Share your ride

Borja Villarroya Rodriguez
Máster universitario De Desarrollo de aplicaciones para dispositivos móviles
Eduard Martín Lineros

Fecha de entrega 30/12/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	Share your ride
Nombre del autor:	Borja Villarroya Rodriguez
Nombre del consultor:	Eduard Martín Lineros
Fecha de entrega (mm/aaaa):	01/2021
Titulación:	*Máster Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles*

Resumen del Trabajo (máximo 250 palabras):

Este proyecto ha querido unir dos mundos que ya comparten muchos ámbitos de uso pero que no están plenamente integrados: las cámaras de acción y los teléfonos móviles. Con este objetivo en mente, he usado las capacidades de procesamiento de los móviles de última generación, sus sensores de telemetría y conexión de red.

El resultado final es una aplicación que se conecta automáticamente a una cámara RTSP automáticamente, obtiene el vídeo emitido, almacena la telemetría en tiempo real y compone un vídeo final donde se asocia el movimiento del móvil al vídeo.

El desarrollo del proyecto no se ha limitado a conseguir la funcionalidad propuesta:

- Se han aplicado diferentes arquitecturas y patrones de diseño para aplicaciones Android como pueden ser servicios, LiveData, repositorios, patrón MVVM, navegación entre fragmentos, paralelización de tareas, programación orientada a eventos, rx, etc.
- Desarrollo de un sistema de mensajería entre servicios y View Models.
- El diseño de la arquitectura de software es altamente escalable y reutilizable.
- Ha sido un ejercicio de experimentación con las distintas herramientas de vídeo disponibles actualmente, donde se explican sus pros y sus contras.

El presente documento muestra todas las fases de diseño del producto, desde la descripción del concepto hasta la particularidades y hándicaps del desarrollo.

Abstract (in English, 250 words or less):

This project joins two world that already share a lot of use in common, but they aren't fully integrated: Action cameras and smartphones. To accomplish this target, I've used the processing capacities of the new generation smartphones, their telemetry sensors and their network connections

The result is an application that connects automatically to a RTSP camera, gets the video stream, stores the smartphone's telemetry in real time and finally, composes a video that joins the video with the movement of the mobile phone.

The development of this project goes beyond the proposed functionality:

- - It applies several architectures and design patterns oriented to Android applications such services, LiveData, repositories, MVVM patter, fragment navigation, use of parallel tasks, event-oriented programing, rx, etc.
- - A custom messenger system has been developed to communicate services and view models.
- - The architecture design is highly scalable and reusable.
- - It has been a test exercise of different video tools available, analyzing their pros and cons.

The present document shows all the design stages of the product, from the definition of the application concept to the description of the caveats and particularities of the development.

Palabras clave (entre 4 y 8):

Video, telemetría, action cam, aire libre, deporte, ocio.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

En la actualidad es común que las actividades deportivas o de ocio se graben con una cámara de acción, ya sea para uso propio, como para compartirlo entre grupos y redes sociales.

Estudios de mercado como los realizados por Rbv research [1] y Grand view research [2], se puede observar que el sector de las *action cam* está consolidado y sigue al alza.

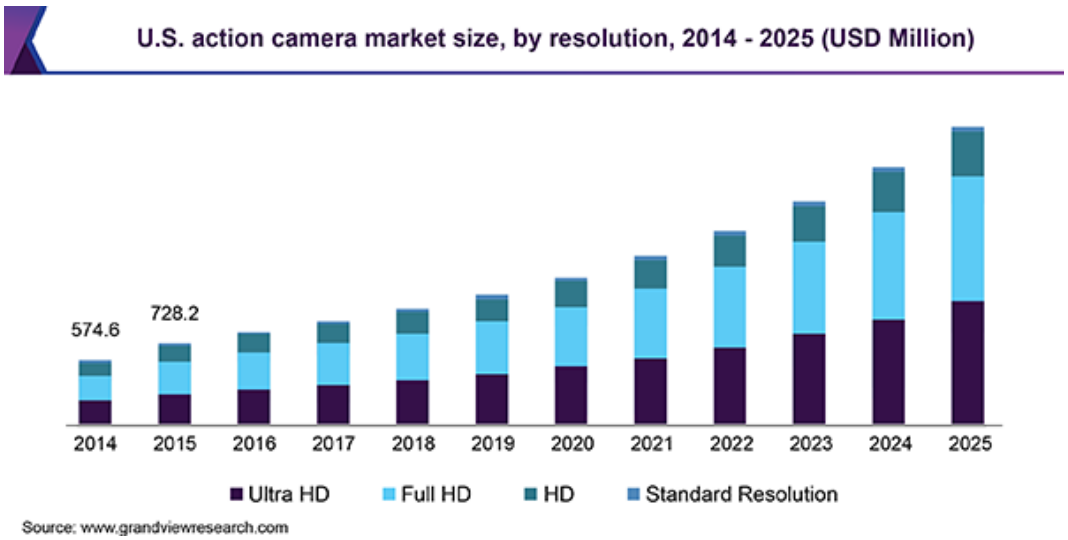


Figura 1 Tamaño del Mercado de action cams (Grand view research, 2019. [2])

El sector no está dominado por una marca o fabricante, sino que hay distintos actores bien posicionados, como GoPro, Garmin, Sony, Panasonic o Xiaomi.

Respecto al usuario final, este las emplea mayormente en el ámbito deportivo y recreacional, quedando en un segundo plano el resto de los ámbitos.

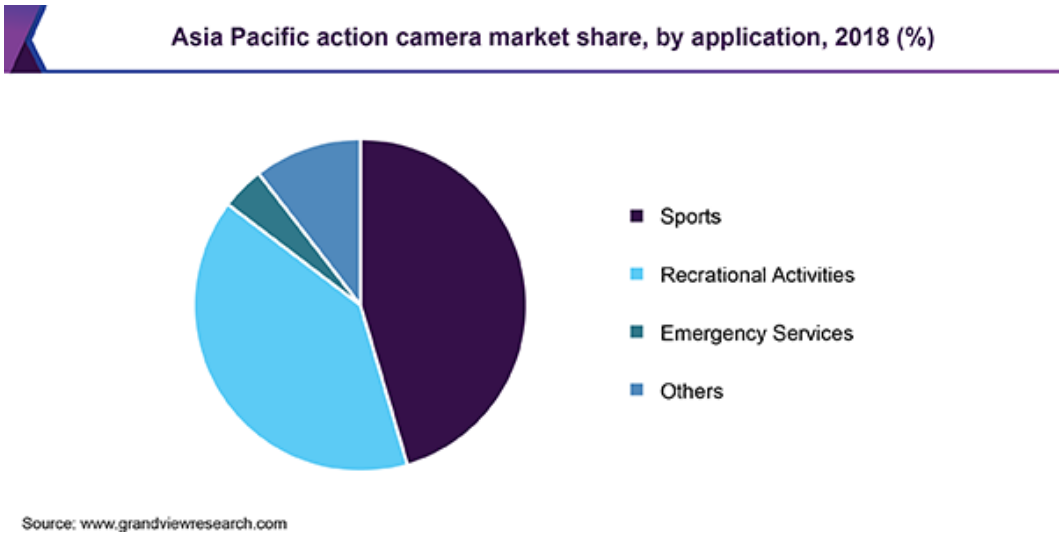


Figura 2 Uso de las action cams en el mercado asiático (Grandview research, 2019, [2])

En el mundo del motor, la nieve o la bicicleta el uso de estas cámaras está especialmente extendido, tanto como la creación de contenidos en las redes sociales con los vídeos resultantes.

Si uno participa activamente en alguno de estos mundos, se dará cuenta que no es solo grabar una actividad, sino mostrar cómo se está realizando, ya sea desnivel, velocidad, ángulo, tiempo o viento, etc. Es decir, mostrar el "qué" y el "cómo". Dos ejemplos prácticos:

- En las grabaciones en las estaciones de esquí, es común que en el vídeo no se aprecie el desnivel real de la bajada, o la velocidad a la que se realiza.
- En el ámbito del mundo de las motos, se suelen comparar mucho las trazadas de las curvas, la velocidad y la inclinación de la moto. En ese aspecto hay mucha competitividad.

Aprovechando un mercado en alza y fragmentado, mi experiencia personal y la de mis conocidos. Surge la idea de share your ride, la cual parte de una premisa: Crear una herramienta que dé un valor añadido a los videos que un usuario puede grabar y así enriquecer sus actividades de ocio.

1.1.1 Competencia

En este campo existen cierto número de cámaras de acción que ya incorporan sensores de localización y acelerómetros [1], pero no son las más extendidas y están limitadas al hardware de la cámara en sí. Cada fabricante tiene su forma de integrar la telemetría en los vídeos. Tenemos el ejemplo de GoPro [4], el cual proporciona una aplicación tanto para dispositivos móviles como para escritorio, la cual permite editar los videos añadiendo telemetría, pero esta funcionalidad, no se

soporta en todas sus cámaras y además el usuario tiene que esperar a que se añada el soporte para las nuevas versiones. Tampoco permite añadir valores externos a los capturados con la cámara.

En este caso, nos tenemos que fijar que el punto fuerte es que solo se requiere de un hardware, por lo tanto, la operativa es más sencilla para el usuario.

Para conseguir resultados similares, existen ciertos sitios web y software para equipos de escritorio que permiten montar un video con unos valores de telemetría dados. Pero esto implica que el usuario debe:

- Descargar el video original.
- Obtener los valores de telemetría con alguna herramienta contabilizadora.
- Tener que sincronizar manualmente el video y la telemetría.
- Usar una aplicación adicional para realizar la composición.

Algunos ejemplos son:

- Telemetry overlay for any camera [6]
- Race render [5]

En este caso el punto fuerte es la flexibilidad que da al usuario para elegir y crear sus propias plantillas de datos en los vídeos.
Por el contrario, es una labor compleja, que puede espantar a los usuarios, además en algunos casos, las herramientas de terceros son muy complejas, como puede ser After Effects [7].

1.1.2 Factor diferenciador

Los factores diferenciadores que hay que conseguir son:

- **Simplificar toda la operativa** de la competencia que usa aplicaciones de terceros. Tiene que ser orgánico y natural.
- Respecto a las soluciones *built in*, se debe proporcionar **mayor compatibilidad**. Si un usuario tiene varias cámaras o cambia de ellas frecuentemente no debe ser traumático. Además, el que el elemento contabilizador sea distinto a la cámara, permite que **la localización de la cámara no afecte a las medidas**, por lo tanto, el dato resultante es más real.
- **Modular y ampliable**. En esta primera versión nos vamos a ceñir a la funcionalidad básica, pero al usar el móvil como punto de entrada para la telemetría, tenemos un potencial para crecer aportando nuevas funcionalidades que una cámara de acción no puede aportar (signos vitales, tiempo, conexión con sensores de terceros, mapas cartográficos, etc.)

El usuario solo necesita una cámara WIFI que transmita vídeo en RTSP y un móvil Android. La aplicación Share your ride va a orquestar todo en segundo plano y al final de la actividad, generará el vídeo resultante,

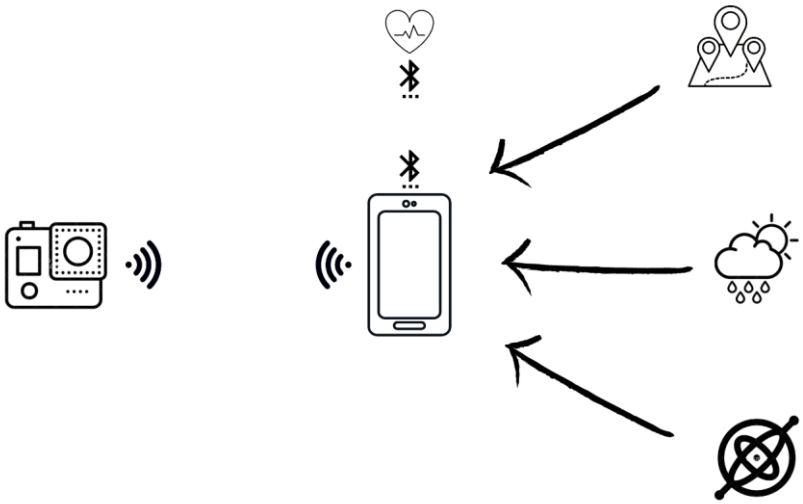


Figura 3 Esquema en alto nivel del ecosistema de la APP

El video resultante va a ser similar al que se puede ver en una competición de Motogp, Formula 1, o de un videojuego:

Figura 4 Ejemplo de video con telemetría

1.2 Objetivos del Trabajo

Para poder concretar los objetivos o requisitos primero necesito saber cómo y porqué se va a usar la aplicación. Este paso previo es fundamental para saber el alcance de cada tarea, definir qué partes tengo que descartar todo lo que me gustaría incluir en la App. y para ello poder construir una planificación realista.

Para ello voy a definir unas fichas de usuario a las que voy a asociar unas historias de usuario de las cuales, de las cuales obtener el listado de objetivos y requisitos.

1.2.1 Historias de usuario

Ficha 1 (Perfil esquiador)

Nombre: Juan
Edad 36
Nivel de estudios: Ingeniería Informática
Trabajo: Desarrollador de software

Ficha de persona
Juan es una persona extrovertida, de carácter afable, trabaja mucho pero también le gusta mucho salir de fiesta. Como buen informático, le encantan las nuevas tecnologías y siempre está probando Apps nuevas.
Juan es muy activo en las redes sociales, sobre todo en Instagram y Facebook, comparte opiniones, sus actividades deportivas y de ocio. Es un esquiador empedernido, es su afición favorita. Le gusta esquiar tanto por pistas como por fuera de ellas. Siempre graba sus sesiones de esquí y las de los que le acompañan.

Ficha de escenario

Es un domingo por la mañana, es el primer día de esquí del viaje anual a los Alpes. Hay muchas ganas de esquí, nervios y emoción, ya que esta estación es nueva, este año toca Italia, la parte italiana, Cervinia, con pistas inmensas, anchas y rápidas. Hace mucho frio, unos 15 grados bajo cero, por lo tanto, el tiempo que puede permanecer quieto tras bajarse la silla es escaso. Juan enchufa su *action cam* y la APP. A los pocos segundos los dos dispositivos se conectan y están listos ya que tenía preconfigurada ya su cámara y que iba a esquiar. Juan inicia la bajada, baja fuerte, se “gusta”.

Por la noche, Juan muestra orgulloso en Facebook y en Instagram los videos donde se le ve bajando más rápido de lo que lo ha hecho nunca, lo largas que son las pistas, su altura y el tremendo desnivel de la bajada.

Ficha 2 (Perfil motorista)
Nombre: Lidia
Edad 30
Nivel de estudios: Grado medio
Trabajo: Comercial

Ficha de persona
Lidia trabaja en una tienda de motos como comercial. Es extrovertida, con un poco de “cara dura” y con mucha labia. Le gustan los móviles y

las nuevas tecnologías a nivel de usuario. Usa las redes sociales a diario y a parte tiene un canal de YouTube.

Comparte la afición por las motos con su novio. Sale de ruta asiduamente, se hace un viaje al extranjero en moto todos los años. Además, frecuenta los circuitos en jornadas de tandas. Siempre graba sus sesiones con varias cámaras, dependiendo la moto y la localización de la cámara.

Ficha de escenario

Lidia se ha cambiado recientemente de moto, es más potente agresiva y difícil de llevar que la anterior. Se ha propuesto mejorar los tiempos de los circuitos a los que va.

Utilizando la aplicación Share your ride, junto con un profesor, primero usa una cámara de gran angular en la parte trasera, donde se le ve la postura junto con los indicadores de inclinación. Esto le ayuda a corregir defectos en su postura. Posteriormente repite la operación con una segunda cámara que lleva en el mentón del casco. Como el móvil va anclado a la moto, los movimientos de Lidia no afectan a valores. Con este video, comparando los valores con los de su profesor, pude comprobar que aún tiene margen para frenar más fuerte y acelerar más rápido en la salida de las curvas. Ven hasta donde se puede apurar frenar, donde puede inclinar más.

Posteriormente, Lidia usa parte del vídeo con sus telemetrías en su video blog en YouTube, donde comparte con sus seguidores, sus progresos.

Ficha 3 (perfil ciclista de montaña)

Nombre: Mike
Edad 26
Nivel de estudios: Grado medio
Trabajo: Administrativo

Ficha de persona
Mike es nativo digital, vive en el estado de California y se ha criado con móviles y ordenadores a su alrededor. Lleva desde pequeño manejando las redes sociales y actualmente compatibiliza su trabajo de administrativo con su afición al mundo de la bicicleta de montaña.

Ficha de escenario

Mike forma parte de un grupo de ciclistas que suben todos los fines de semana a pistas de descenso de bici de montaña. A parte de la amistad y de disfrutar de la bici, amenizan las bajadas con un pique entre ellos a ver cuántas millas por hora alcanzan en algunos tramos. A Mike solo le

interesa ver a la velocidad, que va, el resto de los valores de telemetría, le molestan el vídeo, así que configura la APP a su gusto. El resto del grupo se han instalado también la APP. Ahora sus piques han alcanzado un nuevo nivel.

1.2.2 Fichas de usuario

Partiendo de los escenarios y los perfiles descritos en el apartado anterior se obtienen una serie de caos de uso. Antes de sacar el listado de funcionalidades final, creo primero una recopilación de fichas de usuario AGILE.

Quiero	Para
Guardar la configuración de la cámara y las actividades.	Agilizar el inicio de las actividades.
Seleccionar que tipo de actividad voy a realizar y que la aplicación se configure en consonancia.	No perder el tiempo con configuraciones
Tener valores preestablecidos de configuración de mi cámara	No perder el tiempo con configuraciones
La conexión con la cámara Wifi se haga de forma automática	Agilizar el inicio de las actividades.
Ver a la velocidad	Mostrar la velocidad a la que voy
Ver a la altura que estoy	Mostrar un dato categorizador del esfuerzo.
Ver el desnivel actual	Ver la dificultad de la subida o bajada
Quiero ver la fuerza que sufro	Ver la aceleración de la moto y la fuerza de las frenadas.
Que el móvil genere un vídeo con la telemetría	No quiero perder el tiempo con otras aplicaciones.
Quiero ver la inclinación lateral	Ver cuanto inclino con la moto.
Poder configurar el acceso a mi cámara, aunque no se soporte de forma oficial.	Poder usar mi cámara que no ha sido tenida en cuenta por los desarrolladores.
Poder seleccionar la magnitud entre sistema métrico o imperial	Ver el video con unos valores que entiendo.
Seleccionar que valores quiero ver en el video	Para customiza el contenido del vídeo a mi gusto
Que la APP funcione en segundo plano	Poder bloquear el móvil durante la actividad.

Tabla 1 Fichas de usuario

1.2.3 Listado de objetivos y sus requisitos

Para poder organizar las tareas de mejor forma, enumero un primer nivel, que son los objetivos, y estos los desgrano en requisitos. Los requisitos nos ayudarán posteriormente a adjudicar una estimación de horas y generar las pruebas funcionales.

Listado de objetivos imperativos, el núcleo funcional de la APP para móviles Android.

Objetivo de primer nivel	Requisitos
Conexiones automáticas a redes WIFI	• Configuración de parámetro para conectarse a redes WIFI. • Conexión automática a una red configurada. • Detección de estado de la conexión WIFI
Localización del dispositivo móvil	• Detección del estado del GPS. • Cálculo de coordenadas • Cálculo de altitud • Cálculo de velocidad • Cálculo del desnivel • Representación gráfica de los valores
Manejo de los acelerómetros	• Captura del ángulo de inclinación del móvil en el eje Y (para modo *portrait*). • Obtención de fuerza instantánea sobre el móvil. • Calibración de los sensores. • Representación gráfica de los valores
Generar un vídeo con telemetría	• Obtener el vídeo emitido por una *action cam* en tiempo real. • Generación Almacenamiento del vídeo con la telemetría de la actividad. • La telemetría y el vídeo debe estar sincronizada.
Interfaz clara y sencilla	• Minimizar el número de clics por acción. • Automatizar el máximo número de acciones posibles.
Gestión de la energía en segundo plano	• Al bloquear el móvil la captura de telemetría y vídeo debe continuar. • Al pasar otra APP a primer plano, la captura de telemetría y vídeo debe continuar.

Tabla 2 Objetivos de primer nivel

Esto son una serie de objetivos opcionales que hacen peligrar la integridad de la aplicación si no son desarrollados.

Objetivos de segundo nivel	Requisitos
Configuración de la aplicación.	• Listado de tipos de actividad con parámetros para el vídeo. • Poder editar individualmente qué parámetros se muestran en el vídeo. • Poder configurar el sistema métrico o imperial. • Poder elegir en qué se muestra cada parámetro individualmente. • Hacer persistente la configuración.
Ventana de login de usuario con autenticación en la nube.	• Login usando una cuenta de Google. • Login usando correo. • No requerir siempre el login en un mismo dispositivo (Smart lock)

Tabla 3 Objetivos de segundo nivel

Adicionalmente añado un listado de objetivos del desarrollo, no relacionados con la funcionalidad en sí.

Objetivos personales en el desarrollo
Mejorar el manejo de los sensores de un dispositivo móvil Android
Dominar el patrón MVVM con Jet Pack
Aprender técnicas para el tratamiento de vídeo.
Mejorar mis habilidades a la hora de gestionar librerías y cómo modularizar el código

Tabla 4 Objetivos del desarrollo

1.3 Enfoque y método seguido

Se trata de una aplicación nueva desarrollada desde cero. El desarrollo tiene la particularidad de que en el momento de redacción del documento una parte ya ha sido realizada en mi tiempo libre.

Por este motivo y debido a la complejidad del proyecto el uso de una metodología ágil es el mejor enfoque. El desarrollo de las funcionalidades se plantea de forma individual, cada uno con sus requisitos y haciéndolos independientes, para que, si uno de ellos no puede abordarse, no afecte a la integridad del resto. Por ejemplo:

- Configuración
- Manejo de GPS
- Manejo de acelerómetros
- Gestión de vídeo

La falta de uno de estos módulos puede afectar al resultado final del proyecto, pero no a la integridad del resto de módulos. Esta segmentación en un desarrollo en cascada o en uno de prototipado es complicado, ya que se desarrolla para conseguir un todo desde el principio.

Voy a realizar sprints de SCRUM. En cada sprint se van a planificar unas pocas tareas que van a ser acometidas una a una. Siempre teniendo en cuenta que cumplan al menos uno de los requisitos obtenidos en 1.2.3 Listado de objetivos y sus requisitos. En función de la desviación que se produzca en cada sprint, se recalculará la planificación.

Una vez esté hecha la planificación. La organización de las tareas se realiza mediante un Kanban, para poder identificar rápidamente el estado del desarrollo. Es un recurso psicológico y de motivación.

Durante el desarrollo, se definen los siguientes pasos, para cada funcionalidad ya sea interna o de interacción con el usuario:

- Se revisan los requisitos, los casos de uso y el diseño planteados en la PEC1 y PEC2.
- Codificación.
- Test unitarios
- Se plantean una serie de pruebas funcionales, en el caso que sean posibles.
- Se documenta la parte relevante de la funcionalidad y se justifican las decisiones.

Las otras alternativas en metodologías de desarrollo presentan una serie de desventajas:

- Un desarrollo en cascada no va a dejar margen de maniobra en el caso de que un módulo finalmente no pueda ser implementado o su resultado no sea satisfactorio. Si llega la fase de pruebas en el último estadio del desarrollo y la APP es deficiente, no voy a poder establecer medidas de contingencia.

- La aplicación es demasiado compleja para optar por un desarrollo por prototipado. El núcleo de la aplicación está en las capas de abajo, en el uso de sensores, tratamiento de video etc. Sin una buena base el proyecto no es sostenible, por lo tanto, en las primeras fases del desarrollo no va a haber avances sustanciales en la parte de usuario.

1.4 Planificación del Trabajo

1.4.1 Estado del arte

Para poder realizar una correcta planificación hay que tener en cuenta el trabajo que está ya realizado, lo que afecta a la estimación de las tareas.

- Librería para acceder a base datos con ROOM, con tablas definidas para toda la telemetría.
- Librería de uso de GPS (falta integración con la APP)
- Librería para la captura de los valores de los acelerómetros (falta integración con la APP)
- Librería para la gestión del WIFI (completa)
- Configuración de la APP (Completa).
- Actividad de login (Completa)
- Menú de navegación (Completo)
- Actividad “Home” del usuario (Completa)

Todo este trabajo se ha realizado al hacer pruebas de concepto con actividades de prueba. Todas las librerías las desarrollé pensando en que sean independientes, fáciles de usar e integrar en la arquitectura de la APP.

1.4.2 Horario y recursos

Un horario realista que me permite compaginar el proyecto con mi trabajo a jornada completa:

- 2 horas los días laborables
- 8 horas los días festivos

Esto da un total de 26 horas semanales, por 13 semanas disponibles. Las semanas del 7 al 11 de diciembre, del 21 al 25 y del 28 al 30 se dedica la jornada completa al proyecto, eso son 78 horas adicionales no da un total de 416 horas de trabajo.

Adicionalmente, en Aragón hay 5 días festivos y yo dispongo de dos semanas de vacaciones. Esto podría aportar hasta 36 horas más de presupuesto, pero se guardan de reserva de contingencia. La planificación se va a realizar sin contar estas horas

Estas horas no las incluyo en la planificación, ya que pueden verse comprometidas por otros compromisos y no son seguras.

Recursos hardware para el desarrollo:

- Portátil MSI GL 63 8RD (Intel i7 octava generación, 16 GB de RAM. 1TB de disco duro.

- Monitor philips 276E

Recursos hardware para las pruebas funcionales:

- Móvil Android Pocophone F1
- Móvil Android Honor 7
- Móvil Android Xiaomi
- Action cam Xiaomi Mi 4K
- Action cam Go Pro 3 Hero

Recursos software

- Suite Office 365 Pro plus
- Android studio
- Inkscape
- Gimp
- Cuenta premium en www.canva.com/
- Project Studio
- StarUml 2

1.4.3 Sprints

Definimos 4 sprint:

- Sprint 1: (8 al 18 de octubre) Diseño: Debido al estado del arte actual, no es necesario usar todo el tiempo de la PEC2 para el periodo de diseño.
- Sprint 2: (18 de octubre al 6 de noviembre) Manejo de sensores y telemetría.
- Sprint 3: (7 de noviembre al 4 de diciembre): GPS, navegación, integración de vídeo y entrega de PEC3
- Sprint 4: (5 de diciembre al 30 de diciembre) Finalización del desarrollo de vídeo, pruebas, correcciones y preparación de la entrega final.

1.4.4 Estimaciones y planificación

A la hora de planificar he intentado primar el obtener un producto con una funcionalidad que le dé sentido a la APP, sin comprometer la calidad del resultado final. Una planificación más ambiciosa, habría dado como resultado un producto globalmente más pobre.

A continuación, se muestran un listado de tareas (los objetivos) en alto nivel con las estimaciones de horas, los sprints a los que pertenecen y en que PEC se van a incluir. Las tareas que se han indicado como completadas en 1.4.1 Estado del arte, no se incluyen, ya que solo quedan tareas de documentación y pruebas, por lo tanto, se incluyen en dichas tareas.

Id	Objetivo	Estimación	Sprint	PEC
T1	Revisión de usuarios y fichas	4h	1	2
T2	Definición de casos de uso	8h	1	2
T3	Paso a limpio de la interfaz de usuario	18h	1	2
T4	Paso a limpio de la arquitectura	8h	1	2
T5	Localización del dispositivo móvil	34h	2	3
T6	Manejo de los acelerómetros	36h	2	3
T8	Generar un vídeo con telemetría	160h	3 y 4	3 y 4
T9	Revisión documentación PEC 3	26h	3	3
T10	Gestión de la energía en segundo plano	0h[1]	3	3
T11	Pruebas de usuario y corrección de errores	20h	4	4
T12	Manuales	24h	4	4
T14	Video de presentación	24h	4	4
T15	Revisión de la documentación	16h	4	4
T17	**Total**	396h		

Tabla 5 Estimación de horas del proyecto

A continuación, se muestran dos diagramas de Gantt con el orden de las tareas

[1] Aunque figura como objetivo como entidad propia, este factor tiene que ser tenido en cuenta en las tareas T5, T6 y T8

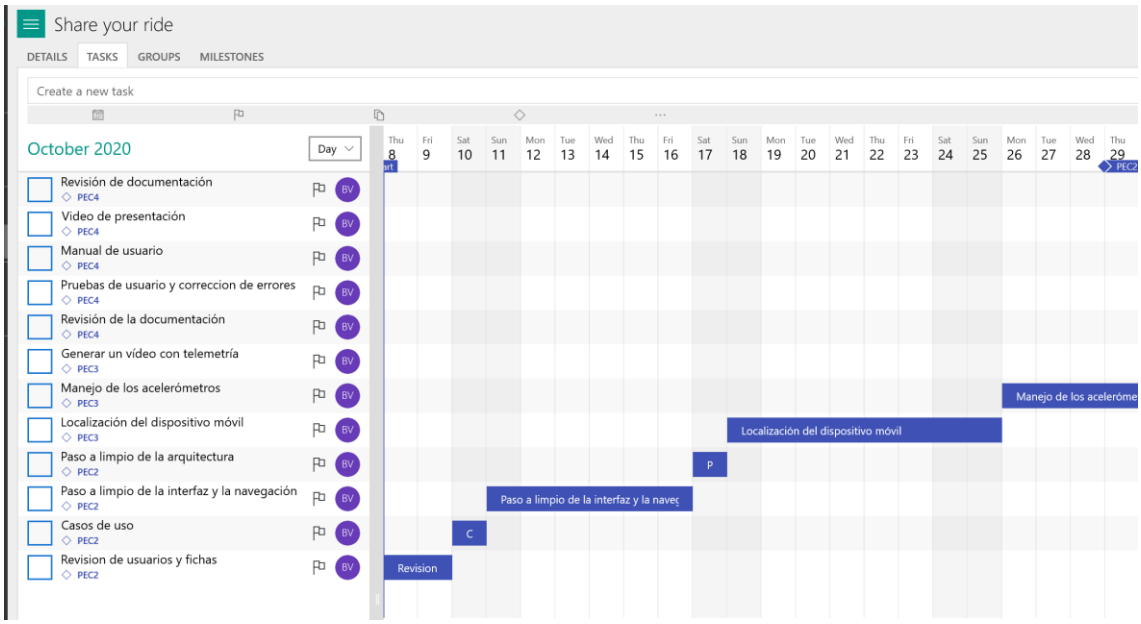


Figura 5 Gantt de la PEC2

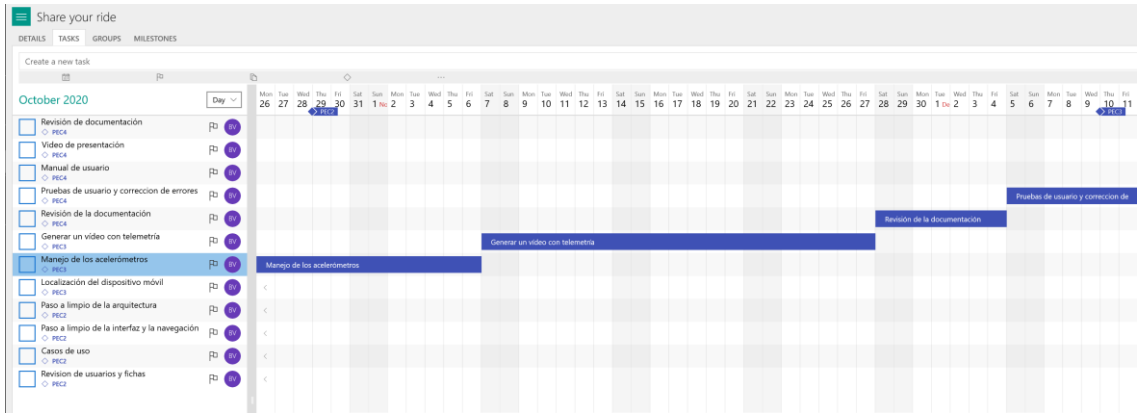


Figura 6 Gantt de la PEC3

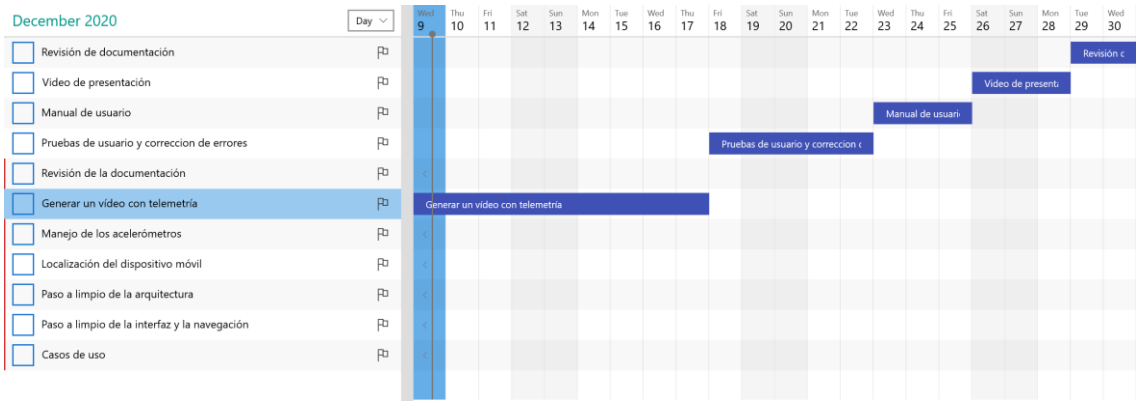


Figura 7 Gantt de la PEC4

1.4.5 Restricciones y evolución futura de la APP

Restricciones por decisión de diseño:

- La aplicación va a requerir de acceso a los sensores, librerías de video compiladas explícitamente para esa plataforma y va a ser muy demandante a nivel de recursos. Hacer la aplicación multiplataforma requiere de un conocimiento de *frameworks* del que no dispongo, por lo tanto, opto por un enfoque más conservador y la aplicación se va a realizar únicamente para Android.
- No es una red social. Ya existen redes sociales para compartir videos y experiencias personales como Facebook, Instagram o Tik Tok.

Restricciones por recursos:

- Solo se provee un número reducido de configuraciones de cámaras preestablecidas, solo a las que se han podido verificar que esa configuración funciona (el listado de cámaras se proporcionará al final de desarrollo).
- Por motivos logísticos y monetarios no se van a almacenar los videos en la nube. Esto requeriría de capitalizar el proyecto, hacer estudios de mercado e instaurar un sistema de cuotas o de usuario *premium*.

Existe una serie de funcionalidades que, por motivos de planificación, no pueden implementarse en la primera versión:

- Galería de videos donde el usuario pueda buscar todas sus actividades, ordenarlas, visualizarlas y compartirlas con el resto de los usuarios.
- Mostrar la temperatura, la dirección y fuerza del aire en el vídeo.
- Mostrar una brújula con la orientación en el vídeo.
- Incluir las pulsaciones y el volumen de oxígeno en sangre del usuario mediante el uso de *wearables*.
- Ofrecer al usuario varias plantillas con la distribución de la telemetría, incluso dotarle de una herramienta para que lo edite.
- Retransmisión en tiempo real del vídeo con la telemetría a otros dispositivos móviles.

1.5 Breve sumario de productos obtenidos

- Memoria del trabajo
- APK para móviles Android
- Manual de la APP
- Manual de instalación y compilación del código
- Código fuente
- Videos de presentación de la APP

2. Casos de uso

Para poder diseñar la aplicación el primer paso es detectar qué casos de uso debe cubrir y hacer un primer planteamiento de cómo el usuario y los subsistemas de la APP van a interactuar (en alto nivel). Para esta tarea parto de los objetivos relatados en 1.2.3 Listado de objetivos y sus requisitos.

2.1 Inicio de la APP

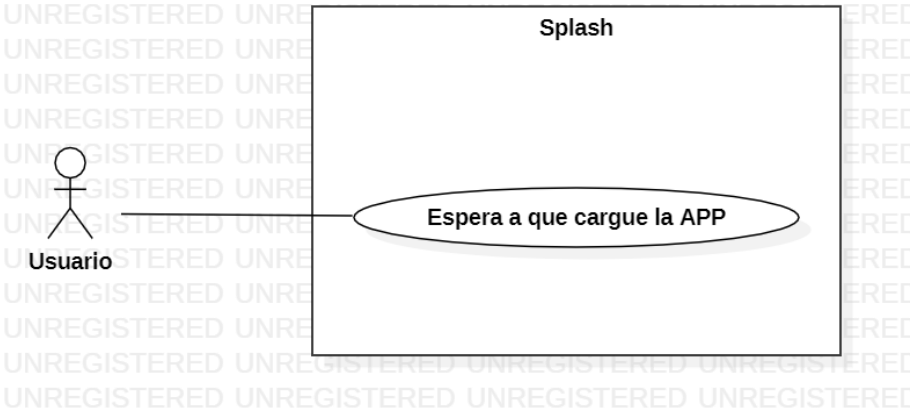


Figura 8 Caso de uso, inicio de la APP

En este caso, el usuario enciende la APP, durante unos segundos, mientras la APP carga la información muestra un Splash de bienvenida.

2.2 Configuración de la APP

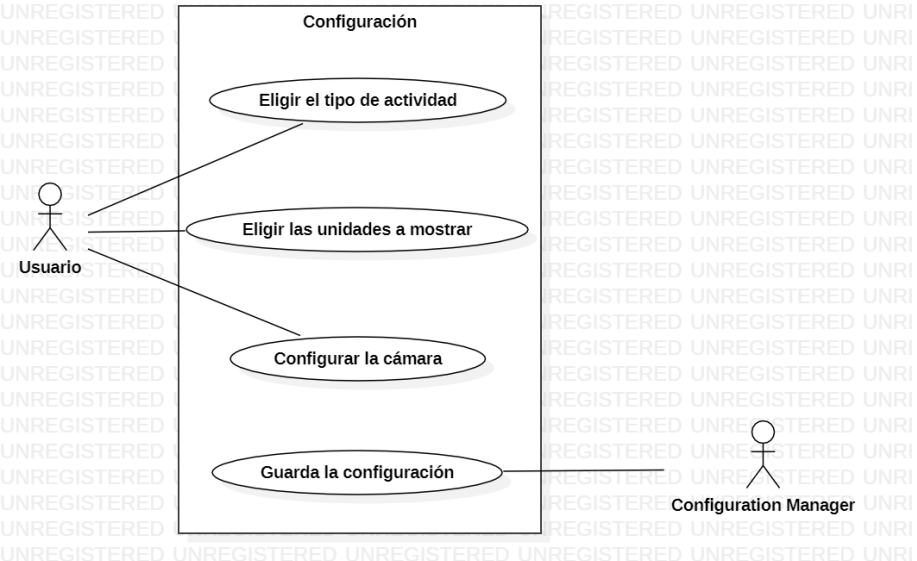


Figura 9 Caso de uso, configuración de la APP

El usuario quiere configurar la actividad que va a realizar, las unidades en las que van a mostrar las métricas y configurar la cámara que se va a usar para capturar el vídeo.

Por la parte del sistema, un componente de la APP se encarga de guardar y persistir dicha configuración.

2.3 Inicio de la actividad

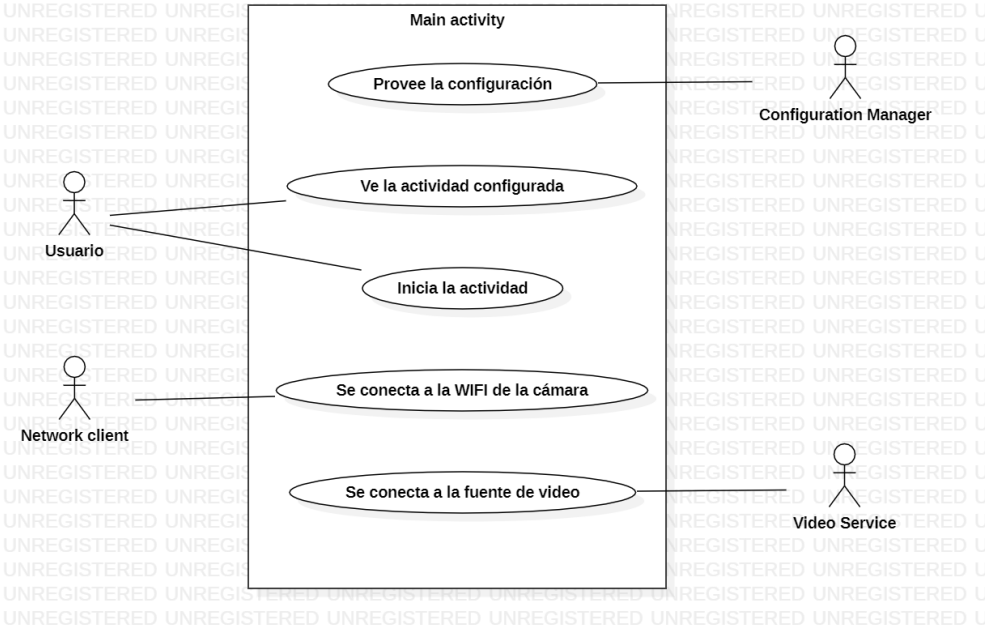


Figura 10 Caso de uso, inicio de la actividad

El componente de la aplicación que maneja la configuración, la presenta al usuario y al resto de componentes del sistema.

El usuario puede ver la configuración que va a aplicar en la actividad y pulsando un botón iniciar la actividad.

El componente de la APP que gestiona las conexiones de red escanea en busca de la red WIFI que crea la cámara configurada y se conecta a dicha red.

El componente de vídeo se conecta al vídeo emitido por la cámara, siempre que se haya establecido una conexión WIFI antes. Una vez conectada al flujo, muestra una indicación de que se ha establecido la conexión.

2.4 Calibración de los sensores

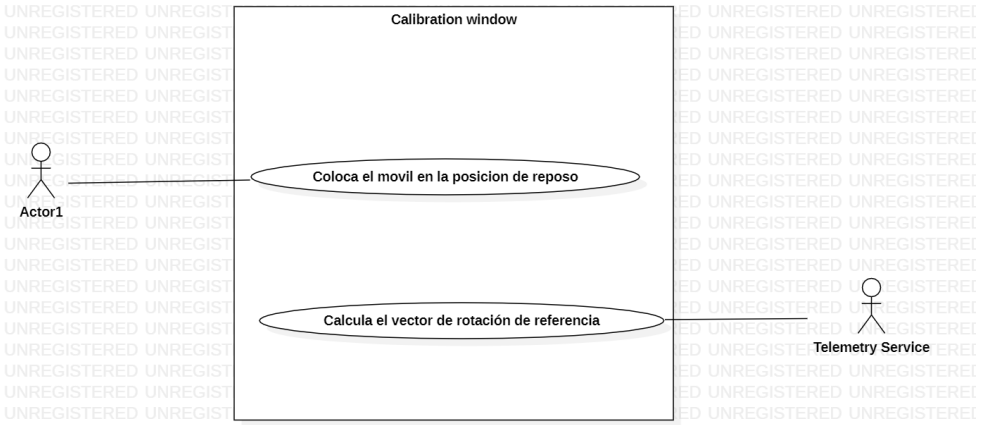


Figura 11 Caso de uso, calibración de los sensores

Para poder calcular la inclinación del móvil durante la actividad, se debe establecer un vector de rotación de referencia. Todos los valores de inclinación calculados serán relativos a dicho vector.

En este caso, el usuario, simplemente debe dejar el móvil en reposo en el sitio donde va a realizar la actividad.

2.5 Sincronización del vídeo

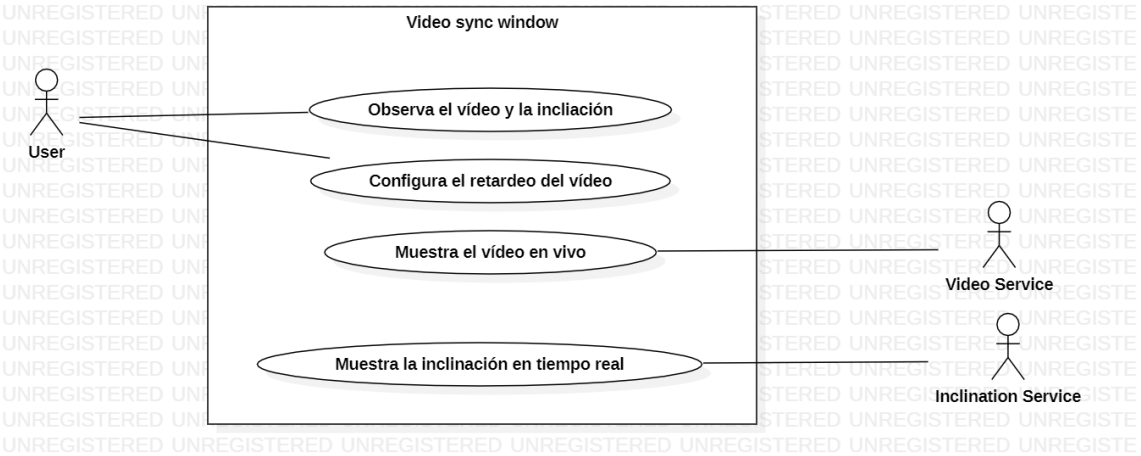


Figura 12 Caso de uso, sincronización del vídeo

El usuario ve en una misma pantalla el vídeo en vivo y la inclinación en tiempo real del dispositivo. Mediante un control, puede configurar el retardo que tiene la imagen respecto a la inclinación y hacer que el movimiento del móvil cuadre con el del vídeo.

2.6 Realización de la actividad

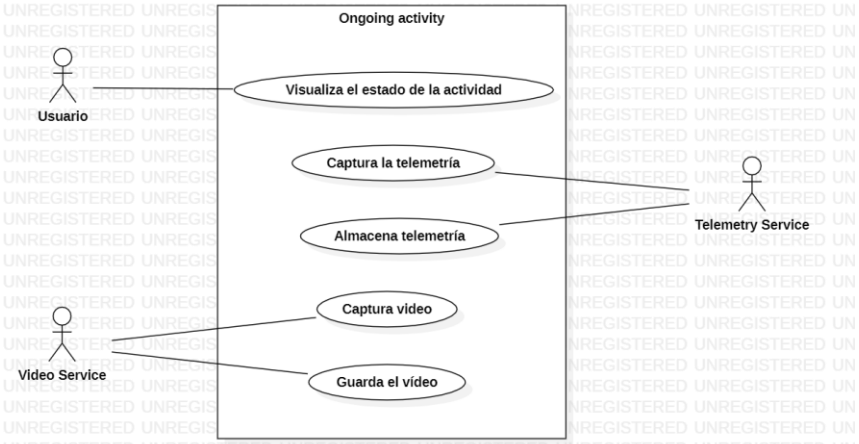


Figura 13 Caso de uso, realización de la actividad

El usuario está realizando la actividad (moto, esquí, etc). No interactúa directamente con la APP, sino que esta hace elemento contabilizador de las acciones del usuario y captura el vídeo. Los dos servicios, representados como actores muestran la interacción del sistema en el caso de uso.
El servicio de video captura cada fotograma, lo marca y lo guarda, mientras que el servicio de telemetría captura los valores en tiempo real y los guarda en una base de datos.

2.7 Generación de vídeo

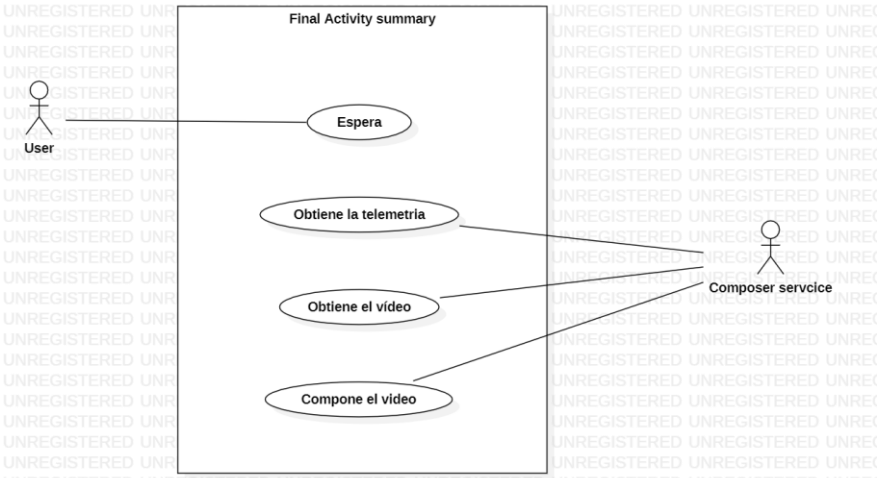


Figura 14 Caso de uso, generación del vídeo

En este caso de uso, el usuario también es “pasivo”. Una vez ha finalizado la actividad, el servicio de composición debe obtener las piezas (telemetría guardada y vídeo) y componer el vídeo final.

3. Diseño de ventanas

El diseño de las ventanas se realiza siguiendo las directrices de que sean coherentes entre sí, para que en todo momento el usuario las identifique como parte de la APP.

También se diseñan simples, fáciles de leer, con la interacción mínima posible con el usuario.

Por último, aunque por la propia naturaleza de la APP, no hay muchos subniveles, se tiene presente, que el usuario nunca tenga tres niveles de navegación, desde el Home.

Siguiendo los casos de uso y los requisitos detectados en un punto anterior, se diseñan las siguientes ventanas con la herramienta mockplus [8]:

3.1 Ventana de bienvenida

Esta ventana se muestra cuando está cargando la ventana inicial. Simplemente muestra el logo de la APP. Su objetivo es mejorar la experiencia de usuario.

Figura 15 Ventana de bienvenida

3.2 Ventana principal

Se trata de la ventana principal, que se muestra tras la carga de la APP. La ventana muestra un menú inferior de navegación, con enlaces a la ventana de configuración, sincronización de la cámara [2], la galería de android.
En esta ventana el usuario, una vez configurada la APP y conectada a una cámara WIFI, podrá empezar a grabar su actividad:

[2] Funcionalidad pendiente de revisión en la PEC3.

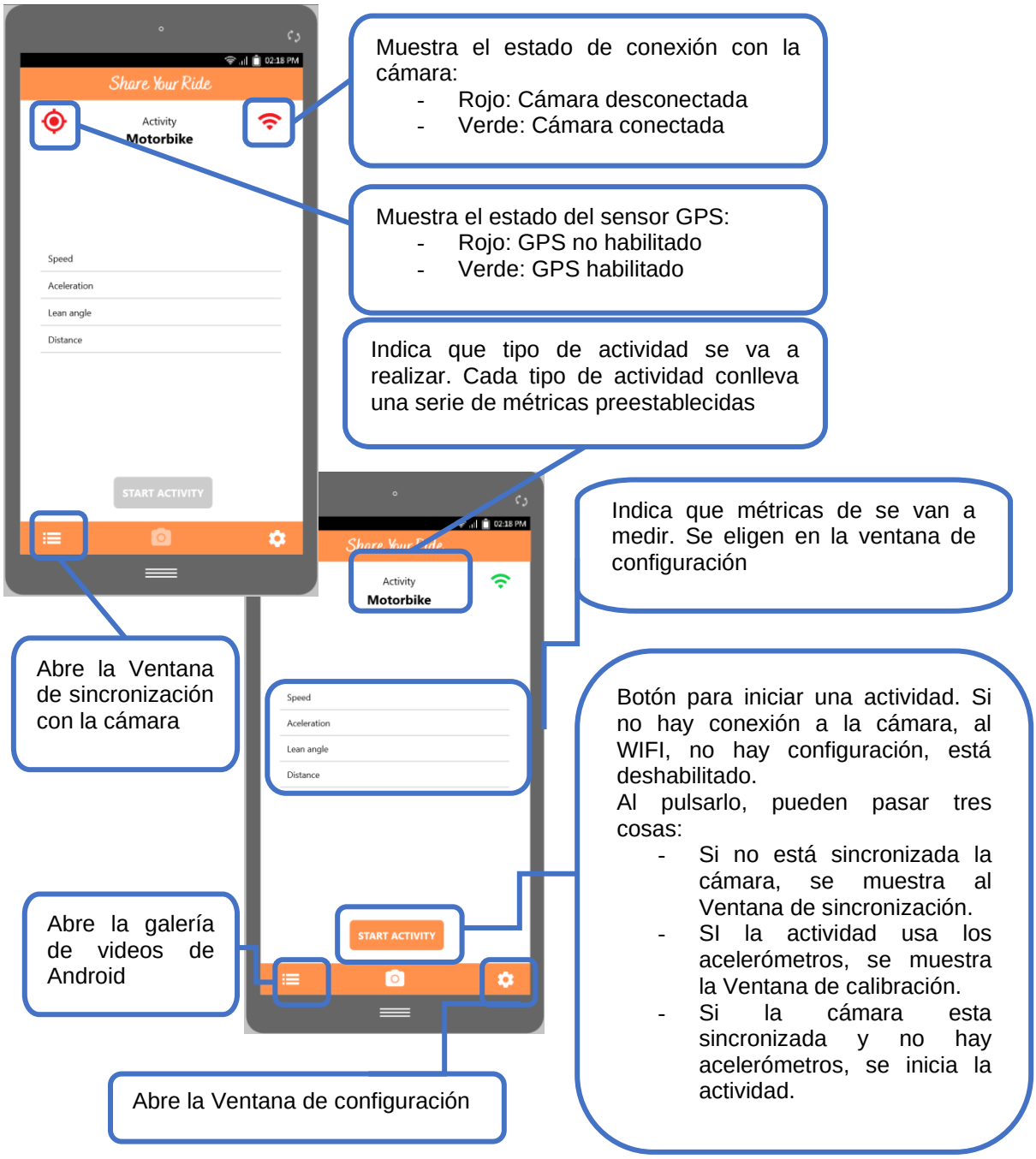


Figura 16 Ventana principal

3.2 Ventana de configuración

Se trata de la configuración de la aplicación. Se usa el estilo standard de configuración de Android y permite que el usuario configure las unidades de medida, que actividad a realizar, las métricas que usa la actividad y la configuración de la cámara. La configuración de cada ámbito se realiza en su propia ventana, las cuales se acceden a través de un menú principal:

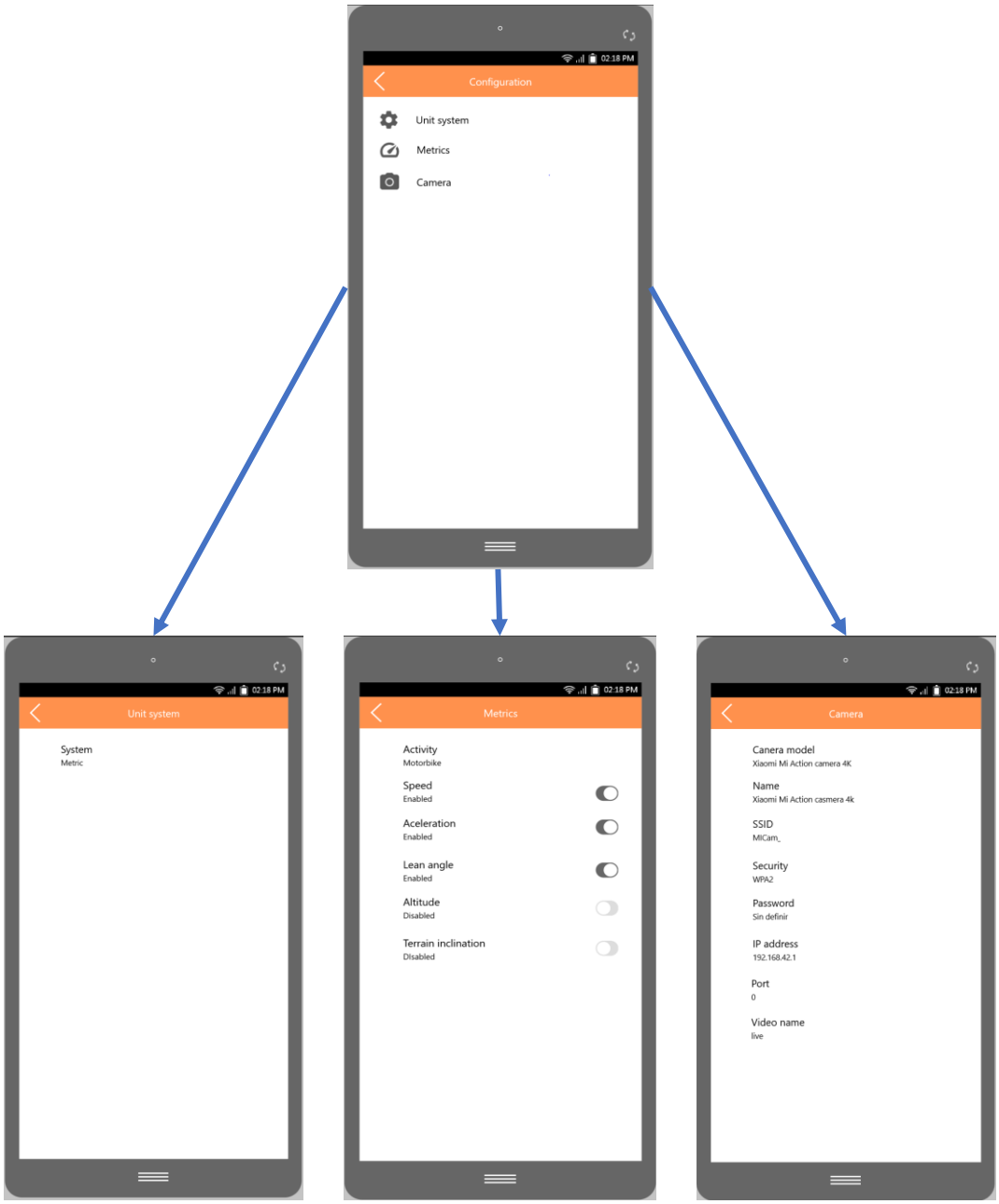


Figura 17 Ventana de configuración

3.2.1 Configuración de unidades

Permite al usuario elegir si quiere usar el sistema métrico o el sistema de unidades imperial, mediante una lista desplegable.

3.2.2 Configuración de métricas

Esta ventana muestra una lista desplegable con una serie de actividades las cuales tienen unas métricas preseleccionadas. También tiene una

opción *custom*, para que el usuario elija libremente las métricas que el desee.

Las métricas se puede seleccionar mediante un control de tipo *switch*, cada una por separado

3.2.3 Configuración de la cámara

Esta es la ventana donde el usuario puede configurar la cámara que va a utilizar y la conexión WIFI de dicha cámara. El primer control es una lista seleccionable con todas las cámaras compatibles con la APP, que han sido probadas y tienen ya una configuración preestablecida.

Sin embargo, como la APP se contacta mediante RTSP a cualquier cámara IP, se provee de una opción *custom,* donde el usuario puede introducir los valores de la conexión WIFI y del flujo de vídeo que emite la cámara.

Esto otorga mucha potencia a la APP, porque no solo puedes vincularla a una cámara de acción, sino que incluso la puedes usar con cámaras fijas que este conectadas a una red WIFI, con lo cual el usuario, puede encontra nuevos usos a la APP, por ejemplo, usar el vídeo emitido por un dron.

Los valores a configurar son los comunes para entrar en una conexión WIFI:
- SSID
- Protocolo de seguridad
- Contraseña del WIFI

Y los valores para acceder al vídeo:
- Dirección IP de la cámara
- Puerto en el que esta el servidor de vídeo en la cámara.
- Nombre que tiene el vídeo en la cámara, el cual puede ser solo una palabra, o una ruta de carpetas.

3.3 Ventana de calibración de sensores

Antes de poder empezar a grabar la actividad y capturar la telemetria del móviles, es necesario saber cual es la rotación relativa del móvil en la posición y soporte donde se va a colocar.

Por ese motivo, en la ventana de calibración se insta al usuario a que coloque el móvil en reposo en el soporte, Durante unos segundos, cuando el móvil esté en reposo, se calculará la posición de referencia del móvil y se habilitará el botón de continuar, que dará paso a la actividad en sí.

Pulsando hacia atrás se muestra la ventana de sincronización de vídeo.

Figura 18 Ventana de calibración de sensores

3.5 Ventana de sincronización de vídeo

Como se ha comentado en anteriores apartados, esta ventana es una forma de paliar el retardo de que puede tener el vídeo desde que es captado por la cámara hasta que se recibe en el móvil. Un retardo de unos cientos de milisegundos es aceptable, pero más de un segundo, sería demasiado apreciable por el usuario.
Para la sincronización, el usuario debe utilizar los controles dados en la interfaz para conseguir que el vídeo y el indicador de inclinación vayan sincronizados.

Pulsando el botón de atrás se vuelve a la ventana principal.

Figura 19 Proceso de sincronización de vídeo

3.5 Durante la actividad

Cuando el usuario inicia una actividad, la aplicación deshabilita los botones de navegación de la barra inferior, para evitar incoherencias en su funcionamiento, además le muestra al usuario un *feedback* del estado actual. También le permite terminar la actividad ya sea pulsando el botón de finalizar, como pulsando en “atrás”.

Figura 20 Ventana de actividad en curso

Si se pierde la conexión de cámara se termina la actividad (generando el vídeo obtenido hasta esa fecha).

3.6 Creación del vídeo

Una vez finaliza la actividad, se muestra la ventana de creación de vídeo. El usuario únicamente tiene esperar mientras la APP genera el vídeo en segundo plano:

Como *feeadback* se muestra una barra de progreso y los valores totales de la actividad.

Figura 21 Ventana de creación del vídeo

Varios iconos han sido obtenidos en:

- Free pick
- Material desing
- Otros los he diseñado yo con canva y guardados como diseños vectoriales.

3.7 Navegación

Aunque durante la explicación de las ventanas, ya se describe de donde provienen y lo que puede hacer el usuario, a continuación, se incluye el diagrama de navegación de la APP. En él, se muestra la relación entre las distintas ventanas y las acciones de usuario que desencadenan la transición de una ventana a otra:

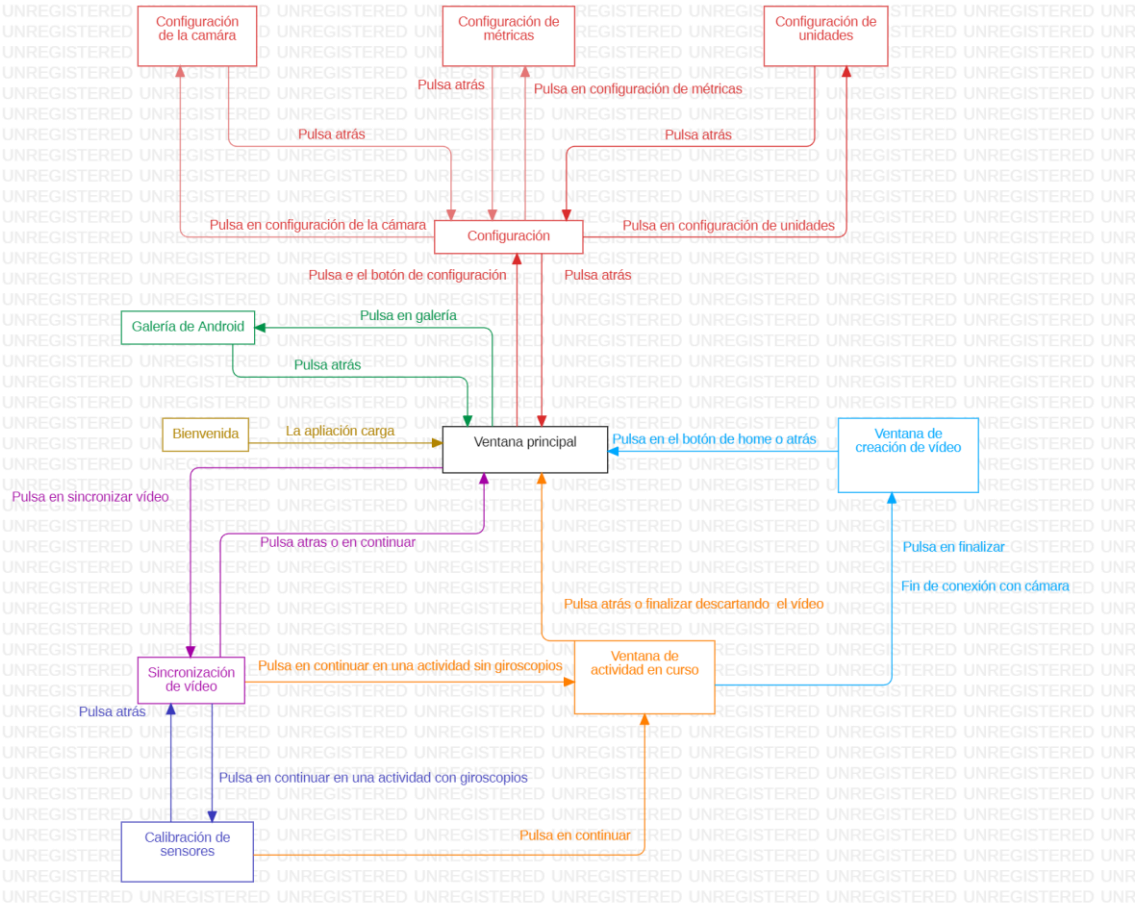


Figura 22 Diagrama de navegación

Los colores son únicamente para facilitar la legibilidad del diagrama.

4. Arquitectura de software

4.1 Componentes

El primer paso es definir en bloques generales, qué partes componen la APP

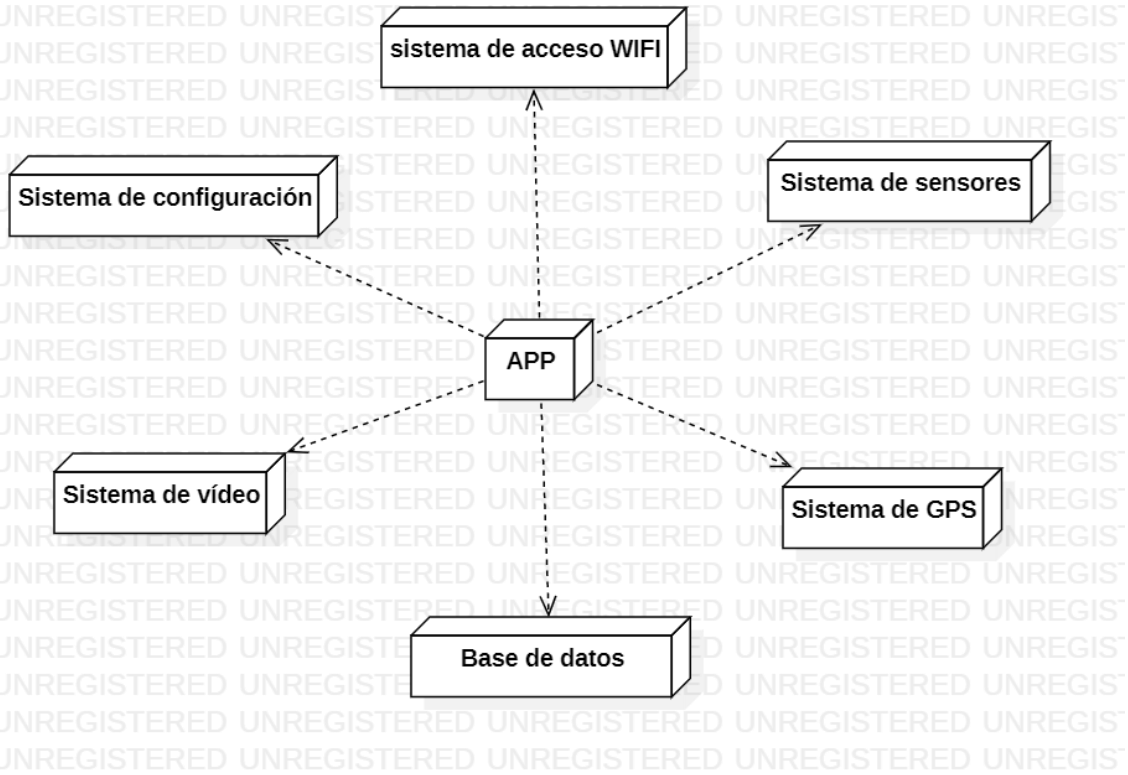


Figura 23 Diagrama de despliegue

Tenemos una serie de sistemas que van a formar parte la APP, cada uno con una responsabilidad:

- ***Sistema de configuración***: Guarda la configuración que ha insertado el usuario y que aplicará a la actividad.
- ***Sistema de video***: Obtiene los flujos de vídeo de la cámara y compone un vídeo con la telemetría.
- ***Sistema de GPS***: Maneja el sensor GPS del móvil, obtiene las variaciones de la posición para calcular altura, velocidad y distancia.
- ***Sistema de sensores***: Maneja los giroscopios y acelerómetros del móvil para obtener la inclinación y las fuerzas que afectan al móvil.
- ***Sistema de acceso WIFI***: Se encarga de buscar la red WIFI de la cámara y gestionar la conexión con ella.
- ***Base de datos***: Persiste la información de la actividad presente así como los datos de actividades pasadas.
- ***APP***: La parte lógica de la actividad y las vistas que se le muestran al usuario. Es el componente que gestiona el comportamiento y con el que interactúa el usuario.

Ahora vamos a pormenorizar un poco estos bloques desgranándolos en una serie de componentes y cómo se relacionan entre ellos en alto nivel:

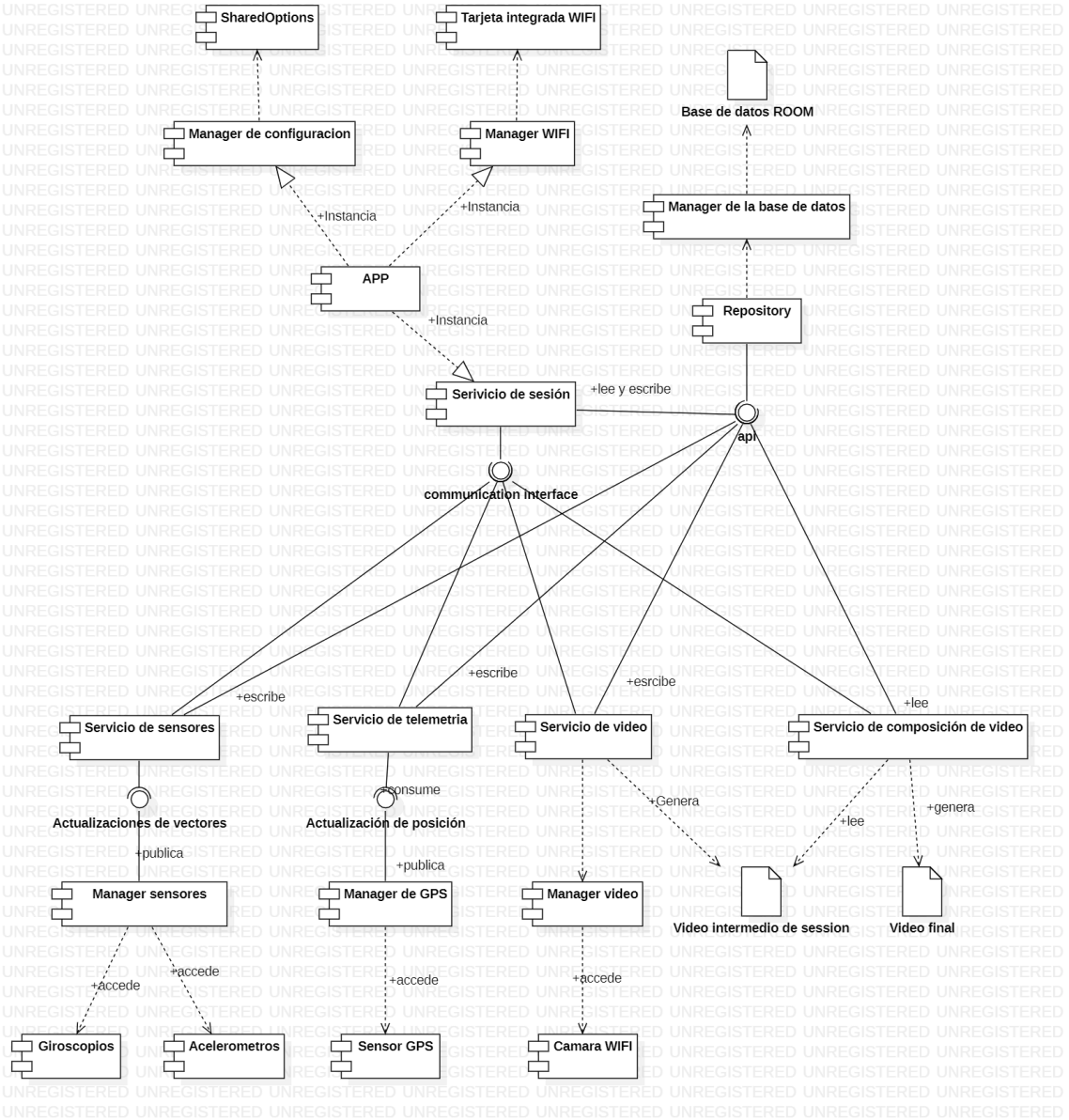


Figura 24 Diagrama de componentes

4.2 MVVM

Para desacoplar la vista de la lógica de negocio y el modelo de datos, se va a utilizar el patrón MVVM (Model – View – View Model). Los puntos clave de este patrón son:

- El View Model está totalmente desacoplado de la vista
- El View Model tiene la lógica de representación y se comunica con la vista con eventos (LiveData en Android).
- La vista con el patrón observer detecta los cambios en el View Model y los representa. Son vistas dinámicas.

Particularidades de Android;

- Los View Models y las Actividades pueden ser destruidos por el sistema Android en cualquier momento.
- Si no se quiere tener *memory leaks*, hay que tener cuidado con los Live cycle owners que observan.
- Mediante Data bindings, se puede programar el Observable directamente en el XML de la vista en vez de en la actividad.

Si le ponemos “nombre” a cada capa del patrón, este sería:

- Modelo: Repositorio, ROOM y Servicios.
- View: Activities y Fragments
- ViewModel: AndroidViewModels

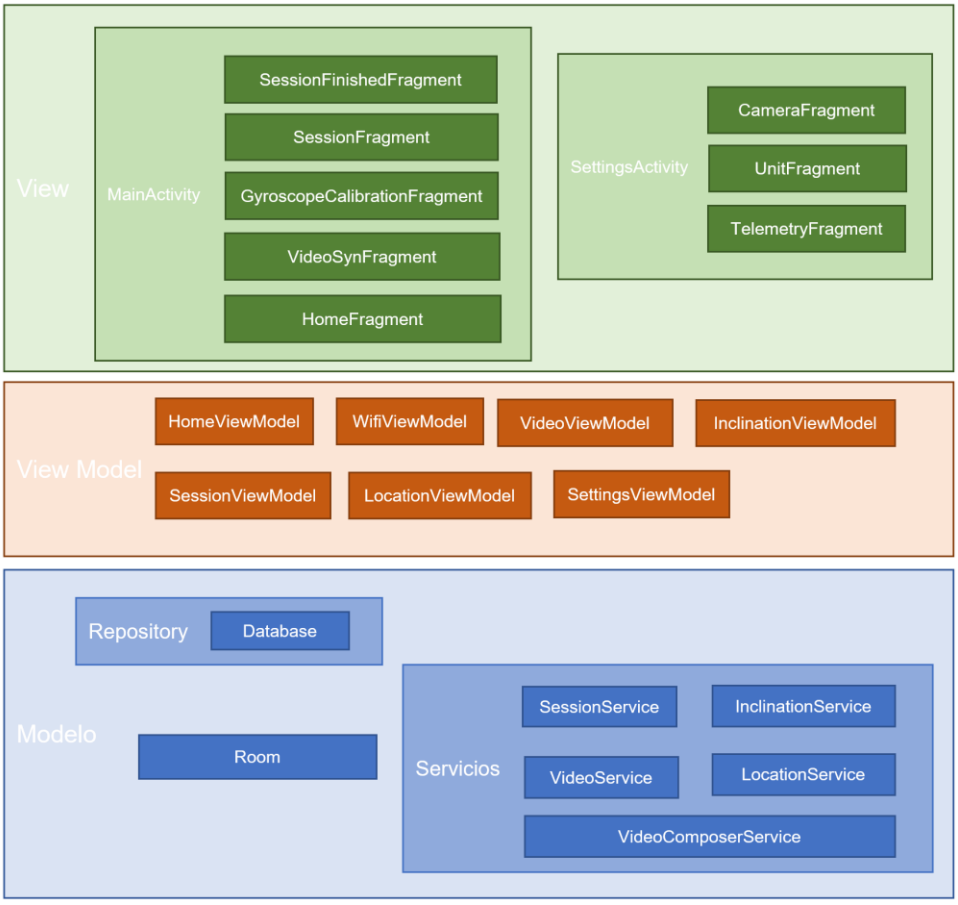


Figura 25 Distribución MVVM

4.3 Servicios

Como se puede observar en el diagrama de componentes, Para la gestión del vídeo, los sensores de GPS, los acelerómetros, la conversión de vídeo y el manejo de la sesión, voy a utilizar servicios. Estos se usan porque todas son tareas susceptibles de realizarse en segundo plano y tenemos que prevenir que el sistema Android limite el acceso a los recursos o simplemente mate los procesos que obtienen la telemetría, el vídeo o realizan la conversión del vídeo. Concretamente vamos a utilizar los servicios de tipo *foreground*, tal y como se explica en [9] y [10].

El resultado final deseado es que los servicios sean independientes los unos de los otros del resto de componentes de la APP. Es decir, una vez configuradas e iniciadas las tareas de las que son responsables, estos servicios continuarán funcionando de forma autónoma.

Siguiendo el criterio de *Single responsability* se define un servicio por responsabilidad, por lo tanto, tenemos el siguiente listado de servicios:

- ***Servicio de sesión***: Se encarga de administrar la actividad de usuario y orquesta al resto de servicios.
- ***Servicio de acceso a sensores***: Se encarga de procesar y guardar los datos de los acelerómetros y giroscopios.
- ***Servicio de GPS***: Se encarga de la localización del móvil, procesa y almacena los eventos de GPS.
- ***Servicio de Video***: Procesa el flujo de vídeo, marca los frames y los almacena en disco.
- ***Servicio de composición de vídeo***: Una vez finalizada la actividad accede al vídeo guardado y los datos almacenados por el resto de los servicios. Con estos datos compone el vídeo.

4.4 Mensajería

Los servicios tienen que comunicarse entre ellos y con los View Model, pero hay que evitar que existan acoplamientos y dependencias entre ellos. He estado leyendo la documentación e intentando encontrar un método que permita que servicios en segundo plano puedan mandar eventos a View Models.

Las dos opciones que he barajado antes de decidir usar un sistema de mensajería han sido:

- Crear servicios de tipo “*bindables*”.
- Acceder a las instancias de los servicios creadas automáticamente por el sistema.

Con la primera opción, no he encontrado garantías de que el servicio pueda ser también de tipo *foreground* y que no sea eliminado por el sistema al *unbindear* los View Models. Además, este concepto, parece que supedita el servicio al View Model y crea una arquitectura más jerarquizada o acoplada, mientras que yo busco una arquitectura distribuida,

Con la segunda, aunque los servicios son creados como *singleton*, no he encontrado cómo acceder a esa instancia de una forma limpia en un repositorio. Seguro que existe alguna, y esta implementación me la guardo para investigar en futuras APPs.

Por lo tanto, voy a implementar un sistema de mensajería que comunique tanto los servicios entre sí, como a los View Models con los servicios. Personalmente, esta estrategia la uso en otros entornos MVVM para la comunicación entre capas y permite minimizar los acoplamientos entre ellas, además de simplificar las cadenas de

callbacks y eventos retransmitidos que se pueden dar a veces esta arquitectura. También permite tener una arquitectura distribuida, con una abstracción total de las otras capas.

El servicio de mensajería va a consistir en un ***broker*** de mensaje que enruta los mensajes entre los distintos clientes que se hayan suscrito.

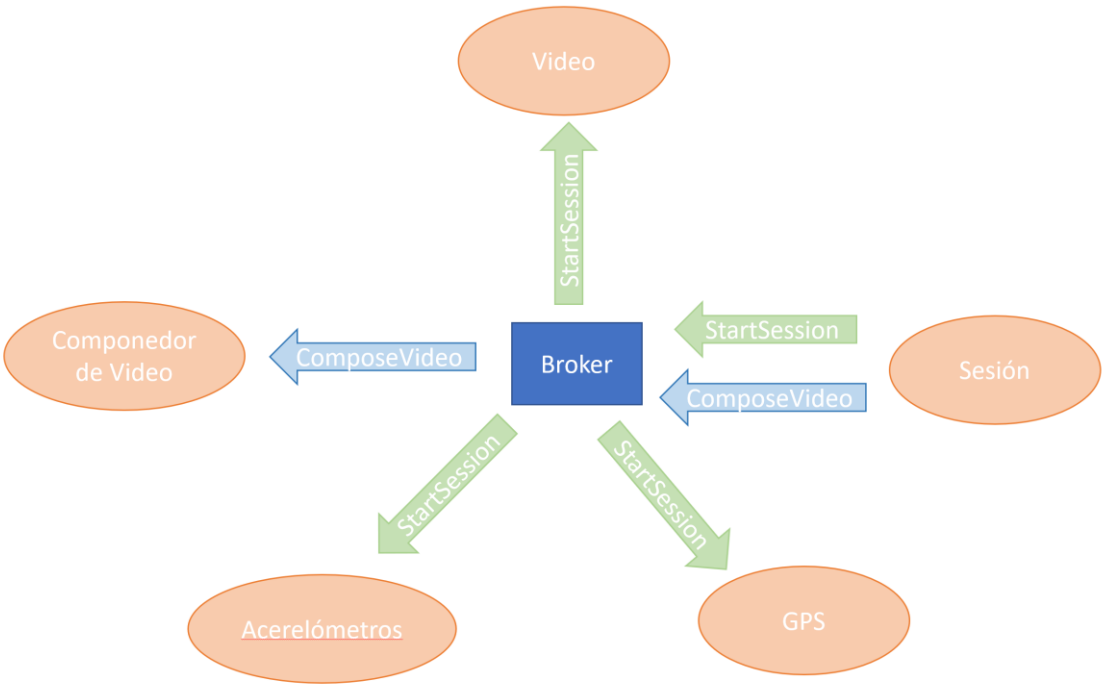


Figura 26 Esquema ejemplo del sistema de mensajería

En el anterior ejemplo, se muestra como el *broker*, va a determinar que cliente destina un mensaje. El cliente Sesión no dice a quién van destinados los mensajes, sino que solo dice que los mensajes son de tipo StartSession y ComposeVideo. El resto de los clientes al suscribirse, indican que tipo de mensajes quieren recibir y el bróker se los envía.

No deja de ser un sistema de eventos y de *callbacks*, pero organizado y escalable.

Para esta tarea, Android provee los Intents, sin embargo, como se va a requerir un uso intensivo de la mensajería para coordinar las acciones de todos los componentes, dispongo de una librería propia que usa colas creadas de forma estática de fácil uso, que va a ser la opción elegida.
Actualmente se definen los siguientes mensajes:

Filtro	Mensaje
SESSION_COMMANDS	StartSession

	EndSession
	DiscardSession
	CancelSession
	ContinueSession
	RetryCalibration
	SessionStateEvent
	SessionStateRequest
	SessionSummaryRequest
	SessionSummaryResponse
SESSION_CONTROL	StartAdquiringData
	StopAdquiringData
	ConfigureVIdeoDelay
	SaveTelemetry
	UpdateTelemetry
GPS_DATA	GpsStateRequest
	GpsStateEvent
	GpsDataEvent
	GpsStartAcquiringAccuracy
VIDEO_DATA	VideoConnectionData
	VideoStateRequest
	VideoStateEvent
	VideoDiscardCommand
	VideoSynchronizationCommand
VIDEO_SYNCHRONIZATION_DATA	ConfiugreVideoDelay
	VideoFrameSynchronizationData
	LeanAngleSynchronizationData
VIDEO_CREATION_DATA	VideoCreationCommand
	VideoCreationStateEvent
INCLINATION_CONTROL	InclinationDataEvent
	InclinationCalibrationStart
	InclinationCalibrationEnd

Tabla 6 Listado de mensajes

La explicación de los mensajes se incluye en el anexo “Mensajería”.

4.4 Flujo de la información

Los eventos que producen los sensores no se envían sin control a las capas superiores, esto podría colapsar fácilmente la APP. Hay que controlar el flujo de la información y cómo se envía. Esto se controla de la siguiente forma:

- El servicio Session tiene dos temporizadores uno que gobierna cuando se escribe en base de datos y otro cuando se envían eventos con los nuevos valores a las capas de arriba para que sean representados en la vista. El temporizador para actualizar la vista tiene menos frecuencia.
- Los servicios de captura de datos al recibir estos mensajes:
    - Guardan la información en la base de datos

    - Envían un mensaje a las capas superiores con la información captada por sus sensores.
- Los View Models al recibir los mensajes, actualizan sus propiedades LiveData,
- La vista observa el LiveData de los View Models, y actualiza los valores de representación cuando los valores de los campos LiveData cambian:

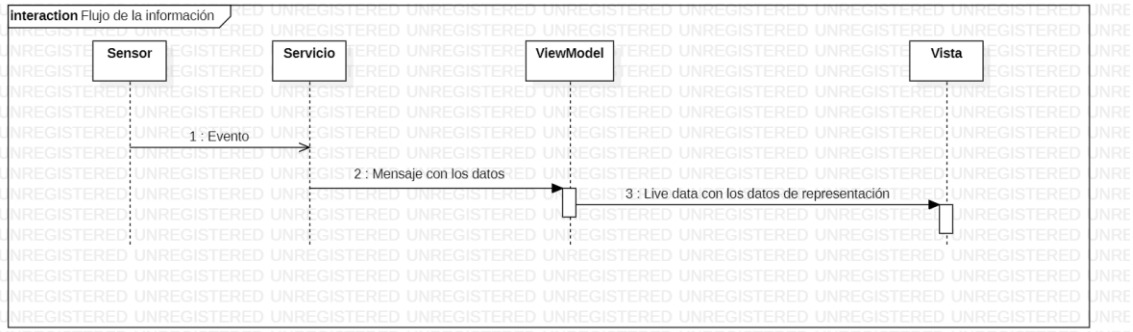


Figura 27 Diagrama de secuencia de ejemplo

El flujo de mensajes e información para cada funcionalidad se detallará en cada uno de sus apartados.

4.5 Hilos de ejecución

La aplicación va a hacer un uso intenso del hardware y va a acceder a los distintos sensores que ofrece el móvil, eso supone varios hándicaps:

- El procesamiento del flujo del video en vivo tiene no puede pararse porque la aplicación está haciendo otras tareas, ni se pueden perder frames.
- Hay que manejar multitud de eventos en tiempo real de los sensores.
- Un hilo que corre en segundo plano no puede actualizar la interfaz de usuario.
- Las operaciones con la base de datos no pueden hacerse en el hilo principal.
- Dos hilos distintos no pueden acceder al mismo tiempo a la base de datos.
- Los servicios corren en el hilo principal por defecto

Por este motivo se plantea la siguiente organización:

- Hilo de gestión de sensores que es implementado por el servicio de sensores
- Hilo de gestión de GPS que es implementado por el servicio de GPS
- Hilo de gestión a vídeo que es usado por el servicio de video
- Hilo para la composición de vídeo que es usado por el servicio de composición de vídeo.

- El servicio de sesión y el resto de la APP va a correr en el hilo principal (salvo cuando tengan que acceder a base de datos que se requiere otro hilo).

Room ya tiene mecanismos que la vuelven *threadsafe*, las operaciones de base de datos que vamos a realizar son atómicas, por lo tanto, cada operación es una transacción, pero si fuera necesario realizar alguna operación múltiple, todas las acciones de la operación se marcarían como una única transacción [11].

Si los servicios notifican por eventos a capas superiores, usaran tipos de LiveData para tal efecto, utilizando acceso a los datos mediante acciones de tipo post. Los observers de la Aplicación (por ejemplo, View Models), escucharan los cambios en el hilo principal.

4.6 Base de datos

Los datos de la actividad que realiza el usuario se van a almacenar en una base de datos, esto va a permitir que si alguna de las actividades o de los View Models que manejan la actividad son destruidos por el sistema Android, cuando la APP pase al primer plano, el usuario no perderá datos.

Del mismo modo, esto permite que si en un futuro se decide añadir nuevas funcionalidades, como puede ser una galería, el modelo de datos ya estará preparado.

4.6.1 Repositorio

Se trata de un patrón de diseño el cual abstrae la implementación del origen de los datos (una base de datos Room en este caso) a los clientes que hacen de eso.

En el caso que nos aplica, el acceso a la base de datos lo realizan los servidores, por lo tanto, se va a realizar una API estática que realiza accesos asíncronos a la base de datos.

4.6.2 Diseño de la base de datos

Para realizar una base de datos ROOM se requiere definir una serie de entidades:

Session	
Descripción	Guarda los datos de la actividad que realiza el usuario
Clave primaria	ID: Identificador único por sesión
Clave foránea	No tiene
Relaciones	- 1 a 1 con SessionTelemetry - 1 a n con Video - 1 a n con Location - 1 a n con Inclination

Tabla 7 Entidad Session

SessionTelemetry	
Descripción	Guarda los valores que el usuario ha configurado para realizar la actividad. Se sitúan en una entidad a parte ya que es previsible que crezca durante las versiones venideras, y por lo tanto quedará más organizado y segmentado.
Clave primaria	SessionId: Identificador único por sesión
Clave foránea	SessionId corresponde al campo id de session
Relaciones	1 a 1 con Session

Tabla 8 Entidad SessionTelemetry

Video	
Descripción	Contiene información de cada fotograma recibido.
Clave primaria	Dupla ID: Identificador único por sesión timeStamp: El momento en milisegundos de sincronización de las tablas. Sirve para coordinar toda la telemetría y el vídeo.
Clave foránea	SessionId corresponde al campo id de session
Relaciones	N a 1 con Session

Tabla 9 Entidad Video

Inclination	
Descripción	Contiene los valores captados por los giroscopios y los acelerómetros.
Clave primaria	Dupla ID: Identificador único por sesión timeStamp: El momento en milisegundos de sincronización de las tablas. Sirve para coordinar toda la telemetría y el vídeo.
Clave foránea	SessionId corresponde al campo id de session
Relaciones	N a 1 con Sesión

Tabla 10 Entidad Inclination

Location	
Descripción	Contiene los valores captados por el GPS.
Clave primaria	Dupla ID: Identificador único por sesión timeStamp: El momento en milisegundos de sincronización de las tablas. Sirve para coordinar toda la telemetría y el vídeo.
Clave foránea	SessionId corresponde al campo id de session
Relaciones	N a 1 con Session

Tabla 11 Entidad Location

Las columnas de cada entidad (tabla), se puede ver en el siguiente diagrama E-R y los nombres son auto explicativos, así que no se entra en detalle de ellos.

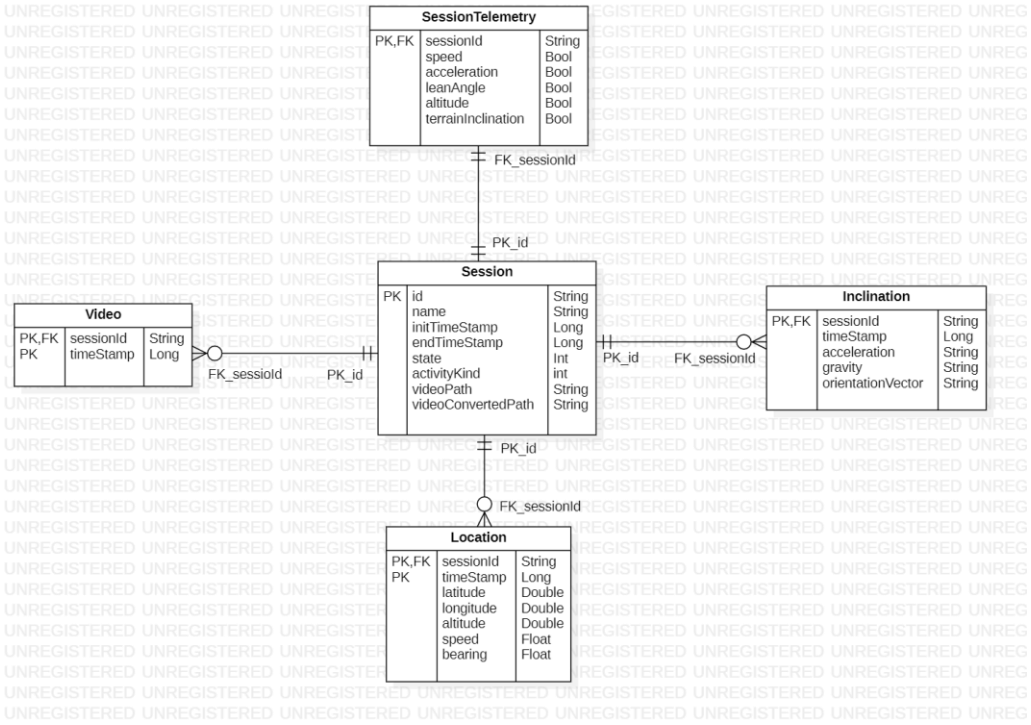


Figura 28 Diagrama entidad – relación de la base de dato

5. Implementación

En este apartado voy a relatar los puntos más importantes del desarrollo. No voy a entrar en comentar el código ni a poner fragmentos de él, sino que voy a relatar las diferentes máquinas de estados, el flujo de la acción y en los casos que sea relevante, los diagramas de clases. Lo que sí es necesario es conocer la distribución del código:

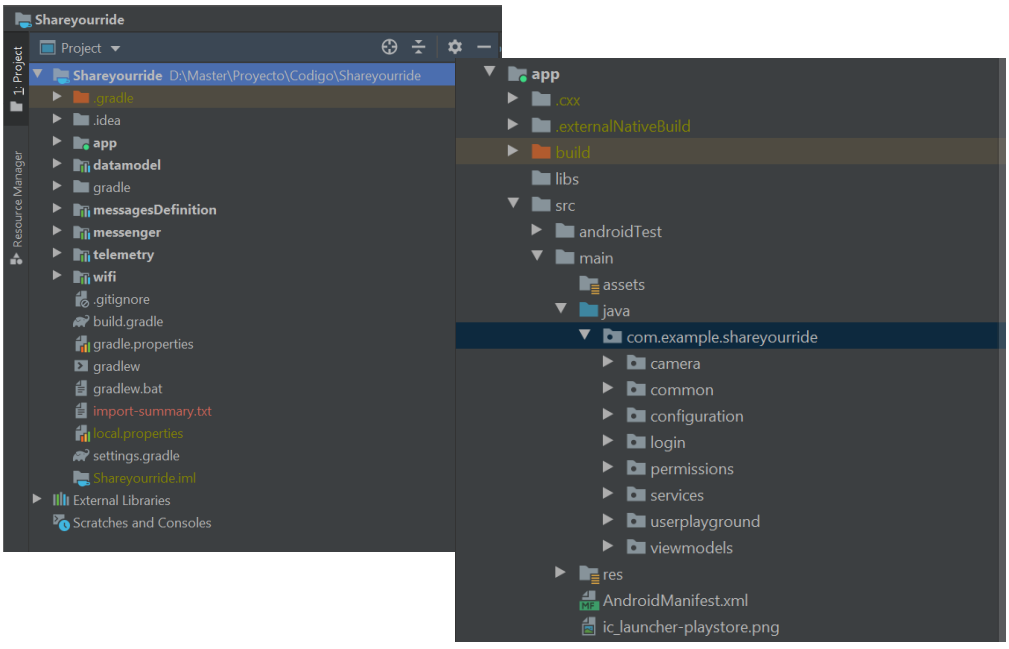


Figura 29 Proyecto de Android Studio

- **App**: Contiene la lógica de negocio de la aplicación.
- **Datamodel**: Contiene las entidades de la base de datos ROOM, la definición de la base de datos, el repositorio y APIS de acceso.
- **MessageDefinitions:** Contiene la definición de todos los mensajes intercambiados en el sistema de mensajería.
- **Messenger**: Sistema de mensajería interno.
- **Telemetry**: Contiene la parte de control del GPS y los sensores.
- **Wifi**: Control del WIFI del dispositivo Android.

Figura 30 Proyecto de Android Studio

- **Camera**: Definición del listado de todas las cámaras compatibles.
- **Configuration**: Implementa el manager de las shared preferences.
- **Permissions**: Implementa el manager de permisos.
- **Services**: Contiene todos los servicios de la APP.
- **Userplayground**: Contiene las vistas de la APP.
- **Viewmodels**: Contiene todos los View Models de la APP.

5.1 Sistema de mensajería

Como es un sistema de mensajería particular que he desarrollado para el proyecto voy a entrar un poco más en detalle de la codificación y uso de él.

El sistema de mensajería esta implementado en el módulo Messenger y consta de los siguientes componentes:

- **MessageQueueManager**: Clase estática que contiene las instancias de los handlers de todos los clientes. Los añade, o elimina del listado de clientes. También es la que realiza el envío de los mensajes
- **MessageHandler**: Deriva de la clase Handler de Android (cola de mensajes).
- **IMessageHandlerClient**: Interfaz la cual obliga a la clase que la use a implementar un método processMessage, esta función se invocará siempre que un cliente reciba un mensaje. También ofrece el método sendMessage el cual envía un mensaje al resto de clientes.
- **MessageBundle**: Es el objeto que contiene la información intercambiada entre los clientes, un filtro para discriminar a donde es enviado, el tipo de mensaje, el tipo de datos incluido y los datos en sí.

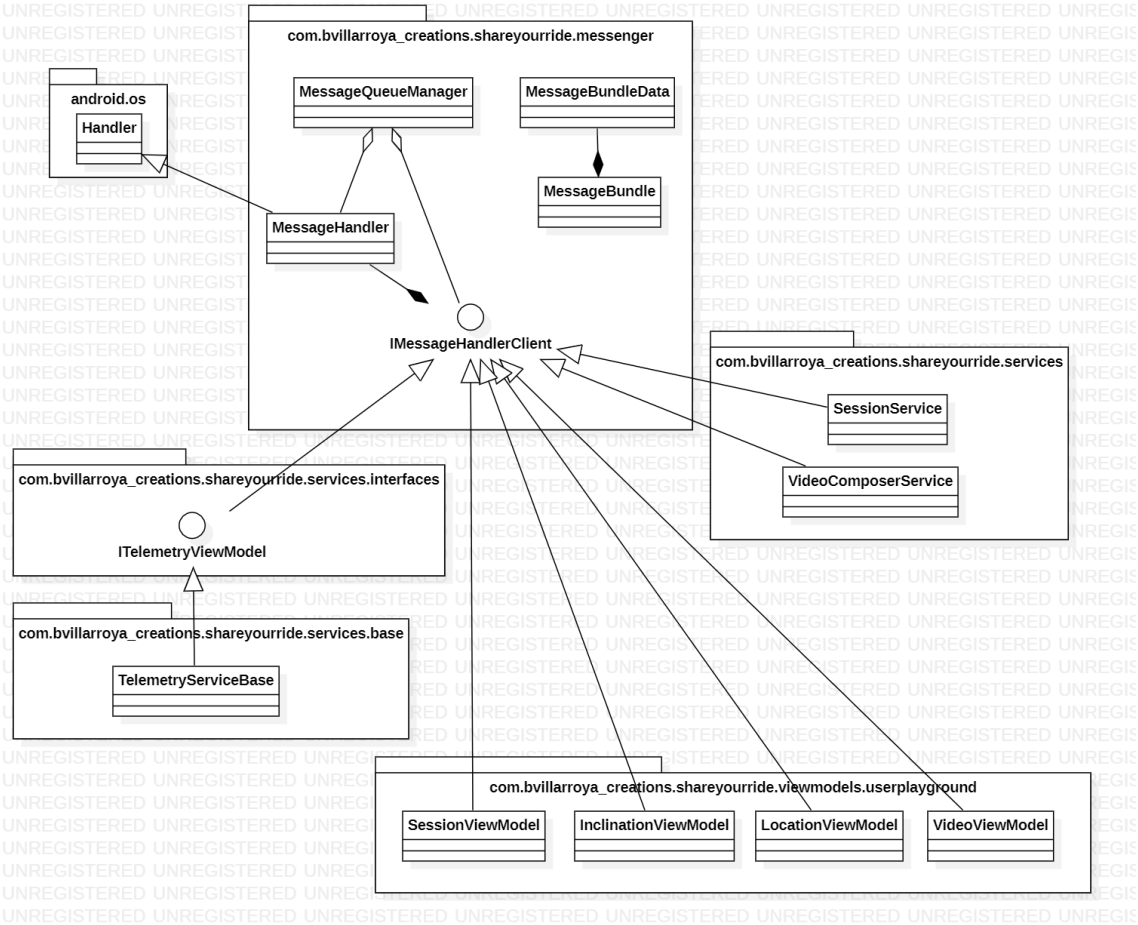


Figura 31 Diagrama de clases de del sistema de mensajería

El funcionamiento es relativamente sencillo:

1- Los clientes se suscriben al sistema de mensajería en su función init. En dicha subscripción indican que filtros de mensajes quieren recibir.
2- Cuando quieren enviar un mensaje, componen un Bundle, en el cual se rellena a que filtro pertenece, el tipo de mensaje, los datos y los tipos de datos.
3- Se envía el bundle de datos al mánager.
4- El mánager, haciendo uso del filtro dado, determina a que clientes va destinado el mensaje e inserta en sus colas de mensajes el bundle. Si no se especifica un filtro de mensaje, el envío se realiza a todos los clientes.
5- El manejador de colas cliente detecta el nuevo mensaje e invoca la función de callback del cliente para procesar el mensaje.
6- El cliente dependiendo del tipo de mensaje determina que tiene que hacer con él y accede a los datos que lleva el mensaje.

En el siguiente ejemplo basado en un caso de uso real dentro de la aplicación, muestro como es flujo del envía de un mensaje en alto nivel:

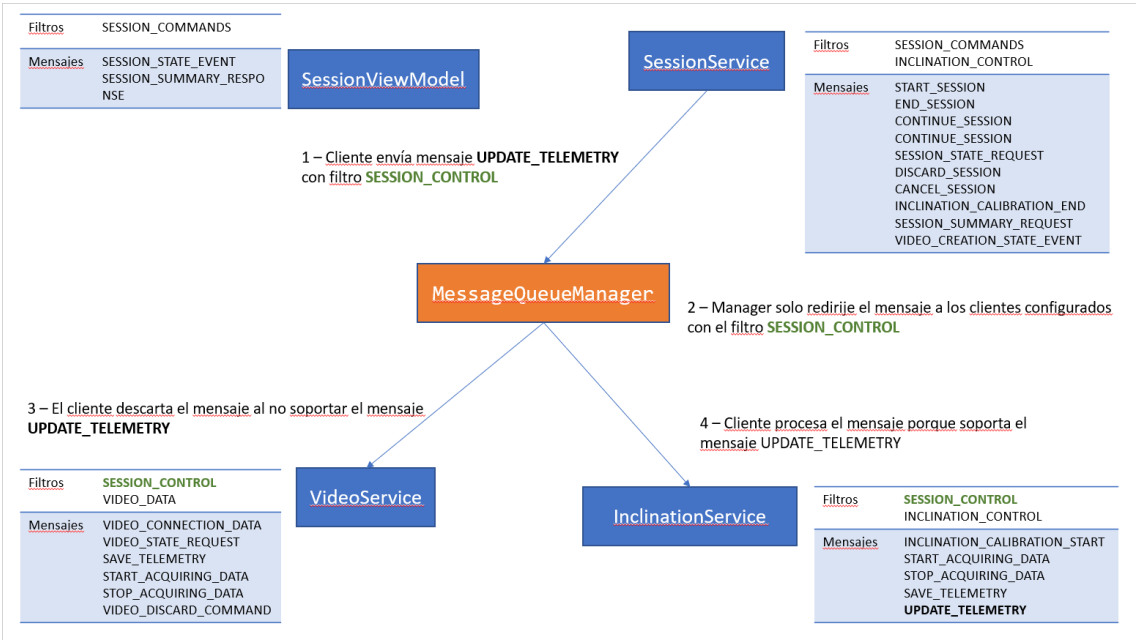


Figura 32 Envío de un mensaje interno

El diagrama no muestra todos los clientes ni mensajes por cuestiones de legibilidad.

Los mensajes y los filtros utilizados se definen en el módulo, messageDefinition, para evitar acoplar la aplicación y el sistema de mensajería, el cual pude ser usado en cualquier otro proyecto sin necesidad de modificar una línea de código. Todos los mensajes se encuentran explicados en el apartado 4.4 Mensajería

5.2 Configuración

La configuración de la aplicación se realiza mediante el sistema de configuración de Android que forma parte del framework de Android Jetpack [21]. Se crea la actividad SettingsActivity y los fragmentos UnitFragment, CameraFragment y TelemetryFragment, para representar la jerarquía de ventanas y configuración.
Para el acceso del resto de la aplicación a la configuración, se crea la clase SettingsPreferenceGetter. Donde todas las configuraciones se mapean a distintas enumerados para que su acceso sea más sencillo y no sea necesario usar la cadena de texto usada como clave en el sistema de configuración (abstraemos el modelo al resto de la aplicación).

Para las actividades o fragmentos que tienen que detectar en caliente los cambios de configuración, se crea el SettingsViewModel, que mediante LiveData, va a notificar los cambios de la configuración.

Lo único destacable en esta parte es el uso de reactive para controlar el flujo de eventos enviados desde el SettingsViewModel a los fragmentos

que acceden a él. Esto se usa con la herramienta *debounce* de las librerías de reactive [22].

Se utiliza este método porque nos interesa que cuando la configuración de la actividad o de la cámara cambie, se actualice el resto de la actividad, pero no interesa que reaccione a cada cambio de cada componente dentro de la jerarquía de la configuración, sino que reaccione al conjunto de cambios una única vez. Con debounce se establece un temporizador de “x” segundos. El temporizador arranca cuando hay un cambio en la configuración y cuando espira el temporizador, se envía el evento al resto de capa. Sin embargo, si la configuración cambia cuando aún no ha expirado al temporizador, este vuelve a empezar, dando como resultado que 10 cambios de configuración muy rápidos solo producen un evento de cambio para las capas que observan dicha información.

5.3 Gestión de conexiones WIFI

5.3.1 Implementación

Este es un punto complicado difícil de resolver. El comportamiento deseado de la APP es que es una vez está conectada la cámara, el móvil detecte su red WIFI y se conecte automáticamente. El primer hándicap es que Google ha cambiado el API de gestión de las conexiones en Android 10 [23], por lo tanto, se necesitan dos implementaciones distintas.

A nivel lógico con la API antigua y la nueva, el procedimiento es similar, en ambos hay que construir un objeto con los datos de la red (**WifiConfiguration** en el API viejo y **WifiNetworkSpecifier** en la nueva API). En estos datos hay que especificar el SSID y las credenciales (sin contraseña, WPA, WPA2 o WPA3). Los pasos difieren en el hecho de conectarse a la red:

- En el API viejo nos conectamos directamente con la herramienta **WifiManager**
- En el API nuevo, podemos añadir la nueva red como sugerencia al **WifiManager** e instar al usuario a que se conecte a esa red, o forzar la conexión mediante la herramienta **ConnectivityManager**.

Mi experiencia con la API para Android 10, es que el método de sugerir una red WIFI no siempre funciona y a veces no se muestra la ventana que insta al usuario a seleccionar una red, sobre todo cuando el móvil ya está conectado a otra red WiFI. Por ese motivo opto por usar también el método de forzar la conexión con el ConnectivityManager. De este modo la nueva red WIFI se guarda en el listado de redes sugeridas del móvil para una reconexión rápida y nos aseguramos de que el móvil se conecte a la WIFI de la cámara automáticamente (siempre que el usuario acepte).

Otro punto importante es que para poder manejar las conexiones WIFI del móvil y subscribirse a los eventos del estado de conexión del móvil, primero hay que garantizar los siguientes permisos:

- CHANGE_WIFI_STATE
- ACCESS_WIFI_STATE
- CHANGE_NETWORK_STATE
- ACCESS_NETWORK_STATE

La implementación es bastante estándar y obedece a los ejemplos de Google [24], por lo tanto, no me voy a explayar en este aspecto, el código se encuentra en módulo wifi de la solución y está completamente comentado:

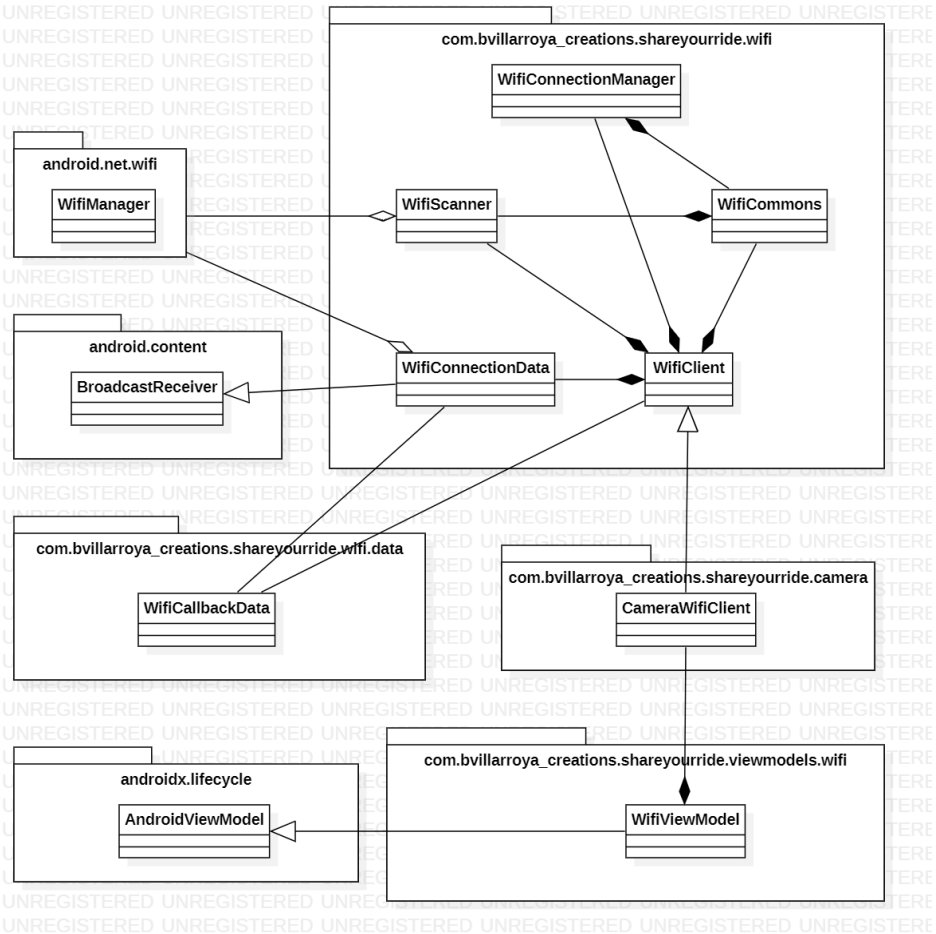


Figura 33 Diagrama de clases de la implementación de conexión WIFI

5.3.1 Operativa

Este aspecto me resulta más interesante, ya que se trata de conseguir que la gestión del WIFI no sea obstáculo para el usuario a la hora de usar la APP.
Consta de los siguientes componentes o responsabilidades:

- Comprobación del estado del dispositivo
- Comprobación del estado de la conexión.

- Escaneo de redes WIFI y detección de la WIFI configurada
- Conexión de la red de la cámara

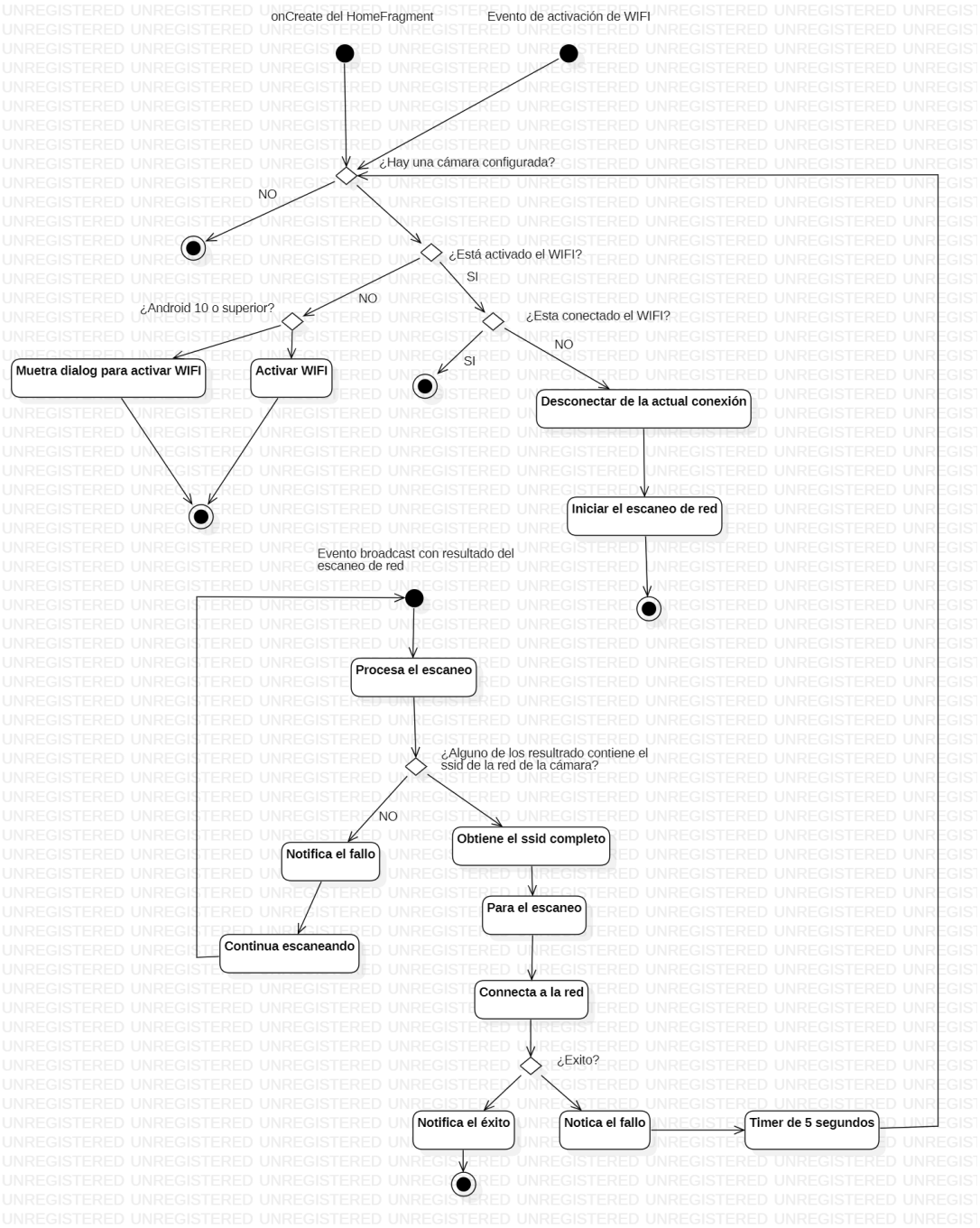


Figura 34 Diagrama de flujo de conexión WIFI

5.4 Gestión de la telemetría

5.4.1 Capas y distribución del código

La parte del código encargada del acceso a los servicios de telemetría del móvil se implementan en el módulo ***Telemetry***

Se ha organizado el código de tal forma que todos los ámbitos de la telemetría implementen las mismas clases e interfaces. La comunicación con el resto de la aplicación es la misma en todos los casos. Aunque inicialmente esto ha supuesto un sobrecoste en el desarrollo, va a permitir en un futuro, reducir el coste de la integración de nuevas funcionalidades como constantes vitales, condiciones atmosféricas o telemetría *onboard* OBD.

El sistema de telemetría consta de los siguientes componentes:

- **DataProvider**: Proveedor de datos que hace uso de las APIs del sistema para subscribirse a los eventos de telemetría que correspondan. Por ejemplo, actualizaciones de la posición GPS.
- **TelemetryManager**: Clase que conecta el servicio con el proveedor de datos. Captura los eventos emitidos por proveedor con la telemetría y realiza un primer procesado de la información.
- **TelemetryData**: Se trata del contenedor que contiene la información de telemetría. Se usa para enviar los datos del proveedor al mánager y del manager al servicio.
  Es útil utilizar una clase propia y no las aportadas por el sistema para minimizar los acoplamientos y ser resilientes a cambios de a APIs del sistema.
- **TelemetryService**: Tiene que ser un servicio de Android que corre en segundo plano. Accede a los datos de telemetría almacenados en el TelemetryManager, los guarda en la base datos y los envía a las capas superiores (View Models).

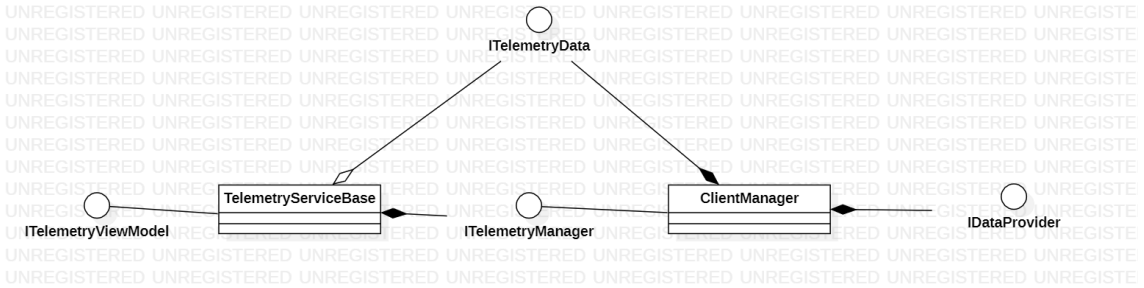


Figura 35 Clases base e interfaces para implementar por los distintos módulos de telemetría

Las clases base, obligan a implementar o instanciar el resto de las componentes de la telemetría, por lo tanto, se minimiza el riesgo de error y se mantiene toda la implementación homogénea.

Para añadir un nuevo componente de telemetría e integrarlo con el sistema bastaría con:

- Crear un nuevo servicio derivado de TelemetrySeriviceBase
- Crear un mánager derivado de ClientManager.
- Crear un data provider que implemente IDataProvider.

- Crear una nueva clase contenedora de los datos de telemetría que implemente ITelemetryData.
- Crear las nuevas entidades en Room con su repositorio asociado.

5.4.2 Flujo de la información

Todos los subsistemas de telemetría (presentes y futuros), van a tener un el mismo flujo de mensajes base, luego cada servicio puede tener sus particularidades, pero la base va a ser siempre la misma:

Inicio y fin de la recolección de telemetría:

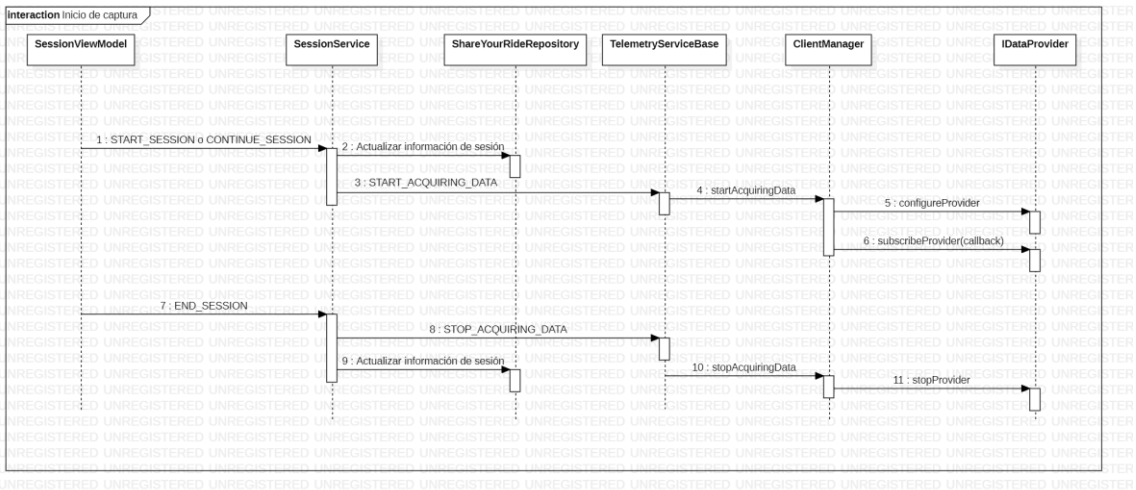


Figura 36 Flujo de mensajes de inicio y fin de captura de telemetría

Guardar telemetría en la BBDD:

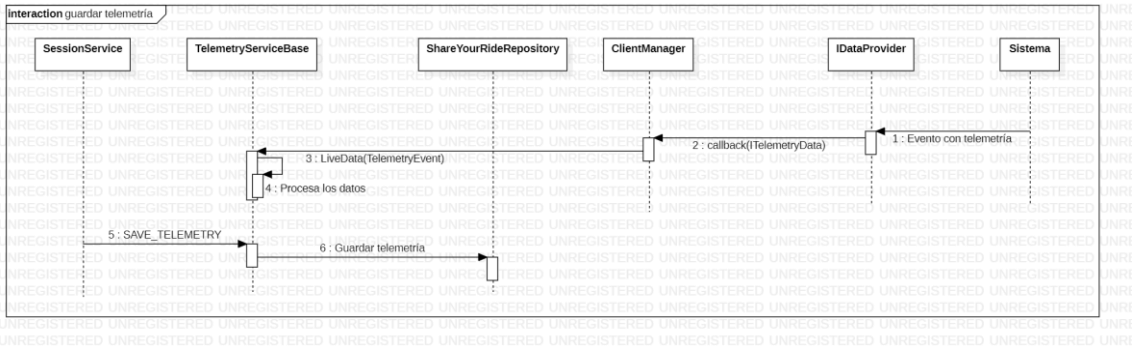


Figura 37 Flujo de mensajes para almacenar la telemetría

La telemetría se sincroniza mediante el mensaje SAVE_TELEMETRY el cual contiene un timeStamp que sirve para asociar toda la telemetría a un instante en el vídeo.

Actualizar la telemetría de la vista:

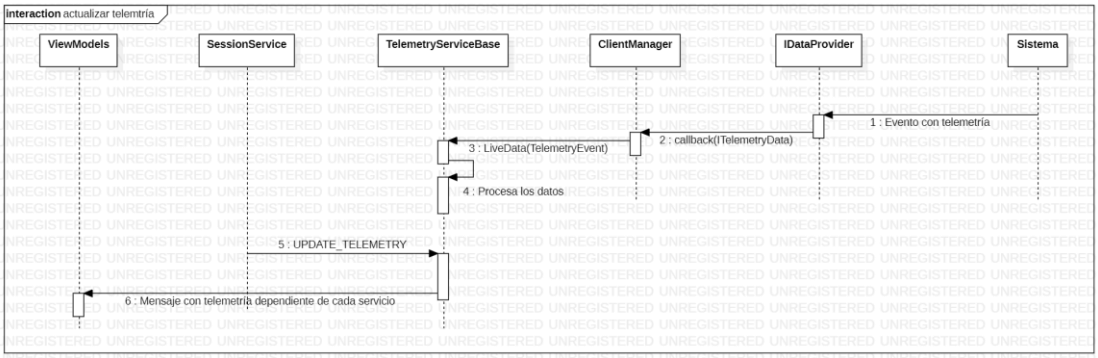


Figura 38 Flujo de mensajes para actualizar en las vistas la telemetría

Para regular la cantidad de datos que se envía a las vistas, se utiliza el mensaje UPDATE_TELEMETRY.

5.4.3 Localización

El acceso a los sensores de localización de la API Android está explicado en [15].

La localización requiere de habilitar una serie de permisos en la APP:

- ACESS_COARSE_LOCATION: Acceso a localización aproximada
- ACESS_FINE_LOCATION: Acceso a localización aproximada
- ACESS_BACKGROUND_LOCATION: Acceso a la localización cuando el servicio esté en segundo plano.

A partir de Android 10, es necesario pedir que el usuario acepte los permisos. Por lo tanto, si el usuario no da permisos de localización a la APP, no se van a poder obtener las métricas de velocidad, altitud, distancia y desnivel.

La obtención de los valores de localización es directa y solo los valores de distancia y de inclinación van a requerir de procesamiento extra:

- **Distancia**: Es un valor acumulativo. Por cada posición GPS obtenida, se tiene que obtener la distancia con el punto anterior en el eje horizontal y el diferencial de altura. Luego usando el teorema Pitágoras, se obtiene el valor de la hipotenusa, es decir la distancia real.

- **Inclinación del terreno**: El mismo método anterior, con la salvedad que, en vez de la longitud de la hipotenusa, obtenemos el dividendo del eje Y respecto al X y lo convertimos en porcentaje.

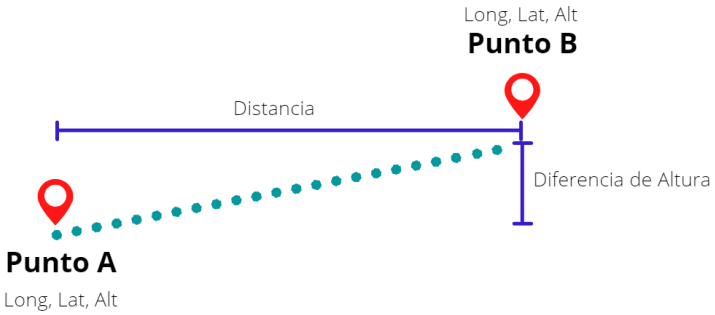
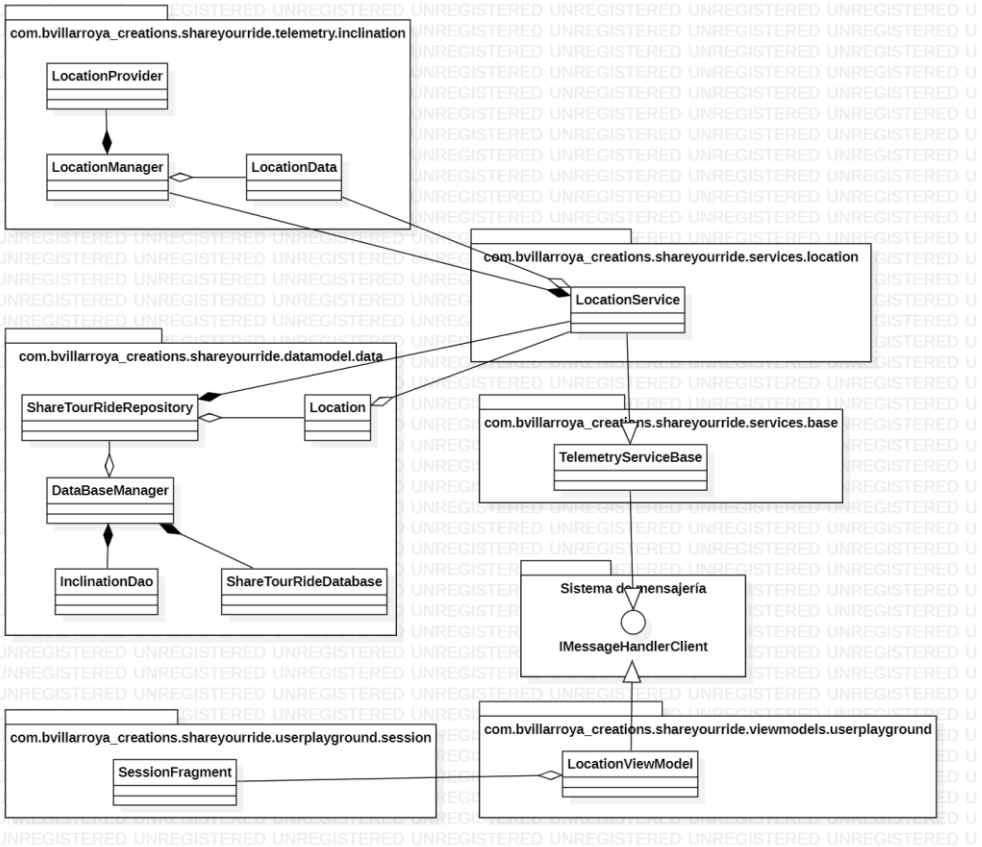


Figura 39 Calculo de distancia e inclinación

Otro de los puntos adicionales que requiere el procesamiento de la telemetría de los sensores de localización, es determinar si los datos recibidos se pueden dar por correctos. La precisión de los valores GPS viene indicada en el evento de localización dado por el sistema [27]. Siguiendo las recomendaciones de Google, no se va a marcar que la actividad está lista para empezar hasta que se reciban posiciones de GPS con una precisión de 68 o más. Del mismo modo, todos los eventos de localización recibidos con una precisión menor a 68 son descartados [3], lo mismo con eventos con códigos de tiempo anteriores a los ya procesados.

La estructura de las clases obedece a la arquitectura de software explicada en 5.4.1 Capas y distribución del código.



[3] Como los programas de simulación de posiciones no envían este dato, la restricción de precisión se deja comentada.

Figura 40 Diagrama de clases del sistema de localización

5.4.4 Giroscopios y acelerómetros

En este apartado, al igual que en el apartado de la localización no voy a entrar en explicar la obtención de los valores aportados por el móvil. La implementación del acceso y muestreo de los sensores están perfectamente explicadas en [13].

En el siguiente diagrama muestro la relación de clases que intervienen en esta funcionalidad, desde el acceso al hardware, al acceso a base datos, servicio, View Model y vista. La parte explicada en el punto 5.4.1 Capas y distribución del código se omite por legibilidad.

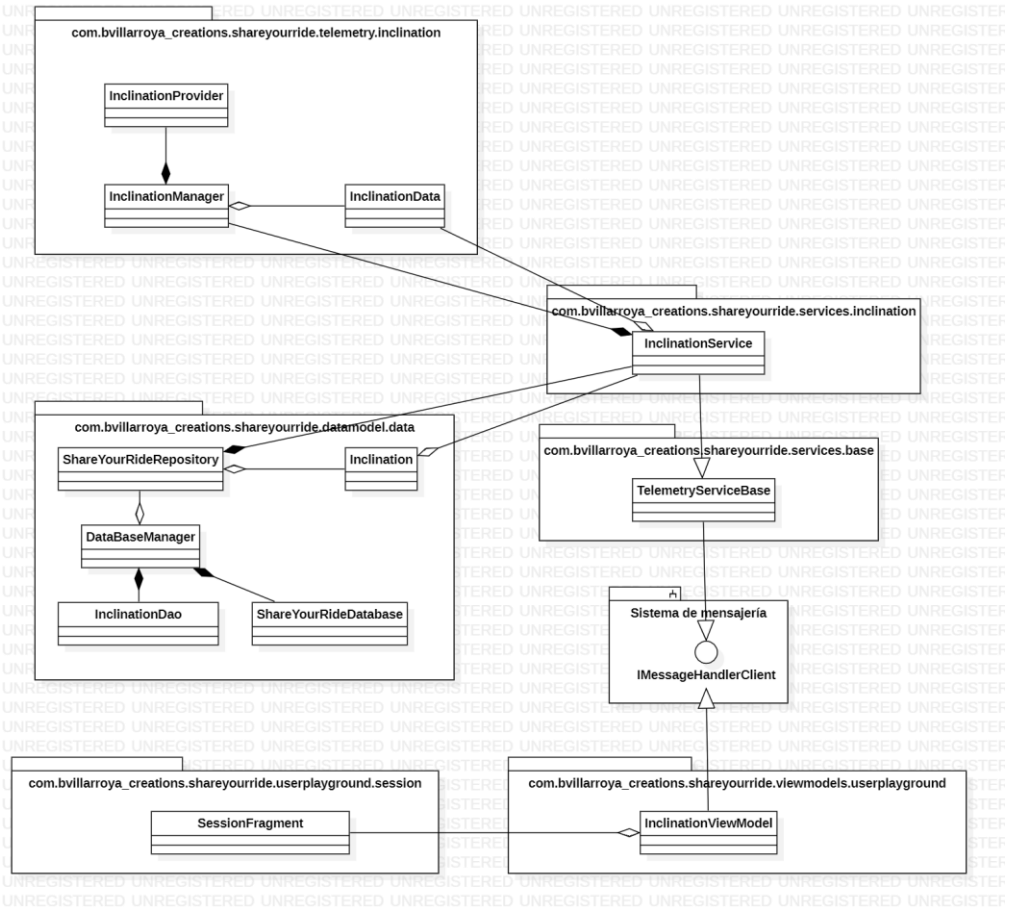


Figura 41 Diagrama de clases del sistema de inclinación

Lo primero de todo es entender como Android proporciona los datos de telemetría y que sensores vamos a utilizar.

Los sensores nos van a proporcionar los valores de rotación y aceleración en un sistema de tres ejes[4]:

[4] Para más información sobre el sistema de coordenadas en Android ver [14]

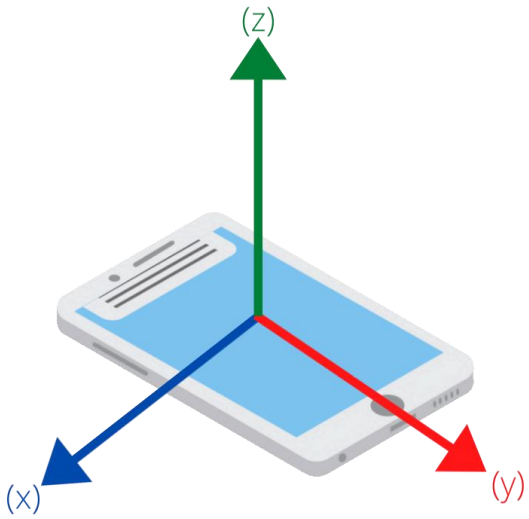


Figura 42 Sistema de coordenadas del móvil

En caso de calcular la inclinación (de una moto), utilizo el sensor denominado ***Rotation vector***. Hay que destacar que es un sensor virtual, el cual por debajo usa el hardware que tenga el móvil disponible para tales tareas, normalmente giroscopios, acelerómetros y orientación.

La API de Android proporciona el ángulo de la rotación del móvil en cada eje (en radianes). Como se ha establecido que solo se soporta el móvil en posición *portrait*, únicamente nos interesa calcular la rotación del móvil en el eje y, denominada *roll*:

Figura 43 Cálculo de inclinación

Esto quiere decir que, en esta primera versión, el móvil tiene que estar situado de forma longitudinal al largo de la moto o superficie que vaya a rotar.

Para la aceleración el cálculo es un poco más complejo. En primer lugar, utilizamos el sensor de aceleración lineal. Este sensor virtual, filtra la gravedad y la orientación de telemetría aportada.

El sensor ***lineal acceleration*** proporciona tres vectores de aceleración, uno por cada eje. Esto quiere decir que, si queremos obtener un valor escalar de representación, tenemos que usar el teorema de Pitágoras.

En condiciones ideales, para calcular las aceleraciones en una moto, solo necesitaríamos muestrear los ejes “x” e “y”, pero como el móvil puede estar colocado en cierto ángulo, también vamos a tener que usar el eje “z”:

Sin embargo, para calcular la dirección el cálculo es un poco más complejo, el móvil puede estar situado tanto plano en la superficie como en un ángulo cercano a los 90 grados. Como hemos limitado el uso del móvil a retrato y asumiendo que está colocado de forma longitudinal al plano en movimiento tenemos los siguientes escenarios:

- El eje x presenta un ángulo de rotación menor a 45º:

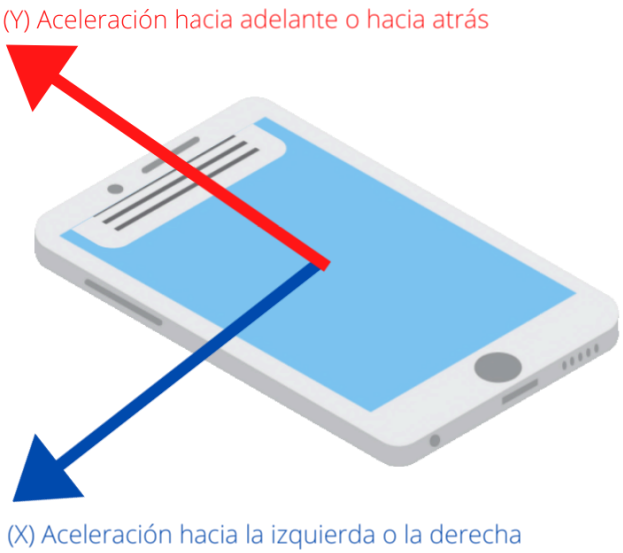


Figura 44 Dirección de la aceleración con el móvil tumbado

- El eje x presenta un ángulo de rotación mayor a 45º:

Figura 45 Dirección de la aceleración con el móvil inclinado

Como se puede deducir de las imágenes, la dirección de la aceleración solo se expresa en un plano de dos dimensiones, ya que la telemetría que nos interesa representar son las aceleraciones laterales y longitudinales.

5.4.4.1 Calibración

Uno de los puntos claves del manejo de estos dos tipos de sensores es realizar una calibración antes de empezar a procesar la información. Con calibración me refiero a la acción de definir unos valores de referencia (dos vectores tridimensionales) que me sirvan para poder:

- **Rotación**: Definir el punto de origen sobre el que calcular la rotación del móvil. Es decir, si el usuario coloca el móvil con una inclinación de 5º en el eje “y”, esos 5º van a ser un desfase que voy a aplicar a todas las mediciones, para obtener la inclinación real.

- **Aceleración**: El vector de referencia va a ser usado para crear un filtro de paso alto. En situación de reposo los sensores que incluya el móvil van a proporcionar lecturas y variaciones, estas medidas las vamos a caracterizar como “ruido” y lo vamos a filtrar. Todas las medias reportadas por el móvil por debajo o igual al vector de referencia, son descartadas (se procesan como 0).

El proceso de calibración incluye la participación del usuario, donde la APP le pide que sitúe el móvil en el soporte donde se va a realizar la actividad (2.4 Calibración de los sensores). Internamente el cálculo conlleva los siguientes pasos:

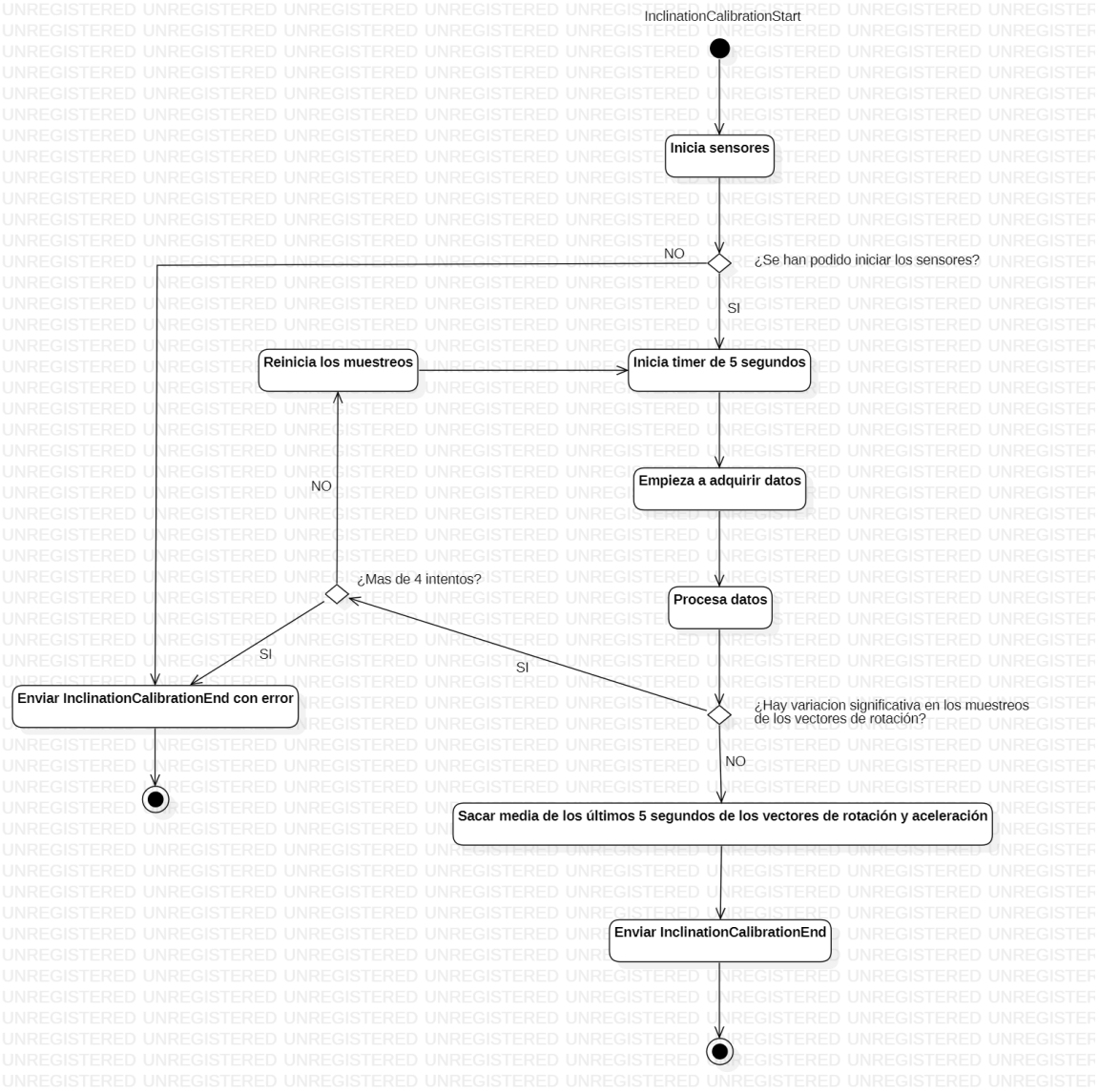


Figura 46 Diagrama de actividad de la calibración

La secuencia de mensajes en el proceso de calibración es el siguiente:

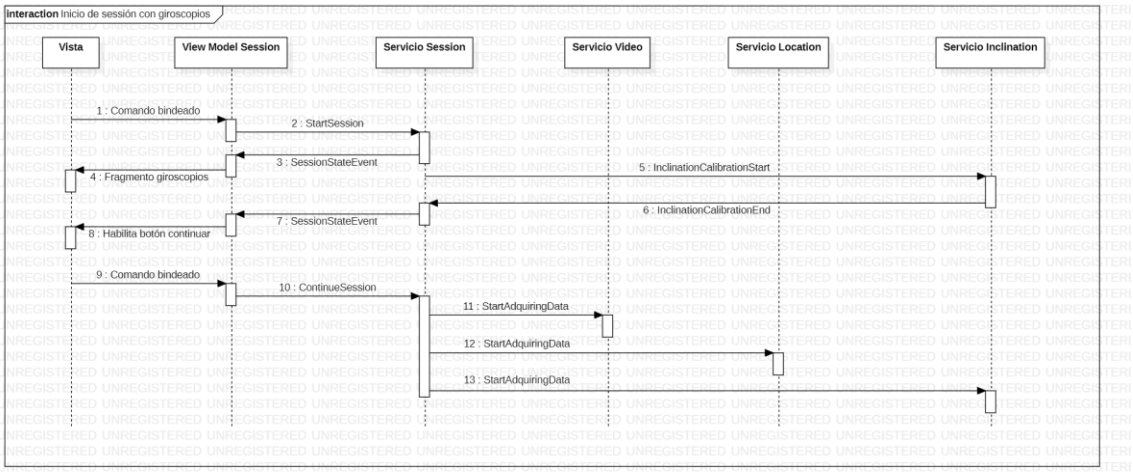


Figura 47 Diagrama de secuencia de la calibración

5.4.4.1 Muestreo

En el ámbito de la recolección de datos tenemos las siguientes magnitudes:

- Roll: Rotación en el eje y, en grados
- Pich: Rotación en el eje x, en grados
- Azimuth: Rotación en el eje z, en grados
- Array de tres posiciones con los valores de aceleración, uno por eje.
- Array de tres posiciones con los valores de la gravedad, uno por eje. No usado actualmente, pero dejamos preparado el sistema para su futuro uso

Como la precisión y la sensibilidad de los sensores es considerablemente alta, al representar los valores, en ciertas situaciones límite (por ejemplo, cuando la inclinación está entre dos grados), se producen muchas variaciones, que, a la hora de representar los valores por pantalla, no producen un efecto agradable a la vista. Por eso hay que añadir cierta histéresis.

Para conseguir eso, en el *provider* se crean una serie de buffers que van almacenando los valores discretos de cada sensor hasta que salta el temporizador de envío a las capas superiores. En este punto se calcula la media y se envía. Esto va a conseguir que en situaciones "estacionarias", como puede ser una inclinación sostenida en una curva, al ofrecer la media de un muestreo, el valor mostrado sea más estable y no baile.

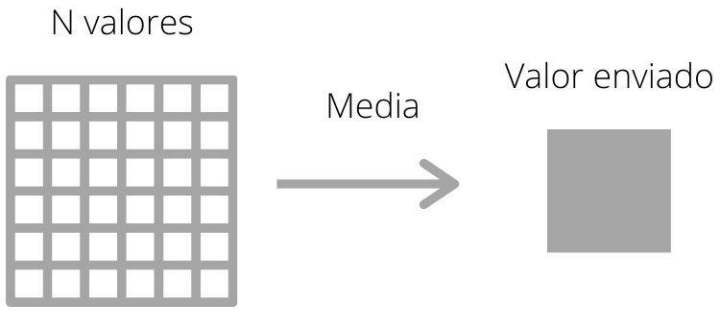


Figura 48 Conversión de valores discretos a media

5.5 Estados de la sesión

La siguiente imagen muestra el diagrama de estados de la sesión, que a su vez es el que va a determinar el comportamiento y las ventanas mostradas en la APP:

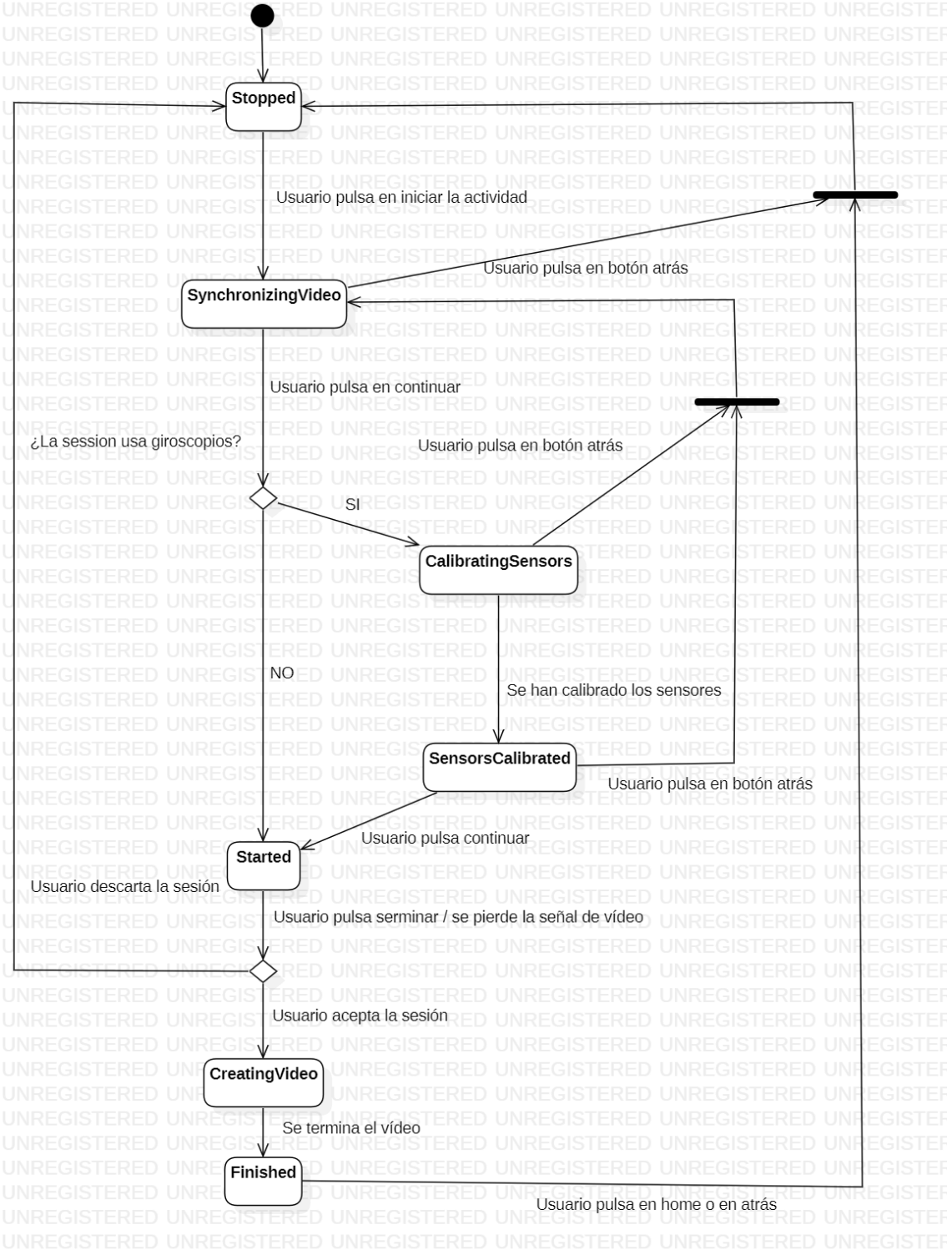


Figura 49 Diagrama de estados de la sesión

Cada estado tiene asociada una vista distinta, y se usan estos estados para determinar qué fragmento debe mostrarse, qué acciones realiza la pulsación hacia atrás
Cada cambio de estado se almacena en la base de datos, junto con información adicional (si la hay).

Estado	Descripción e interacción con el usuario
Stopped	Sesión no iniciada. Se muestra la ventana principal. El usuario puede configurar la aplicación e iniciar una nueva sesión.
SynchronizingVideo	Muestra la ventana de sincronización de vídeo. El usuario puede configurar el retardo del vídeo, cancelar la sesión y continuar con la sesión.
CalibratingSensors	Se está realizando la calibración de sensores. El usuario debe dejar el móvil en reposo en el lugar donde va a estar durante la sesión. El usuario puede volver a atrás a sincronizar el vídeo.
SensorsCalibrated	Se ha realizado la calibración de sensores. Habilita el botón para continuar la sesión. El usuario puede continuar con la sesión, reiniciar la calibración
Started	La actividad está en curso. Se muestra en tiempo real el estado de la sesión. El usuario puede finalizar o descartar la sesión.
CreatingVideo	La sesión de usuario ha terminado, se muestra un resumen de la actividad y se compone el vídeo. Durante este proceso, se bloquea la navegación de la APP.
Finished	El vídeo se ha terminado de crear. Se sigue mostrando un resumen de la sesión. En este punto, el usuario es libre de navegar.

Tabla 12 Descripción de estados de la sesión

Estado	Fragmento asociado
Stopped	HomeFragment
SynchronizingVideo	VideoSyncFragment
CalibratingSensors	GyroscopeCalibrationFragment
SensorsCalibrated	GyroscopeCalibrationFragment
Started	SessionFragment
CreatingVideo	SessionFinishedFragment
Finished	SessionFinishedFragment

Tabla 13 Relación entre estados de sesión y Fragmentos

Estado	Navegación hacia atrás
Stopped	Manejada por el sistema
SynchronizingVideo	Navega a HomeFragment
CalibratingSensors	Navega a VideoSyncFragment
SensorsCalibrated	Navega a VideoSyncFragment
Started	Muestra mensaje para finalizar o descartar la sesión
CreatingVideo	Muestra mensaje advirtiendo que se está creando un vídeo.
Finished	Navega a la ventana de home

Tabla 14 Relación entre estados de sesión y navegación hacia atrás

Toda navegación hacia atrás incluye un cambio en el estado de la sesión.

5.6 Notificaciones, restauración y ciclo de vida de la APP

Una parte importante de la interacción con los usuarios es el sistema de las notificaciones. Además, hay que tener en cuenta que el usuario puede abrir otras aplicaciones a la vez que utiliza la nuestra, atender llamadas, bloquear el teléfono, etc.

Google exige que a los usuarios se les notifique que hay servicios en segundo plano, además este sistema le permite al usuario conocer el estado de la App sin necesidad de acceder a ella.

La implementación de las notificaciones se encuentra en la clase ServiceNotificationBuilder. La forma de cómo utilizar las notificaciones esta explicada en [25]. Lo único destacable de este punto es que he utilizado para todas las notificaciones el mismo ID. De esta forma, aunque existan cuatro servicios mandando notificaciones y además se actualice el estado de la sesión en vivo, el usuario solo ve una única notificación en el sistema.

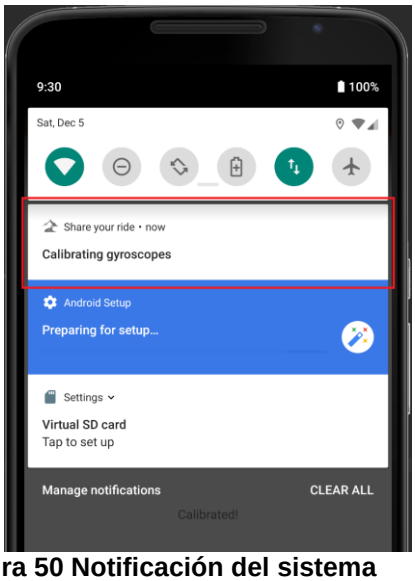


Figura 50 Notificación del sistema

El sistema de notificaciones de Android permite más funciones, pero se deja la notificación de la forma más simple.

Al clicar en la notificación, el sistema abre la aplicación en la MainActivity (que es la que se indica en el *builder* de la notificación). Esto supone que en la MainActivity tenemos que controlar que fragmento debe mostrase al restaurar la actividad.

Otro punto a tener en cuenta es que, a diferencia de otros sistemas, en Android los cambios de orientación, ciertas acciones de navegación, incluso poner en segundo plano la aplicación, puede terminar con la destrucción de las actividades o los View Models. Este ciclo está explicado en [27], por lo tanto voy a obviar su explicación y me voy a centrar en las consideraciones que he tenido en cuenta:

Al usar el patrón MVVM y utilizar View Models para comunicar los fragmentos con las actividades, el ciclo de vida de las actividades Android presenta una serie de implicaciones:

- Cuando un View Model es destruido (función onCleared), se debe des registrar del sistema de mensajería y liberar los recursos que estén siendo observados (LiveData) para evitar memory leaks y tener varias instancias de un mismo View Model funcionando.

- Todos los observers configurados en los View Model, Actividades y Fragmentos, deben estar supeditados al objeto LifeCycle de la actividad o fragmento.

- Los fragmentos no crean View Models, acceden al View Model creado por la actividad padre. Esto es esencial para la comunicación entre los distintos componentes de la vista.

- Una forma fácil de verificar que solo se está usando un View Model en concreto, es asignar a cada instancia un UUID único y *logearlo* en las acciones del View Model.

```
/**
 * Unique UUID to identify each instance of this view model
 */
private val guid: Int = UUID.randomUUID().toString().hashCode()
```

```
try
{
    Log.i( tag: "SessionViewModel",  msg: "SYR -> $guid -> Sending START_SESSION message")
    val message = MessageBundle(MessageTypes.START_SESSION, data: "", MessageTopics.SESSION_COMMANDS)
    sendMessage(message)
}
```

Figura 51 Uso de UUIDs para identificar instancias de View Models

- Android presenta un sistema para recuperar el estado de las actividades, por lo tanto, el estado de sus View Models.
  No voy a usar este sistema ya que la información de la sesión del usuario puede cambiar mientras la Actividad esta destruida. Por lo tanto, al restaurar la Actividad, se va a volver a obtener la información de la sesión de la base de datos Room o de los servicios.

- No se puede programar un viewmodel o una actividad para almacenar datos persistentes. Del mismo modo, no podemos dar por hecho que se ha recibido el último evento de estado enviado por las capas de abajo, ya que puede haberse enviado mientras la Actividad o el View Model no estaban disponibles, por lo tanto, en el arranque de un View Model, siempre tiene que solicitar a los servicios una actualización de información.

- Quienes tienen toda la información de estado de la sesión en tiempo real son los servicios, por lo tanto, son ellos los que tienen la responsabilidad de mantener la coherencia. Por ejemplo, si se solicita cambiar la configuración de red de la cámara en uso, pero estamos en medio de una sesión, ese cambio no tiene sentido y tiene que ser descartado. Estas situaciones se pueden dar al iniciar una Actividad.

- Notificaciones periódicas de estado de los servicios a los View Models. Por el mismo motivo anterior, los servicios deben mantener informados los View Models del estado de la sesión, aunque los View Models no soliciten un refresco de la información.

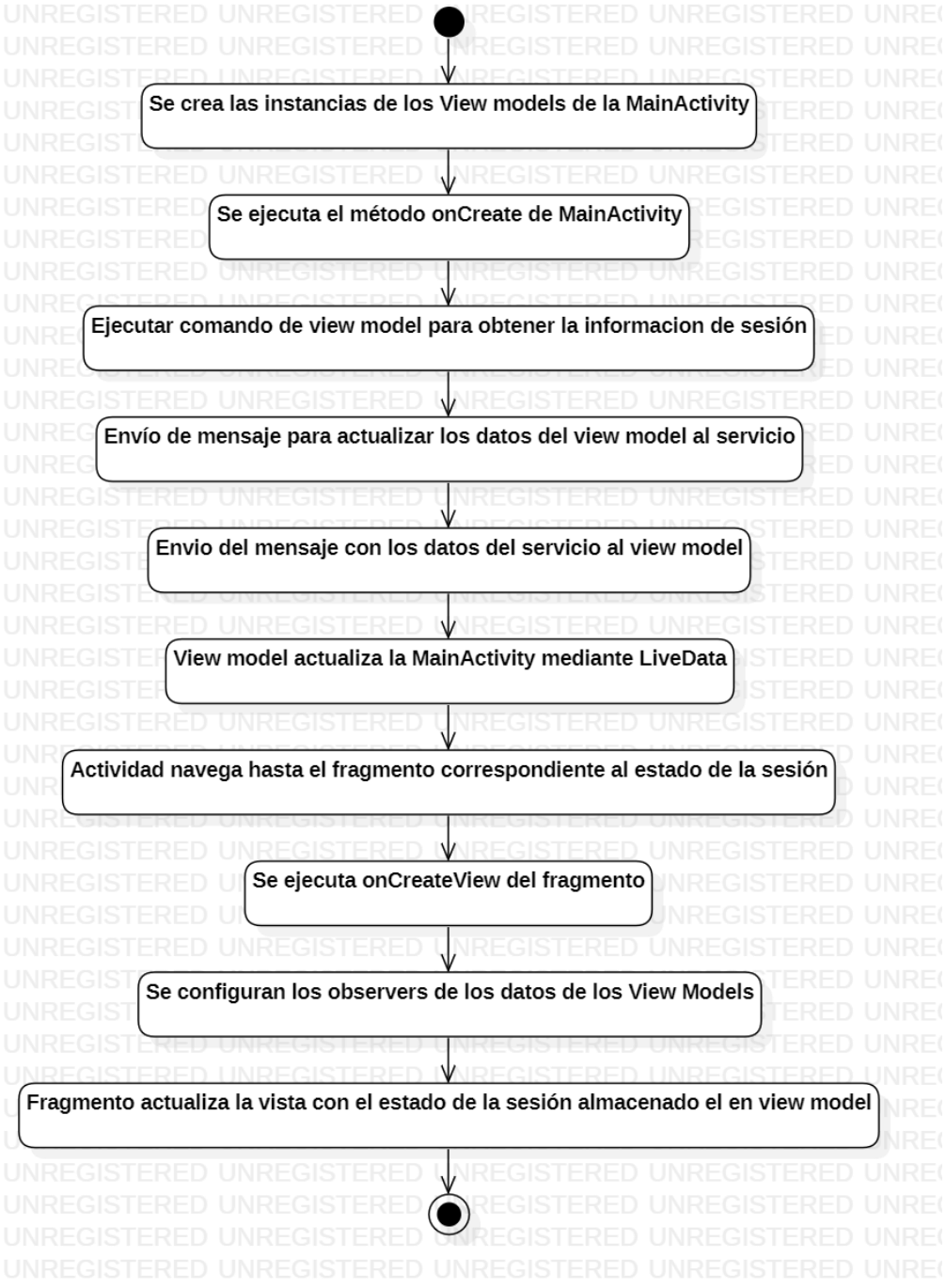


Figura 52 Flujo de acciones al restaurar la actividad principal

5.6 Vídeo

Toda la gestión del vídeo se realiza desde el servicio de vídeo. El cual como el resto de los servicios se lanza en background y es persistente. Ya que, aunque la APP pase a segundo plano, el servicio continuará accediendo al vídeo emitido por la cámara.

Este es el punto más complicado del proyecto y el que más incógnitas me ha dado, y el que mayor riesgo ha supuesto en la planificación y durante el desarrollo.

El primer paso ha sido definir unos requisitos mínimos:

- Acceder a un Stream RTSP
- Poder identificar los fotogramas.
- Guardar el stream de video en un fichero local
- Poder superponer imágenes en un fotograma en concreto
- Poder componer un vídeo con dichos fotogramas.
- La reproducción de vídeo en sí, aunque útil, es secundaría.
- La captura del audio por ahora no se contempla.

Con los requisitos claros, se obtiene un listado de librerías candidatas y se realizan pruebas de integración con algunas de ellas:

- **MediaPlayer** [16]: Player estándar de Google, permite acceder al stream de vídeo. Sin embargo, la forma de procesar el vídeo para el fin de la APP no es directo, requiere de un componente gráfico para tal acción (un SurfaceView o un VideoView), para extraer los frames, con lo cual esto limita usarlo como servicio y es complicado saber si esto puede suponer un impacto en el rendimiento. En las pruebas de integración arroja un retraso de entre 6 y 10 segundos en el vídeo, algo totalmente inaceptable. Además, no permite edición de video.

- **ExoPlayer** [17]: Player avanzado de código abierto desarrollado por Google. Carece de integración RTSP y de edición de vídeo

- **ExoPlayer Branch 3854**[18]: Se trata de un *fork* desarrollado por la comunidad con soporte para RTSP. Mismas contras que el MediaPlayer.

- **FFMPEG:** Las librerías de edición de vídeo por excelencia. Se puede usar mediante el NDK y JNI o librerías ya compiladas. Tras varios intentos de compilar una versión de FFMPEG y comunicarnos con ella mediante JNI se descart, no iba a ser posible cumplir los plazos con este método.
  Las versiones ya compiladas y listas para su uso presentan un problema y es que la interacción con la API ofrecida al desarrollador es el envío de instrucciones como de si una línea de comandos fuese, esto hace muy farragoso su uso en el escenario propuesto,

además de limitar ciertas acciones que no podrían ser hechas del modo que tengo pensando.

- **OpenCV** [19]: Se trata de una librería de visión por ordenador que permite la edición y procesamiento de vídeo e imágenes. También tiene un componente de *machine learning*. Es de Licencia Apache, en su versión 4.5 y superior.
  Como desventajas, su documentación oficial parece un poco anticuada (uso en Eclipse) y pocos ejemplos. Permite acceso a stream de video RTSP sin embargo al intentar integrar la librería en el proyecto y no conseguir conexión con la cámara, se descarta su uso. Por lo que se entiende en las discusiones de desarrolladores, solo admite Streams de video mjpeg dentro de contenedores AVI. No puede ser usada en mi entorno de desarrollo.

- **JavaCV** [20]: Se trata de un wrapper Java de librerías de vídeo OpenCV y FFMPEG (entre otras muchas). Es un proyecto de código abierto el cual no consta del soporte de otros proyectos que se han tenido en consideración antes. Sin embargo, parece que cumple con todos los requisitos.

De todas las opciones candidatas se realizan pruebas de integración con el proyecto o se estudia su viabilidad. Al final se termina por elegir JavaCV, ya que parece ser la que mejor se adapta a los requisitos. Es una opción de riesgo ya que, a la hora de realizar el desarrollo, aún no se conoce como interactuar con dicha librería. Durante las pruebas se estima que poder integrar, compilar y usar la librería ha costado unas 30 horas, buena parte del presupuesto de la parte del vídeo.

5.6.1 Integración del vídeo

JavaCV se puede integrar en una aplicación de varias formas:
- Manual importando los Jars
- Por Maven
- Por Gradle.

Como uso Gradle para la compilación y definición del proyecto, opto por este acercamiento. Cabe destacar que la documentación oficial en este supuesto no es exacta, no está actualizada. Por lo tanto, ha sido un ejercicio de ensayo y error además de bucear mucho en el Github del proyecto.

Entre los fallos encontrados tenemos duplicidad de archivos json, que hacen que falle la compilación y errores del *linkado* dinámico de librerías en tiempo de ejecución.

Todos estos fallos los encontramos si intentamos realizar la instalación manual, además de otros más graves, como duplicidad de ficheros .so y de java.

La forma de solucionarlos ha sido importar los siguientes paquetes y además configurar una serie de ficheros JSON, para que solo sea cogido el primero de ellos detectado y así evitar la excepción de múltiples ficheros con el mismo nombre en una misma ruta:

```
//Gives us access to the video processing library JavaCv, with ffmpeg and openCV functionalities
implementation 'org.bytedeco:javacv:1.5.4'
implementation group: 'org.bytedeco', name: 'javacpp', version: '1.5.4'
implementation group: 'org.bytedeco', name: 'javacpp', version: '1.5.4', classifier: 'android-arm64'
implementation group: 'org.bytedeco', name: 'javacpp', version: '1.5.4', classifier: 'android-arm'
implementation group: 'org.bytedeco', name: 'javacpp', version: '1.5.4', classifier: 'android-x86_64'
implementation group: 'org.bytedeco', name: 'javacpp', version: '1.5.4', classifier: 'android-x86'
implementation group: 'org.bytedeco', name: 'ffmpeg', version: '4.3.1-1.5.4'
implementation group: 'org.bytedeco', name: 'ffmpeg', version: '4.3.1-1.5.4', classifier: 'android-arm64'
implementation group: 'org.bytedeco', name: 'ffmpeg', version: '4.3.1-1.5.4', classifier: 'android-arm'
implementation group: 'org.bytedeco', name: 'ffmpeg', version: '4.3.1-1.5.4', classifier: 'android-x86_64'
implementation group: 'org.bytedeco', name: 'ffmpeg', version: '4.3.1-1.5.4', classifier: 'android-x86'
implementation group: 'org.bytedeco', name: 'opencv', version: '4.4.0-1.5.4'
implementation group: 'org.bytedeco', name: 'opencv', version: '4.4.0-1.5.4', classifier: 'android-arm64'
implementation group: 'org.bytedeco', name: 'opencv', version: '4.4.0-1.5.4', classifier: 'android-arm'
implementation group: 'org.bytedeco', name: 'opencv', version: '4.4.0-1.5.4', classifier: 'android-x86_64'
implementation group: 'org.bytedeco', name: 'opencv', version: '4.4.0-1.5.4', classifier: 'android-x86'
```

Figura 53 Paquetes de la librería JavaCV

```
//to avoid More than one file was found with OS independent path
packagingOptions {
    pickFirst 'META-INF/native-image/android-arm/jnijavacpp/jni-config.json'
    pickFirst 'META-INF/native-image/android-arm64/jnijavacpp/jni-config.json'
    pickFirst 'META-INF/native-image/android-x86_64/jnijavacpp/jni-config.json'
    pickFirst 'META-INF/native-image/android-x86/jnijavacpp/jni-config.json'

    pickFirst 'META-INF/native-image/android-arm/jnijavacpp/reflect-config.json'
    pickFirst 'META-INF/native-image/android-arm64/jnijavacpp/reflect-config.json'
    pickFirst 'META-INF/native-image/android-x86/jnijavacpp/reflect-config.json'
    pickFirst 'META-INF/native-image/android-x86_64/jnijavacpp/reflect-config.json'
}
```

Figura 54 Archivos duplicados en JavaCV

Como se puede observar se añaden librerías para soportar arquitecturas arm, arm64, x86 y x86-64. Y se compone de las siguientes partes:

- **JavaCV**: Se trata de la librería en sí, contiene las definiciones de la API del core.
- **Javacpp**: Se requiere para el "*linkado"* dinámico en tiempo de ejecución, entre otras cosas.
- **Ffmpeg**: Se usa para la conexión, codificación y almacenamiento del stream de vídeo que viene de la cámara.

5.6.1 Procesamiento de imágenes

El tratamiento del vídeo consta de dos fases:
1- Recepción y almacenamiento del stream de vídeo de la cámara.
2- Generación de vídeo con telemetría.

Se divide en dos partes y no se hace todo de una vez pensando en los dispositivos que tengan menor potencia de procesamiento.

5.6.2.1 Recepción y almacenamiento de vídeo

La gestión de la conexión con la cámara es directa y muy simple, por lo tanto, no se requiere un extenso diagrama de estados como podría ser con el MediaPlayer de Android. Simplemente, una vez se tiene configurada la dirección RTSP a la que conectarse, con FFmpegFrameGrabber se ejecuta el comando de start. Este se conecta de forma bloqueante. Si devuelve el hilo de la acción sin una excepción, es que hay conexión. Si no, no ha habido y hay que reintentarlo.

El intento de conexión tiene un timeout de 5 segundos configurado, si esta falla, se volverá a intentar, pasados 10 segundos. Esto se hace indistintamente del estado de la conexión WIFI por una sencilla razón, no tiene penalización y no acoplamos distintos componentes. Si no hay WIFI disponible, la conexión saltará por timeout o porque se reciben fotogramas nulos.

Cabe destacar que, con cada intento, hay que destruir el objeto Grabber y usar otro, ya que las pruebas realizadas indican que no se liberan bien ciertos recursos y tras una conexión, la API no funciona correctamente y pueden fallar los codecs o la gestión de la conexión.

El proceso de recepción y almacenamiento del vídeo se explica en el siguiente diagrama de actividad:

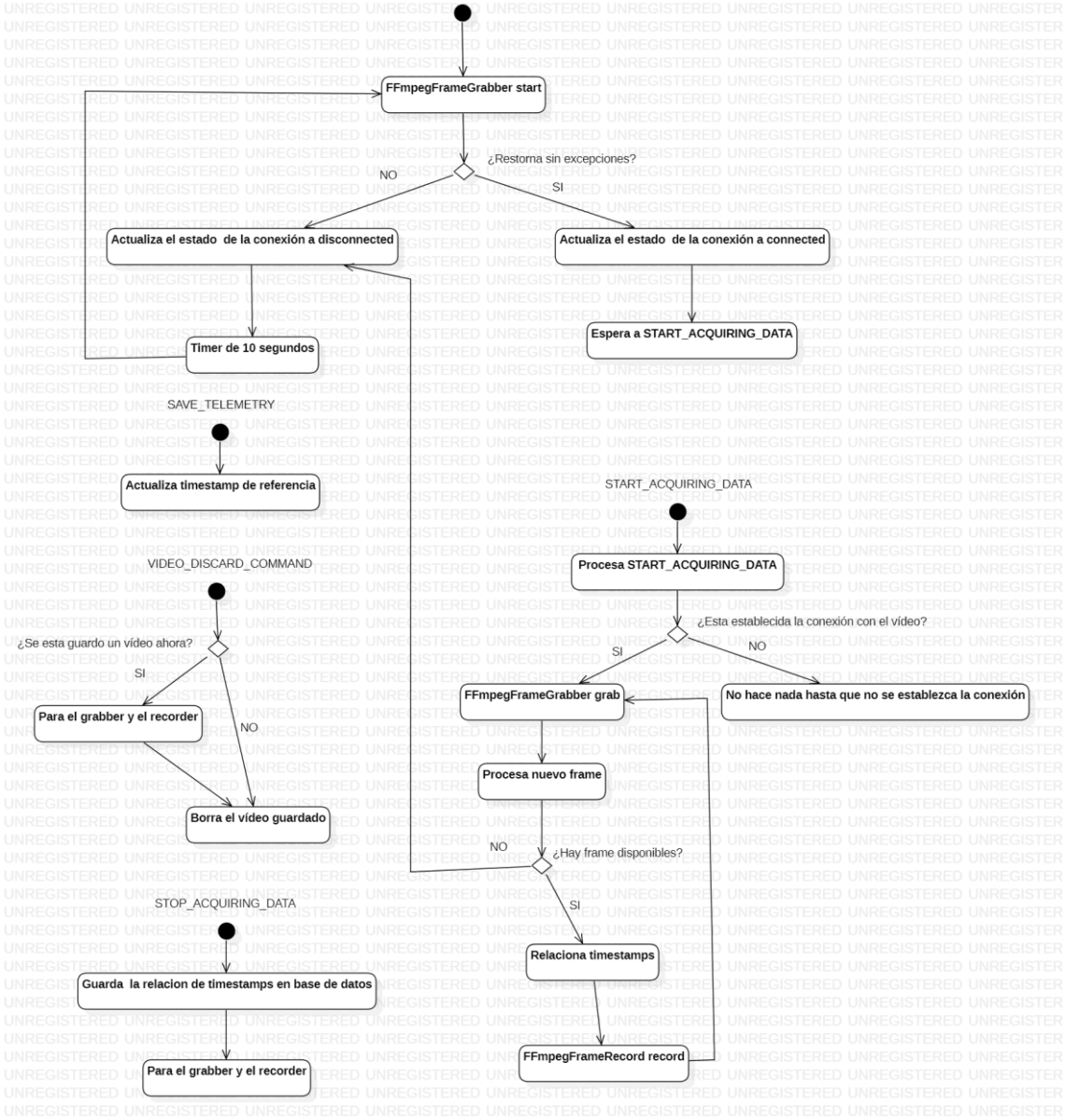


Figura 55 Diagrama actividad de la gestión del vídeo

Cuando se realiza la conexión con el vídeo, se guarda una entrada en la base de datos con la información de este (nombre, resolución, FPS,codec, etc) asociada a la sesión.

Se usa la clase FFmpegFrameGrabber de la librería JavaCV para conectarnos a un stream remoto. En este caso a un stream RTSP mediante TCP (para evitar la pérdida de paquetes). Durante el desarrollo se realizan múltiples pruebas con UDP, pero la perdida de paquetes es demasiado grande, afectando seriamente al resultado final del vídeo.

Configuramos el cliente RTSP con las opciones fflags nobuffer, avioflags direct, sync ext, probesize 32, preset superfast, tune zerolatency. Todas ellas sugeridas por FAQ de FFMPEG, para mejorar la latencia del vídeo. Para ser sincero, ninguna de ellas parece mejorar el delay.

El vídeo captado en el *stream* de vídeo se guarda en un archivo mp4, con el *codec* H264 y con el formato *Matroska*. Esta es la combinación que mejores resultados ha dado en las pruebas.

El *framerate*, el tamaño y el *bitrate* vienen dados por el *stream* de entrada. El archivo se guarda con el nombre de la sesión en un directorio privado de la APP usando la herramienta FFmpegFrameRecorder. Este se borrará cuando se descarte la sesión o se haya compuesto el vídeo final.

He decidido este flujo de acción para procesar el vídeo entrante, porque ha sido la única manera de resolver el problema de productor consumidor que se presenta:

- Si almacenamos los frames en bruto para procesarlos mas tarde, terminamos excediendo los límites de RAM y se producen excepciones del tipo *out of memory.*
- Si convertimos los fotogramas a Bitmaps y almacenamos dichos bitmaps, tenemos un problema de rendimiento, que hace que, en mi dispositivo de pruebas, solo se puedan procesar diez fotogramas por segundo.
- Teniendo en cuenta el punto anterior, se podría poner una cola intermedia, un hilo productor y otro consumidor. Pero el resultado final tras varios minutos sería el de excepciones de *out of memory.* Tampoco se pueden poner más hilos consumidores porque no se sabe el hardware que va a correr la aplicación y más hilos no tienen por qué suponer más rendimiento.
- Si cada vez que se recibe un fotograma, se almacena en otro archivo, sin procesarlo ni codificarlo, mis pruebas arrojan que se pueden procesar al menos 25 fotogramas por segundo en una resolución de 1240x720.

Este es un procedimiento bloqueante, el cual se lanza en un hilo a parte del principal. En este caso, como el servicio de vídeo corre ya en un hilo distinto al principal, no es necesario lanzar ningún otro adicional.

También es necesario determinar y notificar el estado del vídeo, esto se realiza en otra tarea paralela que se ejecuta cada 10 segundos, que comprueba periódicamente si está establecida la conexión con el vídeo y lo notifica al resto de capas con el mensaje VIDEO_STATE_EVENT

5.6.2.2 Sincronización de vídeo

Para sincronizar los frames recibidos con la telemetría del móvil debemos tener claras las siguientes premisas:

- El vídeo recibido va a tener un retardo inherente a los buffers y a la conexión con la cámara.
- El flujo del vídeo no es constante y a veces puede sufrir retardos los cuales pueden ser o no acumulativos.

- Las fechas indicadas en el protocolo RTSP no corresponden con las del móvil.
- El timeStamp de cada fotograma lo posiciona en el tiempo respecto al inicio del vídeo.
- Sabemos cuándo procesamos el primer fotograma del vídeo y podemos asociarlo a una fecha y tiempo concretos del móvil.
- La temporización y frecuencia de la telemetría es constante y obedece al tiempo del móvil.

Con estos factores en mente se desarrolla el siguiente algoritmo, bastante sencillo pero que ha sido eficaz en las pruebas [5]:

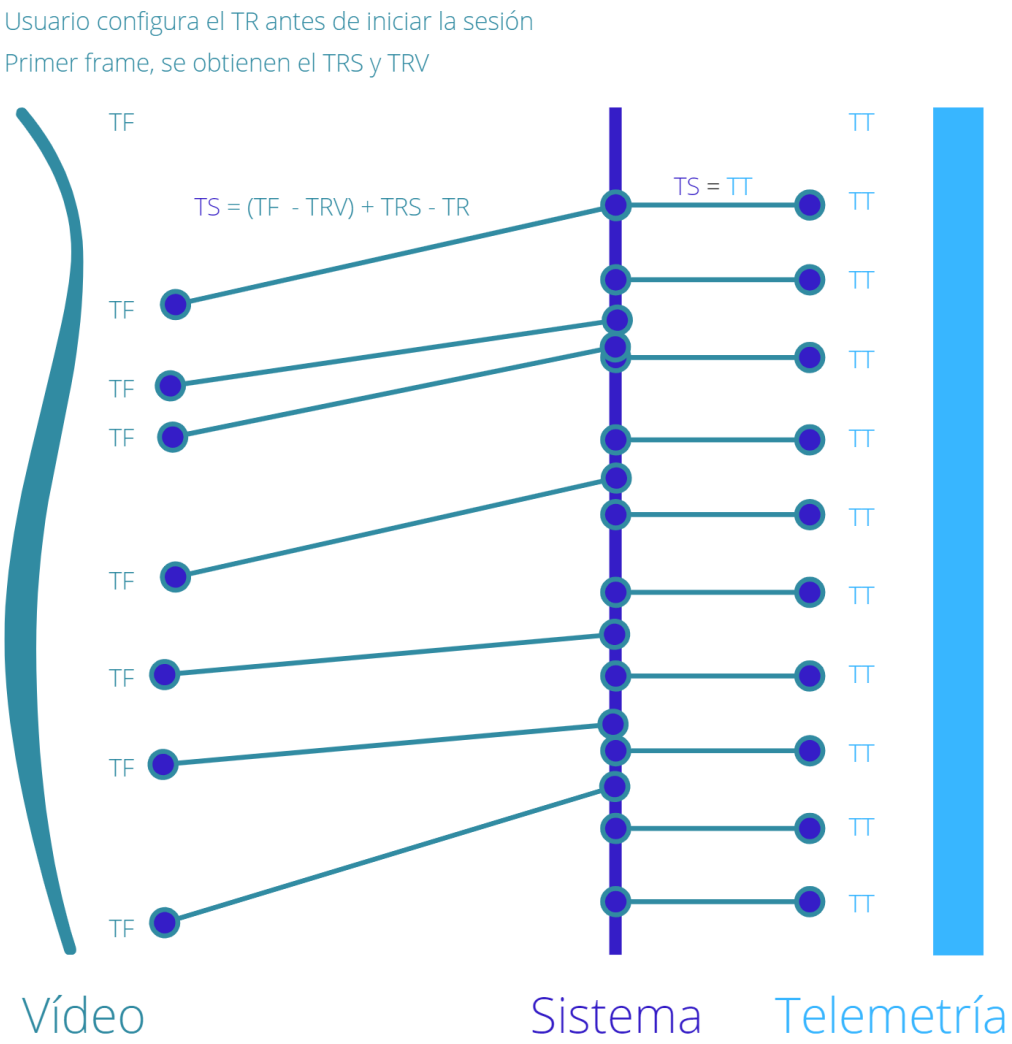


Figura 56 Sincronización de vídeo

- **TS** (Tiempo del sistema)

[5] Las pruebas se han realizado con una cámara Xiaomi Mi 4k, la cual introduce un timeStamp de 6 dígitos en cada fotograma. Los tres dígitos menos significativos se pueden descartar y nos quedamos con los más significativos que indican los segundos transcurridos de vídeo.

- **TR** (Tiempo de retardo). Es el retardo que tiene la imagen desde que es captada por la cámara hasta que se muestra en el móvil. Es configurado por el usuario antes de empezar la sesión.
- **TRV** (Tiempo de referencia del vídeo): Timestamp RTSP del primer fotograma procesado del vídeo, segundos desde el inicio del vídeo.
- **TRS** (Tiempo de referencia del sistema): TimeStamp del sistema de cuando ha sido procesado el primer fotograma del vídeo.
- **TF** (Tiempo de fotograma): Timestamp del fotograma. Segundos desde el inicio del vídeo.
- **TT**: (Tiempo de telemetría): TimeStamp de generación de la telemetría para un tiempo del sistema dado.

Si nos imaginamos la telemetría como un flujo constante de datos, sincronizados con el sistema y el vídeo como un flujo variable de datos, con su propia sincronización de tiempo.

Al procesar el primer fotograma, obtenemos la referencia del tiempo actual en el sistema y la marca temporal del fotograma dentro del vídeo. Posteriormente para cada fotograma, vamos a sumar el tiempo transcurrido respecto al inicio del vídeo, al tiempo de referencia del sistema obtenido anteriormente. A ese tiempo, se le suma el tiempo de retardo que ha configurado el usuario. Con eso obtenemos a que instante del sistema realmente pertenece la imagen del fotograma.

Como se muestra en el gráfico, el tiempo obtenido para un fotograma no tiene por qué corresponder con una medida de telemetría. Por ese motivo a la hora de componer un fotograma se va a buscar la medida de telemetría con el tiempo más cercana al fotograma.

Esta asociación entre fotogramas y el tiempo real en el sistema se guarda en base de datos por motivos de persistencia. Se guardará únicamente al finalizar la conexión de vídeo (guardado en bloque), para minimizar los accesos a la BBDD y acelerar el procesamiento de imágenes. Mientras se guarda en RAM, dentro del servicio de vídeo, en un mapa HASH.

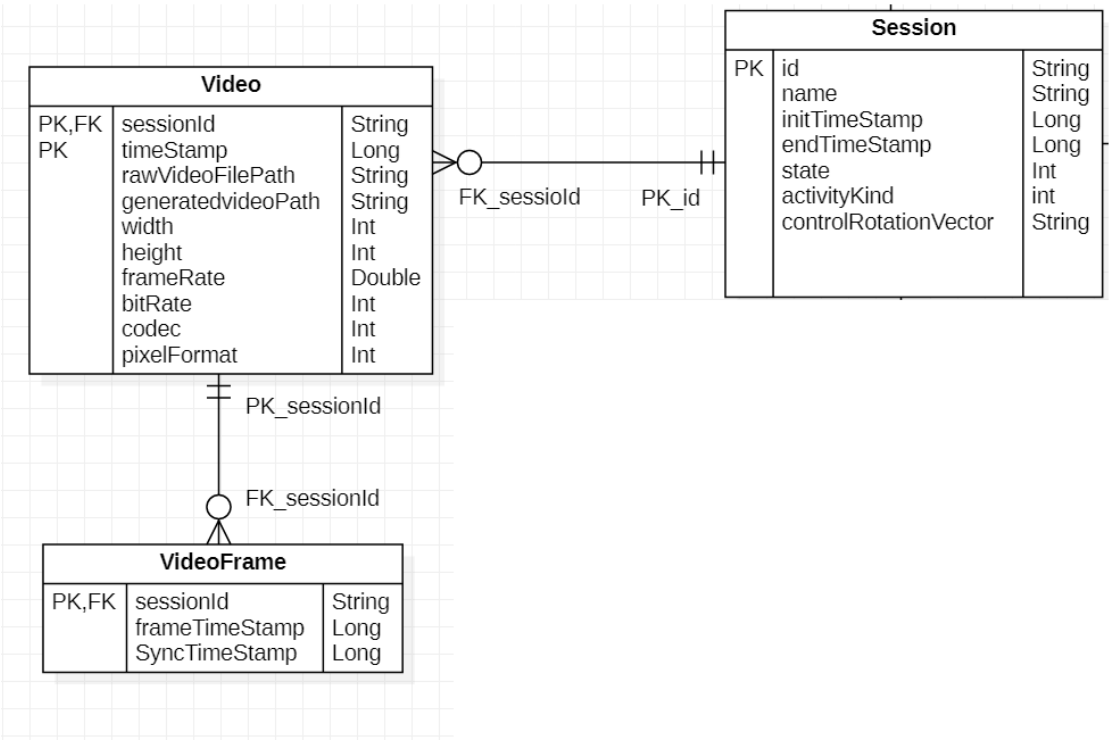


Figura 57 Relación entre los frames, el vídeo y la sesión

5.6.2.3 Composición de vídeo

La composición de vídeo se realiza en el servicio VideoCreationService y se divide en tres fases:

- **Adquisición de datos**: Primero se obtiene de la base de datos el listado completo de fotogramas de vídeo asociados a las marcas temporales que sincronizan el vídeo con la telemetría. Después con FFmpegFrameGrabber, se obtienen uno a uno los fotogramas del vídeo guardado en el apartado anterior.

- **Composición de frames**: Cada fotograma recibido se convierte en un bitmap. se busca en la base de datos su telemetría asociada y mediante Canvas, se elabora el fotograma completo con la telemetría. Finalmente, dicho fotograma se inserta en el vídeo final mediante FFmpegFrameRecorder.

  Todos los recursos de vídeo son vectoriales, pero estos hay que convertirlos a bitmaps para su inserción en un canvas. El tamaño que se les da en el proceso de conversión del vídeo se realiza mediante una simple regla de tres. Hay especificados unos tamaños fijos para la resolución de 1240x720. El diferencial de resolución que exista con el vídeo a generar será el mismo diferencial aplicado al tamaño de los recursos que se van a insertar en el vídeo y los márgenes de de estos con respecto a los bordes.

- **Publicación**: se finaliza el proceso de grabación en disco del vídeo y se notifica a las capas superiores del fin del proceso. El vídeo se guarda en el directorio público de videos.

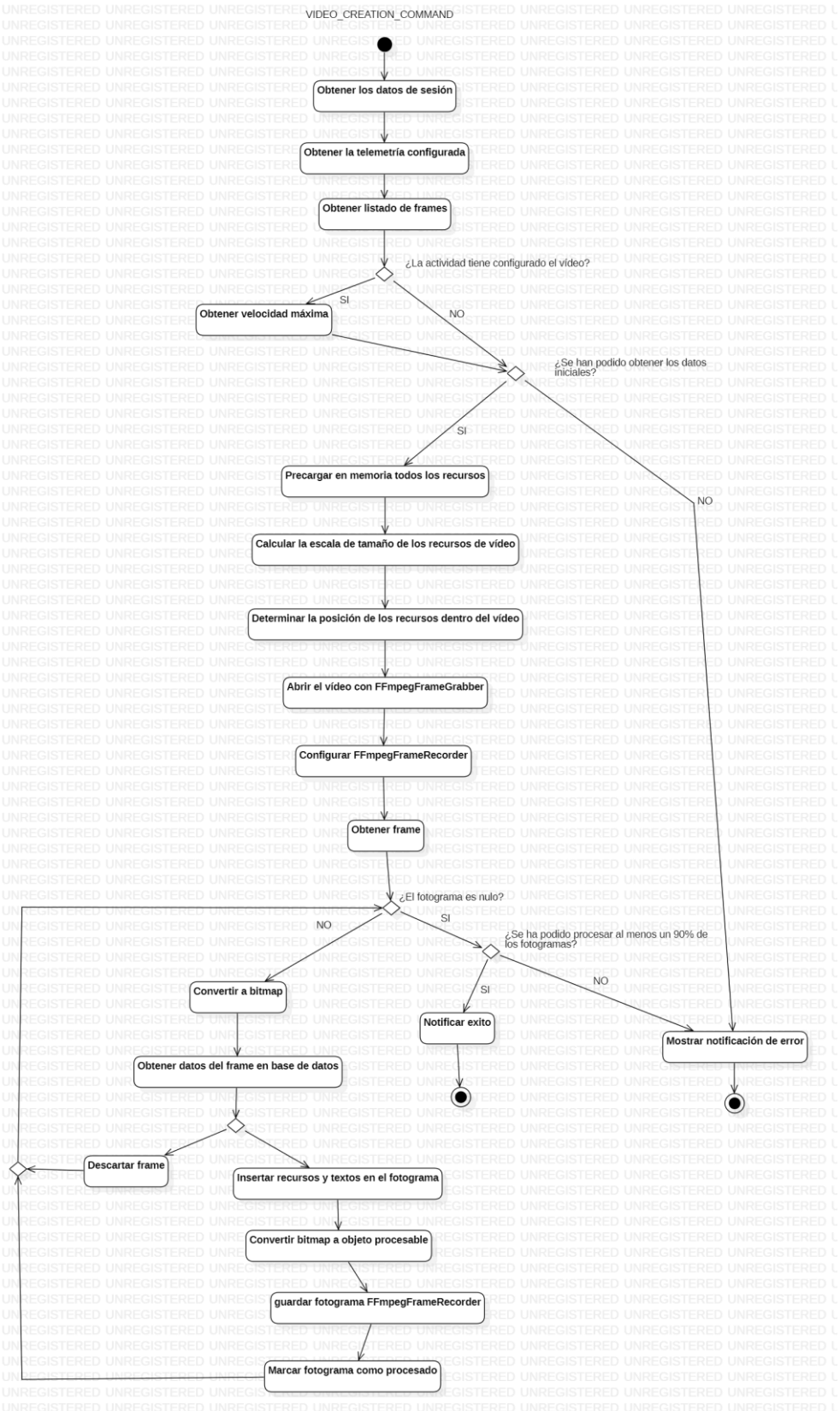


Figura 58 Proceso de composición del vídeo

Se tienen las siguientes consideraciones a la hora de generar el vídeo para intentar minimizar el tiempo de creación de vídeo y evitar *Memory Leaks*.

- Todos los tamaños y posiciones de todos los elementos se calculan antes de empezar la creación del vídeo.
- Todos los recursos de imágenes usados se cargan también antes de empezar la creación del vídeo.
- Si el fotograma a generar tiene la misma marca de sincronización temporal el en fotograma anterior, se usan directamente sus imágenes de velocidad, fuerza y ángulo de inclinación, para evitar cálculos innecesarios.
- Si por algún motivo no se puede generar alguno de los recursos de velocidad, fuerza o ángulo de inclinación, se usará la imagen anteriormente generada, para evitar parpadeos y *gliches* en la imagen.

5.6.2.4 Manejo de recursos de vídeo

Primero clarificar que con recursos me refiero a las imágenes vectoriales usadas para mostrar gráficamente la telemetría en el vídeo. Estas se encuentran en la carpeta drawable, en la zona de recursos del proyecto.

La telemetría asociada a la velocidad, aceleración y ángulo de inclinación, se muestran también con una representación visual, para hacer más dinámico el vídeo.

Los gráficos se insertan dinámicamente en el vídeo, pero no se generan de forma dinámica. Por simplificar su manejo y la fase de creación de vídeo, se han generado una serie de imágenes para cada uno que son asociadas a sus respectivos valores. En cada fotograma se comprueba que valor tiene asignado cada uno y se elige su representación más aproximada. De este modo, aunque se trate de un trampantojo, el usuario percibe un movimiento y transiciones fluidas.

Velocidad:

Las imágenes las diseño y creo usando la herramienta canva.com, que permite realizar diseños de forma rápida, sencilla y exportar en SVG con fondo transparente. Todos los recursos se generan del mismo modo.

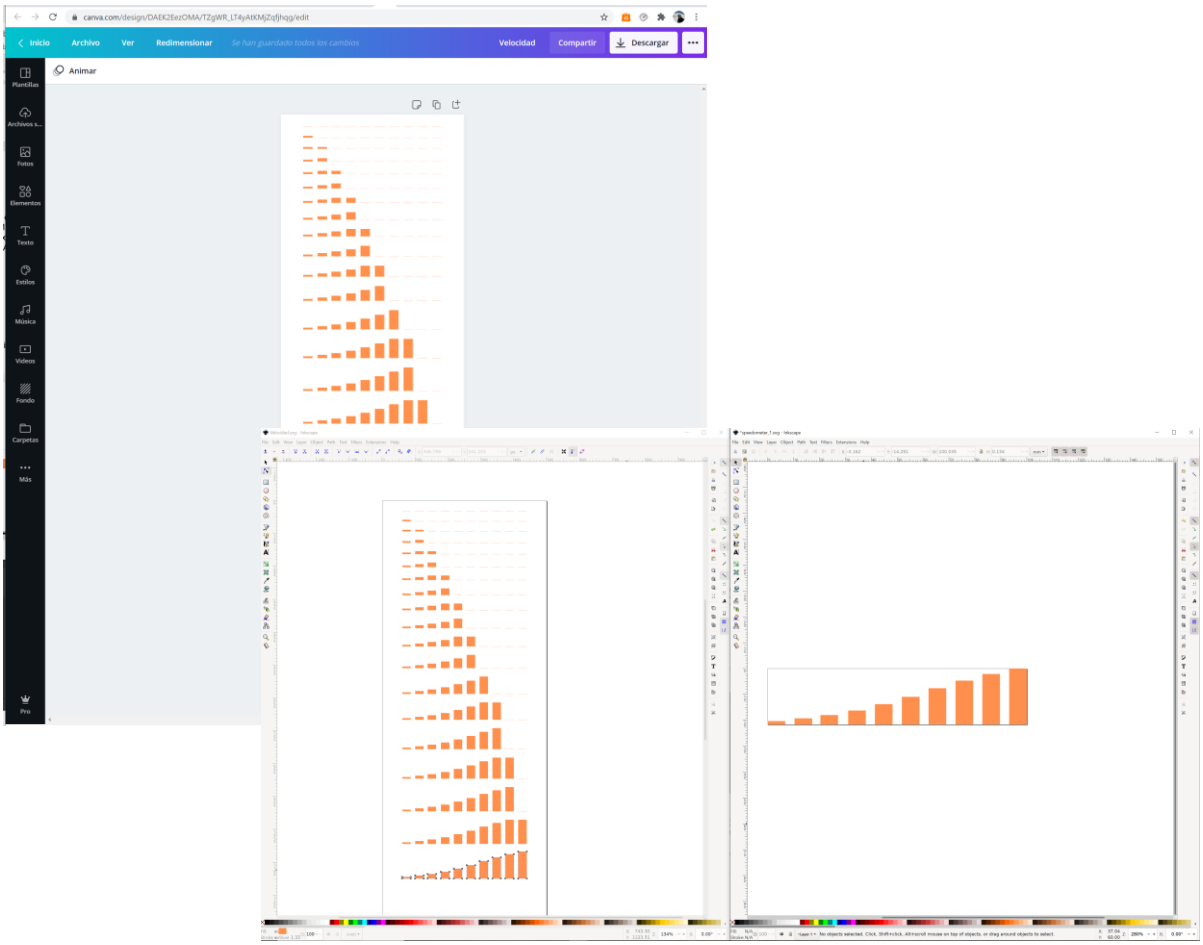


Figura 59 Imágenes del velocímetro en Camba e Inkscape

Una vez se exporta a SVG la imagen, se procesan con Inkscape, para dividir la serie en imágenes individuales, que son insertadas como recursos vectoriales dentro de la carpeta drawable de Android Studio:

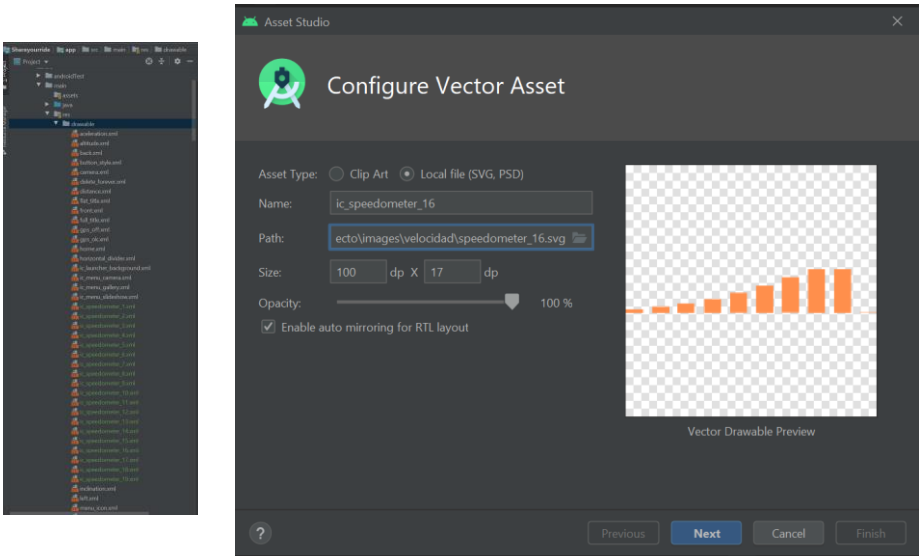


Figura 60 Importación de las imágenes del velocímetro en Android

Antes de iniciar la conversión del vídeo, todos los recursos de velocidad estos se insertan en un mapa ya convertidos a bitmaps, con un tamaño ya definido. Este mapa usa como clave un índice entre 0 y 18.

También antes de empezar la creación se obtiene la velocidad máxima de la actividad y se divide en 19 segmentos. En cada fotograma, se calcula la velocidad a la que va el móvil y se compara con la velocidad máxima para determinar en qué segmento de velocidad está y por lo tanto que imagen le corresponde

Figura 61 Velocidad máxima de la actividad

Ángulo de inclinación:

La forma de importar las imágenes del ángulo de inclinación es similar a las de velocidad, así que no entro en detalles de este punto. La diferencia respecto a la velocidad es que en la fase de Inkscape, se ha añadido también el ángulo, dibujado con trazos vectoriales. Se han generado imágenes para cada segmento de 5º, tanto positivos como negativos.

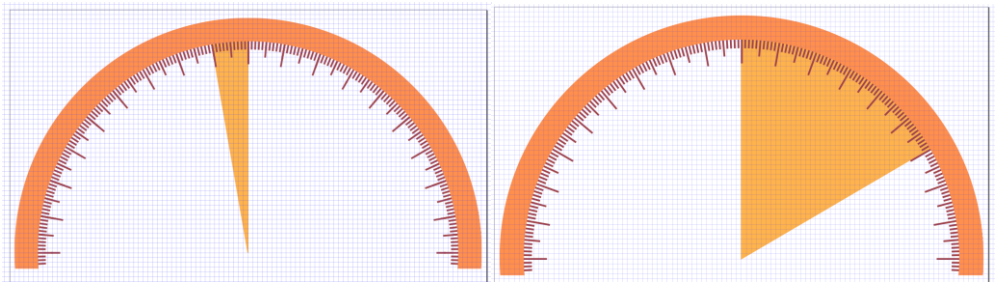


Figura 62 Dos ejemplos de imágenes de ángulo de inclinación en Inkscape

Se ha elegido esta forma por simplicidad, la primera aproximación fue generar el ángulo de forma dinámica anidado canvas y rotando un puntero para señalar el ángulo de forma dinámica, pero no logré conseguir un resultado satisfactorio, mientras que una serie de imágenes discretas, además de ser una implementación más sencilla, también es más limpia y segura.

Al igual que la velocidad, todas las imágenes se cargan al inicio en un mapa. En este caso son dos mapas, uno por sentido, cuyo índice es el ángulo de inclinación.

Fuerza G:

Lo deseable en este punto sería la generación dinámica del gráfico durante la actividad. Debido a los compromisos de fechas, no es abordable esta implementación y opto por crear una serie de imágenes que representan la dirección de la fuerza y si esta es menor, igual o mayor que 1G.

El proceso de generación de las imágenes es el mismo que el de la velocidad, todas las imágenes se han cargado en un mapa. La particularidad en este caso es que el mapa esta indexado por el par dirección y magnitud de la fuerza.

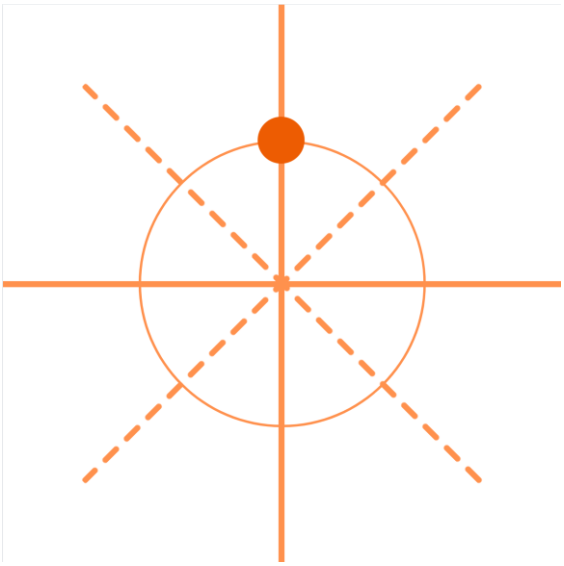


Figura 63 Representación de 1 G de fuerza

6. Decisiones de diseño

En este punto voy a relatar una serie de decisiones de diseño que he ido realizando y que varían o pueden variar el resultado propuesto en la primera entrega:

- Se suprime el login de usuario, una vez que se tiene toda la funcionalidad planteada, se observa que la ventana de login ahora mismo no aporta ningún valor añadido a la aplicación.

- Por simplificar la implementación, ya que el tiempo es muy limitado para tener en cuenta todas las consecuencias, se limita el funcionamiento de la APP a modo retrato. El modo apaisado no se soporta.

- La implementación ideal de los repositorios sería crear un repositorio por entidad de la base de datos, pero en este momento para llegar a un compromiso entre calidad de software y tiempo disponible, se deja así con un único repositorio para la BBDD.

- Debido a los compromisos de fechas, se elimina la previsualización de los videos en las ventanas de home y de transcurso de la sesión.

- Al realizar pruebas de campo, se ha visto que, con la implementación actual, el vídeo con mucho movimiento no es procesable a 720p, con lo cual se reduce la resolución del vídeo a 960x518. No se ha podido determinar la causa del problema de rendimiento. Los *profiling* realizados no dan consumos de CPU excesivos.

- Se soportan única y exclusivamente conexiones RTSP de vídeo. Las librerías soportan otros protocolos de transmisión, que pueden ser fácilmente implementados, pero hay delimitar el alcance de la primera versión.

- No se procesa el audio, este no se incluye en el vídeo resultante.

- Por compromisos de fechas, la aceleración y la inclinación tienen una representación visual limitada con valores discretos.

- La aceleración pasa a medirse como fuerza G (1G = 9.8m*s^2), al ser más visual.

- No se implementan test unitarios ni funcionales con Espresso, por falta de tiempo. Quedan pendientes y es lo próximo a abordar, para garantizar una base con suficiente calidad de software como para poder ampliar la funcionalidad.

- No se soportan versiones anteriores a Android 10 debido que esta versión tiene demasiados cambios de API como para poder gestionar todas las diferencias en este momento.

7. Conclusiones

Si tengo que definir lo que ha supuesto este trabajo, creo que lo que mejor lo resume es la palabra reto. El trabajo ha supuesto un desafío tanto en lo técnico, personal y organizativo, en el cual he tenido que poner mucho esfuerzo y sacrificar gran parte de tiempo libre.

Esta tarea me ha servido para aprender mucho acerca del desarrollo de Android. Me ha obligado a conocer mucho mejor las “tripas” de su arquitectura y las grandes diferencias que tiene respecto a desarrollar para un sistema convencional. No puedo realizar una enumeración de todo lo aprendido, porque la mayor parte del proyecto ha sido un ejercicio de investigación, con lo cual, se puede decir, que todo lo que va más allá de lo básico, ha sido conocimiento adquirido nuevo.

En lo personal, me ha servido para aprender a ser un poquito más expeditivo y tomar decisiones para primar el tiempo de desarrollo frente a ampliar la funcionalidad o la calidad de producto final. Como punto positivo me ha servido para ver lo que soy capaz de hacer, teniendo libertad creativa y de diseño, permitiéndome volver a disfrutar del desarrollo de software.

En líneas generales, creo que he sido capaz de cumplir todos los requisitos funcionales que se plantearon al inicio del proyecto. Algunos han sido recortados, pero todos cumplen, aunque sea de forma básica. Durante el planteamiento inicial, se hizo un filtrado de los requisitos y se eliminaron varias funcionalidades que me hubiera gustado incluir. Esta decisión ha demostrado ser la correcta ya que no habrían sido abordables.

La planificación del proyecto ha sido difícil, en la que cometí algunos errores. He pecado de ser excesivamente optimista en el tiempo que me llevaría realizar la documentación y los trabajos con el vídeo. También tendría que haber dejado una bolsa de horas más grande para la parte de pruebas y calidad de software. He podido seguir con la planificación marcada hasta casi el final de la PEC3, en la cual he tenido que tomar una serie de medidas:

- He tomado decisiones de diseño para simplificar funcionalidades.
- He usado días de vacaciones para corregir la desviación de tiempo. En este punto el actual estado de pandemia ha colaborado a que me sobrasen días de vacaciones.
- Desgranar las tareas en pequeños bloques, “kanbanizando” el desarrollo, me ha ayudado bastante ponerme pequeños hitos y controlar la desviación de tiempos.

He terminado bastante satisfecho con el resultado del proyecto. Tanto que estoy sopesando el seguir desarrollando la APP, tras un buen descanso. Para ello lo siguientes pasos a corto y medio plazo son:

- Automatización de pruebas.

- Mejorar el consumo de recursos al procesar el vídeo en vivo.
- Integración con transmisiones por UDP y RTMP para usar cámaras de otros fabricantes como GoPro, DJI o Sony).
- Soportar distintas resoluciones de vídeo
- Modo apaisado
- Insertar un módulo para enviar comandos para controlar las cámaras remotas (HTTP para GoPro y Telnet para chipsets Ambarela).
- Vista de galería pasa sesiones y vídeos.
- Mejorar los gráficos de fuerza G.
- Audio.
- Traducción al castellano.

8. Glosario

Termino	Significado
Acelerómetro	En el ámbito de este proyecto, elemento destinado a medir las fuerzas de aceleración que afectan al teléfono móvil,
Androd Jetpack	Framework de desarrollo de aplicaciones de Android destinado a simplificar y homogeneizar los desarrollos de aplicaciones.
Colas de mensajes	Sistema que permite a diferentes partes de la arquitectura del Software comunicarse entre ellas de manera asíncrona sin generar interdependencias.
Framework	Conjunto de librerías o tecnologías destinadas al desarrollo de software
Fuerza G	Medida de aceleración que produce la gravedad de la tierra a un objeto. G equivale a 9.8 m/s^2. En el ámbito de la aplicación se usa como referencia para cuantificar las aceleraciones del teléfono móvil en los planos X e Y comparándolas con la gravedad terrestre.
Giroscopio	En el ámbito de este proyecto, elemento hardware que mire la orientación en el espacio del teléfono móvil.
GPS	Sistema de posicionamiento global que permite localizar un objeto en unas coordenadas determinadas en la superficie de la tierra.
Gráfico vectorial	Imagen formada por elementos geométricos en vez de un mapa de pixeles. Esto permite que las imágenes sean escalables sin perdida de calidad.
LiveData	Implementación del patrón observer por parte de Google. Añadoda en el framwork JetpPack.
Memory Leak	Fallo en el código que provoca acumulación paulatina de recursos en memoria RAM que nunca son liberados y pueden causar el fallo de la aplicación.
MVVM	Model View View Model: Patron de diseño adoptado por Google recientemente para el desarrollo de aplicaciones Android. Desacopla la interfaz de la lógica de negocio y es altamente reusable
Patrón observer	Patrón de comportamiento donde el cambio de estado de un componente es notificado al resto de componentes que lo están observando.
Programación orientada a eventos	Se trata de una metodología de programación en la cual se estructura el código y el flujo de acción para responder por eventos sucedidos dentro del sistema o por acción del usuario. Su uso es esencial si se quiere una arquitectura de software asíncrona.
Room	Biblioteca de persistencia de Google añadida dentro del *framework* Jetpack para abstraer al programados de la

	gestión de la base de datos de la aplicación
RTSP	Real Time Streaming Protocol: protocolo de transmisión en tiempo real de vídeo y/o audio. Es común mente utilizado en cámaras IPs.
RX o reactive extensions	Librerías que simplifican y extienden la funcionalidad del patrón observer. Dotan de mayor de control de los flujos de datos y los eventos.
Servicios	Componente de la arquitectura de Android destinado a la ejecución de tareas de larga duración que no requieren de interfaz de usuario.
SSID	Identificador de una red WIFI
Stream de vídeo	Flujo de datos perteneciente a una transmisión de vídeo.
UUID	Identificador único universal, utilizado para identificar inequívocamente partes del código.
Vector	Termino matemático que representa medidas como la fuerza, la velocidad o la aceleración indicando magnitud, dirección y sentido.
View Model	
WIFI	Tecnología de comunicación inalámbrica
Wrapper	En programación se refiere a clases o librerías que envuelven a otras clases o librerías. Por norma general o simplifican o extienden el uso de la parte envuelta.

Tabla 15 Glosario de términos

9. Bibliografía

1. **Kbv research** (2020), Action camera market size <https://www.kbvresearch.com/action-camera-market/>
2. **Grand view reseach** (2019), Action camera market size <https://www.grandviewresearch.com/industry-analysis/action-camera-market>
3. **Wondershare** (2020), Top 12 Action cams with GPS: <https://filmora.wondershare.com/action-camera/best-action-cameras-with-gps.html>
4. **GoPro** (2020), Telemtry and GOS add new layers to your GoPro story <https://gopro.com/es/es/news/New-Telemetry-Feature-Quik-Desktop-App>
5. **Track Addick** (2020), Using your smartphone for the track is easier than ever <http://racerender.com/TrackAddict/Features.html>
6. **GoPro telemetry extractor** (2020), Telemetry overlay for any camera <https://goprotelemetryextractor.com/gpx-telemetry-overlay>
7. **Adobe aftereffects** (2020) <https://www.adobe.com/products/aftereffects.html>
8. **Mockplus** (2020) < https://www.mockplus.com/>
9. **Android** (2020,)Android Foregorund services (2020): <https://robertohuertas.com/2019/06/29/android_foreground_services/>
10. **Android** (2020), Services (2020): https://developer.android.com/guide/components/services
11. **Android** (2020), Transaction: <https://developer.android.com/reference/android/arch/persistence/room/Transaction>
12. **Android** (2020), Transaction: <https://developer.android.com/reference/android/arch/persistence/room/Transaction>
13. **Android** (2020), Sensores de movimiento: https://developer.android.com/guide/topics/sensors/sensors_motion
14. **Android** (2020): Sistema de coordenadas <https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-coords>
15. **Android** (2020) Cómo conocer la ubicación más reciente <https://developer.android.com/training/location/retrieve-current#last-known>
16. **Android**(2020) MediaPlayer <https://developer.android.com/reference/android/media/MediaPlayer>
17. **Android**(2020) Exoplayer <https://developer.android.com/guide/topics/media/exoplayer?hl=es-419>
18. **Gihub**(2020) Exoplayer Branch(3854) < https://github.com/tresvecesseis/ExoPlayer>
19. **OpenCv**(2020) OpenCV <https://opencv.org/>
20. **Gihub** (2020) JavaCv <https://github.com/bytedeco/javacv>
21. **Android developers** (2020) Configuración <https://developer.android.com/guide/topics/ui/settings?hl=es-419>

22. **ReactiveX** (2020) Debounce <http://reactivex.io/documentation/operators/debounce.html>
23. **Android developers** (2020) Restricciones sobre el acceso directo a redes WiFi configuradas <https://developer.android.com/about/versions/10/privacy/changes#configure-wifi>
24. **Android developers** (2020) Descripción general de la búsqueda de Wi-Fi <https://developer.android.com/guide/topics/connectivity/wifi-scan?hl=es-419>
25. **Android developers** (2020): Como crear una notificación <https://developer.android.com/training/notify-user/build-notification>
26. **Android developers** (2020): Como interpretar el ciclo de vida de una actividad <https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es>
27. **Android developers** (2020) Location accuracy <https://developer.android.com/reference/android/location/Location.html#getAccuracy()>

10. Anexos

- Mensajería interna
- Manual de usuario
- Manual de instalación y configuración del entorno de desarrollo y test.
- Presentación