eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Ingeniería de Software

Infraestructura de pruebas en Android: estudio de Firebase y otras herramientas de monitoreo de la calidad

Unai Pinedo Molina

Dirección
Maider Azanza Sesé
Beatriz Pérez Lamancha

22 de junio de 2023

Resumen

Este documento corresponde a la memoria del Trabajo de Fin de Grado con título "Infraestructura de pruebas en Android: estudio de Firebase y otras herramientas de monitoreo de la calidad" en colaboración con la empresa LKS NEXT. El proyecto se ubica dentro de la rama de calidad del software, concretamente en el mantenimiento y testing de las aplicaciones móviles.

Se han estudiado las herramientas que ofrece Firebase para el seguimiento de la calidad: Crashlytics, Performance Monitoring y Test Lab y se ha diseñado una forma de trabajo para integrarlas en el proceso de desarrollo de aplicaciones Android de la empresa. Además, se ha automatizado la exportación de los datos que generan estas herramientas, y se han comparado con otras herramientas similares en algunos casos.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de códigos

CAPÍTULO 1

Introducción

El uso de aplicaciones móviles ha experimentado un gran crecimiento en los últimos años debido al aumento del uso de dispositivos móviles en todo el mundo. Sin embargo, el éxito de una aplicación móvil depende en gran medida de su calidad y usabilidad.

La calidad del software [1] es el grado con el que un producto o proceso cumple con los requisitos especificados anteriormente, por lo que es vital que los requisitos estén bien definidos y sean precisos y medibles. En las aplicaciones móviles, la calidad es fundamental para proporcionar una experiencia de usuario satisfactoria y mantener la fidelidad de los usuarios. Los usuarios esperan que las aplicaciones móviles sean rápidas, fiables y fáciles de usar. Por esto, la gestión de la calidad en las aplicaciones móviles es un aspecto de gran importancia en su desarrollo.

Lograr una gestión de calidad eficaz requiere considerar un enfoque en el que se tengan en cuenta aspectos como pruebas funcionales, de rendimiento, de compatibilidad, de usabilidad y de seguridad. También es importante considerar los requisitos específicos de la plataforma móvil, como la compatibilidad con diferentes dispositivos, sistemas operativos y versiones.

Por otro lado, la falta de un control de calidad adecuado en el desarrollo de aplicaciones móviles puede tener graves consecuencias para los desarrolladores y usuarios. Una mala aplicación móvil puede provocar la pérdida de datos o incluso exponer información confidencial. Además, si la aplicación no funciona correctamente, los usuarios pueden abandonar la aplicación y buscar alternativas.

Por todo esto, es de vital importancia que los desarrolladores sean conscientes de la importancia de la calidad en aplicaciones móviles y estén dispuestos a invertir tiempo y recursos en la gestión de la misma.

Dentro de este contexto de la gestión de la calidad en entornos de desarrollo para aplicaciones móviles, este Trabajo de Fin de Grado (TFG) se ha llevado a cabo en la empresa LKS NEXT[1]. LKS NEXT se encarga de ofrecer al mercado soluciones informáticas. Durante los últimos años, se ha adaptado a las necesidades de los clientes, introduciendo en su oferta de productos el desarrollo de aplicaciones Android desde hace aproximadamente diez años.

[1] LKS NEXT: https://www.lksnext.com/es

En el área de calidad, la metodología de desarrollo móvil de la empresa se divide en dos etapas. Por una parte, cuando la aplicación se encuentra en desarrollo, el equipo de desarrolladores Android se encarga de diseñar y desarrollar tests unitarios y de integración utilizando Robolectric[2] y tests funcionales utilizando UI Automator[3], Espresso[4] y Cucumber[5]. Además, se utiliza Sonarqube[6] para analizar estáticamente todos los proyectos en los que se trabaja.

Por otro lado, cuando la aplicación se ha desplegado, se utiliza Crashlytics[7] para monitorizar los errores que ocurren en la aplicación, aunque resulta difícil para el equipo manejar toda la información que ofrece la herramienta. Además, la manipulación de estos datos se hace de manera manual.

En este proyecto la responsable de calidad de LKS NEXT y directora del proyecto, Beatriz Pérez Lamancha, busca solucionar este problema además de integrar en la metodología de la empresa el resto de las herramientas para control de calidad que proporciona Firebase[8]. Principalmente, se busca integrar estas herramientas cuando la aplicación ya ha sido puesta en producción.

Concretamente, se utilizarán las siguientes herramientas de Firebase:

- **Crashlytics:** Esta herramienta se utiliza para registrar y crear reportes de los crashes y errores recuperables que ocurren en una aplicación en tiempo real. Estos reportes son de gran utilidad para detectar fallos y solucionarlos rápidamente.
- **Performance Monitoring[9]:** Esta herramienta se utiliza para obtener estadísticas sobre los problemas de rendimiento de una aplicación. De esta manera, se puede utilizar esa información para detectar donde se encuentran estos problemas de rendimiento y poder mejorarlos.
- **Test Lab[10]:** Esta herramienta es una infraestructura de prueba de aplicaciones basada en la nube que permite probar aplicaciones en una gran variedad de dispositivos y configuraciones, para ver y probar el funcionamiento de la aplicación en diferentes entornos.

El objetivo de este TFG ha sido analizar estas herramientas para posteriormente crear cuadros de mando en los que se muestra la información más relevante que proporcionan de manera lo más automatizada posible, haciendo posible presentar de manera clara y sencilla la información más importante sobre el desempeño de las aplicaciones. De esta forma, se permite a los clientes evaluar el estado de las aplicaciones en las reuniones periódicas que se tengan con la empresa.

El resultado principal de este proyecto ha sido la documentación para el uso y manipulación de los datos que ofrecen estas herramientas. Además se ha hecho una propuesta de

[2] Robolectric: https://robolectric.org
[3] UI Automator: https://developer.android.com/training/testing/ui-automator
[4] Espresso: https://developer.android.com/training/testing/espresso
[5] Cucumber: https://cucumber.io
[6] Sonarqube: https://docs.sonarqube.org
[7] Crashlytics: https://firebase.google.com/products/crashlytics
[8] Firebase: https://firebase.google.com
[9] Performance Monitoring: https://firebase.google.com/products/performance
[10] Test Lab: https://firebase.google.com/products/test-lab

proceso que permite incorporar las herramientas en el desarrollo móvil de la empresa y aplicarlo a los proyectos en los que trabaja LKS NEXT.

De esta manera, se ha podido proporcionar a los clientes información sobre el estado de las aplicaciones que esté desarrollando LKS NEXT, teniendo en cuenta los errores que se hayan producido, el rendimiento de la aplicación y las pruebas que se hayan realizado en variedad de dispositivos gracias a Firebase.

CAPÍTULO 2

Planificación

En este capítulo se detalla la planificación del proyecto, describiendo el alcance, periodos de realización de tareas y la gestión del tiempo, de riesgos, de comunicaciones e información y de los interesados. El objetivo de esta planificación es que al final del proyecto se satisfagan todos los objetivos definidos en la misma.

2.1. Alcance

Este proyecto trata sobre el análisis en profundidad de las herramientas de Firebase: Crashlytics, Performance y Test Lab, de las que se pretende hacer un estudio para posteriormente poder integrarlas en la metodología de desarrollo de la empresa.

Actualmente, la única de ellas que se utiliza en la empresa es Crashlytics, creando informes con los datos que proporciona sobre la cantidad de crashes en los últimos treinta días que ha tenido una aplicación en producción (usuarios sin fallos, usuarios totales, crashes encontrados y cuáles de ellos han sido revisados) y detalles sobre los crashes.

Esta tarea actualmente se realiza a mano por Beatriz Pérez, ya que Crashlytics no permite exportar los datos de manera sencilla ni dispone de API. Por esto, una de las metas del proyecto es encontrar la manera de hacer este proceso más rápido y sencillo.

Aun así, el alcance del proyecto no está definido al completo, ya que dada la falta de conocimiento previo hace que sea difícil estimar de antemano el esfuerzo que conllevará llevarlo a cabo, por lo que cabe la posibilidad de cambiar el alcance a lo largo del proyecto.

2.1.1. Objetivos concretos del proyecto

El objetivo principal de este proyecto es analizar en profundidad las herramientas Crashlytics, Performance y Test Lab de Firebase e integrarlas en la metodología de desarrollo de aplicaciones móviles de LKS NEXT. Aun así, junto a este objetivo encontramos otros objetivos, que ayudan a la consecución del objetivo principal:

- **Analizar las herramientas para exportar datos de Crashlytics:** Realizar un estudio de las herramientas que se encuentren para exportar los datos de Crashlytics y manipularlos.

- **Analizar el apartado de Performance de Firebase:** Analizar en profundidad la herramienta, seleccionar métricas que sean útiles para mostrar en el apartado de Performance según los intereses de la empresa y del cliente.
- **Analizar el apartado de Test Lab de Firebase:** Analizar en profundidad la herramienta.
- **Estimar un coste del uso de las herramientas:** Como es un servicio que se pretende vender a clientes en un futuro, se requiere una estimación de los costes que tendría por parte de Firebase dependiendo de la aplicación que se esté analizando.
- **Crear una forma de trabajo en la que se integre el uso de estas herramientas en la empresa:** Analizar la forma de trabajo actual de la empresa y encontrar la manera de integrar las herramientas que se han estudiado.
- **Aplicar las herramientas en un proyecto real de la empresa:** Una vez se haya diseñado la forma de trabajo, aplicar lo aprendido a un proyecto real de LKS NEXT. Para llevar a cabo esta tarea, se utilizarán como casos de prueba aplicaciones móviles reales desarrolladas dentro de la empresa, como BIDEmotion (una aplicación para cuidar la salud emocional de los trabajadores a través de un itinerario personalizado) o GPMobile (una aplicación para realizar las imputaciones de horas de los trabajadores de la empresa).

2.1.2. Requisitos

Para que el proyecto se realice satisfactoriamente, es vital que los requisitos estén bien definidos y validados por la responsable de la empresa, para poder así dar por concluido el proyecto cuando todos los requisitos se cumplan.

Requisitos:

- **R-1:** Estudio de lo que ofrece cada una de las herramientas de Firebase para pruebas no funcionales. En concreto:
  - **R-1.1:** Estudio de Crashlytics.
  - **R-1.2:** Estudio de Performance Monitoring.
  - **R-1.3:** Estudio de Test Lab.
- **R-2:** Conseguir detectar y localizar fallos que se hayan identificado previamente en las aplicaciones utilizando las herramientas que ofrece Firebase.
- **R-3:** Análisis de las posibilidades que brinda Crashlytics para la exportación y manipulación de los datos que ofrece.
- **R-4:** Estimación de los costes por el uso de las herramientas que proporciona Firebase.
- **R-5:** Creación de una forma de trabajo para integrar Firebase en la metodología de desarrollo móvil de la empresa.
- **R-6:** Aplicación de las herramientas en un proyecto real de la empresa LKS NEXT como BIDEmotion o GPMobile.
- **R-7:** Documentación de las lecciones aprendidas durante el proyecto.

2.1.3. Fases del proyecto

Como ya se ha comentado, el alcance del proyecto aún no está definido al completo, por lo que se propone un ciclo de vida iterativo-incremental, en el que cada en cada una de las iteraciones se analizará una de las tres herramientas de las que se ha hablado: Crashlytics, Performance y Test Lab, finalizando con una iteración en la que se propondrá una forma de trabajo en la que se integre Firebase en la metodología de desarrollo de la empresa y se aplique a un proyecto real de la misma.

Gracias a este tipo de ciclo de vida, se alcanzan los objetivos del proyecto de manera progresiva, realizando los diferentes procesos en ciclos, teniendo así feedback directo por parte de las directoras del proyecto Beatriz Pérez y Maider Azanza, evitando posibles retrasos al conocer en todo momento el estado del proyecto y pudiendo redirigirlo si fuera necesario.

2.1.4. Descomposición de tareas

La Estructura de Descomposición de Trabajo (EDT) del proyecto se ha creado teniendo en cuenta el ciclo de vida iterativo-incremental del proyecto. Se puede ver un diagrama de la misma en la figura 2.1.

Se divide en los siguientes paquetes:

- **Estudio inicial (EI):** En la fase inicial del proyecto se debe realizar un estudio corto sobre lo que se trabajará durante el proyecto.
  - *Estudio Android (EA):* Las herramientas de Firebase se quieren utilizar en aplicaciones Android, por lo que es imprescindible aprender lo básico sobre cómo funcionan las aplicaciones Android para posteriormente entender los fallos que den las mismas, y poder hacer pruebas sobre ellas. También se estudiarán aplicaciones de la empresa con las que se podrán ver datos reales, como GPMobile o BIDEmotion.
- **Crashlytics (CR):** En esta primera iteración se estudiará la herramienta Crashlytics.
  - *Análisis (ACR):* Estudio y análisis de la herramienta.
  - *Estudio Big Query (BQ):* Estudio de la herramienta Big Query.
  - *Estudio Exportación y Manipulación de Datos (EMD):* Estudio de las posibilidades que brinda Crashlytics para exportar y manipular los datos que ofrece.
  - *Estimación Costes (EC):* Estimación de los costes de Firebase por el uso de la herramienta dependiendo del tamaño del proyecto.
- **Performance (PR):** En la segunda iteración se estudiará la herramienta de Performance.
  - *Análisis (APR):* Estudio y análisis de la herramienta.
  - *Estimación Costes (EC):* Estimación de los costes de Firebase por el uso de la herramienta dependiendo del tamaño del proyecto.
- **Test Lab (TL):** En la tercera iteración se estudiará la herramienta Test Lab.

- *Análisis (ATL):* Estudio y análisis de la herramienta.
- *Estimación Costes (EC):* Estimación de los costes de Firebase por el uso de la herramienta dependiendo del tamaño del proyecto.

- **Forma de trabajo (FT):** Creación de una forma de trabajo en la que se integra Firebase en la metodología de desarrollo de la empresa.
  - *Análisis (AFT):* Análisis de la metodología de desarrollo de la empresa.
  - *Diseño (DFT):* Diseño de forma de trabajo integrando Firebase para incorporar a la metodología de desarrollo de la empresa.
- **Gestión (G):**
  - *Captura de requisitos (CR):* Captura de requisitos bien definidos y validados al comienzo del proyecto.
  - *Planificación (PL):* Planificación a realizar para mantener el control del proyecto.
  - *Seguimiento y Control (SyC):* Seguimiento y control de los puntos definidos en la planificación, que se controlarán con reuniones periódicas con las directoras del proyecto (cada 2 semanas aproximadamente).
- **Trabajo académico (TA):**
  - *Gestión TFG (GT):* Gestionar los trámites académicos relacionados con el TFG.
  - *Memoria (M):* Redacción de la memoria a lo largo de todo el proyecto.
  - *Defensa (D):* Preparación de la defensa del proyecto, incluida la realización de una presentación para la misma.

2.2. Periodos de realización de las tareas e hitos

En este apartado se analizarán las dependencias entre las diferentes tareas, y se estimará la duración y fechas estimadas de cada una de ellas.

Figura 2.1: Diagrama EDT

2.2.1. Dependencias entre tareas

Las tareas del proyecto tienen dependencias entre sí, importantes de identificar para llevar a cabo una buena planificación. A continuación, en la figura 2.2 se presentan los paquetes que dependen de otros:

El proyecto comienza con la captura de requisitos **(G.CR)** para posteriormente poder hacer la planificación correctamente **(G.PL)**. A continuación, se realizará el estudio de Android **(EI.EA)**, para posteriormente realizar las 3 iteraciones correspondientes a cada herramienta.

En las tres encontramos primero una fase de Análisis **(XX.AXX)** y una fase de estimación de costes **(XX.EC)**. Además, la iteración en la que se estudia Crashlytics también tiene fases de estudio de BigQuery **(CR.BQ)** y de Estudio Exportación y Manipulación de Datos **(CR.EMD)** después del análisis.

Después de estas iteraciones se realizará la iteración final en la que se analizará la forma de trabajo de LKS NEXT **(FT.AFT)** y se diseñará la integración de Firebase en la misma **(FT.DFT)**.

Al mismo tiempo, durante todo el proyecto se llevará a cabo el seguimiento y control, con reuniones periódicas **(G.SyC)**, mientras se va escribiendo la memoria **(TA.M)**, según se vayan completando el resto de las tareas.

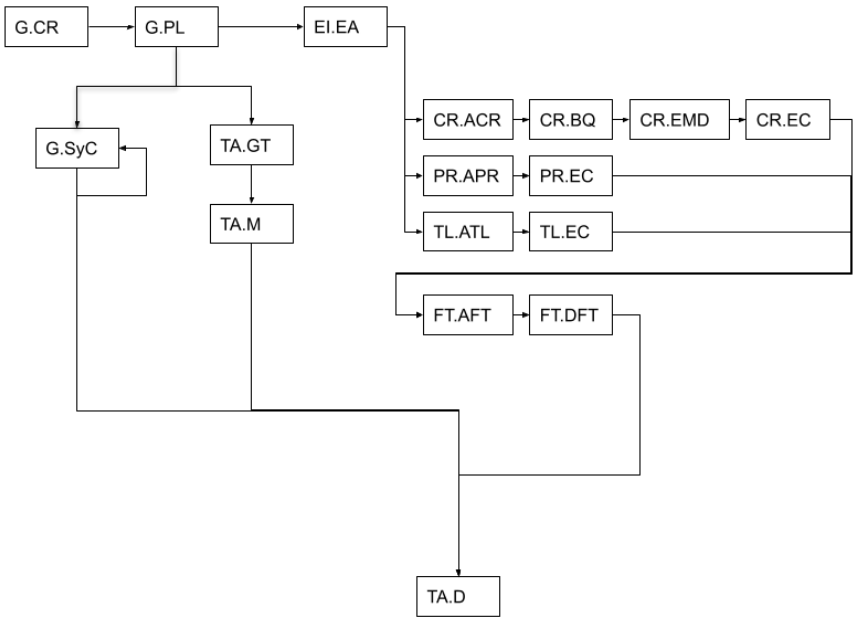


Figura 2.2: Dependencias entre tareas

2.2.2. Diagrama de Gantt

En la figura 2.3 se presenta el diagrama de Gantt, en el que se puede ver de manera aproximada la duración del proyecto en el tiempo, y a qué paquete se dedicará el tiempo en cada momento.

2.2.3. Hitos

En la tabla 2.1 encontramos los hitos que se esperan en el desarrollo del proyecto.

Se ha planificado que la memoria se finalice para el 31 de mayo, habiendo 25 días hasta la fecha límite para la entrega de la misma para la convocatoria de julio. De esta manera, se cuenta con un margen de tiempo por si se dieran desviaciones en la planificación inicial.

2.3. Gestión del tiempo

Teniendo en cuenta el alcance definido, se gestiona el tiempo del que se dispone para cumplir con todos los objetivos establecidos.

2.3.1. Estimación de cada tarea

La estimación de cada tarea se puede ver en la tabla 2.2.

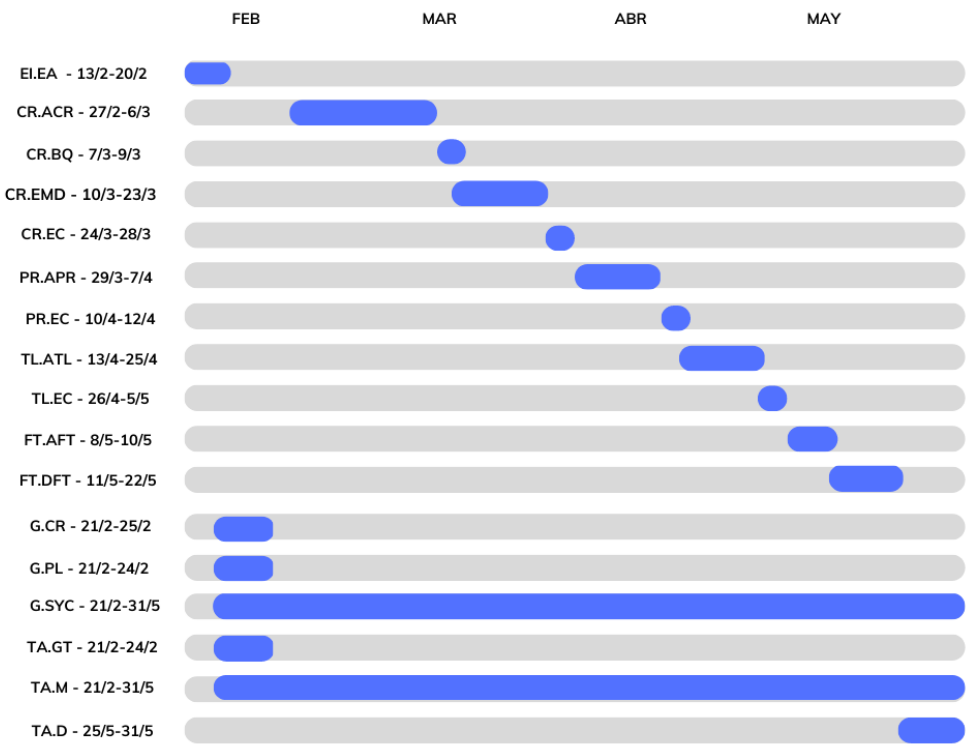


Figura 2.3: Diagrama de Gantt del proyecto

HITO	FECHA
Inicio del proyecto	13/02/2023
Fin de estudio	20/02/2023
Fin It. 1 (Crashlytics)	28/03/2023
Fin It. 2 (Performance)	12/04/2023
FIn It. 3 (Test Lab)	05/05/2023
Fin It. 4 (Forma de Trabajo)	24/05/2023
Fin memoria	31/05/2023
Defensa del proyecto	03-14/07/2023

Tabla 2.1: Hitos del proyecto

PAQUETE DE TRABAJO	TAREA	DEDICACIÓN ESTIMADA
ESTUDIO INCIAL (EI)	Estudio Android (EA)	30h
	SUBTOTAL	**30h**
CRASHLYTICS (CR)	Análisis (ACR)	20h
	Estudio Big Query (BQ)	20h
	Estudio Exportación y Manipulación de Datos (EMD)	30h
	Estimación de costes (EC)	5h
	SUBTOTAL	**75h**
PERFORMANCE (PR)	Análisis (APR)	25h
	Estimación de costes (EC)	5h
	SUBTOTAL	**30h**
TEST LAB (TL)	Análisis (ATL)	25h
	Estimación de costes (EC)	5h
	SUBTOTAL	**30h**
FORMA DE TRABAJO (FT)	Análisis (DFT)	5h
	Diseño (DFT)	30h
	SUBTOTAL	**35h**
GESTIÓN (G)	Captura de requisitos (CR)	5h
	Planifiación (PL)	20h
	Seguimiento y Control (SyC)	20h
	SUBTOTAL	**45h**
TRABAJO ACADEMICO (TA)	Gestión TFG (GT)	3h
	Memoria (M)	70h
	Defensa (D)	10h
	SUBTOTAL	**83h**
HORAS TOTALES		**328h**

Tabla 2.2: Horas estimadas para cada tarea del proyecto

2.4. Gestión de riesgos

Uno de los aspectos más importantes del proyecto es la gestión de riesgos, debido a que es un proyecto con un riesgo medio-alto, ya que el alcance no está aún definido por completo.

Para mitigar esto, se define la siguiente gestión de riesgos:

- **R1-Compaginación con curso académico:**
  - *Descripción:* Debido a que el proyecto se desarrollará durante el segundo cuatrimestre del curso académico, es importante tener en cuenta las semanas en las que más carga de trabajo habrá, ya que podrían ocurrir desviaciones en la planificación si fueran necesarias.
  - *Prevención:* Realizar la planificación teniendo en cuenta semanas agrupadas, exámenes y cualquier otro punto de carga de trabajo alta que pudiese provocar desviaciones.
  - *Plan de acción:* Si la semana del 10 al 14 de abril (la anterior a la segunda semana agrupada) se detecta un retraso de la planificación de más de 25 horas a lo estimado para esa fecha, teniendo en cuenta que deberían haberse realizado unas 200 horas, se realizará una nueva planificación.
- **R2-Captura de requisitos:**
  - *Descripción:* Es de gran importancia que los requisitos estén bien definidos y validados por la empresa, ya que la planificación se hace basándose en ellos. Si no estuviesen bien definidos, la planificación podría sufrir cambios.
  - *Prevención:* Definir los requisitos en una fase temprana del proyecto y que la directora del proyecto por parte de LKS NEXT los valide.
  - *Plan de acción:* Si los requisitos cambian a lo largo del proyecto, documentarlo y hacer cambios en la planificación si fuese necesario.
- **R3-Tecnologías desconocidas por la empresa:**
  - *Descripción:* La empresa no dispone de un gran conocimiento sobre las herramientas que se van a estudiar durante el proyecto y con las que luego se debe trabajar, lo que dificulta pedir ayuda en caso de necesitarla.
  - *Prevención:* Se estimará una gran cantidad de horas al estudio de cada herramienta, para que haya tiempo para estudiarla en profundidad y solucionar los problemas que puedan surgir.
  - *Plan de acción:* Si en algún momento se encuentran grandes dificultades para entender alguna de las herramientas, la empresa comprará cursos que ayuden al alumno a entenderlas completamente.
- **R4-Planificación incorrecta:**
  - *Descripción:* Debido a que el alcance del proyecto no está definido del todo y tiene una gran amplitud, puede ser que a lo largo del proyecto el alcance cambie y haya que hacer cambios en la planificación.
  - *Prevención:* Se realizará un seguimiento estricto por parte de las directoras del proyecto, con reuniones cada 15 días para revisar en todo momento el proyecto y redirigirlo si fuese necesario.
  - *Plan de acción:* Si el alcance cambiase a lo largo del proyecto, documentarlo y hacer los cambios pertinentes en la planificación.

2.5. Gestión de Comunicaciones e Información

2.5.1. Sistema de Información

Es de gran importancia mantener la información disponible y evitar posibles pérdidas en todo momento.

Se utilizará Google Drive como sistema de información, almacenando toda la información y archivos relacionados con el proyecto en una carpeta compartida (p. ej.:gráficas, tablas, archivos para el seguimiento de horas...). De esta manera, la información está accesible y segura en todo momento, siendo muy baja la probabilidad de pérdida de los archivos. Además, las directoras del proyecto tendrán acceso a esta carpeta, facilitando así la labor de dirección.

Además, la memoria del proyecto se escribirá en la plataforma Overleaf, lo que permite un control de versiones y la posibilidad de compartir el documento a los interesados.

2.5.2. Sistema de Comunicación

La comunicación en este proyecto es de suma importancia, ya que es necesario que haya una comunicación rápida y directa entre los interesados: alumno, directora académica y directora en la empresa.

Por ello, se han definido las siguientes herramientas de comunicación:

- **Correo electrónico:** Será la principal herramienta de comunicación. Se utilizará para comunicarse tanto con Beatriz Perez como Maider Azanza, para resolver dudas, concretar reuniones y para el seguimiento y control del proyecto.
- **Google meet:** Será la herramienta que se utilizará para las reuniones telemáticas. Es la elegida ya que es la herramienta que utilizan en LKS NEXT para las reuniones, y es sencilla de usar y fiable. Las reuniones se harán de manera telemática para mayor comodidad de los interesados, aunque se plantea la posibilidad de hacerlas híbridas en ocasiones.

Reuniones

Todas las reuniones que se realicen se verán reflejadas en las actas, que se pueden encontrar anexas a este documento. De esta manera, se documenta el progreso del proyecto y se tiene constancia de las tareas a realizar para la próxima reunión, vital para llevar un seguimiento de manera más sencilla y evitar malentendidos.

2.6. Gestión de los interesados

Es importante identificar a los interesados del proyecto, ya que todos los interesados deberían estar al tanto del desarrollo del proyecto y satisfechos con el resultado final. En este caso, las personas interesadas son:

- **Alumno:** Unai Pinedo Molina
- **Directora académica (UPV/EHU):** Maider Azanza Sese

- **Directora por parte de la empresa (LKS NEXT):** Beatriz Pérez Lamancha

La directora por parte de la universidad tiene como objetivo encargarse de la parte académica del proyecto: gestión y calidad académica del proyecto y supervisión de la memoria.

Por otro lado, la directora por parte de la empresa se encargará de que los objetivos del proyecto se cumplan, liderando al alumno para que todos los requisitos definidos se cumplan al final del mismo.

CAPÍTULO 3

Aplicaciones a utilizar

En este capítulo se describirán y explicarán en detalle las aplicaciones que se van a utilizar para llevar a cabo el análisis de las herramientas de Firebase y extraer conclusiones sobre las mismas. Se presentarán en detalle las características y funcionalidades de estas aplicaciones, y por qué se han seleccionado para realizar el estudio.

Debido a la sensibilidad de los datos de las aplicaciones de clientes con los que trabaja LKS NEXT, se ha decidido no trabajar con estas aplicaciones para hacer las pruebas, por lo que se utilizarán dos aplicaciones internas.

La primera está desarrollada por el alumno teniendo en cuenta las necesidades específicas de cada una de las herramientas, para poder modificarla según las funcionalidades específicas que se necesiten probar de cada herramienta. La segunda aplicación se utiliza para la gestión e imputación de horas de la empresa. Esta es más compleja y cuenta con tests ya implementados.

3.1. Aplicación desarrollada por el alumno

Esta aplicación está desarrollada en Java completamente por el alumno. De esta manera, además, se adquirirá conocimiento sobre desarrollo en Android, lo cual será muy útil para poder más adelante analizar las herramientas entendiendo las métricas e información que ofrecen.

La aplicación no tiene ninguna funcionalidad específica, ya que se utilizará dependiendo de las necesidades de cada herramienta. Por ejemplo, podrían añadirse botones en diferentes fragmentos que lanzan excepciones recuperables y no recuperables, simplemente para generar datos sobre estos errores en Crashlytics y poder trabajar con ellos y analizarlos. También se podrán añadir solicitudes de red que cargan imágenes para poder analizar las métricas de rendimiento que recoge Performance Monitoring.

De esta manera, se conseguirá generar una buena cantidad de datos con los que poder trabajar posteriormente de manera rápida y sencilla, cosa mucho más complicada si es una aplicación real y de la que no saltan prácticamente excepciones, por ejemplo.

3.2. Aplicación de gestión de horas de LKS NEXT

Esta aplicación está desarrollada en Java por el equipo de movilidad de LKS NEXT y se utiliza para la gestión e imputación de horas. Sigue una arquitectura MVVM (Model View ViewModel) [2]. Cuenta con cuatro pantallas principales, como se puede ver en la Figura 3.1:

Figura 3.1: Fragmento de captura de aplicación de imputación de horas de LKS NEXT

- **Imputaciones:** En este apartado el usuario puede imputar cada día las horas que haya trabajado indicando en qué proyecto ha estado trabajando. Se puede navegar entre las diferentes semanas y entre los días de cada semana, en los que aparece lo previamente imputado y un resumen de las horas imputadas de cada semana en la parte inferior de la pantalla.
- **Resumen:** En este apartado el usuario puede ver un resumen de sus horas imputadas de cada mes, en las que aparecen a qué proyecto han sido imputadas.
- **Mis proyectos:** En esta pantalla aparecen los proyectos y códigos asociados en los que está trabajando actualmente el usuario. Incluye un buscador con el que buscar entre los diferentes proyectos.
- **Vacaciones:** En este apartado aparecen las horas de las que dispone el usuario para tomárselas como vacaciones. Aparecen las horas totales que le corresponden, las consumidas y las disponibles. Además, aparecen las horas a compensar.

Es una aplicación real que se utiliza actualmente por varios trabajadores de la empresa, por lo que genera datos reales en Crashlytics y Performance Monitoring del uso real de los usuarios, siendo muy útil para ver los datos que genera realmente una aplicación en producción de estas herramientas.

Además, cuenta con pruebas instrumentadas ya desarrolladas con Cucumber, UiAutomator y Espresso, lo que será de gran utilidad para probar Test Lab, ya que ya hay varias pruebas que se pueden ejecutar sin necesidad de tener que crearlas desde cero. Entre ellas,

encontramos tests que pruebas el inicio de sesión, que el resumen de las horas mensuales funcione correctamente con varios escenarios o que los proyectos y vacaciones que corresponden a cada usuario sean los que deberían ser.

Por último, como se ha explicado anteriormente, es una aplicación con determinada complejidad, lo que permite también utilizarla para las pruebas Robo de Test Lab.

Debido a esto, se decidió utilizar esta aplicación para las pruebas ya que los resultados y conclusiones deberían ser extrapolables a los resultados que se obtendrían con otro tipo de aplicaciones reales.

CAPÍTULO 4

Crashlytics

En esta primera iteración se estudiará y analizará en profundidad la herramienta Crashlytics, una herramienta para detectar e informar sobre fallos y errores recuperables para hacer un seguimiento de la calidad y estabilidad de la aplicación.

Con el uso de esta herramienta, la detección de fallos cuando la aplicación se encuentra en producción es mucho más rápida y eficaz, ya que los errores se detectan y registran de manera automática. La rápida detección de estos errores sería de gran utilidad en la empresa, posibilitando solucionarlos en versiones posteriores de las aplicaciones que desarrollan.

Además, se pretende definir una forma de manipulación y exportación de los datos, para poder visualizar de manera clara los datos más importantes para el equipo de calidad de LKS NEXT y poder presentarlo a clientes o comunicarlo a los desarrolladores para la solución de los errores.

Para comenzar a utilizar la herramienta, basta con modificar los archivos Gradle[1] del proyecto como se indica en la documentación [3]. Una vez realizados los pasos que se indican, los errores empezarán a registrarse.

En los siguientes apartados se analizará la herramienta en profundidad, se explicarán las opciones que existen para personalizar los registros que crea, su integración con Android Studio y el coste de la misma. Además, se explicará cómo manipular y exportar los datos utilizando las herramientas necesarias para ello.

4.1. Análisis general

Después de configurar Crashlytics en un proyecto móvil, cada vez que la aplicación tenga un problema, la herramienta lo registrará en tiempo real, posibilitando hacer un seguimiento de la estabilidad de la aplicación. De esta manera, la detección, priorización y corrección de los mismos será más rápida y eficiente, ya que agrupa los errores y ofrece información sobre cada uno de ellos como su traza, dispositivo, versión de la aplicación, versión de Android, número de eventos, número de usuarios afectados y fecha del mismo. En la Figura 4.1 se puede ver un ejemplo de crash en el que se muestran estos detalles.

[1]Gradle: https://gradle.org

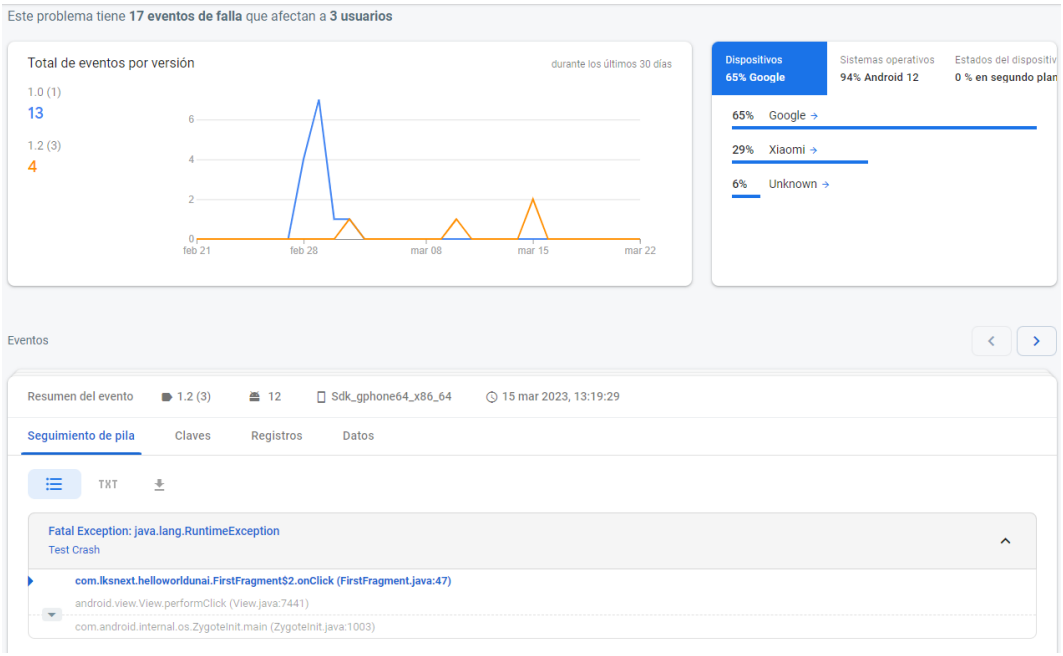


Figura 4.1: Ejemplo de un error en Crashlytics y sus detalles

Además, se ofrecen otro tipo de estadísticas durante un periodo de tiempo concreto, como usuarios que no experimentaron bloqueos o errores durante ese periodo. En la Figura 4.2 se puede ver un ejemplo de estas estadísticas durante los últimos 30 días.

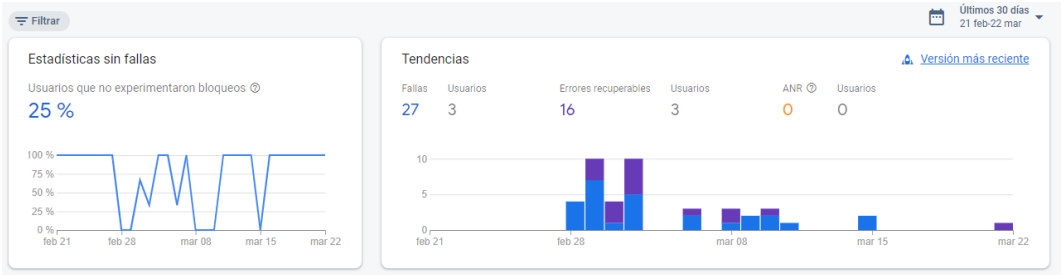


Figura 4.2: Ejemplo de estadísticas de Crashlytics durante los últimos 30 días

Los errores se clasifican en tres tipos:

- **Crash:** Un crash (o fallo) ocurre cuando una aplicación se detiene inesperadamente debido a un error. Esto puede ser causado por un error en el código, una falta de memoria del dispositivo o un problema con el sistema operativo.
- **Error recuperable:** Ocurre cuando la aplicación encuentra un problema, pero puede recuperarse y continuar funcionando sin cerrarse por completo. Normalmente, es una excepción que salta dentro de un bloque try-catch y se trata, siendo un fallo que no provoca que la aplicación se bloquee, sino que simplemente hace que no funcione correctamente.
- **Aplicación No Responde (ANR):** Ocurre cuando la aplicación se bloquea temporalmente y deja de responder a las interacciones del usuario. Esto puede ocurrir por

múltiples causas, como una tarea intensiva de procesamiento o un bloqueo en la interfaz de usuario.

Cuando ocurre un crash o error recuperable, se genera un informe que incluye toda la información relativa al estado del dispositivo y de la aplicación, para que sea fácilmente identificable por los desarrolladores.

Sin embargo, cuando ocurre un error de tipo ANR, se genera un informe que incluye información sobre el tiempo que la aplicación tardó en responder y el estado de la aplicación en el momento en que se produjo el ANR, además de etiquetar todos los subprocesos de la aplicación en ese momento para identificar la causa más fácilmente.

Hemos visto el tipo de información que nos proporciona Crashlytics por defecto, sin embargo, es posible que nos interese información más precisa sobre el estado concreto de la aplicación o el usuario que ha sufrido ese error, lo cual no hace por defecto. La siguiente sección nos indica cómo se pueden personalizar los informes para añadir este tipo de información.

4.2. Personalización de los informes

En Crashlytics existen cuatro mecanismos para añadir información a los reportes de fallos que se capturan. Para ilustrar de manera más clara estas opciones, en los ejemplos se hablará de una aplicación de compras:

- **Claves personalizadas:** Las claves personalizadas o custom keys se utilizan para obtener el estado específico de la aplicación hasta o en el momento del error. Por ejemplo, en una aplicación de compras, en el código de la pantalla final, el Checkout, se añadiría el fragmento de código de la Figura 4.1. De esta manera, si ocurre un error en está pantalla, quedaría registrado que ha sido en esa pantalla concreta, y se podrían filtrar los errores por claves, obteniendo los errores de una pantalla específicamente.

```
crashlytics = FirebaseCrashlytics.getInstance();
crashlytics.setCustomKey("Fragmento", "Checkout" );
```

Código 4.1: Ejemplo de clave-valor en Java para una aplicación de compras

  Se definen con una clave, que siempre será un string, y un valor asociado a ella, que puede ser cualquier tipo primitivo. Este valor puede ser actualizado posteriormente. El par clave-valor se guardará cuando se ejecute la instrucción con el método `setCustomKey`, como se ha visto anteriormente.

  Además de registrar el estado específico en el que ocurre el error, si a lo largo de código se han definido otras claves y posteriormente ocurre un error, en el informe de este quedarán registradas todos los pares clave-valor con claves diferentes, pudiendo ver todos los estados por los que ha pasado la aplicación, es decir, se podría ver la traza de la ejecución hasta que se haya producido el error.

  Estos pares clave-valor se pueden consultar en el apartado “Claves” de Crashlytics.

- **Mensajes personalizados:** Para dar más contexto sobre los fallos, también se pueden agregar mensajes de registro personalizados. Estos fallos se registran utilizando `log`,

como se ve en el ejemplo 4.2, y luego se muestran en el apartado "Registros" del crash.

En este ejemplo, se agrega un mensaje personalizado para identificar la acción que ha provocado el error, en este caso añadir un producto al carrito.

```
crashlytics = FirebaseCrashlytics.getInstance();
crashlytics.log("Anadir al carrito ha fallado");
```

Código 4.2: Ejemplo de mensaje personalizado en Java para aplicación de compras

- **Identificadores de usuarios:** Para diagnosticar determinados problemas, resulta útil identificar a los usuarios que los han sufrido. Para ello, se puede asociar un identificador único a cada usuario en formato ID, token o valor de hash utilizando `setUserId`.

  Cuando el usuario haga login, se le asignaría un valor como se puede ver en el ejemplo 4.3. En este caso, al hacer login, se le asociaría al usuario su username.

```
crashlytics = FirebaseCrashlytics.getInstance();
crashlytics.setUserId(username);
```

Código 4.3: Ejemplo de asignación de token a usuario al hacer login en aplicación de compras

  Es importante que se le asigne un valor que permita mantener el anonimato del usuario pero al mismo tiempo identificar los errores asociados a él.

- **Captura de excepciones recuperables:** Además de los crashes que hacen que la aplicación se cierre o deje de funcionar, Crashlytics permite registrar las excepciones recuperables, es decir, excepciones que se producen pero se capturan y tratan con un try-catch. Se puede ver un ejemplo en 4.4.

```
try{
    //codigo que hace que se produzca una excepcion
}catch (Exception e){
    FirebaseCrashlytics.getInstance().recordException(e);
    //tratar la excepcion
}
```

Código 4.4: Ejemplo mensaje personalizado en Java

  Estás excepciones se registrarán en Crashlytics de la misma manera que los crashes, pero se indicará que son de tipo recuperable.

Todas estas opciones de personalización son muy interesantes y pueden aportar gran valor a los informes, pero para sacar el máximo rendimiento a estos informes que genera Crashlytics, se puede integrar en el entorno de desarrollo como se describe en el siguiente apartado.

4.3. Integración con Android Studio

Android Studio[2] ofrece la posibilidad de ver los datos que proporciona Crashlytics y trabajar con ellos directamente desde la ventana de *App Quality Insights* [4]. De esta

[2] Android Studio: https://developer.android.com/studio

manera, no hace falta revisar los registros desde el navegador, encontrarlos en el código y arreglarlos, ya que desde el propio Android Studio se podrán ver estos datos y se indicará en el propio código dónde se produce cada error, haciendo esta tarea mucho más rápida y efectiva.

La configuración es muy simple, al iniciar sesión con la cuenta de desarrollador en el IDE, los datos de *Issues*, *Sample Stack Trace* y *Details* aparecerán en la ventana de *App Quality Insigts*.

También se puede ver el seguimiento de pila de los problemas principales, y saltar a las líneas relevantes del código haciendo click en él. Además se pueden agrupar estos problemas por dispositivo, versión de Android, gravedad, fecha y versión de la aplicación.

En la Figura 4.3 se puede ver un ejemplo de la ventana App Quality Insights en Android Studio.

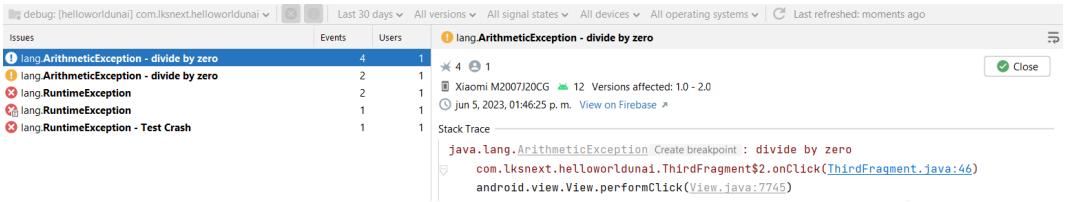


Figura 4.3: Ejemplo de ventana App Quality Insights en Android Studio

Después de analizar la herramienta y las posibilidades que ofrece en profundidad, se quiere ver cómo exportar y manipular los datos que se han conseguido. Para ello, en la siguiente sección se explica cómo hacerlo utilizando BigQuery.

4.4. Exportación y manipulación de datos con BigQuery

Como se ha comentado anteriormente, uno de los objetivos de este proyecto es manipular y exportar los datos que proporciona Crashlytics. Firebase no dispone de API, por lo que la manera en la que se exportan estos datos para su posterior manipulación es utilizando su integración con BigQuery[3].

4.4.1. BigQuery

BigQuery es un servicio de análisis de datos en la nube desarrollado por Google e integrado en Google Cloud[4] que permite a los usuarios realizar consultas SQL sobre grandes conjuntos de datos de manera rápida y eficiente. También ofrece una serie de características y herramientas avanzadas para mejorar la eficiencia del análisis de datos. En concreto, proporciona integración con otras herramientas de Google y servicios externos, lo que facilita el intercambio de datos. Además, BigQuery también admite la importación y visualización de datos en tiempo real, lo que permite a los usuarios monitorizar y analizar el rendimiento de sus aplicaciones y servicios en tiempo real.

[3]BigQuery: https://cloud.google.com/bigquery
[4]Google Cloud: https://cloud.google.com

Esta herramienta es de pago, aunque dispone de una versión de prueba la cual se utilizará en este proyecto, ya que sus límites son más que suficientes para los datos que se quieren almacenar y las consultas que se quieren realizar.

Las exportaciones en tiempo real no están permitidas en esta versión de prueba, pero no son necesarias, ya que en la empresa la exportación de datos se quiere hacer mensualmente.

4.4.2. Integración de BigQuery y Crashlytics

Para configurar la exportación de datos, se ha realizado una guía como documentación interna para la empresa en la que se explican detalladamente los pasos a seguir en cualquier proyecto de Firebase para que los datos se exporten correctamente. Esta guía se puede encontrar en el Anexo A.

Se divide en tres partes:

- **Configuración inicial de la exportación:** En este apartado se explica cómo activar la exportación de los reportes que genera Crashlytics a BigQuery.

  Para ello, simplemente hay que vincular BigQuery en la pestaña de Integraciones del proyecto. En esta página hay que seleccionar todas las herramientas de Firebase que se quieren exportar a BigQuery. En este caso, las herramientas a seleccionar son Google Analytics, ya que hacen falta para calcular algunas de las estadísticas que se quieren exportar, y Crashlytics.

  Es importante que para ambas herramientas se escoja la región “us”, ya que para poder hacer consultas sobre varias tablas a la vez estas tienen que estar en la misma región y la tabla de Crashlytics solo puede estar en la región de Estados Unidos. También se debe seleccionar que la exportación sea diaria.

  Una vez realizados estos pasos, en la consola de BigQuery aparecerán los conjuntos de datos correspondientes. Además, en la pestaña “Transferencia de datos” aparecerá la transferencia "Firebase Crashlytics Export". Es posible que tarden 24 horas en aparecer, ya que la exportación se hace diariamente.

- **Solución si la exportación no funciona correctamente:** Si pasadas las 24 horas horas alguno de los conjuntos de datos no se han creado, hay que revisar la pestaña de actividad de Google Cloud para encontrar posibles errores. Si se encuentran errores de permisos, es necesario otorgar funciones de editor a la cuenta de servicio que se encarga de crear los conjuntos de datos.

  Cada uno de los conjuntos de datos lo crea una cuenta de servicio diferente, por lo que si alguna cuenta no tiene los permisos suficientes, se deben de otorgar manualmente.

- **Reabastecimiento de datos anteriores:** Al activar la exportación, los errores comenzarán a exportarse de ese momento en adelante, pero a veces es interesante exportar también los errores ocurridos anteriormente a activar la exportación, si el proyecto tiene ya cierta antigüedad. Para ello, BigQuery permite hacer un reabastecimiento de los reportes generados anteriormente a la activación de la exportación.

  Para realizarlo, en la pestaña “Transferencias de datos” de BigQuery, dentro del flujo de datos de Crashlytics, se programa un nuevo reabastecimiento. Se escogen la fecha

inicio y fecha fin del reabastecimiento y en la próxima ejecución se exportarán los reportes de crashes creados durante esas fechas.

Es importante recalcar que se debe poner como fecha fin siempre fechas anteriores a la actual, ya que, si no, se pueden exportar datos de un mismo día varias veces y después encontrarlos duplicados.

También se explican los límites de uso de la versión de prueba, que se explican en detalle en el apartado de Costes de este mismo capítulo.

Para aplicar esta guía, la configuración se debe hacer desde perfiles de propietario o administrador del proyecto en Firebase. LKS NEXT ha desarrollado aplicaciones móviles para otros clientes que se encuentran en producción actualmente y son estos clientes los que tienen acceso a este tipo de cuentas. Es por esto por lo que se ha creado una guía detallada para configurar la exportación, para que se pueda mandar a clientes. La guía se ha mandado a varios clientes reales de LKS NEXT y estos han realizado la configuración satisfactoriamente.

4.4.3. Diseño de manipulación de datos

Para diseñar la forma de manipular y exportar los datos se ha hecho uso de los datos que se exportan a BigQuery, creando diferentes consultas para obtener los datos más relevantes teniendo en cuenta la forma de trabajo de la empresa. De esta manera, solo hay que analizar los datos relevantes y se hace de una manera mucho más rápida y eficiente, ya que se evita tener que navegar por los menús de Firebase, que presentan mucha información y a menudo se hace complicado extraer la información realmente importante para la empresa.

También es importante comentar que, debido a la sensibilidad de los datos de los clientes con los que trabaja LKS NEXT y con los que se quiere trabajar eventualmente, todas las consultas se han diseñado sobre un proyecto de prueba creado por el alumno, dificultando esto considerablemente su tarea, ya que a menudo no se disponían de datos suficientes para probar las consultas que se preparaban.

Principalmente, se han preparado dos consultas SQL, que se pueden dejar guardadas en los proyectos para que solo haya que ejecutarlas para obtener la información:

- **Errores con más de diez eventos en los últimos treinta días:** En el ejemplo 4.5 se ve un ejemplo de la consulta para un proyecto de prueba.

```
DECLARE project STRING DEFAULT 'com.lksnext.
    HelloWorldUnai';

SELECT
    MAX(blame_frame.file) AS File,
    MAX(blame_frame.line) AS Line,
    MAX(error_type) AS Tipo,
    CONCAT("https://console.firebase.google.com/u/0/
        project/helloworldunai-4957e/crashlytics/app/
        android:",project,"/issues/",issue_id) as Link,
    MIN(application.display_version) AS Min_version,
    MAX(application.display_version) AS Max_version,
    COUNT(event_id) AS Eventos,
```

```
    COUNT(DISTINCT installation_uuid) AS Usuarios
FROM
    `helloworldunai-4957e.firebase_crashlytics.
        com_lksnext_helloworldunai_ANDROID`
WHERE
    UPPER(bundle_identifier)=UPPER(project)
    AND CAST(event_timestamp AS DATE ) >= CURRENT_DATE()
        - 30
GROUP BY
    issue_id
HAVING
    Eventos > 10
ORDER BY
    eventos DESC
```

Código 4.5: Consulta para obtener errores con más de diez eventos en los últimos 30 días

Una vez ejecutada la consulta, se obtiene una tabla que se puede exportar como archivo csv como se ve en el ejemplo 4.1, en la que aparece el archivo y línea en la que ocurre el error, el tipo de error, el enlace al error en Crashlytics, la versión mínima y la versión máxima en la que ocurre, el número de eventos de ese error y el número de usuarios de una manera muy clara.

File	Line	Enlace	Min_version	Max_version	Eventos	Usuarios
FirstFragment.java	47	https://console.fireb	1.0	1.2	34	6
SecondFragment.java	55	https://console.fireb	1.0	1.2	22	4
ThirdFragment.java	56	https://console.fireb	1.0	1.2	19	3
ThirdFragment.java	46	https://console.fireb	1.0	1.2	14	2
SecondFragment.java	65	https://console.fireb	1.0	1.2	11	2

Tabla 4.1: Ejemplo de tabla de problemas con más de diez eventos en los últimos treinta días

- **Usuarios y problemas en los últimos treinta días:** En el ejemplo 4.6 se ve un ejemplo de la consulta para un proyecto de prueba.

```
SELECT
    app_info.id AS Aplicacion,
    COUNT(DISTINCT user_pseudo_id) AS UsuariosA30Dias,
    1-(COUNT(DISTINCT installation_uuid)/COUNT(DISTINCT
        user_pseudo_id)) AS UsuariosSinFallosA30Dias,
    COUNT(DISTINCT issue_id) AS ProblemasA30Dias
FROM
    `helloworldunai-4957e.analytics_355776645.*` as
        Events FULL JOIN `helloworldunai-4957e.
        firebase_crashlytics.
        com_lksnext_helloworldunai_ANDROID` as Crashes ON
        Events.app_info.id=Crashes.bundle_identifier
WHERE
    CAST(event_date AS DATE FORMAT 'YYYYMMDD') >= (
        CURRENT_DATE() - 30) AND
    CAST(Crashes.event_timestamp AS DATE ) >= (
        CURRENT_DATE() - 30) AND
    app_info.version = "1.2" AND
```

```
        application.display_version = app_info.version AND
        bundle_identifier='com.lksnext.helloworldunai'
    GROUP BY
        app_info.id
```

Código 4.6: Consulta para obtener usuarios y problemas en los últimos 30 días

Una vez ejecutada la consulta, se obtiene una tabla que se puede exportar también como archivo csv, como se ve en el ejemplo 4.2, en la que aparece el id de la aplicación y el número de usuarios, errores y usuarios sin errores durante los últimos treinta días.

Aplicacion	UsuariosA30Dias	UsuariosSinFallosA30Dias	ProblemasA30Dias
com.lksnext.helloworldunai	78	0.76	16

Tabla 4.2: Ejemplo de tabla de usuarios y problemas en los últimos treinta días

Además de estos datos, se ha detectado la necesidad de llevar un control de los problemas que han sido revisados y qué se ha decidido realizar con los mismos, ya que muchas veces no se pueden o no interesa solucionarlos, bien porque sea un error controlado que Crashlytics detecta como error pero no lo es, o porque sea una librería la que utiliza ese error y no se pueda actualizar. Para ello, se hace uso de las notas. El principal problema que se ha encontrado es que las notas no se exportan a BigQuery, lo que hace que a estas tablas les falte información.

Una vez identificada esta necesidad, el alumno se ha puesto en contacto con el soporte de Firebase para exponer este problema. Firebase mostró interés en la forma de trabajo que se había planteado por parte del alumno por el que se proponía que las notas fuesen exportadas a BigQuery y se abrió una petición interna para agregar esta funcionalidad, aunque a 20/06/2023 aún no se ha implementado.

4.5. Costes

LKS NEXT trabaja con diferentes clientes que se podrían beneficiar de las posibilidades que ofrece Crashlytics. Por tanto, es importante realizar un análisis de los costes que supone de cara a hacerles una propuesta y empezar a usar la herramienta en sus proyectos.

El uso de Crashlytics es totalmente gratuito para cualquier aplicación. Por otro lado, y como se ha comentado anteriormente, para la manipulación y exportación de datos que se ha diseñado, es suficiente con la versión de prueba *Sandbox* de BigQuery, que es totalmente gratuita. Los límites son:

- 10 GB de almacenamiento y 1 TB de datos de consulta procesados por mes.
- Todos los conjuntos de datos tienen un tiempo de vencimiento de 60 días.

Por lo tanto, el coste de uso de estas herramientas, la manipulación de sus datos y exportación de los mismos sería gratuito.

Si en algún momento se trabajase con un proyecto de gran envergadura que generase una gran cantidad de reportes y se sobrepasaran los límites anteriormente mencionados, habría que configurar la facturación.

En ese caso, los costes serían los siguientes:

- 0.02$ por GB almacenado si los datos se han modificado en los últimos 90 días.
- 0.01$ por GB almacenado si los datos no se han modificado en los últimos 90 días.
- 5$ por TB de datos de consulta.

Además, los conjuntos de datos no vencerían automáticamente.

CAPÍTULO 5

Performance Monitoring

En esta segunda iteración se estudiará y analizará en profundidad la herramienta Performance Monitoring, una herramienta para detectar e informar sobre aspectos de rendimiento de la aplicación.

El uso de esta herramienta brinda una mayor eficacia y rapidez en la detección de posibles problemas en la aplicación en producción, ya que genera estadísticas de rendimiento de las que, si se observa una variación en una métrica, se pueden deducir posibles causas que la hayan provocado, como una actualización, y solucionarlas si fuese necesario. La detección temprana de estos problemas sería de gran utilidad en la empresa, haciendo posible solucionarlos lo antes posible.

Para comenzar a utilizar la herramienta, basta con modificar los archivos Gradle del proyecto como se indica en la documentación [5]. Una vez realizados los pasos que se indican, las métricas básicas comenzarán a registrarse.

En los siguientes apartados se analizará la herramienta en profundidad, se explicarán las métricas que se recogen automáticamente y las que se pueden recoger sobre código específico, las métricas más importantes, la exportación de datos con BigQuery y el coste de la misma.

5.1. Análisis general

El rendimiento de una aplicación es un factor determinante en la experiencia del usuario. Un bajo rendimiento puede generar una percepción negativa, lo que conduce a una disminución en el uso y una mayor probabilidad de búsqueda de alternativas. Por otro lado, un buen rendimiento puede generar una percepción positiva y aumentar la satisfacción y el uso de la aplicación.

Performance Monitoring es una herramienta de Firebase que permite a los desarrolladores medir y monitorizar el rendimiento de sus aplicaciones móviles en tiempo real. Esta herramienta brinda información detallada sobre tiempos de respuesta, renderización, solicitudes de red y otros aspectos importantes de rendimiento de la aplicación.

Estas métricas permiten a los desarrolladores y al equipo de calidad identificar los

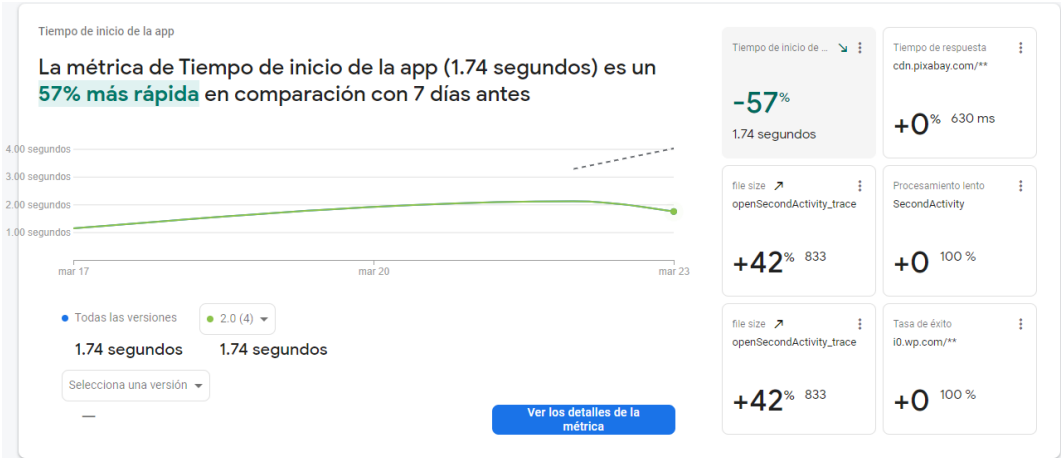


Figura 5.1: Ejemplo de un panel en Firebase Performance

problemas de rendimiento, mejorando así la experiencia del usuario. En el siguiente capítulo se especifican cuáles son todas las métricas que se recogen de manera automática y cómo se pueden recoger sobre código específico para poder analizarlas posteriormente.

5.2. Datos y métricas recopiladas

Esta herramienta recopila una gran cantidad de métricas, tanto automáticamente como especificadas por el desarrollador en un fragmento de código. Las métricas que se consideren más importantes para el proyecto se pueden agregar al panel principal de Firebase, para verlas más fácilmente y observar gráficas por versiones y tiempo sobre ellas, como se puede ver en la Figura 5.1.

El panel muestra los datos de métricas recopilados en el tiempo, de forma gráfica y como cambio porcentual. De esta manera, destacan mucho más los cambios en las métricas, lo que facilita abordar y minimizar con rapidez el impacto de los problemas de rendimiento.

Es importante recalcar que estas métricas se miden en cada dispositivo, y pueden cambiar dependiendo de sus características. Es decir, si la aplicación se ejecuta en un dispositivo con pocos recursos, es posible que veamos que el tiempo de inicio es alto o que hay varias pantallas congeladas, lo que no tiene que significar que la aplicación funcione mal necesariamente si obtenemos mejores métricas en dispositivos con más recursos. Por esto, es recomendable tener seleccionado el percentil en 90 para ver una representación global de los datos, ya que si se escoge un percentil más bajo, es posible que los datos se vean afectados por las métricas que recogen estos dispositivos con pocos recursos, cuando no deberían afectar para la visión general del rendimiento de la aplicación.

5.2.1. Métricas recopiladas automáticamente

Esta herramienta recoge una gran cantidad de métricas de manera automática, que se dividen en tres grandes grupos:

- **Inicio de la app, primer plano y segundo plano:** Se recopila automáticamente el tiempo que tarda la aplicación en iniciarse, y el tiempo que está en primer y segundo

plano durante la ejecución de la misma.

- **Renderizaciones de pantallas:** También se realiza un seguimiento del tiempo que tarda en renderizarse por completo cada pantalla, para identificar cuales tienen mal rendimiento. Dependiendo del tiempo que tarden en renderizarse por completo, se clasifican en dos grupos:
  - *Fotogramas de renderización lenta:* Porcentaje de instancias de pantalla que tardan más de 16 ms en renderizarse.
  - *Fotogramas congelados:* Porcentaje de instancias de pantalla que tardan más de 700 ms en renderizarse.
- **Solicitudes de red HTTP/S:** Se recogen las siguientes métricas por cada solicitud de red HHTP o HTTPS que realice la aplicación, desde que se envía hasta que se completa:
  - *Tiempo de respuesta:* Tiempo que transcurre desde que se envía la solicitud de red a un servidor hasta que se recibe la respuesta, como una solicitud de una imagen o una petición a una API.
  - *Tamaño de la carga útil de la respuesta:* Tamaño en bytes de los datos que recibe la aplicación durante la solicitud de red. Por ejemplo, si recibe un archivo JSON, el tamaño del archivo.
  - *Tamaño de la carga útil de la solicitud:* Tamaño en bytes de los datos que envía la aplicación durante la solicitud de red. Por ejemplo, si envía un archivo JSON, el tamaño del archivo.
  - *Tasa de éxito:* Porcentaje de respuestas correctas en comparación con el total de respuestas. Para determinar si una solicitud ha sido exitosa, se utilizan los códigos de respuesta del protocolo HTTP. Son enviados por el servidor como parte de la respuesta y están divididos en diferentes rangos. Si el código está entre 100 y 399, se considera una respuesta correcta, y si se encuentra entre 400 y 599, se considera una respuesta errónea. Si se esperasen códigos diferentes como correctos, se puede indicar.

  Todas las solicitudes realizadas por la aplicación se agrupan por patrones URL dependiendo de la cantidad de solicitudes que haga la misma, posibilitando detectar posibles fallos de las consultas más fácilmente. Estas agrupaciones se realizan de manera automática, aunque también se pueden especificar patrones personalizados.

5.2.2. Métricas recopiladas sobre código específico

A menudo, es interesante recopilar métricas sobre un método o fragmento específico de código de nuestra aplicación, bien porque sea una carga de datos importantes, una consulta o modificación en la base de datos, o por cualquier otra razón.

Este seguimiento personalizado se puede hacer de dos maneras:

- **Seguimiento personalizado de un método:** Para recoger métricas sobre un método, simplemente hay que añadir la anotación `@AddTrace` en su cabecera, en la que hay que indicar el nombre de la traza para identificarla posteriormente en

la consola de Firebase. Por ejemplo, en una aplicación de compras con un método que carga la lista de productos, para recoger el tiempo que tarda en cargarse la lista, añadiríamos la siguiente cabecera al método, como se ve en el ejemplo 5.1.

```
@AddTrace(name = "cargarLista", enabled = true)
public void cargarListaProductos () {...}
```

Código 5.1: Ejemplo de anotación para seguimiento personalizado de un método en una aplicación de compras

- **Seguimiento personalizado de un fragmento de código:** Para recoger métricas sobre un fragmento de código específico, hay que utilizar el API de Trace[1] para crear la traza, indicando su nombre, y el lugar del código en el que inicia y termina el seguimiento. Por ejemplo, para una aplicación de compras en la que se quiere monitorizar el tiempo que tarda la aplicación en cargar el detalle de un producto al hacerle click, se añadiría el siguiente Código 5.2.

```
import com.google.firebase.perf.metrics.Trace;

Trace myTrace = FirebasePerformance.getInstance().
    newTrace("producto_detalle");
myTrace.start();
//codigo para mostrar detalle de producto.
myTrace.stop();
```

Código 5.2: Ejemplo de seguimiento personalizado de un fragmento de código para una aplicación de compras

Sobre estas trazas, además del tiempo, que se recoge automáticamente, se pueden registrar métricas y atributos personalizados:

- **Métricas personalizadas:** A lo largo del código, se puede incrementar una métrica con el método `incrementMetric`. En la aplicación de compras, en el método que se encarga de cargar la lista con todos los productos, se podría utilizar para recoger la cantidad de productos que se cargan, como se puede ver en el ejemplo 5.3.

```
myTrace.incrementMetric("productos_cargados", 1);
```

Código 5.3: Ejemplo de incremento de métrica personalizada en aplicación de compras

- **Atributos personalizados:** Es posible añadir atributos personalizados a las trazas con el método `putAttribute`, además de los que recoge Performance Monitoring de manera automática. Por ejemplo, en la aplicación de compras, en el trozo de código encargado de cargar el detalle de cada producto, se podría añadir el ID del producto como atributo, como se ve en el ejemplo 5.4.

```
myTrace.putAttribute("producto_id", product.getId());
```

Código 5.4: Ejemplo de atributo personalizado en aplicación de compras

[1] Trace API: https://firebase.google.com/docs/reference/android/com/google/firebase/perf/metrics/Trace

5.2.3. Atributos de las métricas

Además de esto, todas las métricas tienen atributos asociados a las mismas por defecto. Después de pulsar en una métrica, los datos de esta se pueden filtrar por versión de la aplicación, país, versión de Android, dispositivo o proveedor, que son los atributos que se recopilan automáticamente por cada sesión, como se puede ver en el ejemplo 5.2.

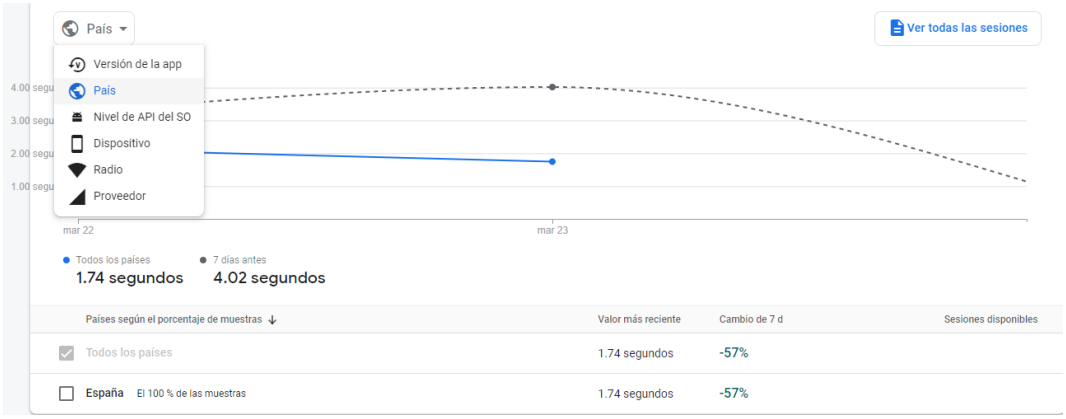


Figura 5.2: Ejemplo de filtrado por atributos en Firebase Performance

Ahora que conocemos todas las métricas que recoge Performance Monitoring de manera automática y personalizada, es interesante conocer cuales de estas son verdaderamente relevantes para nuestro proyecto, para centrarnos en estas y monitorizarlas de cerca.

5.3. Métricas a tener en cuenta

Dependiendo del proyecto, las métricas a tener en cuenta pueden variar, por lo que se debería analizar cada proyecto y seleccionar las métricas más importantes para el mismo.

En general, las métricas que siempre se deberían monitorizar son las siguientes:

- **Tiempo de inicio:** El tiempo de inicio de la aplicación es vital para saber si algún cambio introducido recientemente está provocando que la aplicación se inicie de manera más lenta y poder solucionarlo.
- **Pantallas congeladas:** Gracias a esta métrica podemos conocer problemas con alguna pantalla en concreto. Dependiendo del tipo de aplicación, conviene hacer el seguimiento sobre unas pantallas u otras. Por ejemplo, en una aplicación de compras convendría hacer un seguimiento de la pantalla final, ya que si esta es muy lenta puede provocar que el usuario abandone la compra.
- **Tiempo de respuestas de solicitudes de red (APIs):** Está es usualmente la métrica más importante, ya que indica posibles problemas con los APIs que utilicemos y si las peticiones se están realizando de manera satisfactoria y en un tiempo razonable.

Con la monitorización de estas métricas, la detección de posibles problemas es mucho más eficaz, ya que se pueden detectar problemas en estas áreas para un país o dispositivo concreto gracias a los atributos de cada sesión.

Después de analizar la herramienta y todas las posibilidades que ofrece en profundidad, se quiere ver como exportar y manipular estas métricas. Para ello, en la siguiente sección se explican los pasos a seguir.

5.4. Exportación de datos con BigQuery

Todas estas métricas pueden ser exportadas a BigQuery siguiendo el mismo procedimiento que se explica en el capítulo de Crashlytics, lo que haría posible obtener varias tablas en la que aparezcan datos sobre las métricas más importantes en los últimos treinta días para poder llevar un seguimiento de estas métricas de manera más sencilla.

Se han propuesto tres consultas SQL, que se pueden guardar en el proyecto para obtener las tablas más rápidamente en posteriores ejecuciones:

- **Media de tiempo de inicio por versión de la aplicación:** En el Código 5.5 se puede ver un ejemplo de la consulta para un proyecto de prueba.

```
SELECT
    app_display_version AS Version,
    AVG(trace_info.duration_us)/1000000 AS Segundos
FROM
    `helloworldunai-4957e.firebase_performance.
       com_lksnext_HelloWorldUnai_ANDROID`
WHERE
    CAST(event_timestamp AS DATE ) >= CURRENT_DATE() - 30
        AND
    event_name = "_app_start"
GROUP BY
    app_display_version
```

Código 5.5: Consulta para obtener la media de tiempo de inicio por versión

Una vez ejecutada la consulta, se obtiene una tabla que se puede exportar como archivo csv como se ve en el ejemplo 5.1, en la que aparece la versión y el tiempo en segundos que ha tardado en iniciarse la aplicación de media durante los últimos 30 días.

Version	Segundos
2.0	2,29
2.1	3,73
3.9	1,04

Tabla 5.1: Ejemplo de tabla de tiempo de inicio por versión

- **Porcentaje de pantallas congeladas por versión de la aplicación:** En el Código 5.6 se puede ver un ejemplo de la consulta para un proyecto de prueba.

```
SELECT
    AVG(trace_info.duration_us / 1000000) AS
       SegundosEnPantalla,
    AVG(trace_info.screen_info.frozen_frame_ratio) AS
       PorcentajePantallasCongeladas,
```

```
    event_name AS Pantalla,
    app_display_version AS Version
FROM
    `helloworldunai-4957e.firebase_performance.
        com_lksnext_HelloWorldUnai_ANDROID`
WHERE
    CAST(event_timestamp AS DATE ) >= CURRENT_DATE() - 30
    AND event_type = "SCREEN_TRACE"
GROUP BY
    event_name,
    app_display_version
ORDER BY
    event_name,
    app_display_version;
```

Código 5.6: Consulta para obtener el porcentaje de pantallas congeladas por versión

Una vez ejecutada la consulta, se obtiene una tabla que se puede exportar como archivo csv como se ve en el ejemplo 5.2, en la que aparece de media durante los últimos 30 días, el tiempo que ha estado el usuario en la pantalla, el porcentaje de pantallas congeladas, el nombre de la pantalla y la versión de la aplicación.

SegundosEnPantalla	PorcentajePantallasCongeladas	Pantalla	Version
4,4	0	_st_MainActivity	2.0
16,5	0.018	_st_SecondActivity	2.0
4,9	0	_st_MainActivity	2.2
19,0	0.106	_st_SecondActivity	2.2

Tabla 5.2: Ejemplo de tabla de porcentaje de pantallas congeladas por versión

- **Tiempo de respuesta de APIs:** En el Código 5.7 se puede ver un ejemplo de la consulta para un proyecto de prueba.

```
SELECT
    event_name AS Consulta,
    AVG(network_info.response_completed_time_us -
        network_info.response_initiated_time_us)/100000 AS
         TiempoDeRespuestaSegundos
FROM
    `helloworldunai-4957e.firebase_performance.
        com_lksnext_HelloWorldUnai_ANDROID`
WHERE
    CAST(event_timestamp AS DATE ) >= CURRENT_DATE() - 30
    AND event_type = "NETWORK_REQUEST"
GROUP BY
    event_name
```

Código 5.7: Consulta para obtener el tiempo de respuesta de las APIs con las que se comunica la aplicación

Una vez ejecutada la consulta, se obtiene una tabla que se puede exportar como archivo csv como se ve en el ejemplo 5.3, en la que aparece el patrón sobre el que se ha hecho la consulta y el tiempo de respuesta en segundos.

Consulta	TiempoDeRespuestaSegundos
i0.wp.com/**	0,290
google.com/search/**	0,631
google.com/images/**	0,528

Tabla 5.3: Ejemplo de tabla de tiempo de respuesta de APIs

En el siguiente apartado se detallan los costes que tendría el uso de esta herramienta.

5.5. Costes

Debido a que LKS NEXT trabaja con varios clientes, es de gran importancia conocer los posibles costes que tendría el uso de esta herramienta antes de ofrecerla a dichos clientes.

El uso de Performance Monitoring es totalmente gratuito para cualquier aplicación. Si se quiere activar la exportación a BigQuery, habrá que tener en cuenta sus costes, los cuales se encuentran detallados en el apartado de costes de Crashlytics.

Todas estas métricas e indicadores son de gran utilidad para detectar cambios en el rendimiento de la aplicación, pero a la hora de detectar un fallo en concreto es necesario hacer uso de otras herramientas que nos permitan analizar la causa de este cambio en el rendimiento para poder solucionarla.

5.6. Herramientas para detectar las causas de los problemas de rendimiento

Cuando se detecta una bajada de rendimiento, es de gran importancia detectar la causa, que usualmente será algún cambio introducido en las últimas versiones, es por ello que, aunque no fuera parte del alcance inicial del proyecto, la directora por parte de la empresa solicitó al alumno que analizara herramientas que pueden ayudar en esta tarea. En esta sección se analizarán dos herramientas para facilitar a los desarrolladores la detección de fallos que dan lugar a problemas de rendimiento en aplicaciones Android. Se explicará cómo se configuran y utilizan para ello.

5.6.1. Android Profiler

Android Profiler[2] es una herramienta integrada en el propio Android Studio que proporciona datos en tiempo real que ayudan al desarrollador a comprender la forma en la que una aplicación utiliza los recursos de la CPU, la memoria, la red y la batería [6].

Comenzar a recopilar estos datos es muy sencillo, simplemente se debe ejecutar la aplicación y seleccionar Profiler en Android Studio. A continuación, se debe crear una nueva sesión en la que se seleccionarán el dispositivo y la aplicación que se quiere monitorizar.

A continuación, veremos un panel que contiene información sobre los eventos relacionados con las entradas de usuario y el estado de la CPU, memoria y consumo de batería del dispositivo en ese momento, como se ve en el ejemplo 5.3. Al pulsar en cada uno de estos apartados, obtenemos información mucho más detallada sobre el rendimiento de la aplicación a lo largo de toda la ejecución.

[2]Android Profiler: https://developer.android.com/studio/profile/android-profiler

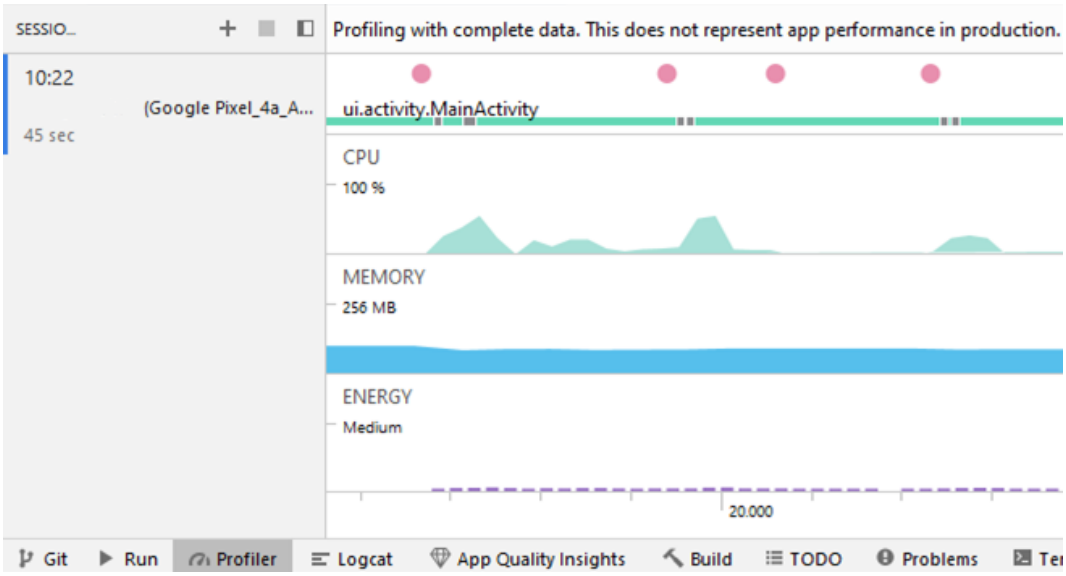


Figura 5.3: Ejemplo de panel de Android Profiler sobre una aplicación de prueba

En el apartado de CPU, podemos ver el porcentaje de uso de CPU y las tareas concretas de la pila que se están ejecutando a lo largo de toda la ejecución. En el apartado de memoria, vemos la cantidad de memoria (en MB) que se utiliza en cada momento, dividida en Java, Nativo, Gráficos, Pila, Código y Otros. Además, se registran todas las llamadas al Garbage Collector, que, si aparecen demasiadas, puede ser un indicador de que algo no está funcionando correctamente. Por último, en el apartado de energía, podemos ver el consumo de energía del dispositivo a lo largo de la ejecución, clasificado en Ligero, Medio o Alto.

Esta información es de gran utilidad para detectar posibles fallos si sabemos que el rendimiento de la aplicación ha disminuido. Por ejemplo, si vemos demasiadas llamadas al recolector de basuras en el apartado de la memoria, puede indicar que ciertos objetos se están utilizando mal y se están creando demasiados o no se están eliminando correctamente, lo que provoca una bajada de rendimiento.

Además de esto, existen otras causas más concretas de posibles problemas de rendimiento en las aplicaciones, como memory leaks. En el siguiente apartado se analizará la herramienta LeakCanary, que se utiliza para detectar y resolver memory leaks en Android.

5.6.2. LeakCanary

LeakCanary[3] es una librería para detectar memory leaks en aplicaciones Android. Un memory leak ocurre cuando un objeto que ya no se necesita sigue siendo referenciado y, por lo tanto, no puede ser recolectado por el Garbage Collector de Android. Esto puede resultar en un consumo excesivo de memoria a medida que la aplicación se ejecuta y puede llevar a problemas como un rendimiento lento, bloqueos o incluso cierres inesperados de la aplicación.

Para comenzar a utilizar la herramienta, simplemente hay que añadir una dependencia de la librería de LeakCanary en el archivo Gradle de la aplicación como se indica en la documentación [7]. Es importante añadirla como `debugImplementation` dado que LeakCanary

[3]LeakCanary: https://square.github.io/leakcanary

solo debería funcionar en versiones debug de la aplicación, ya que se instala una aplicación en el dispositivo con la información que recopila la herramienta.

Una vez añadida la dependencia, al ejecutar la aplicación normalmente, aparecerá una notificación en el dispositivo que indica el número de objetos retenidos que ha detectado LeakCanary en la pila. Si el número de objetos retenidos es menos que cinco, se puede hacer click en la notificación para volcar y analizar la pila. En cambio, si es mayor que 5, la pila se vuelca automáticamente. Los objetos se consideran retenidos cuando han pasado más de cinco segundos desde que un objeto se ha destruido pero una referencia a él sigue activa.

Una vez volcada y analizada la pila, en la aplicación 'Leaks' que crea la herramienta se pueden ver todos los memory leaks que se han detectado, como se ve en el ejemplo 5.4. Los leaks se categorizan en dos grupos:

- **Application leaks:** Memory leaks propios de la aplicación que se esté analizando.
- **Library leaks:** Memory leaks causados por un error conocido en código de terceros sobre el que el desarrollador no tiene control. Tiene impacto en el código pero el desarrollador de la aplicación que se esté analizando no lo puede arreglar.

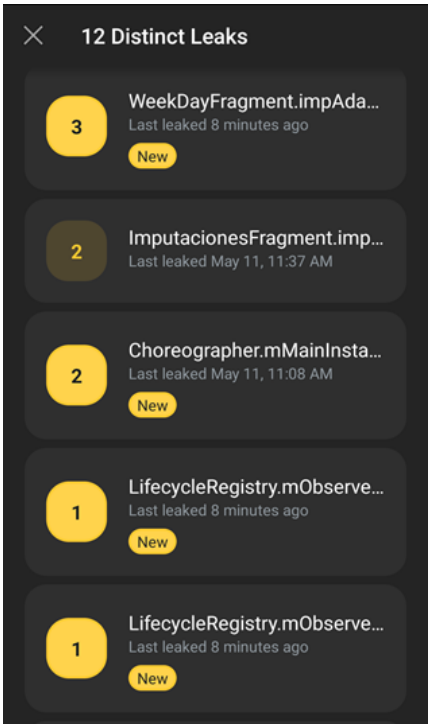


Figura 5.4: Ejemplo de leaks en una aplicación de imputación de horas

Al pulsar en cada uno de los leaks, podremos ver la pila de la ejecución y más información sobre el leak, como se puede ver en el ejemplo 5.5

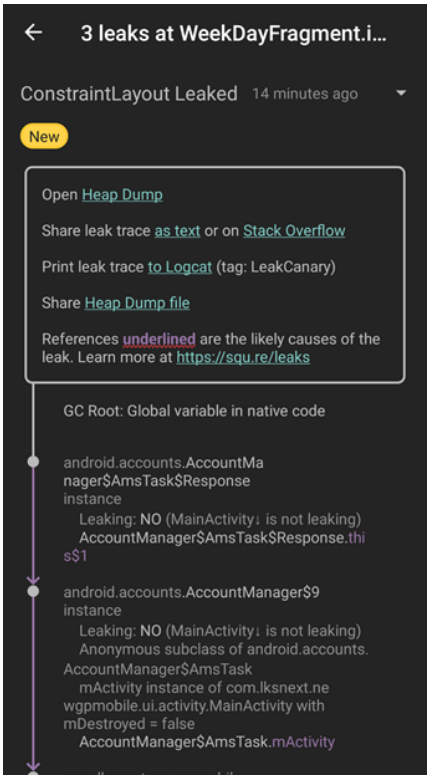


Figura 5.5: Ejemplo de memory leak

Esta información facilita al desarrollador solucionar el memory leak, y así mejorar el rendimiento de la aplicación.

CAPÍTULO 6

Test Lab

En esta tercera iteración se estudiará y analizará en profundidad la herramienta de Firebase Test Lab, que permite probar aplicaciones en una gran variedad de dispositivos y configuraciones.

Debido a la gran variedad de dispositivos Android, en ocasiones probar una aplicación puede ser complicado. Test Lab ofrece la posibilidad de testear aplicaciones de manera rápida y sencilla, además de generar informes detallados de las pruebas, lo que permite identificar posibles errores de manera más sencilla.

Para comenzar a utilizar la herramienta, solo es necesario tener la aplicación conectada con un proyecto de Firebase como se indica en la documentación [8].

En los siguientes apartados se analizará la herramienta en profundidad, se explicarán el tipo de pruebas que se pueden realizar, los diferentes modos de ejecución que ofrece, los resultados que genera y los costes de la misma. Además se explicará también la herramienta Gradle Managed Devices, que ofrece una funcionalidad similar a la de Test Lab pero utilizando dispositivos propios.

6.1. Análisis general

Antes de publicar una aplicación, es de gran importancia que esta haya sido probada de forma exhaustiva en diferentes escenarios, permitiendo descubrir problemas y errores que se pueden solucionar con un coste mucho más bajo del que tendría si la aplicación estuviese en producción.

Usualmente, estas pruebas se realizan con un único dispositivo, por lo que a menudo hay errores que no se descubren debido a las diferencias entre diferentes dispositivos. Por ello, es vital probar la aplicación en la mayor cantidad de dispositivos que los usuarios puedan utilizar, para descubrir todo tipo de errores y asegurarse de que la aplicación sea compatible con ellos y funcione correctamente.

Además, testeando aplicaciones en múltiples dispositivos se pueden detectar oportunidades para mejorar la experiencia del usuario y maximizar el rendimiento, como cambiar

ciertos aspectos de la aplicación para adecuarse a distintos tamaños de pantalla o cambios en la usabilidad por especificaciones de dispositivos particulares.

Esto supone un gran coste a la hora de probar la aplicación, debido a la multitud de dispositivos con Android en el mercado. Para solucionar este problema, Test Lab permite ejecutar pruebas en una amplia selección de dispositivos tanto físicos como virtuales sin necesidad de tener que adquirirlos y mantenerlos.

A continuación se exponen los dos tipos de pruebas que ofrece Test Lab actualmente y se explican brevemente.

6.2. Tipos de pruebas

Esta herramienta permite ejecutar dos tipos de pruebas:

- **Pruebas de instrumentación:** Estas pruebas requieren de un contexto y un entorno real y testean una funcionalidad de la aplicación en concreto, asegurándose de que funcione correctamente. Estas pruebas están escritas por el equipo desarrollador o de calidad utilizando los frameworks Espresso o UI Automator, que permiten crear scripts de prueba que imitan las acciones del usuario en la aplicación. También pueden incluir pruebas de integración, donde se evalúa la capacidad de la aplicación para interactuar con otros servicios.
- **Pruebas Robo:** Estas pruebas analizan la estructura de la interfaz de usuario de la aplicación y la exploran inteligentemente. Es decir, la herramienta intenta simular las acciones de un usuario de forma automática, sin escribir ningún tipo de test, interactuando con los botones, campos de texto o pantallas de la aplicación.

  Además, se pueden grabar e indicar acciones que se deben realizar antes de comenzar la prueba, por ejemplo, dando unas credenciales de prueba para hacer login en la aplicación, y una vez hecho login, se empiece a interactuar con la misma.

En el siguiente apartado se explican los diferentes métodos de ejecución que ofrece Test Lab para ejecutar estas pruebas.

6.3. Modos de ejecución

Test Lab ofrece diferentes maneras de ejecutar estos tests:

- **A través de la consola web de Firebase:** Tanto las pruebas de instrumentación como las pruebas Robo pueden ejecutarse desde la consola web de Firebase.

  Para las pruebas Robo, basta con subir el APK de la aplicación a testear y un archivo indicando acciones iniciales si fuese necesario. En cambio, para las pruebas de instrumentación hace falta subir el APK de la aplicación y el APK de pruebas. Ambos APKs deben estar firmados y pueden ser generados fácilmente desde Android Studio.
- **A través de Android Studio:** Android Studio solo permite ejecutar pruebas de instrumentación. Para ello, simplemente hay que crear una configuración para los tests instrumentados en la que se seleccione como objetivo 'Firebase Test Lab Device

Matrix' donde indiquemos los dispositivos y configuraciones en las que queremos que se ejecuten las pruebas, como se ve en la Figura 6.1. Después, basta con ejecutar los tests como normalmente se haría en el entorno de desarrollo, y obtendremos los resultados en la consola de Test Lab.

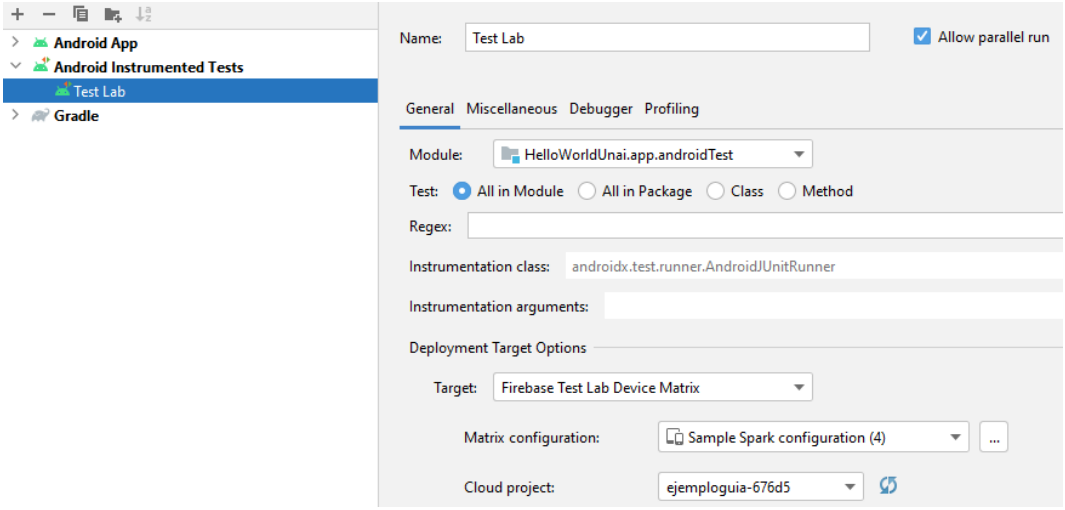


Figura 6.1: Ejemplo de una configuración de ejecución de Tests en Android Studio con Test Lab

- **A través de la CLI de Google Cloud:** Test Lab también ofrece la opción de ejecutar las pruebas a través de la línea de comandos de GCLoud, lo que permite integrar estas ejecuciones de test en entornos de Integración Continua de manera sencilla, tanto para pruebas Robo como de instrumentación.

  Para ello, primero debe de configurarse el cliente de Google Cloud, seleccionando una cuenta que tenga los permisos necesarios para ejecutar las pruebas.

  A continuación, desde la consola podemos ver los dispositivos disponibles con el comando `gcloud firebase test android models list`, como se puede ver en la Figura 6.2.

```
C:\Users\u.pinedo\AppData\Local\Google\Cloud SDK>gcloud firebase test android models list
+--------------------+------------------+-----------------------------------------+----------+------------+-----------------------+
|      MODEL_ID      |       MAKE       |               MODEL_NAME                |   FORM   | RESOLUTION |     OS_VERSION_IDS    |
+--------------------+------------------+-----------------------------------------+----------+------------+-----------------------+
| 1610               | Vivo             | vivo 1610                               | PHYSICAL | 1280 x 720 | 23                    |
| ASUS_X00T_3        | Asus             | ASUS_X00TD                              | PHYSICAL | 1080 x 2160| 27,28                 |
| AmatiTvEmulator    | Google           | Google TV Amati                         | VIRTUAL  | 1080 x 1920| 29                    |
| AndroidTablet270dpi| Generic          | Generic 720x1600 Android tablet @ 270dpi| VIRTUAL  | 1600 x 720 | 30                    |
| F01L               | FUJITSU          | F-01L                                   | PHYSICAL | 1280 x 720 | 27                    |
| FRT                | HMD Global       | Nokia 1                                 | PHYSICAL |  854 x 480 | 27                    |
| G8142              | Sony             | G8142                                   | PHYSICAL | 1080 x 1920| 25                    |
| GoogleTvEmulator   | Google           | Google TV                               | VIRTUAL  |  720 x 1280| 30                    |
| HWANE-LX2          | Huawei           | ANE-LX2                                 | PHYSICAL | 1080 x 2280| 28                    |
| HWCOR              | Huawei           | COR-L29                                 | PHYSICAL | 1080 x 2340| 27                    |
| HWMHA              | Huawei           | MHA-L29                                 | PHYSICAL | 1920 x 1080| 24                    |
| MediumPhone.arm    | Generic          | Medium Phone, 6.4in/16cm (Arm)          | VIRTUAL  | 2400 x 1080| 26,27,28,29,30,32,33  |
| MediumTablet.arm   | Generic          | Medium Tablet, 10in/25cm (Arm)          | VIRTUAL  | 2560 x 1600| 26,27,28,29,30,32,33  |
| Nexus10            | Samsung          | Nexus 10                                | VIRTUAL  | 2560 x 1600| 21,22                 |
| Nexus4             | LG               | Nexus 4                                 | VIRTUAL  | 1280 x 768 | 21,22                 |
| Nexus5             | LG               | Nexus 5                                 | VIRTUAL  | 1920 x 1080| 21,22,23              |
| Nexus5X            | LG               | Nexus 5X                                | VIRTUAL  | 1920 x 1080| 23,24,25,26           |
| Nexus6             | Motorola         | Nexus 6                                 | VIRTUAL  | 2560 x 1440| 21,22,23,24,25        |
| Nexus6P            | Google           | Nexus 6P                                | VIRTUAL  | 2560 x 1440| 23,24,25,26,27        |
| Nexus7             | Asus             | Nexus 7 (2012)                          | VIRTUAL  | 1280 x 800 | 21,22                 |
| Nexus7_clone_16_9  | Generic          | Nexus7 clone, DVD 16:9 aspect ratio     | VIRTUAL  | 1280 x 720 | 23,24,25,26           |
| Nexus9             | HTC              | Nexus 9                                 | VIRTUAL  | 2048 x 1536| 21,22,23,24,25        |
| NexusLowRes        | Generic          | Low-resolution MDPI phone               | VIRTUAL  |  640 x 360 | 23,24,25,26,27,28,29,30|
```

Figura 6.2: Ejemplo de dispositivos disponibles desde CLI de GCloud

Por último, para ejecutar las pruebas en un entorno de integración continua, podemos añadir un nuevo paso al proceso de compilación en el que se ejecuten las pruebas.

Para ejecutar una prueba Robo, se debe ejecutar el comando que se puede ver en el Código 6.1, y para ejecutar una prueba de instrumentación, el comando que se puede ver en el Código 6.2

```
gcloud firebase test android run
    --type robo
    --app app.apk
    --robo-script script
    --device model=Pixel4,version=29,locale=en,
        orientation=portrait
    --device model=Pixel5,version=31,locale=es,
        orientation=landscape
    --timeout 90s
    --client-details matrixLabel="Matriz de ejemplo"
```

Código 6.1: Ejemplo de comando para ejecutar prueba Robo desde CLI de GCloud

```
gcloud firebase test android run
    --type instrumentation
    --app app.apk
    --device model=Pixel4,version=29,locale=en,
        orientation=portrait
    --device model=Pixel5,version=31,locale=es,
        orientation=landscape
    --timeout 90s
    --client-details matrixLabel="Matriz de ejemplo"
```

Código 6.2: Ejemplo de comando para ejecutar prueba de Instrumentación desde CLI de GCloud

Los resultados de las pruebas aparecerán en la consola e incluirán un enlace al detalle de la misma en Test Lab.

Para cualquiera de los dos tipos de prueba y tipos de ejecución, es necesario seleccionar en qué dispositivos queremos que se ejecuten las pruebas y con qué configuración (nivel de API, idioma y orientación de la pantalla). En la figura 6.3 se puede ver un ejemplo de dispositivos disponibles desde la consola.

Una vez seleccionados todos los dispositivos, se generará una matriz de pruebas, en la que veremos todos los resultados de las ejecuciones, uno por cada dispositivo y prueba. En la siguiente sección se explica más detalladamente qué es una matriz de pruebas y cómo analizar estos resultados.

6.4. Resultados

6.4.1. Matriz de pruebas

Para analizar los resultados de las pruebas, es importante entender qué es una matriz de pruebas. Una matriz de pruebas contiene todos los resultados de las ejecuciones de prueba. Se compone de los dispositivos y las ejecuciones, existiendo un resultado por cada par de dispositivo y test.

Además, si una de las ejecuciones de una matriz falla, Test Lab considera que toda la matriz ha fallado.

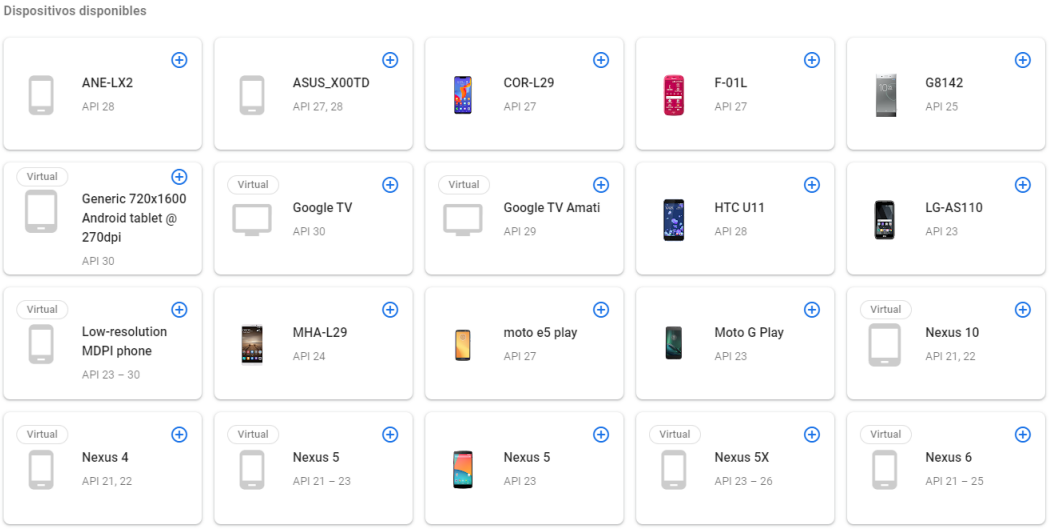


Figura 6.3: Ejemplo de una pantalla de selección de dispositivos en Firebase Test Lab

6.4.2. Resultados de pruebas Robo

Al ejecutar una prueba robo, los resultados incluirán los siguientes apartados:

- **Problemas de la prueba:** Tanto si la prueba falla como si no lo hace, Test Lab mostrará los fallos que han surgido a lo largo de la ejecución, y si la prueba ha fallado la razón por la que lo ha hecho. En la Figura 6.4 podemos ver un ejemplo de prueba Robo que ha fallado, en la que se indica la razón por la que ha fallado, en este caso una excepción no recuperable, y otros problemas relacionados con versiones del SDK.
- **Grafo de acciones:** Al finalizar la prueba, se generará un gráfico con todas las acciones que ha tomado la herramienta y todas las pantallas por las que ha pasado la ejecución, como se puede ver en la figura 6.5.
- **Registros:** Se mostrarán todos los registros (o logs) que haya generado la aplicación desde el comienzo de la prueba hasta su finalización, haya fallado o no.
- **Capturas de pantalla:** Se mostrarán también varias capturas de pantalla, que se realizan automáticamente cada vez que la aplicación cambia de pantalla, lo que hace posible ver todas las pantallas por las que ha pasado la ejecución.
- **Vídeo:** Si la prueba se hace utilizando un dispositivo físico, podremos ver un vídeo de la ejecución completa.
- **Análisis de rendimiento:** Si la prueba se ejecuta en un dispositivo físico, se recogen diferentes métricas y estadísticas sobre el rendimiento de la aplicación en la ejecución concreta de la prueba, de las que se destacan las siguientes:
  - *Latencia de entrada alta:* El porcentaje de eventos de entrada (eventos que realiza el usuario, como tocar un botón) que han tardado más de 24 ms en procesarse dividido por la cantidad de fotogramas que han tardado más de 16 ms en procesarse.

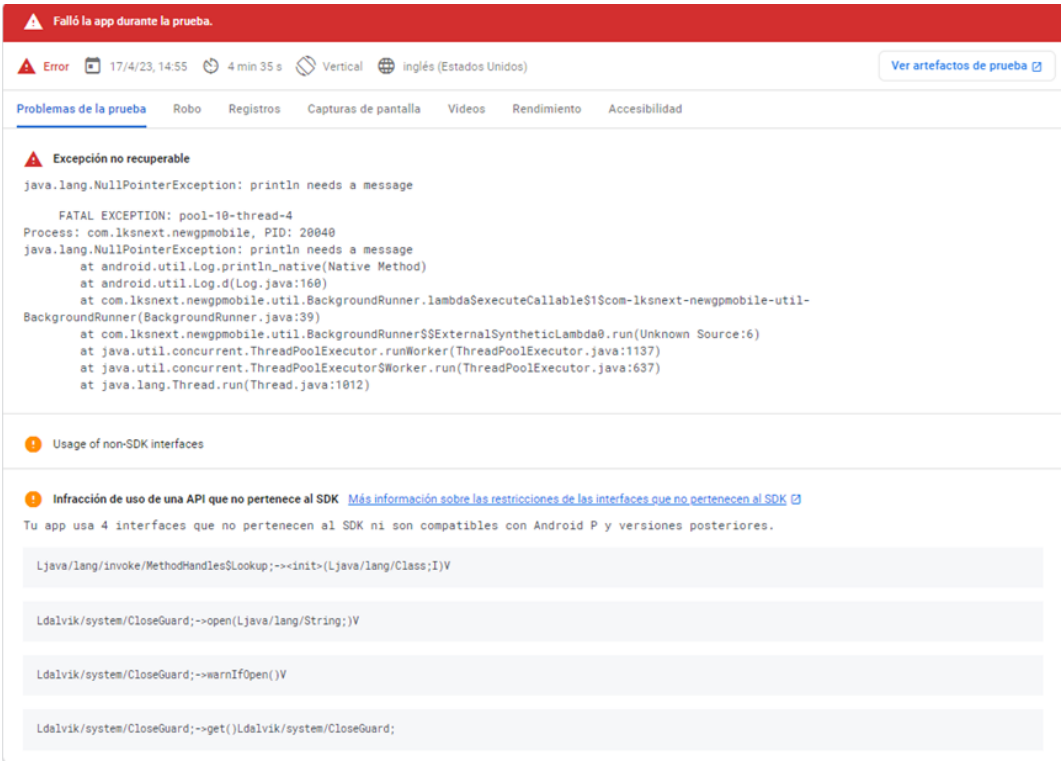


Figura 6.4: Ejemplo de problemas de una prueba Robo fallida en Firebase Test Lab

- *Subproceso de IU lento:* El porcentaje de veces que el subproceso de la interfaz de usuario ha tardado más de 8 ms en finalizar dividido por la cantidad de fotogramas que han tardado más de 16 ms en procesarse.
- *Tiempo de procesamiento:* Tiempo en ms que tarda en renderizarse por completo la visualización inicial.

Además de esto también podemos ver tres gráficas, del uso de la CPU, de la memoria RAM y de las solicitudes de Red a lo largo de la ejecución de la prueba, como se puede ver en el ejemplo 6.6.

- **Mejoras de accesibilidad:** En este apartado se muestran todos los problemas de accesibilidad que se han detectado en la prueba, además de las recomendaciones asociadas a ellos. Estos problemas se clasifican en tipo de problema y gravedad del mismo. Entre los tipos de problemas se encuentran 'Tamaño de objetivos táctiles', 'Contraste bajo', 'Etiquetas de contenido' e 'Implementación' y entre los niveles de gravedad 'Advertencias', 'Problemas menores' y 'Sugerencias'. En la Figura 6.7 podemos ver un ejemplo de esta clasificación.

  Algunos ejemplos de los problemas de accesibilidad que detecta Test Lab son tamaños de objetos demasiado pequeños, contraste bajo de ciertos textos, unidades de tamaño que no son correctas, alturas o anchuras que no deberían ser fijas o botones a los que les faltan etiquetas para que los lectores de pantalla los puedan leer correctamente.

  En la Figura 6.8 podemos ver un ejemplo de problema de ajuste de texto detectado por la herramienta para una aplicación de imputación de horas de LKS NEXT. En

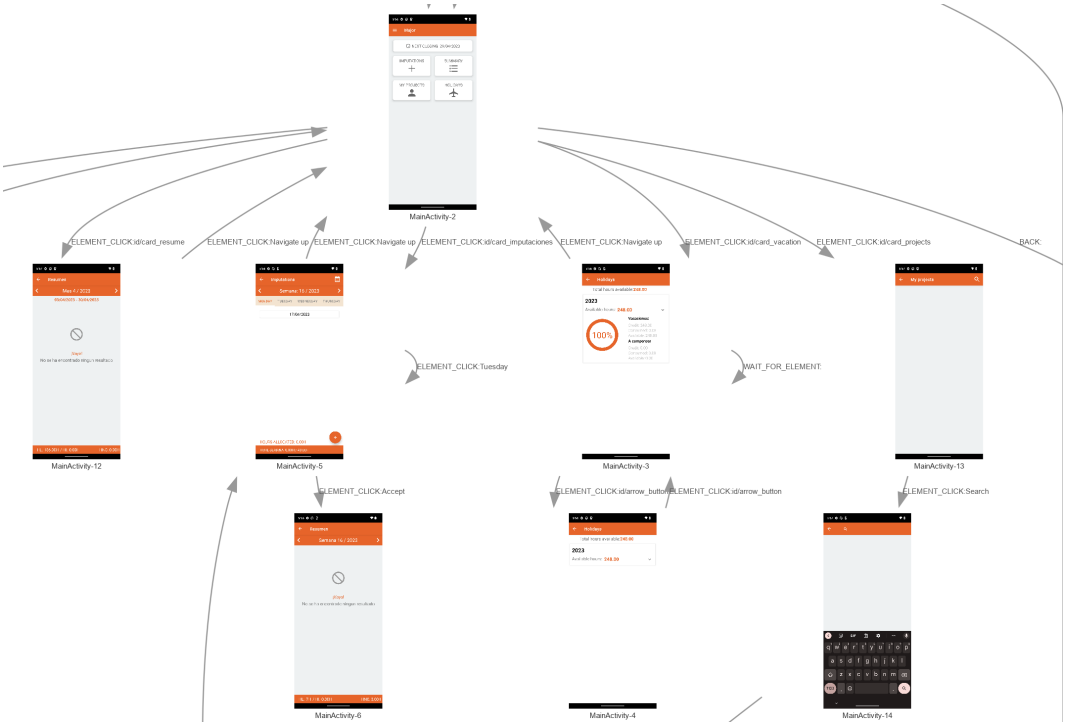


Figura 6.5: Fragmento de grafo de acciones de una prueba Robo en Firebase Test Lab

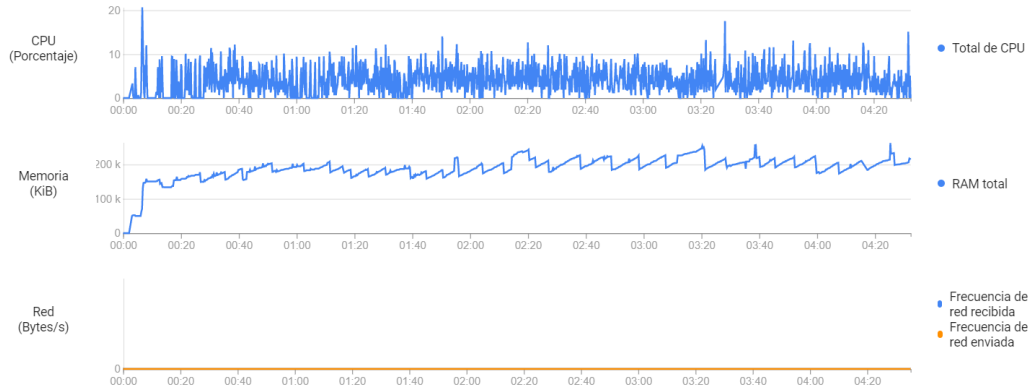


Figura 6.6: Ejemplo de rendimiento de CPU, RAM y red de una prueba Robo en Firebase Test Lab

este caso, indica que el `ViewGroup` seleccionado en la imagen tiene una altura fija y la `TextView` tiene texto ajustable, por lo que sugiere modificar los `LayoutParams` para permitir que el texto se amplíe. Además indica la ruta de acceso al elemento y todas los ejemplos de este problema, que en este caso son 12.

Es importante recalcar que estos problemas son sugerencias de Test Lab, y muchas veces detecta problemas que no lo son realmente, como contrastes bajos por determinadas acciones que tienen sentido y no hay que solucionar.

Tipos de problemas	Advertencias	Problemas menores	Sugerencias
	20	50	10
Tamaño de objetivos táctiles	10	3	0
Contraste bajo	1	42	0
Etiquetas de contenido	8	1	2
Implementación	1	4	8

Figura 6.7: Ejemplo de problemas de accesibilidad de una prueba Robo en Firebase Test Lab

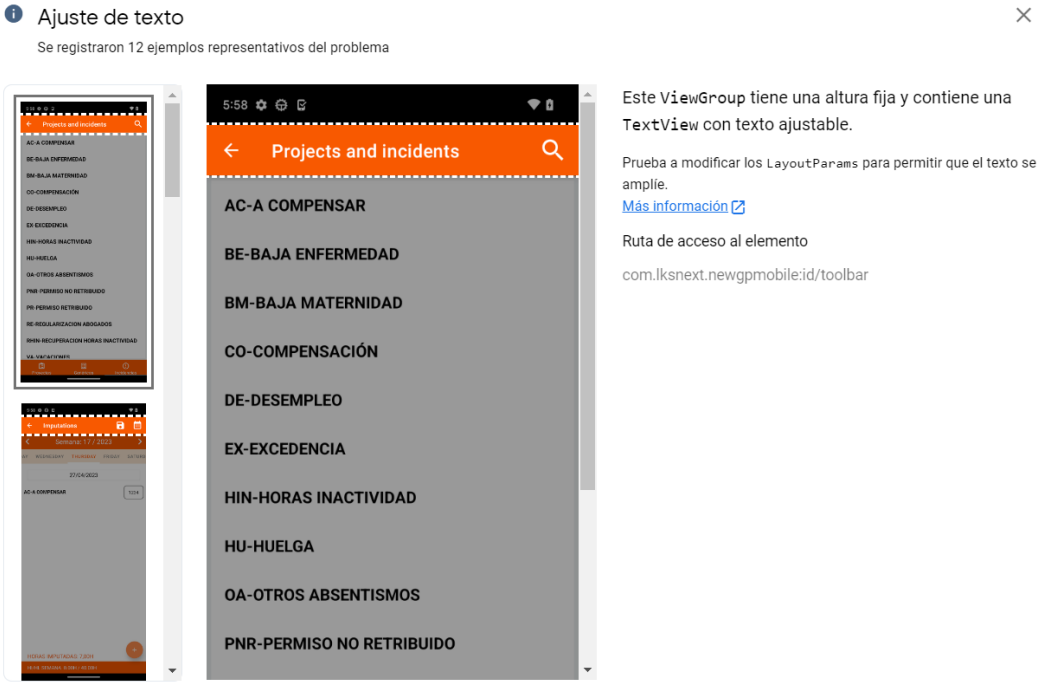


Figura 6.8: Ejemplo de problema concreto de accesibilidad de una prueba Robo en Firebase Test Lab

6.4.3. Resultados de pruebas de instrumentación

Al ejecutar una prueba de instrumentación, los resultados incluirán los siguientes apartados:

- **Pila:** Si la prueba ha fallado, se mostrará la pila de la ejecución, como se puede ver en el ejemplo 6.9.
- **Registros:** Al igual que en las pruebas Robo, se mostrarán todos los registros (o logs) que haya generado la aplicación desde el comienzo de la prueba hasta su finalización, haya fallado o no.
- **Vídeo:** Si se ha escogido un dispositivo físico para realizar la prueba, podremos ver un vídeo de la ejecución de la misma.

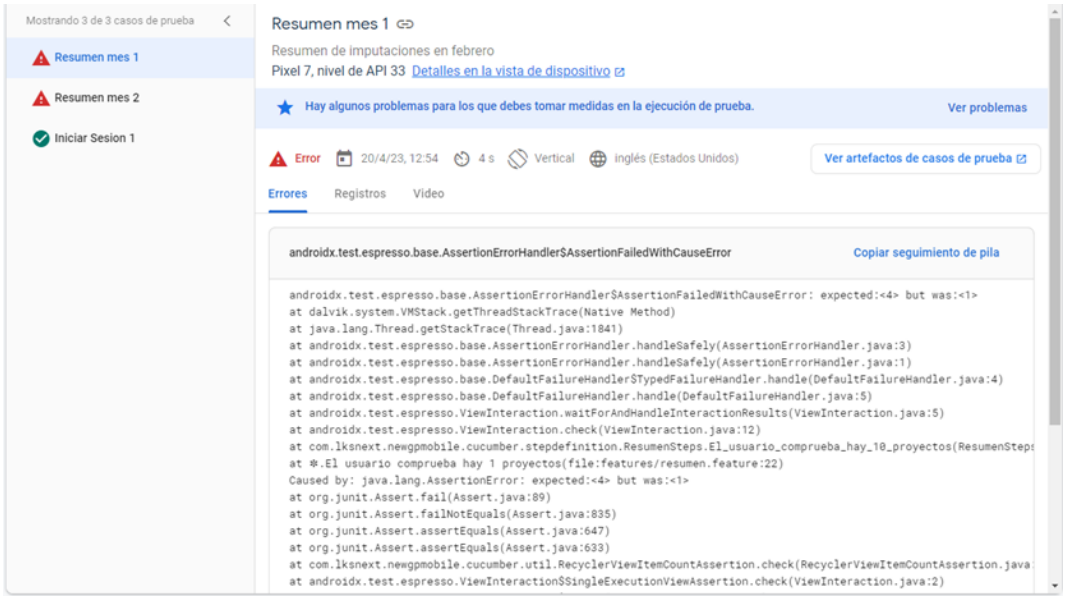


Figura 6.9: Ejemplo de una pila de una prueba instrumentada fallida en Firebase Test Lab

Haciendo uso de esta información la detección de posibles problemas de la aplicación o del test es mucho más sencillo, ya que permite ver exactamente los pasos que ha seguido la ejecución y en qué momento ha fallado si lo ha hecho.

En la siguiente sección se detallan los costes de uso por las ejecuciones de estas pruebas.

6.5. Costes

Para LKS NEXT es importante conocer los costes que tiene esta herramienta de cara a hacer una propuesta a posibles clientes y empezar a utilizar esta herramienta en sus proyectos. Por ello, se han analizado los costes que tendría utilizar Test Lab en un proyecto real.

Firebase ofrece dos planes:

- **Plan gratuito (Spark):** Este plan ofrece diariamente:
  - Diez ejecuciones de prueba en dispositivos virtuales.
  - Cinco ejecuciones de prueba en dispositivos físicos.
- **Plan de pago (Blaze):** Este plan tarifica según los minutos que tarde la ejecución de las pruebas. Ofrece diariamente, de manera gratuita:
  - 60 minutos de ejecución de prueba en dispositivos virtuales.
  - 30 minutos de ejecución de prueba en dispositivos físicos.

  Si se superan estos límites, se aplican las siguientes tarifas:
  - 1$/h en dispositivos virtuales.
  - 5$/h es dispositivos físicos.

Además de esto, existen los siguientes límites de las APIs de Test Lab:

- **API de Cloud Testing:**
  - 10.000.000 llamadas diarias.
  - 120.000 llamadas en 1 minuto.
- **API de Cloud Tool Results:**
  - 200.000 llamadas diarias.
  - 2.400 llamadas en 1 minuto.

Además de esta herramienta, existen otras alternativas para probar aplicaciones en granjas de dispositivos. Después de presentar esta herramienta a la persona que dirige el departamento de movilidad de la empresa, comentó que sería interesante para LKS NEXT analizar otras herramientas que ofrezcan granjas de dispositivos para ejecución de pruebas, ya que varios clientes habían mostrado interés por ellas. En el siguiente apartado se analiza la granja de dispositivos de Amazon Web Services[1] y se compara con Test Lab.

6.6. AWS Device Farm

AWS Device Farm[2] es un servicio en la nube ofrecido por Amazon Web Services que ofrece, al igual que Test Lab, una granja de dispositivos físicos para realizar pruebas en aplicaciones Android, lo que ayuda a garantizar la calidad y compatibilidad de las aplicaciones antes de su lanzamiento [9].

6.6.1. Análisis general

Device Farm permite ejecutar gran variedad de pruebas para aplicaciones Android. Para ello es necesario generar tanto el apk de la aplicaciones a probar cómo el apk de pruebas, al igual que en Test Lab. Además, permite ejecutar diferentes tipos de prueba utilizando diferentes frameworks, entre los que destacan los siguientes:

- **Built-in: Fuzz:** Este tipo de prueba no requiere que haya ningún test desarrollado, simplemente se envían eventos aleatorios para ver cómo se comporta la aplicación.
- **Appium**[3]**:** Utiliza el framework Appium, que simula las acciones del usuario como toques o gestos, similar a Selenium[4] pero para aplicaciones Android.
- **Calabash**[5]**:** Utiliza el framework Calabash para desarrollar las pruebas, que simula también las acciones del usuario. Permite utilizar Cucumber, pero lleva años sin mantenimiento.

[1] Amazon Web Services: https://aws.amazon.com/es
[2] AWS Device Farm: https://aws.amazon.com/es/device-farm
[3] Appium: https://appium.io
[4] Selenium: https://www.selenium.dev
[5] Calabash: https://github.com/calabash/calabash-android

- **Instrumentation:** Permite ejecutar todo tipo de pruebas de instrumentación, creadas con UI Automator o Espresso.

Además, la configuración de la ejecución se encuentra en un archivo YAML personalizable que se puede editar en caso de ser necesario.

Estas pruebas se pueden ejecutar en multitud de dispositivos físicos que ofrece Device Farm, de los que se pueden crear y guardar grupos. Para crear estos grupos, los dispositivos se pueden filtrar por diferentes atributos, como nivel de OS o disponibilidad. Además de estos dispositivos, AWS ofrece la posibilidad de tener dispositivos privados en su granja, es decir, dispositivos dedicados exclusivamente a un cliente, con las configuraciones que especifiquen.

Por último, Device Farm permite añadir cualquier tipo de información o aplicación externa que tenga que instalarse en el dispositivo para el correcto funcionamiento de la aplicación a probar.

En cuanto a la integración de la herramienta en Integración Continua, AWS cuenta con la interfaz de línea de comandos AWS CLI, que permite integrar las ejecuciones de pruebas en un entorno de Integración Continua/Entrega Continua (CI/CD), para poder ejecutar las pruebas automáticamente al crear una nueva versión de la aplicación, por ejemplo.

6.6.2. Comparación con Test Lab

A continuación se presenta en la tabla 6.1 que compara las diferencias y características más relevantes de Test Lab de Firebase con Device Farm de AWS. Esta comparativa es muy interesante para la empresa ya que ofrecen un servicio muy similar.

CARASTERÍSTICA	FIREBASE TEST LAB	AWS DEVICE FARM
Precio	Dispositivos físicos: 5$/hora Dispositivos virtuales: 1$/hora	Pago por uso: 0.17$/min Tarifa plana: 250$/mes Mantenimiento en caso de dispositivo privado: 200-500$/mes
Tipo de dispositivos	Públicos	Públicos y privados
Tipos de pruebas	Robo e Instrumentadas	Fuzz, Appium, Calabash e Instrumentadas
Configuración de dispositivo	Versiones y configuraciones estandar	Permite especificar versiones y configuraciones propias
Integración en CI/CD	CLI GCloud	CLI AWS
Informes y registros	Informes detallados, video y rendimiento	Informes detallados, video y rendimiento

Tabla 6.1: Comparativa entre Test Lab y Device Farm

Por último, es importante recalcar que Device Farm tiene ciertos problemas con la ejecución de tests utilizando JUnit 4, ya que no soporta definiciones de suites de pruebas, lo que no permite realizar ciertas ejecuciones de tests que requieran de determinadas etiquetas que indican el orden de la ejecución. Para utilizar Cucumber, Device Farm da soporte a Calabash, pero este framework lleva años sin actualizarse, por lo que no permite llevar a cabo la estrategia Behaviour Driven Development [10] que se lleva a cabo en la empresa.

Además de estas herramientas, existen otras alternativas para testear aplicaciones en una granja de dispositivos propia. Debido a los altos costes de ejecutar pruebas en Test Lab y AWS Device Farm, es interesante para LKS NEXT analizar otras herramientas que permitan ejecutar las pruebas en servidores propios. En el siguiente apartado se explica una de ellas y cómo utilizarla.

6.7. Gradle Managed Devices

Gradle Managed Devices[6] es una funcionalidad de Gradle que permite definir conjuntos de dispositivos en los que se probará la aplicación utilizando pruebas de instrumentación ya diseñadas e implementadas previamente. Permite escoger entre una amplia variedad de dispositivos y configuraciones de Google, pero únicamente pueden ser dispositivos virtuales [11].

En el siguiente apartado se explica cómo definir estos dispositivos y configuraciones.

6.7.1. Configuración inicial

Para utilizar esta funcionalidad, primero se deben configurar los dispositivos que se quieren utilizar. Para ello, en el archivo Gradle de la aplicación, se deben definir los dispositivos, dándoles un nombre y seleccionando el modelo, el nivel de API y la imagen a utilizar, como se puede ver en el ejemplo 6.10, en el que hay definidos tres dispositivos.

```
testOptions {
    managedDevices {
        devices {
            pixel2api29 (com.android.build.api.dsl.ManagedVirtualDevice) {
                // Use device profiles you typically see in Android Studio.
                device = "Pixel 2"
                // Use only API levels 27 and higher.
                apiLevel = 29
                // To include Google services, use "google".
                systemImageSource = "aosp"
            }
            pixel4api30atd (com.android.build.api.dsl.ManagedVirtualDevice) {
                device = "Pixel 4"
                apiLevel = 30
                systemImageSource = "aosp-atd"
            }
            pixel5api28 (com.android.build.api.dsl.ManagedVirtualDevice) {
                device = "Pixel 5"
                apiLevel = 28
                systemImageSource = "aosp"
            }
        }
```

Figura 6.10: Ejemplo de una configuración de dispositivos utilizando Gradle Managed Devices

Una vez definidos los dispositivos, se puede crear un grupo con ellos para ejecutar las pruebas en todos ellos.

Al ser una funcionalidad de Gradle, es muy sencillo integrar las pruebas en estos dispositivos en entornos de integración continua, ya que solo hay que ejecutar la tarea que se ve en el Código 6.3, indicando nombre de grupo o dispositivo y variante de compilación.

```
./gradlew 'GroupName'Group'BuildVariant'AndroidTest
```

[6] Gradle Managed Devices: https://developer.android.com/studio/test/gradle-managed-devices

Código 6.3: Task de gradle para ejecutar pruebas con Gradle Managed Devices

Estos dispositivos pueden tener diferentes tipos de imágenes cargadas, las cuales se explican en el siguiente apartado.

6.7.2. Tipos de imagen

Al definir un dispositivo, es necesario seleccionar qué tipo de imagen que llevará montada. Disponemos de tres opciones:

- **Aosp:** Es la imagen básica, que incluye todas las aplicaciones que llevan los teléfonos por defecto, pero sin los servicios de Google.
- **Google:** Esta imagen también incluye todas las aplicaciones que llevan los teléfonos por defecto pero además incluye los servicios de Google y sus aplicaciones.
- **Aosp-atd:** Significa 'Automated Test Devices'. Es una imagen mucho más ligera que las anteriores ya que no incluye aplicaciones que no sean estrictamente necesarias para el funcionamiento del dispositivo, lo que hace que las pruebas se ejecuten de manera mucho más rápida si la aplicación a probar no requiere de aplicaciones externas.

6.7.3. Resultados

Después de ejecutar esta tarea, se genera automáticamente un reporte de la ejecución en el que se indican el número de tests, el número de fallos, el número de tests saltados y la duración total de los mismos. Además, se muestra la duración de cada test en cada uno de los dispositivos. En la Figura 6.11 podemos ver un ejemplo de reporte para una ejecución de dos pruebas en tres dispositivos diferentes.

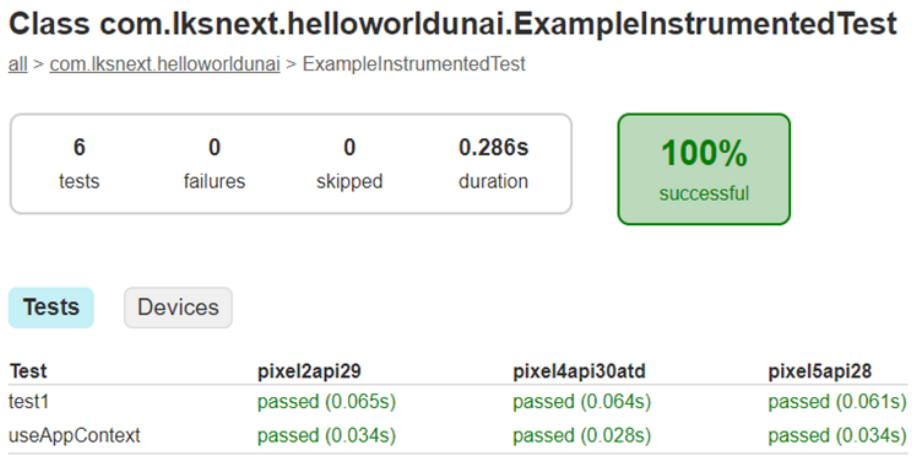


Figura 6.11: Ejemplo de reporte de ejecución de pruebas utilizando Gradle Managed Devices

CAPÍTULO 7

Diseño de forma de trabajo

En este capítulo se explica en detalle la forma de trabajo que se ha diseñado y que surge como el resultado de un análisis en profundidad de las herramientas para la gestión de la calidad que ofrece Firebase y la exportación de sus datos.

La implementación de una forma de trabajo clara y eficiente es fundamental en cualquier empresa, por lo que el objetivo de este capítulo es diseñar un proceso estructurado y adaptable que permita a los equipos incorporar estas herramientas a su forma de trabajo actual para ayudarles en sus tareas.

El alumno es consciente de que la definición de procesos es una tarea que exige un conocimiento exhaustivo de la cultura organizacional, pero al finalizar el proyecto surgió la idea de llevarlo a cabo como forma de integrar todo lo realizado a lo largo del mismo. Una vez definido, sería analizado por las personas responsables de la organización y, después de adaptarlo si fuera necesario, se desplegaría en los equipos correspondientes.

7.1. Proceso

Para que la forma de trabajo sea clara y más fácilmente aplicable por los trabajadores de la empresa, se ha diseñado un proceso utilizando BPMN 2.O [12] en el que se identifican claramente los roles, artefactos y tareas implicadas.

El proceso se centra en el desarrollo y gestión de la calidad de la aplicación dentro de la organización, por lo que se ha decidido no incluir al cliente en el proceso, aunque se encuentre involucrado indirectamente en ciertos puntos.

En primer lugar, es importante recalcar que, como se puede ver, el proceso se lleva a cabo para cada una de las versiones desarrolladas de la aplicación, y se divide en dos grandes partes. Por un lado, el tiempo en el que la versión de la aplicación se encuentra en desarrollo, y por otro lado, el tiempo en el que la versión de la aplicación se encuentra en producción.

7.1.1. Desarrollo

En la primera parte, que podemos ver en la Figura 7.1, el equipo de desarrolladores Android desarrollan la versión de la aplicación, y el equipo de calidad se encarga de diseñar e implementar tests unitarios y funcionales de las nuevas funcionalidades de la aplicación que incorpora la nueva versión. Posteriormente, estas pruebas se ejecutan en emuladores de manera local con Gradle Managed Devices y en diferentes dispositivos utilizando Test Lab. Además, se ejecutará una prueba Robo por cada versión para detectar posibles problemas de funcionalidad o rendimiento.

Estas ejecuciones de pruebas generarán un documento de resultados, que posteriormente utilizarán los desarrolladores para solucionar los posibles errores detectados por estas pruebas. Este proceso de desarrollo se repetirá hasta que no se encuentren nuevos errores.

Además, durante el desarrollo se llevarán a cabo ciertas acciones a nivel de código para dar más información a los crashes que recoge Crashlytics y haga más sencillo solucionarlos:

- **Identificador de usuario:** Se guardará en cada ejecución el identificador de cada usuario que ha sufrido el error.
- **Log indicando Activity:** Se guardará al cambiar de pantalla el activity en el que se encuentra, para, en caso de error, identificar en qué Activity ha ocurrido.

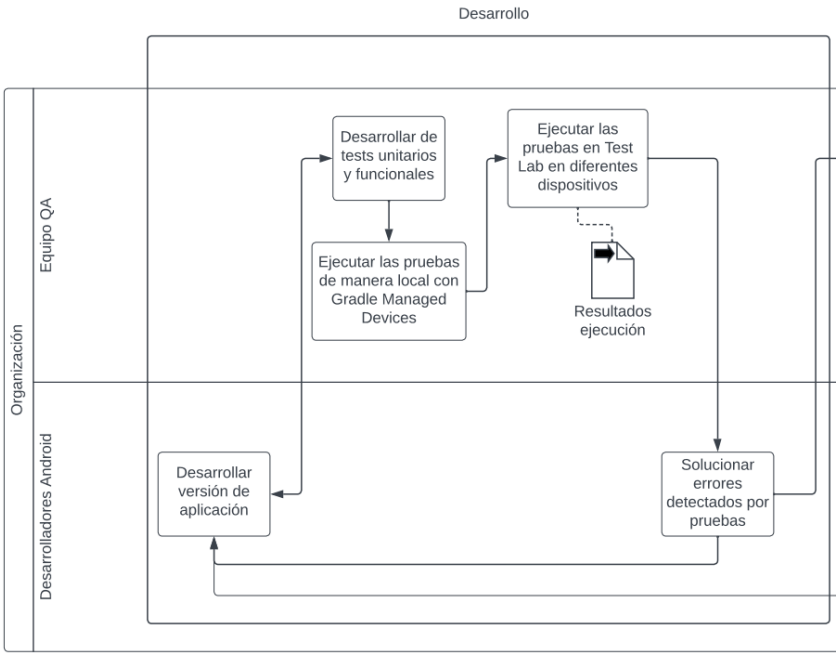


Figura 7.1: Proceso de integración de Firebase en desarrollo Android durante el desarrollo

7.1.2. Producción

En la segunda parte, que podemos ver en la Figura 7.2, el equipo de calidad utilizará las consultas diseñadas en BigQuery para exportar los datos recopilados por Crashlytics, obteniendo un documento con los resultados de estas consultas. El equipo de calidad revisará

estos datos e indicará a los desarrolladores qué errores deben solucionar. Los desarrolladores revisarán estos errores y anotarán en Crashlytics si el error ha sido solucionado o no, y a partir de qué versión. Para ello, harán uso de las notas de la propia consola de Crashlytics.

Este proceso se repetirá con las consultas diseñadas para Performance Monitoring, obteniendo también un documento con los resultados, que solucionará en caso de ser necesario el equipo de desarrolladores.

Por último, con toda esta información se generarán cuadros de mando que se mostrarán al cliente en reuniones que se tendrán periódicamente con el equipo de QA. A partir de estos cuadros de mando, se tomarán ciertas acciones sobre el desarrollo de la aplicación, y se creará una nueva versión si es necesario.

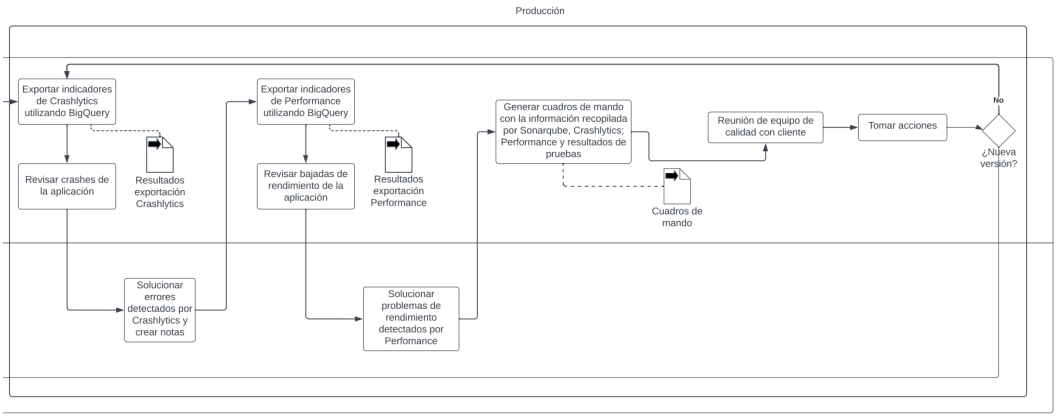


Figura 7.2: Proceso de integración de Firebase en desarrollo Android en producción

Debido al tamaño del proceso completo, podemos ver una copia del mismo en el Anexo B[1].

7.2. Situación actual de la forma de trabajo en la empresa

Actualmente, este proceso se está implementando en la empresa. Ciertas herramientas ya se están utilizando y otras aún están en proceso.

La guía creada para activar la exportación de los datos de Crashlytics y Performance se ha utilizado con varios clientes y los datos se están exportando y utilizando en las reuniones periódicas que tiene la empresa con clientes, haciendo este proceso que hacían manualmente de forma automática.

Por otro lado, el uso de Test Lab se está estudiando con los clientes ya que tiene un coste muy elevado, pero se pretende incorporar la herramienta Gradle Managed Devices para hacer pruebas en integración continua de manera local.

[1]También encontramos el proceso completo en la URL: https://drive.google.com/file/d/11YhSMOtmEetHsygRce0tKuyyvb4KOQoK

CAPÍTULO 8

Seguimiento y control del proyecto

En este capítulo se documentan todas las tareas relacionadas con el seguimiento y control que se han ido realizando a lo largo del desarrollo del proyecto. Desde la gestión del alcance y la gestión de riesgos, hasta la gestión del tiempo.

8.1. Gestión del alcance

El alcance del proyecto no ha cambiado a lo largo de todo el proyecto, aunque, debido a que las dedicaciones de las tareas iban mejor de lo previsto y el alumno aún disponía de tiempo, la directora del proyecto por parte de la empresa decidió añadir dos tareas dentro de algunos paquetes de trabajo. Las tareas eran las siguientes:

- **PERFORMANCE: Pruebas de performance (PR.PPR):** Después de utilizar Performance Monitoring para detectar cambios en el rendimiento de las aplicaciones, la directora del proyecto por parte de la empresa consideró relevante estudiar algunas herramientas que ayudasen al desarrollador a solucionar las bajadas de rendimiento de la aplicación.
- **TEST LAB: Análisis de alternativa AWS (TL.AWS):** Después de analizar la herramienta para realizar pruebas en diferentes dispositivos de Firebase, la directora del equipo de movilidad, a la que se le presentó el análisis realizado, propuso realizar una análisis de una herramienta similar que ofrece AWS: Device Farm.

Por otro lado, en una de las reuniones de seguimiento del proyecto, se decidió dibujar un proceso que explicara la forma de trabajo de manera más gráfica, por lo que se tuvo que añadir otra tarea para realizar esto al paquete de la forma de trabajo.

Estos cambios se comentaron con las directoras del proyecto, pero al no suponer una carga excesiva, el alcance del proyecto no cambió por lo que no fue necesaria una replanificacion para incluir estas tareas.

8.2. Gestión de riesgos

En este apartado se enumeran los riesgos que finalmente se han transformado en incidencias para el proyecto. Se explica cuál ha sido su impacto y cómo se ha solucionado.

8.2.1. R2-Captura de requisitos

Al principio del proyecto, cuando el alumno definió y escribió los requisitos, no había entendido del todo cuales eran y escribió requisitos funcionales cuando, debido a la naturaleza del proyecto, todos los requisitos del mismo son no-funcionales. Después de escribirlos, la directora del proyecto por parte de la empresa los revisó y tuvo una reunión con el alumno en la que el alumno comprendió cuáles eran verdaderamente los requisitos, e hizo los cambios necesarios, por lo que finalmente esta incidencia no tuvo ningún impacto en la planificación o en el desarrollo del proyecto.

8.2.2. R3-Tecnologías desconocidas por la empresa

Uno de los mayores desafíos de este proyecto era que nadie en la empresa conocía las herramientas que se iban a utilizar, por lo que el alumno tenía que valerse por sí mismo para resolver cualquier duda o problema que le pudiera surgir.

Durante el estudio de las herramientas, el alumno se encontró con dos principales problemas que tuvo dificultades para resolver.

- En primer lugar, al configurar la exportación de los datos de Crashlytics a BigQuery, el alumno no conseguía que los datos se exportasen siguiendo los pasos de la documentación de Crashlytics. Además, la exportación se hacía diariamente, por lo que tenía que esperar 24 horas para ver si los datos se habían exportado.
- Por otro lado, también encontró problemas a la hora de ejecutar suites de tests realizadas con JUnit 4 y Cucumber en la granja de dispositivos de AWS, las pruebas no se ejecutaban correctamente.

Finalmente, el alumno logró resolver ambos problemas leyendo preguntas y respuestas de usuarios en varios foros, descubriendo que el problema con la exportación se debía a un error de permisos de las cuentas del proyecto, y que el problema con AWS se debía a que ciertos aspectos de JUnit 4 no estaban implementados, como se documenta en la memoria.

Por ello, finalmente esto no supuso un gran problema para el desarrollo del proyecto.

8.3. Gestión del tiempo

A lo largo de todo el proyecto nunca se vieron desviaciones importantes, solo pequeñas desviaciones en algunas tareas, por lo que nunca se vio la necesidad de replanificar.

8.3.1. Desviaciones de fechas

En cuanto a las iteraciones, las fechas estimadas en las que se tenía que terminar cada iteración no se han cumplido, debido a que, a menudo, y ya que las iteraciones no tenían dependencias entre ellas, se ha estado trabajando en varias de ellas al mismo tiempo, puesto

que había momentos en los que no se podía seguir trabajando en ciertas iteraciones, por ejemplo, cuando había que esperar 24 horas para ver que si la exportación se había hecho correctamente. Además de esto, a menudo se han dado por finalizadas iteraciones antes de la fecha prevista pero más adelante en el proyecto se ha tenido que volver a realizar ciertas tareas nuevas que se han añadido dentro de estas.

De todas maneras, estos pequeños retrasos o adelantos en los hitos de finalización de iteraciones no han supuesto ningún problema para el desarrollo del proyecto, ya que además, el alumno contaba con más de 20 días para finalizar la memoria una vez finalizado el convenio con la empresa.

En la Figura 8.1 se puede ver las fechas estimadas y reales de los diferentes hitos del proyecto, teniendo en cuenta que en ciertas ocasiones se han tenido que realizar ciertas tareas de cada iteración posteriormente.

HITO	FECHA ESTIMADA	FECHA REAL
Inicio del proyecto	13/02/2023	13/02/2023
Fin de estudio	20/02/2023	20/02/2023
Fin It. 1 (Crashlytics)	28/03/2023	22/03/2023
Fin It. 2 (Performance)	12/04/2023	06/04/2023
FIn It. 3 (Test Lab)	05/05/2023	28/04/2023
Fin It. 4 (Forma de Trabajo)	24/05/2023	26/05/2023
Fin memoria	31/05/2023	20/06/2023
Defensa del proyecto	03-14/07/2023	03-14/07/2023

Figura 8.1: Fecha estimada y real de hitos del proyecto

8.3.2. Dedicaciones

En la Figura 8.2 podemos ver las dedicaciones estimadas y reales para cada tarea, además de sus desviaciones en horas, a falta de terminar de preparar la defensa del proyecto.

Como se puede ver, el paquete de *Crashlytics (CR)* ha llevado menos tiempo del estimado ya que la herramienta que se pretendía utilizar, BigQuery, permitía exportar los datos de manera muy sencilla, por lo que no fue necesario estudiar otras herramientas ni diseñar una exportación muy complicada, solo había que preparar las consultas SQL. Por ello, el estudio de la exportación de los datos llevó menos tiempo del esperado.

Por otro lado, los paquetes *Performance (PR)* y *Test Lab (TL)* han llevado más tiempo del esperado debido a las tareas que se añadieron a cada paquete, las pruebas de performance y el análisis de AWS respectivamente. El análisis de estas herramientas también llevó más tiempo del esperado, ya que eran herramientas que ofrecen una gran cantidad de información y era importante extraer lo realmente relevante para la empresa.

Por último, el paquete *Forma de trabajo (FT)* también ha llevado ligeramente más tiempo del esperado debido a la tarea de diseño de proceso que se añadió. Además, una vez empezada la tarea, el alumno se dio cuenta de que crear una forma de trabajo era una tarea que requería de mucho análisis previo de la forma de trabajo actual de la empresa, lo que llevó más tiempo del esperado.

PAQUETE DE TRABAJO	TAREA	DEDICACIÓN ESTIMADA	DEDICACIÓN REAL	DIFERENCIA
ESTUDIO INCIAL (EI)	Estudio Android (EA)	30	30	0
	SUBTOTAL	**30**	**30**	0
CRASHLYTICS (CR)	Análisis (ACR)	20	17	-3
	Estudio Big Query (BQ)	20	19	-1
	Estudio Exportación y Manipulación de Datos (EMD)	30	22	-8
	Estimación de costes (EC)	5	3	-2
	SUBTOTAL	**75**	**61**	-14
PERFORMANCE (PR)	Análisis (APR)	25	35	10
	Estimación de costes (EC)	5	1	-4
	Pruebas de Performance (PPR)		17	17
	SUBTOTAL	**30**	**53**	23
TEST LAB (TL)	Análisis (ATL)	25	38	13
	Estimación de costes (EC)	5	1	-4
	Analisis Alternativa AWS (AWS)		16	16
	SUBTOTAL	**30**	**55**	25
FORMA DE TRABAJO (FT)	Análisis (DFT)	5	8	3
	Diseño (DFT)	30	27	-3
	Diseñar Proceso (PC)		10	10
	SUBTOTAL	**35**	**45**	10
GESTIÓN (G)	Captura de requisitos (CR)	5	4	-1
	Planifiación (PL)	20	18	-2
	Seguimiento y Control (SyC)	20	18	-2
	SUBTOTAL	**45**	**40**	-5
TRABAJO ACADEMICO (TA)	Gestión TFG (GT)	3	1	-2
	Memoria (M)	70	75	5
	Defensa (D)	10	2	-8
	SUBTOTAL	**83**	**78**	-5
HORAS TOTALES		328	362	34

Figura 8.2: Dedicaciones estimadas y reales del proyecto

Aun así, la estimación de los costes fue mucho más rápida de lo estimada en todos los casos, por lo que las desviaciones no fueron excesivamente grandes.

Al calcular la desviación total del proyecto respecto a la planificación, obtenemos un 10,37 %, lo que, como se puede ver en el gráfico 8.3, indica que no ha habido grandes desviaciones en ninguno de los paquetes ni en el cómputo total del proyecto, además de haberse desarrollado según lo previsto.

De hecho, las desviaciones de los proyectos en informática [13], pueden llegar hasta el 50 % del proyecto, por lo que podemos considerar una desviación ligeramente superior al 10 % pequeña.

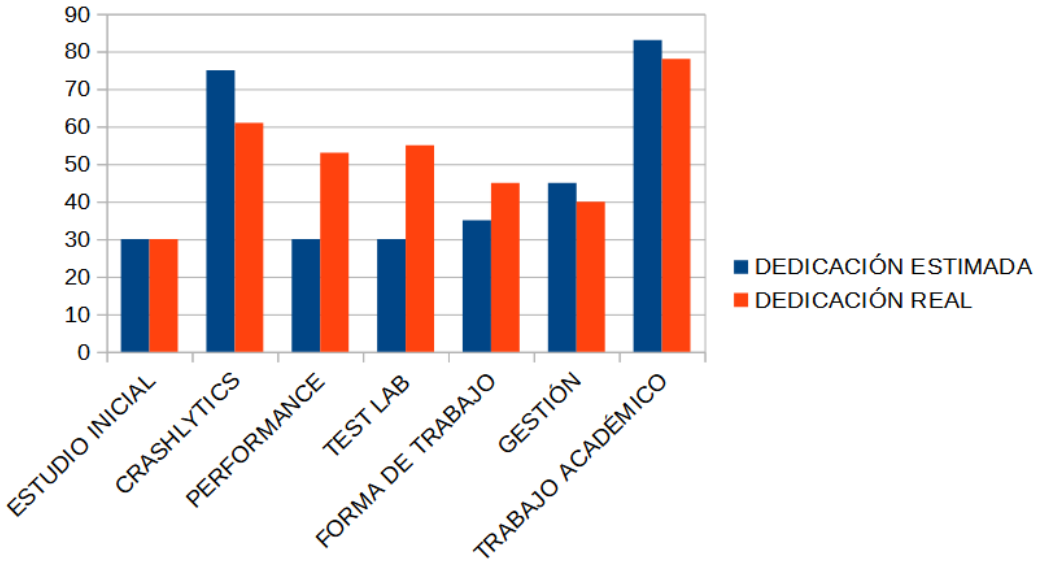


Figura 8.3: Dedicaciones estimadas y reales del proyecto

CAPÍTULO 9

Conclusiones y líneas futuras

En este último capítulo se enumeran todas las conclusiones que se han obtenido al finalizar el TFG y se repasan los objetivos definidos al inicio del mismo. Además, se explican brevemente los retos que han surgido desde el planteamiento del proyecto hasta el fin del mismo. Por último, se mencionan las líneas de futuro del proyecto que se podrían realizar en la empresa.

9.1. Conclusiones

El proyecto fue planteado con el objetivo de analizar las herramientas que ofrece Firebase para la gestión de calidad e integrarlas en la forma de desarrollo Android de la empresa. Este objetivo se ha cumplido con creces, ya que además de analizar las herramientas en profundidad, se ha diseñado una forma de exportar estos datos, se han analizado herramientas para solucionar problemas que detectan estas herramientas, se han comparado con otras del mercado y se ha creado un proceso en el que se integran estas herramientas.

9.1.1. Revisión de objetivos

Al plantear este TFG, se definieron varios objetivos, de los cuales se han cumplido los siguientes:

- **Análisis de herramientas para exportar datos de Crashlytics:** Después de analizar Crashlytics, se estudió la herramienta BigQuery para poder exportar los datos que genera. Además, se creó una guía en la que se explica como activar la exportación y resolver los posibles problemas que puedan surgir. También se han documentado las diferentes consultas con las métricas más importantes y cómo adaptarlas a cada proyecto. Esta guía se ha mandado a varios clientes y se utiliza a día de hoy para exportar los datos.
- **Análisis de Performance Monitoring:** Se ha analizado la herramienta, obtenido las métricas más interesantes para la empresa y diseñado consultas en BigQuery para obtener los datos más importantes que ofrece esta herramienta.

- **Análisis de Test Lab:** Se han analizado todas las posibilidades que ofrece la herramienta, además de analizar también Gradle Managed Devices y AWS Device Farm, con las que se ha comparado ya que ofrecen también hacer pruebas en granjas de dispositivos.
- **Estimación de los costes de uso de cada herramienta:** Se han documentado todos los costes que conlleva el uso de cada herramienta, comparándolo con otras alternativas en el caso de Test Lab.
- **Creación de forma de trabajo:** Se ha diseñado un proceso en el que se muestra detalladamente cómo integrar las herramientas analizadas en la forma de trabajo de desarrollo Android actual de la empresa.
- **Aplicación de las herramientas en aplicaciones internas de la empresa:** Hay ciertas herramientas que ya se están utilizado en aplicaciones internas de LKS NEXT, como Crashlytics, Performance y la exportación de sus datos con BigQuery. En cambio, hay otras, como Test Lab, que aún se están estudiando por parte de la empresa, ya que conllevan un coste elevado. Además de esto, y aunque no estaba planificado en el alcance inicial del proyecto, algunas de estas herramientas se están utilizando con clientes reales de la empresa.

En cuanto a objetivos personales, la evaluación personal del alumno es muy positiva, ya que se han alcanzado varios objetivos personales, como aprender a programar en Android o aprender a trabajar con un proyecto real, aspectos que el alumno no había tenido oportunidad de trabajar aún.

9.2. Retos del proyecto

En este apartado se explican los principales retos que han surgido a lo largo de todo el proyecto:

- **Aplicaciones Android:** Debido a que el proyecto se quería desarrollar en aplicaciones Android, el primer reto con el que se encontró el alumno fue entender y estudiar cómo se desarrollan aplicaciones Android: la arquitectura que seguían estas aplicaciones, layouts, paneles de navegación... Aprender a programar en Android era fundamental para poder probar todas las herramientas.
- **Entender forma de trabajo de la empresa:** Para poder diseñar una forma de trabajo en la que se integren las herramientas estudiadas, el alumno ha tenido que estudiar y entender cómo se trabaja actualmente en la empresa y por qué, desde cómo se desarrollan las aplicaciones, que equipos toman parte en el desarrollo y cómo, hasta las comunicaciones con el cliente y las decisiones que toma el cliente en base a lo que se le muestra. Precisamente para esto era importante el proyecto que se ha desarrollado, ya que ofrece información al cliente de manera clara sobre cómo está funcionando la aplicación.
- **Exportaciones de BigQuery cada 24 horas:** Uno de los grandes problemas que se encontró el alumno a la hora de configurar la exportación de los datos de Firebase a BigQuery, fue que los datos sólo se exportan cada 24 horas en la versión de prueba

de BigQuery que se utilizó, por lo qué era muy complicado hacer pruebas para saber por qué los datos no se estaban exportando, ya que solo se podía hacer una prueba al día hasta ver el resultado. Esto supuso un retraso a la hora de diseñar la exportación de los datos, pero no supuso una incidencia en la planificación ya que se empezó con otra iteración mientras se esperaba.

- **Exportación de notas a BigQuery:** En la forma de trabajo actual de LKS NEXT, se utilizan las notas de Crashlytics para llevar un registro de las revisiones de los crashes por parte de los desarrolladores, en las que se indican si está resuelto o no, en qué versión y por qué si el problema no se ha resuelto.

  A la hora de preparar las consultas con los datos que exporta Crashlytics a BigQuery, el alumno observó que las notas no se exportan, un gran problema teniendo en cuenta el uso que se hace de ellas en la empresa.

  El alumno se puso en contacto con los desarrolladores de Firebase, y estos confirmaron que actualmente las notas no se exportaban a BigQuery. Además de esto, mostraron interés en el uso que lleva a cabo la empresa y pidieron al alumno que explicase el caso de uso para hacer una petición interna que se revisaría internamente por los desarrolladores de Firebase.

9.3. Líneas futuras

En este apartado, se detallan las líneas futuras sobre las que se podría seguir trabajando.

- **Testing con IA:** La inteligencia artificial está tomando un papel cada vez mayor en todos los aspectos tecnológicos. En este proyecto se ha analizado un tipo de prueba, las pruebas Robo en Test Lab, que analiza la aplicación inteligentemente y la prueba sin desarrollar tests. Aún así, estas pruebas no analizan las funcionalidades como tales, y sería interesante probar ciertas herramientas que prueban aplicaciones utilizando inteligencia artificial.
- **Pruebas de rendimiento:** Durante el proyecto se han probado dos herramientas que prueban el rendimiento de la aplicación, pero existen muchas otras que se podrían analizar y que es posible que sean de gran ayuda para mejorar el rendimiento de aplicaciones Android.

Anexo A

Guía para la exportación de datos de Crashlytics utilizando BigQuery

A continuación se presenta la guía realizada para la empresa en la que se explica cómo configurar la exportación de datos de Crashlytics utilizando BigQuery.

Exportación de datos de Crashlytics utilizando BigQuery

16/03/2023

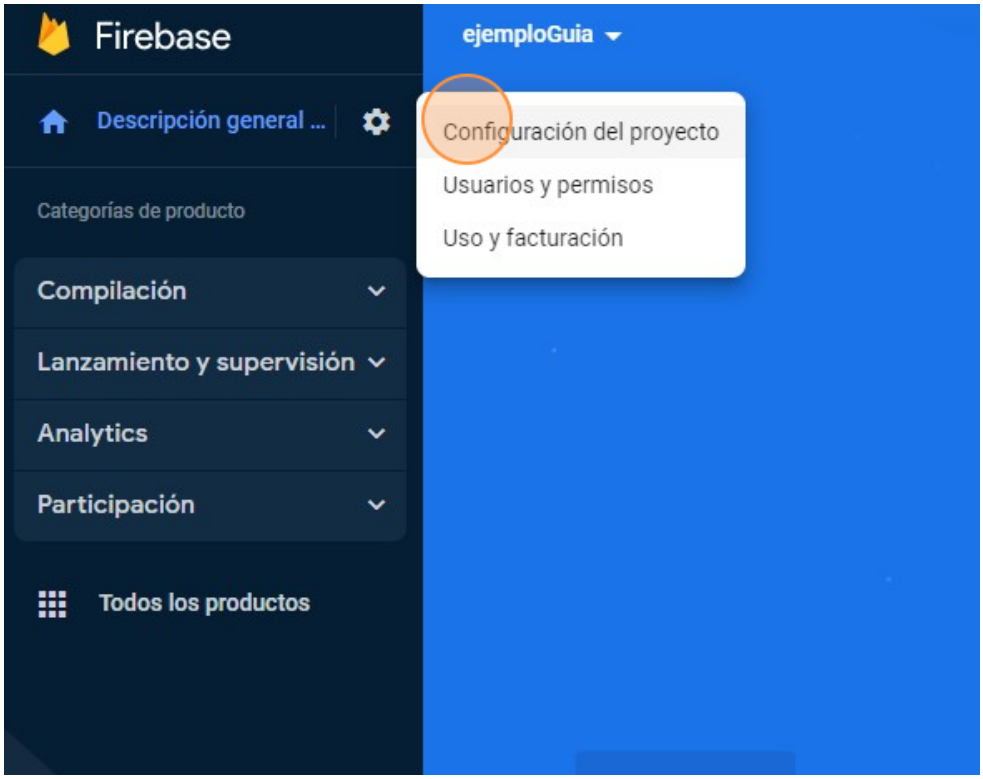
HOJA DE REVISIONES

Nº Revisión	Fecha	Revisión
00	**17/03/2023**	**Primera Edición**

Índice

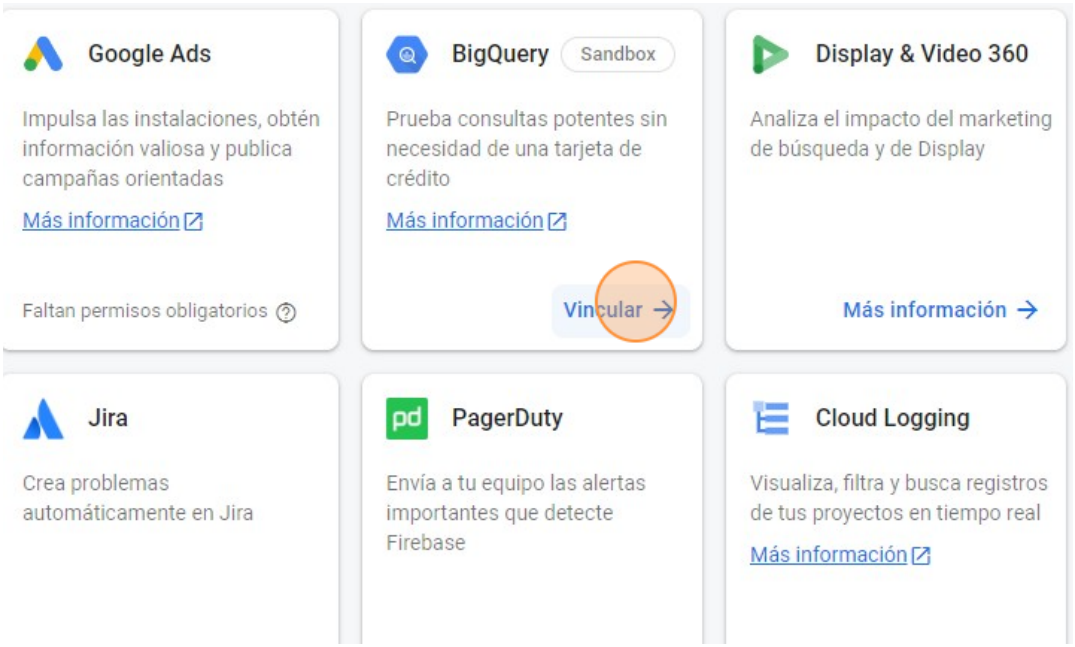
Conexión de BigQuery con Firebase

Acceder a la consola de Firebase en el proyecto que se quiera vincular a BigQuery: https://console.firebase.google.com/.
Pulsar en "Configuración del proyecto".

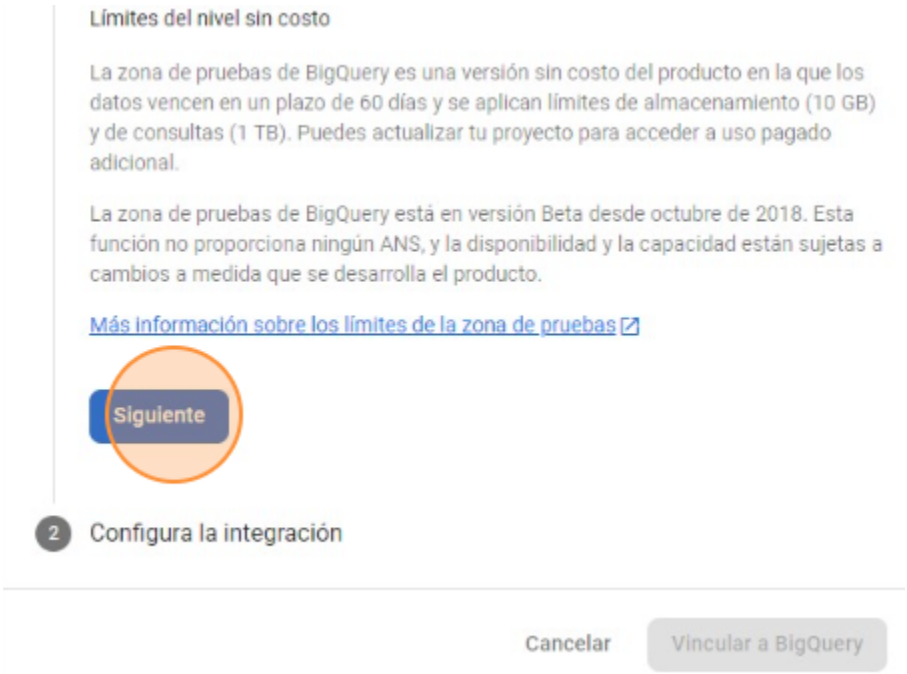


Pulsar en la pestaña "Integraciones".

Buscar la tarjeta BigQuery, pulsar en “Vincular”.

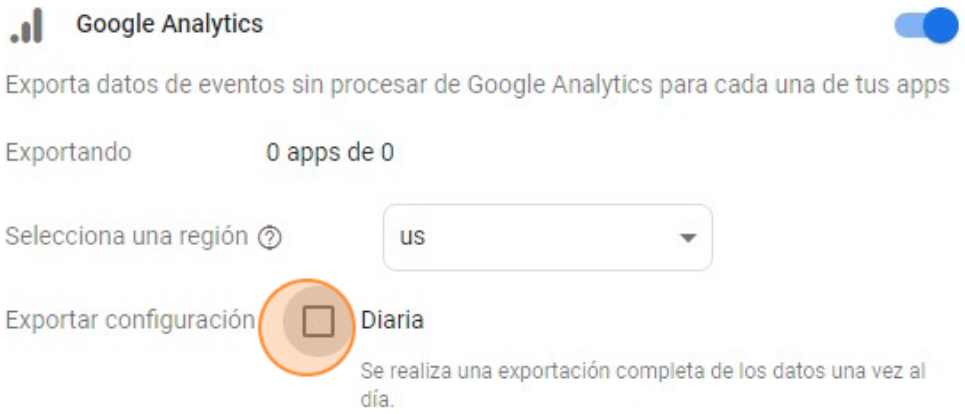


Pulsar “Siguiente”.



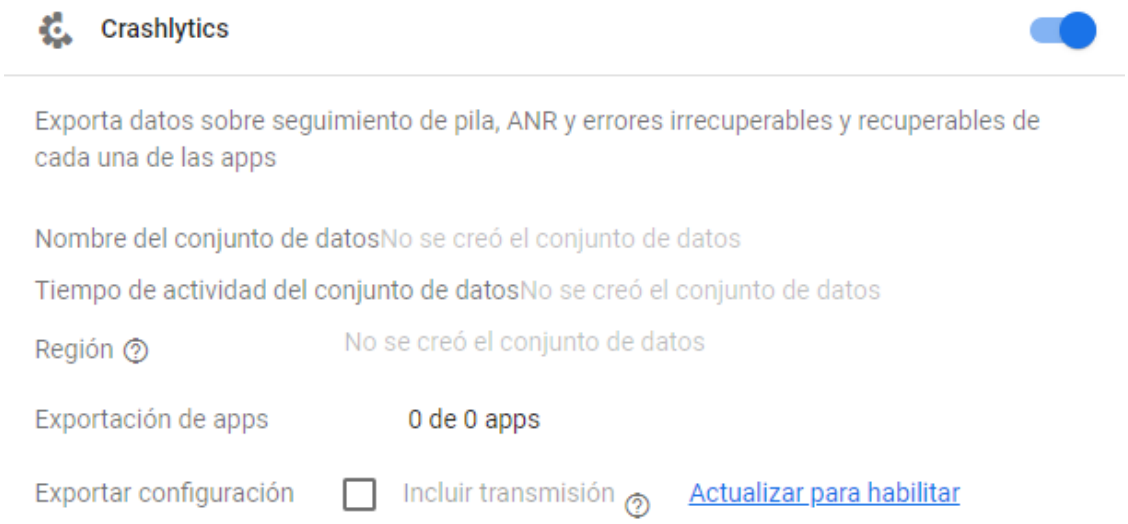
Activar las herramientas de las que se quieran exportar datos. Primero activar Google Analytics.

Seleccionar las aplicaciones que se quieran exportar, la región 'us' y que se exporte diariamente. Es importante seleccionar la misma región en todas las exportaciones para poder hacer las consultas posteriormente, y debe ser 'us' ya que Crashlytics solo permite exportar a esa región.

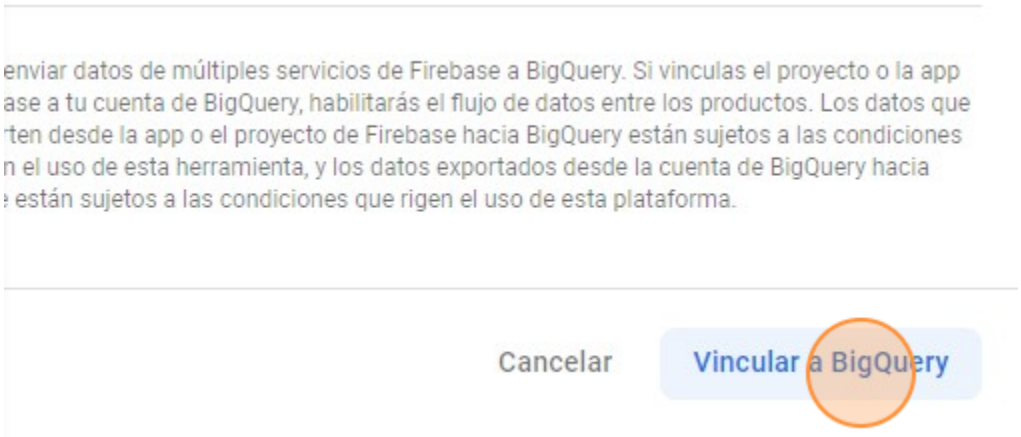


Después, activar Crashlytics.

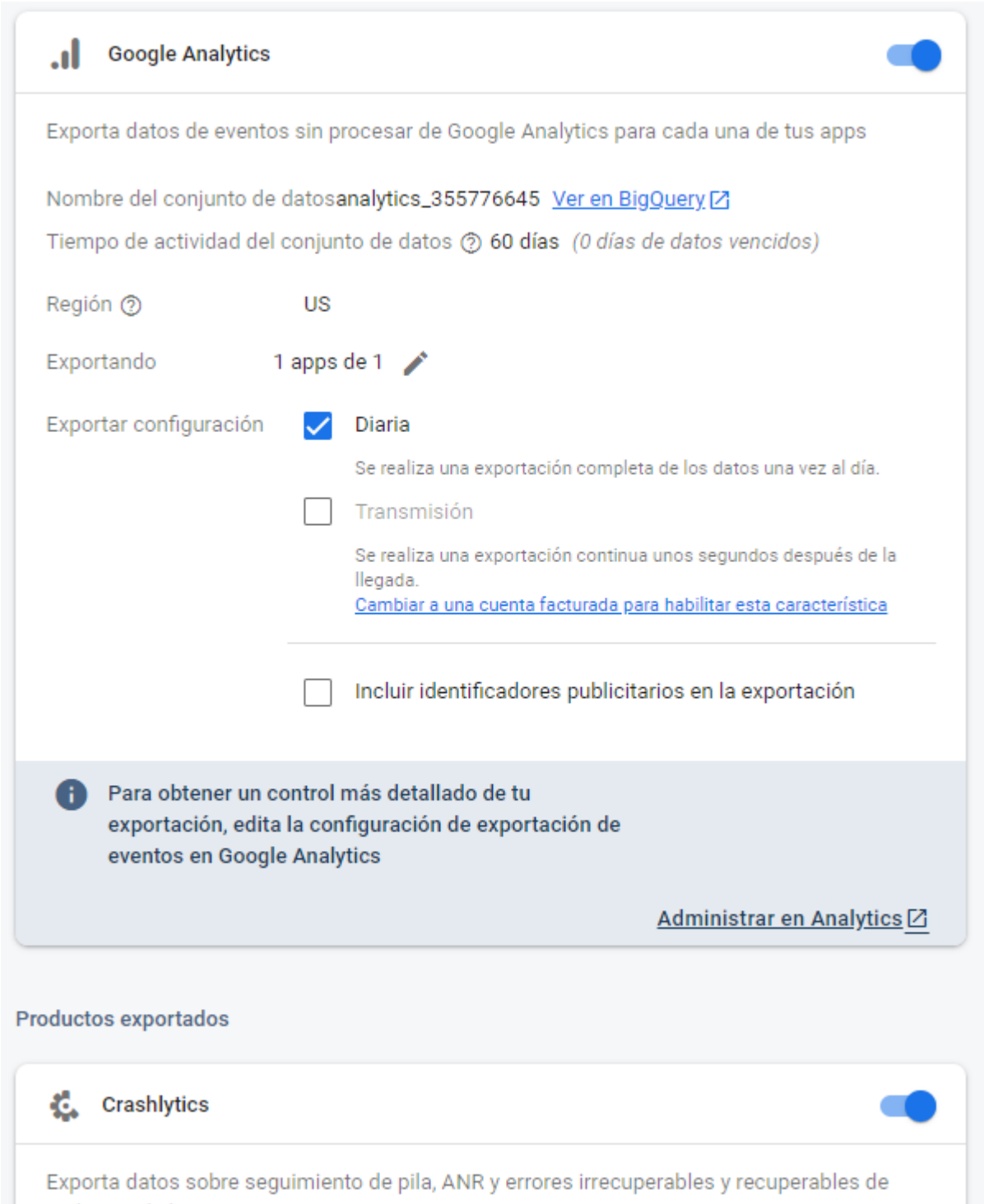
Seleccionar las aplicaciones que se quieran exportar, la región será ‘us’ estará seleccionada por defecto.



Pulsar “Vincular a BigQuery”.

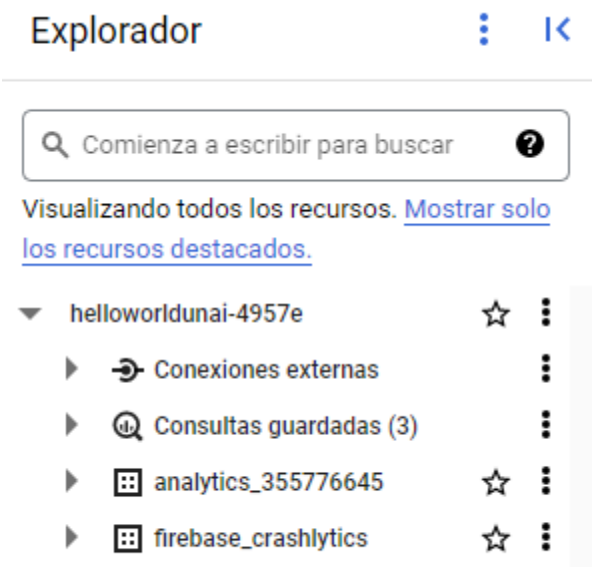


En la pestaña de BigQuery de Firebase, revisar que ambas estén activadas.

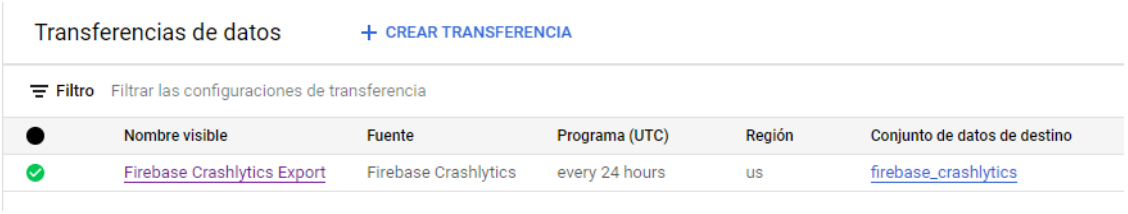


Acceder a BigQuery: https://console.cloud.google.com/bigquery.
Seleccionar en la parte superior el proyecto con el que se está trabajando. Se utilizará la zona de pruebas de BigQuery, en la que no hace falta configurar la facturación.

Revisar que los conjuntos de datos "analytics_xxxx" y "firebase_crashlytics" se han creado. Es posible que estos conjuntos de datos tarden 24 horas en crearse, ya que la exportación de datos se hace diariamente.

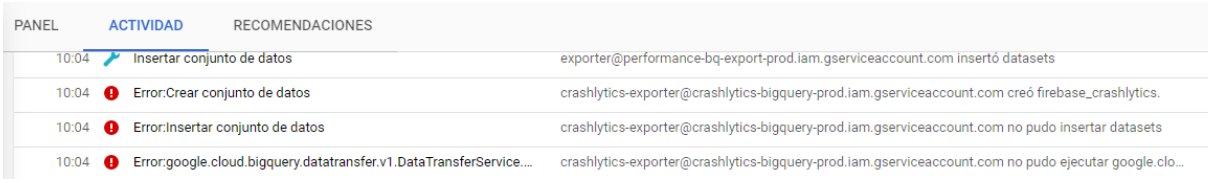


Revisar que en la pestaña "Transferencia de datos" la transferencia de datos "Firebase Crashlytics Export" está creada. Es posible que esta transferencia de datos tarde también 24 horas en crearse, ya que la exportación de datos se hace diariamente.



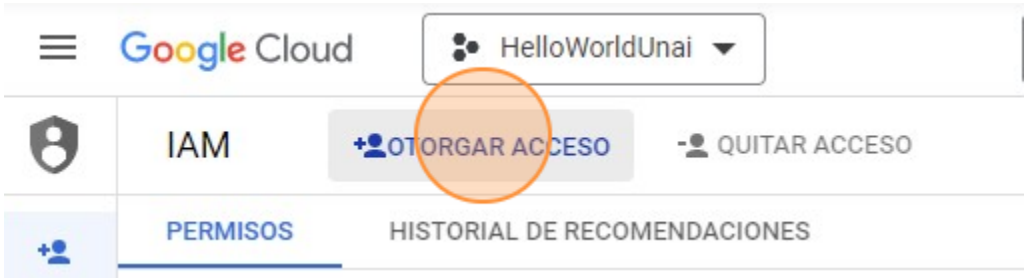
Solución si los conjuntos de datos no se han creado

Si pasadas las 24 horas horas alguno de los conjuntos de datos no se han creado, revisar en pestaña de actividad de Google Cloud: https://console.cloud.google.com/home/activity si aparece algún error de permisos.



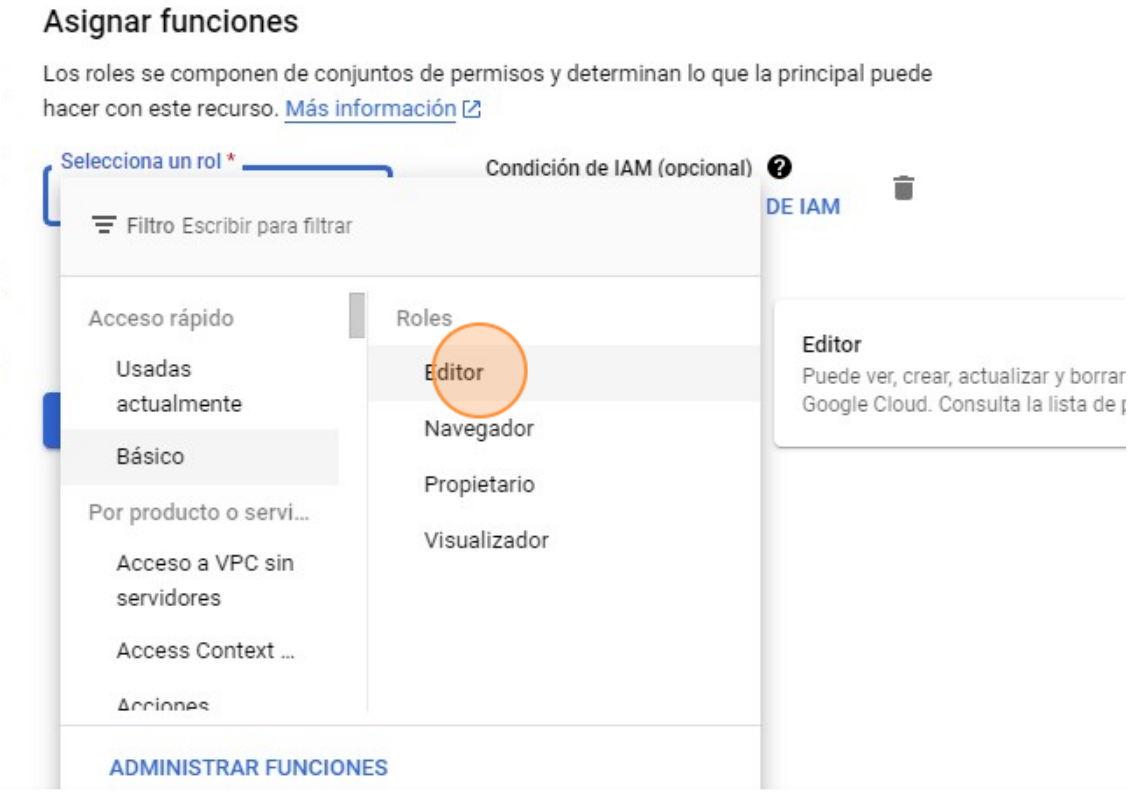
Si se encuentra un error de permisos, acceder a la página de administración de permisos de Google Cloud: https://console.cloud.google.com/iam-admin/iam.

Pulsar "Otorgar acceso".



En “Agregar principales” añadir el correo electrónico del servicio al que le faltan permisos que se ha visto en la consola de actividad.

En “Asignar funciones” seleccionar el rol “Editor”.

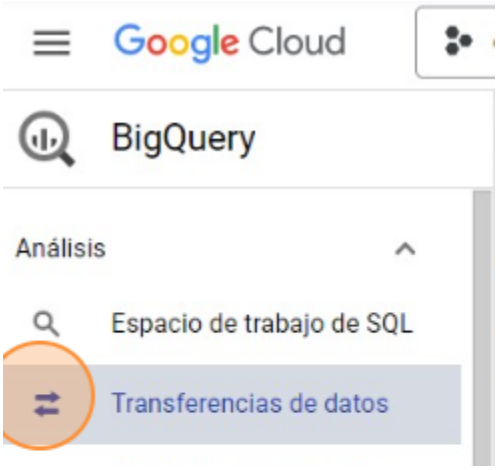


Pulsar “Guardar”.

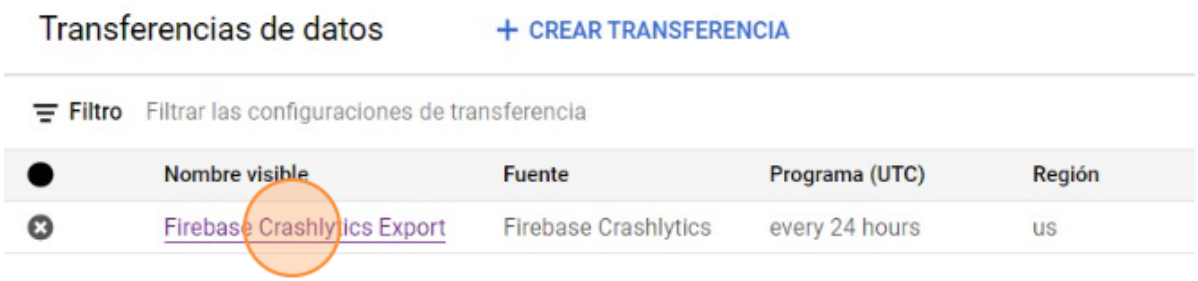
Deshabilitar y volver a habilitar la exportación en la consola de Firebase. Revisar que esta vez no aparezcan errores de permisos en la consola de actividad de Google Cloud y que los conjuntos de datos se creen correctamente.

Reabastecimiento de datos anteriores

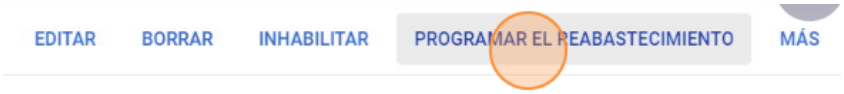
Si se quiere, la exportación de datos de Crashlytics permite hacer un reabastecimiento de los datos anteriores a activar la exportación. Para ello, en la consola de BigQuery, se accede a la pestaña de “Transferencias de datos”.



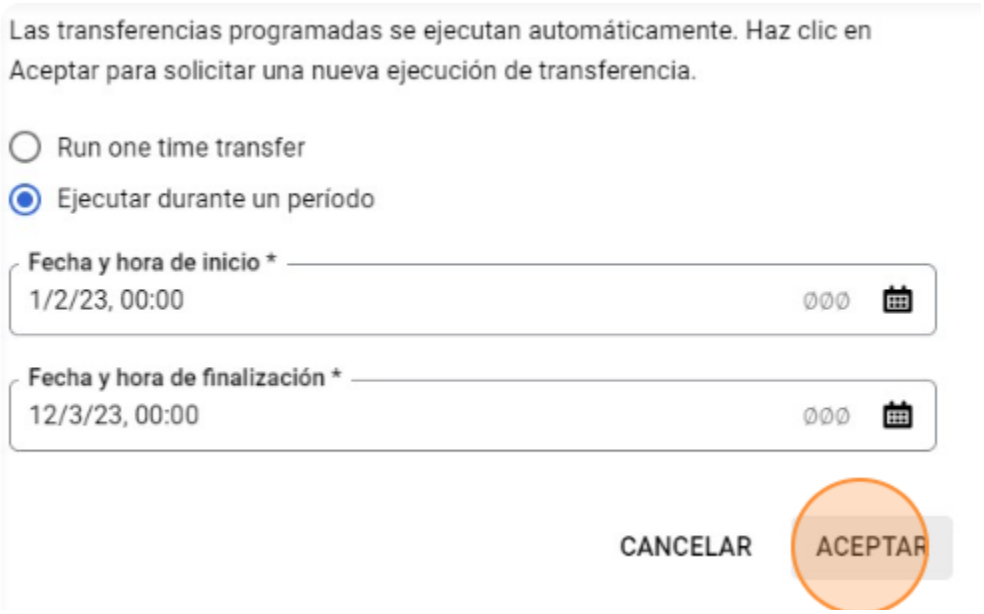
Seleccionar la transferencia de la que queremos hacer el reabastecimiento. En este caso, "Firebase Crashlytics Export".



Pulsar "Programar el reabastecimiento".



Seleccionar "Ejecutar durante un periodo", escoger las fechas de inicio y fin de las que se quiere exportar los datos y pulsar "Aceptar".
Es importante recalcar que se debe poner como fecha fin siempre fechas anteriores a la de la primera ejecución, ya que si no se pueden exportar datos de un mismo día varias veces y después encontrarlos duplicados.

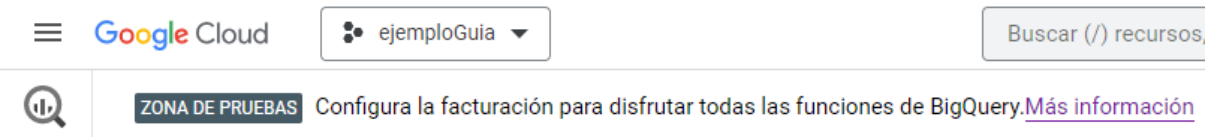


Ahora, aparecen todas las transferencias programadas y arriba, en “Fecha objetivo de la próxima ejecución”, la fecha en la que se ejecutarán. Una vez se ejecuten se podrán ver todos los crashes correspondientes a esas fechas en la tabla correspondiente a Crashlytics.

28 de febrero de 2023	13 de marzo de 2023, 12:26:44 UTC+1	La ejecución de la transferencia está pendiente (solicitud manual)
27 de febrero de 2023	13 de marzo de 2023, 12:26:59 UTC+1	La ejecución de la transferencia está pendiente (solicitud manual)
26 de febrero de 2023	13 de marzo de 2023, 12:27:14 UTC+1	La ejecución de la transferencia está pendiente (solicitud manual)
25 de febrero de 2023	13 de marzo de 2023, 12:27:29 UTC+1	La ejecución de la transferencia está pendiente (solicitud manual)
24 de febrero de 2023	13 de marzo de 2023, 12:27:44 UTC+1	La ejecución de la transferencia está pendiente (solicitud manual)
23 de febrero de 2023	13 de marzo de 2023, 12:27:59 UTC+1	La ejecución de la transferencia está pendiente (solicitud manual)
22 de febrero de 2023	13 de marzo de 2023, 12:28:14 UTC+1	La ejecución de la transferencia está pendiente (solicitud manual)

Límites de Uso

Utilizaremos la versión de pruebas “Sandbox” de BigQuery, para la que no hace falta configurar facturación y es totalmente gratuita.
Se puede revisar que se está utilizando esta versión ya que aparece un banner indicandolo en la consola de BigQuery:



Los límites son:

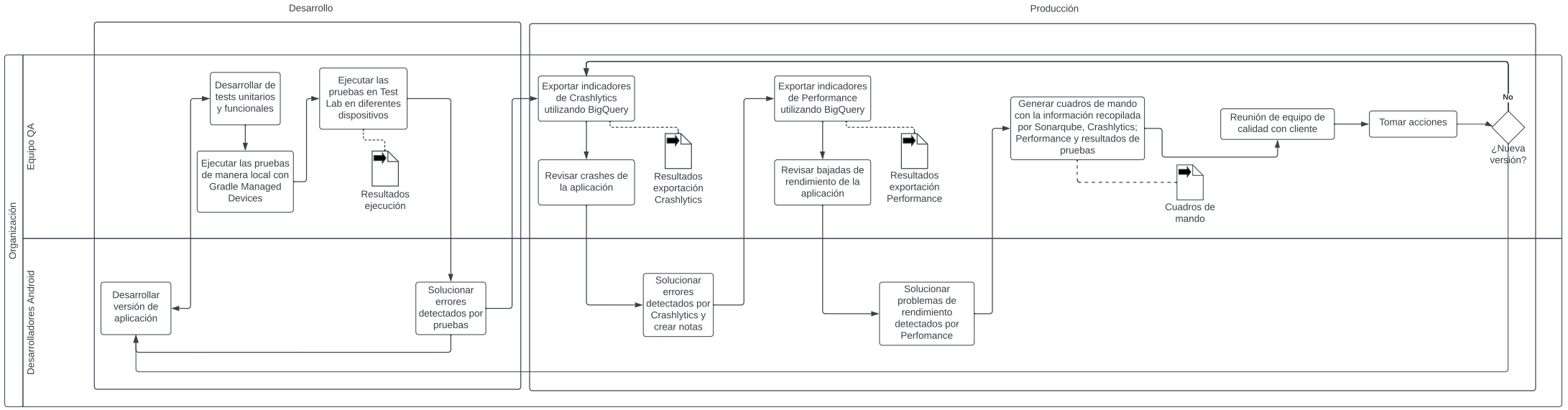
- 10 GB de almacenamiento y 1 TB de datos de consulta procesados por mes.
- Todos los conjuntos de datos tienen un tiempo de vencimiento de 60 días.

Anexo B

Proceso de integración de Firebase en desarrollo Android

A continuación podemos ver el proceso de integración de Firebase en desarrollo Android diseñado por el alumno a página completa.

Integración de Firebase en desarrollo Android



Anexo C

Actas de reuniones

Este anexo recopila todas las actas de las reuniones que se han llevado a cabo durante el proyecto. En estas actas se presentan los puntos más importantes que se han tratado en las mismas y las decisiones tomadas. De esta manera, se puede ver el desarrollo del proyecto solo leyéndolas.

Las actas han sido realizadas con los interesados del proyecto y redactadas por los alumnos. En todas se sigue la misma plantilla, con el objetivo de mantener una misma estructura que sea fácil de seguir y útil al mismo tiempo.

Acta de reunión TFG

Fecha: 21/02/2023
Inicio: 10:00
Fin: 11:15
Lugar: Reunión telemática
Tipo de Reunión: Reunión de Inicio
Asistentes:

Unai Pinedo Molina
Maider Azanza Sese
Beatriz Pérez Lamancha

Orden del día

1. Iniciar el proyecto.
2. Definir los requisitos del proyecto.
3. Definir el alcance del proyecto.

Resumen de la reunión

Beatriz comenzó la reunión explicando el proyecto en más detalle y los requisitos del proyecto, los cuales fueron apuntados por Unai. Se comentó que el alcance era difícil de acotar, por lo que se hará un seguimiento estricto para minimizar el riesgo.

Estado del proyecto

El proyecto se encuentra en su fase inicial.

Decisiones

- **Decisiones adoptadas:**
    - El ciclo de vida del proyecto será iterativo incremental.
    - Se creará un documento en el que se recogerán todas las horas dedicadas al proyecto y en qué tarea.
    - Se divide el proyecto en cuatro iteraciones:
        - Crear informes con los datos de crashlytics.
        - Estudiar Performance en Firebase
        - Estudiar Test Lab en Firebase
        - Proponer una forma de integrar de Firebase a la metodología de desarrollo

- **Asignación de tareas a realizar:**

Tarea	Responsable	Fecha límite
Dar acceso a Unai al Firebase de GPMobile	Beatriz	23/02/2023
Estudiar si curso de Udemy merece la pena	Unai	22/02/2023
Crear planificación	Unai	
Crear documento de requisitos	Unai	
Comenzar con estudio de Crashlytics	Unai	

- **Próxima reunión:** 07/03/2023 a las 12:00.

Acta de reunión TFG

Fecha: 07/03/2023
Inicio: 12:15
Fin: 13:00
Lugar: Reunión telemática
Tipo de Reunión: Seguimiento y control
Asistentes:

Unai Pinedo Molina
Maider Azanza Sese
Beatriz Pérez Lamancha

Orden del día

1. Revisión de la planificación y requisitos.
2. Presentación del estudio de Crashlytics y BigQuery.
3. Definir la estructura de la memoria y documentación.

Resumen de la reunión

La reunión comienza con la revisión de la planificación y los requisitos. Se validan los requisitos y se acepta la planificación. A continuación Unai presenta el estudio de Crashlytics y la integración con BigQuery hecha durante las semanas anteriores. Después se acuerda la estructura de la memoria y de la documentación interna para la empresa.

Estado del proyecto

El proyecto se encuentra en la primera iteración, la fase de estudio de manipulación y exportación de los datos de Crashlytics a través de BigQuery.

Decisiones

- **Decisiones adoptadas:**
    - Estudiar las limitaciones que ofrece la versión Sandbox de BigQuery.
    - Se debe crear documentación sobre el análisis de Crashlytics y la integración con Firebase.

- **Asignación de tareas a realizar:**

Tarea	Responsable	Fecha límite
Hacer los cambios comentados en la planificación.	Unai	10/03/23
Estudiar las limitaciones de la versión Sandbox de BigQuery.	Unai	20/03/23
Crear documentación sobre el análisis de Crashlytics y la forma de integrar BigQuery.	Unai	20/03/23
Mostrar la forma de trabajo con Crashlytics en Orona.	Beatriz	20/03/23

- **Próxima reunión:** 20/03/2023 a las 10:00.

Acta de reunión TFG

Fecha: 20/03/2023
Inicio: 12:00
Fin: 12:30
Lugar: Reunión presencial
Tipo de Reunión: Seguimiento y Control
Asistentes:

Unai Pinedo Molina
Maider Azanza Sese
Beatriz Pérez Lamancha

Orden del día

1. Presentación de consultas preparadas en BigQuery y problemas surgidos.
2. Presentación del estudio inicial de Performance.

Resumen de la reunión

Unai comenzó explicando las consultas SQL y la guía que había preparado para configurar la exportación de Crashlytics a BigQuery. Se comentaron los problemas encontrados y posibles soluciones. También se explicó brevemente lo que ofrece la herramienta de Performance Monitoring.

Estado del proyecto

El proyecto se encuentra entre la primera y la segunda iteración, finalizando la iteración sobre Crashlytics y comenzando con la de Performance.

Decisiones

- **Decisiones adoptadas:**
    - Cambiar consulta de Analytics para poder especificar versión.
    - Estudiar cómo se pueden utilizar las herramientas que ofrece Performance Monitoring.

- **Asignación de tareas a realizar:**

Tarea	Responsable	Fecha límite
Estudiar cómo utilizar Performance Monitoring	Unai	20/04/23

- **Próxima reunión:** 20/04/2023 a las 12:00.

Acta de reunión TFG

Fecha: 20/04/2023
Inicio: 10:00
Fin: 11:00
Lugar: Reunión telemática
Tipo de Reunión: Reunión de Seguimiento y Control
Asistentes:

Unai Pinedo Molina
Maider Azanza Sese
Beatriz Pérez Lamancha

Orden del día

1. Presentar forma de trabajo con Performance Monitoring
2. Presentar análisis de Test Lab
3. Analizar dedicaciones

Resumen de la reunión

Unai comenzó la reunión explicando la forma de utilizar Performance Monitoring en un desarrollo real. Se decidió crear una presentación con esta información para presentarla a los desarrolladores y ver qué es lo más reseñable.
Luego, Unai presentó y explicó el análisis que se ha hecho de Test Lab, y se decidió que se probaría con una aplicación real de la empresa, GPMobile.
Después, se analizaron las dedicaciones y se decidió que no era necesario replanificar ya que las desviaciones son pequeñas.

Estado del proyecto

El proyecto se encuentra finalizando la segunda iteración y comenzando la tercera.

Decisiones

- **Decisiones adoptadas:**
  - Se hará una reunión con desarrolladores para seleccionar las métricas más reseñables de Performance Monitoring.
  - Se utilizará la aplicación GPMobile para pruebas en Test Lab

- **Asignación de tareas a realizar:**

Tarea	Responsable	Fecha límite
Explicar la importancia de introducir Firebase al desarrollo móvil de la empresa	Beatriz	24/04/2023
Crear PPT para reunión sobre Performance	Unai	24/04/2023
Utilizar GPMobile para Test Lab	Unai	

- **Próxima reunión:** Sin especificar.

Acta de reunión TFG

Fecha: 11/05/2023
Inicio: 16:00
Fin: 16:45
Lugar: Reunión híbrida
Tipo de Reunión: Reunión de Seguimiento y Control
Asistentes:

Unai Pinedo Molina
Maider Azanza Sese
Beatriz Pérez Lamancha

Orden del día

1. Decidir cómo diseñar y estructurar la forma de trabajo.
2. Mostrar herramientas estudiadas para hacer tests de performance: Android Profiler y LeakCanary.

Resumen de la reunión

Unai comenzó la reunión explicando la presentación que se ha preparado la semana que viene con equipo de Movilidad en la que se mostrará el trabajo realizado y se concretará la forma de trabajo. Después, mostró las herramientas estudiadas para realizar tests de rendimiento, Android Profiler y LeakCanary.

Estado del proyecto

El proyecto se encuentra comenzado la cuarta y última iteración, diseñando la forma de trabajo.

Decisiones

- **Decisiones adoptadas:**
  - Se incluirán las herramientas para pruebas de performance en la presentación.

- **Asignación de tareas a realizar:**

Tarea	Responsable	Fecha límite
Incluir herramientas de pruebas de performance en presentación	Unai	17/05/23
Matricular TFG en GAUR	Unai	
Incluir tarea de estudio de pruebas de performance	Unai	

- **Próxima reunión:**

Acta de reunión TFG

Fecha: 21/02/2023
Inicio: 11:00
Fin: 12:00
Lugar: Facultad Ingeniería Informática UPV/EHU
Tipo de Reunión: Reunión de Cierre
Asistentes:
Unai Pinedo Molina
Maider Azanza Sese
Beatriz Pérez Lamancha

Orden del día

1. Enseñar apartados nuevos de la memoria: Aplicaciones utilizadas, LeakCanary, AndroidProfiler.
2. Enseñar comparativa de Test Lab con AWS.
3. Enseñar proceso de herramientas Firebase.
4. Definir defensa del proyecto.

Resumen de la reunión

Unai comenzó explicando los apartados que se habían añadido a la memoria. Después hizo una pequeña demo de cómo se utiliza AWS Device Farm y enseñó la comparativa que se había hecho con Test Lab. A continuación se enseñó el proceso que se había hecho introduciendo las herramientas de Firebase al desarrollo Android de la empresa. Por último, se habló sobre la defensa del proyecto.

Estado del proyecto

El proyecto se encuentra en su fase final, a falta de terminar de redactar la memoria y preparar la defensa.

Decisiones

- **Decisiones adoptadas:**
  - Se explicará en la comparativa con AWS que Device Farm ofrece frameworks antiguos sin soporte.
  - Quitar al cliente del proceso y hacerlo cíclico.

- **Asignación de tareas a realizar:**

Tarea	Responsable	Fecha límite
Modificar memoria	Unai	
Modificar proceso	Unai	

Preparar defensa	Unai	

Bibliografía

[1] P. Bourque and R.E. Fairley. Guide to thesoftware engineering body of knowledge. *IEEE Computer Society*, pages 177–180, 2014. Ver página 1.

[2] Schonning N. Dun C. Britch, D. and J. Osborne. El patrón model-view-viewmodel. https://learn.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm. Accessed: 2023-03-23. Ver página 16.

[3] Documentación de firebase crashlytics. https://firebase.google.com/docs/crashlytics. Accessed: 2023-04-06. Ver página 19.

[4] Cómo analizar los problemas de firebase crashlytics con app quality insights. https://developer.android.com/studio/debug/app-quality-insights. Accessed: 2023-03-29. Ver página 22.

[5] Documentación de firebase performance monitoring. https://firebase.google.com/docs/perf-mon. Accessed: 2023-04-21. Ver página 29.

[6] Perfila el rendimiento de tu app. https://developer.android.com/studio/profile. Accessed: 2023-04-28. Ver página 36.

[7] Documentación de leakcanary. https://square.github.io/leakcanary/. Accessed: 2023-04-29. Ver página 37.

[8] Documentación de test lab. https://firebase.google.com/docs/test-lab. Accessed: 2023-05-04. Ver página 41.

[9] Documentación de device farm. https://aws.amazon.com/es/device-farm/. Accessed: 2023-05-24. Ver página 50.

[10] Wynne M. Dees, I. and A. Hellesoy. Cucumber recipes: Automate anything with bdd tools and techniques. *Pragmatic Bookshelf*, 2013. Ver página 51.

[11] Ajusta tus pruebas con dispositivos administrados por gradle. https://developer.android.com/studio/test/gradle-managed-devices. Accessed: 2023-05-13. Ver página 52.

[12] Dr. Bernhard Hitpass. *BPMN Manual de Referencia y Guia Practica 5 Edicion: Con una introducción a CMMN y DMN*. CreateSpace Independent Publishing Platform, 2017. Ver página 55.

[13] La gestión de proyectos. conceptos básicos. http://cv.uoc.edu/annotation/ebc1adfc61836d7205ad7dde343367b5/603266/PID_00153570/PID_00153570.html. Accessed: 2023-06-13. Ver página 62.