueva implementación de la cache manager antes de realizar la petición del NetworkServiceAdapter, comprobamos si los datos ya están disponibles en la caché. Si lo están, los devolvemos inmediatamente. Si no, hacemos la petición de red como siempre, pero también almacenamos la respuesta en la caché antes de devolverla.

Es importante tener en cuenta que estrategia de almacenamiento en caché sólo almacena datos en memoria, por lo que la caché se borrará cuando se cierre la aplicación o si el sistema decide recuperar memoria. La estrategia aplica para todos los metodos Get actualmente implementados.

3.5. Análisis de desempeño (Resultados del Perfilamiento)

3.5.1. Captura de Datos

Para poder establecer esta comparación, las instrucciones a seguir con ambos proyectos son las siguientes:

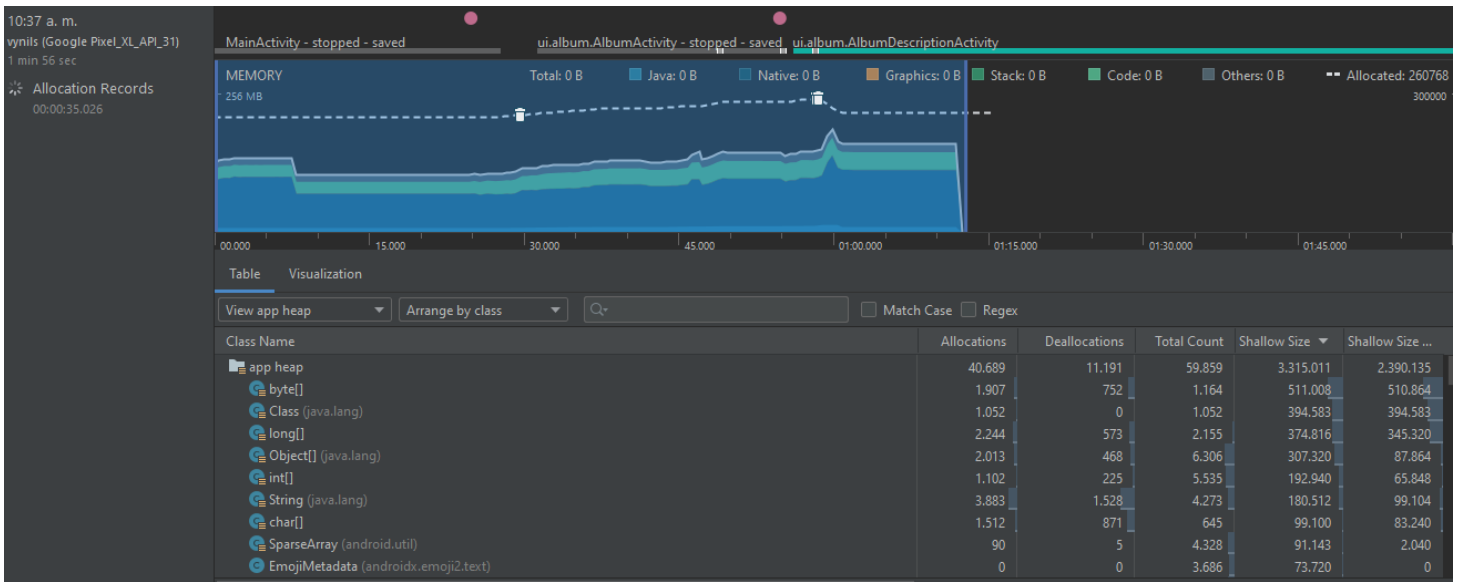
- Ejecutar el perfilamiento con el botón "*profile 'app'*". En caso de que la información de los coleccionistas no cargue al inicio, volver a presionar este botón.
- Seleccionar *Listar Albumes*. Esperar un tiempo a que el *profiler* registre el evento del clic y la creación del fragmento nuevo.
- Seleccionar el primer álbum. Esperar un tiempo a que se registre el evento y la interfaz muestre el comentario.

Se lleva un registro del tiempo en que ejecutó cada evento.

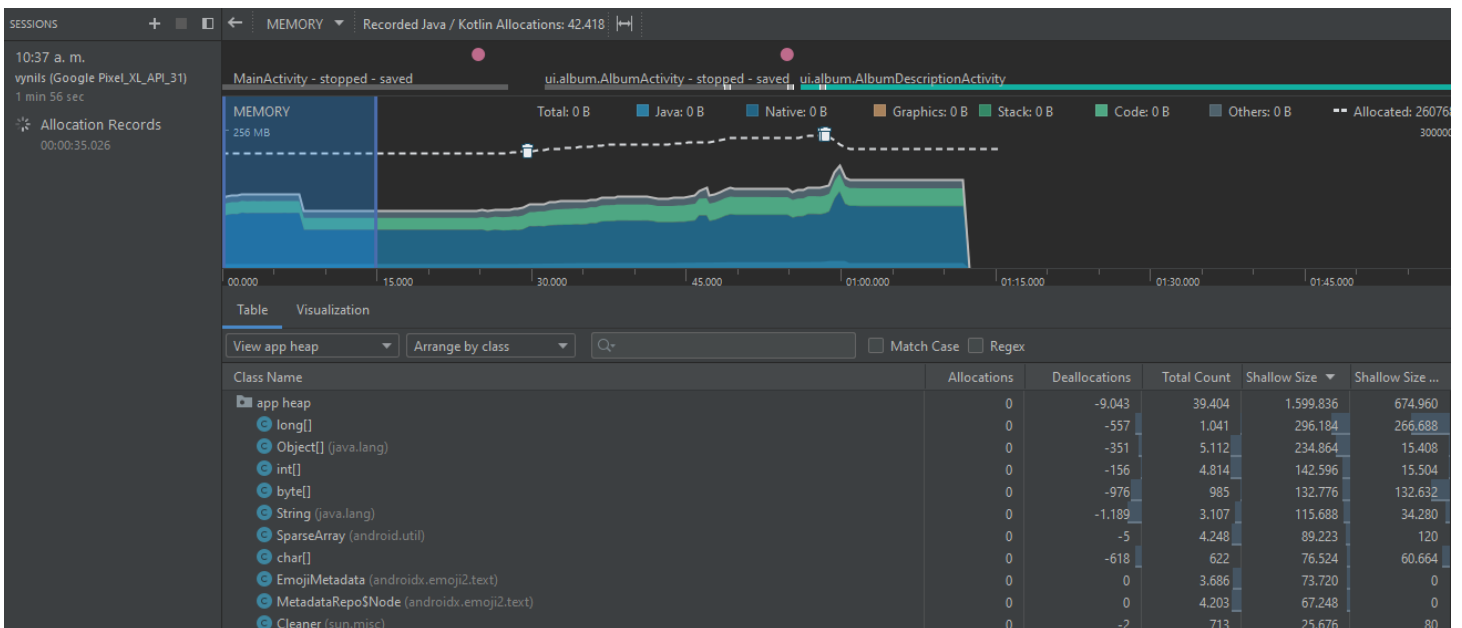
➤ **Perfilar Rendimiento de la Aplicación – Previo Micro Optimizaciones**

Para capturar el perfil de rendimiento de la app en el estado inicial se ejecutaron las pruebas desde la rama `feature/HU06_frontend` ya que en esta rama se tenía el desarrollo completo de la Iteración #2.

Consumo de Memoria durante todo el test.



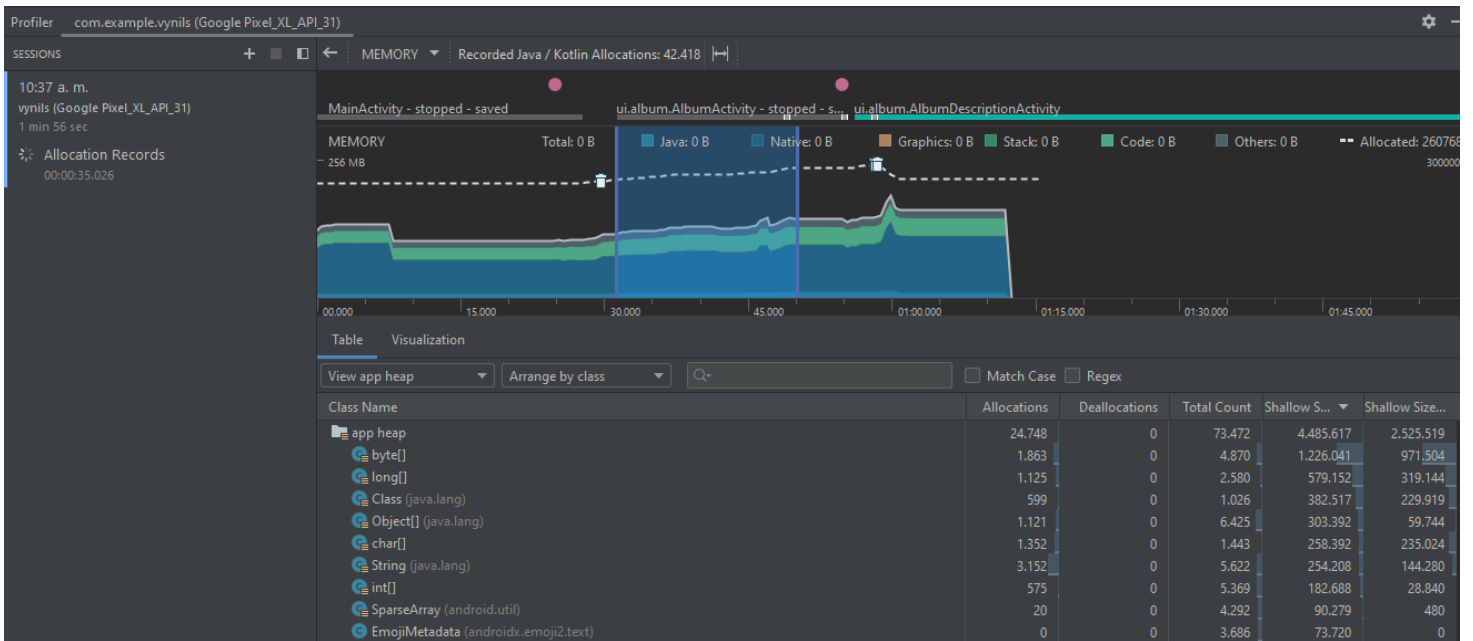
Consumo de Memoria Main Activity.



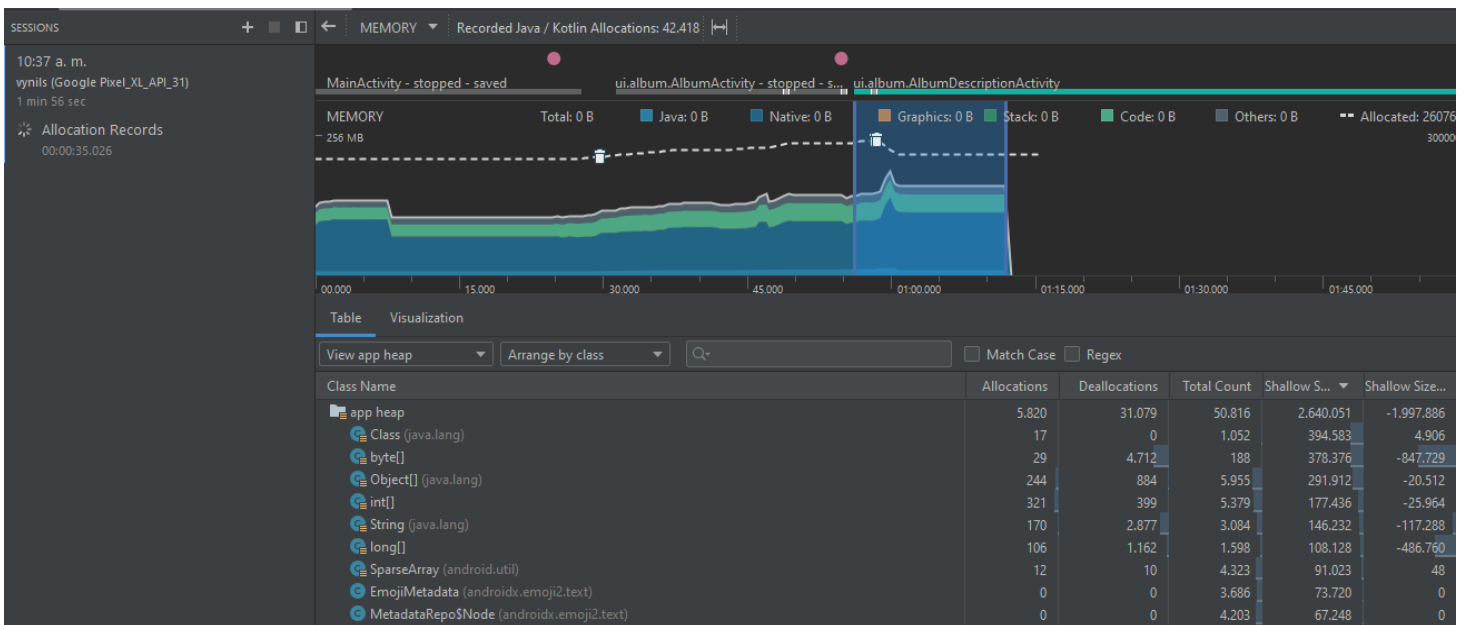
Plantilla elaborada por

THE SW DESIGN LAB

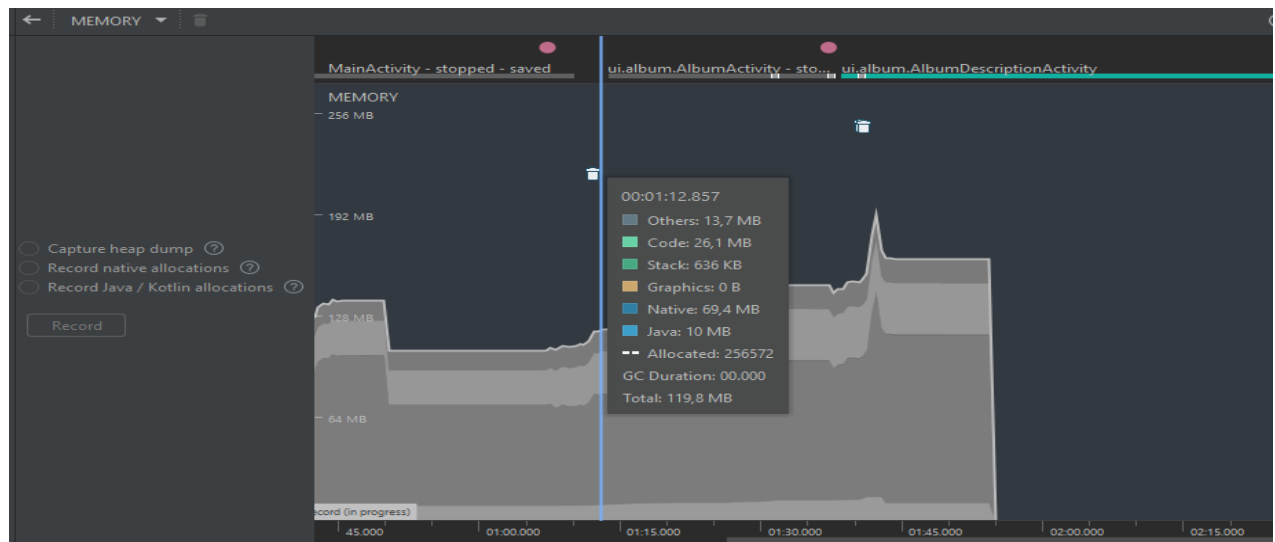
Consumo de Memoria Album List Activity.



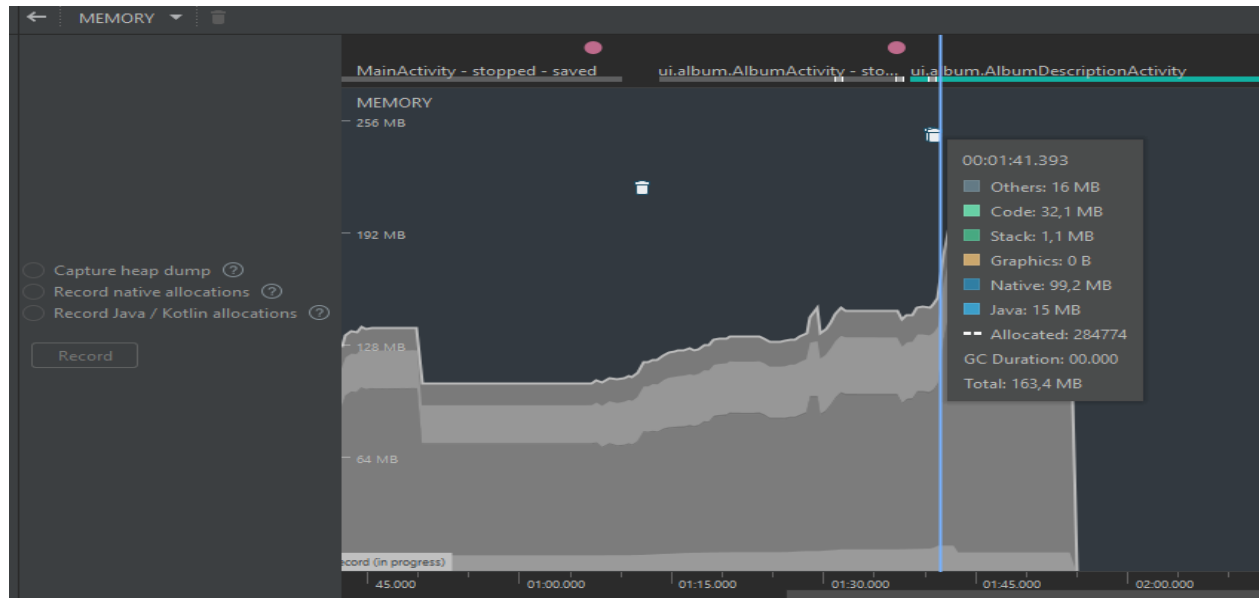
Consumo de Memoria Album Detail Activity.



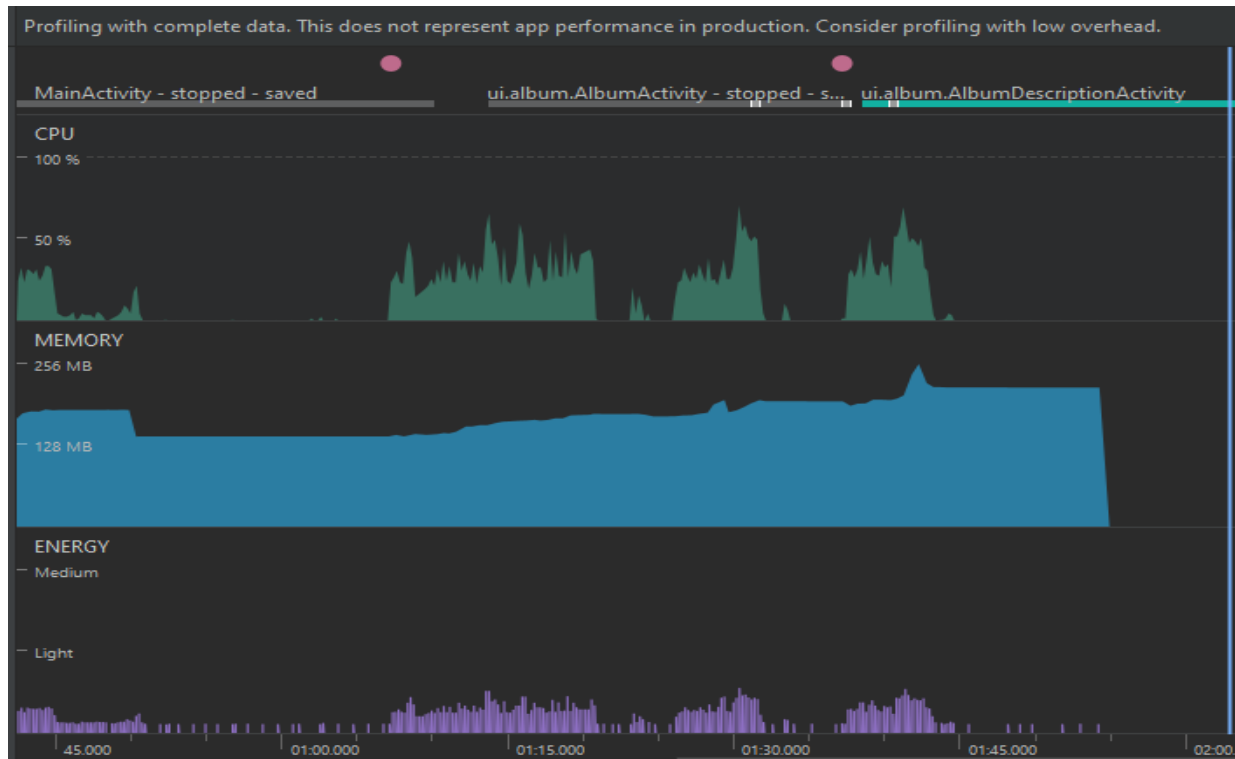
Porcentajes Consumo de Memoria – Inicio Album Activity.



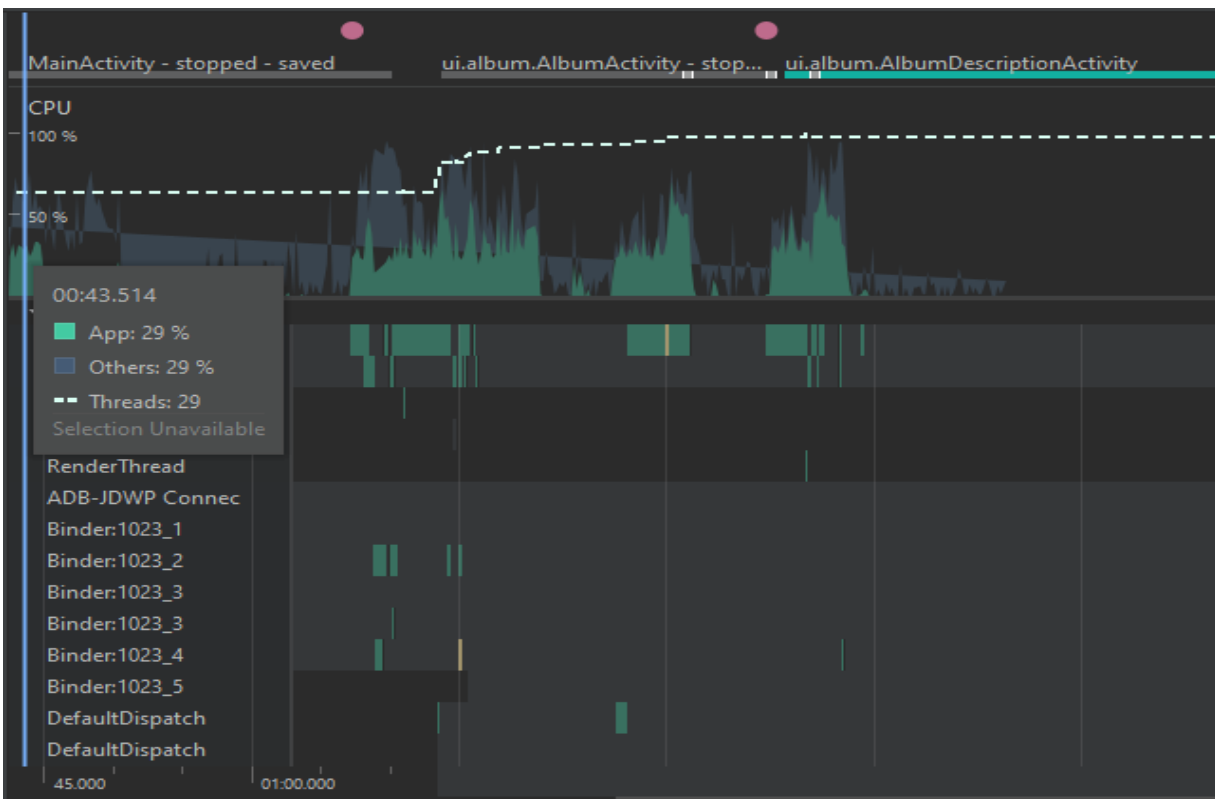
Porcentajes Consumo de Memoria – Inicio Album Detail Activity.



Data Completa Profiler.



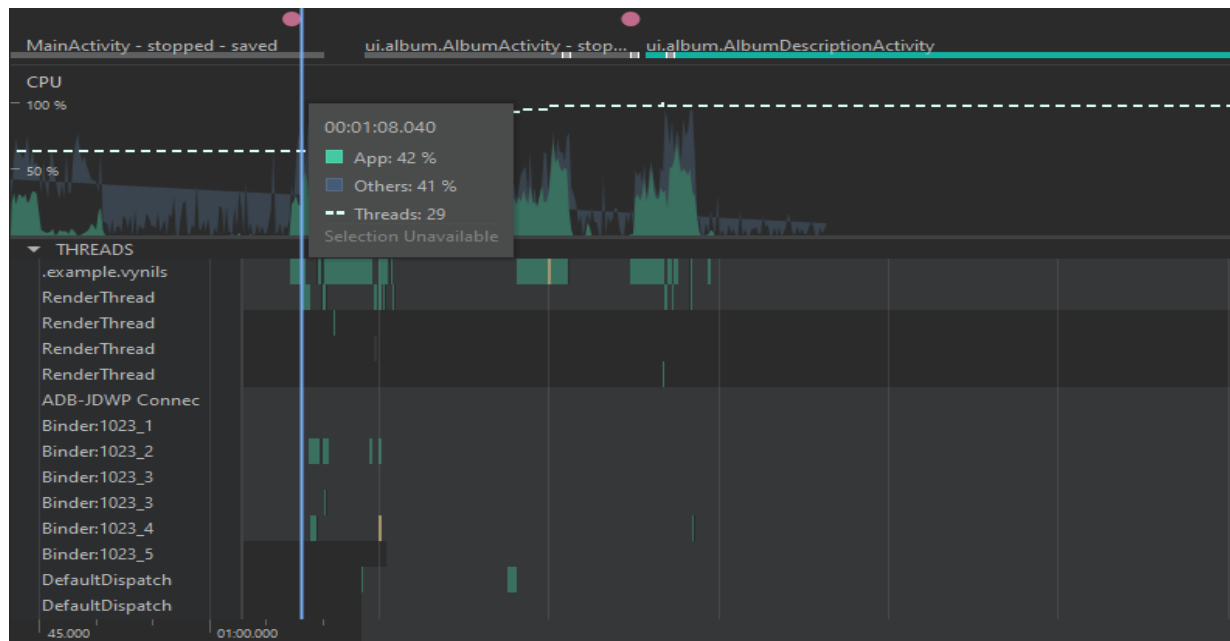
Consumo CPU –Inicio Main Activity.



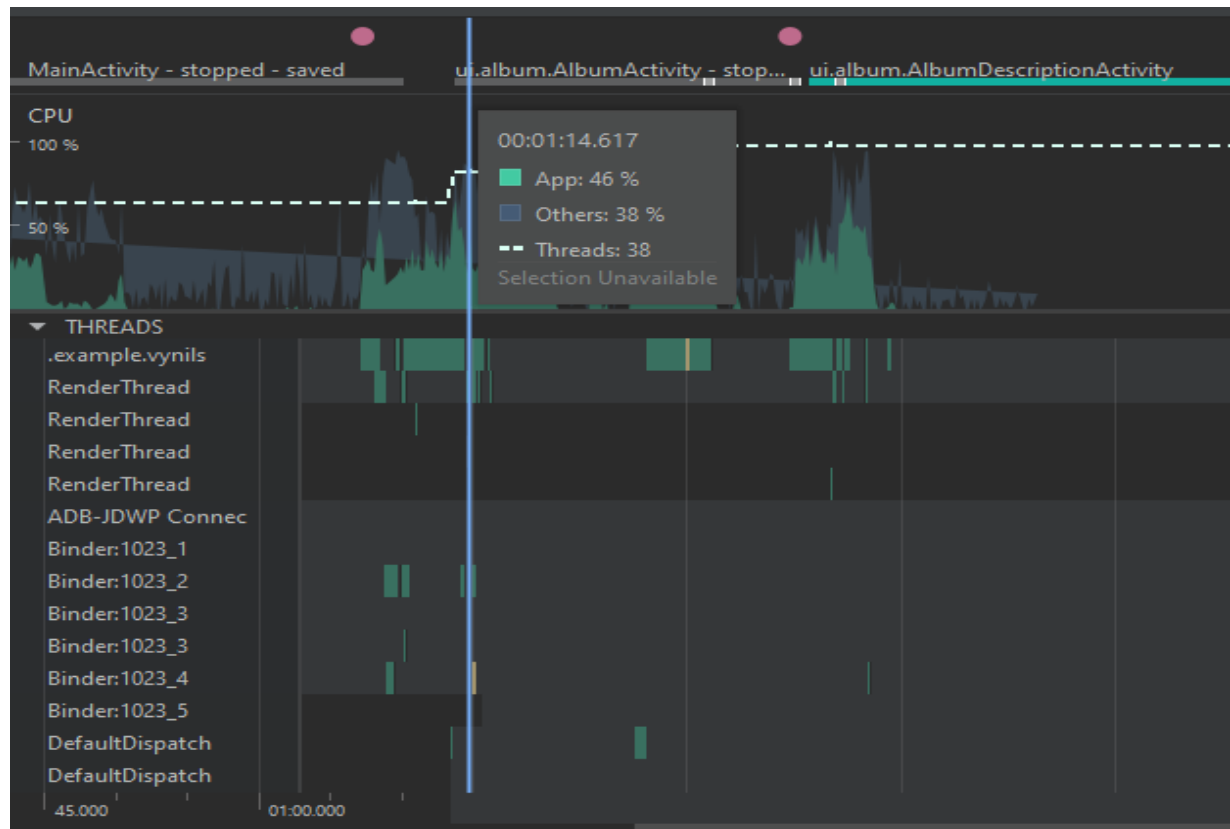
Plantilla elaborada por

THE SW DESIGN LAB

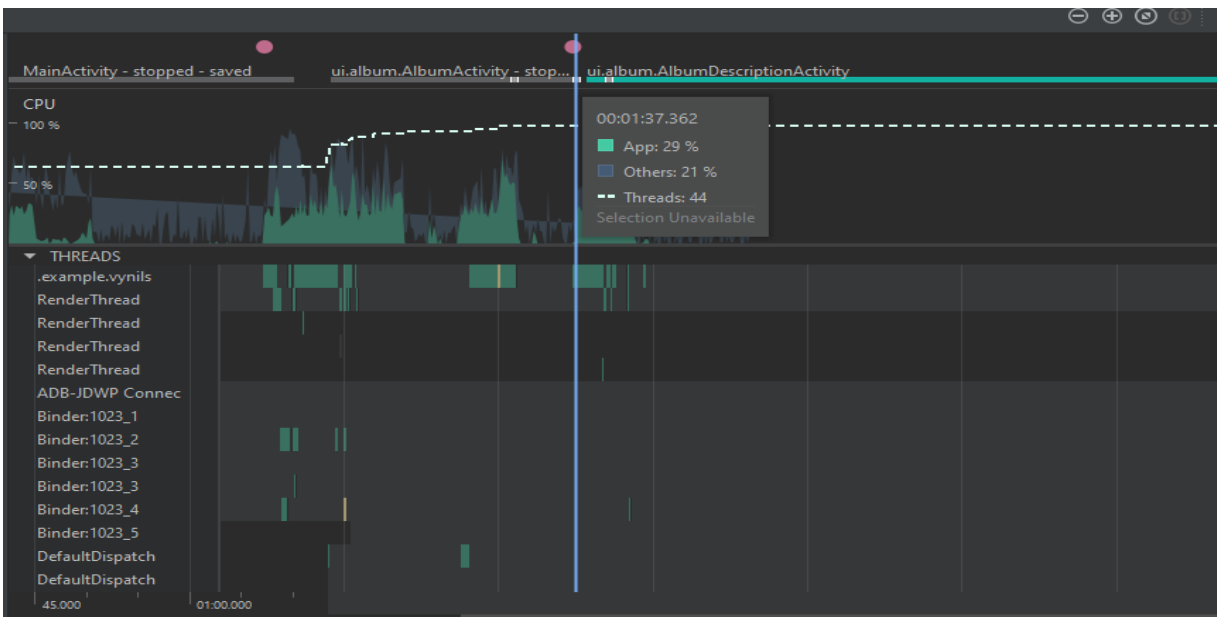
Consumo CPU –Finalización Main Activity.



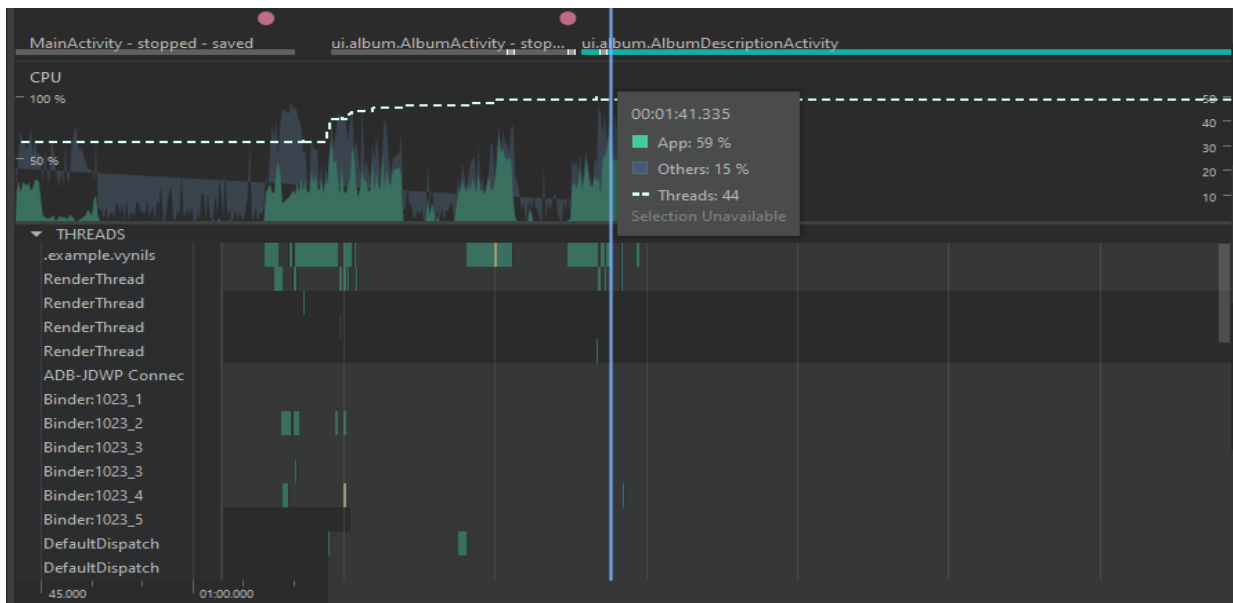
Consumo CPU –Inicio Album Activity.



Consumo CPU –Finalización Album Activity.



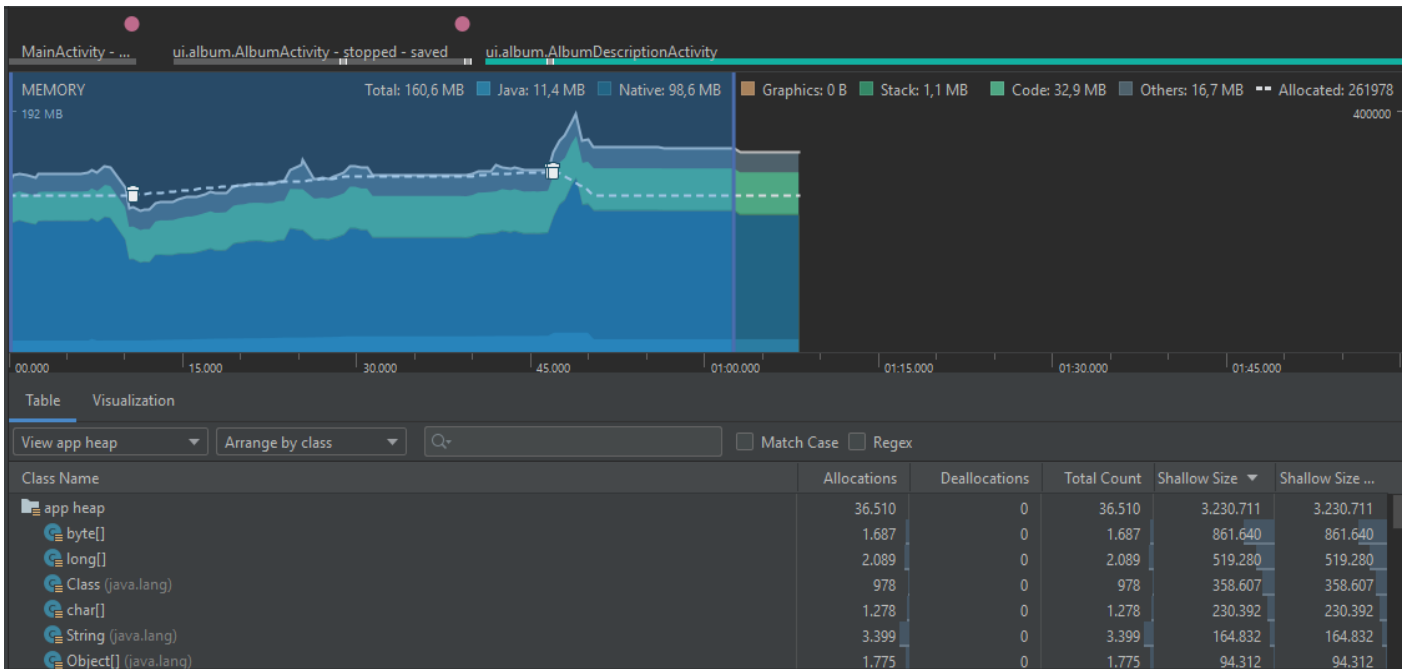
Consumo CPU –Inicio Album Detail Activity.



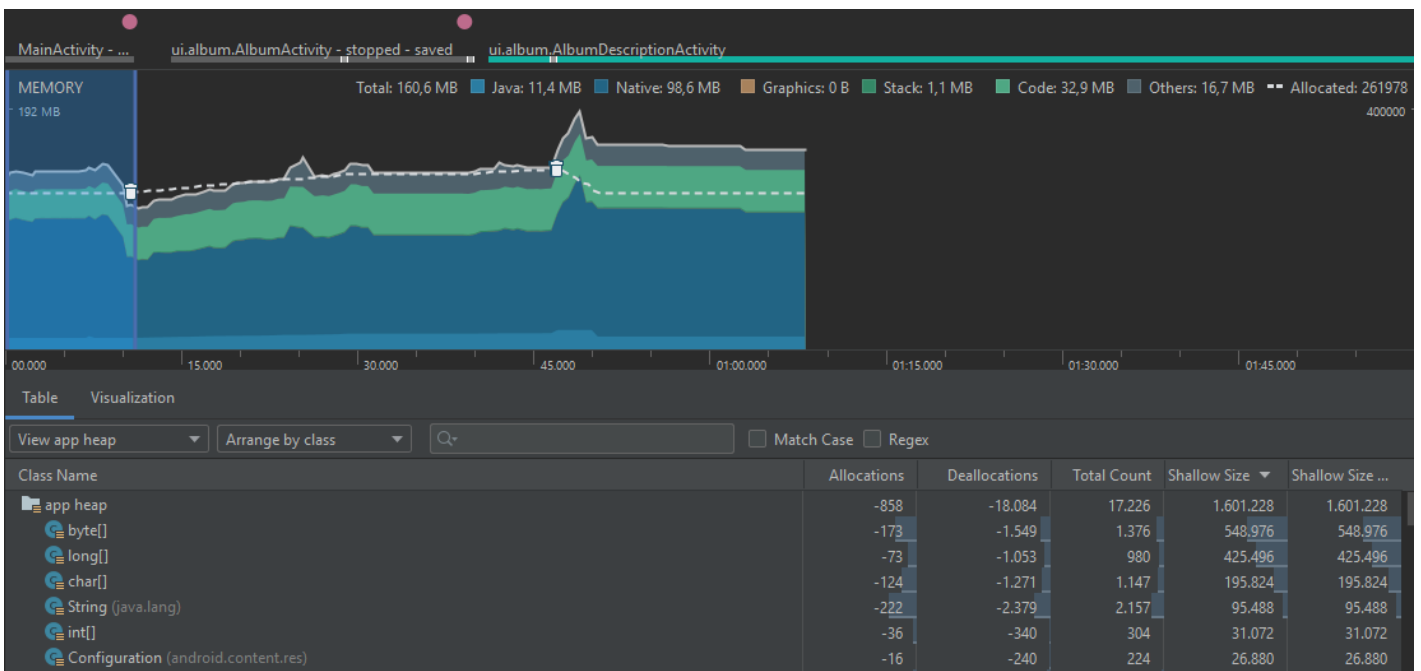
➤ Perfilar Rendimiento de la App – Post Micro Optimizaciones

La versión de la aplicación después de realizar las micro-optimizaciones está disponible en la rama develop. Desde esa rama se corrió nuevamente el perfil de rendimiento de la aplicación y se obtuvieron los siguientes resultados:

Consumo de Memoria durante todo el test.



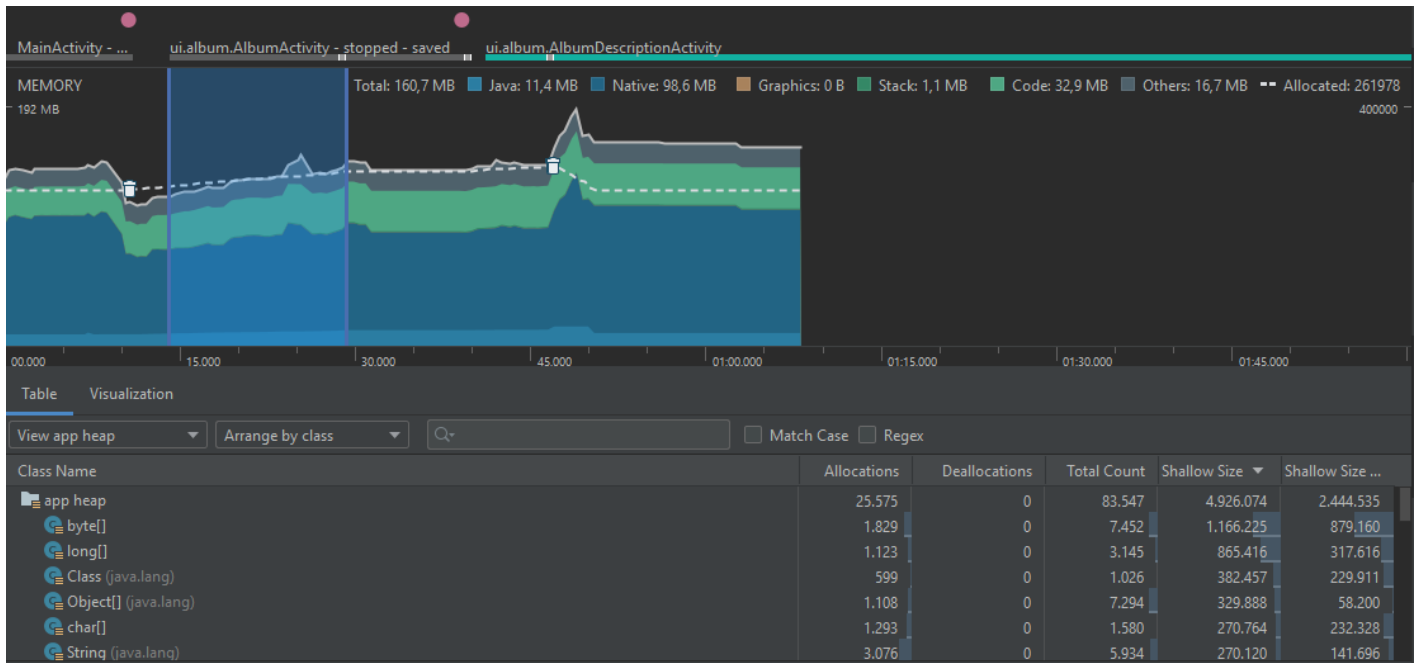
Consumo de Memoria Main Activity.



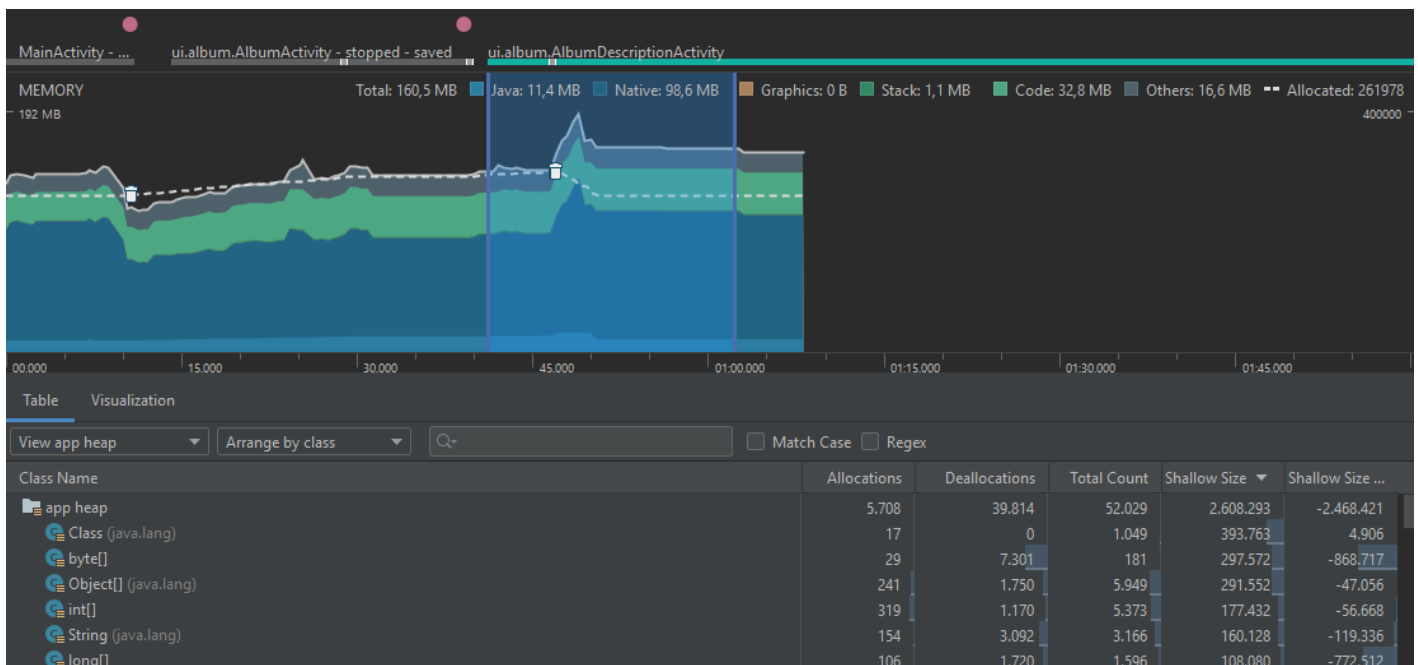
Plantilla elaborada por

THE SW DESIGN LAB

Consumo de Memoria Album List Activity.



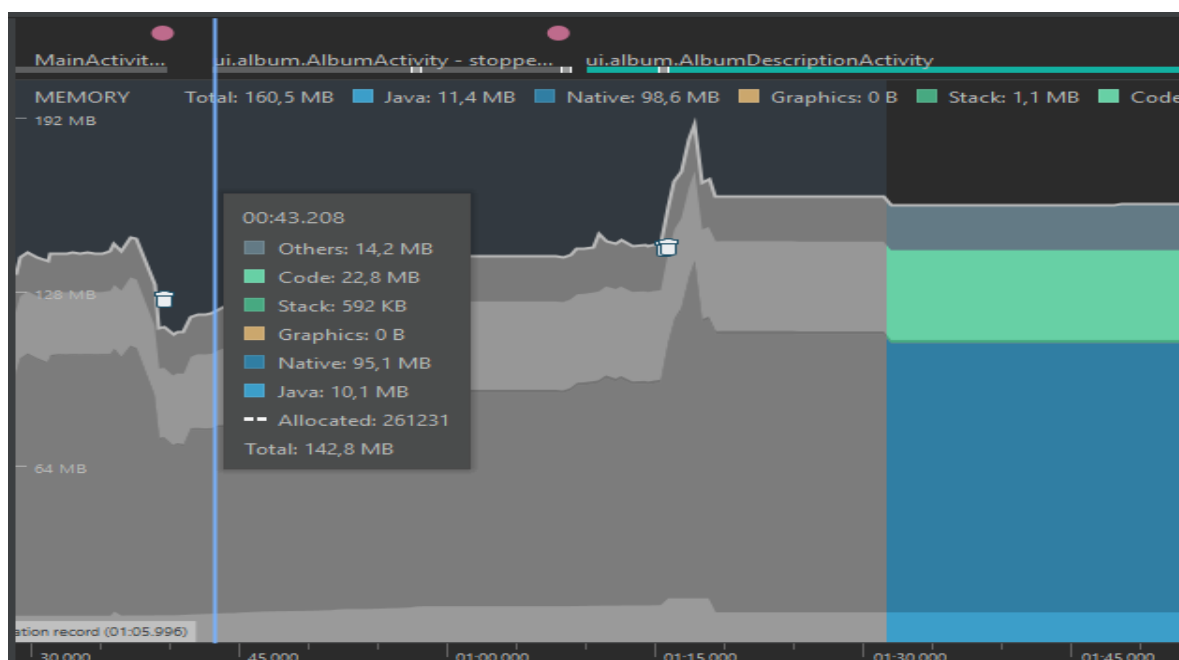
Consumo de Memoria Album Detail Activity.



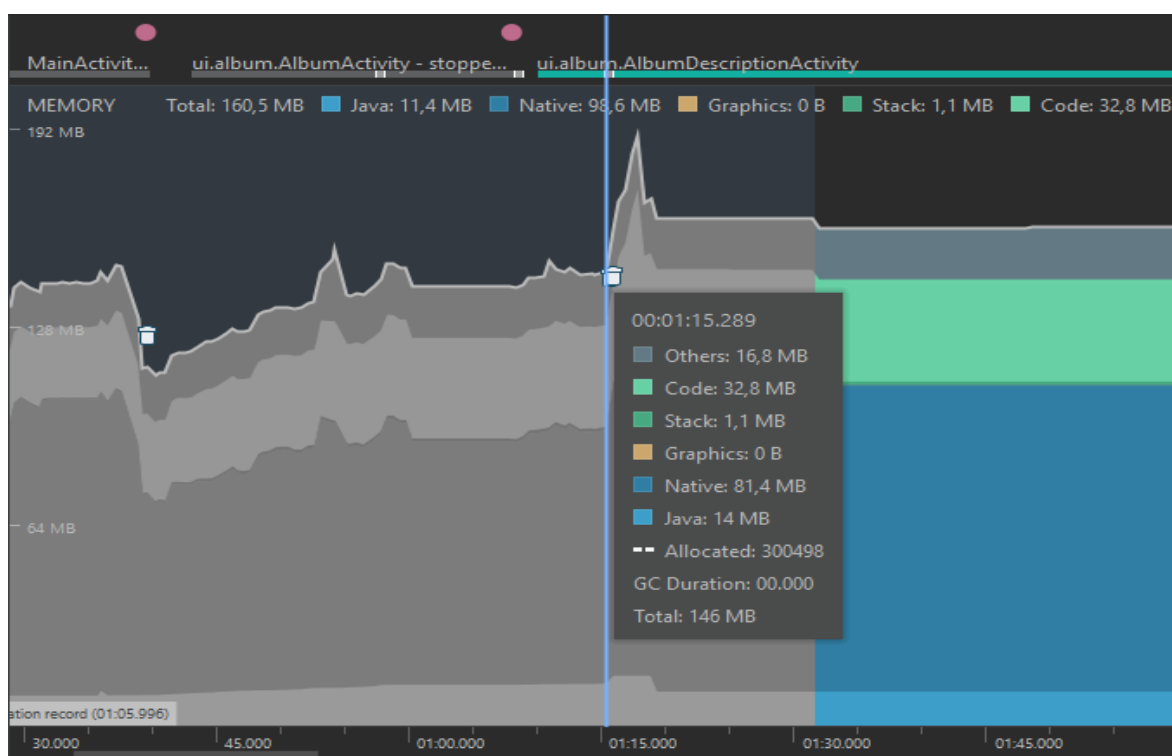
Plantilla elaborada por

THE SW DESIGN LAB

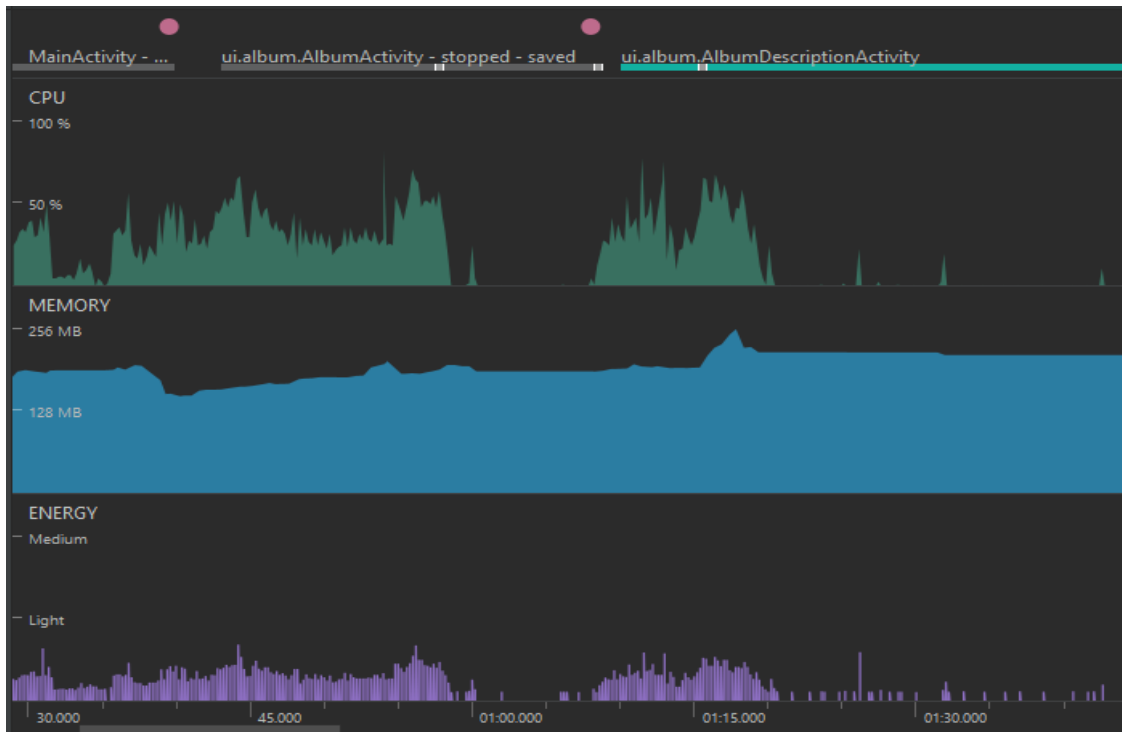
Porcentajes Consumo de Memoria – Inicio Album Activity.



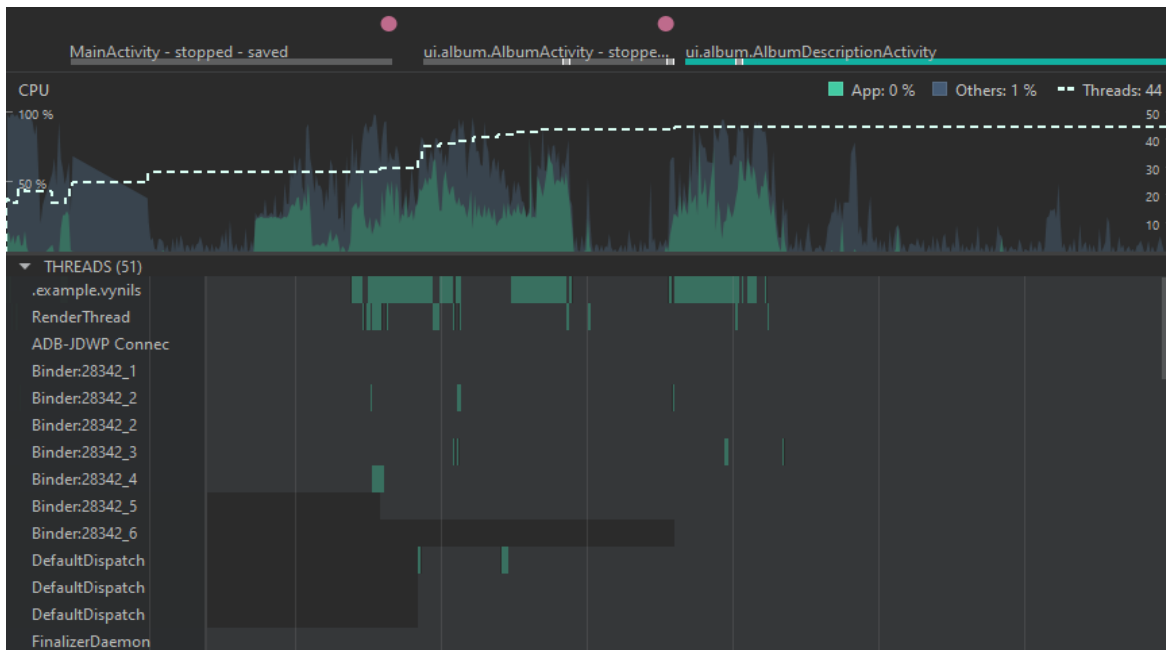
Porcentajes Consumo de Memoria – Inicio Album Detail Activity.



Data Completa Profiler.



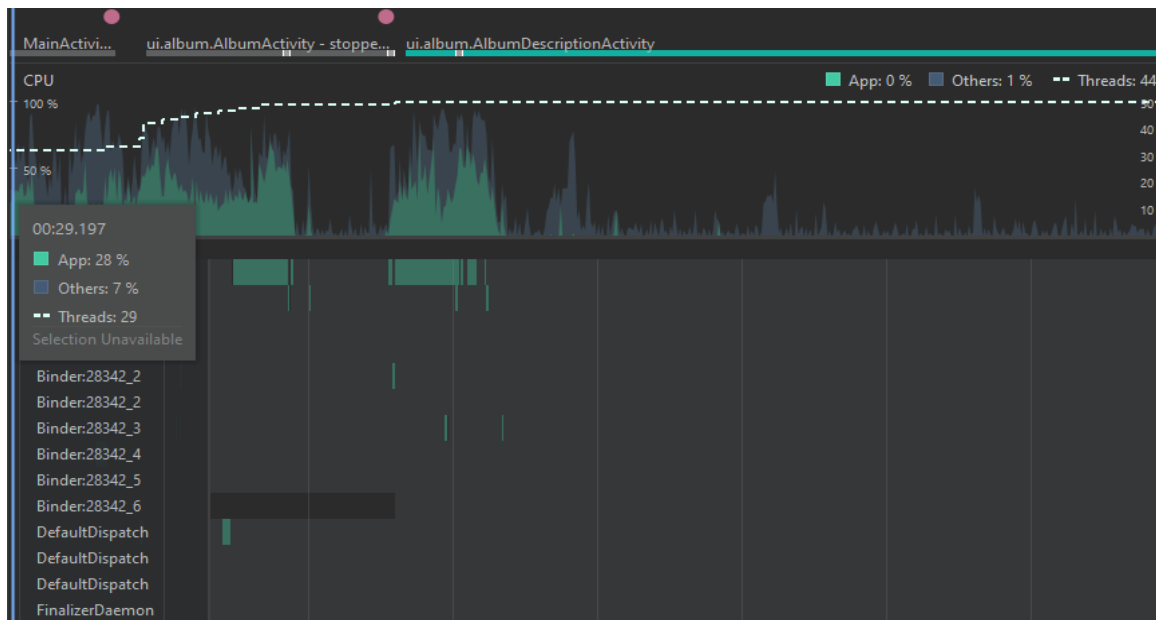
Consumo de CPU durante todo el test.



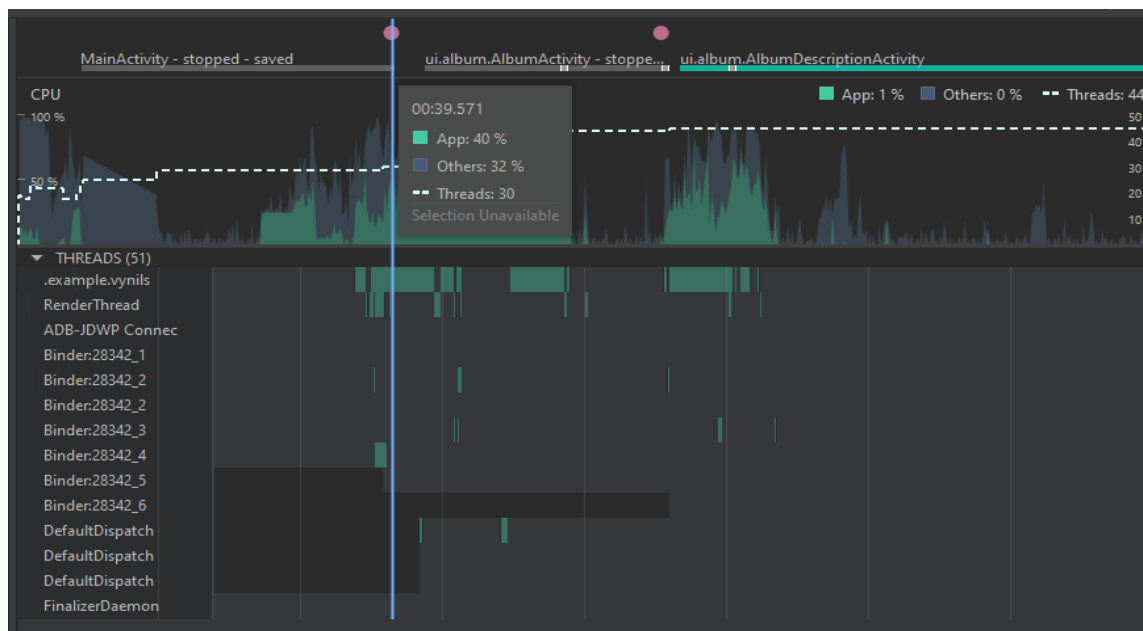
Plantilla elaborada por

THE SW DESIGN LAB

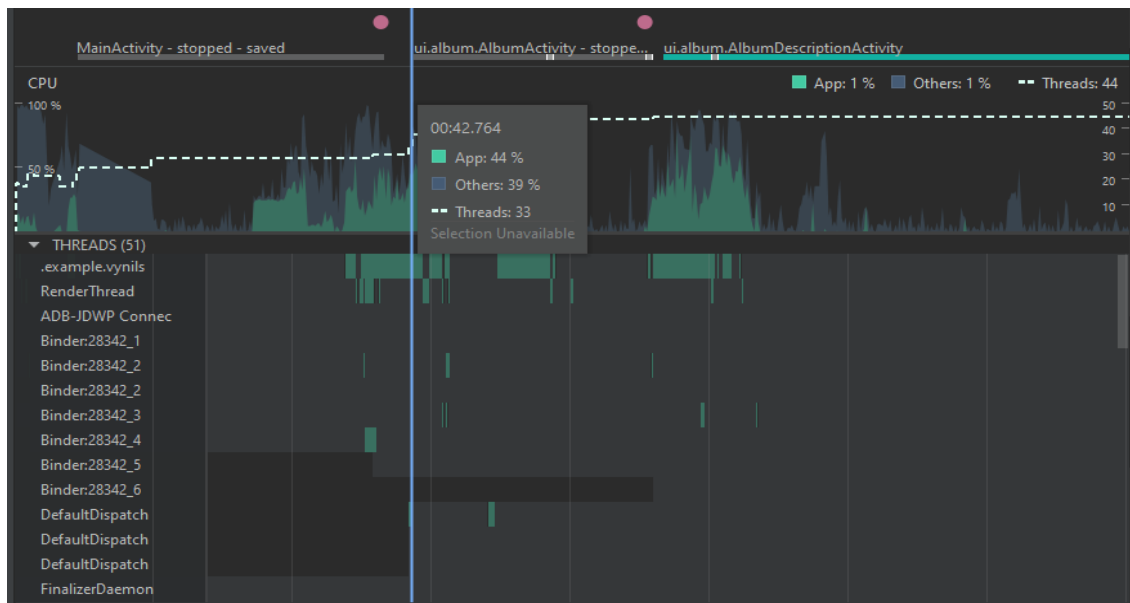
Consumo CPU –Inicio Main Activity.



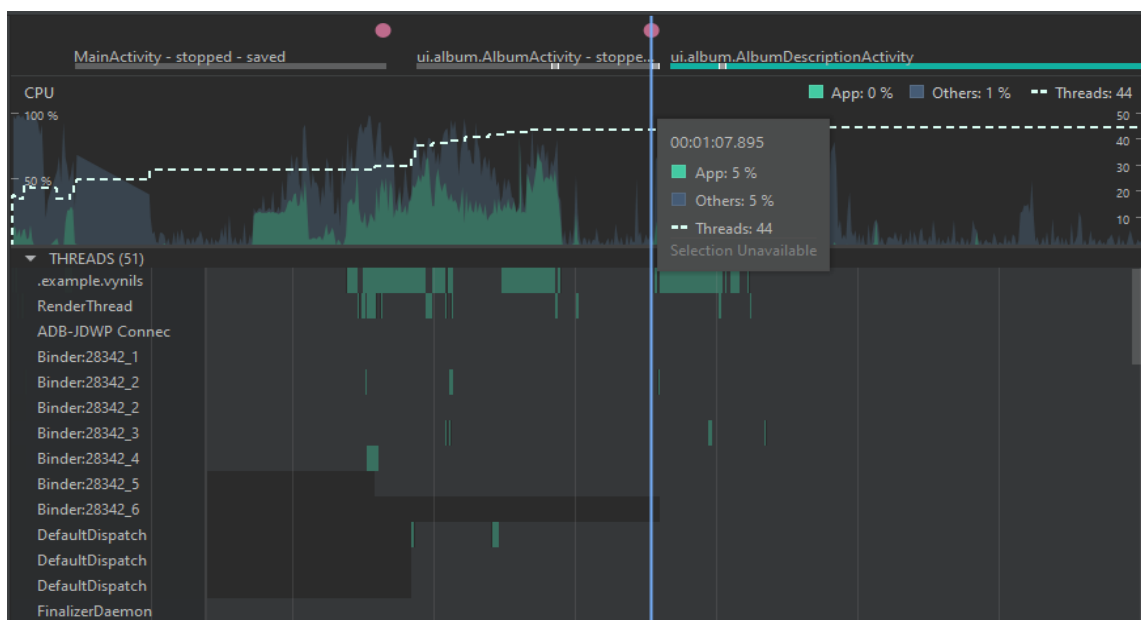
Consumo CPU –Finalización Main Activity.



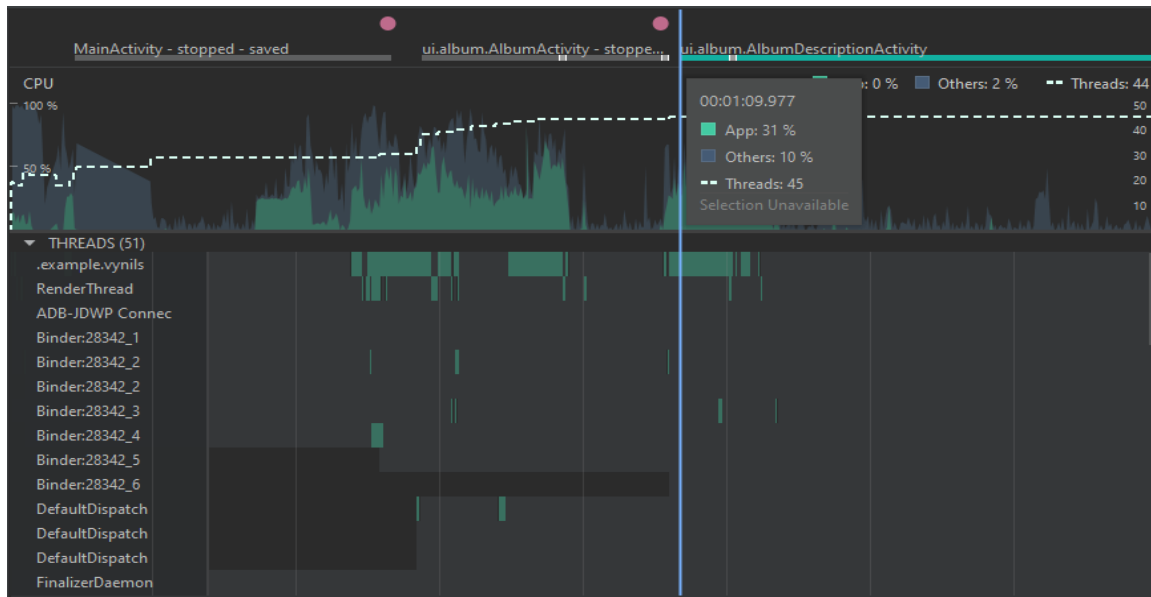
Consumo CPU –Inicio Album Activity.



Consumo CPU –Finalización Album Activity.



Consumo CPU –Inicio Album Detail Activity.



3.5.2. Análisis de Resultados Micro-Optimizaciones

Los resultados obtenidos en tabla de App Heap muestran que después de realizar las micro-optimizaciones a la aplicación el número de Allocations registradas se redujo en 4.179. Esta disminución en número de Allocations mejora el desempeño de la aplicación ya que se reduce el trabajo realizado por el sistema en el proceso de creación de nuevos objetos, así como el esfuerzo y tiempo requerido para la liberación de dichos objetos más tarde (garbage collectors).

El comparativo del App Heap Total Count previo y post micro-optimizaciones permitió conocer que este parámetro se redujo en 23.349. La disminución en el App Heap permite reducir la probabilidad de ocurrencia de OutOfMemoryErrors mejorando de esta forma el desempeño de la aplicación y la experiencia de usuario.

Finalmente, la revisión de los resultados del consumo de CPU muestran una reducción del 2% al inicializar la Actividad Listar Album y una reducción del consumo de CPU de 28% durante la inicialización de la Actividad Detalle Album. Esta disminución en el consumo de recursos favorece el desempeño de la aplicación.